МИНИСТЕРСТВО РАЗВИТИЯ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОММУНИКАЦИЙ РЕСПУБЛИКИ УЗБЕКИСТАН

ТАШКЕНТСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ ИМЕНИ МУХАММАДА АЛ-ХОРЕЗМИ

РАЗРАБОТКА WEB ПРИЛОЖЕНИЙ

Учебное пособие

Ташкент – 2018

Авторы: К.Ф.Керимов, Ш.К.Камалов, Ш.Ш.Мухсинов, Н.П.Сиддикова "Разработка web приложений". Учебное пособие /ТУИТ. 444с. Ташкент, 2018

АННОТАЦИЯ

Целями освоения пособия являются освоение студентами технологий Web программирования, а также получение студентами навыков создания, программирования статических и динамических Web-документов, клиентских приложений, выполнимых браузером, а также создания собственного Web-ресурса и использования готовых Web приложений. Пособие входит в вариативную часть профессионального цикла образовательной программы. Изучение данной дисциплины базируется на следующих курсах: «Оптимизация и развитие web приложений», «Программирование на SQL», «Системный анализ и требования». Содержание пособия направлено на формирование и закрепление следующей компетенции: способность разрабатывать компоненты аппаратно-программных комплексов и баз данных, используя современные инструментальные средства и технологии программирования. В результате изучения пособия бакалавр должен: - знать функциональные возможности и архитектурные особенности сети Интернет; - уметь создавать информационные ресурсы и использовать ресурсы Интернет. Современные технологии Internet. Протоколы Internet. Взаимодействие браузера и Web-сервера; - технологии разработки Web-документов и ресурсов; - определение структуры Web-документов; - спецификации Web-документов; - структура, списки, графика, таблицы Web-документов; - разработка интерактивных Web-документов; - языки сценариев JavaScript, PHP; - основные положения языка JavaScript. Основные объекты языков JavaScript и PHP; - средства разработки, программ на языке JavaScript; - организация вычислений в языках JavaScript и PHP; - создание клиентских и серверных приложений; - заключение. Перспективы развития Internet-технологий.

Рецензенты:

Латипова Н.Х. – кандидат технических наук, доцент кафедры «Системное и прикладное программирование» Ташкентского университета информационных технологий

Худайбердиев М.Х. – кандидат технических наук, Старший научный сотрудник «Центр разработки программных продуктов и аппаратно-программных комплексов».

ANNOTATION

The purpose of the manual is the students 'mastery of Web programming technologies, as well as the acquisition of students' skills in creating, programming static and dynamic Web documents, client applications that the browser can run, and creating their own Web resource and using ready-made Web applications. The manual is included in the variable part of the professional cycle of the educational program. The study of this discipline is based on the following courses: "Optimization and development of web applications", "SQL programming", "System analysis and requirements". The content of the manual is aimed at the formation and consolidation of the following competencies: the ability to develop components of hardware-software complexes and databases using modern tools and programming technologies. As a result of studying the manual, a bachelor must: - know the functionality and architectural features of the Internet; - be able to create information resources and use Internet resources. Modern Internet technologies. Internet protocols. Interaction between the browser and the Web server; - technologies for developing Web-based documents and resources; - Definition of the structure of Web-documents; - specification of Web-documents; - structure, lists, graphics, tables of Web-documents; - development of interactive Web-documents; - scripting languages JavaScript, PHP; - the main provisions of the JavaScript language. The main objects of JavaScript and PHP; - Development tools, programs in the language of JavaScript; - Organization of calculations in the languages of JavaScript and PHP; - Creation of client and server applications; conclusion. Prospects for the development of Internet technologies.

Reviewers:

Latipova N.H. – Candidate of technical sciences, Associate professor of the department "System and application programming", Tashkent university of information technologies.

Hudoyberdiev M.H . – candidate of technical sciences, Senior Researcher of «Centre for the development of software and hardware-program complex».

АННОТАЦИЯ

Мазкур қўлланма talabalarning veb-dasturlash texnologiyalaridan mahoratini oshirish, shuningdek, talabalarning statik va dinamik web-hujjatlarni yaratish, dasturiy vositalarni yaratish, brauzer ishlashi mumkin bo'lgan mijoz ilovalari va o'z web-resurslarini yaratish va ulardan foydalanish tayyor web-ilovalar. Qo'llanma ta'lim dasturining professional siklining o'zgaruvchan qismiga kiritilgan. Ushbu intizomni o'rganish "web dasturlarni optimallashtirish va rivojlantirish", "SQL dasturlash", "Tizim tahlillari va talablari" kabi yo'nalishlarga asoslangan. Qo'llanmaning mazmuni quyidagi kompetentsiyalarni shakllantirish va mustahkamlashga qaratilgan: zamonaviy uskunalar va dasturiy texnologiyalardan foydalangan holda apparat-dasturiy komplekslar va ma'lumotlar bazalari komponentlarini ishlab chiqish qobiliyati. Qo'llanmani o'rganish natijasida bakalavr: - Internetning funktsional va arxitektura xususiyatlarini bilish; - axborot resurslarini yaratish va Internet resurslaridan foydalanish. Zamonaviy Internet texnologiyalari. Internet protokollari. Brauzer va web-server o'rtasidagi o'zaro bog'liqlik; - web-hujjatlar va resurslarni ishlab chiqish texnologiyalari; - web-hujjatlarni tuzilishini aniqlash; - web-hujjatlarni aniqlashtirish; - web-hujjatlar tuzilmalari, ro'yxatlar, grafikalar, jadvallar; - interfaol web-hujjatlarni ishlab chiqish; - kript tillari JavaScript, PHP; - JavaScript tilining asosiy qoidalari. JavaScript va PHP ning asosiy ob'ektlari; - JavaScript-ni tilida ishlab chiqish vositalari; - JavaScript va PHP tillarida hisob-kitoblarni tashkil etish; - mijozlar va server dasturlarini yaratish; Xulosa. Internet-texnologiyalarni rivojlantirish istiqbollari.

Такризчилар:

Латипова Н.Х. – техника фанлари намзоди, Тошкент ахборот технологиялари университети «Тизимли ва амалий дастурлаш» кафедраси доценти.

Худайбердиев М.Х. – техника фанлари намзоди, Тошкент ахборот технологиялари университети «Дастурий махсулотлар ва аппарат-дастурий мажмуалар яратиш маркази»нинг катта илмий ходими.

ТУИТ имени Мухаммада ал-Хорезми, 2018

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 9

РАЗДЕЛ 1. ВВЕДЕНИЕ В ПРЕДМЕТ “РАЗРАБОТКА WEB-ПРИЛОЖЕНИЙ” 15

Глава 1. Основные понятия в разработке Web-приложений. Web-страница, Web-сайт, Web-сервер 15

Глава 2. Языки разметок HTML, XML, XHTML, WML. 24

Глава 3.Языки сценариев 32

Глава 4. Технология “клиент-сервер” 44

РАЗДЕЛ 2. ОСНОВЫ HTML 64

Глава 5. Структура документа. Основные теги HTML 64

Глава 6. Оформление текстов. Гиперссылки. Списки. Таблицы. Фреймы. 74

Глава 7. Графика и работа с изображением. 90

Глава 8. Формы 94

Глава 9. Мультимедийные возможности HTML-5. 99

РАЗДЕЛ 3. ОСНОВЫ CSS 118

Глава 10. Основы CSS (Cascading Style Sheets). 118

Глава 11. Селекторы и их правила. Групповой селектор 126

Глава 12 Селекторы потомков. 139

Глава 13. Псевдокласы и псевдоэлементы 153

Глава 14. Селекторы атрибутов 169

Глава 15. Наследование и каскадность 200

Глава 16. Работа со шрифтами 217

Глава 17. Кроссбраузерность сайта 238

РАЗДЕЛ 4. ПРОГРАММИРОВАНИЕ НА JAVASCRIPT 245

Глава 18. Выражения, переменные, объекты JavaScript. 245

Глава 19. Иерархия объектов JavaScript. Объект Location 263

Глава 20. Работа с DOM 287

Глава 21. Основы ООП на JavaScript. 308

Глава22. Разработка мультимедийных веб-приложений........................ 321
Глава23. Разработка игр, калькуляторов и анимационных приложений328
Глава24. Разработка интернет приложений 338
РАЗДЕЛ 5. ОСНОВЫ PHP 347
Глава 25. Введение в программирование на стороне сервера................ 347
Глава 26. Программирование на PHP. Введение в ООП на PHP. Регулярные выражения в PHP. Работа с БД на PHP. 356
ЛИТЕРАТУРА 441

МУНДАРИЖА

КИРИШ.. 4

1 БЎЛИМ. “WEB ИЛОВАЛАРНИ ИШЛАБ ЧИҚИШ” ФАНИГА КИРИШ 8

1 боб. Web иловаларни ишлаб чиқиш асосий тушунчалари.Web-саҳифа, Web-сайт, Web-сервер. 8

2 боб. Разметкали тиллар: HTML, XML, XHTML, WML................ 15

3 боб. Сценарийли тиллар. 21

4 боб. "Клиент-сервер" технологияси 30

2 БЎЛИМ. HTML АСОСЛАРИ 46

5 боб. Хужжат тузилиши. HTML нинг асосий теглари 46

6 боб. Матнларни безаш.Рўйхатлар. Жадваллар. Фреймлар........... 54

7 боб. Графика ва тасвирлар билан ишлаш. 66

8 боб. Формалар. 69

9 боб. HTML -5 мультимедиа имкониялари 73

3 БЎЛИМ. CSS АСОСЛАРИ 89

10 боб. CSS (Cascading Style Sheets) асослари 89

11 боб. Селекторлар ва уларнинг қоидалари. Селектор гурухлари 96

12 боб. Авлод селекторлар 108

13 боб. Псевдокласс селекторлари ва псевдо элементлар 120

14 боб. Селектор атрибутлари 135

15 боб. Насллик ва каскадлик 162

16 боб. Шрифтлар билан ишлаш 177

17 боб. Cross браузерлар 194

4 БЎЛИМ. JAVASCRIPT ДАСТУРЛАШ.. 201

18 боб. JavaScript ифодалар,ўзгарувчилар, объектлар 201

19 боб. JavaScript объектлар иерархияси. Location объекти............ 215

20 боб. DOM асослари 234

21 боб. JavaScript да ОЙД асослари. 251

22 боб. Мультимедиали веб илова ишлаб чиқиш............................ 262

23 боб. Ўйин, каркульятор, анимацияли иловалар ишлаб чиқиш .. 267
24 боб. Интернетда ишлайдиган иловалар ишлаб чиқиш 278
5 БЎЛИМ. PHP АСОСЛАРИ 286
25 боб. Сервер томонида дастурлаш асослари 286
26 боб. PHPда дастурлаш. PHPнинг ОЙД асослари. PHPда регуляр ифодалар. PHPда маълумотлар базаси билан ишлаш. 293
АДАБИЁТЛАР РЎЙҲАТИ 363

TABLE OF CONTENTS

INTRODUCTION 9

SECTION 1. INTRODUCTION TO THE SUBJECT "DEVELOPMENT OF WEB-APPLICATIONS" 15

Chapter 1. Basic concepts in the development of Web-applications. Web-page, Web-site, Web-server 15

Chapter 2. HTML, XML, XHTML, WML markup languages. 24

Chapter 3. Scripting Scripts 32

Chapter 4. “Client-server” technology 44

SECTION 2. THE BASICS OF HTML 64

Chapter 5. Structure of the document. HTML Basic tags 64

Chapter 6. Text processing. Hyperlinks. Lists. Tables. Frames 74

Chapter 7. Graphics and work with the image 90

Chapter 8. Forms 94

Chapter 9. Multimedia features of HTML-5 99

SECTION 3. THE BASIS OF CSS 118

Chapter 10. The Basics of CSS (Cascading Style Sheets) 118

Chapter 11. Selectors and their rules. Group selector 126

Chapter 12 Selectors of descendants 139

Chapter 13. Pseudo-classes and pseudo-elements 153

Chapter 14. Attribute Selectors 169

Chapter 15. Inheritance and cascade 200

Chapter 16. Working with Fonts 217

Chapter 17. Cross-browser site 238

SECTION 4. PROGRAMMING ON JAVASCRIPT 245

Chapter 18. Expressions, variables, JavaScript objects 245

Chapter 19. The hierarchy of JavaScript objects. Location object 263

Chapter 20. Working with DOM 287

Chapter 21. The basics of OOP in JavaScript 308

Chapter 22. Development of multimedia web applications 321

Chapter 23. Development of games, calculators and animation applications 328
Chapter24. Internet Application Development 338
SECTION 5. BASICS OF PHP 347
Chapter 25. Introduction to server-side programming 347
Chapter 26. Programming in PHP. Introduction to OOP in PHP. Regular expressions in PHP. Working with the database on PHP 356
REFERENCES 441

ВВЕДЕНИЕ

Постановление Президента Республики Узбекистан Ш. Мирзиёев 15 марта 2017 г., № ПП-2834 «О мерах по дальнейшему совершенствованию деятельности Ташкентского университета информационных технологий» открывает Великий ученый Востока Мухаммад ал-Хоразмий, живший и творивший в VIII-IX веках, является автором научных трудов по математике, геометрии, астрономии, истории, географии и другим наукам. В настоящее время все вычислительные операции, выполняемые в сфере информационно-коммуникационных технологий на основе высоких технологий, базируются на научных открытиях Мухаммада ал-Хоразмий.

Факультет «Программный инжиниринг» создан на основе Постановления Президента Республики Узбекистан №ПП-1942 «О мерах по дальнейшему совершенствованию системы подготовки кадров в области информационно-коммуникационных технологий» от 26 марта 2013 года.

Факультет готовит специалистов бакалавриата и магистратуры по специальности управлений и разработке программного обеспечения компьютерных и мобильных устройств, по сохранению, обработке и передаче данных, по разработке программ для компьютерных и телекоммуникационных систем и сетей, а также программ по управлению базами данных.

Влияние глобальной компьютерной сети Internet на современный мир не имеет исторических аналогов. Его сегодняшний день – это начало эпохи электронного проникновения во все сферы человеческой жизни, это нечто большее, чем просто маркетинговая кампания, это основа новой философии и новой деловой стратегии.

Вполне логично предположить, что и с точки зрения рекламы продукции или услуги Интернет – наиболее значимый ресурс. Большинство

современных людей пользуются Интернетом, как наиболее доступным источником информации.

Web-технология полностью перевернула представления о работе с информацией, да и с компьютером вообще. Оказалось, что традиционные параметры развития вычислительной техники - производительность, пропускная способность, емкость запоминающих устройств - не учитывали главного "узкого места" системы - интерфейса с человеком. Устаревший механизм взаимодействия человека с информационной системой сдерживал внедрение новых технологий и уменьшал выгоду от их применения. И только когда интерфейс между человеком и компьютером был упрощен до естественности восприятия обычным человеком, последовал беспрецедентный взрыв интереса к возможностям вычислительной техники.

Создание Web-сайтов является одной из важнейших технологий разработки ресурсов Internet. Хороший сайт, вбирая в себя всю полезную информацию, является лучшей визитной карточкой и коммерческой фирмы и образовательного учреждения, работая на них в любое время суток.

К тому же сейчас, когда число онлайновых пользователей оценивается более чем в 500 миллионов человек (причем это число растет со скоростью приблизительно 7% в месяц), приходится искать пути использования новых технологий, которые могли бы оказать содействие в привлечении и удержании внимания пользователей, путешествующих по безбрежным просторам Web.

Веб-разработка – это наиболее демократичное занятие, поскольку требования к используемому для нее оборудованию выполнимы практически на всех используемых компьютерах. Например, для начала достаточно только наличие браузера и текстового редактора. Т.е. простые веб-страницы можно формировать даже на мобильных устройствах с самой простой операционной системой, главное чтобы там был текстовый редактор и браузер. Для более продвинутой работы – разработку серверной части –

необхродимы уже веб-сервер, интерпретатор и СУБД. Все это, как правило, доступно через использование многочисленных LAMP или WAMP сборок (веб-сервер, интерпретатор PHP, сервер баз данных) – денвер, XAMPP, vertrigo, WAMP и многих (более 50) других.

Предметом учебного пособия является web-технологии. Объект – среды и языки разработки web-приложений. Цель работы рассмотреть возможности практического использования языков и средств для разработки web-приложений.

Настоящее пособие предназначено в первую очередь для студентов ТУИТ, изучающих основные предметы информационно-коммуникационных технологий. Особенностью данного пособия является параллельное изучение материала на основе практических заданий, которое будет играть роль наставника, показывающего как научиться выполнять тот или иной прием процесса веб-разработки. В виду такой особенности, изучение основ веб-разработки лучше начать с самого начала, однако, студенты имеющие знания и навыки могут пропустить главы, изучение которых, по их мнению, будет напрасной тратой времени.

Предполагается, что после изучения настоящего курса студенты смогут самостоятельно разрабатывать веб-приложения малой и средней сложности и быть готовыми к изучению более сложных технологических методов и приемов разработки сложных веб-приложений.

При разработке пособия авторы придерживались «принципа Паретто»: 20 процентов первоначально передаваемых студентам знаний могут решать 80 процентов задач веб-разработчика. И, как уже отмечалось, главная цель пособия заключается в реализации принципа «делай как я».

Ниже представим кратко содержимое пособия, охарактеризовав каждую главу. Пособие состоит из пяти разделов, разбитых на 26 глав.

Первый раздел посвящен начальным знаниям о веб приложениях и их основных понятиях. Раздел состоит из 4 глав. В главе 1 рассматриваются

основные технологии и понятия, применяемые в разработке веб-приложений, сходства и различия процессов разработки прикладных программ и веб-приложений. Также рассматриваются платформы для разработки веб-приложений от использования простых текстовых редакторов и веб-серверных сборок. В главе 2 дается краткий обзор языков разметок и особенности, преимущества и недостатки технологических приемов. В главе 3 дается краткий обзор языков сценарий, а так же их особенности, преимущества и недостатки. В главе 4 приводится общий обзор о технологии клиент-сервер, вопросам установки и настройки веб-серверов, интерпретаторов языков программирования и других приложений баз данных, применяемых протоколов, объясняется принцип отработки запросов пользователей и ответов серверов.

Второй раздел посвящен основам языка гипертекстовой разметки HTML и которая состоит из 5 глав (5-9 главы). В главе 5 приводится информация о гипертекстовом языке разметки, с акцентом на новинки, реализованные в пятой версии, понятия о тегах и структуре документа. Глава 6 сосредоточена на описания тегов для оформления текста, гипперсылок, создания списков, таблиц и фреймов. Глава 7 сосредоточена на описания тегов для работы с графикой и работой с изображениями при помощи языка гипертекстовой разметки. Глава 8 посвящена изучению основ верстки – программированию форм и элементов управления форм. Глава 9 посвящена изучению мультимедийных возможностей языка гипертекстовой разметки.

Третий раздел посвящен основам оформления веб-страниц и использования каскадных таблиц стиля состоящая из 6 глав (10-17 главы). В главе 10 приводится информация об основах каскадных таблиц стиля и их составляющих. Глава 11 посвящена правилам описания селекторов и групповых селекторов. Глава 12 посвящена правилам описания селекторов потомков и их практические навыки. Глава 13 посвящена псевдоклассам и

псевдоэлементам, а так же их взаимосвязках в веб-приложениях. Глава 14 посвящена описаниям селекторов и их атрибутов, создания глобальных и локальных идентификационных атрибутов. Глава 15 описаны методы тиражирования различных CSS свойств, относящихся к одному элементу страницы на вложенные в него элементы. Глава 16 посвящена селекторам свойств форматирования текста и шрифта. Глава 17 посвящена понятие кроссбраузерности, о сервисе и их методике тестирования.

Четвертый раздел посвящен языку JavaScript – основному средству обеспечения интерактивности веб-приложений на стороне клиента и состоит из 7 глав (18-24 главы). Глава 18 вводит понятие объектной модели документа и объясняет, почему важно использовать эту модель. А так же посвящен объектам, переменным и выражения языка JavaScript. Глава 19 описывает иерархию объектов JavaScript и программирование для объекта Location. Глава 20 посвящена работе с объектами DOM. Глава 21 описывает объектно-ориентированные возможности языка JavaScript. Глава 22 описывает разработку мультимедийных веб-приложений средствами языка JavaScript. Глава 23 посвящена практической реализации мультимедийных средств языка JavaScript на примерах анимаций и игр. Глава 24 посвящена основам разработки веб приложений на языке JavaScript.

Пятый раздел посвящен разработке веб-приложений на стороне сервера, и сотоит из 2 глав. Глава 25 рассматривает общие вопросы программирования на стороне сервера. Приводит краткое описание возможных платформ и технологий. Подробно рассмотрено использование языка веб-программирования PHP и программного обеспечения баз данных MySQL. Глава 26 рассматриваются современные средства разработки веб-приложений на основе широкого использования шаблонов программирования, реализованных в различных сборках: системах управления контентом и фреймворках.

Минимальные требования к знаниям студентов.

Поскольку пособие пишется для студентов Ташкентского университета информационных технологий, то предполагается, что студенты обладают навыками работы на персональном компьютере: с клавиатурой, текстовыми редакторами, интегрированными оболочками, браузерами. Также предполагается, что студенты знакомы с архитектурой современных компьютеров и средств связи, а также имеют элементарные знания о хранящейся и обрабатываемой информации, и ее представлении в компьютере. Кроме того, желательно горячее стремление к самостоятельному изучению учебных дисциплин.

РАЗДЕЛ 1. ВВЕДЕНИЕ В ПРЕДМЕТ “РАЗРАБОТКА WEB-ПРИЛОЖЕНИЙ”

Глава 1. Основные понятия в разработке Web-приложений. Web-страница, Web-сайт, Web-сервер

Развитие современной компьютерной техники и внедрение новейших технологий положили начало нового направления жизни на Земле. За довольно короткий промежуток времени развития микроэлектроники и кибернетики произошло много изменений.

Прогрессивное развитие техники вызвало появление новых программных продуктов. С каждым годом внедряется все большее и большее количество языков программирования. Все они ориентированы, прежде всего, на целевую аудиторию.

Развиваются не только компьютеры, но и сети. Если еще несколько десятков лет назад Интернет представлял собой небольшую частную сеть, то теперь это гигантская система взаимосвязанных компьютеров, без которой, возможно, мы не сможем представить себе жизнь.

Web-технология полностью перевернула представления о работе с информацией, да и с компьютером вообще. Оказалось, что традиционные параметры развития вычислительной техники - производительность, пропускная способность, емкость запоминающих устройств - не учитывали главного "узкого места" системы - интерфейса с человеком. Устаревший механизм взаимодействия человека с информационной системой сдерживал внедрение новых технологий и уменьшал выгоду от их применения. И только когда интерфейс между человеком и компьютером был упрощен до естественности восприятия обычным человеком, последовал беспрецедентный взрыв интереса к возможностям вычислительной техники.

Информация, доступная пользователям Internet, располагается на компьютерах (Web-серверах), на которых установлено специальное программное обеспечение. Значительная часть этой информации организована в виде Web-сайтов. Каждый из них имеет свое имя (адрес) в

Internet. Web-сайт – это информация, представленная в определенном виде, которая располагается на Web-сервере и имеет свое имя. Для просмотра Web-сайтов на компьютере пользователя используются специальные программы, которые называются браузерами. Наиболее распространенными браузерами в настоящее время являются InternetExplorer и NetscapeNavigator. Взависимости от того, какое имя сайта мы зададим в строке "Адрес", браузер будет загружать в свое окно соответствующую информацию.

Web-сайт состоит из связанных между собой Web-страниц. Web-страница представляет собой текстовый файл с расширением *.htm, который содержит текстовую информацию и специальные команды – HTML-коды, определяющие в каком виде эта информация будет отображаться в окне браузера. Вся графическая, аудио и видео информация непосредственно в Web-страницу не входит и представляет собой отдельные файлы.

Современные web-приложения — это, в основном, порталы, предоставляющие услуги, которыми нельзя воспользоваться откуда-либо еще, Одно из неудобств подобных сервисов — сложность обмена информацией между компаниями. В частности, даже контактную и другую личную информацию приходится на каждом таком сайте вводить заново.

В настоящее время с точки зрения назначения различают три основных типа порталов:

- Публичные, или горизонтальные, порталы (называемые иногда мегапорталами), такие как Yahoo, Lycos, Excite, Rambler. Такие порталы нередко являются результатом развития поисковых систем. Предназначены они для самой широкой аудитории, что отражается на содержании предоставляемой ими информации и услуг. Как правило, эта информация носит общий характер, равно как и предоставляемые услуги (электронная почта, новостные рассылки и так далее).

- Вертикальные порталы. Этот вид порталов предназначен для специфических видов рынка и обслуживает аудиторию, пользующуюся услугами этого рынка или работающую на нем. Примерами таких порталов

могут служить, например, туристические агентства, предоставляющие услуги по бронированию мест в гостиницах, заказу и доставке билетов, доступу к картам и сведениям об автомобильных маршрутах, либо порталы типа B2B (business-to-business), позволяющие своим клиентам реализовывать совместные бизнес-операции (например, выбирать поставщиков и осуществлять закупку товаров, проводить аукционы).

- Корпоративные порталы предназначены для сотрудников, клиентов и партнеров одного предприятия. Пользователи такого портала получают доступ к предназначенным им сервисам и приложениям в зависимости от их роли и персонального профиля.

Другие наиболее распространённые web-приложения:

- Региональные Интернет-порталы, универсальные по своему направлению, но ограниченные географией заинтересованных посетителей;
- Поисковые системы — это Интернет-порталы, которые предназначены для того, чтобы предоставить их посетителю возможность найти сайты, на которых встречаются заданные слова или целые фразы (metabot.ru);
- Каталог — это коллекция ссылок на сайты. Зачем же нужны каталоги, если есть поиск? Очень часто мы не знаем точно, что нам нужно, не можем это сформулировать парой слов (mail.ru);
- Электронные доски объявлений - являются местом в Интернет, где практически любой желающий может оставить информацию ознакомительного, пригласительного или рекламного характера;
- Форумы — это специальные сайты или разделы на сайтах, предназначенные для того, чтобы посетители, оставляя свои сообщения, обменивались мнениями;
- Чаты - являются еще одним местом для общения в Интернет, только его назначение не обмен мнениями на какую-то тему, а просто времяпрепровождение;
- Файлы для скачивания;

- Фотогалереи;
- Элементы статистики;
- Хранение в интернете различной информации;
- Серверы почтовых рассылок, они предлагают услуги по доставке информации широкому кругу читателей (subscribe.ru);
- Интернет-магазины и аукционы (ozon.ru, molotok.ru).

Требования к Web-приложениям

Отправной точкой в web-проекте является анализ целей сайта и функций, которые будут предложены пользователю.

Вторым этапом будет построение информационной архитектуры сайта.

После того как будут известны все материалы сайта и его структура, можно перейти к дизайну навигации и самих страниц.

Карта сайта

Необходимо разместить хорошо различимую ссылку на карту сайта на каждой странице. Страница с отображением карты по размеру не должна превышать двух страниц на экране пользователя. Используется два вида карт статические и динамические. Наиболее эффективно использование карты со статическим отображением информационной архитектуры, так как динамический вид приводит пользователя ещё в большую запутанность. Карта создана для того, чтобы лучше сориентировать пользователя, поэтому на карте необходимо отображать не только текущее положение пользователя на сайте, но и те разделы, которые он уже посещал.

Главная страница

Главная страница сайта компании – это лицо сайта, обращённое ко всей сети (рисунок 2 Приложение Б). На главной странице чётко прописываются цели компании, при этом каждый элемент дизайна должен позволять пользователю ещё лучше разобраться в сайте и определиться с последовательностью действий для решения своих задач.

Именно с главной страницы большинство пользователей начинает путешествие по сайту. При этом наш сайт должен быть сделан так, чтобы

люди, пришедшие на сайт по глубокой ссылке, чувствовали себя комфортно. Для этого необходимо на каждой странице сайта разместить:

· название компании или логотип в левом верхнем левом углу

· прямую ссылку на главную страницу

· индивидуальный слоган для сайта, состоящий из одного предложения

Нельзя предполагать, что пользователь попал на данную страницу, преодолев весь путь от начала до конца. Вполне возможно, что он не видел той информации, которую мы указали на страницах верхнего уровня иерархии. При глубоких ссылках мы должны сориентировать пользователя по месту, сообщить ему, где он сейчас находится на сайте, указав на странице цепочечную ссылку. Если есть необходимость, чтобы какая-то страница не индексировалась в поисковиках, необходимо в элементе "HEAD" ввести <META NAME="robots" CONTENT="noindex">.

"О нас"

В разделе "О нас" необходимо представить основные сведения о компании:

· состав верхнего управляющего звена (с библиографическим списком и фотографиями);

· контактная информация;

· философия организации (видение бизнеса, связи с обществом, стремления);

· основные исторические вехи;

Пространное объяснение в верхней части страницы "О нас" помогает пользователю лучше понять то, что содержится в глубинах, этого раздала.

Контент

Текст должен быть легко читаемым, как молодыми, так и престарелыми пользователями, поэтому нам потребуется указывать размер шрифта в относительных единицах, а не в абсолютных, так как размер должны выбирать сами пользователи. При этом текст должен полностью

умещаться на странице, что бы ни раздражать использованием горизонтальной прокрутки страниц сайта.

Пользователи редко читают web-страницы слово за словом. При первом просмотре выискивает факты и игнорирует детали, но найдя интересующий материал, может зайти и глубже. Примерная схема просмотра страницы выглядит так: первая строчка по горизонтали просматривается полностью, следующая просматривается не на всю глубину и с каждой последующей строкой выхватывание информации сокращается всё больше.

Поэтому пользователи часто просматривают только верхнюю часть статьи. И лишь самые заинтересованные читатели промотают страницу, и лишь эти немногие поймут историю статьи во всех деталях. Поэтому важно применять "принцип перевёрнутых пирамид", который состоит в том, что статья начинается с "вывода", после идёт сообщение о самой важной информации, а в конце даётся подоплёка события.

Для того, чтобы страницы было удобно просматривать, мы будем использовать:

· выделение ключевых слов;

· грамотно составленные заголовки;

· списки с маркерами;

· один абзац на одну идею (пользователи пропустят все остальные идеи, если их не привлекут основные слова абзаца);

· стиль перевёрнутой пирамиды (где статья начинается с заключения);

· меньшее количество слов, чем в обычной статье;

· минимум мало понятных терминов;

· указание даты создания сообщения (что бы знать на сколько, оно актуально на данный момент);

· контекстную информацию атрибута " ALT".

· электронные заголовки часто выводятся вне контекста:

· в списке статей на сайте;

· в заголовках сообщений электронной почты;

· в результатах поиска;

· в закладках браузера.

При этом пользователи не должны обладать какими-то сверхзнаниями, чтобы понять наш заголовок, если они будут пропускать текст под заголовками. По этой причине текст заголовка должен быть самостоятельной порцией информации, позволяющей понять его в отсутствие остальной части контента. При написании заголовка необходимо использовать нейтральный язык, и не применять: игры слов, "навороченности", "завлекалочек", прописных букв (так как создаётся впечатление, что мы кричим на пользователя)

Пользователи предпочитают содержательные заголовки. Чем короче микроконтент, тем легче его просматривать. Необходимо исключить использование одного и того же слова в начале каждой статьи и названия страницы, так как при использовании одновременно нескольких таких статей, будет трудно на панели или закладках выбрать необходимую статью.

Почта

При просьбе предоставить нам адрес электронной почты, необходимо сказать, здесь и сейчас, что именно пользователь будет получать, и как часто будете нарушать спокойствие его почтового ящика. Ссылки mailto должны быть использованы так, чтобы можно было точно понять, что перед нами почтовый адрес. Не размещать ссылки почты на именах, так как щелчок по имени должен вести на его биографию. При общении по электронной почте нет причин пользоваться конкретным именем одного из работников, если только пользователь не установил с ним личные отношения (письма от неизвестных людей чаще всего попадают в корзину).

Что бы поднять доверие к сайту необходимо использовать качественные графические изображения. При этом не стоит использовать большие изображения, по умолчанию. Желательно использовать эскизы, а при необходимости увеличивать их на столько, на сколько пожелает пользователь. Эскизы должны содержать всю основную информацию об

изображении. На сайте правильнее будет использовать фотографии реальных работников и клиентов, а не фотомоделей с ослепительной голливудской улыбкой. Так же необходимо использовать хороший стиль письма и исходящие гипертекстовые ссылки на другие сайты. Не нужно использовать "рекламную воду", так как она несёт дополнительную эмоциональную нагрузку, и пользователям приходится тратить время на отсеивание гипербол от реальных фактов.

Привязанность

Классический путь увеличения привязанности к сайту – это публикация свежих материалов с определённой регулярностью (от ежедневных до ежемесячных). Для желающих необходимо организовать рассылки свежего материала. Почти 100% из тех, кто подписался, будут периодически заходить и на сам сайт, пока автор будет напоминать им о себе.

Дизайн сайта должен приспосабливаться к людям. Одним из вариантов такого приспособления служит профайл, но его заполнение требует терпения, из-за этого его заполняют устоявшиеся посетители сайта. Поэтому для отслеживания пользователей лучше применять "cookie", в файл которого заносится вся необходимая информация, а не нудную процедуру регистрации. Если всё же необходимо для безопасной работы регистрация, то не стоит создавать слишком строгие формы для ввода. Если есть возможность использовать автозаполнение и автоматически устанавливать курсор в первом поле формы.

Не стоит создавать формы для поиска по Internet, если пользователю будет необходимо, он воспользуется специальной поисковой системой.

Ссылки

Ссылки должны быть предсказуемы, пользователь должен знать, что произойдёт, если он нажмет на ссылку. Ясно указывать, что является ссылкой: если это текст пусть он будет цветным или подчёркнутым. Не скрывать различия между посещёнными и не посещёнными ссылками.

Создавать ссылки с расчётом того что поисковые системы являются самыми важными посетителями, а они глухи и слепы по отношению к высоким технологиям. Оформить теги "title" так, чтобы они хорошо смотрелись в поисковых системах и закладках, поэтому начинать ссылки необходимо с ключевого слова.

Не допускать появления ломаных ссылок, так как другие сайты и поисковые системы не будут в этом случае на нас ссылаться.

При навигации по сайту предоставить пользователю самому, выбирать, как необходимо открывать страницы либо во всплывающих окнах, либо как-то иначе.

Реклама

В WEB-е пользователи не обращают внимания ни на что, кроме своей цели, и реклама, которая отвлекает их от достижения цели, ими попросту игнорируется. Если наша реклама появится раньше, чем пользователь найдёт интересующий его материал, то, скорее всего она будет закрыта. И когда он выполнит свою основную задачу, и готов будет присмотреться к рекламе, он её попросту уже не увидит так как закрыл всплывающее окно. Поэтому никогда не стоит отвлекать пользователя от своей цели.

Когда пользователь набирает слова запроса на поисковом сервере, рекламные объявления которые выйдут в списке результатов запроса, будут непосредственно связаны с тем, что нужно человеку. И поэтому он читает эти объявления и щёлкает по ссылкам. Такая реклама стоит гораздо дешевле, чем распространение рекламных листовок.

Итак, секрет успеха в том, что бы совместить рекламу с целями пользователей.

В рекламе пользователей больше всего раздражает:

· реклама во всплывающих окнах;

· её медленная загрузка;

· отсутствие кнопки "Закрыть";

· уловки заставляющие кликнуть;

· реклама закрывает читаемый материал;

· мигающая реклама;

· двигающаяся реклама;

· реклама произвольно начинающая играть музыку или проигрывать видео.

Наличие такой рекламы на сайте неизбежно приведёт к падению доверия посетителей. Необходимо использовать в минимальных количествах FLASH анимацию.

Если хочется разместить, рекламу она должна соответствовать следующим правилам:

· четко указывать, что произойдёт по щелчку по ней;

· четко идентифицировать себя как реклама;

· описывать, что именно рекламируется;

· даёт дополнительную информацию о себе без надобности покидать текущую страницу.

Контрольные вопросы

1. Отличительные особенности процесса разработки веб приложений от настольных.

2. Какие основные языки используются при разработке веб приложений?

3. Перечислите фундаментальные концепции веб программирования.

4. Что такой web браузер?

5. В чем отличие Wiki от блога?

6. Мультимедиа. Что может содержать мультимедиа контейнер?

Глава 2. Языки разметок HTML, XML, XHTML, WML.

Все программы, выполняющиеся на компьютере, написаны на каком-либо языке программирования. Web-страницы здесь не исключение - для их создания также применяют специальный язык программирования, который называется "языком разметки". Почему разметки? Потому что главная задача

этого языка состоит в «размещении» (разметке или верстке) визуальной информации на Web-странице. Все языки разметки очень просты в изучении и вы без труда их освоите.

HTML (HyperTextMarkupLanguage)

Первоначально система WorldWideWeb предназначалась для отображения текстовой информации, и **HTML** разрабатывался как язык форматирования текста. В настоящее время HTML является самым популярным языком в WWW. Документ, написанный на языке HTML, представляет собой **текстовый файл, содержащий данные и инструкции по их форматированию**.

Средства, предназначенные для связывания документов и форматирования данных, называются **тэгами** (tag).

Текст Web-страницы и тэги размещаются в одном файле, который называется **HTML-документом**. От того, какие тэги вы выберете и как их примените, зависит внешний вид Web-страницы в окне броузера. Для форматирования и размещения данных в языке HTML предусмотрены сотни тэгов. Например, тэги **<p>** и **</p>** формируют абзац, а пара тэгов **<i>** и **</i>** указывает на то, что текст, содержащийся между ними, должен отображаться курсивом. Тэги также используются для создания **гипертекстовых ссылок**. Благодаря наличию гипертекстовых ссылок пользователь может вызвать другой документ щелчком мыши на фрагменте текста или изображении.

Язык HTML постоянно совершенствуется. Устаревшие элементы удаляются, а новые включаются в состав языка. Несмотря на это, структура HTML остается неизменной.

XML (eXtensibleMarkupLanguage)

XML похож на язык HTML тем, что для описания различных разделов документа в нем используются тэги. В отличие от HTML, XML позволяет разработчикам определять собственные тэги и ставить в соответствие им

способы воспроизведения информации. XML-тэги чувствительны к регистру символов.

XHTML

XHTML представляет собой сочетание HTML и XML. XHTML 1.0 в настоящее время является стандартным языком разметки и рекомендован W3C для использования вместо HTML, но на сегодняшний день большинство разработчиков продолжают применять HTML. Язык XHTML гарантирует, что внешний вид документа не будет изменяться в зависимости от платформы (Windows, Mac или Unix). Несмотря на различия в структуре, в XHTML используются дескрипторы HTML, которые распознаются современными браузерами, т.е. имеющиеся в наличие браузеры по сути поддерживают XHTML 1.0.

В настоящее время разрабатываются новые языки разметки, такие как, **WML (WirelessMarkupLanguage)** применяется для создания информационных ресурсов, предназначенных для владельцев сотовых телефонов; **CDF (ChannelDefinitionFormat)**, используемый для создания push-каналов в браузерах производства Microsoft; язык **SMIL (SynchronizedMultimediaInterchangeLanguage)** служит для создания презентаций и поддерживается программой RealPlayer.

WML

WML (беспроводной язык разметки) применяется для создания информационных ресурсов, предназначенных для владельцев сотовых телефонов. К счастью, если вы имеете уже созданную HTML-страницу, и хотите, чтобы она была доступна через WAP-браузер сотового телефона, Вам не нужно изучать новый язык. Существуют специальные программы, позволяющие конвертировать HTML-код в WML.

Кроме перечисленных выше языков разметки, для каждой предметной области существуют специализированные языки (**MathML** - для описания математических формул, **VRML** - для создания трехмерных объектов и т.д.)

И HTML, и XHTML - это языки для создания веб-страниц. HTML построен на основе SGML, а XHTML - на основе XML. Они похожи на две стороны одной медали. XHTML был создан из HTML с целью соответствия стандартам XML. Следовательно, XHTML является более строгим по сравнению с HTML и не позволяет отступать от правил написания кода.

Причиной разработки XHTML послужила путаница с некоторыми тегами. Страницы, написанные на HTML, выводились в разных браузерах по-разному.

Сравнительная таблица

	HTML	XHTML
Определение (из Википедии)	HTML или HyperTextMarkupLanguage - это основной язык разметки для создания веб-страниц и других документов, которые могут быть просмотрены в браузере.	XHTML (ExtensibleHyperTextMarkupLanguage) - это семейство языков XML-разметки, которые продолжают и расширяют HypertextMarkupLanguage (HTML), на котором написаны веб-страницы.
Расширения файлов	.html, .htm.	.xhtml, .xht, .xml, .html, .htm.
Формат использования	текст/html.	приложение/xhtml+xml.
Разработан	W3C и WHATWG.	WorldWideWebConsortium.
Тип формата	Формат документов.	Язык разметки.

Расширен из	SGML.	XML, HTML.
Расшифровка	Язык разметки гипертекста.	Расширяемый язык разметки гипертекста.
Приложение	Приложение Standard Generalized Markup Language (SGML).	Приложение XML.
Функции	Веб-страницы написаны на HTML.	Расширенная версия HTML, более строгая, основанная на XML.
Поведение	Гибкие фреймворки не требуют анализа синтаксиса HTML.	Ограничен правилами XML и требует их соблюдения.
Происхождение	Предложен ТимомБернерсом-Ли в 1987 году.	Рекомендация World Wide Web Consortium 2000 года.
Версии	HTML 2, HTML 3.2, HTML 4.0, HTML 5.	XHTML 1, XHTML 1.1, XHTML 2, XHTML 5.

Обзор HTML и XHTML

HTML является основным языком разметки веб-страниц. Он создает структурированные документы, выделяя в них такие элементы, как заголовки, списки, ссылки, цитаты и т.д. Это позволяет встраивать изображения и объекты для создания интерактивных форм. HTML задается с помощью тегов в угловых скобках - например, <html>. Также в его коде могут содержаться скрипты, написанные на JavaScript.

XHTML представляет собой семейство языков XML, которые расширяют или продолжают версии HTML. Они не допускают пропусков любых тегов или минимизации атрибутов. XHTML требует, чтобы каждому

открывающемуся тегу соответствовал закрывающийся тег в корректном порядке. Например, если в языке гипертекста допускается использование одиночного тега <br>, то в XHTML в отличие от HTML нужно написать тег <br/>. В этом и заключается отличие.

Функции документов HTML и XHTML

Синтаксис HTML состоит из следующих компонентов: открывающий и закрывающий тег, атрибуты элементов (*задаваемые в тегах*), текстовый и графический контент. HTML-элемент - это все, что находится между тегами, включая сами теги.

Документ XHTML содержит только один корневой элемент. Все элементы, включая переменные, должны быть написаны в нижнем регистре, а присвоенные значения - заключены в кавычки, закрыты и вложены. В XHTML это является обязательным требованием - в отличие от HTML. Объявление DOCTYPE XHTML определяет правила для документов, которым необходимо следовать.

Основной синтаксис HTML допускает использование множества сокращений, чего не допускается в XHTML. Например, элементов, для которых необязательно наличие и открывающегося, и закрывающегося тега. XHTML требует, чтобы все элементы имели и открывающийся, и закрывающийся тег. В то же время XHTML вводит новые сокращения: тег XHTML может быть открыт и закрыт с помощью косой черты (<br/>).

Введение такого синтаксиса, который не используется в объявлениях SGML для HTML 4.01, могло привести к путанице в приложениях на ранних стадиях. Чтобы решить эту проблему, нужно использовать пробел перед закрытием тега:

Спецификация XHTML и HTML

HTML и XHTML могут быть задокументированы совместно. И HTML 4.01, и XHTML 1.0 имеют три подспецификации - строгую, нестрогую и фрэймовую. Отличие документов HTML и XHTML заключается в декларировании документов. Другие отличия

синтаксические. HTML допускает отсутствие закрывающегося тега, пустые элементы без закрывающегося тега. Расширяемый язык разметки гипертекста очень строг в отношении открывающихся и закрывающихся тегов XHTML. Он использует встроенный язык определения функционала атрибутов. Все требования к синтаксису XML соблюдаются в XHTML-документе.

Но эти различия проявляются только тогда, когда XHTML-документ используется как приложение XML; то есть как MIME-типы приложение / XHTML + XML, приложение / XML или текст / XML. Документ XHTML, используемый как MIME-тип текст / HTML должен интерпретироваться как HTML, так что в данном случае применяются правила HTML. CSS, написанный для XHTML, используемого, как MIME-тип текст / HTML, может работать некорректно в документе, который применяется как, как MIME-тип приложение / XHTML + XML. Для получения дополнительной информации о MIME-типах ознакомьтесь с соответствующей документацией.

Это может быть важно, когда вы используете документы XHTML, как текст / HTML. Если не знать о данных различиях, вы можете создать CSS, который не будут работать как ожидается, если документ используется, как XHTML.

Там, где встречаются термины "*XHTML*" и "*XHTML document*", предполагается, что в оставшейся части этого раздела они определяют использование разметки XHTML, как MIME-тип XML. XHTML-разметка, используемая в качестве текста / HTML, является HTML-документом.

Как перейти с HTML на XHTML

В соответствии с рекомендациями W3C для перехода с HTML на XHTML (*документы XHTML 1.0*) должны быть выполнены следующие шаги:

- Включите атрибуты xml:lang и lang для элементов, устанавливающих язык;

- Используйте синтаксис пустого элемента для элементов, указанных в HTML, как пустые;
- Используйте дополнительный пробел в тегах пустых элементов: <html />;
- Используйте закрывающиеся теги для элементов, которые могут содержать контент, но являются пустыми: <html></ html>;
- Не включайте объявление XML.

Если следовать рекомендациям W3C по совместимости, то браузер должен уметь интерпретировать документы как HTML, так и XHTML.

Чтобы понять, чем отличается HTML от XHTML, рассмотрим преобразование документа XHTML 1.0 в HTML 4.01. Для этого необходимо выполнить следующие действия:

- Язык для элемента должен быть указан с помощью атрибута lang, а не атрибута XHTML xml:lang;
- Удалите пространство имен XML (xmlns=URI). HTML не имеет средств для работы с пространствами имен;
- Измените объявление типа документа с XHTML 1.0 на HTML 4.01;
- Удалить объявление XML, если оно присутствует. Как правило, это: <?xmlversion="1.0" encoding="utf-8"?>;
- Убедитесь в том, что для MIME-типа документа задано: text/html. И в HTML, и в XHTML, это задается в HTTP-заголовке Content-Type, отправляемом сервером;
- Измените синтаксис пустого элемента XML на стиль пустого элемента HTML (с
 на <br>).

Глава 3.Языки сценариев

Язык сценариев (или скриптовый) – это язык программирования, который разработан для записи последовательностей операций («сценариев»), выполняемых пользователем на своем компьютере. Раньше назывался языком пакетной обработки.

Сценарий или скрипт – это программа, автоматизирующая задачу. Без скрипта пользователю бы пришлось ее выполнять вручную с помощью интерфейса программы.

Скрипты используют для раскрутки сайта в следующих целях:

• для расширения функционала ресурса;

• для автоматического обмена ссылками;

• для выполнения работ по поисковой оптимизации;

• для сбора статистики, анализа позиций сайта и т.д.

Выделяют следующие типы скриптовых языков:

Для создания пользовательских расширений язык сценариев удобен в ряде случаев:

• безопасность. Скриптовый язык обеспечивает программируемость без риска дестабилизации системы. Скрипты не компилируются, а интерпретируются. Поэтому неправильно написанная программа выведет диагностическое сообщение, не вызывая падение системы;

• наглядность. Язык сценариев используется, если необходим выразительный код. Концепция программирования в скриптовом языке может кардинально отличаться от основной программы;

• простота. Код имеет собственный набор программ, поэтому одна строка может выполнять те же операции, что и десятки строк на обычном языке. Поэтому для написания кодов не требуется программист высокой квалификации;

- кроссбраузерность. Скриптовые языки программирования ориентированы на кроссбраузерность. Например, JavaScript может исполняться браузерами практически под всеми современными операционными системами.

Классификация

В зависимости от быстродействия различают языки сценариев предварительно компилируемые (например, широко используемый для создания и продвижения сайтов Perl) и динамического разбора (command.com, sh). Первые транслируют программу в байт-код, который затем исполняют. Языки динамического разбора считывают инструкции из файла программы минимально необходимыми блоками, которые исполняют, не читая, дальнейший код.

- универсальные: Forth, AngelScript, Perl, PHP, Python, Tcl (Tool command language), Squirrel, REBOL, Ruby, AutoIt, Lua;
- встроенныевприкладныепрограммы: VBA, UnrealScript, AutoLISP, Emacs Lisp, Game Maker Language, MQL4 script, ERM;
- командныеоболочки: sh, AppleScript, bash, csh, ksh, JCL, cmd.exe, command.com, REXX, Visual Basic Script;
- встраиваемые: Guile, Script.NET, ActionScript, Lingo (используетсявредакторе Director), Sleep, браузерные Jscript и JavaScript.

Некоторые приложения имеют встроенную возможность расширения сценариями, написанными на любом универсальном скриптовом языке, например, автоматический планировщик задач или библиотека SWIG.

К скриптам также относят многие консольные утилиты, которые поддерживают выполнение записанной в файл последовательности команд.

Скриптовые языки быстро становятся языками общей реализации для многих областей, блистая там, где время разработчика более важно, чем время исполнения (и даже там, где важно время исполнения; например,

благодаря встроенным операциям высокого уровня быстродействие программ, написанных на Python, такое же, или даже быстрее, чем программ, написанных на Java). Многие сейчас предпочитают использовать обозначение "динамические языки" вместо "скриптовые языки", ссылаясь на отсутствие выполняемого в процессе компиляции контроля типов[2]. Какое место занимают скриптовые языки в современных компьютерных кругах?

- Скриптовые языки позволяют разработчикам сцеплять вместе различные пакеты программ, а также согласовывать полученные в результате системы.
- Все чаще скриптовые языки сами по себе используются в качестве полноценных базовых инструментальных платформ. Например, многие крупные коммерческие Интернет-приложения сейчас программируются преимущественно на языках Perl, Python или PHP.
- Естественно, скриптовые языки используются для автоматизации задач системного администрирования.

Не исключено, что интерпретируемые или оперативно компилируемые языки все больше и больше будут заступать на смену предварительно компилируемым языкам. Компиляцию со временем будут рассматривать просто как инструмент оптимизации (чем она собственно и является), использование которого во всех случаях едва ли разумно. Она все еще будет полезна при отправке автономного кода за пределы среды, которой вы управляете, однако, компиляция все чаще будет рассматриваться просто как способ упаковки. С другой стороны, граница между компиляцией и интерпретацией, которая всегда была немного произвольна, будет размыта еще больше. У языка Perl уже есть фаза оперативной компиляции перед интерпретацией. Будущее будет за совместимостью платформ, так что компиляторы все чаще будут нацелены на абстрактные "виртуальные машины" (как JVM у Sun или CLR у Microsoft), которые наслаиваются на аппаратные средства. На каком этапе программирования вы компилируете или интерпретируете? И имеет ли это значение?

Динамические языки в качестве доминирующих языков реализации во многих областях могут со временем перегнать Java и C++. Закон Мура[3] на стороне динамических языков.

Статические языки во время компиляции пытаются поставить все точки над "i". Долгое время разработчики полагали, что безопасность типов статических языков означает большую надежность их кодов. Однако все чаще разработчики приходят к заключению, что дело не в этом. Конечно, теоретически возможно иметь в распоряжении переменную под названием "ИмяПользователя", но обнаружить во время исполнения, что она ссылается на объект класса "ЗаказНаПоставку". Однако на практике подобное все же маловероятно.

В чем же состоит привлекательность скриптовых языков?

- Скриптовые языки обладают более сложным инструментарием и поддерживают более прогрессивные техники программирования. Например, возможности сортировки данных в Perl встроены прямо в язык. То, что в язык встроены все основные инструменты программирования, избавляет от необходимости создавать их самостоятельно и означает, что для решения конкретной проблемы нужно писать меньше кода, что увеличивает производительность разработчика.
- Скриптовые языки позволяют быстро выполнять доработку кода без раздражающей потери времени на ожидание окончания компиляции.
- Количество людей, не обладающих подготовкой, которую имеют традиционные компьютерные специалисты, но могущих заняться написанием скриптов, стало на порядок больше. Иначе говоря, программированию на скриптовых языках проще научиться. Чтобы стать средним программистом на C++, необходим больший опыт работы, чем для того, чтобы стать средним программистом на PHP.

Какие можно назвать недостатки скриптовых языков?

- Время исполнения все еще является главной проблемой. Конечно, есть области, где скорость слишком важна, чтобы можно было

программировать непосредственно на скриптовом языке. Эта проблема обычно решается тем, что код тщательно выбранной части приложения (скажем, 10-30%) пишется на языке низкого уровня (таком, как C или C++); например, в Python есть развитые механизмы для того, чтобы вставить такой код (как и в большинстве других динамических языков).

- Общей проблемой всех скриптовых языков является отсутствие хорошей интегрированной среды разработки (IDE). Конечно, какие-то интегрированные среды разработки существуют, однако в них недостает мощности, как у VisualStudio.
- Ключевым нетехническим, однако важным недостатком является отсутствие маркетингового бюджета. Многие динамические языки идеально подходят для многих проектов, однако им тяжело конкурировать с такими локомотивами маркетинга, как Sun (Java) и Microsoft (C#), которые продолжают продвигать свои технологии как единственно возможные. В истории есть примеры того, как техническое превосходство подавляется превосходным маркетингом.

Ниже следует обзор некоторых наиболее популярных скриптовых языков.

VBScript

VisualBasicScriptingEdition (или просто VBScript) — это язык программирования от компании Microsoft, предназначенный для создания сценариев (скриптов). Он является подмножеством языка VisualBasic и широко используется при создании административных сценариев в системе Windows. VBScript по умолчанию поддерживается в WindowsScriptHost (WSH), который в свою очередь по умолчанию устанавливается вместе с почти любой версией Windows. Если у вас слишком старая версия Windows, вы можете скачать WSH с сайта Microsoft и самостоятельно установить его.

Синтаксис VBScript является несколько упрощенной версией стандартного синтаксиса VisualBasic. Например, в VBScript не

поддерживается типизация: все переменные имеют тип Variant. Сценарии на языке VBScript чаще всего используются в следующих областях:

- Автоматизация администрирования систем Windows.
- Серверный программный код на страницах ASP в Web-приложениях.
- Клиентские сценарии на Web-страницах (в основном только в браузере InternetExplorer).

JScript

JScript — это язык программирования от компании Microsoft. Он предназначен для создания сценариев и является реализацией стандарта ECMAScript. Синтаксис JScript во многом аналогичен языку JavaScript от компании Netscape. JScript по умолчанию поддерживается в WindowsScriptHost (WSH), который в свою очередь по умолчанию устанавливается вместе с почти любой версией Windows. Если у вас слишком старая версия Windows, вы можете скачать WSH с сайта Microsoft и самостоятельно установить его.

Сценарии на языке JScript чаще всего используются в следующих областях:

- Клиентские сценарии на Web-страницах.
- Автоматизация администрирования систем Windows.
- Серверный программный код на страницах ASP в Web-приложениях.

Язык JScript получил дальнейшее развитие в виде языка JScript.NET, который ориентирован на работу в рамках платформы Microsoft ASP.NET.

JavaScript

JavaScript - это язык программирования от компании Netscape, который является реализацией стандарта ECMAScript. Microsoft выпустила похожие

версии языка под названием JScript, поэтому под названием "JavaScript" часто понимается любая версия языка, в том числе и MicrosoftJScript.

В большинстве случаев при упоминании JavaScript подразумевается так называемый клиентский JavaScript, интерпретатор которого встроен в Web-браузеры. Однако JavaScript изначально был разработан как универсальный язык программирования для встраивания в любое приложение и обеспечения возможности написания в нем сценариев. Например, ActionScript, язык сценариев, доступный в MacromediaFlash 5 и MX, также смоделирован в соответствии со стандартом ECMAScript.

Интерпретатор JavaScript от Netscape был выпущен в виде открытого исходного кода и доступен через организацию Mozilla (http://www.mozilla.org/js/). Mozilla предоставляет две различные версии интерпретатора JavaScript - "SpiderMonkey" (написана на C) и "Rhino" (написана на Java).

Вопреки распространенному заблуждению, кроме некоторой синтаксической схожести, языки Java и JavaScript ничего не связывает. Схожесть имен - не более, чем уловка маркетологов (первоначальное название языка - LiveScript - было изменено на JavaScript в последнюю минуту).

Python

Python (питон) — интерпретируемый, объектно-ориентированный язык программирования высокого уровня. Он поддерживает классы, модули (которые могут быть объединены в пакеты), обработку исключений, а также многонитевую обработку. Python относится к классу языков с динамической типизацией, предоставляет программисту автоматическую «сборку мусора» и удобные высокоуровневые структуры данных, такие как словари, списки, кортежи и др. Питон объединяет поразительную мощь с простым и ясным синтаксисом, продуманной модульностью и масштабируемостью. Одной из интересных синтаксических особенностей языка является выделение блоков

программы с помощью отступов (пробелов или табуляций), поэтому в Python отсутствуют операторные скобки ("begin/end", как в языке Паскаль или фигурные скобки, как в Си). Python — одно из самых простых средств обучению и применению ООП. Часто является как первым (для обучения), так и последним (в череде используемых опытными программистами) языком программирования.

Pythonпортируеми работает почти на всех известных платформах. Существуют порты под Windows, все варианты UNIX (включая Linux), Mac OS и Mac OS X, Palm OS, OS/2 и т.д. При этом, в отличие от многих портируемых систем, на каждой платформе Python поддерживает все характерные для данной платформы технологии (например, Microsoft COM/DCOM). Более того, существует специальная версия Python для виртуальной машины Java — Jython (http://www.jython.org/), что позволяет интерпретатору выполняться на любой системе, поддерживающей Java, при этом классы Java могут непосредственно использоваться из Python и даже быть написанными на Python.

Интерпретатор языка Python распространяется свободно на основании лицензии PythonSoftwareFoundation (PSF) Licence (http://python.org/psf/license.html), которая в некотором роде даже более демократична, чем GNU GeneralPublicLicense (http://gnu.org/copyleft/). Официальный сайт проекта языка Python располагается по адресу http://python.org/. Здесь же в разделе «Download» можно скачать свежую версию для вашей операционной системы. Русскоязычные сайты, посвященные Python: http://www.python.ru/, http://zope.net.ru/.

В стандартный комплект поставки Python входит интегрированная среда разработки IDLE, в которой редактировать программы будет намного удобнее, чем в простом текстовом редакторе. IDLE написан на Python с использованием платформонезависимой библиотеки Tcl, поэтому легко запускается в любой операционной системе, для которой существует реализация Python. IDLE также имеет встроенную систему отладки.

Стандартная библиотека языка Python богата и предоставляет программисту множество возможностей. Однако, если вам не достаточно возможностей стандартной библиотеки, то существует множество библиотек, предоставляющих интерфейс ко всем мыслимым системным вызовам на разных платформах; в частности, на платформе Win32 поддерживаются все вызовы Win32 API, а также COM в объёме не меньшем, чем у VisualBasic или Delphi. Кроме того, количество прикладных библиотек для Python в самых разных областях без преувеличения огромно (веб, базы данных, обработка изображений, обработка текста, численные методы, приложения операционной системы, и т. д.). Python легко расширяется языками C и C++, а на платформе Windows — также с помощью COM. Библиотека NumericPython для работы с многомерными массивами позволяет достичь производительности научных расчётов, сравнимой с MATLAB. Кроме того, существует специальная библиотека psyco (http://psyco.sf.net/), позволяющая оптимизировать выполнение некоторых программ, после чего скорость их выполнения можно сравнивать с программами на Си. В среде коммерческих приложений скорость выполнения программ на Python часто сравнивают с Java-приложениями. Существует реализация Python для .NET (http://ironpython.com/).

Для Python существуют библиотеки доступа к СУБД (на платформе Windows доступ к БД возможен через ADO). Существуют модули расширения для Python под Windows и Unix/Linux для доступа к Oracle, Sybase, Informix и MySQL. Существует также пакет mxODBC для доступа к СУБД через ODBC, также поддерживаемый на платформах Windows и Unix.

С Python поставляется библиотека tkinter для создания кроссплатформенных программ с графическим интерфейсом. Данная библиотека является фактически стандартом для GUI-приложений, написанных на Python. Многие люди при написании GUI программ пользуются также библиотекой wxPython, основанной на библиотеке wxWidgets. Также часто используются библиотеки PyQT (PyQt) и PyGTK.

Python и подавляющее большинство библиотек к нему бесплатны и поставляются в исходных кодах. Более того, в отличие от многих открытых систем, лицензионная политика на Python никак не ограничивает его использование в коммерческих системах и не налагает никаких обязательств, кроме указания авторских прав.

Tcl

Tcl (ToolCommandLanguage) — интерпретируемый язык программирования высокого уровня. Официальный сайт языка - http://www.tcl.tk/. Ссылки на русскую документацию можно найти здесь:http://www.opennet.ru/links/sml/36.shtml. Tcl ориентирован преимущественно на автоматизацию рутинных процессов ОС и крупных программных систем и состоит из мощных команд, ориентированных на работу с абстрактными нетипизированными объектами. Принципиальное отличие Tcl от командных языков ОС состоит в независимости от типа системы (когда не надо утруждать себя изучением нового командного языка) и, самое главное, он позволяет создавать переносимые программы с графическим интерфейсом (GUI).

Tcl очень часто применяется совместно с библиотекой Tk (ToolKit). Связку Tcl/Tk по-русски иногда называют "Так-тикль". Tcl/Tk распространяется в исходных текстах бесплатно. Tcl/Tk разрабатывался одновременно как язык и библиотека. Tk - это популярный графический инструментарий, позволяющий очень быстро создавать графические программы. Варианты Tcl/Tk доступны для множества платформ (Windows, Macintosh, практически все UNIX-платформы, включая Linux). Самые последние версии и полезные расширения Tcl доступны по адресу http://www.tcl.tk/. Библиотека Tk содержит стандартизованный набор команд поддержки GUI в стиле Motif. Управляющие элементы, хранящиеся в Tk, называются виджетами (widgets). Большое количество нетиповых виджетов можно найти в Сети.

Tcl - расширяемый язык. Можно самостоятельно определять новые команды языка (как в Форте). На Tcl написана оболочка VisualTcl, которая позволяет разрабатывать кроссплатформенное ПО для UNIX, Windows и Macintosh. Фирмой Sun разработана версия Tcl, написанная на Java - Jacl (JAvaCommandLanguage).

Ruby

Ruby — интерпретируемый скриптовый язык высокого уровня для быстрого и удобного объектно-ориентированного программирования. Ruby имеет большое количество средств для обработки текстов, для решения системных задач. Ruby является полностью свободным языком программирования с возможностью копирования, модификации и распространения. Ruby перенесён на множество платформ. Он разрабатывался на Linux, но работает на многих версиях Unix, DOS, Windows 95/98/Me/NT/2000/XP, Mac OS, BeOS, OS/2, и т.д. Целью создания Ruby был настоящий объектно-ориентированный интерпретируемый язык программирования. Название отсылает к языку Perl, наследником которого является Ruby (драгоценные камни: англ. pearl — жемчужина, англ. ruby — рубин).

Ruby имеет простой и понятный синтаксис, позволяет обрабатывать исключения в стиле Java и Python, позволяет легко переопределять операторы, которые на самом деле являются методами. Ruby — полностью объектно-ориентированный язык программирования. Все данные в Ruby являются объектами в понимании SmallTalk. Например, число «1» — это экземпляр класса Fixnum. Также поддерживается добавление методов в класс и даже в конкретный экземпляр во время исполнения программы. Ruby сознательно не поддерживает множественное наследование, вместо которого существует концепция модулей. Ruby содержит автоматический сборщик мусора. Он работает для всех объектов Ruby, так что не надо заботиться о подсчёте ссылок даже во внешних библиотеках. Ruby не требует объявления переменных. Язык использует простые соглашения для обозначения области

видимости. Пример: просто 'var' — локальная переменная, '@var' — переменная экземляра (член или поле объекта класса), '$var' — глобальная переменная. Ruby имеет независимую от ОС поддержку многопоточности.

Новости Ruby: http://rubynews.ru. Специализированный форум для программистов на языке Ruby: http://ruby-forum.ru/. Примеры конструкций языка: http://pleac.sourceforge.net/pleac_ruby/index.html.

PHP

PHP (пи-эйч-пи) — интерпретируемый скриптовый язык программирования, созданный для генерации HTML-страниц на веб-сервере и работы с базами данных. В области веб-программирования PHP является на сегодняшний день одним из самых распространённых технологий (наряду с Perl, ASP/.NET и Python) благодаря простоте, скорости выполнения и богатой функциональности. PHP распространяется свободно. Синтаксис языка похож на синтаксис C++. PHP поддерживается подавляющим большинством предоставителей сетевого хостинга.

Название "PHP" представляет собой самоповторяющую (рекурсивную) аббревиатуру и расшифровывается как "PHP: HypertextPreprocessor", или "PersonalHomePage". PHP был создан в качестве надстройки над Perl для облегчения разработки веб-страниц. За свою жизнь PHP значительно изменялся. Одной из сильнейших сторон PHP является возможность расширения ядра. Интерфейс написания расширений привлек к PHP множество сторонних разработчиков, работающих над своими модулями, что дало PHP возможность работать с огромным количеством баз данных, протоколов, поддерживать большое число API. PHP поддерживает ООП (деструкторы, открытые, закрытые и защищённые члены и методы, final-члены и методы, интерфейсы и клонирование объектов). PHP поддерживает XML.

Официальный сайт: http://php.net/. Русское руководство по PHP: http://ru.php.net/manual/ru/. Русскоязычный ресурс о PHP: http://www.phpclub.ru/.

Perl

Perl — интерпретируемый скриптовый язык программирования, один из самых распространённых в области веб-программирования. По одной из версий, Perl — аббревиатура, которая расшифровывается как "PracticalExtractionandReportLanguage" (практический язык извлечений и отчётов). Существует также ряд других вариантов. Согласно самому красивому из них, название "perl" произошло от слова "pearl" (жемчужина).

Основной особенностью языка считаются его богатые возможности для работы с текстом, реализованные при помощи регулярных выражений (regularexpressions). Перл также знаменит огромной коллекцией дополнительных модулей CPAN, находящейся по адресу http://www.cpan.org/.

Глава 4. Технология “клиент-сервер”

Применительно к системам баз данных архитектура "клиент-сервер" интересна и актуальна главным образом потому, что обеспечивает простое и относительно дешевое решение проблемы коллективного доступа к базам данных в локальной сети.

При работе с файл-серверной версией вся ответственность за сохранность и целостность базы данных лежит на программе и сетевой операционной системе. Обработка всех данных происходит на рабочих местах, а сервер используется только как разделяемый накопитель. Каждый пользователь непосредственно использует информацию и вносит изменения в файлы данных и в индексные файлы. При больших объемах данных и работе во многопользовательском режиме существенно снижается быстродействие - ведь чем больше пользователей, тем выше требования к разделению данных. Кроме того, может возникнуть повреждение баз данных. Например, в момент записи в файл может возникнуть сбой сети или авария

питания. В этом случае компьютер пользователя прерывает работу, база данных может оказаться поврежденной, а индексный файл - разрушенным. Переиндексация, которую необходимо провести после подобных сбоев, может длиться несколько часов.

Клиент-серверная версия позволяет обойти эти проблемы, так как вся работа с базой данных происходит на сервере, не проходит по проводам и не зависит от сбоев на рабочих станциях. Все запросы на запись в файл перехватываются сервером. В файл изменения вносятся только после того, как сервер получит сообщение о том, что корректировка файла завершена. Это исключает повреждение индексных файлов и существенно повышает быстродействие системы.

Архитектура "клиент-сервер"

Открытые системы

Реальное распространение архитектуры "клиент-сервер" стало возможным благодаря развитию и широкому внедрению в практику концепции открытых систем. Поэтому мы начнем с краткого введения в открытые системы.

Основным смыслом подхода открытых систем является упрощение комплексирования вычислительных систем за счет международной и национальной стандартизации аппаратных и программных интерфейсов. Главной побудительной причиной развития концепции открытых систем явились повсеместный переход к использованию локальных компьютерных сетей и те проблемы комплексирования аппаратно-программных средств, которые вызвал этот переход. В связи с бурным развитием технологий глобальных коммуникаций открытые системы приобретают еще большее значение и масштабность.

Ключевой фразой открытых систем, направленной в сторону пользователей, является независимость от конкретного поставщика. Ориентируясь на продукцию компаний, придерживающихся стандартов открытых систем, потребитель, который приобретает любой продукт такой

компании, не попадает к ней в рабство. Он может продолжить наращивание мощности своей системы путем приобретения продуктов любой другой компании, соблюдающей стандарты. Причем это касается как аппаратных, так и программных средств.

Практической опорой системных и прикладных программных средств открытых систем является стандартизованная операционная система. В настоящее время такой системой является UNIX. Фирмам-поставщикам различных вариантов ОС UNIX в результате длительной работы удалось придти к соглашению об основных стандартах этой операционной системы. Сейчас все распространенные версии UNIX в основном совместимы по части интерфейсов, предоставляемых прикладным (а в большинстве случаев и системным) программистам. Как кажется, несмотря на появление претендующей на стандарт системы Windows NT, именно UNIX останется основой открытых систем в ближайшие годы.

Технологии и стандарты открытых систем обеспечивают реальную и проверенную практикой возможность производства системных и прикладных программных средств со свойствами мобильности (portability) и интероперабельности (interoperability). Свойство *мобильности* означает сравнительную простоту переноса программной системы в широком спектре аппаратно-программных средств, соответствующих стандартам. *Интероперабельность* означает упрощения комплексирования новых программных систем на основе использования готовых компонентов со стандартными интерфейсами.

Преимуществом для пользователей является то, что они могут постепенно заменять компоненты системы на более совершенные, не утрачивая работоспособности системы. В частности, в этом кроется решение проблемы постепенного наращивания вычислительных, информационных и других мощностей компьютерной системы.

Клиенты и серверы локальных сетей

В основе широкого распространения локальных сетей компьютеров лежит известная идея разделения ресурсов. Высокая пропускная способность локальных сетей обеспечивает эффективный доступ из одного узла локальной сети к ресурсам, находящимся в других узлах.

Развитие этой идеи приводит к функциональному выделению компонентов сети: разумно иметь не только доступ к ресурсами удаленного компьютера, но также получать от этого компьютера некоторый сервис, который специфичен для ресурсов данного рода и программные средства. Так мы приходим к различению рабочих станций и серверов локальной сети.

Рабочая станция предназначена для непосредственной работы пользователя или категории пользователей и обладает ресурсами, соответствующими локальным потребностям данного пользователя.

Сервер локальной сети должен обладать ресурсами, соответствующими его функциональному назначению и потребностям сети. Заметим, что в связи с ориентацией на подход открытых систем, правильнее говорить о логических серверах (имея в виду набор ресурсов и программных средств, обеспечивающих услуги над этими ресурсами), которые располагаются не обязательно на разных компьютерах. Особенностью логического сервера в открытой системе является то, что если по соображениям эффективности сервер целесообразно переместить на отдельный компьютер, то это можно проделать без потребности в какой-либо переделке как его самого, так и использующих его прикладных программ.

Примерами сервером могут служить:

•сервер телекоммуникаций, обеспечивающий услуги по связи данной локальной сети с внешним миром;

•вычислительный сервер, дающий возможность производить вычисления, которые невозможно выполнить на рабочих станциях;

•дисковый сервер, обладающий расширенными ресурсами внешней памяти и предоставляющий их в использование рабочим станциями и, возможно, другим серверам;

•файловый сервер, поддерживающий общее хранилище файлов для всех рабочих станций;

•сервер баз данных фактически обычная СУБД, принимающая запросы по локальной сети и возвращающая результаты.

Сервер локальной сети предоставляет ресурсы (услуги) рабочим станциям и/или другим серверам.

Принято называть клиентом локальной сети, запрашивающий услуги у некоторого сервера и сервером - компонент локальной сети, оказывающий услуги некоторым клиентам.

Системная архитектура "клиент-сервер"

Понятно, что в общем случае, чтобы прикладная программа, выполняющаяся на рабочей станции, могла запросить услугу у некоторого сервера, как минимум требуется некоторый интерфейсный программный слой, поддерживающий такого рода взаимодействие (было бы по меньшей мере неестественно требовать, чтобы прикладная программа напрямую пользовалась примитивами транспортного уровня локальной сети). Из этого, собственно, и вытекают основные принципы системной архитектуры "клиент-сервер".

Система разбивается на две части, которые могут выполняться в разных узлах сети, - клиентскую и серверную части. Прикладная программа или конечный пользователь взаимодействуют с клиентской частью системы, которая в простейшем случае обеспечивает просто надсетевой интерфейс. Клиентская часть системы при потребности обращается по сети к серверной части. Заметим, что в развитых системах сетевое обращение к серверной части может и не понадобиться, если система может предугадывать потребности пользователя, и в клиентской части содержатся данные, способные удовлетворить его следующий запрос.

Интерфейс серверной части определен и фиксирован. Поэтому возможно создание новых клиентских частей существующей системы (пример интероперабельности на системном уровне).

Основной проблемой систем, основанных на архитектуре "клиент-сервер", является то, что в соответствии с концепцией открытых систем от них требуется мобильность в как можно более широком классе аппаратно-программных решений открытых систем. Даже если ограничиться UNIX-ориентированными локальными сетями, в разных сетях применяется разная аппаратура и протоколы связи. Попытки создания систем, поддерживающих все возможные протоколы, приводит к их перегрузке сетевыми деталями в ущерб функциональности.

Еще более сложный аспект этой проблемы связан с возможностью использования разных представлений данных в разных узлах неоднородной локальной сети. В разных компьютерах может существовать различная адресация, представление чисел, кодировка символов и т.д. Это особенно существенно для серверов высокого уровня: телекоммуникационных, вычислительных, баз данных.

Общим решением проблемы мобильности систем, основанных на архитектуре "клиент-сервер" является опора на программные пакеты, реализующие протоколы *удаленного вызова процедур* (RPC - RemoteProcedureCall). При использовании таких средств обращение к сервису в удаленном узле выглядит как обычный вызов процедуры. Средства RPC, в которых, естественно, содержится вся информация о специфике аппаратуры локальной сети и сетевых протоколов, переводит вызов в последовательность сетевых взаимодействий. Тем самым, специфика сетевой среды и протоколов скрыта от прикладного программиста.

При вызове удаленной процедуры программы RPC производят преобразование форматов данных клиента в промежуточные машинно-независимые форматы и затем преобразование в форматы данных сервера.

При передаче ответных параметров производятся аналогичные преобразования.

Если система реализована на основе стандартного пакета RPC, она может быть легко перенесена в любую открытую среду.

Особенности обработки данных в различных архитектурах показаны на рис.1.

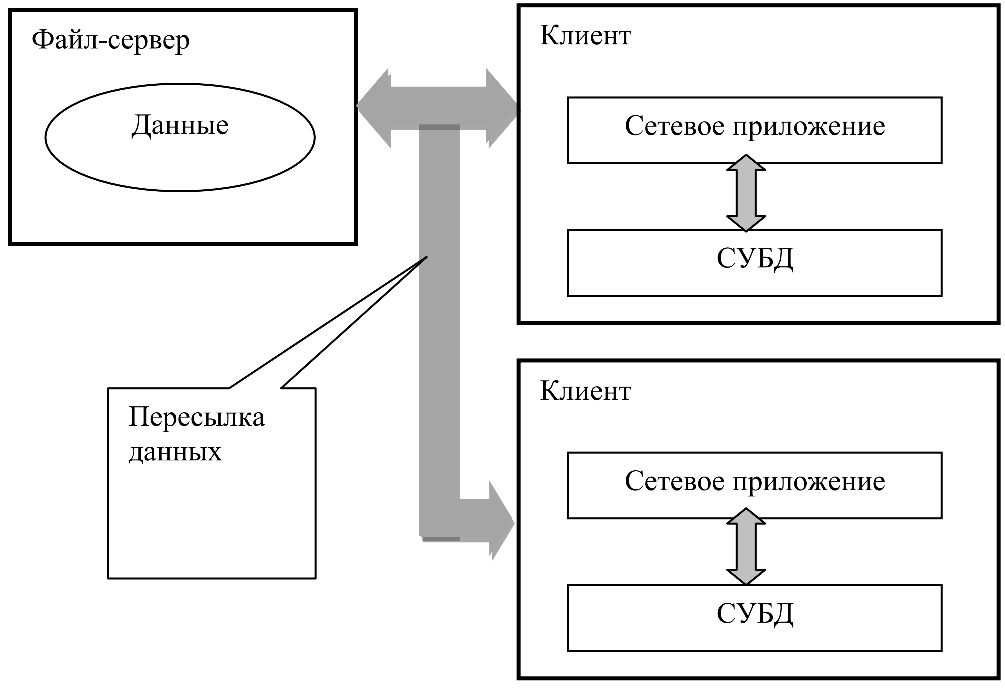
Локальный компьютер

Локальное приложение

СУБД

Данные

Архитектура “файл-сервер”



Архитектура “клиент-сервер”

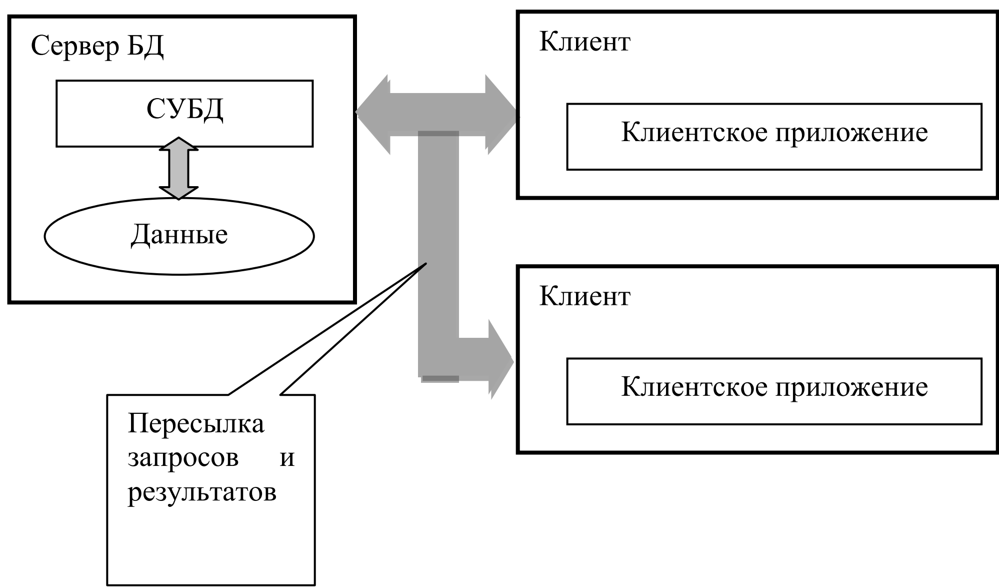


Рис.1. Обработка данных в различных архитектурах

Технология “клиент-сервер” применительно к СУБД сводится к разделению системы на две части – приложение-клиент (front-end) и сервер базы данных (back-end). Эта архитектура совмещает лучшие черты обработки данных на мэйнфреймах и технологии “файл-сервер”. От мэйнфреймов технология “клиент-сервер” позаимствовала такие черты, как централизованное администрирование, безопасность, надежность.

От технологии “файл-сервер” унаследованы низкая стоимость и возможность распределенной обработки данных, используя ресурсы компьютеров-клиентов.

Сейчас графический интерфейс пользователя стал стандартом для систем “клиент-сервер”.

Кроме того, архитектура “клиент-сервер” значительно упрощает и ускоряет разработку приложений за счет того, что правила проверки целостности данных находятся на сервере. Неправильно работающее клиентское приложение не может привести к потере или искажению данных.

Все эти возможности, ранее свойственные только сложным и дорогостоящим системам, сейчас доступны даже небольшим организациям. Стоимость оборудования, программного обеспечения и обслуживания для персональных компьютеров в десятки раз ниже, чем для мэйнфреймов.

Серверы баз данных

Термин "сервер баз данных" обычно используют для обозначения всей СУБД, основанной на архитектуре "клиент-сервер", включая и серверную, и клиентскую части.

Такие системы предназначены для хранения и обеспечения доступа к базам данных.

Хотя обычно одна база данных целиком хранится в одном узле сети и поддерживается одним сервером, серверы баз данных представляют собой простое и дешевое приближение к распределенным базам данных, поскольку общая база данных доступна для всех пользователей локальной сети.

Принципы взаимодействия между клиентскими и серверными частями

Доступ к базе данных от прикладной программы или пользователя производится путем обращения к клиентской части системы. В качестве основного интерфейса между клиентской и серверной частями выступает язык баз данных SQL.

Это язык по сути дела представляет собой текущий стандарт интерфейса СУБД в открытых системах. Собирательное название SQL-сервер относится ко всем серверам баз данных, основанных на SQL.

Серверы баз данных, интерфейс которых основан исключительно на языке SQL, обладают своими преимуществами и своими недостатками. Очевидное преимущество – стандартность интерфейса. В пределе, хотя пока это не совсем так, клиентские части любой SQL-ориентированной СУБД могли бы работать с любым SQL-сервером вне зависимости от того, кто его произвел.

Недостаток тоже довольно очевиден. При таком высоком уровне интерфейса между клиентской и серверной частями системы на стороне клиента работает слишком мало программ СУБД. Это нормально, если на стороне клиента используется маломощная рабочая станция. Но если клиентский компьютер обладает достаточной мощностью, то часто возникает желание возложить на него больше функций управления базами данных, разгрузив сервер, который является узким местом всей системы.

Одним из перспективных направлений СУБД является гибкое конфигурирование системы, при котором распределение функций между клиентской и пользовательской частями СУБД определяется при установке системы.

Преимущества протоколов удаленного вызова процедур

Упоминавшиеся выше протоколы удаленного вызова процедур особенно важны в системах управления базами данных, основанных на архитектуре "клиент-сервер".

Во-первых, использование механизма удаленных процедур позволяет действительно перераспределять функции между клиентской и серверной частями системы, поскольку в тексте программы удаленный вызов процедуры ничем не отличается от удаленного вызова, и следовательно, теоретически любой компонент системы может располагаться и на стороне сервера, и на стороне клиента.

Во-вторых, механизм удаленного вызова скрывает различия между взаимодействующими компьютерами. Физически неоднородная локальная сеть компьютеров приводится к логически однородной сети взаимодействующих программных компонентов. В результате пользователи не обязаны серьезно заботиться о разовой закупке совместимых серверов и рабочих станций.

Типичное разделение функций между клиентами и серверами

В типичном на сегодняшний день случае на стороне клиента СУБД работает только такое программное обеспечение, которое не имеет

непосредственного доступа к базам данных, а обращается для этого к серверу с использованием языка SQL.

В некоторых случаях хотелось бы включить в состав клиентской части системы некоторые функции для работы с "локальным кэшем" базы данных, т.е. с той ее частью, которая интенсивно используется клиентской прикладной программой. В современной технологии это можно сделать только путем формального создания на стороне клиента локальной копии сервера базы данных и рассмотрения всей системы как набора взаимодействующих серверов.

С другой стороны, иногда хотелось бы перенести большую часть прикладной системы на сторону сервера, если разница в мощности клиентских рабочих станций и сервера чересчур велика. В общем-то при использовании RPC это сделать нетрудно. Но требуется, чтобы базовое программное обеспечение сервера действительно позволяло это. В частности, при использовании ОС UNIX проблемы практически не возникают.

Архитектуры процессора базы данных

Основная часть любой системы “клиент-сервер” – это сервер БД. Со времени возникновения архитектуры “клиент-сервер” появилось много вариантов архитектуры процессора БД, поскольку он во многом определяет успех всей системы. Основное требование к серверу БД – обеспечение минимального времени выполнения запросов при максимально возможном числе пользователей. Существуют две основные архитектуры для построения процессора БД: архитектура с несколькими процессами и многопоточная архитектура.

1.*Архитектура с несколькими процессами*

Характеризуется тем, что несколько экземпляров исполняемого файла работают одновременно. Эти системы отличаются хорошей масштабируемостью, но требуют значительных расходов памяти, так как память каждому экземпляру приложения выделяется отдельно. Эта архитектура подразумевает наличие эффективного механизма

взаимодействия процессов и полагается на операционную систему при разделении процессорного времени между отдельными экземплярами приложения. Самый известный пример сервера, построенного по этой архитектуре, - OracleServer. Когда пользователь подключается к БД Oracle, он в действительности запускает отдельный экземпляр исполняемого файла процессора базы данных.

2. *Многопоточная архитектура*

Эта архитектура использует только один исполняемый файл, с несколькими потоками исполнения. Главное преимущество – более скромные требования к оборудованию, чем для архитектуры с несколькими процессами. Здесь сервер берет на себя разделение времени между отдельными потоками, иногда давая преимущество некоторым задачам над другими. Кроме того, отпадает необходимость в сложном механизме взаимодействия процессов. По этой архитектуре построены MS SQL Server и Sybase SQL Server.

Трехуровневая архитектура “клиент-сервер”

На верхнем уровне абстрагирования взаимодействия клиента и сервера достаточно четко можно выделить следующие компоненты:

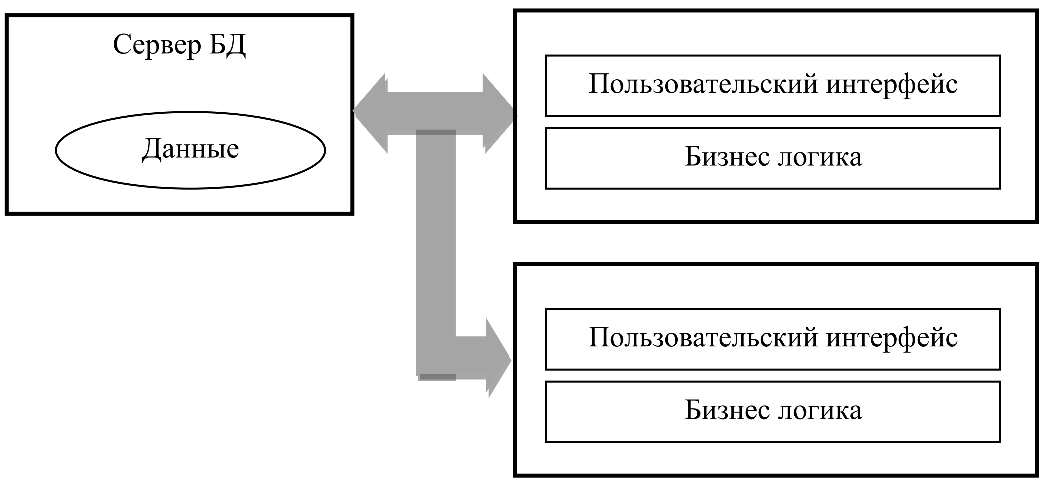
•презентационная логика (PresentationLayer - **PL**), предназначенная для работы с данными пользователя;

•бизнес-логика (BusinessLayer - **BL**), предназначенная для проверки правильности данных, поддержки ссылочной целостности;

•логика доступа к ресурсам (AccessLayer - **AL**), предназначенная для хранения данных;

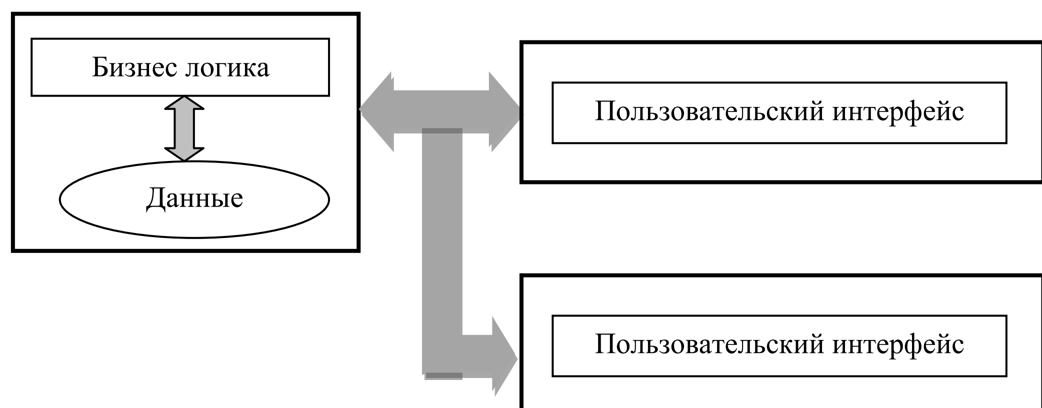
Таким образом можно, можно придти к нескольким моделям клиент-серверного взаимодействия:

1. *"Толстый" клиент.* (fatclient)



Наиболее часто встречающийся вариант реализации архитектуры клиент-сервер в уже внедренных и активно используемых системах. Такая модель подразумевает объединение в клиентском приложении как PL, так и BL, таким образом обеспечивается полная децентрализация управления бизнес-логикой. Однако в случае необходимости выполнения каких-либо изменений в клиентском приложении придется менять исходный код. Серверная часть, при описанном подходе, представляет собой сервер баз данных, реализующий AL. К описанной модели часто применяют аббревиатуру RDA - RemoteDataAccess.

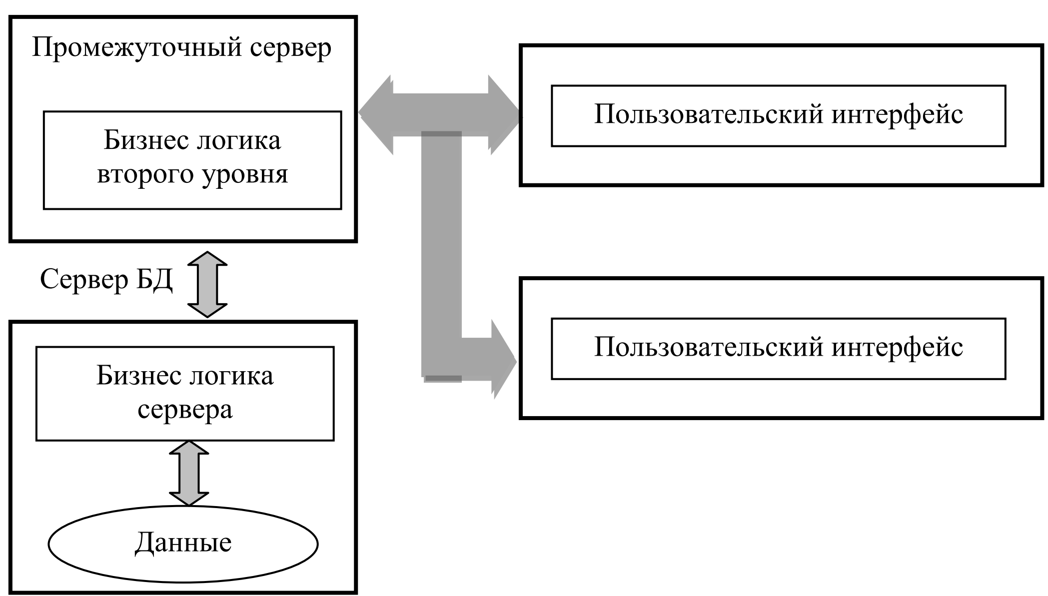
2. *"Тонкий" клиент.* (thinclient)



Модель, начинающая активно использоваться в корпоративной среде в связи с распространением Internet-технологий и, в первую очередь, Web-браузеров. В этом случае клиентское приложение обеспечивает реализацию PL, поэтому клиент может довольствоваться довольно скромной аппаратной платформой, а сервер объединяет BL и AL. Максимальная загрузка сервера

предусматривает выполнение бизнес-логики только с помощью хранимых процедур сервера (*Хранимые процедуры* – откомпилированные SQL-инструкции, хранящиеся на сервере). Это позволяет максимально централизовать контроль над данными и легко изменять правила работы сразу для целого предприятия. С другой стороны, незначительная корректировка правил, касающаяся только части пользователей, потребует длительной процедуры согласования. В этом случае невозможно реализовать какие-то исключения из общих правил для некоторых пользователей или приложений. В принципе, это хорошо и является залогом безопасности и целостности данных.

3. *Сервер бизнес-логики. (трехуровневая архитектура)*



Модель с физически выделенным в отдельное приложение блоком BL, таким образом получаем трехуровневую архитектуру “клиент-сервер”. На сервере БД может функционировать “универсальная” часть бизнес-логики (правила на уровне предприятия или группы связанных приложений). Такая схема позволяет поддерживать тонких клиентов на пользовательских компьютерах и в то же время разгрузить сервер БД от чрезмерной загрузки

при сохранении гибкой системы работы с бизнес-правилами. В качестве промежуточного сервера может использоваться второй SQL-сервер, но чаще рациональней задействовать персональную СУБД, которая менее требовательна к аппаратным ресурсам и может обеспечить удобные средства построения и поддержки бизнес-логики.

Программные средства разработки

Универсальные средства

Для разработки клиентских приложений существует громадное число универсальных пакетов программ, которые позволяют выполнить соединение с сервером и разработать для пользователя удобный графический интерфейс, позволяющий эффективно работать с данными. Некоторые из этих средств для разработки приложений в архитектуре “клиент-сервер” перечислены в таблице.

Наименование	Краткая характеристика
CA-OpenROAD	Полнофункциональная объектно-ориентированная среда для разработки приложений на основе языка четвертого поколения 4GL.
Delphi Client/Server	Универсальный пакет для разработки клиентских приложений. Обеспечивает объектно-ориентированную разработку с использованием визуальных средств. Поддерживает групповую работу над приложением.
Magic 6.0	Таблично-управляемый инструментарий для разработки трехуровневых приложений “клиент-сервер”.

MS VisualBasic 5.0	Универсальный пакет разработки пользовательских приложений. Обеспечивает визуальное построение форм и компиляцию приложения. В полном объеме поддерживаются OLE 2.0 и OLE Automation. Для работы с данными предназначен визуальный инструментарий VisualDatabaseTools.
PowerBuilder 4.0	Объектно-ориентированное средство разработки приложений “клиент-сервер”. Имеет мощные визуальные средства; поддерживает стандарты OLE и ODBC.
Progress 8	Пакет поддерживает компонентную объектно-ориентированную разработку приложений. Используется новая технология SmartObject и среда компонентов приложения (ACE).
SAS System	Обеспечивает инструментарий для доступа, управления, анализа и представления данных в приложении для громадного числа систем и компьютерных платформ, включая мэйнфреймы. Имеет 35 видов интерфейса для различных систем и язык программирования четвертого поколения. Поддерживает ODBC.
UnifaceSix	Независимая среда разработки. Поддерживает управление на уровне модели и компонентное программирование. Имеет мощные визуальные средства. Допускает групповую разработку. Имеет интерфейс к более чем 30 серверам БД на различных платформах.

Персональные СУБД

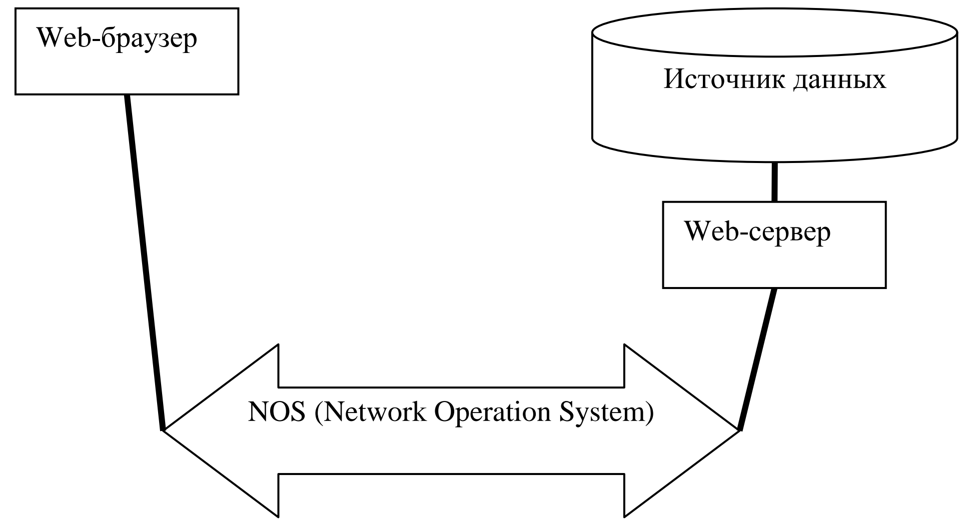
Для разработки клиентских приложений в большинстве случаев вместо универсальных средств разработки удобнее использовать персональные СУБД. Использование персональных СУБД позволяет не только эффективно организовывать работу с бизнес-правилами, но и поддержать независимую работу клиентского приложения за счет наличия собственных форматов хранения данных. Краткая характеристика некоторых персональных СУБД приведена в таблице.

Наименование	**Краткая характеристика**
LotusApproach 97	Позволяет выполнять все виды обработки данных. Имеет очень простой интерфейс. СУБД тесно интегрирована с базами данных Notes и электронными таблицами Lotus 1-2-3. Поддерживает технологию электронного обмена сообщениями MAPI.
MS Access 97	Полнофункциональная СУБД, обладающая богатым набором визуальных средств, многочисленными мастерами и мощным языком программирования VisualBasicforApplications. Имеет гибкую систему подготовки отчетов. Поддерживаются технологии ODBC и OLE 2.0. СУБД тесно интегрирована со всеми приложениями MS Office.

MS VisualFoxPro 5	Одна из наиболее быстрых персональных СУБД, сочетающая технологию xBase и объектно-ориентированный язык программирования. Имеет богатый набор визуальных средств разработки и мастеров для быстрого построения приложений и отчетов. Поддерживаются технологии ActiveX, ODBC и OLE 2.0. Позволяет создавать OLE-сервера и имеет очень развитые средства разработки и поддержки приложений “клиент-сервер”.
Paradox 7	Поддерживает все виды работы с данными. Для визуального выполнения стандартных задач имеется специальное средство Experts. Наделен собственным достаточно сложным языком ObjectPAL. Поддерживает технологии OLE 2.0, ActiveX, MAPI и ODBC.

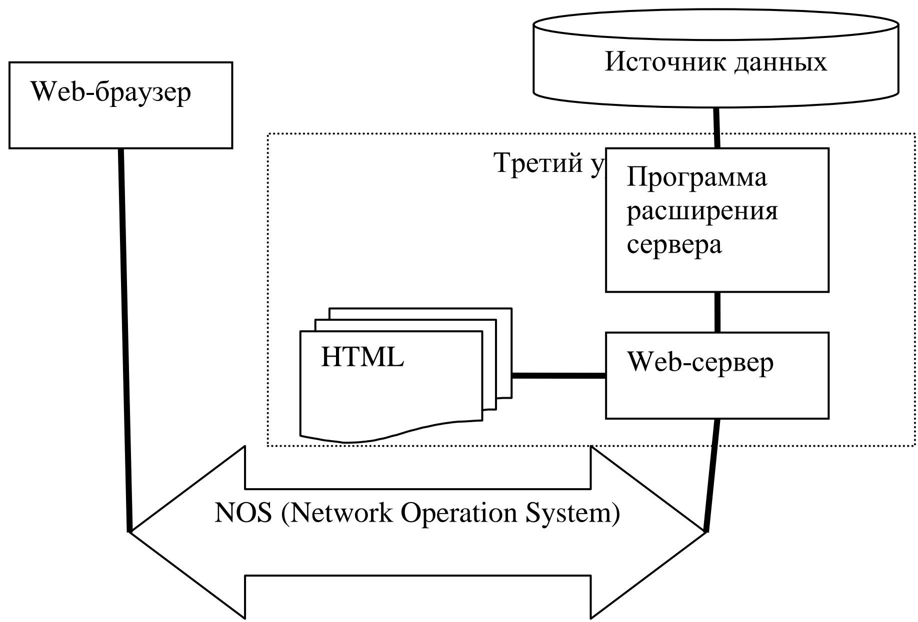
Intranet и архитектура “клиент-сервер”

Двухуровневая архитектура “клиент-сервер”



Разграничение функций между Web-браузером и Web-сервером является очень четким. Web-сервер предоставляет HTML-страницы, а браузер отображает эти страницы путем интерпретации тегов HTML.

Трехуровневая архитектура “клиент-сервер”



Клиентский уровень занимает браузер, на уровне сервера находится сервер БД, а на промежуточном уровне располагаются Web-сервер и программа расширения сервера. Такое архитектурное решение позволяет уменьшить сетевой трафик, делает компоненты взаимозаменяемыми и повышает уровень безопасности. Однако такая архитектура также затрудняет обработку транзакций БД ввиду природы протокола HTTP, не запоминающего состояния (этот протокол использует для передачи данных между браузером и сервером БД).

Браузер посылает Web-серверу запросы на доставку Web-страниц или данных. Web-сервер обслуживает заявки на Web-страницы, а запросы отправляет программе-расширению серверной части. Последняя принимает передаваемые ей запросы, преобразует их в форму, понятную серверу БД, и передает их серверу БД.

Затем сервер БД выполняет работу по обслуживанию запроса и возвращает результат программе-расширению серверной части. Наконец та преобразует результаты в формат, приемлемый для браузера, и передает их Web-серверу, а тот в свою очередь – браузеру.

Программы расширения серверной части

Одной из главных причин использования программ-расширений серверной части на промежуточном уровне является возможность использовать стандарты, существующих для двух крайних уровней, путем осуществления трансляции между ними. Другие применения расширений серверной части состоят в поддержании соединений между БД с целью уменьшить трафик в сети и в поддержании резерва соединений между БД для уменьшения затрат ресурсов на открытие/закрытие БД. Расширения серверной части также поддерживают взаимозаменяемость в своих стандартных интерфейсах. Поэтому Web-серверы и серверы БД можно сравнительно легко заменять или наращивать.

Заключение

Использование технологии «клиент-сервер» позволило создавать надежные (в смысле целостности данных) многопользовательские ИС с централизованной базой данных, независимые от аппаратной (а часто и программной) части сервера БД и поддерживающие графический интерфейс пользователя (ГИП) на клиентских станциях, связанных локальной сетью. Причем издержки на разработку приложений существенно сокращались. Не удивительно, поэтому, что эта технология завоевала большую популярность среди разработчиков прикладного ПО, а приложения на ее основе - широкое распространение на рынке ИС.

Несмотря на объективные сложности и предубеждения пользователей, ситуация меняется - все большее количество фирм-разработчиков пытается не взирая на трудности перейти на использование этой прогрессивной технологии. Появляются инструментальные средства и технологии создания ИС, в том числе и в трехслойной архитектуре.

РАЗДЕЛ 2. ОСНОВЫ HTML

Глава 5. Структура документа. Основные теги HTML

Основные понятия

HTML*(HyperTextMarkupLanguage)* - язык разметки гипертекста - предназначен для создания Web-страниц.

Под ***гипертекстом*** в этом случае понимается текст, связанный с другими текстами указателями-ссылками.

HTML представляет собой достаточно простой набор кодов, которые описывают структуру документа. HTML позволяет выделить в тексте отдельные логические части (заголовки, абзацы, списки и т.д.), поместить на Web-страницу подготовленную фотографию или картинку, организовать на странице ссылки для связи с другими документами.

HTML не задает конкретные и точные атрибуты форматирования документа. Конкретный вид документа окончательно определяет только ***программа-браузер*** на компьютере пользователя Интернета.

HTML также не является языком программирования, но web-страницы могут включать в себя встроенные программы-скрипты на языках ***Javascript*** и ***VisualBasicScript*** и программы-апплеты на языке ***Java***.

Даже, если вы не собираетесь в дальнейшем редактировать "вручную" текст HTML (предполагая использовать графические редакторы), знание языка HTML даст вам возможность как лучше использовать эти средства, так и увеличит ваши шансы сделать HTML-документ более доступным и "читаемым" при просмотре браузерами разных фирм.

Основными компонентами HTML являются:

- **Тег (tag).** Тег HTML это компонент, который командует Web- браузеру выполнить определенную задачу типа создания абзаца или вставки изображения.
- **Атрибут** (или **аргумент**). Атрибут HTML изменяет тег. Например, можно выровнять абзац или изображение внутри тега.

- **Значение**. Значения присваиваются атрибутам и определяют вносимые изменения. Например, если для тега используется атрибут выравнивания, то можно указать значение этого атрибута. Значения могут быть текстовыми, типа *left* или *right*, а также числовыми, как например ширина и высота изображения, где значения определяют размер изображения в пикселях.

Теги представляют собой зарезервированные последовательности символов, начинающиеся с < (знака меньше) и заканчивающиеся > (знаком больше).

Закрытие тега отличается от открытия только наличием символа **'/'**.

Предположим, у нас есть гипотетический атрибут форматирования текста, управляемый кодом **<X>**, и мы хотим применить его к словам "Это мой текст".

HTML-последовательность кодов и собственно текста будет выглядеть так:

<X>Это мой текст</X>

Теги могут вкладываться друг в друга иерархически, но без пересечений, то есть допустимо вложение вида ***<teg1>*****<teg2></teg2>*****</teg1>***, но не ***<teg1>*****<teg2>*****</teg1>*****</teg2>**.

Действие вложенных таговобъединяется.Например, если внутрь тега, создающего жирное начертание шрифта, вложен тег курсива, то в результате получится жирный курсив.

Первое правило

Первое правило HTML: закрывайте все, что вы открыли!

НО! Из этого правила, как и из всех остальных, существуют исключения.

HTML- программа должна начинаться тегом **<HTML>**и заканчиваться тегом **</HTML>**

```
<HTML>
   ...... (здесь будут другие теги программы)
</HTML>
```

HTML- программы состоят из двух основных частей: заголовка и тела. Заголовок ограничивается парой тегов**<HEAD>** и **</HEAD>**, а тело - парой тегов**<BODY>** и**</BODY>**.

В результате HTML- программа выглядит следующим образом:

```
<HTML>
<HEAD>
    ... (здесь будет заголовок)
</HEAD>
<BODY>
   .... (здесь будут другие теги тела программы)
</BODY>
</HTML>
```

Кроме того, каждая HTML- программа имеет заголовок, который помещается в заголовок окна браузера. Заголовок окна браузера создается при помощи двух тегов **<TITLE>** и **</TITLE>** и содержится между тегами **<HEAD>** и**</HEAD>**.

Тогда программа принимает следующий вид:

```
<HTML>
<HEAD>
<TITLE>Основы HTML </TITLE> ;
</HEAD>
<BODY>
```

.... (здесь будут другие теги тела программы)

</BODY>

</HTML>

Некоторые авторы, пишущие об языке HTML, советуют записывать теги прописными буквами, другие - используют строчные. Редактор HTML - AllaireHomeSite 4.5.1, например использует по умолчанию нижний регистр для записи тегов. При создании моих страниц использовались оба варианта написания тегов. Как видите, допустимо и то и другое. Современные браузеры допускают запись тегов в любом регистре.

Существуют теги и атрибуты "чуствительные" к написанию прописными или строчными буквами. Это регламентируется стандартами языка HTML, определенными Консорциумом W3C.

При написании HTML-программ возникает необходимость вставки ***комментариев*** - поясняющих текстов, которые невидны при загрузки документа в браузер. Для этой цели служит тег <**!**>. Все, что заключено между символами <**!** и > считается комментарием и не отображается в браузере.

Еще один тег, который очень важен в HTML-программе, но так же не предназначается для отображения какого-либо объекта в браузере - тег <***META***>. Этот тег служит специальным целям, а именно - указания языка, на котором написан документ, его кодовой страницы, ключевых слов, используемых поисковыми системами для классификации этого документа и т.п. Теги <***META***> обычно вставляются в HTML-программу на заключительном этапе создания Web-страницы -*публикации*.

Для вставки в HTML-программу фрагмента программ, написанных на языке JavaScript или ViualBasicScript *сценариев* используют теги ***<SCRIPT>*** и ***</SCRIPT>***.

Общая структура HTML-файла

Суммируя вышесказанное приведем общую структуру HTML-файл:

<HTML>

<HEAD>

<Мета-теги>

<Функции скриптов>

<TITLE>Заголовок документа</TITLE>

</HEAD>

<BODY>

Основная часть документа

</BODY>

</HTML>

Заголовки, абзацы, разрывы строк

Заголовки

Каждый пользователь компьютера, работающий в текстовом редактором MicrosoftWord знаком с понятием ***стиля заголовка***. В HTML тоже применяется это понятие для структурирования документа и выделения важности заголовка. Всего существуют 6 стилей заголовка. Каждый из них обозначается в HTML-документе парными тегами ***<Hi>*** и ***</Hi>*** Здесь ***i*** обозначает важность стиля. ***H1*** обозначает самый важный стиль заголовка, ***H2*** - стиль заголовка второго уровня, а ***H6*** - стиль заголовка самого нижнего уровня.

В подавляющем большинстве случаев для заголовков Web-страниц используют три первых уровня заголовков ***<H1>***, ***<H2>*** и ***<H3>***. Объясняется

это тем, что размеры шрифтов оставшихся заголовуов (теги ***<H4>*** - ***<H5>***) меньше размера обычного шрифта Web-страницы.

Вот как в документ можно добавить очень важный заголовок.

<H1>An important heading</H1>

Абзацы

Понятие абзаца в HTML-документе также аналогично понятию абзаца в Microsoft Word. Абзац обозначается в документе парными тегами ***<P>*** и ***</P>*** . Впрочем, применение закрывающего тега не является строго обязательным.

НО! Специфика тега ***<P>*** заключается в том, что после текста, который находится в его пределах, пустая строка добавляется ***автоматически***.

Следует помнить и о другой особенности текстовых абзацев: когда текст достигает правой границы окна Web-браузера. переход на новую строку осуществляется автоматически, независимо от расположения тега ***<P>***.

Для отдельного абзаца можно указать тип, размер и цвет шрифта отличным от стиля остального документа.

Например:

<P ALIGN="CENTER"><B><I><FONT FACE="Arial" SIZE=6 COLOR="#C07080">My greetings to you!</P>

Разрывы строк

Если в середине строки появилась необходимость ее разорвать - используйте **одиночный** тег переноса строки ***
***.(Это соответствует нажатию клавишной комбинации [Shift]-[Enter] в текстовых процессорах Word). Код
 не означает конца логического абзаца, и за строкой с этим кодом дополнительная пустая строка не появится.

Вот и нарушено первое правило!

Кроме тега <***BR***> не требует закрытия тег добавления изображения <***IMG***>.

Использование закрывающего тега абзаца <***/p***> также не является строго обязательным.

HTML довольно "демократичен": неправильный тег или неправильное вложение тегов обычно не приводят к "зависаниям" браузера, а только вызывает сообщение об ошибке в строке состояния окна браузера вашего Интернет-читателя. Разумеется ошибки могут привести к неправильному форматированию HTML-документа.

Примером использования тега <***BR***> может служить написание почтового адреса или стихотворения.

Браузеры показывают текст в своем окне, автоматически осуществляя перенос слов. Поэтому, если вы считаете необходимым запретить разрыв блока текста с пробелами между словами - воспользуйтесь специализированным символом *** *** - символом незазрывного пространства (non-breakingspace). Например,

Освежающий и бодрящий напиток Coca Cola,

приобрел широкую популярность в нашей стране.

А вот и результат:

Освежающий и бодрящий напиток Coca Cola, приобрел широкую популярность в нашей стране.

Совет

Не следует использовать цепочку символов *** *** для выравнивания текста в окне браузера. Для этой целей рекомендуется использовать *таблицы стилей*.

Предварительно отформатированный текст

Достоинством браузеров является их способность самостоятельно распределять текст в окне браузера. Но иногда вы не нуждаетесь в этой услуге, хотите самостоятельно определить представление вашего текста в окне браузера. Например, вы хотите представить код программы в наиболее удобочитаемом виде. Такую возможность вам предоставляет тег ***<pre>***. Посмотрим пример:

```
<pre>
voidNode::Remove()
{
if (prev)
prev->next = next;
else if (parent)
parent->SetContent(null);
if (next)
next->prev = prev;
parent = null;
  }
</pre>
```

Обратите внимание на то, что в случае использования тега **<pre>**, текст отображается браузером точно в таком виде, как он был создан в HTML-документе. Сохраняются все пробелы, табуляции и переводы строк. Исключением является только новая строка, следующая сразу же за тегом **<pre>**. Таким образом, эти два примера кода HTML на экране дисплея будут показаны одинаково:

```
<pre>предварительно отформатированный текст</pre>
<pre>
предварительно отформатированный текст
</pre>
```

А именно:

```
предварительно отформатированный текст
```

предварительно отформатированный текст

У этого тега существует необязательный атрибут, указывающий желаемый размер строки в символах, а именно:

<PRE width="N">

Попробуем?

Предмет истории есть жизнь народов и человечества.

Непосредственно уловить и обнять словом - описать жизнь не только

человечества, но одного народа, представляется невозможным.

Обычно, для отображения предварительно отформатированного текса используются моноширинные шрифты, все символы которого имеют одинаковую ширину. Для того, чтобы браузер "не забыл" об этом следует применить *таблицы стилей*. Например, это можно сделать так:

```
<style type="text/css">
pre { color: green; background: white; font-family: monospace; }
</style>
```

Результат такого определения стиля для тега <pre> вы можете видеть на этой странице.

Совет

Если вы устанавливаете цвет текста, желательно также установить цвет фона. Это поможет вам избежать ситуации, когда буквы будут практически неразличимы на близком к ним по цвету фоне.

Совет № 2. Вместо того, чтобы устанавливать цвет фона для элемента <pre>, установите цвет фона для элемента <body>. Например, это можно сделать так:

```
<style type="text/css">
body { color: black; background: white; }
pre { color: green; font-family: monospace; }
```

</style>

Выделенный текст

В тех случаях, когда необходимо обратить особое внимание пользователя на тот или иной фрагмент текста, следует использовать стандартные средства форматирования.

Стандартные средства форматирования представлены специальными тегами, которые ***обязательно*** являются ***парными*** (т.е. имеются открывающий и закрывающий теги).

Чтобы увеличить размер шрифта на один пункт, используйте тег <big>.

Чтобы выделить текст **полужирным шрифтом**, воспользуйтесь **тегом <b>** или **тегом <strong>**.

Чтобы выделить текст *курсивом*, воспользуйтесь *тегом <i>* или ***тегом <em>***.

Вы заметили, что в данном случае выделение текста тегами <i> и <em>различаются?

Объяснение простое - в HTML-коде этой страницы используется внедренная таблица стилей, в которой для тега <em> указан шрифт: полужирный курсив.

Шрифт пишущей машинки можно имитировать с помощью тега <tt>.

С помощью тега <small> размер шрифта можно уменьшить на один пункт.

Существуют и другие теги, которые также предназначены для выделения текста.

Кроме того, в HTML включена поддержка математических символов и научных обозначений. Для построения простейших равенств и выражений вам могут пригодиться два тега ***<sub>*** (нижний индекс) и ***<sup>*** (верхний индекс). Например:

$$A^2+B^2=C^2$$

CO_2=углекислый газ

В HTML-коде это записано так:

```
<p align="center">A<sup>2</sup>+B<sup>2</sup>=C<sup>2</sup></p>
<p align="center">CO<sub>2</sub>=углекислыйгаз</p>
```

Совет

Если вы намерены придерживаться хорошего стиля программирования Web-страниц (с помощью HTML и XHTML), то вместо тегов форматирования используйте каскадные таблицы стилей.

На сегодняшний день многие старые теги HTML, которые предназначались специально для выделения текста, вытеснены новыми возможностями форматирования с помощью каскадных таблиц стилей.

Глава 6. Оформление текстов. Гиперссылки. Списки. Таблицы. Фреймы.

Основные средства форматирования текста

Как и в Word, основой структуры текста в HTML является абзац. Для выделения абзаца служит тэг<P>, закрывать который не обязательно. В тэге<P> можно указать способ выравнивания текста абзаца в виде <P ALIGN="опция_выравнивания">. Для выравнивания текста абзаца по левому краю, правому краю, по центру или по ширине следует использовать, соответственно, опции LEFT, RIGHT, CENTER, JUSTIFY (последний - только в новых браузерах). Точно так же можно выравнивать и заголовки в тэгах<H1>,...,<H6> (см. табл. 2).

При "ручном" наборе текста для HTML-документа следует **избегать** переносов слов по слогам и выравнивания текста с помощью пробелов или табуляций. Все форматирование в HTML осуществляется через тэги, а переносы строк и количество символов в строке не имеют значения.

Для принудительного разрыва строки внутри абзаца служит тэг
. Если Вы хотите, напротив, запретить переносы в какой-то части текста, следует заключить ее в тэг<NOBR>...</NOBR>.

Некоторые символы являются для HTML служебными и при "ручном" наборе HTML-документа вместо них следует использовать сочетания символов из таблицы 1:

Символ	**Обозначение в HTML**
<	<
>	>
&	&
"	"

Таблица 1 Служебные символы HTML

Если нужно вставить в текст один или несколько неизменяемых пробелов, для этого используется сочетание символов

Для форматирования текста абзацев используйте тэги, приведенные в таблице 2:

Тэг	**Пояснение**	**Образец**
<H1>...</H1>	Заголовок 1 уровня	*Заголовок1*
<H2>...</H2>	Заголовок 2 уровня	*Заголовок2*
<H3>...</H3>	Заголовок 3 уровня	*Заголовок3*
<H4>...</H4>	Заголовок 4 уровня	*Заголовок4*
<H5>...</H5>	Заголовок 5 уровня	Заголовок5
<H6>...</H6>	Заголовок 6 уровня	Заголовок6
<BIG>...</BIG>	Большой	Большой
<SMALL>...</SMALL>	Маленький	Маленький
^{...}	Верхний индекс	Верхний индекс
_{...}	Нижний индекс	Нижний индекс
<B>...</B>	Жирный	**Жирный**

<I>...</I>	Курсив	*Курсив*
<U>...</U>	Подчеркнутый	Подчеркнутый
<S>...</S>	Перечеркнутый	~~Перечеркнутый~~

Таблица 2 Некоторые тэги форматирования текста

Кроме указанных выше элементов *физического* форматирования, существуют тэги *логического* форматирования, служащие, например, для выделения цитат, важных фрагментов текста, имен переменных и т.д. Информацию о них можно найти в справочной литературе.

Размерами и начертаниями шрифта можно управлять также с помощью тэга

<FONT FACE="Шрифт" SIZE="Размер">

где "Шрифт" - имя шрифта Windows, а размер указывается цифрой от 1 до 7, например, этот текст отформатирован тэгом<FONT FACE="Arial" SIZE="5">. Размер шрифта по умолчанию обычно равен 3. Разумеется, для корректного отображения текста каким-либо шрифтом он должен быть установлен на машине пользователя, поэтому следует избегать "экзотических" шрифтов. По умолчанию браузеры поддерживают один стандартный масштабируемый шрифт (обычно это TimesNewRoman) и один шрифт для отображения предварительно отформатированного текста в тэге<PRE> (обычно CourierNew).

Опция face позволяет также указать несколько шрифтов через запятую - в этом случае для форматирования текста будет использован первый подходящий шрифт из списка:

<font face="Verdana, Arial, Helvetica">

Для отображения **предварительно отформатированного текста**, с сохранением переносов строк, табуляций и интервалов, используется тэг<PRE width="N">, где N - необязательная опция, указывающая желаемый размер строки в символах. Если нужна гарантия, что все символы отформатированного текста будут иметь одинаковую ширину, следует использовать тэг<TT>. Примеры:

Текст в тэге TT

Текст

втэге

PRE

Для создания в документе текстового **раздела** используется тэг<div>...</div>, с опцией align, указывающей способ выравнивания текста в разделе (так же, как в тэге<p>). Указанный в тэге<div> способ выравнивания текста используется по умолчанию во всех абзацах этого раздела, если в абзаце в явном виде не указан другой способ выравнивания. Для размещения части документа по центру окна браузера можно также использовать тэг<center>...</center>

Отделить часть текста горизонтальной чертой можно с помощью тэга<HR>. Опция ALIGN со значением LEFT, RIGHT или CENTER определяет выравнивание черты на странице, опция SIZE="размер" - толщину линии в пикселах, а опция WIDTH="ширина" - ширину, указанную в пикселах (например, 600) или процентах (100%). Если указана опция NOSHADE, линия создается без трехмерных эффектов. Пример:

Эта черта создана тэгом вида <HR ALIGN="CENTER" SIZE="1" WIDTH="50%">

Гиперссылки

Гиперссылками (или просто ссылками) называют выделенные области документа, позволяющие перейти к другой его части или к другому документу в Сети.

Гиперссылка состоит из "внутренней" части, то есть, адреса документа, на который она ссылается, и "внешней" части, видимой на экране и называемой *якорем* гиперссылки. Якорь гиперссылки может представлять из себя слово или группу слов, картинку или часть картинки. Если подвести указатель “мыши” к ссылке, он примет форму ладони с вытянутым указательным пальцем – и это самый надежный способ ее определить. При

этом, в строке состояния браузера можно будет прочитать адрес, на который указывает ссылка.

Кроме того, практически всегда текстовые ссылки выделены другим цветом и очень часто подчеркнуты. Графические ссылки часто заключены в рамку того же цвета, которым выделяются текстовые ссылки. По умолчанию это синий цвет, но каждая Web-страница может использовать свое собственное оформление. Цвет ссылок, которые уже выбирались пользователем, обычно отличается от цвета не посещённых ссылок и выглядит более “бледным” (по умолчанию – фиолетовый).

Для **создания** гиперссылки служит тэг вида

<A HREF="адрес" TARGET="окно">текст ссылки</A>

Об адресах документов сказано ниже. Опция TARGET может быть не указана, в этом случае документ открывается в текущем окне браузера, или указана в виде TARGET="_BLANK">, тогда документ открывается в новом окне. Текст ссылки внутри тэга<A> может быть любым. Точно так же в тэг<A> можно поместить и изображение.

Чтобы научиться создавать ссылки, нужно иметь представление о том, как адресуются документы в Сети.

Полный адрес документа в сети называется **URL** (UniformResourceLocator; принято читать “урл”). URL может состоять из следующих частей:

- *префикс протокола*, то есть, указание на используемую сетевую службу. Основные протоколы, с которыми Вы столкнетесь, приведены в таблице 3.

Протокол	Пояснение
http://	HTTP – основной протокол, обеспечивающий доступ к Web-страницам. Используется по умолчанию, поэтому часто не указывается в URL.
ftp://	Протокол передачи файлов FTP, позволяющий при помощи программы FTP-клиента обмениваться файлами

	с удаленным компьютером.
mailto:	Доступ к электронной почте. Вслед за mailto: указывается адрес электронной почты, имеющий общий вид имя_пользователя@адрес_почтового_домена.
file://	Указывается вместо имени протокола при обращении к Web-странице, расположенной на локальной машине.

Таблица 3 Основные префиксы протоколов в URL

- *доменное имя компьютера* или его IP-адрес вместо доменного имени. Например, сервер НГАСУ некогда имел доменное имя www.ngasu.nsk.su и IP-адрес 62.76.97.33;
- *номер порта*, через который происходит взаимодействие с сервером. Перед номером порта ставится двоеточие. С точки зрения пользователя указание порта бывает полезно, например, для “принудительной” перекодировки документа. Так, адреса http://www.newmail.ru:8100 и http://www.newmail.ru:8101 адресуют один и тот же сервер, но в первом случае документ читается в кодировке KOI-8, а во втором – в кодировке Windows. Вообще же, номер порта включается в URL только при нестандартных настройках сервера;
- *имя файла* на сервере, которое может включать и путь от корневого каталога сервера. В записи пути по дереву каталогов сервера используется символ ‘/’, а не ‘\’, как принято в Dos и Windows. Корневой каталог на сервере – совсем не обязательно “головной каталог диска”, как на локальной машине, а при соединении с сервером мы получаем доступ не ко всем его папкам и файлам, а только к тем, которые на нем специальным образом “размещены” и открыты для просмотра через WWW.
- *закладка*, позволяющая перейти в нужную часть документа. Имя закладки отделяется от имени файла символом '#'.

В целях совместимости имена файлов, размещаемых в Интернет, обычно строятся по правилам DOS - то есть, состоят из латинских букв, цифр

и символа подчеркивания и имеют длину не более 8 символов. С другой стороны, web-страницам принято давать раширение *.html, а не *.htm.

Следует также помнить, что URL чувствителен к регистру символов, то есть, http://www.TNT.ru и http://www.tnt.ru – это не один и тот же адрес.

Например, один из файлов сайта pers.narod.ru имеет полный URL http://pers.narod.ru/text/eten_x500plus.html, из чего можно заключить, что он находится на сервере pers.narod.ru в папке с именем text.

Если в URL не указано имя html-файла, это означает, что документ имеет имя по умолчанию, которое может назначаться при администрировании сервера. Чаще всего это имя index.html, так что URL http://www.host.ru может означать то же самое, что http://www.host.ru/index.html.

Ссылка, в которой указан полный URL документа, называется **абсолютной**. Абсолютные ссылки используются для связи с внешними ресурсами Интернет, URL которых известен нам и не меняется.

Относительная ссылка ссылается на документ, опуская общую адресную часть. Например, если из документа index.html нужно сослаться на документ test.html, находящийся в той же папке, это можно сделать ссылкой вида

документ test.html

При использовании относительной ссылки можно ссылаться на папки, которые являются как вложенными, так и родительскими по отношению к папке, в которой расположен исходный документ. Например, ссылка на рисунок с именем my.jpg, находящийся во вложенной папке images может иметь вид

посмотритерисунок

Внутренние ссылки предназначены для навигации в пределах документа. Они имеют вид якорь ссылки

Для создания **закладки** служит тэг вида **якорь закладки**. Имена закладок должны быть уникальными в пределах документа.

Например, если первое слово документа заключить в тэг..., в нижней части документа можно разместить ссылку вида

к началу страницы...

При разработке собственного web-сайта для перехода между его страницами используются, как правило, относительные ссылки, что позволяет просматривать сайт на локальной машине, не внося в него изменений, а также при необходимости легко переместить сайт на другой сервер.

Посмотрите примеры различных видов ссылок в таблице 4:

Обычная ссылка	Это обычная гипертекстовая ссылка. Щелчок по ней вызовет переход к новому документу (в данном случае - к главной странице нашего сайта).
Почтовая ссылка	Это ссылка на почтовый адрес. Щелчок по ней приведет к запуску окна создания сообщения, которое можно отправить по указанному в ссылке адресу. Если же ни одна почтовая программа на машине не установлена, браузер сообщит об этом в окне диалога. Опция target при этом не указывается.
FTP-ссылка	Это ссылка на FTP-сервер, содержащий файловые архивы. При щелчке по ней браузер перейдет в режим FTP-клиента и покажет содержимое головного каталога архива. При этом, панели инструментов браузера изменятся. Если же на машине установлена внешняя программа поддержки FTP, она будет запущена.
Ссылка на файл	Это ссылка на архивный файл типа *.zip, находящийся на сервере. После щелчка по ней браузер обычно спрашивает в окне диалога, что следует сделать с архивом - открыть в текущем

	положении или сохранить на диске. Такие же вопросы задаются и о многих других типах файлов, в том числе об исполняемых программах *.exe
Ссылка на картинку	Это ссылка на картинку типа *.gif, которая откроется в новом окне.
Локальная ссылка	Щелчок по этой ссылке откроет в новом окне содержимое диска с: Ссылка имеет вид file://c:/
Ссылка с хитростью	Это внутренняя ссылка на закладку в документе. Но HTML-код ссылки таков, что браузер при щелчке по ней откроет в новом маленьком окне картинку. Поэтому не удивляйтесь, если при посещении некоторых страниц окна вдруг "вываливаются ниоткуда"... это один из рекламных приемов в Сети.

Списки

HTML поддерживает **нумерованный и маркированный списки**, открывающиеся. соответственно, тэгами<OL> и <UL>. Опция COMPACT делает список более компактным. Опция TYPE позволяет указать тип маркеров списка. Для нумерованного списка ее возможные значения - "A", "I" или "1"(по умолчанию), обозначающие, соответственно, нумерацию латинскими буквами, римскими или обычными цифрами. Для маркированного списка опция TYPE указывает вид маркера - circle, disc или square. Опция START="число" нумерованного списка позволяет начать нумерацию с цифры, отличной от 1 или буквы, отличной от A.

Отдельные **элементы списка** заключаются в тэг<LI>. Для нумерованного списка можно изменить нумерацию, указав в тэге<LI> опцию VALUE="число".

Для создания **комбинированных списков** тэги нумерованных и маркированных списков можно вкладывать друг в друга. Слева показан пример списка, а справа его HTML-код:

1. Компьютеры • PentiumPro • Pentium MMX 2. Принтеры ▪ матричный ▪ струйный ▪ лазерный	<olcompact> - Компьютеры <ulcompacttype="disc"> - PentiumPro - Pentium MMX - Принтеры <ulcompacttype="square"> - матричный - струйный - лазерный

Кроме того, HTML поддерживает **список определений**, содержащий чередующиеся пары вида "термин"-"его описание". Список определений начинается с тэга<DL>, термин выделяется тэгом<DT>, а описание, которое обычно выводится со сдвигом вправо, тэгом<DD>.

Таблицы

Важным инструментом Web-дизайна являются **таблицы**, которые используются не только для вывода табличных данных, но и для управления взаимным размещением текста и графики, создания колонок газетного типа, цветовых эффектов и т.д. Следует помнить, однако, что браузер отображает содержимое таблицы только по окончании ее загрузки, поэтому если весь 50- или 100-килобайтный документ разместить в одной гигантской таблице, ни один пользователь, скорее всего, не дождется загрузки такого документа. Таблица размещается в тэге<TABLE>, имеющем ряд опций:

Таблица 5 Основные опции тэга TABLE

Опция	Назначение
ALIGN="выравнивание"	Выравнивание всей таблицы относительно текста, в котором она находится (left, center или right)
BORDER="число"	Ширина обрамления таблицы в пикселах
CELLSPACING="число"	Расстояние между ячейками в пикселах
CELLPADDING="число"	Размер свободного пространства между границами ячейки и ее содержимым в пикселах
HSPACE="число"	Размер свободного пространства слева и справа от таблицы, в пикселах
VSPACE="число"	Размер свободного пространства сверху и снизу от таблицы, в пикселах
WIDTH="ширина"	Требуемая ширина таблицы в пикселах или в процентах от ширины окна бруазера

Таблица формируется **по строкам**, причем, каждая строка заключена в тэг<TR>...</TR>, а каждая ячейка строки - в тэг<TD>...</TD>.Закрыватьтэг</TABLE> и тэги строк и столбцов обязательно. Можно изменять выравнивание объектов в ячейках таблицы, используя в тэгах<TR> и <TD> опции ALIGN (так же, как в тэге<TABLE>) и VALIGN для указания способа выравнивания ячеек по вертикали - TOP (по верхнему краю), BOTTOM (по нижнему краю) или CENTER (по центру).

Для явного указания ширины ячейки в тэге<td> также можно использовать опцию width, а опция nowrap запрещает браузеру распределять текст по всей ячейке, так, что на экране отображается лишь та часть текста, которая умещается по длине.

Об управлении цветами в таблице будет сказано в разделе 7.

Для того, чтобы **растянуть** ячейку таблицы на несколько строк или столбцов, в тэге ячейки <TD> используются опции ROWSPAN="число" и COLSPAN="число", указывающие, сколько строк и сколько столбцов таблицы охватывает данная ячейка. Если ячейку следует оставить пустой, в нее обычно помещают "жесткий" символ пробела

Примеры:

<table>
<tr><td colspan="2" rowspan="2"> </td><td colspan="2">Характеристики</td></tr>
<tr><td>Средний рост, см</td><td>Средний вес, кг</td></tr>
<tr><td rowspan="2">Пол</td><td>Мужской</td><td>177</td><td>73</td></tr>
<tr><td>Женский</td><td>166</td><td>65</td></tr>
</table>

Данная таблица создана с помощью следующего HTML-кода:

```
<table border="1" cellpadding="5" cellspacing="0">
<tr>
<td colspan="2" rowspan="2"> </td>
<tdcolspan="2">Характеристики</td>
</tr>
<tr>
<td>Средний рост, см</td>
<td>Средний вес, кг</td>
</tr>
<tr align="center">
<td rowspan="2">Пол</td>
<td>Мужской</td>
<td>177</td>
<td>73</td>
</tr>
<tr align="center">
<td>Женский</td>
<td>166</td>
<td>65</td>
</tr>
</table>
```

Следующая таблица использует 90% ширины окна браузера и различные способы выравнивания данных в ячейках:

Заголовок	
Данные, выровненные по левому краю ячейки	Данные, выровненные по правому краю ячейки
25% ширины таблицы	75% ширины таблицы

```
<table width="90%" align="center" border="5"
cellpadding="2" cellspacing="0">
<trvalign="center">
<td align="center" colspan="2"><big>Заголовок</big></td>
</tr>
<trvalign="top">
<tdalign="left">Данные, выровненныепо<br>
левомукраюячейки</td>
<tdalign="right">Данные, выровненные по<br>
правому краю ячейки</td>
</tr>
<tr>
<tdwidth="25%">25% ширины таблицы</td>
<tdwidth="75%">75% ширины таблицы</td>
</tr>
</table>
```

Обратите внимание на оформление таблиц - как и при написании программ, при "ручном" создании Web-страниц удобнее всего набирать операторы одного уровня с одинаковым сдвигом вправо. Во втором примере

также характерно, как ширина ячеек второй строки неявно определяется шириной ячеек третьей.

Сложные и красивые эффекты могут быть достигнуты вложением таблиц друг в друга - внутренняя таблица при этом должна быть целиком вложена в тэг<td> внешней таблицы.

Фреймы

Фреймами или кадрами называют независимые окна внутри окна браузера, в которых могут одновременно отображаться разные документы. Фреймы удобны при создании страниц, которые должны иметь как динамическое, так статическое содержимое. Например, узкий левый фрейм может содержать оглавление сайта, а широкий правый будет предназначен для вывода информации. Возможны также любые другие конфигурации.

Документ, использующий фреймы, содержит только описание внешнего вида фреймов, которое выполняется в тэге **<frameset>...</frameset>**, используемом **вместо тэга<body>**. У тэга<frameset> есть две главных опции - rows="список величин" и cols="список величин", описывающие, соответственно, **строки** и **столбцы** таблицы фреймов.

Строка "список величин" представляет собой разделенный запятыми перечень значений в **пикселах**, **процентах** или **относительных величинах**. Например, тэг<framesetrows="100,240,140"> создает сетку из 3 кадров, **высота**которых равна 100, 240 и 140 пикселов соответственно, <framesetcols="25%,75%"> создает 2 кадра с **шириной** 25 и 75 процентов от ширины окна браузера, а <framesetcols="*,2*"> - 2 кадра с шириной 1/3 и 2/3 ширины окна браузера.

Эти способы можно использовать и **совместно** - например, <frameset cols="128,*"> указывает оставить 128 пикселов слева под первый фрейм, а все остальное пространство - под второй. При определении обоих атрибутов rows и cols, например, <frameset rows="*,2*" cols="*,2*"> получается сетка кадров.

Другие опции тэга<frameset>:

border="ширина" - указывает **ширину обрамления всех рамок** для всех кадров, в пискелах;

frameborder=yes или frameborder=no - включает или выключает **отображение обрамления** кадров. В случае yes рамка имеет трехмерную форму, иначе она невидима, то есть имеет цвет фона окна по умолчанию. Некоторые браузеры "понимают" эту опцию только в виде frameborder=1 или frameborder=0, поэтому обычно указывают оба способа;

framespacing="ширина" - указать **ширину промежутка** между смежными кадрами в пискелах. Не действует в NetscapeNavigator.

Внутри тэга<frameset> находятся **описания отдельных кадров**, каждое в собственном тэге **<frame>**, закрывать который не нужно. Число тэгов<frame> должно быть равно числу кадров, определенных в тэге<frameset>, при этом считается, что кадры описываются слева направо и сверху вниз.

Перечислим основные опции тэга<frame>:

Опция src="url" указывает **URL исходного документа** для данного кадра;

Опция name="строка" указывает **имя кадра**. Это необходимо сделать, если предполагается ссылаться из одних кадров на другие;

Опция scrolling="значение" управляет **линейками прокрутки** кадра. Значение может быть задано в виде yes (линейки есть всегда), no (никогда) или auto (если необходимо);

Опция noresize запрещает изменять **размеры кадра**. Опция noresize, указанная для данного кадра, влияет также и на все кадры, смежные с ним;

Опция frameborder может указываться также внутри тэга<frame>, со всеми замечаниями, которые сделаны относительно нее. Указание этой опции в тэге<frame> отменяет указание, сделанное в тэге<frameset> для данного кадра и всех, смежных с ним;

Опции magrinheight="ширина" и marginwidth="ширина" задают **размещение** по верхней-нижней и боковым сторонам кадра **областей свободного пространства**, ширина которых указывается в пикселах.

После того, как тэг<frameset> закрыт, можно использовать тэг **<noframes>...</noframes>**, определяющий содержимое, которое будет выводиться браузерами, не поддерживающими кадры.

Для создания более сложных конфигураций кадров тэги<frameset> могут **вкладываться** друг в друга - внутренний тэг<frameset> может быть вложен вместо любого из тэгов<frame>.

Пример документа с фреймами:

```
<html>
<head></head>
<frameset rows="40,*" border="0" frameborder="0" frameborder="no">
<frame src="reclama.html" name="top" scrolling="no" noresize>
<frameset cols="128,*" border="0" frameborder="0" frameborder="no">
<frame src="menu.html" name="menu" scrolling="auto">
<frame src="index.html" name="main" scrolling="yes">
</frameset>
</frameset>
<noframes>
Извините, Ваш браузер не поддерживает кадры!
</noframes>
</html>
```

Здесь верхний кадр высотой 40 пикселов может служить, например, для вывода рекламы и связан с файлом reclama.html. Остальная часть окна разбита на 2 колонки. Левая колонка имеет ширину 128 пикселов и может быть предназначена для вывода меню (документ menu.html). Третий кадр занимает основную часть окна и предназначен для вывода информации (файл index.html).

Для создания **ссылки из одного кадра в другой** достаточно указать в тэге ссылки опцию вида **target="имя кадра"**. Например, для ссылки из кадра menu в кадр main достаточно написать в документе menu.html ссылку вида ... При щелчке по этой ссылке содержимое правого кадра, то есть, документ index.html, будет заменено на новый документ.

При работе с кадрами можно также указывать опцию **target** в одном из видов **_self** (загрузить в тот же кадр, откуда делается ссылка), **_parent** (загрузить в родительский для данного кадр; если такого нет - результат действия аналогичен _self) или **_top** (загрузить в полное окно, разрушая все кадры).

Глава 7. Графика и работа с изображением.

Графика

Как правило, браузеры поддерживают рисунки в форматах GIF и JPG.

Формат GIF обычно используется для хранения рисунков с четкими деталями, небольшим набором цветов (до 256) и возможностью анимации ("мультипликация" на Web-страницах, графические кнопки, "украшения"). Кроме того, формат GIF поддерживает эффект прозрачности, то есть, точки определенного цвета можно сделать того же цвета, что и фон страницы. Второй полезный эффект - возможность черезстрочной загрузки изображения GIF - то есть, изображение может постепенно "проявляться" по мере загрузки страницы, что даст пользователю возможность уже на ранней стадии загрузки получить представление о содержании картинки.

Формат JPG используется для хранения полноцветной графики и фотоизображений. Этот формат поддерживает 24-битовую графику, то есть, 16.7 миллиона цветов. Благодаря компрессии изображений, столь многоцветные файлы JPG имеют приемлемые размеры, но сжатие производится за счет некоторой потери качества. Поэтому при подготовке

иллюстраций для Web-страницы всю предварительную обработку следует производить с несжатыми изображениями, например, в формате BMP, а в GIF или JPG конвертировать только окончательный вариант картинки.

Из доступных приложений эффективно управлять файлами в форматах GIF и JPG позволяет MicrosoftPhotoEditor, входящий в пакет программ MicrosoftOffice. Профессионалы используют гораздо более мощные графические пакеты, такие как AdobePhotoShop и CorelDraw.

Рисунок в любом формате вставляется в документ HTML тэгом следующего вида:

<IMG SRC="URL рисунка" WIDTH="ширина" HEIGHT="высота" BORDER="рамка" ALIGN="выравнивание" HSPACE="отступ по горизонтали" VSPACE="отступ по вертикали" ALT="текст">

Закрывать этот тэг не нужно.

Опция SRC содержит абсолютный или относительный адрес рисунка, если рисунок не найден, на его месте выведется пустая рамка. При разработке сайта используются, как правило относительные адреса. Хотя SRC - единственная обязательная опция тэга, указание всех остальных опций настоятельно рекомендуется.

Опции WIDTH и HEIGHT определяют ширину и высоту рисунка в пикселах (точках). Если ширина и высота не указаны, загрузка страницы замедляется и часто приводит к некрасивому эффекту "скачущего" текста на экране. Если указанные ширина и высота не соответствуют действительным размерам рисунка, при выводе он будет отмасштабирован, что также существенно замедляет загрузку (единственное допустимое исключение - уменьшение или увеличение при загрузке в 2 раза).

Значение опции BORDER также указывается в пикселах и определяет ширину рамки вокруг рисунка, являющегося гиперссылкой. Если рамка не нужна, значением этой опции следует сделать 0, а если рисунок не является ссылкой, опцию можно не указывать.

Опция ALIGN определяет размещение рисунка относительно текста, единственные понимаемые всеми браузерами значения - LEFT (слева) и RIGHT (справа). Опции HSPACE и VSPACE, указываемые в пикселах, определяют размер свободного места вокруг рисунка по горизонтали и вертикали. Рисунок не выглядит сливающимся с текстом при их значениях в пределах 5-15.

Текст, переданный в опции ALT, выводится вместо рисунка, если в браузере отключен вывод рисунков или они не успели загрузиться, а также показывается в качестве подсказки при наведении "мышки" на рисунок. Обычно этот текст содержит название рисунка.

Картированные изображения

Сегодня многие Web-страницы располагают интересной разновидностью меню - *картированными изображениями*, то есть, картинками, отдельные части которых чувствительны к нажатию кнопки "мыши". Обычно чувствительные части изображения связаны с HTML-документами, то есть, являются ссылками.

В качестве графического формата для изображения-карты, как правило, выбирается формат GIF с чересстрочной загрузкой, который поддерживается практически всеми браузерами и позволяет выводить изображение, постепенно уточняя детали. Следует позаботиться о том, чтобы размер файла-изображения не был слишком большим - как и всегда при работе с графикой.

Использование изображения-карты вместо обычных гиперссылок предпочтительнее, если этого требует графический дизайн страницы. Однако, не следует забывать о пользователях, браузер которых не поддерживает вывод графики или этот вывод отключен - если на странице, помимо картированного изображения не предусмотрено других средств навигации, они не смогут воспользоваться Вашей страницей. Поэтому наряду с картированным изображением Web-мастер обычно разрабатывает обычный текстовый вариант меню.

Изображение-карта может обслуживаться как сервером, так и на стороне клиента. Мы рассмотрим только последний вариант, так как для его реализации не нужно иных средств, кроме тэгов HTML.

Для определения **областей картинки**, чувствительных к нажатию "мыши", их следует описать в специальном тэге<MAP>, имеющем вид

<MAP NAME="имя_карты">...</MAP>

Располагать тэг<MAP> можно в любом месте документа, но для удобства это обычно делают в начале документа после тэга<BODY>.

Внутри тэга<MAP>**каждая чувствительная область изображения** описывается в тэге<AREA>. Его общий вид следующий:

<areashape="форма" coords="список координат" href="URL">

Закрывать тэг<AREA> не нужно.

Форма может принимать одно из значений **rect** (прямоугольник), **poly** (многоугольник), **circle** (круг). Если эта опция не указана, подразумевается значение rect. При наложении двух зон действует значение, определенное первым тэгом<AREA>.

Список координат зависит от выбранной формы и включает в себя координаты в пикселах, перечисленные через запятую. Как обычно, сначала указывается X-, а затем Y-координата и при этом считается, что ось Y расположена сверху вниз от верхнего левого угла рисунка. Левый верхний угол имеет координаты (0,0). Для прямоугольника указываются координаты левого верхнего и правого нижнего углов, например <areashape="rect" coords="0,0,200,100">. Для круга указываются X- и Y-координаты центра и радиус в пикселах, например, <areashape="circle" coords="263,200,50" href="1.html">. Для многоугольника перечисляются пары X- и Y-координат вершин. Для надежности многоугольник должен быть замкнутым, то есть, первая пара вершин совпадает с последней.

В опции href="URL" как обычно указывается **URL-адрес** документа, адресуемого данной областью. Если чувствительная область определена, но

не связывается ни с одним документом, вместо данной опции указывается значение nohref.

Для связывания изображения с созданной картой достаточно в тэге <img>, с помощью которого вставлено изображение, указать опцию вида usemap="#имя карты", например, <imgsrc="1.jpg" width="526" height="400" border="1" usemap="#a1">. Здесь использована карта с именем a1, находящаяся в текущем документе.

Глава 8. Формы.

Формы являются наиболее популярным способом "обратной связи" с пользователем. С помощью HTML можно создавать как простые формы, предполагающие выбор одного из нескольких ответов, так и сложные формы для заказов или для того, чтобы получить от пользователей страницы какие-либо комментарии и пожелания.

Форма представляет собой несколько полей, где пользователь может ввести некоторую информацию, либо выбрать какую-то опцию. После того, как пользователь отправит информацию, она обрабатывается программой (скриптом), размещенной на сервере. Существует также возможность обрабатывать формы "на стороне клиента", встраивая в свои страницы скрипты, написанные на языках JavaScript и VisualBasicScript.

Форма открывается тагом **<FORM>**, имеющим несколько опций.

Опция **action="url"** указывает URL, который примет и обработает данные формы. Если эта опция не указана, данные отправляются по адресу страницы, на которой размещена форма.

Опция **method="стиль"** указывает метод передачи данных программе-обработчику формы. "Стиль" может принимать одно из двух значений. Значение get, используемое по умолчанию, предписывает посылать информацию формы вместе с URL, а значение post предписывает посылать информацию формы отдельно от URL. Значение post используется обычно в

случае отправки данных формы по электронной почте или при необходимости передавать значительный объем информации.

Опция **name="имя"** указывает имя формы. Это необходимо, если требуется доступ к данным формы с помощью встроенного скрипта на JavaScript или VisualBasicScript, а также в том случае, если данные формы предназначены для отправки по электронной почте.

Опция **enctype="кодирование"** задает способ кодирования данных формы. В случае отправки данных как текста указывается в виде enctype="text/plain".

Примеры:

Заголовок формы, используемой на поисковой машине Яndex, имеет вид:

<formname="web" method="get" action="/yandsearch">

Заголовок формы, отправляющей информацию на адрес E-mail, может иметь вид:

<formaction="mailto:vita@lvs.ru?subject=Internet-Test" method="post" enctype="text/plain" name="Q" onSubmit="returnValidate()">

Здесь форма имеет имя Q, в качестве action указана отправка сообщения по E-mail на адрес vita@lvs.ru с темой "Internet-Test", опция enctype предписывает отправлять данные формы как текст, а опция onSubmit связывает отправку формы с функцией Validate(), написанной на Javascript. Этот прием широко используется для проверки корректности заполнения формы.

Внутри тага<form> находятся **поля** формы. Перечислим основные из них:

<textareaname="имя" cols="число столбцов" rows="число строк" wrap="стиль">...</textarea>

Таг предназначен для создания **многострочного поля ввода**. Опции cols и rows указывают число строк и столбцов в поле, опция wrap указывает

на режим автоматического распределения текста в ячейке. Она может принимать одно из значений off (выключен), virtual (распределять текст по всей ячейке, но на сервер передавать как одну строку) или physical (распределять текст по всей ячейке и передавать на сервер так, как он отображается). Опция name здесь и далее обозначает имя поля и предполагается обязательной. Текст, написанный в таге<textarea>, становится его значением по умолчанию.

Таг<selectname="имя" size="размер" multiple>...</select> определяет в форме **меню** с одним или несколькими вариантами выбора или **список** с полосой прокрутки. Если опция size указана в виде size="1", отображение элементов будет организовано в виде ниспадающего меню, в противном случае будет использован список прокрутки, включающий указанное число элементов. Опция multiple, если она указана, разрешает выбирать из списка более одного значения. Это можно сделать, например, выбирая элементы списка при нажатой клавише Ctrl.

Внутри тага<select> размещаются **элементы** меню или списка, каждый - в своем таге **<option>**, имеющем общий вид

<optionvalue="значение" selected>текст</option>

Опция value указывает значение, возвращаемое программе обработки при выборе пользователем данной опции, опция selected указывает на элемент списка, выбранный по умолчанию. Внутри тага<option> пишется текст, видимый в меню или списке на экране.

Таг<input>, в отличие от <textarea> и <select>, не должен закрываться и предназначен для сбора информации различными способами, включая **текстовые поля**, поля для ввода **пароля**, **переключатели**, **флажки**, **кнопки** для отправки данных и очистки формы.

<inputtype="text" name="строка" maxlength="максимальный размер" size="число символов" value="строка">

Создает **поле ввода**. Опция maxlength ограничивает максимальную длину вводимого текста, а опция size показывает максимальное количество

отображаемых символов. Опция value указывает исходное значение поля ввода.

Текстовое поле **с защитой** вводимых символов (то есть, с заменой их на звездочки, как принято при вводе паролей) можно создать, если заменить опцию type="text" наtype="password". Остальные опции поля ввода пароля - те же самые.

<inputtype="checkbox" name="строка" value="строка" checked>

Создает **флажок**. Опция value определяет значение, возвращаемое программе обработки при выборе пользователем флажка, опция checked, если она указана, делает флажок выбранным по умолчанию.

<inputtype="radio" name="имя" value="строка" checked>

Создает **радиокнопку**. Радиокнопки можно группировать, задавая одно и то же значение опции name. Опции value и checked имеют те же значения, что и у флажка.

<inputtype="reset" value="строка">

Создает кнопку **для очистки формы**. Значение опции value позволяет указать надпись для кнопки. После очистки все элементы формы принимают значения, которые они имели по умолчанию.

<inputtype="submit" value="строка">

Создает кнопку **для отправки данных** формы приложению-обработчику. Опция value позволяет указать надпись на кнопке.

Для создания произвольной кнопки вместо reset или submit указывается значение button. В основном это требуется при написании скриптов, работающих на стороне клиента. Например, кнопка, созданная тагом<INPUT TYPE="button" VALUE="Вывести" onClick="Look()"> подписана словом "Вывести", а при нажатии на нее вызывает функцию Look(), написанную на JavaScript.

Пример разработки формы:

Ваше имя:

Ваш пол: Мужской

Женский

На каком курсе Вы учитесь? Первый

Какие языки программирования Вы знаете?

Паскаль

Си

Ассемблер

Напишите несколько слов о себе:

Отправить данные Очистить форму

Эта форма имеет следующий HTML-код:

<form method="post" action="mailto:pers@mail.ru?subject=Information">

<pre>Вашеимя: <input name="name" type="text" maxlength="40" size="40" value="">

Вашпол: <input type="radio" name="pol" value="male">Мужской

<input type="radio" name="pol" value="female">Женский

На каком курсе Вы учитесь? <select name="curs" size="1">

<option value="1" selected>Первый</option>

<option value="2">Второй</option>

<option value="3">Третий</option>

<option value="4">Четвертый</option>

<option value="5">Пятый</option>

</select>

Какие языки программирования Вы знаете?

<input type="checkbox" name="pascal" value="yes" checked>Паскаль

<input type="checkbox" name="c" value="yes">Си

<input type="checkbox" name="asm" value="yes">Ассемблер Напишите несколько слов о себе:</pre>

<textarea name="about" rows="6" cols="60" wrap="off"></textarea><p><input type="submit" value="Отправить данные">

<input type="reset" value="Очистить форму">

</form>

Эта форма пытается отправить данные по указанному в заголовке адресу электронной почты, используя тему письма "Information". Дальнейшие события зависят от того, установлена ли на машине пользователя почтовая программа-клиент. Если да, то информация будет отправлена, причем данные из полей формы придут в виде строк текста name=value, например, информация о курсе в виде curs=1. Если на машине нет программы электронной почты, Windows предупредит об этом.

Обратите внимание, что для лучшего выравнивания данных часть формы заключена в тэг <pre>, а в части, находящейся вне тэга<pre>, использованы стандартные для HTML способы форматирования текста, например, тэг абзаца.

Глава 9. Мультимедийные возможности HTML-5.

Четвертая версия языка разметки html была создана еще в далеком 1997 году. С тех пор прошло уже 20 лет, и за это время представление информации на страницах сайтов изменилось кардинально. Современный интернет это обилие мультимедийных сайтов, масса разнообразного контента, к сожалению одного html для реализации поставленной задачи тут недостаточно, разработчикам приходится обращаться к дополнительным решениям, таким как java-скрипты, flash.

Выход HTML 5 призван решить эту проблему, обновленная версия языка разметки способна решить множество проблем, возникающих не только у разработчиков, но и пользователей. С ростом версии, возникли и требования к точности, браузеры должны точно понимать, как следует себя

вести в той или иной ситуации, и как исправлять ошибки в случае их возникновения. HTML 5 призван для более простого создания сайтов поддерживающих современные технологии. Теперь просмотр видео, веб-софт, мобильные версии ресурсов можно реализовать, используя лишь «голый» html. Но, не стоит забывать о сайтах, построенных еще с применением HTML 4, их отображение в браузере так же должно быть корректным.

Структура HTML 5

В настоящее время, если просмотреть исходный код практически любого сайта, можно увидеть, что страница формируется с использованием тега

<div>, позиционирование которого управляется в каскадных таблицах (css). Если смотреть на страницу глазами браузера, они не могут нести какой-либо смысловой нагрузки. В основе html 5 заложены «смысловые» блоки для формирования страницы. То есть, теперь для отображения информации в «шапке» сайта используется тег

<header>, в «подвале» ресурса - <footer>

Причем это касается не только основной страницы, тег

<header> может быть использован в теле тега

<article> (статьи), тем самым сформировав «шапку» статьи, указав в ней автора статьи, дату публикации и прочее.

Для работы с мультимедийной частью сайта можно использовать теги

<display>, тем самым указав браузеру что в теле тега содержится контент, его, по желанию пользователя, можно скрыть или отобразить. Для отображения информации при свернутом блоке<display> используется вложенный в него тег <summary>

Благодаря такому подходу, со временем, можно полностью отказаться от применения небезопасного JavaScript.

Мультимедийные возможности HTML5

В истории всемирной сети каждый очередной виток перехода на новый уровень развития начинался с какого-нибудь технологического нововведения. Когда в HTML добавился элемент *img*, это в корне изменило облик сети. Затем введение Javascript сделало ее более динамичной и интерактивной. Чуть позже появился Ajax, что открыло возможности для создания в сети полноценных приложений.

Современные веб-стандарты настолько продвинуты, что сейчас можно создать почти что угодно, используя лишь возможности HTML, CSS и Javascript. *Почти* что угодно.

В спецификациях этих стандартов все еще есть пробелы. Так, если вы хотите сваять страницу с текстом и картинками, вы вполне обойдетесь HTML и CSS. Но если вам нужно опубликовать аудио или видео, тут неизбежно придется обратиться к сторонним технологиям — Flash или Silverlight.

Эти технологии — «плагины», эдакие «затычки», заполняющие «дыры» в сети. Они делают относительно простой публикацию игр, фильмов и музыки онлайн, но они не открыты и принадлежат и контролируются частными компаниями. Да, тот же Flash — мощный инструмент, но его применения в какой-то мере схоже со сделкой со злыми силами: мы получаем новые, недоступные другим путем, возможности, но взамен теряем часть свой независимости.

HTML5 призван восполнить этот недостаток. В данный момент он вступает в прямую конкуренцию с собственническими технологиями, вроде Flash и Silverlight, и главное его преимущество в этой борьбе — ему не требуется плагины, так как его мультимедиа-возможности «вшиты» в браузеры.

Canvas

Когда в браузере Mosaic появилась возможность вставлять на страницы картинки, это дало сети мощный толчок вперед. Но вплоть до настоящего момента картинки остаются статичными. Да, можно делать анимированные

гифки, обновлять стили картинок на лету при помощи JS, генерировать их на стороне сервера. Но в любом случае — как только картинка открыта в браузере, нельзя изменить ее содержимое.

И тут приходит элемент *canvas*, предназначенный для создания динамически изменяемых изображений.

Сам по себе он очень прост. Все, что вы указываете в параметрах тега — это размеры холста:

```
<canvas id="my-first-canvas" width="360" height="240"></canvas>
```

Как видно, тег этот парный. Но то, что вы поместите внутри его, предназначено только для браузеров, которые этот элемент не поддерживают:

```
<canvas id="my-first-canvas" width="360" height="240">
<p>No canvas support? Have an old-fashioned image instead:</p>
<img src="puppy.jpg" alt="a cute puppy">
</canvas>
```

Пользователи браузеров без поддержки canvas увидят фотку милого щенка.

Вся работа по отрисовке возлагается на Javascript. Перво-наперво, надо указать, с каким элементом мы работает и в каком контексте. «Контекст» в данном случае — это API, и он на сегодняшний день всего один — двухмерный (как видно, есть куда расти):

```
var canvas = document.getElementById('my-first-canvas');
var context = canvas.getContext('2d');
```

Теперь мы можем начать рисовать на этом двухмерном холсте элемента *canvas*, используя API задокументированный в спецификации HTML5. Набор инструментов здесь схож с тем, что вы найдете в любом графическом редакторе вроде Illustrator-а: линии, заливки, градиенты, тени, фигуры, кривые Безье. Разница лишь в том, что вместо рисования мышкой, нужно пользоваться командами в Javascript.

Рисование кодом

Так цвет линий делается красным:

```
context.strokeStyle = '#990000';
```

Теперь все, что вы нарисуете, будет иметь красную обводку. К примеру, синтаксис создания прямоугольника выглядит так:

```
strokeRect ( left, top, width, height )
```

Если вы хотите сделать этот прямоугольник высотой 50 пикселей, шириной 100, и расположить его на 20 пикселей от левого края элемента *canvas* и на 30 от верхнего, написать нужно следующее:

```
context.strokeRect(20,30,100,50);
```

Прямоугольник, нарисованный командами JS в canvas-е.

Конечно, это элементарный пример. Двухмерный API включает множество разных методов: fillStyle, fillRect, lineWidth, shadowColor и еще много других.

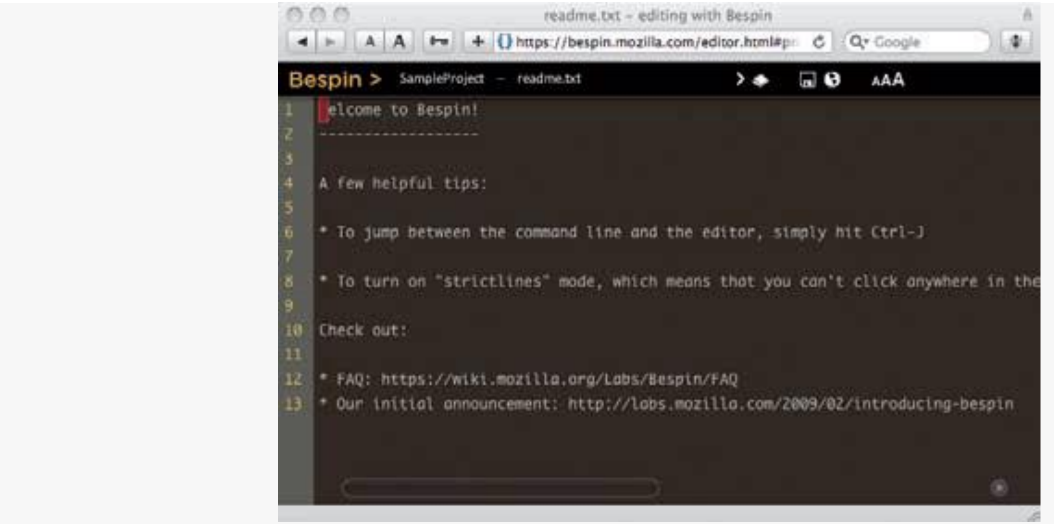
Теоретически, любое изображение, которое можно создать в том же Иллюстраторе, может быть точно так же создано при помощи элемента *canvas*. На практике, впрочем, это может оказаться довольно трудоемко и даст на выходе огромное количество кода. С другой стороны, это и не совсем то, для чего был *canvas* придуман.

И для чего же тогда? Какой с него толк?

Это здорово, что используя Javascript и *canvas*, можно создавать векторные изображения на лету, но если они делаются сколько-нибудь сложными, эта работа не будет оправдывать себя.

Смысл и главная фича *canvas*-а заключаются в том, что его содержимое можно динамически обновлять, отрисовывая новые элементы в ответ на действия пользователя. Эта его способность реагировать на события, инициируемые посетителем страницы, делает возможным создание таких инструментов и игр, которые раньше требовали ли бы применения сторонних технологий, вроде Флеша.

Одним из первых примеров возможностей *canvas*-а стал проект Bespin от Mozilla Labs — приложение, являющееся простым текстовым редактором для кодинга, которое работает в окне браузера.



Веб-приложение Bespin, созданное при помощи элемента canvas.

Это очень мощная вещь. Она впечатляет. Но также, она — хорошим пример того, что *не надо* делать с *canvas*-ом.

Доступ запрещен

Текстовой редактор, по своей сути, предназначен для работы с текстом. Bespin успешно работает с текстом внутри элемента *canvas*, с одной только оговоркой — это уже не текст. Это просто набор векторных форм, похожих на него.

Каждая страница в сети может быть описана с помощью DOM — Document Object Model, «объектная модель документа». DOM может включать в себя много разных «узлов», самые важные из них — элементы, атрибуты и текстовые объекты. Этих трех видов «кирпичиков» достаточно, чтобы построить практически любую страницу, какую вы только себе можете представить. В свою очередь, элемент *canvas*не имеет DOM — его содержимое нельзя разделить на отдельные части и представить как дерево узлов.

Устройства для чтения с экрана и другие технологии для схожих задач основываются на возможности анализа DOM для понимания структуры и смысла документа. Нет DOM — нет доступа.

Такая «изолированность» *canvas*-а — большая проблема для HTML5. К счастью, кое-какие умные люди работают над ее решениями.

Умный холст

Исходя из вышеизложенного, можно предположить, что на данный момент веб-дизайнерам нет смысла применять *canvas* в своих проектах. Это не так.

Когда используем Javascript на данном сайте, он выполняет исключительно роль дополнительного улучшения. Те посетители, у которых он отключен, не будут лишены доступа к информации, разве что только определенных второстепенных удобств. Такой многоуровневый подход под названием «Ненавязчивый Javascript» может быть так же применен и к *canvas*-у. Вместо создания нового контента, он будет использоваться для лучшего представления уже имеющегося.

Допустим, у вас есть таблица с каким-то данными. Вы хотите проиллюстрировать наблюдающуюся в них тенденцию при помощи графика. В случае если данные статичны — можно просто сгенерировать график-картинку, используя Google Chart API, к примеру. Но если данные можно изменять по ходу дела, то было бы здорово создать график, который каждый раз отрисовывается по новой с учетом изменений.*Canvas* здесь отлично подойдет — можно использовать Javascript для извлечения содержимого элемента *table* и создания на его основе математически рассчитанной иллюстрации.

Умные ребята из Filament Group даже разработали для этого плагин к jQuery, кстати говоря.

Примеры графиков, сгенерированных при помощи canvas

Есть и другой вариант. На самом деле, элемент *canvas* — не единственный API для генерации динамических изображений. SVG — Scalable Vector Graphics, масштабируемая векторная графика — XML формат, который может использоваться для описания тех же фигур, что и *canvas*. И исходя из сути XML как такового, содержимое SVG теоретически «понимаемо» для устойств для чтения с экрана.

Но на практике, SVG не произвел исключительного впечатления на разработчиков, в отличие от *canvas*, который, хоть и появился совсем недавно, уже широко поддерживается доступными браузерами. Его возможности даже реализуемы в IE, при помощи дополнительной Javascript-библиотеки.

Учитывая лозунги WHATWG про то, что надо «мостить натоптанные тропы» и «не изобретать велосипед», может показаться странным, что они предпочли включить *canvas* в HTML5, когда уже существует аналогичная технология SVG. Как это часто происходит, спецификация HTML5 на самом деле лишь документирует многое из того, что уже поддерживается браузерами. *Canvas* не был придуман для HTML5 — он впервые появился в Safari и был создан Apple. Когда разработчики других браузеров увидели эту идею, она им понравилась и была скопирована.

Это может показаться слегка бессистемным и неорганизованным, но факт — так рождаются многие наши стандарты. Взять хотя бы лежащий в основе Ajax объект XMLHttpRequest, впервые реализованный еще в конце прошлого века в Microsoft Internet Explorer 5.

В мире браузеров, где выживает наиболее приспособленный, *canvas* в данный момент набирает силу и популярность. Как только он станет более «открытым» для доступа извне, его позиции будут надежно закреплены.

Аудио

В те переходные годы, MP3 в конечном итоге вышел победителем за звание самого расхожего музыкального формата. Но для того, чтобы дать возможность посетителям элементарно прослушать звуковой файл со страницы, до сих пор требуется применение сторонних технологий. Здесь победитель явно Flash.

И теперь HTML5 планирует лишить его этого титула.

Вставка аудио-файла на страницу в HTML5 выглядит просто:

```
<audio src="witchitalineman.mp3">
</audio>
```

Но это слишком просто. Вам наверняка понадобится немного больше возможностей.

Допустим, есть на свете такой злобный гад, который ненавидит всемирную сеть и всех ее пользователей. Такому человеку наплевать, что это

очень грубо и просто глупо вставлять на страницу аудио-файл, который начинает проигрываться автоматически. Параметр *autoplay* придется ему по душе.

```
<audiosrc="witchitalineman.mp3" autoplay>
</audio>
```

Заметьте, что этот параметр не имеет значения; в смысле — используется сам по себе. Штуки такого рода называется *булевы параметры*, в честь величайшего математика Джоржа Буля. На разработанной им двоичной логике основаны все компьютерные системы, где базовые значения переменных всегда либо ноль, либо единица, true или false.

Не следует однако путать *булевы параметры* с *булевыми значениями*. Простительно подумать, что булев параметр обязан принимать значение *true* или *false*, но это не так. Сама суть его существования уже булева, то бишь бинарна, — он либо есть (*true*), либо его нет вовсе (*false*). Даже если вы добавите к такому атрибуту какое-либо значение, оно не даст никакого эффекта: *autoplay=«false»* или *autoplay=«no thanks»* — то же самое, что и просто *autoplay*.

Если же вы пользуетесь исключительно синтаксисом XHTML, то для вас — *autoplay=«autoplay»*. Одобрено Международным Департаментом Департамента Избыточности.

Когда авто-воспроизведение аудио-файла кажется недостаточно подлым, можно еще добавить другой булев параметр — *loop*, который зациклит звук и сделает пытку им бесконечной.

```
<audio src="witchitalineman.mp3" autoplay loop>
</audio>
```

Сочетание этих параметров дополнительно уменьшит ваши шансы остаться в живых при встрече со мной.

Все под контролем

Но давайте уже перейдем к тому, как применять элемент *audio* во благо. Логично добавить возможность управления вопроизведением, и это легко можно сделать при помощи булева параметра *controls*.

```
<audio src="witchitalineman.mp3" controls>
</audio>
```

Он включит отображение стандартных браузерных контролов для кнопки «плей-пауза» и регулятора громкости.

controls отображает «родной» браузеровский интерфейс управления воспроизведением

Если вам не нравится стиль контролов, задаваемый браузером, вы можете оформить их по своему вкусу. При помощи Javascript вы можете обратиться к Audio API, который обеспечит доступ к методам *play* и *pause* и свойству *volume*. Вот грубый пример такой кастомизации с применением элементов *button* и неблаговидных обработчиков событий внутри тегов:

```
<audio id="player" src="witchitalineman.mp3">
</audio>
<div>
<button onclick="document.getElementById('player').play()">
   Play
</button>
<button onclick="document.getElementById('player').pause()">
   Pause
</button>
<button onclick="document.getElementById('player').volume += 0.1">
   Volume Up
</button>
```

```
<button onclick="document.getElementById('player').volume -= 0.1">
Volume Down
</button>
</div>
```

Play Pause Volume Up Volume Down

Кастомные контролы, сделанные при помощи элементов button.

Буферизация

Какое-то время спецификация HTML5 включала в себя еще один булев парамер — *autobuffer*. Этот был куда вежливее и полезнее, чем мерзкий *autoplay*, — он указывал браузеру, что, хоть и аудио-файл не должен сразу же начать играть, его следут презагрузить в фоне, так как рано или поздно его все равно запустят.

Это был бы очень полезный параметр, но, к сожалению, Safari пошел дальше. Это браузер начал презагружать аудио файла, не обращая внимания на *autobuffer*. А так как — вы помните — *autobuffer* является булевым параметром, никак нельзя использовать его, чтобы запретить презагрузку: *autobuffer=«false»* — это то же, что и *autobuffer=«true»*, как и вообще любое значение (описание бага).

Так что теперь *autobuffer* был заменен на параметр *preload*. Он не булев, он принимает одно из следующих трех значений: *none*, *auto* и *metadata*. Используя *preload=«none»*, мы явно говорим браузеру, что презагружать этот файл не нужно.

```
<audio src="witchitalineman.mp3" controls preload="none">
</audio>
```

Если у вас на странице только один элемент *audio*, *preload=«auto»* вполне оправдан и логичен. Но не стоит делать этого если их несколько, дабы не обижать посетителей с не самым быстрым и/или безлимитным интернетом.

Я спою «поребрик», ты споешь «бордюр»

Вроде, кажется, что элемент *audio* идеален во всем. К сожалению, это так, но не со спецификацией элемента как такового, а с аудио-форматами.

Хоть MP3 и является в данный момент самым распространенным, этот формат не открытый. За возможность с ним работать требуется платить определенную сумму обладателям патента. Для больших корпораций вроде Apple или Adobe это явно не проблема, в отличие от более мелких компаний и опен-сорс групп. Оттого MP3 прекрасно работает в Safari, но не проигрывается в Firefox.

Существует, конечно, и другие форматы. Например, кодек Vorbis — обычно дающий на выходе файлы .ogg — не обременен никакими патентами. Этот работает в Firefox, но, в свою очередь, не поддерживается Safari.

Не нужно делать жестокий выбор в пользу лишь одного определенного формата. Вместо параметра *src* в теге*<audio>*, можно использовать несколько заключенных в него элементов source для разных файлов:

```
<audio controls>
  <source src="witchitalineman.ogg">
  <source src="witchitalineman.mp3">
</audio>
```

Браузер с поддержкой Ogg Vorbis подхватит первый файл и не пойдет дальше. Браузер, который умеет проигрывать MP3, но не понимает Ogg, пропустит первый файл и загрузит второй.

Можно помочь им в выборе, указав *mime-type* для каждого файла:

```
<audio controls>
  <source src="witchitalineman.ogg" type="audio/ogg">
  <source src="witchitalineman.mp3" type="audio/mpeg">
</audio>
```

Элемент *source* — это одиночный, «неконтейнерный» элемент, так что если вы пользуетесь XHTML-синтаксисом, тег нужно закрывать косой чертой:*<source/>*.

Для не столь одаренных

Возможность указать несколько *source*-ов — это здорово, но ведь есть браузеры, которые не поддерживает элемент *audio* совсем.

Internet Explorer и ему подобные требуют, чтобы аудио файлы им скармливали с ложки, через старый-добрый Flash. Модель элемента *audio* поддерживает это: все, что заключено внутри тега и не является элементом *source*, будет подано браузерам, которые не поддерживают *audio* — как элемент *object* в данном случае:

```
<audio controls>
  <source src="witchitalineman.ogg" type="audio/ogg">
  <source src="witchitalineman.mp3" type="audio/mpeg">
  <object type="application/x-shockwave-flash"
data="player.swf?soundFile=witchitalineman.mp3">
          <param name="movie"
value="player.swf?soundFile=witchitalineman.mp3">
  </object>
</audio>
```

Элемент *object* сам по себе тоже позволяет вставку альтернативного контента — для тех, у кого с браузером совсем все плохо, можно сделать стандартную ссылку:

```
<audio controls>
  <source src="witchitalineman.ogg" type="audio/ogg">
  <source src="witchitalineman.mp3" type="audio/mpeg">
  <object type="application/x-shockwave-flash"
data="player.swf?soundFile=witchitalineman.mp3">
```

```
        <param name="movie"
value="player.swf?soundFile=witchitalineman.mp3">
        <a href="witchitalineman.mp3">Download the song</a>
    </object>
</audio>
```

Таким образом, приорететы идут в следующем порядке:

1. Звук в формате Ogg Vorbis через элемент *audio*
2. Звук в формате MP3 через элемент *audio*
3. Звук через Flash
4. Ссылка для скачивания файла напрямую

Доступность содержимого

Модель элемента *audio* очень удобна для указания альтернативных методов представления его содержимого. Но если, к примеру, вы хотите предоставить текстовой вариант песни для тех, кто ее прослушать не может, не делайте следующего:

```
<audio controls>
  <source src="witchitalineman.ogg" type="audio/ogg">
  <source src="witchitalineman.mp3" type="audio/mpeg">
  <p>I am a lineman for the county...</p>
</audio>
```

В данном случае содержимое*<p>* отобразится только если браузер посетителя не поддерживает элемент *audio*. Это не будет полезно для глухого пользователя с современным браузером.

```
<audio controls>
  <source src="witchitalineman.ogg" type="audio/ogg">
  <source src="witchitalineman.mp3" type="audio/mpeg">
</audio>
<p>I am a lineman for the county...</p>
```

Видео

Если встроенная поддержка аудио впечатляюща, то перспективы на возможность проигрывания видео так вообще вызывают дикий восторг. С распространением доступности широкополосных соединений, видео стало очень популярным в интернете. В данных момент для его отображения в основном используется Flash — HTML5 планирует его сменить.

Элемент *video* работает по схожей схеме с *audio*: те же параметры *autoplay*, *loop*, *preload*; такая же система с атрибутом *src* или несколькими вложенными элементами *source*; точно так же можно добавить стандартный интерфейс управления при помощи *controls*, или сделать свой собственный.

Главная разница между аудио и видео заключается в том, что последнее обычно занимает немного больше места на странице, так что вы наверняка захотите указать точные размеры:

```
<video src="movie.mp4" controls width="360" height="240">
</video>
```

Что еще можно интересного сделать — это добавить превьюшку в виде картинки, используя параметр *poster*:

```
<video src="movie.mp4" controls width="360" height="240"
poster="placeholder.jpg">
</video>
```

Превью отображается до того, как видео начало проигрываться.

Не самые приятные новости в том, что борьба форматов в среде видео еще более жесткая, чем среди аудио. Главные игроки: MP4 (обременен патентом) и Theora Video (свободен и чист). Но вы уже знаете, как такие проблемы решаются:

```
<videocontrolswidth="360" height="240" poster="placeholder.jpg">
  <source src="movie.ogv" type="video/ogg">
  <source src="movie.mp4" type="video/mp4">
  <object type="application/x-shockwave-flash" width="360" height="240"
data="player.swf?file=movie.mp4">
        <param name="movie" value="player.swf?file=movie.mp4">
        <a href="movie.mp4">Download the movie</a>
  </object>
</video>
```

Все, что нам остается, — это дождаться, когда разработчики браузеров сойдутся на одном стандарте, чтобы все начало работать так, как задумано разработчиками спецификации — без лишнего клонирования файлов в разных форматах.

Модно, стильно, нативно

Возможность вставлять видео стандартными средствами языка разметки может быть самым потрясающим нововведением после создания элемента *img*. Даже такие крупные игроки как Google не стесняются проявлять энтузиазм по этому поводу. Вот, например, что им задумано для нового YouTube-а: www.youtube.com/html5

Главной проблемой в использовании плагинов для работы с мультимедия всегда была их изолированность от остальной части документа. Теперь же, когда все эти элементы — часть общей системы, они легко доступны для скриптов и стилей.

Элемент video встречает CSS3. Попробуйте сделать такое с плагином.

Поддержка аудио и видео как часть HTML5 получила наши одобрение и восторг. Но ведь мы знаем, что всемирная сеть — не столько презентационная среда, она еще и интерактивная. А самым старым, но бессменно мощным, инструментом достижения интерактивности на веб-страницах всегда были формы.

Пользовательские элементы управления

Варианты элемента <input> с различными значениями атрибута type:

Число из интервала:

Целое число:
10

Дата и время:

Дата:
02.мм.гггг

Время:
--:--

Месяц:
-------- ----

Неделя:
Неделя --, ----

Цвет:

url:
domain.uz

email:
@domain.uz

Подтвердить

Если значение адреса не соответствует формату *url*, то после нажатия [подтвердить] браузер предпримет попытку его исправить. Если он этого

сделать не сможет, то выведет сообщение об ошибке. Если значение электронной почты не соответствует формату *email*, то после нажатия [подтвердить] браузер либо исправит адрес, либо сообщит об ошибке ввода.

```
<form>
Число из интервала:<br/><input type='range' min='0' max='100' step='10' value='10'/><br/>
Целое число:<br/><input type='number' min='-100' max='100' step='1' value='10'/><br/>
Дата и время:<br/><input type='datetime'/><br/>
Дата:<br/><input type='date'/><br/>
Время:<br/><input type='time'/><br/>
Месяц:<br/><input type='month'/><br/>
Неделя:<br/><input type='week'/><br/>
Цвет:<br/><input type='color'/><br/>
url:<br/><input type='url' value='domain.ru'/><br/>
email:<br/><input type='email' value='@domain.ru'/><br/><br/>
<input type='submit' value='Подтвердить'/>
</form>
<div><meter min='-50' low='-10' high='30' max='50' value='-15' title='градусы'></meter></div>
<div><meter min='-50' low='-10' high='30' max='50' value='20' optimum='20' title='градусы'></meter></div>
<div><meter min='-50' low='-10' high='30' max='50' value='35' title='градусы'></meter></div>
<div><progress max='100' value='70' title='%'></progress></div>
```

РАЗДЕЛ 3. ОСНОВЫ CSS

Глава 10. Основы CSS (Cascading Style Sheets).

Что такое CSS?

Как вы знаете, при создании сайта мы формируем его содержимое при помощи языка гипертекстовой разметки - HTML(HypretextMarkupLanguage). С помощью него мы создаем навигационные блоки, наполняем веб-страницу текстовым, аудио/видео-контентом. В общем, создаем структуру сайта.

После формирования содержания сайта, на следующем этапе, мы переходим на стадию оформления его внешнего вида, дабы он был красивым и приятным. А также чтобы он хорошо смотрелся на разных разрешениях дисплеев. Как раз для этого мы используем каскадные таблицы стилей CSS.

Значит, для изучения и эффективного использования CSS необходимо знать основы HTML. Без этого постижение каскадных таблиц стилей не имеет практического смысла.

Наглядный пример

Образно говоря, создание сайтов можно сравнить с рисованием. Перед тем как мы начнем ловко управляться кисточкой и холстом, нам стоит определиться с тем, что именно будет изображено на нашей картине. Представим, что мы решили нарисовать пейзаж, на котором будет изображена птица на фоне деревьев и гор. Можно сказать, что на этом этапе мы определили само содержимое картины.

Содержание будущей картины

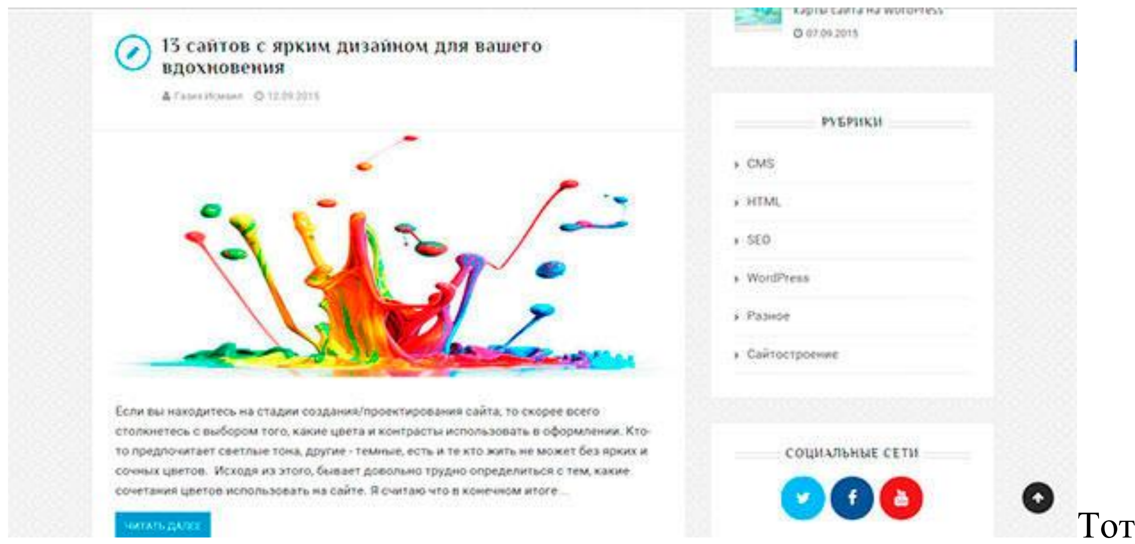
Следующим шагом мы решаем, каких цветов и размеров будут деревья, горы, птицы и также то, как они будут располагаться относительно друг друга.

Итоговая картина

Подобная ситуация обстоит с HTML, при помощи которого мы формируем содержание веб-страницы.

Сайт на голом HTML, без CSS

Далее, при помощи CSS мы определяем цвета, размеры и расположение элементов на веб-странице, то есть занимаемся внешним оформлением сайта.



Тот же сайт с подключенными таблицами стилей

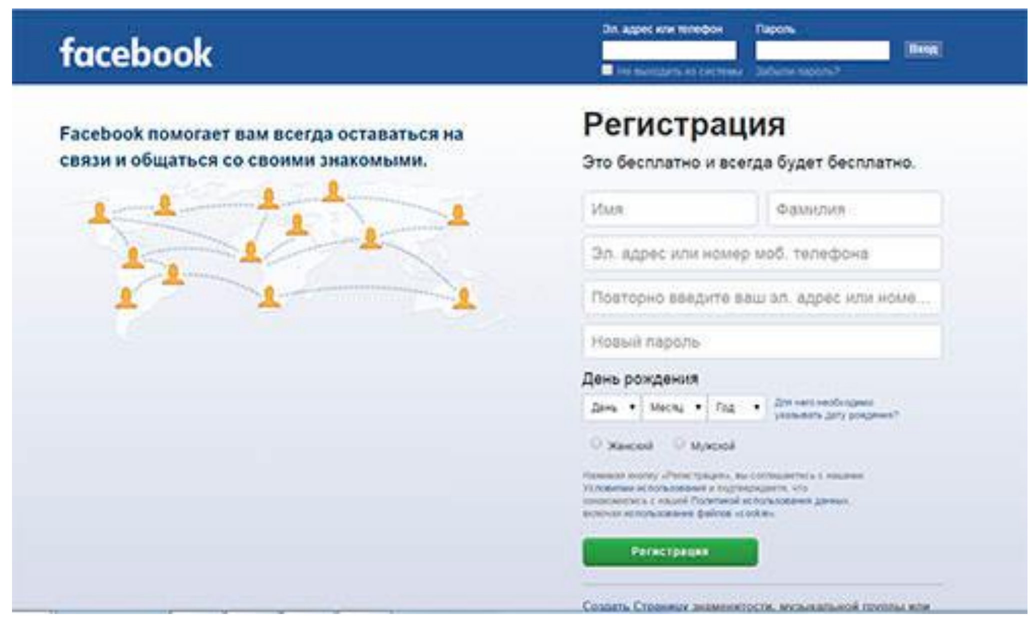
Для большей наглядности, давайте перейдем на какой-нибудь веб-сайт, например, facebook.com. Следующим шагом нам потребуется установить расширение для браузера под названием WEbDeveloper. Вам будет достаточно ввести в поисковике фразу WEbDeveloper, перейти по ссылке и в открывшемся окне кликнуть по кнопке "Установить".

У меня это расширение установлено и управляется при помощи иконки с шестеренкой в правой верхней части экрана. Предлагаем на сайте faceboook.com отключить таблицы стилей и посмотреть только на ее содержимое. Для этого мы переходим в раздел CSS и нажимаем на DisableAllStyles. Стили отключаются и мы видим как невзрачно выглядит содержимое данной веб-страницы без оформления.

Внешний вид социальной сети без подключенных CSS файлов

То есть сейчас мы видим сайт на голом HTML. Чтобы включить css файлы этой страницы мы возвращаемся к WEbDeveloper и снимаем галочку с DisableAllStyles.

Таблица стилей снова подключилась и мы видим содержимое веб-страницы с оформлением.



Соц. сеть с подключенными CSS файлами

Для начала давайте разберемся, что же такое CSS. Все знают что **HTML** - это язык разметки гипертекста. А **CSS** в свою очередь **язык визуального оформления** этой самой разметки. **CSS – это CascadingStyleSheets — каскадные таблицы стилей**. В HTML существуют свои инструменты визуального оформления, но они во многом уступают возможностям CSS, да и если у вас много страничек, то представьте себе, вам вдруг захотелось поменять цвет всех заголовков, вам придется перелопачивать все HTML файлы, а на css это делается в одной строке. Хватит только этого аргумента, для того чтобы начать изучать CSS.

CSS кстати используется не только для визуального оформления страниц размеченных на HTML, но и тех страниц написанных с помощью языка разметки XHTML, также отлично подходит для оформления XML документов.

Так как мы с вами только новички то и начнем с самых основ, в дальнейшем будем уже углубляться в эту технологию, так как CSS это довольно мощная штука и в рамках одной статьи все просто не уместить, поэтому разобьем изучение CSS на несколько уроков, с постепенным углублением в эти каскадные таблицы стилей.

Ну что переходим к практике, и начнем мы с вами с простого синтаксиса.

Выглядит он примерно так:

```
 Элемент оформления{
свойство: значение;
}
```

Элемент оформления принято называть селектор.

Чтобы был понятен синтаксис, приведем простейший пример:

```
p {
  color: red;
}
```

где,

p – это тег абзаца;

color – это свойство т.е. в нашем случае это цвет;

red – это значение, в нашем случае это красный.

И в итоге у вас текст во всех абзацах станет красным цветом.

С синтаксисом немного разобрались, перейдем к тому моменту, где это все писать, т.е. **подключать CSS к нашему HTML документу**.

Существует несколько способов:

1. Непосредственно в самом элементе документа **по средствам атрибута style**, например, если перевести выше указанный пример в сам документ и описать его в самом элементе, то это будет выглядеть так:

```
<p style = "color: red;">Пример</p>
```

Данный способ не удобен тем, что опять же если у вас будет много документов, то вам придется менять во всех этих документах, причем в каждом элементе.

2. Немного по удобней способ это применения css когда таблица стилей описана в самом документе. В разделе head применяется элемент style, в котором и описывается сама таблица стилей. Например, снова задействуем выше указанный пример, но уже с использованием этого способа:

```
<head>
<style type="text/css">
p {
  color: red;
}
</style>
</head>
<body>
<p >Пример</p>
</body>
```

Получится тоже самое, что и было выше. Этот способ иногда используют, но он все равно не идеален.

3. В третьем способе используется **импорт css документа**, редко встречается применение этого способа, может им и никто и не пользуется, но знать о нем нужно. Здесь таблица стилей здесь уже описана в отдельном документе.

```
<head>
<style type="text/css" media="all">
    @import url(css-file.css);
</style>
</head>
<body>
```

```
<p>Пример</p>
</body>
```

где,

css-file.css – это файл css где и описана таблица стилей, если он лежит в другой папке то нужно писать путь к этому файлу.

4. Самый распространенный способ применений каскадной таблицы стилей это **подключение css файла** к вашему документу. Этим способом пользуются практически все, так как он самый удобный. Подключается css по средствам тега link в элементе head.

```
<head>
<link rel="stylesheet" type="text/css" href="/css-file.css">
</head>
<body>
<p>Пример</p>
</body>
```

где,

css-file.css – это файл в котором описана таблица стилей, снова если файл лежит в папке отличной от расположения самого документа то необходимо писать путь к нему.

Так с подключением css разобрались теперь перейдем к селекторам, т.е. способам поиска этих самых элементов в вашем документе, другими словами, если вам не нужно, что все абзацы были красным цветом как в выше указанном примере, а например, у одних абзацев он был черным, а у других он был красный. На самом деле их девять видов, но мы пока рассмотрим самые популярные.

С **селекторами элементов** (тегов) мы уже познакомились выше перечисленные примеры, применялись ко всем тегам:

```
p {
  color: red;
```

```
}
```

p – это и есть селектор элемента, здесь могут быть практически любые теги HTML документа, такие как body, div, table, tr, td, h1 и много, много других.

На мой взгляд, самые популярные виды селекторов это с использованием **классов и идентификаторов**. Другими словами при разметке своего документа вы помечаете элементы документа с помощью идентификаторов (которые являются уникальными во всем документе) или приводите их к одному классу (он может повторяться в документе). Что было понятней, приведем пример, в котором у нас будет два абзаца, один из которых будет красным цветом, а другой черным.

Текст css будет таким:

```
#idred {
  color: red;
}
.black {
  color: black;
}
```

Текст документа будет таким:

```
<pid = "idred">Пример красного абзаца </p>
<pclass = "black">Пример черного абзаца </p>
```

Для практики подключите сами любым из способов каскадную таблицу стилей к вашему документу.

Наверное, вы уже поняли, что идентификаторы в css обозначаются с помощью символа # (#idred) а классы с помощью точки и названия класса (.black).

В HTML документе они обозначаются по средствам соответствующих тегов: id для идентификатора и class для классов.

Глава 11. Селекторы и их правила. Групповой селектор

Основная функция селекторов заключается в том, чтобы контролировать дизайн веб страниц, выбирая элемент, или элементы по определенным критериям и стилизуя их с помощью CSS свойств, которые вы указываете в блоке объявлений (описаний).

Селектор типа

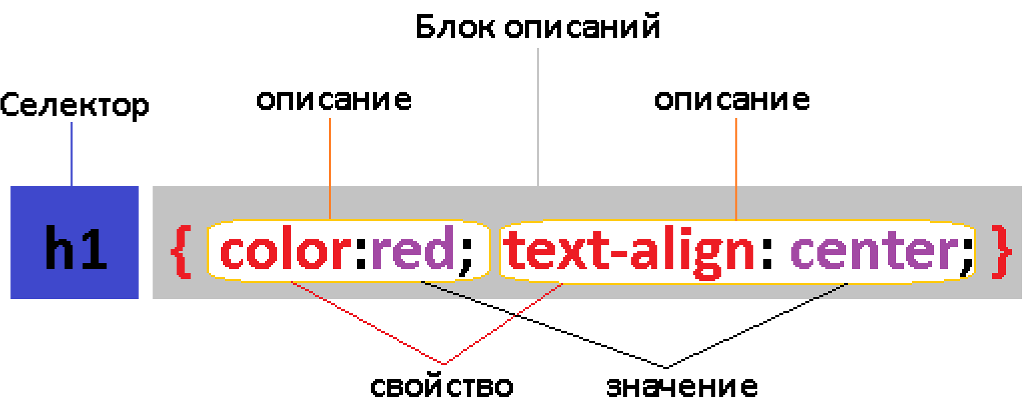


Рис. 3а Селектор типа в CSS.

В предыдущих примерах, да и в практическом задании предыдущей статьи "**Создание первой таблицы стилей**" мы с Вами использовали селекторы типа, которые определяют HTML элемент, к которому будет применен стиль. Если вы хорошо знакомы с HTML тегами, то вам будет легко определять в стилях селекторы типа, так как они имеют одноимённые наименования с форматируемыми элементами, например:

- **p** - сообщает браузеру, что необходимо отформатировать все HTML теги <p> (параграф).
- **table** - сообщает браузеру, что необходимо отформатировать все HTML теги <table> (таблица).
- **li** - сообщает браузеру, что необходимо отформатировать все HTML теги <li> (элемент списка).

Давайте рассмотрим пример, в котором с использованием селектора типа мы стилизуем все заголовки второго уровня (<h2>):

```
<!DOCTYPE html>
<html>
<head>
  <metacharset = "UTF-8">
  <title>Селектортипа</title>
<style>
h2{
font-family: Arial; /* задаемтипшрифта Arial */
color: green; /* задаем цвет текста*/
margin-top: 50px; /* добавляем отступ от верхнего края элемента */
}
</style>
</head>
<body>
  <h2>Обычный заголовок второго уровня</h2>
  <h2>Обычный заголовок второго уровня</h2>
  <h2>Обычный заголовок второго уровня</h2>
</body>
</html>
```

Результат нашего примера:

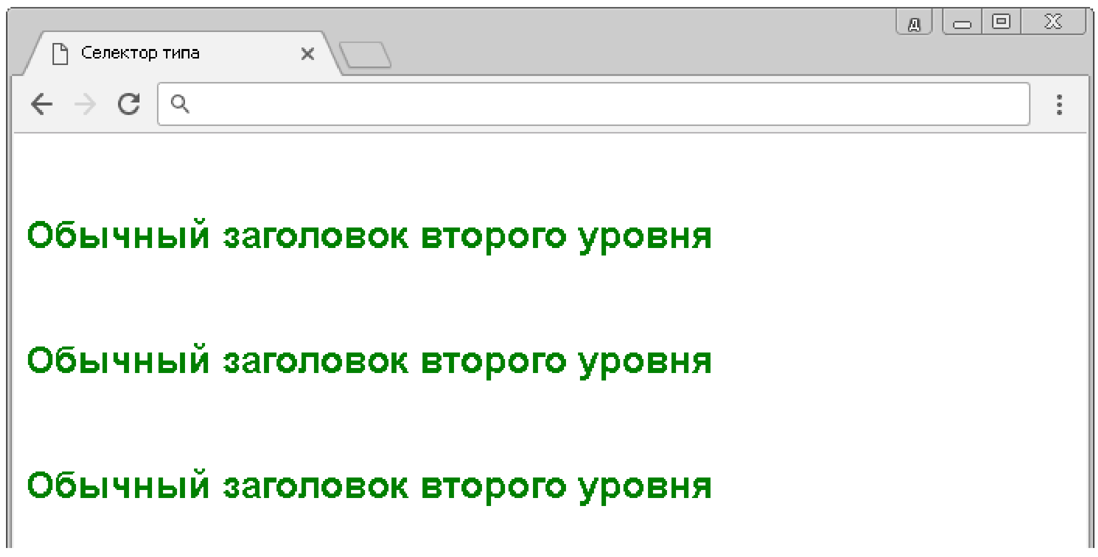


Рис. 36 Пример использования селектора типа.

Как вы могли заметить, селектор типа позволяет стилизовать сразу все элементы подобного типа, но что делать если нам необходимо стилизовать какой-то однотипный элемет отлично от других? В этом случае селектор типа нам уже не подойдет и нам придется использовать **селектор класса.**

Селектор класса

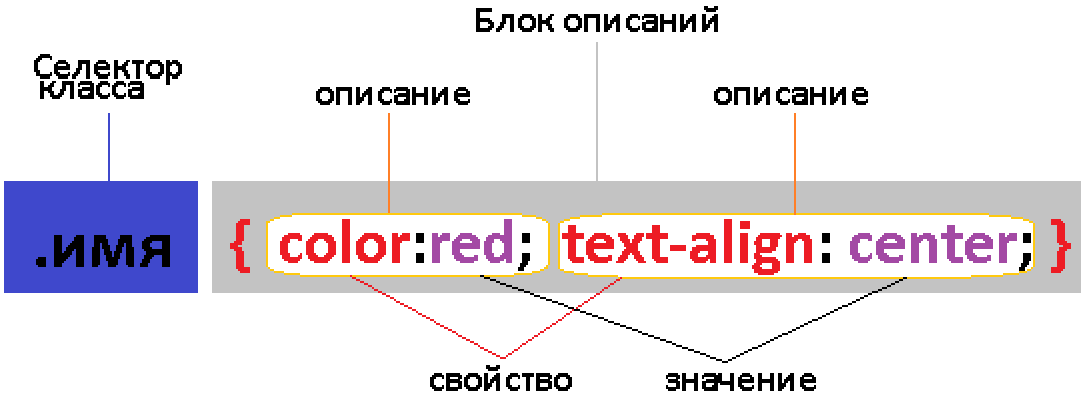


Рис. 4 Селектор класса в CSS.

Прошу от Вас максимального внимания, так как селектор класса является одним из самых распространенных и востребованных селекторов CSS, который мы будем постоянно использовать в дальнейшем изучении CSS.

И так, мы хотим, чтобы один из элементов выглядел не так, как другие. Для этого нам необходимо создать селектор, назначив ему имя, которое нам необходимо придумать самостоятельно:

```
.test{/* имя класса в таблице стилей задается через точку */
text-align: center; /* горизонтальное выравнивание текста по центру */
font-family: Courier; /* задаемтипшрифта Courier */
color: green; /* задаем цвет текста*/
margin-top: 50px; /* добавляем отступ от верхнего края элемента */
```

```
}
```

После того как мы создали наш класс в таблице стилей, нам необходимо применить его к интересующему нас элементу, или элементам, которые мы хотим стилизовать. Чтобы указать класс для определенного элемента, необходимо добавить к этому элементу глобальный HTML атрибут **class** со значением нашего класса, но без точки:

```
<h2 class = "test">Заголовок второго уровня</h2>/* задаем класс для элемента*/
```

Обратите внимание на следующие правила, которые необходимо соблюдать при работе с селекторами класса:

- в отличии от селекторов типа все названия селекторов класса должны начинаться с точки (благодаря ей браузеры находят эти селекторы в таблице стилей). Точка требуется только в названии селектора таблицы стилей (в значении глобального HTML атрибута **class** она не ставится, **будьте внимательны из-за этого возникает много ошибок**).
- используйте только буквы алфавита (A-Z, a-z), числа, дефисы, знаки подчеркивания.
- название после точки всегда должно начинаться с символа (неправильно: **.50cent**, **.-vottakvot**).
- Учитывайте регистр при наименовании стилевых классов, т.к. они к этому чувствительны и очень ранимы (**.vottakvot** и **.VotTakVot** разные классы).

Теперь соберем это в одном примере и посмотрим результат:

```
<!DOCTYPE html>
<html>
<head>
  <metacharset = "UTF-8">
  <title>Селекторы класса</title>
<style>
.test{
```

```
text-align: center; /* горизонтальное выравнивание текста по центру */
font-family: Courier; /* задаемтипшрифта Courier */
color: green; /* задаем цвет текста*/
margin-top: 50px; /* добавляем отступ от верхнего края элемента */
}
</style>
</head>
<body>
  <h2>Обычный заголовок второго уровня</h2>
  <h2class = "test">Заголовок с заданным классом</h2>
</body>
</html>
```

Результат нашего примера:

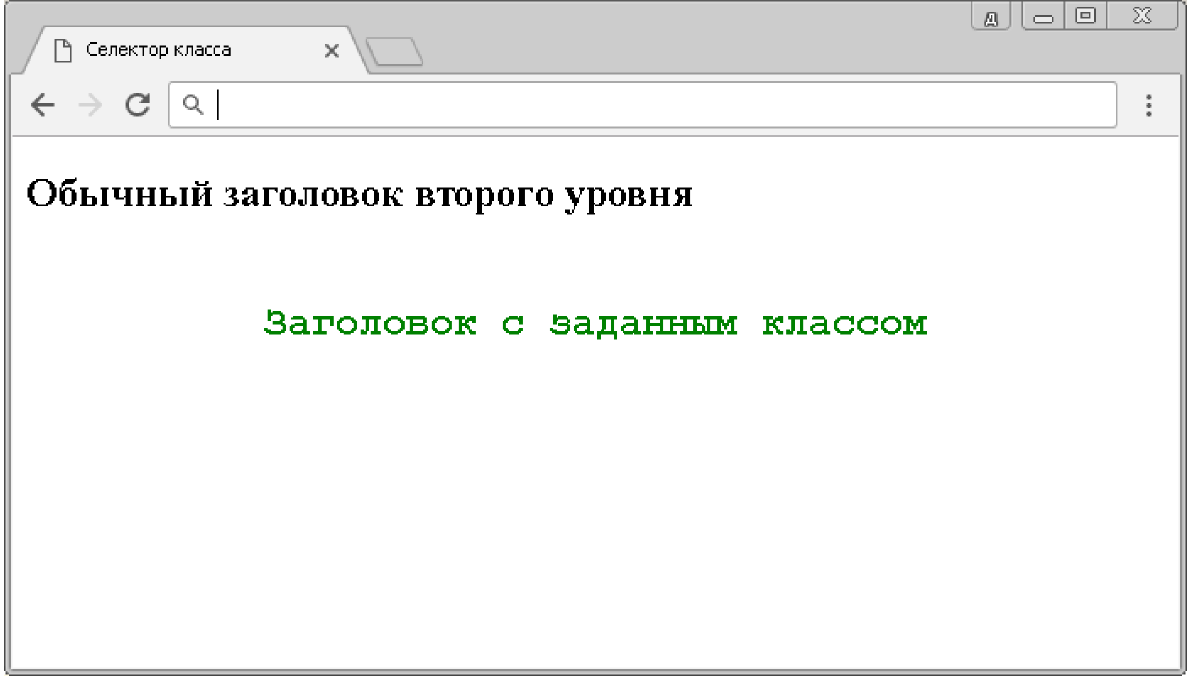


Рис. 5 Использование селектора класса.

Селекторы класса очень гибкий инструмент, который позволяет указать конкретный стиль даже к одному слову предложения. Для этого мы поместим это слово внутри элемента <span> и назначим этому элементу определённый класс, который опишем во внутренней таблице стилей:

```
<!DOCTYPE html>
<html>
<head>
  <metacharset = "UTF-8">
  <title>Пример выделения одного слова</title>
<style>
.test{
color: #0F0; /* задаём цвет текста в hex формате */
font-size: 30px; /* задаем размер шрифта*/
}
</style>
</head>
<body>
  <h2>Обычный заголовок <spanclass = "test">второго</span>
уровня</h2>
</body>
</html>
```

Результат нашего примера:

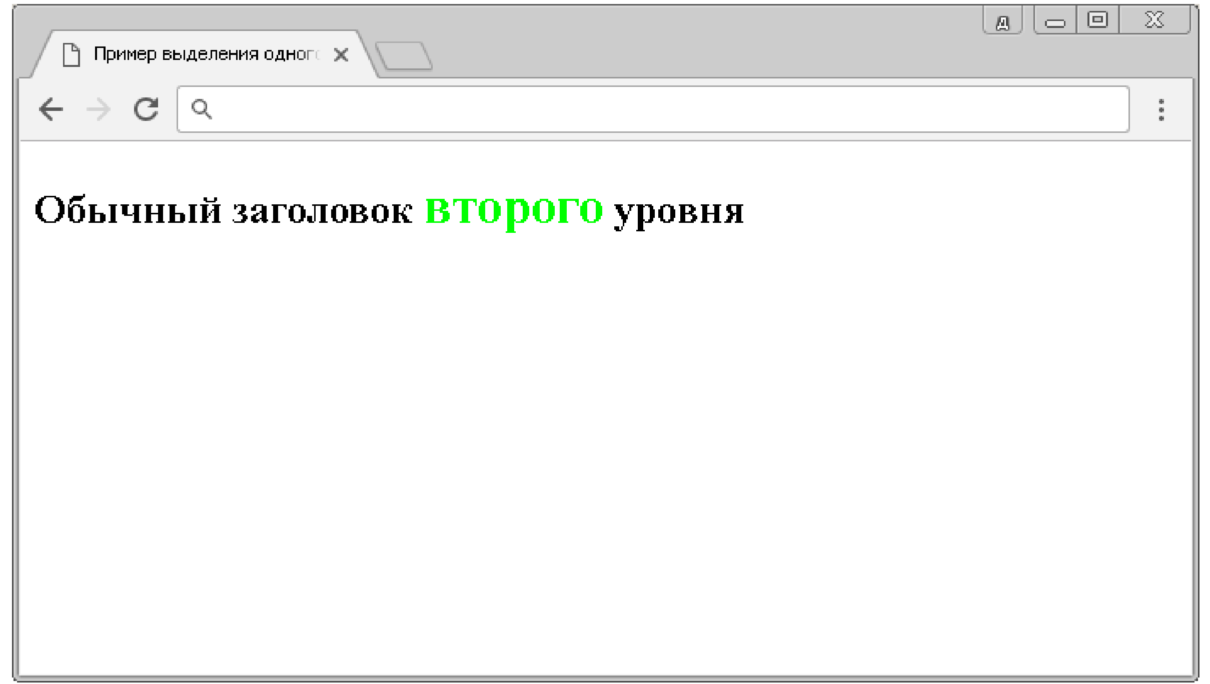


Рис. 6 Пример выделения одного слова с использованием селектора класса.

Обращаю Ваше внимание, что вы можете использовать селекторы класса напрямую к HTML элементам, используя следующий синтаксис:

```
a.test{/* выбирает все элементы <a> с классом test */
блок объявлений;
}
p.intro{/* выбирает все элементы <p> с классом intro */
блок объявлений;
}
```

ID селекторы

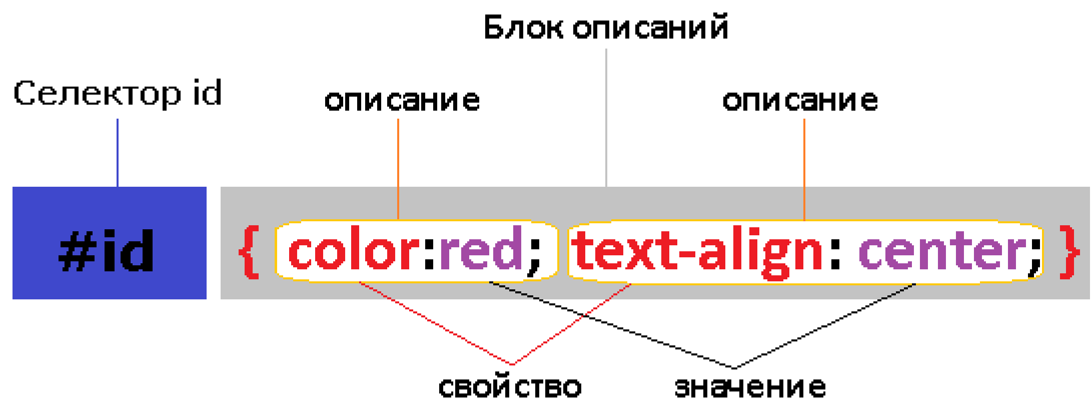


Рис. 7 Селектор id в CSS.

Основная задача селектора **id** заключается в том, чтобы выбрать элемент с определённым идентификатором. Для начала необходимо создать идентификатор, назначив ему имя, которое нам необходимо придумать самостоятельно:

```
#test{/* имя идентификатора в таблице стилей задается через решетку */
background-color: #00FF00; /*Задаём цвет заднего фона*/
color: white; /*Задаём цвет шрифта белый*/
font-size: 30px; /*Указываем размер шрифта*/
width: 600px; /*Указываем ширину элемента*/
height: 40px; /*Указываем высоту элемента*/
```

```
    text-align: center; /*Выравниваем текст по центру внутри элемента*/
    }
```

После того как мы создали наш идентификатор в таблице стилей, нам необходимо применить его к интересующему нас элементу, который мы хотим стилизовать. Чтобы указать идентификатор для элемента, необходимо добавить к этому элементу глобальный HTML атрибут id со значением нашего идентификатора, но уже без решетки:

```
<h2 id = "test">Обычный заголовок второго уровня</h2><!-- задаем идентификатор для элемента -->
```

Обратите внимание на следующие правила, которые необходимо соблюдать при работе с id селекторами:

- значение идентификатора должно быть уникально на странице, размещение нескольких одноименных идентификаторов на странице считается ошибкой (выборка остановится на первом идентифакторе).
- все названия **id** селекторов должны начинаться с решётки (благодаря ей браузеры находят эти селекторы в таблице стилей). Решетка требуется только в названии селектора таблицы стилей (в значении глобального HTML атрибута id она не ставится, **будьте внимательны из-за этого возникает много ошибок**).
- используйте только буквы алфавита (A-Z, a-z), числа, дефисы, знаки подчеркивания.
- название после решётки всегда должно начинаться с символа (неправильно: **#50cent**, **#-vottakvot**).
- Учитывайте регистр при наименовании id селекторов, т.к. они к этому чувствительны (**#vottakvot** и **#VotTakVot** разные идентификаторы).

Теперь соберем это в одном примере и посмотрим результат:

```
<!DOCTYPE html>
<html>
<head>
```

```
  <metacharset = "UTF-8">
  <title>ID селекторы</title>
<style>
#test{
background-color: #0F0; /* задаёмцветзаднегофона */
color: white; /* задаём цвет текста */
font-size: 30px; /* указываем размер шрифта */
height: 100px; /* указываем высоту элемента */
text-align: center; /* горизонтальное выравнивание текста по центру
*/
}
</style>
</head>
<body>
  <h2 id = "test">Обычный заголовок второго уровня</h2>
</body>
</html>
```

Результат нашего примера:

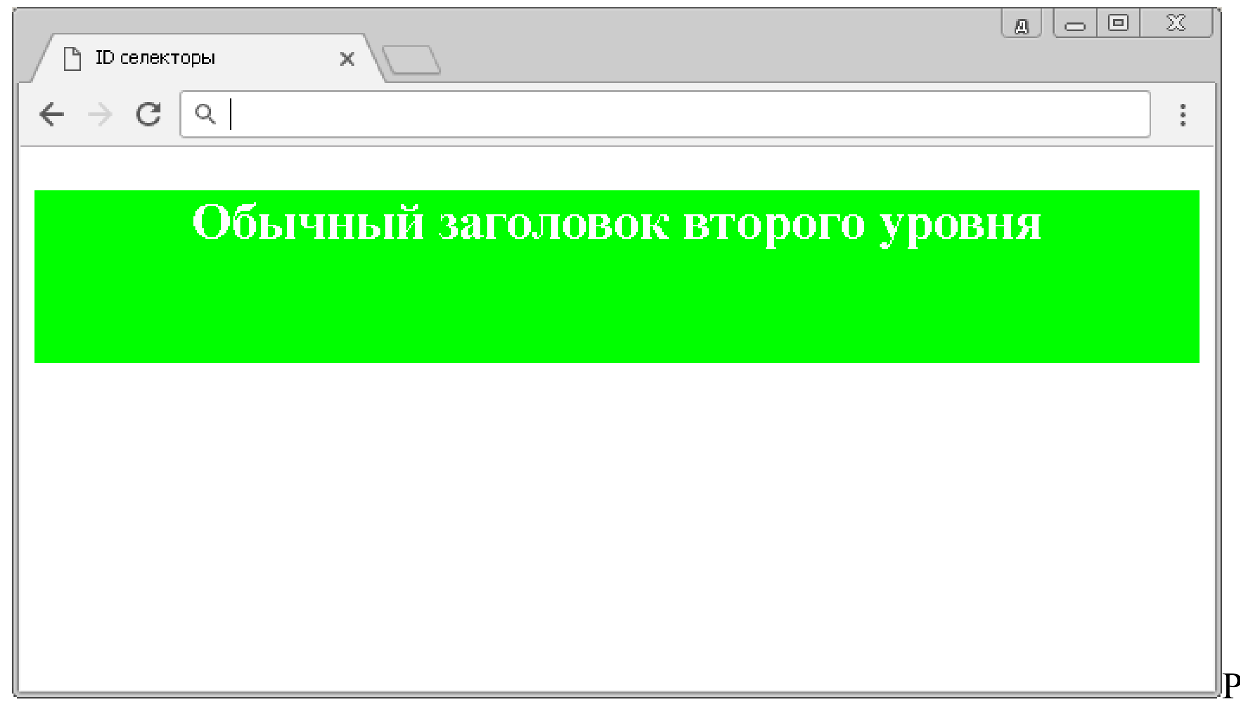


Рис. 7а Пример использования id селектора.

В настоящее время **id** селекторы в основном применяются на веб страницах с целью динамического управления элементами с использованием языка программирования **JavaScript**.

Групповые селекторы

В CSS допускается задать единый стиль не только одному селектору, но и группе селекторов. В группу могут входить как *селекторы типа*, *селекторы класса*, так и рассмотренные нами *id селекторы*. При перечислении любых селекторов (даже, которые мы будем рассматривать далее в учебнике) необходимо **обязательно разделять их запятой**, позднее вы поймете почему.

Например:

```
<style>
h1, h2,.test,#test{/* выбираем заголовки первого и второго уровня, элементы с классом test и элемент с идентификатором test */
background-color: green;  /* задаем цвет заднего фона */
}
.test, #test{/* выбираем элементы с классом test и элемент с идентификатором test */
color: blue; /* задаем цвет текста */
}
</style>
```

Первым групповым селектором мы указываем, что заголовки <h1> и <h2>, элементы с классом **test** и элемент с идентификатором **test** получат цвет заднего фона *зеленый*.

Вторым групповым селектором мы указываем, что кроме заднего фона элементы с классами **test** и элемент с идентификатором **test** получат *синий* цвет текста.

Рассмотрим следующий пример:

```
<!DOCTYPEhtml>
<html>
<head>
  <metacharset = "UTF-8">
  <title>Групповые селекторы</title>
<style>
h1, h2,.test,#test{/* выбираем заголовки первого и второго уровня, элементы с классом test и элемент с идентификатором test */
color: red;  /* задаем цвет текста */
}
h3,h4{/* выбираем заголовки третьего и четвертого уровня */
color: blue; /* задаем цвет текста */
}
h1,h2,h3,h4{/* выбираем заголовки от первого до четвертого уровня */
font-style: italic;  /* задаем стиль шрифта - курсивное начертание */
}
</style>
</head>
<body>
  <h1>Заголовок первого уровня</h1>
  <h2>Заголовок второго уровня</h2>
  <p class = "test">Абзацсклассом test</p>
  <h3>Заголовок третьего уровня</h3>
  <pid = "test">Абзац с идентификатором test</p>
  <h4>Заголовок четвертого уровня</h4>
  </body>
</html>
```

В этом примере мы использовали три групповых селектора:

- *Первым* групповым селектором мы указываем, что заголовки <h1> и <h2>, элементы с классом **test** и элемент с идентификатором **test** получат цвет текста *красный*.
- *Вторым* групповым селектором мы указываем, что заголовки <h3> и <h4> получат цвет текста *синий*.
- *Третьим* групповым селектором мы указываем, что все заголовки (от <h1> до <h4>) будут иметь *курсивное начертание шрифта*.

Результат нашего примера:

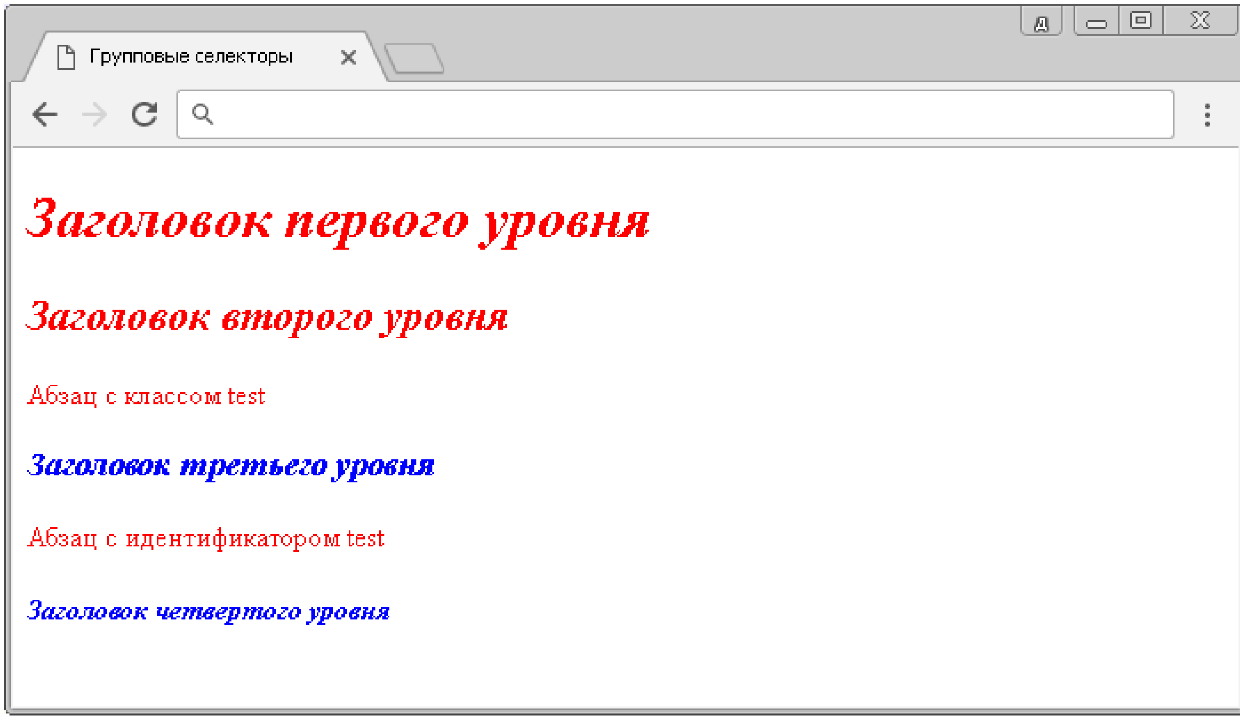


Рис. 76 Пример использования групповых селекторов.

Универсальный селектор

Селектор * позволяет выбрать все элементы внутри документа (универсальный селектор).

Давайте рассмотрим пример использования:

```
<!DOCTYPEhtml>
<html>
<head>
  <metacharset = "UTF-8">
  <title>Универсальный селектор</title>
<style>
```

```
*{/* выбираем все элементы внутри документа */
color: rgb(50,100,150); /* задаем цвет текста в формате rgb */
font-style: italic;  /* задаем стиль шрифта - курсивное начертание */
}
</style>
</head>
<body>
  <h1>Заголовок первого уровня</h1>
  <h2>Заголовок второго уровня</h2>
  <p>Абзац, который не несет никакой смысловой нагрузки</p>
</body>
</html>
```

В этом примере с использованием универсального селектора мы указали, что все элементы на странице получат определенный цвет текста и будут иметь курсивное начертание шрифта.

Результат нашего примера:

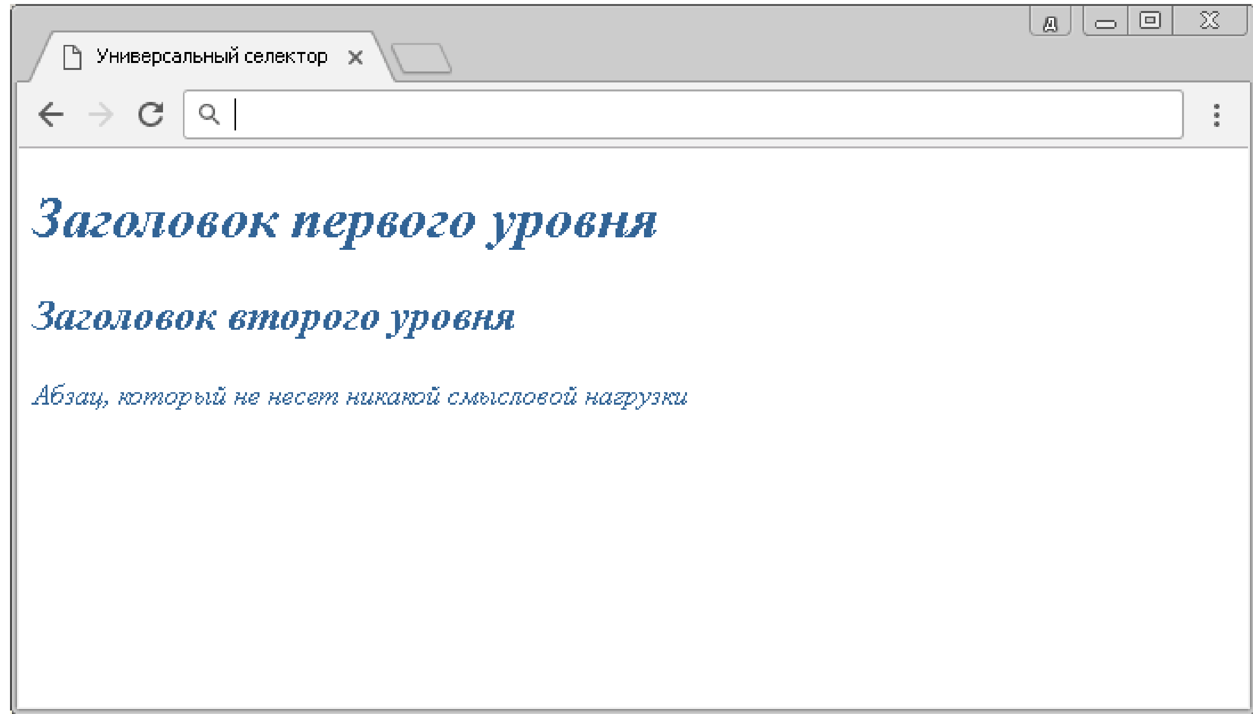


Рис. 7в Пример использования универсального селектора.

Кроме того универсальный селектор может использоваться в качестве селектора потомков и выбирать все элементы, которые находятся внутри другого элемента.

```
.test *{/* выбирает все элементы внутри элемента с назначенным классом test */
объявление;
}
#test *{/* выбирает все элементы внутри элемента с назначенным идентификатором test */
объявление;
}
div *{/* выбирает все элементы внутри элемента <div> */
объявление;
}
```

Глава 12 Селекторы потомков.

В этой главе мы с Вами углубим знания о селекторах CSS, особое внимание мы уделим селекторам потомков, ведь понимание как работают эти селекторы является очень важной темой, требующей обязательного изучения. Кроме того, мы с Вами разберемся в родственных связях HTML элементов, узнаем какие элементы являются родительскими, дочерними и даже сестринскими.

Многие термины, которые вы прочитаете в этой статье, будут использоваться в дальнейших статьях этого учебника, прошу Вас внимательно их изучить и понять. Понимание древовидной структуры документа Вам понадобится не только при изучении CSS, но и при изучении, например, клиентского JavaScript.

Древовидная структура HTML документа

Рассмотрим следующее изображение:

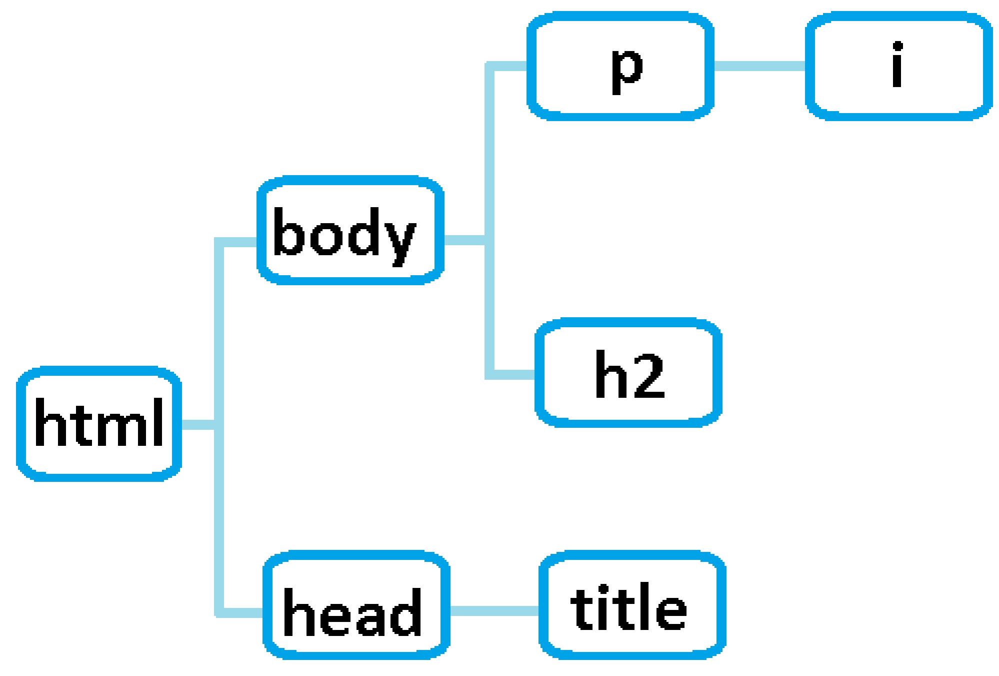


Рис. 8 Пример древовидной структуры HTML.

Любой HTML код имеет древовидную структуру, в которой первый используемый тег - <html> выступает **предком** (прародителем) всех остальных элементов, из него выходят такие элементы как <head> и <body>:

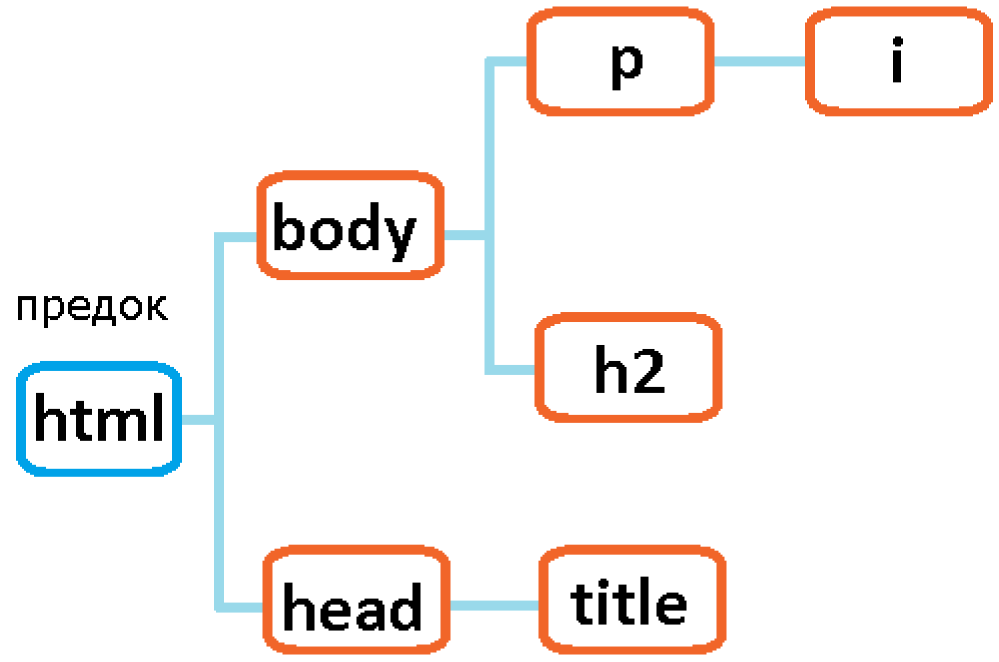


Рис. 8а Предок в HTML документе.

HTML элемент, расположенный внутри другого элемента является его **потомком**. Элемент <title> потомок <head>, а элемент <h2> и <p> на изображении потомки <body>.

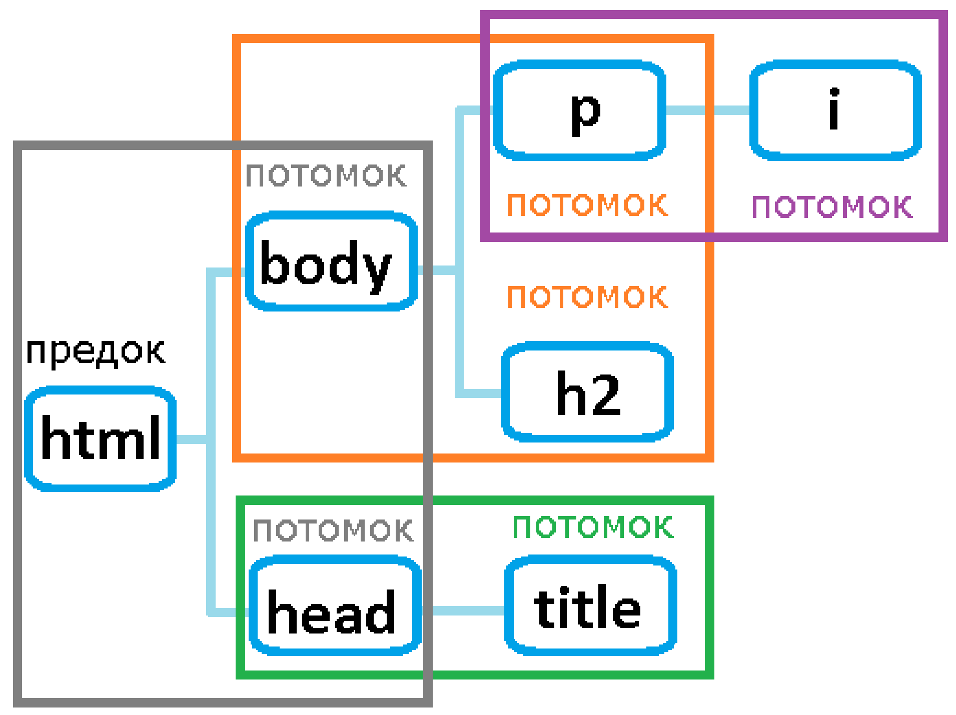


Рис. 8б Потомки в HTML документе.

Обратите внимание на важный момент, что элемент <i> является потомком одновременно для элемента <p>, <body> и для <html>.

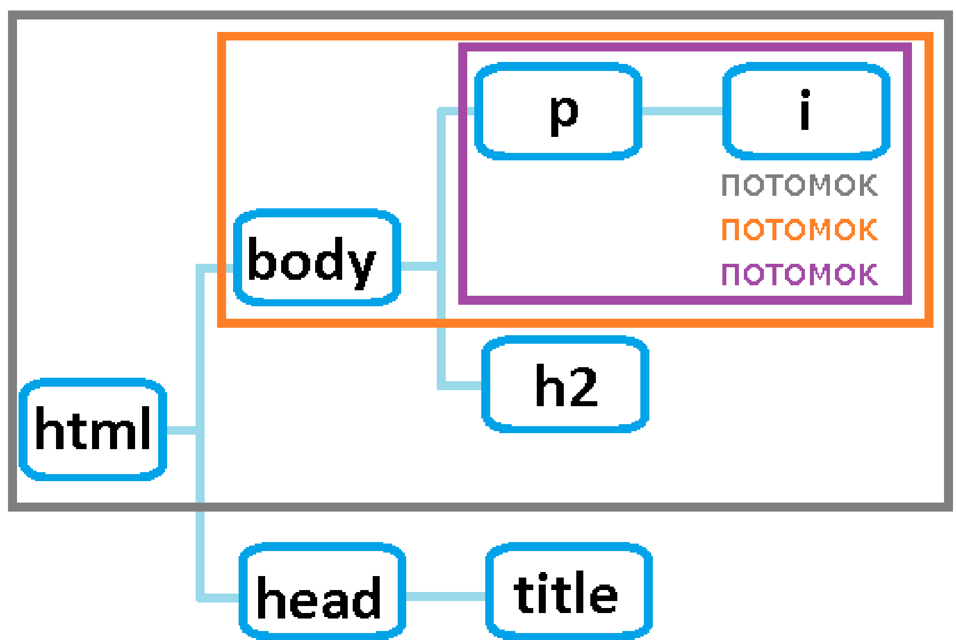


Рис. 8в Потомок для нескольких элементов.

Родительский элемент связан с другими элементами более низкого уровня и находится на один уровень выше их. Например, <html>элемент является родительский элементом для <head> и <body>, а элемент <p> является родительским для элемента <i>.

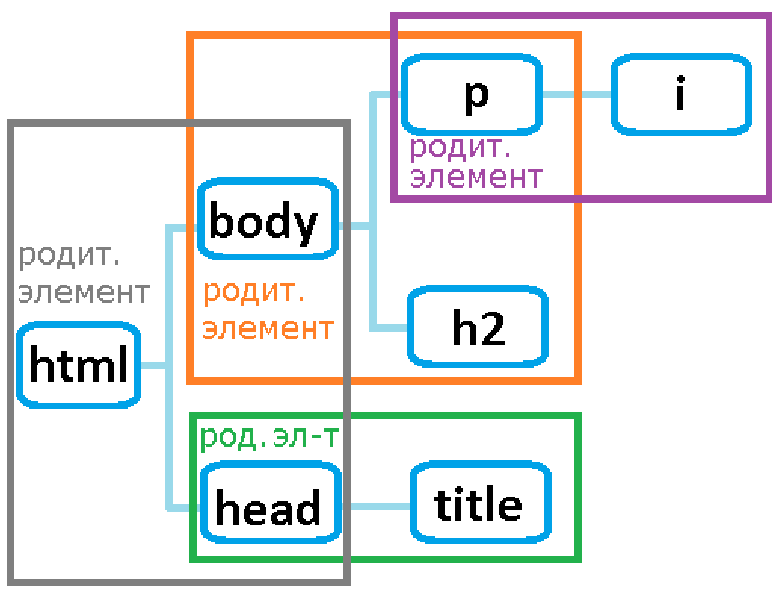


Рис. 8г Родительские элементы в HTML документе.

Элемент, подчиненный другому элементу более высокого уровня, является **дочерним**. На изображении ниже оба элемента <h2> и <p>являются **дочерними** по отношению к <body>, но элемент <i> при этом **не является дочерним** для элемента <body>, так как он расположен внутри тега <p>, и **является дочерним** именно для него.

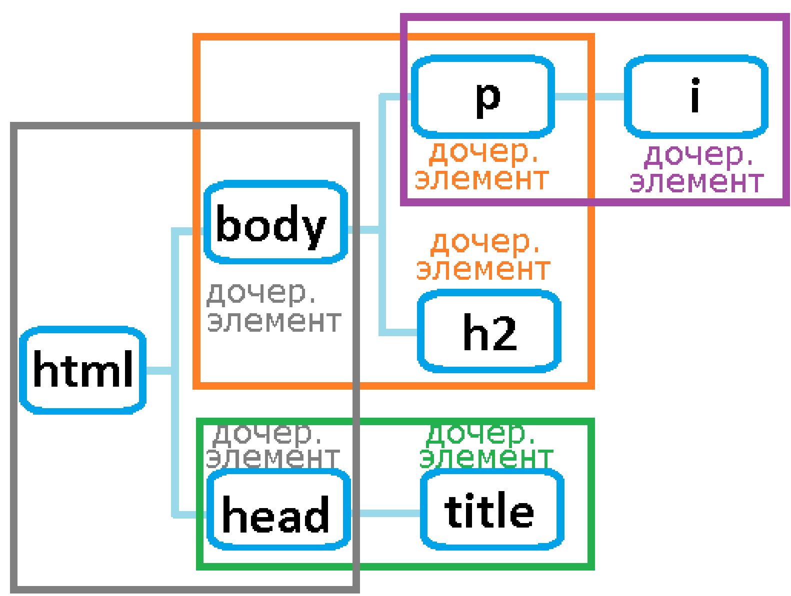


Рис. 8д Дочерние элементы в HTML документе.

Дочерние элементы для одного и того же родительского элемента, называются **элементами одного уровня** (соседними элементами, или **сестринскими элементами**). Например, такие элементы как [<head>] и [<body>], или [<h2>] и [<p>].

Еще раз акцентирую ваше внимание на том, что **сестринские элементы** расположены на одном уровне **в пределах одного родителя**. К примеру, элементы [<h2>] и [<title>] являются просто **элементами одного уровня, а не сестринскими элементами.**

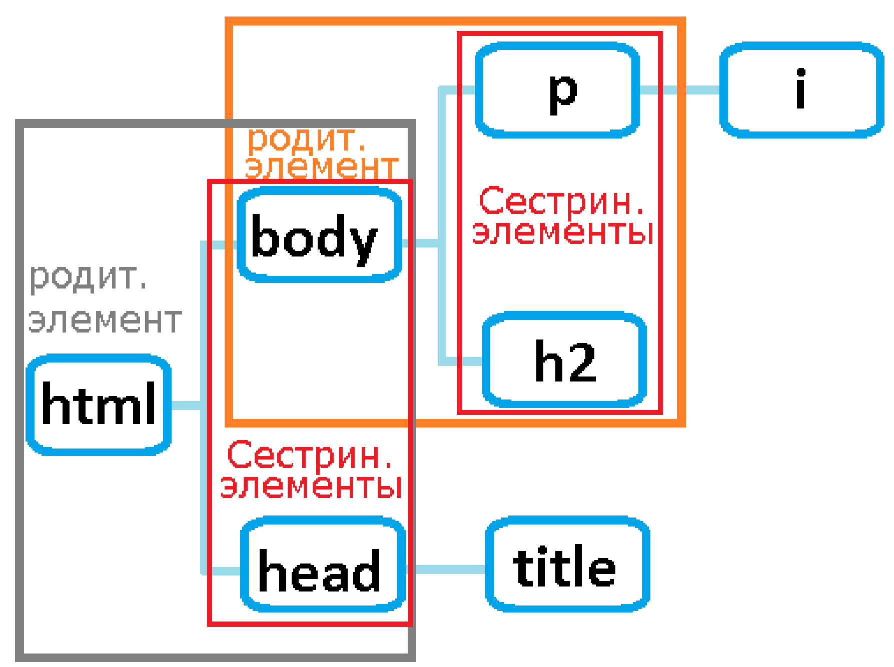


Рис. 8е Сестринские элементы в HTML документе.

И так, с родственными связями мы пока остановимся, у нас же тут не ток шоу с Андреем Малаховым, верно? Вернемся к селекторам потомков, которые, используя древовидную структуру HTML документа, позволяют форматировать вложенные элементы по нашему усмотрению.

Селекторы потомков

В конце предыдущей статьи, мы с Вами уже рассматривали примеры с **селекторами потомков**, их основное назначение заключается в том, чтобы

единообразно отформатировать элементы, которые расположены внутри других элементов (содержатся внутри других элементов).

Например, во всех заголовках <h2>, в которых находится текст, отражённый курсивом (элемент <i>), вы хотите выделить определённым цветом. Решение этого вопроса, надеюсь, для Вас выглядит вполне очевидно, и вы используете для этой задачи **селектор потомков**:

```
h2 i{
color: red;
}
```

Вместо **h2 i** в нашем случае можно воспользоваться и **универсальным селектором h2 ***, так как внутри элемента <h2> находится только один элемент, но если мы захотим, добавить еще какой-то элемент, то придется изменять стили.

У Вас может возникнуть вопрос: почему мы просто не создали **селектор класса** для **i**? Благодаря **селектору потомка** у нас отсутствует необходимость с использованием глобального атрибута class назначать каждому элементу <i> свой класс, что может значительно сэкономить наше время.

Рассмотрим этот пример:

```
<!DOCTYPEhtml>
<html>
<head>
  <metacharset = "UTF-8">
  <title>Селекторыпотомков</title>
<style>
.test{
background-color: lime; /* задаёмцветзаднегофона */
color: white; /* задаёмцветшрифта*/
font-size: 30px; /*Указываем размер шрифта*/
height: 40px; /* указываем высоту элемента*/
```

```
text-align: center; /* горизонтальное выравнивание текста по центру */
}
h2 i{
color:red; /* задаём цвет шрифта*/
}
</style>
</head>
  <body>
        <h2 class = "test">Заголовок<i>второго</i>уровня</h2>
        <h2>Заголовок <i>второго</i> уровня</h2>
        <h2>Заголовок <i>второго</i> уровня</h2>
  </body>
</html>
```

В этом примере мы использовали **селектор потомков**, чтобы выделить определенным цветом все элементы <i>, размещенные внутри элементов <h2>. Кроме того, для первого элемента <h2> мы создали с помощью **селектора класса** определенный стиль.

Результат нашего примера:

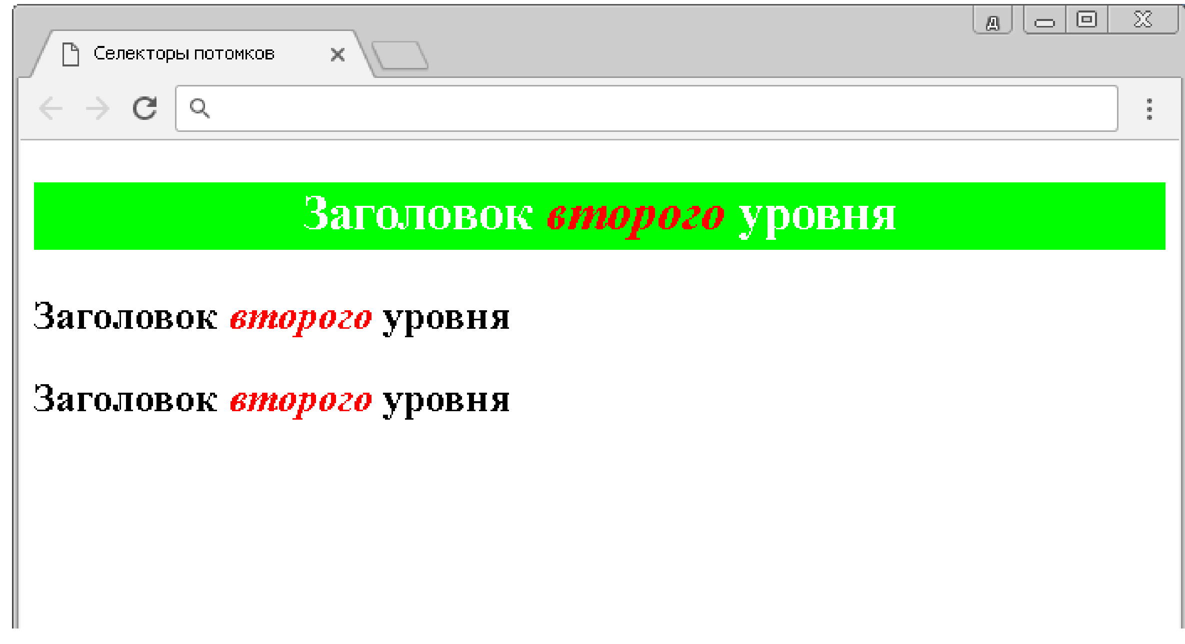


Рис. 9 Пример использования селектора потомков.

Думаю, Вы уже начали осознавать какие возможности даёт CSS в оформлении веб страниц, но это только начало нашего пути в изучении, давайте рассмотрим другие примеры, постепенно все более углубляясь в тонкости использования таблиц стилей на ваших страницах.

В следующем примере размещены три ссылки (элементы <a>) внутри элемента маркированного списка (элемент <li>), и разместил еще одну ссылку внутри абзаца:

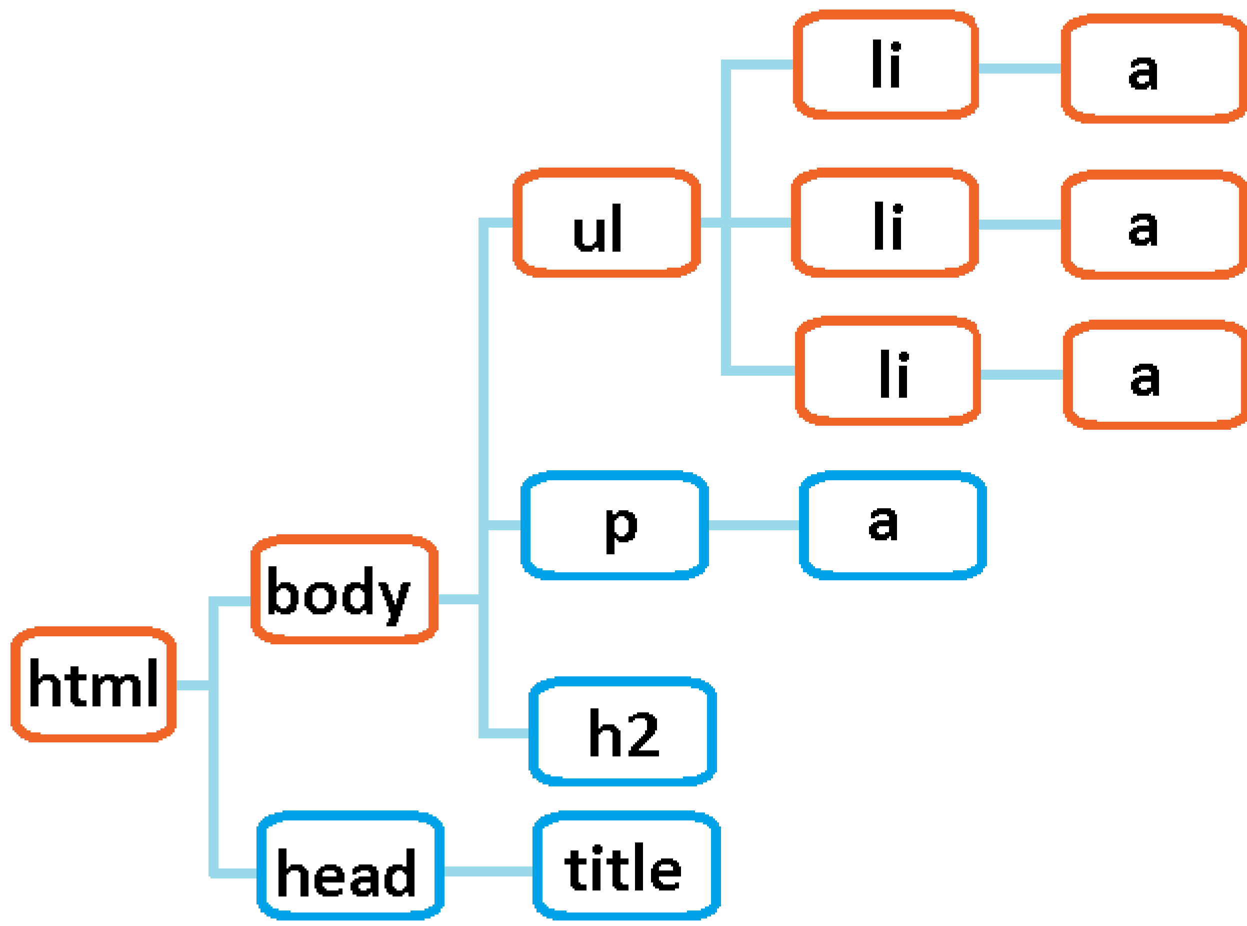


Рис. 10 Древовидная структура страницы.

При создании селектора потомков вы объединяете селекторы вместе согласно "ветви дерева" документа, помещая самого старшего родителя слева, а форматируемый элемент располагаете справа.

Селекторы, которые будут работать аналогично и выбирать элементы <a> внутри элементов <li>:

```
htmlbodyulli a {блок объявлений}
htmlulli a {блок объявлений}
htmlli a {блок объявлений}
bodyulli a {блок объявлений}
bodyli a {блок объявлений}
ulli a{блок объявлений}
li a{блок объявлений}
```

Обратите внимание, что селекторы потомков могут включать более двух элементов, то есть содержать весь путь по дереву страницы, или наоборот могут не включать какие-то элементы, которые вложены по пути к форматируемому элементу и это тоже будет работать. Старайтесь без необходимости не включать "лишние" селекторы по пути к форматируемому элементу и использовать короткие селекторы.

Используя селектор потомков, давайте отформатируем ссылки, которые находятся только внутри элементов маркированного списка:

```
li a {
text-decoration: none; /* убираем декорирование текста (нижнее подчеркивание под ссылкой) */
font-size: 20px; /* устанавливаем размер шрифта */
}
```

Как вы понимаете, ссылка, помещённая внутри элемента <p> не будет отформатирована, для неё мы создадим свой стиль:

```
p a {
color: #000; /* устанавливаем черный цвет текста */
background-color: yellow; /* устанавливаем цвет заднего фона */
}
```

Теперь соберем это в одном примере и посмотрим результат:

```
<!DOCTYPE html>
```

```
<html>
<head>
  <metacharset = "UTF-8">
  <title>Декорирование ссылок с использованием селекторов потомков</title>
<style>
li a {
text-decoration: none; /* убираем декорирование текста (нижнее подчеркивание под ссылкой) */
font-size: 20px; /* устанавливаем размер шрифта */
}
p a {
color: #000; /* устанавливаем черный цвет текста */
background-color: yellow; /* устанавливаем цвет заднего фона */
}
</style>
</head>
  <body>
        <p>Нажмите для перехода к поиску <ahref = "http://yandex.ru">Яндекс</a></p>
        <ul>
              <li><a href = "http://yandex.ru">Яндекс</a></li>
              <li><ahref = "http://yandex.ru">Яндекс</a></li>
              <li><ahref = "http://yandex.ru">Яндекс</a></li>
        </ul>
  </body>
</html>
```

Результат нашего примера:

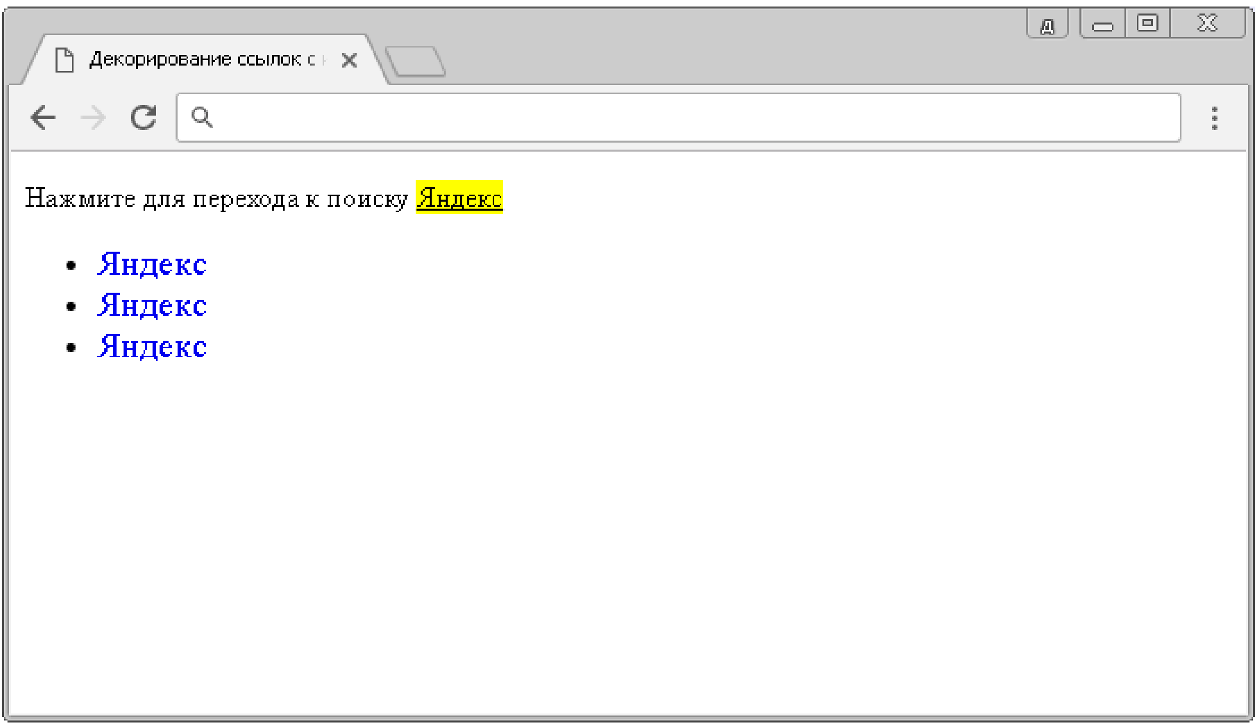


Рис. 11 Пример декорирования ссылок с использованием селекторов потомков.

Продвинутое использование селекторов потомка

Рассмотрим пример, который более приближен к текущим реалиям, а именно комбинирование селекторов класса, а не селекторов типа, как мы рассматривали в примерах выше.

Предположим, что вы создаете страницу о сотрудниках Вашей организации. Вам необходимо разместить следующую информацию о каждом сотруднике: его **фото**, **фамилия и имя**, **телефон** и **адрес электронной почты**.

Предлагаю каждого сотрудника поместить в отдельный контейнер (элемент <div>) и внутри контейнера отдельно стилизовать контактные данные о сотруднике:

```
<divclass = "top-card">/* создаем контейнер с классом  top-card */
  <imgsrc = "nich.jpg">/* добавляем фото сотрудника */
  <pclass = "name">BorisBritva</p>/* добавляем абзац с классом name (фамилия имя сотрудника) */
  <pclass = "phone">88005553535</p>/* добавляем абзац с классом phone (телефон сотрудника) */
```

```
<p>mail@any.com</p>/* добавляем абзац с адресом электронной почты сотрудника */
</div>
```

Затем мы создаем стили для контейнера и селекторов потомков:

```
.top-card{/* выбирает контейнер с классом  top-card */
border: 5px solidorange; /* создаем сплошную границу оранжевого цвета размером 5 пикселей */
width: 100px; /* задаем ширину элемента */
}
.top-cardimg{/* выбирает все изображения (элемент <img>) внутри контейнера с классом top-card */
width: 100px; /* задаем ширину изображения */
height: 100px; /* задаем высоту изображения */
border-bottom: 1px solid; /* устанавливаем сплошную границу снизу размером 1 пиксель (черный цвет по умолчанию) */
}
.top-card .name{/* выбирает элементы с классом name внутри контейнера с классом top-card */
color: blue; /* устанавливаем цвет текста */
}
.top-card .phone{/* выбирает элементы с классом name внутри контейнера с классом top-card */
color: red; /* устанавливаем цвет текста */
}
.top-card p{/* выбирает все абзацы (элемент <p>) внутри контейнера с классом top-card */
text-align: center; /* горизонтальное выравнивание текста по центру */
}
```

Теперь соберем это в одном примере и посмотрим результат:

```
<!DOCTYPE html>
<html>
<head>
  <metacharset = "UTF-8">
  <title>Стилизация карточки сотрудника</title>
<style>
.top-card{/* выбирает контейнер с классом  top-card */
border: 5px solidorange; /* создаем сплошную границу оранжевого цвета размером 5 пикселей */
width: 100px; /* задаем ширину элемента */
}
.top-cardimg{/* выбирает все изображения (элемент <img>) внутри контейнера с классом top-card */
width: 100px; /* задаем ширину изображения */
height: 100px; /* задаем высоту изображения */
border-bottom: 1px solid; /* устанавливаем сплошную границу снизу размером 1 пиксель (черный цвет по умолчанию) */
}
.top-card .name{/* выбирает элементы с классом name внутри контейнера с классом top-card */
color: blue; /* устанавливаем цвет текста */
}
.top-card .phone{/* выбирает элементы с классом name внутри контейнера с классом top-card */
color: red; /* устанавливаем цвет текста */
}
.top-card p{/* выбирает все абзацы (элемент <p>) внутри контейнера с классом top-card */
text-align: center; /* горизонтальное выравнивание текста по центру */
```

```
        }
        </style>
        </head>
          <body>
                    <div class = "top-card">/* создаемконтейнерсклассом top-card */
                              <imgsrc = "nich.jpg">/* добавляем фото сотрудника */
                              <pclass = "name">BorisBritva</p>/* добавляем абзац с классом name (фамилия имя сотрудника) */
                              <pclass = "phone">88005553535</p>/* добавляем абзац с классом phone (телефон сотрудника) */
                              <p>mail@any.com</p>/* добавляем абзац с адресом электронной почты сотрудника */
                    </div>
          </body>
        </html>
```

Результат нашего примера:

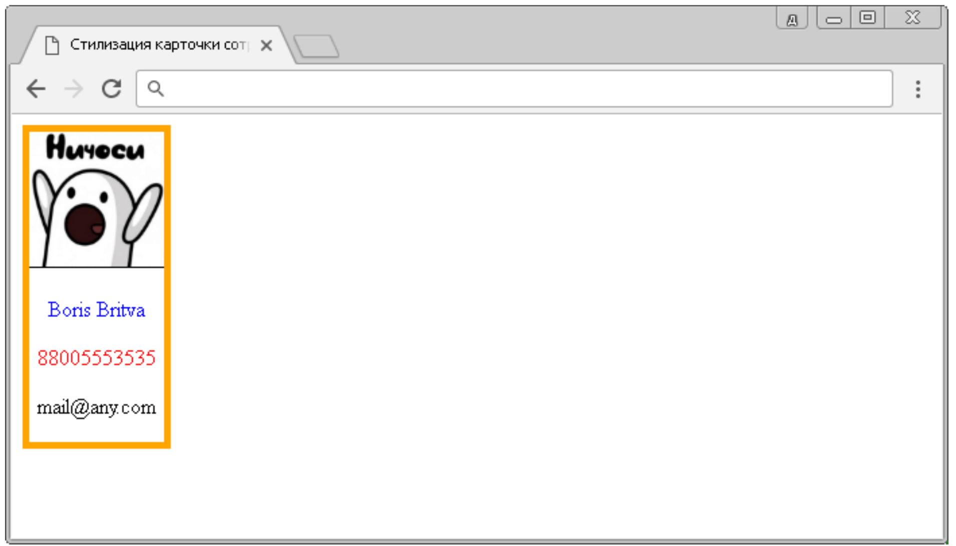


Рис. 12 Пример стилизации карточки сотрудников с использованием селекторов потомков.

Глава 13. Псевдокласы и псевдоэлементы

Псевдоэлементы

Псевдоэлементы ::first-letter и ::first-line

В современном стандарте CSS 3 используются 5 (пять) основных псевдоэлементов, которые добавляются к селекторам и имеют следующий синтаксис:

```
/* синтаксис CSS 3 */
селектор::псевдоэлемент{/* двойное двоеточие */
CSS свойство: значение;
}
/* синтаксис CSS 2 */
селектор:псевдоэлемент{/* одинарное двоеточие */
CSS свойство: значение;
}
```

Обратите внимание, что в настоящее время используется синтаксис, который предусматривает двойное двоеточие перед псевдоэлементом. Не смотря на то, что браузеры поддерживают оба варианта, рекомендуется использовать синтаксис CSS 3.

Первыми псевдоэлементами, которые мы рассмотрим будут **::first-letter** и **::first-line**. Псевдоэлемент **::first-letter** позволяет создавать инициал (буквицу) — начальный символ текстового блока. Такой метод часто можно встретить в печатных изданиях, например, в сказках.

Псевдоэлемент **::first-line** позволяет стилизовать первую строку абзаца, которая будет отличаться от основного текста. Данный псевдоэлемент дает более широкие возможности для оформления текста.

Давайте рассмотрим пример их использования:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset = "UTF-8">
```

```
<title>Псевдоэлементы ::first-letter и ::first-line</title>
<style>
::first-letter{/* изменяем начальный символ текстового блока */
color: red; /* устанавливаем цвет шрифта красный */
font-size: 24px; /* задаем размер для первого символа */
}
.test::first-line{/* изменяем первую строку элемента с классом test */
color: green; /* устанавливаем цвет шрифта первой строки зелёный
*/
font-family: Arial; /* устанавливаем шрифт Arial для первой строки
*/
}
</style>
</head>
  <body>
      <p>Яндекс. Найдется, всё.</p>
      <pre>У лукоморья дуб зелёный;
      Златая цепь на дубе том:
      И днём и ночью кот учёный
      Всё ходит по цепи кругом;
      Идёт направо - песнь заводит,
      Налево - сказку говорит.</pre>
  </body>
</html>
```

В этом примере с помощью псевдоэлемента **::first-letter** мы установили, что первая буква каждого текстового блока (в нашем случае абзаца - элемент <p>) будет красного цвета и размером **24** пикселя. Кроме того, с использованием псевдоэлемента **::first-line**, мы указали, что первая строка в элементе с классом *test* (элемент <pre>) будет шрифтом *Arial зеленого* цвета.

Результат нашего примера:

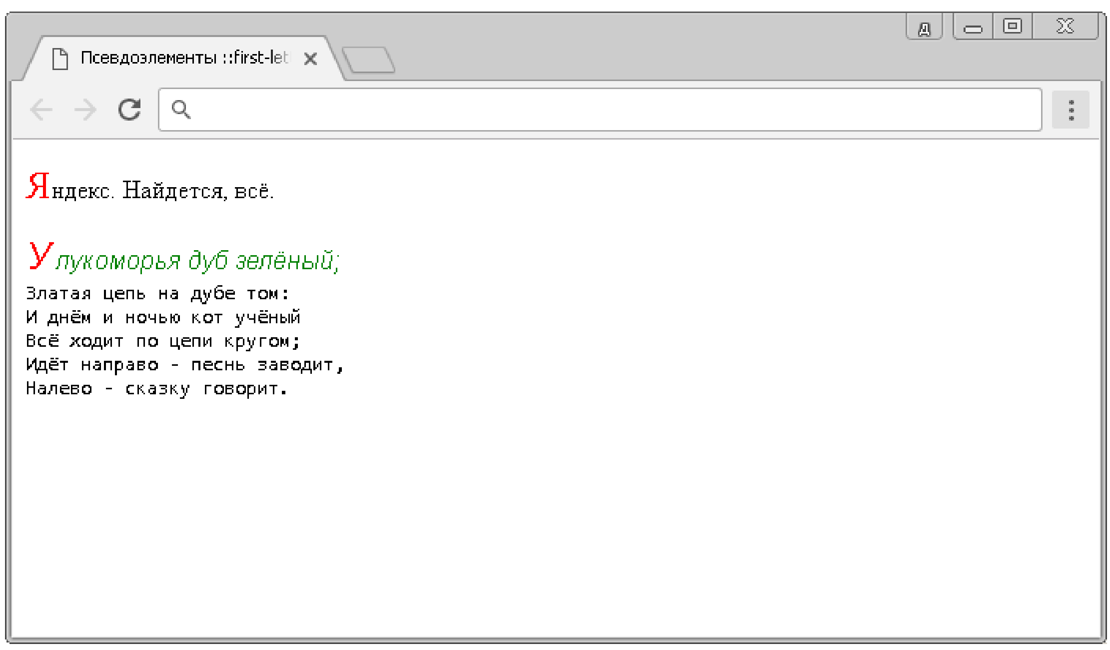


Рис. 13 Пример использования псевдоэлементов :first-letter и :first-line.

Псевдоэлемент ::selection

Следующий псевдоэлемент - **::selection**, он позволяет добавлять тень к тексту, управлять цветом фона и цветом текста, выделенного пользователем (по умолчанию: фон голубой, цвет текста белый). Это основные CSS свойства, которые используются с этим псевдоэлементом. Работу с текстовой тенью мы подробно рассмотрим далее в учебнике в статье: "**Текстовая тень в CSS**".

В использовании этого псевдоэлемента есть небольшой нюанс. Браузер *Firefox* поддерживает этот псевдоэлемент только с префиксом производителя. Чтобы на Вашем сайте была поддержка этого псевдоэлемента всеми основными браузерами, то обязательно включайте описание и для *Firefox*:

```
::selection{/* описание псевдоэлемента для всех браузеров */
color: blue; /* устанавливаем цвет текста */
background-color: orange; /* устанавливаем цвет заднего фона */
}
::-moz-selection{/* описание псевдоэлемента для Firefox */
color: blue; /* устанавливаем цвет текста */
```

```
background-color: orange; /* устанавливаем цвет заднего фона */
}
```

Мы еще неоднократно в этом учебнике будем использовать CSS свойства, которые поддерживаются теми, или иными браузерами только со специальным вендорными префиксами производителей браузеров, они используются для того, чтобы задействовать то, или иное экспериментальное свойство. Работу с вендорными префиксами мы подробно рассмотрим далее в учебнике в статье: "**CSS функции: линейные градиенты**".

Псевдоэлемент **::selection** был разработан для селекторов **CSS уровня 3**, но был удалён до того как получил рекомендательный статус. Так, в настоящее время он не принадлежит к какой-либо спецификации и возможно будет добавлен в будущей спецификации CSS.

Рассмотрим пример использования псевдоэлемента **::selection**:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset = "UTF-8">
  <title>Псевдоэлемент ::selection</title>
<style>
::selection{/* описание псевдоэлемента для всех браузеров */
color: blue; /* устанавливаем цвет текста */
background-color: orange; /* устанавливаем цвет заднего фона */
}
::-moz-selection{/* описание псевдоэлемента для Firefox */
color: blue; /* устанавливаем цвет текста */
background-color: orange; /* устанавливаем цвет заднего фона */
}
</style>
```

```
</head>
  <body>
<p> Обыкновенный бегемот, или гиппопотам (лат. Hippopotamusamphibius) — млекопитающее из отряда парнокопытных, подотряда свинообразных (нежвачных), семейства бегемотовых, единственный современный вид рода Hippopotamus. Характерной особенностью бегемота является его полуводный образ жизни — большую часть времени он проводит в воде, выходя на сушу лишь ночью на несколько часов для кормёжки. Бегемот обитает только у пресной воды, хотя может изредка оказываться в море.</p>	</body>
</html>
```

Результат нашего примера:

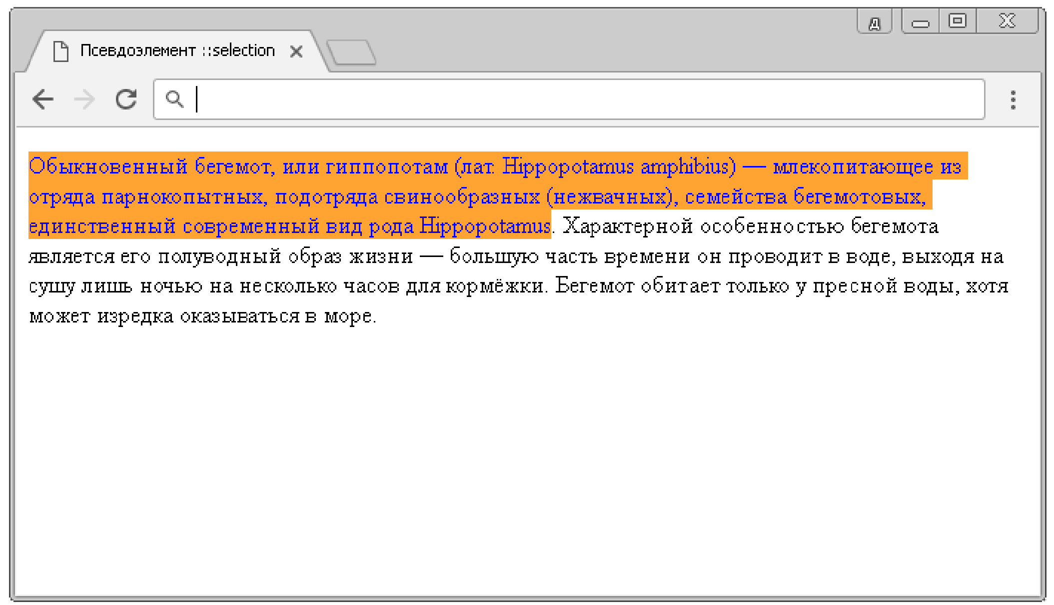


Рис. 14 Пример использования псевдоэлемента ::selection.

Псевдоэлементы ::before и ::after

Псевдоэлемент **::before** добавляет определённое содержимое перед каждым указанным элементом. Псевдоэлемент **::before**используется вместе со свойством **content**, которое необходимо для вставки сгенерированного контента.

Рассмотрим пример, в котором перед каждым блоком с изображением и перед каждым абзацем будет генерироваться (добавляться) определенная фраза:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset = "UTF-8">
  <title>Пример использования псевдоэлемента ::before</title>
<style>
img{/* выбираем все изображения */
width: 100px; /* задаем ширину элемента */
height: 100px; /* задаем высоту элемента */
}
div.omg::before{/* выбираем все элементы <div> с классом omg и добавляем перед каждым содержимое */
content: "Внимание! Спасибо за внимание!"; /* содержимое, которое будет добавлено */
}
p::before{/* выбираем все элементы <p> и добавляем перед каждым содержимое */
content: "Ответ: "; /* содержимое, которое будет добавлено */
}
</style>
</head>
  <body>
      <divclass = "omg">
          <img src = "nich.jpg"alt = "nich">
      </div>
      <div class = "omg">
          <img src = "nich.jpg" alt = "nich">
```

```
                </div>
                <p>Нет</p>
                <p>Да</p>
    </body>
</html>
```

В этом примере мы указали фиксированную ширину для всех изображений (ширина и высота **100** пикселей). Кроме того, мы выбрали все элементы <div> с классом *omg* и добавили перед ними определенную фразу. Перед абзацами (элементы <p>) также добавляется определенная фраза.

Обращаю Ваше внимание, что к таким элементам как <input> и <img> напрямую псевдоэлемент **::before** применить нельзя. Один из выходов из этой ситуации – заключить элемент в блочный элемент <div>.

Результат нашего примера:

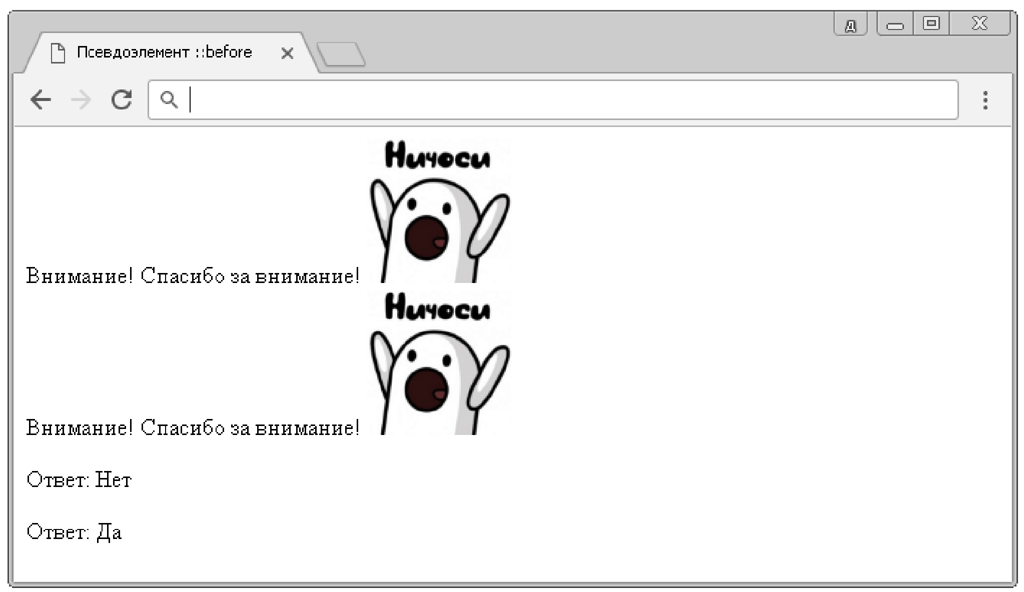


Рис. 15 Пример использования псевдоэлемента ::before.

Псевдоэлемент **::after** выступает прямой противоположностью **::before** и добавляет содержимое после определенного элемента, а не до. Он также используется вместе со свойством **content**, которое используется для вставки сгенерированного контента.

Рассмотрим пример, в котором после каждого блока с изображением и после каждого абзаца будет генерироваться (добавляться) определенная фраза:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset = "UTF-8">
  <title>Пример использования псевдоэлемента ::after</title>
<style>
img{/* выбираем все изображения */
width: 100px; /* задаем ширину элемента */
height: 100px; /* задаем высоту элемента */
}
div.omg::after{/* выбираем все элементы <div> с классом omg и добавляем после каждого содержимое */
content: "Спасибо за внимание!"; /* содержимое, которое будет добавлено */
}
p::after{/* выбираем все элементы <p> и добавляем после каждого содержимое */
content: " - Ваш ответ"; /* содержимое, которое будет добавлено */
}
</style>
</head>
  <body>
        <divclass = "omg">
              <img src = "nich.jpg"alt = "nich">
        </div>
        <div class = "omg">
              <img src = "nich.jpg" alt = "nich">
```

```
            </div>
            <p>Нет</p>
            <p>Да</p>
    </body>
</html>
```

В этом примере мы указали фиксированную ширину для всех изображений (ширина и высота **100** пикселей). Кроме того, мы выбрали все элементы <div> с классом *omg* и добавили после них определенную фразу. После абзацев (элементы <p>) также добавляется определенная фраза.

Обращаю Ваше внимание, что к таким элементам как <input> и <img> напрямую псевдоэлемент **::after** применить нельзя.

Результат нашего примера:

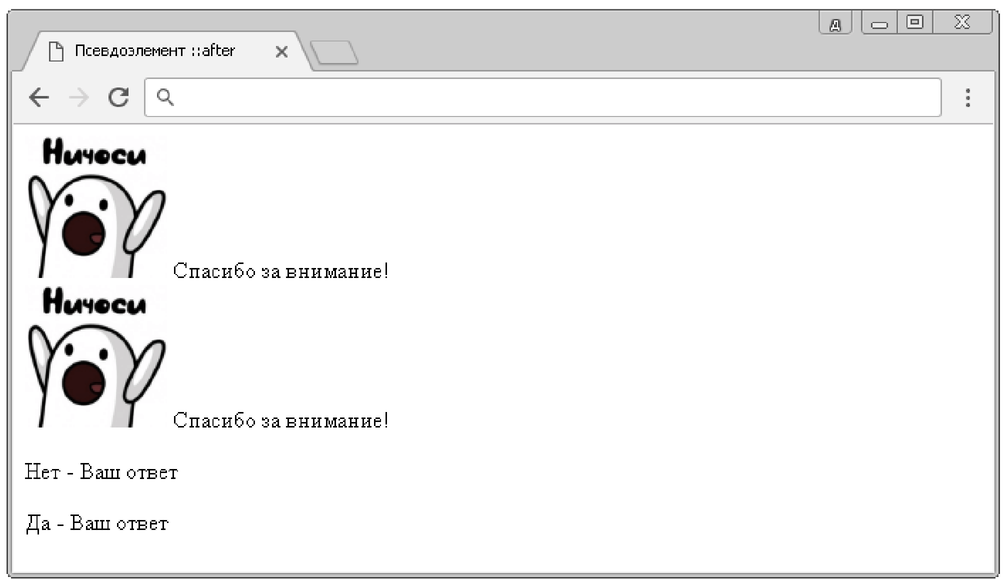


Рис. 16 Пример использования псевдоэлемента ::after.

Применение псевдоэлементов **::before** и **::after** позволяют добавлять перед элементами не только текстовую информацию, но даже изображения, счетчики, разноцветные буллиты, которые используются в маркированных списках и так далее. Мы с Вами еще обязательно вернемся к этим пседоэлементам с более глубокими знаниями CSS.

Псевдоклассы

В настоящее время в CSS 3 существует более **тридцати** псевдоклассов.

В этой статье мы рассмотрим псевдоклассы, которые дают возможность форматировать ссылки в зависимости от состояния и рассмотрим псевдокласс, который позволяет изменить стиль элемента, когда элемент получает фокус, либо при выборе его пользователем при помощи клавиатуры.

Псевдоклассы в отличие от псевдоэлементов добавляются к селекторам с одним двоеточием:

```
селектор:псевдокласс{/* одинарное двоеточие */
CSS свойство: значение;
}
```

Фокус на элементе

Псевдокласс **:focus** производит выбор элементов, которые в настоящий момент находятся в фокусе (допускается использовать на элементы, которые принимают события клавиатуры или другие данные, вводимые пользователем).

Давайте рассмотрим пример использования псевдокласса:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset = "UTF-8">
  <title>Псевдокласс :focus</title>
<style>
input:focus{/* выбирает все элементы <input>, которые находятся в фокусе */
background-color: khaki; /* устанавливаем цвет заднего фона */
color: brown; /* устанавливаем цвет шрифта */
font-size: 18px; /* устанавливаем размер шрифта */
```

```
        border: 2px solidorange; /* устанавливаем сплошную границу
оранжевого цвета размером 2  пикселя */
        }
        </style>
        </head>
          <body>
                <form>
                        Логин<inputtype = "text"name =
"login"></input><br><br>
                        Пароль<inputtype = "password"name =
"password"></input>
                </form>
          </body>
        </html>
```

В данном примере мы создали два элемента <input>, первый с типом **text** (однострочное текстовое поле), а второй с типом **password** (поле с паролем).

Если пользователь производит выбор элементов <input> (поле получает фокус), то поле получает стиль, который мы задали, а если поле теряет фокус, то стиль возвращается на первоначальный.

Результат нашего примера:

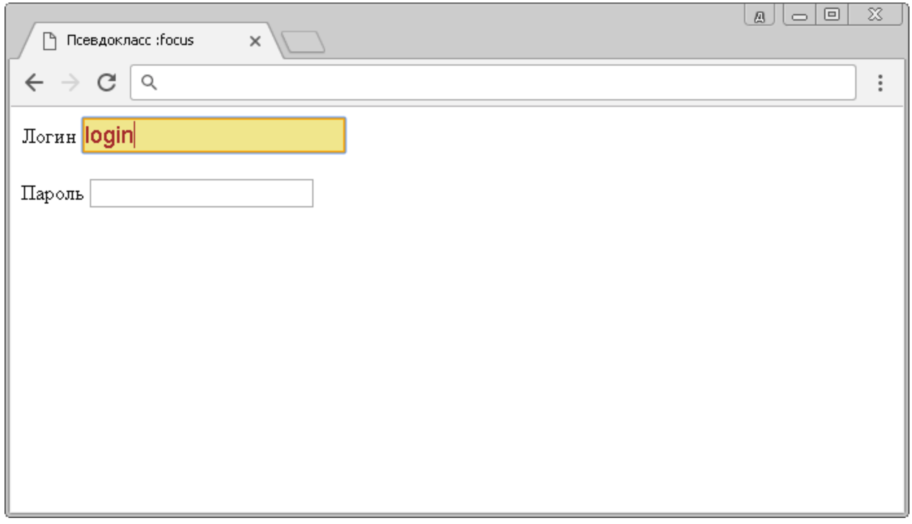


Рис. 17 Пример использования псевдоэлемента :focus.

Стилизация ссылок

link

Псевдокласс **:link** задаёт стиль всем ссылкам, по которым пользователь не произвёл переход (не посещенные ссылки). Как правило, используется с псевдоклассом **:visited**, который определяет стиль для посещённых пользователем ссылок.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset = "UTF-8">
  <title>Псевдокласс :link</title>
<style>
a:link{
background-color: orange;
}
</style>
</head>
  <body>
        <p>Непосещённыессылки:</p>
        <ahref =
"https://ru.wikipedia.org/wiki/Квантовая_механика">Квантовая_механика
</a>
        <ahref =
"https://ru.wikipedia.org/wiki/Начертательная_геометрия">Начертательна
я_геометрия</a>
  </body>
</html>
```

В этом примере с использованием псевдокласса **:link** мы указали, что ссылки (элементы <a>) по которым пользователь не переходил подсвечиваются оранжевым цветом:

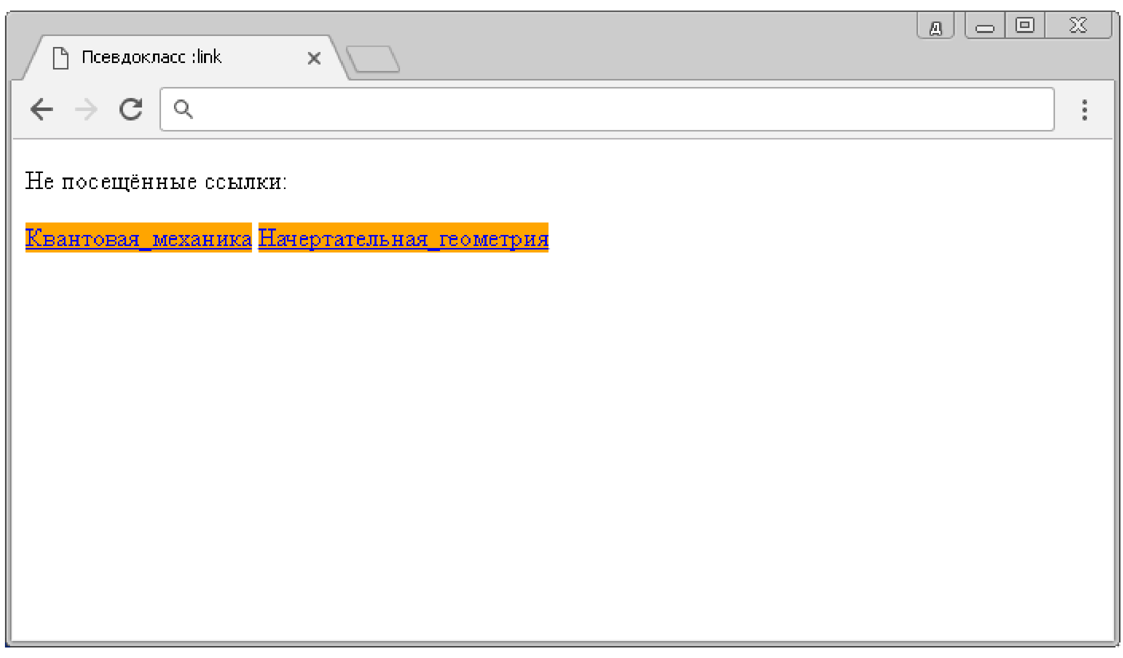


Рис. 17а Пример использования псевдокласса :link.

visited

Псевдокласс **:visited** задаёт стиль всем ссылкам, по которым пользователь производил переход (посещенные ссылки).

```
<!DOCTYPE html>
<html>
<head>
  <meta charset = "UTF-8">
  <title>Псевдокласс :visited</title>
<style>
a:visited{
color: orange;
}
</style>
</head>
  <body>
        <p>Ссылки, которые Вы уже посещали:</p>
        <ahref                                    =
"https://ru.wikipedia.org/wiki/Квантовая_механика">Квантовая_механика
</a>
```

```
<ahref =
"https://ru.wikipedia.org/wiki/Начертательная_геометрия">Начертательна
я_геометрия</a>
    </body>
</html>
```

В этом примере с использованием псевдокласса **:visited** мы указали, что ссылки (элементы <a>) по которым пользователь производил переход имеют красный цвет:

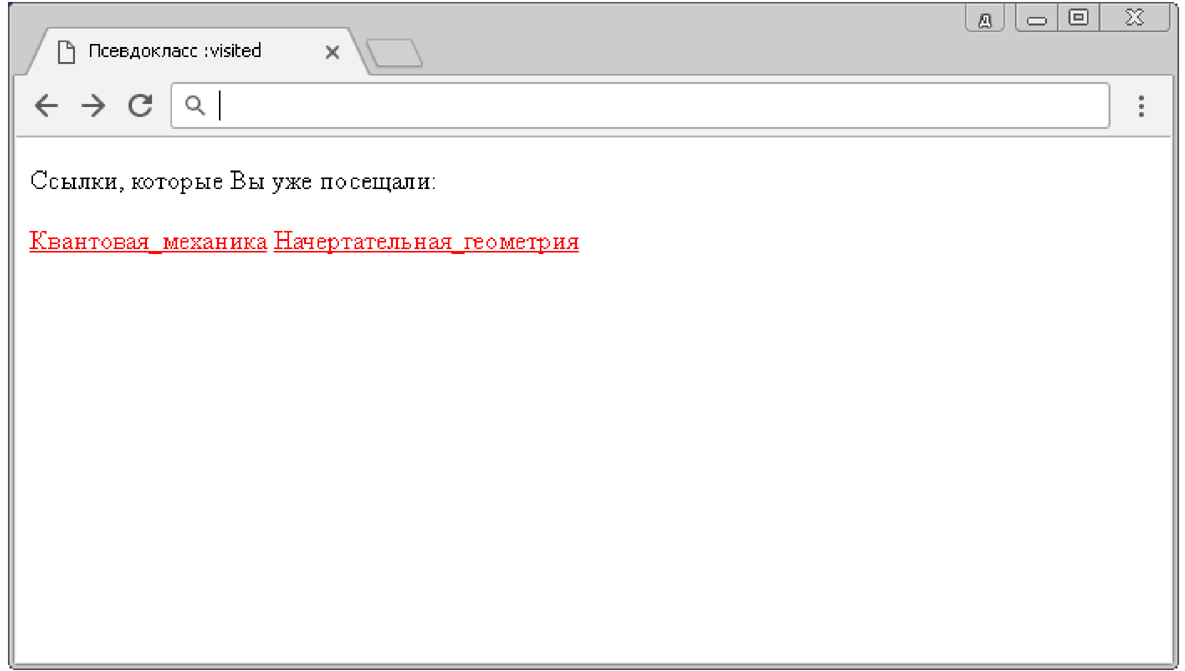


Рис. 176 Пример использования псевдокласса :visited.

active

Псевдокласс **:active** используется для выбора активного элемента - элемента на который в данный момент пользователь кликнул мышкой. Стиль для элемента применится и будет работать пока удерживается кнопка мыши.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset = "UTF-8">
  <title>Псевдокласс :active</title>
<style>
```

```
p.test:active, h3.test:active, a.test:active{/* групповойселектор */
background-color: orange; /* устанавливаем цвет заднего фона */
}
</style>
</head>
  <body>
        <h3class = "test">Заголовок третьего уровня с классом test</h3>
        <pclass = "test">Абзацсклассом test</p>
        <aclass = "test"href = "#">Ссылкасклассом test</a>
  </body>
</html>
```

Выберите любой элемент и удерживайте на нём кнопку мыши:

Заголовок третьего уровня с классом test

Абзац с классом test

Ссылка с классом test

hover

Ну и заключительный псевдокласс, который мы рассмотрим в этой статье - **:hover**, он используется практически на каждой странице любого сайта. Псевдокласс **:hover** используется для стилизации любого элемента, на который в данный момент указывает курсор мыши. Чаще всего используют данный псевдокласс с ссылками, кнопками, в меню навигации и таблицами.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset = "UTF-8">
  <title>Псевдокласс :hover</title>
<style>
table{
```

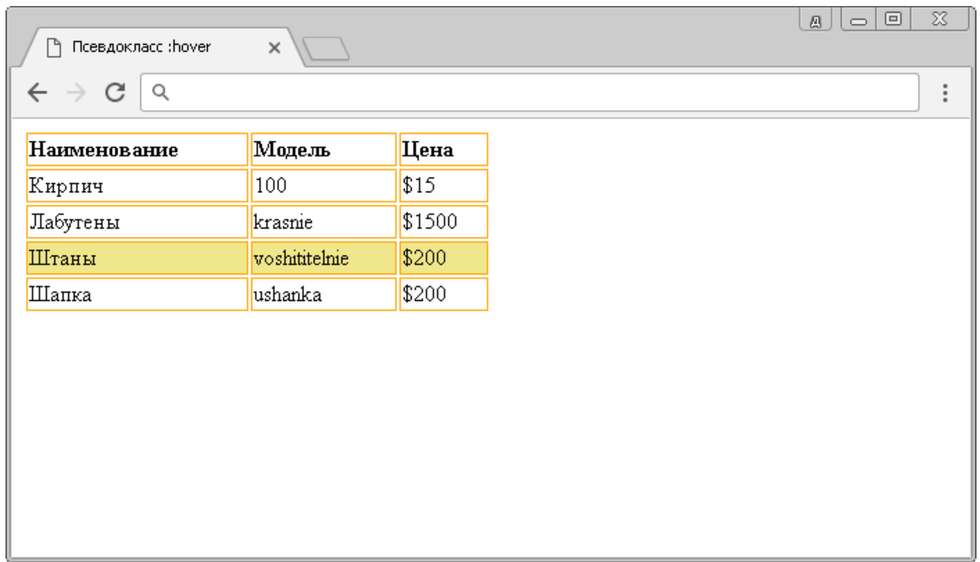
```
width: 50%; /* указываем ширину элемента в процентах от ширины области просмотра */
}
th, td{
text-align: left; /* горизонтальное выравнивание текста по левому краю */
border: 1px solidorange; /* сплошная граница размером 1 пиксель оранжевого цвета */
}
tr:hover{
background-color: khaki; /* устанавливаем цвет заднего фона */
}
</style>
</head>
  <body>
        <tr>

  <th>Наименование</th><th>Модель</th><th>Цена</th>
        </tr>
        <tr>
              <td>Кирпич</td><td>100</td><td>$15</td>
        </tr>
        <tr>
              <td>Лабутены</td><td>krasnie</td><td>$1500</td>
        </tr>
        <tr>
              <td>Штаны</td><td>voshititelnie</td><td>$200</td>
        </tr>
        <tr>
              <td>Шапка</td><td>ushanka</td><td>$200</td>
```

```
            </tr>
      </body>
    </html>
```

В этом примере с использованием псевдокласса **:hover** мы подсвечиваем строку на которую в настоящее время наведен курсор:

Обращаю Ваше внимание, что если вы используете псевдоклассы **:link** и **:visited**, то псевдокласс **:hover** должен быть расположен после них.

Результат нашего примера:



Наименование	Модель	Цена
Кирпич	100	$15
Лабутены	krasnie	$1500
Штаны	voshititelnie	$200
Шапка	ushanka	$200

Рис. 17в Пример использования псевдокласса :hover.

В этом примере были использованы относительные единицы измерения - процентные значения.

Глава 14. Селекторы атрибутов

Селекторы атрибутов

Исходя из названия заголовка, вы можете догадаться, что благодаря использованию селекторов атрибутов вы можете выбрать те элементы страницы, которые имеют определённые HTML атрибуты.

Давайте рассмотрим следующий селектор и попробуем разобраться, что он значит:

```
img[title]{/* выбирает все элементы <img> с атрибутом title */
блок объявлений;
}
```

img в данной случае аналогичен селектору типа, т.е выбирает все элементы <img>, а в квадратных скобках мы задаем имя атрибута этого элемента. То есть происходит выборка всех элементов данного типа с определённым атрибутом, в нашем примере это глобальный атрибут **title**, определяющий текстовую подсказку о содержимом элемента.

Вы можете использовать селекторы атрибутов не только напрямую к элементам, но и использовать их с прочими селекторами, например с селекторами класса или id селекторами:

```
.main[title]{/* выбирает все элементы с классом main и атрибутом title */
блок объявлений;
}
#main[title]{/* выбирает элемент с идентификатором main и атрибутом title */
блок объявлений;
}
```

В данном случае первый селектор выбирает все элементы, которые имеют значение глобального атрибута **class** равным **main** и глобальным атрибутом **title**. Второй селектор выбирает элемент, который имеют значение глобального атрибута **id** равным **main** и глобальным атрибутом **title**.

Рассмотрим следующий пример в котором, мы выберем все элементы <img>, у которых присутствует атрибут **alt**, который задает альтернативный текст для изображения.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset = "UTF-8">
```

```
<title>Пример использования селектора атрибутов</title>
<style>
img{/* выбираем все изображения */
width: 100px; /* задаем ширину элемента */
height: 100px; /* задаем высоту элемента */
}
img[alt]{/* выбираем все изображения с атрибутом alt */
border: 1px solidgreen; /* устанавливаем сплошную границу размером 1 пиксель зеленого цвета */
}
</style>
</head>
<body>
<img src = "nich.jpg"alt = "nich">
<imgsrc = "nich.jpg"><!-- элемент не будет стилизован (отсутствует атрибут alt) -->
<img src = "nich.jpg"alt = "nich">
</body>
</html>
```

Результат нашего примера:

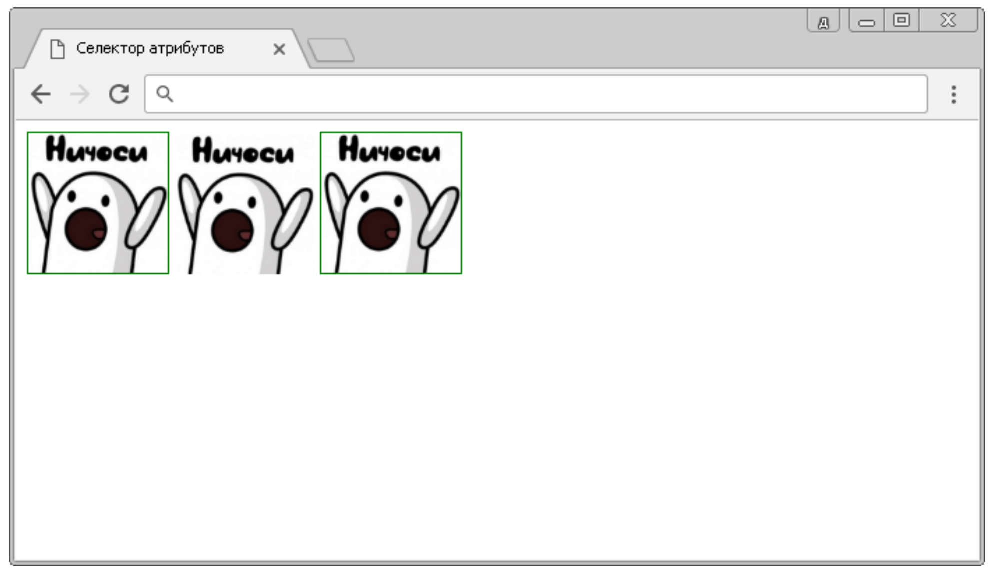


Рис. 17г Пример использования селектора атрибутов.

С помощью селектора атрибутов мы можем выбрать элементы не только с определённым атрибутом, но и указать с каким значением должен быть этот атрибут.

Давайте рассмотрим пример в котором нам необходимо по особенному стилизовать поле, предназначенное для ввода пароля.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset = "UTF-8">
  <title>Селектор атрибута с указанным значением</title>
<style>
input[type=password]{/* выбираем все элементы <input>, атрибут type которых имеет значение password */
border: 1px dottedred; /* устанавливаем точечную границу размером 1 пиксель красного цвета */
}
input[type=password]:focus{/* выбираем элемент <input>, атрибут type которого имеет значение password и который находится в фокусе */
border: 1px solidgreen; /* устанавливаем сплошную границу размером 1 пиксель зеленого цвета */
}
</style>
</head>
  <body>
      <form>
            Login: <input type = "text"name = "login"placeholder = "Введитевашлогин"><br><br>
            Password: <input type = "password"name = "password"placeholder = "Введитевашпароль"><br><br>
```

```
                <input type = "submit"name = "submit"value =
"Далее">
            </form>
    </body>
</html>
```

В этом примере мы использовали два селектора атрибутов с указанным значением, которые позволили нам выбрать поле, предназаченное для ввода пароля, а во втором случае с использованием псевдокласса **:focus** мы создали стили для того состояния, когда это поле находится в фокусе (пользователь кликнул на него, или выбрал с помощью клавиатуры). Псевдокласс **:focus** мы уже с Вами рассматривали в конце предыдущей статьи учебника "**Псевдоклассы и псевдоэлементы**".

Результат нашего примера:

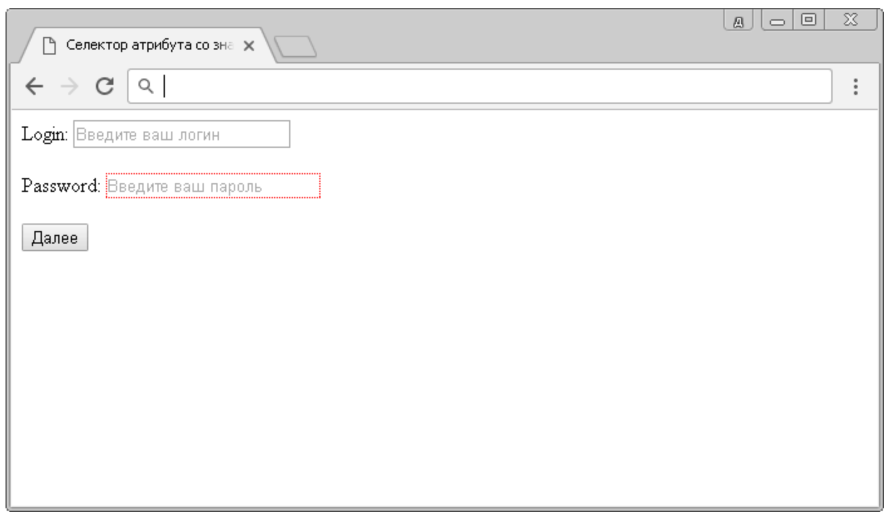


Рис. 17д Пример использования селектора атрибута с указанным значением.

В предыдущем примере для выборки мы указали конкретное значение атрибута, но в некоторых случаях нам необходимо выбрать элементы, значение атрибута которых, начинается с определённых символов. В CSS 3 для этих целей введен специальный селектор атрибута, который выбирает элементы у которых определённый атрибут имеет значение, начинающееся с определённых символов. Этот селектор атрибута имеет следующий синтаксис:

```
[attribute ^ = value]{
```

```
блок объявлений;
}
```

Давайте рассмотрим пример в котором выберем на странице все абсолютные адреса внешних гиперссылок, которые начинаются с *http://*, либо *https://*:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset = "UTF-8">
  <title>Селектор атрибута со значением, начинающимся с определённых символов</title>
<style>
a[href ^ = "http://"], a[href ^ = "https://"]{/* групповой селектор атрибутов */
color: orange; /* устанавливаем цвет текста */
}
</style>
</head>
  <body>
        <p>Сайт компании доступен как по протоколу http, так и по протоколу https:</p>
              <ul>
                    <li><a href = "http://сайткомпании.сccp">Адресс http</a></li>
                    <li><a href = "https://сайткомпании.сccp">Адресс https</a></li>
              </ul>
        <p>Может вы настроите перенаправление?</p>
  </body>
</html>
```

Обратите внимание, что значение атрибута в этом случае мы указываем в кавычках, чтобы браузер не интерпретировал это как начало гиперссылки.

Результат нашего примера:

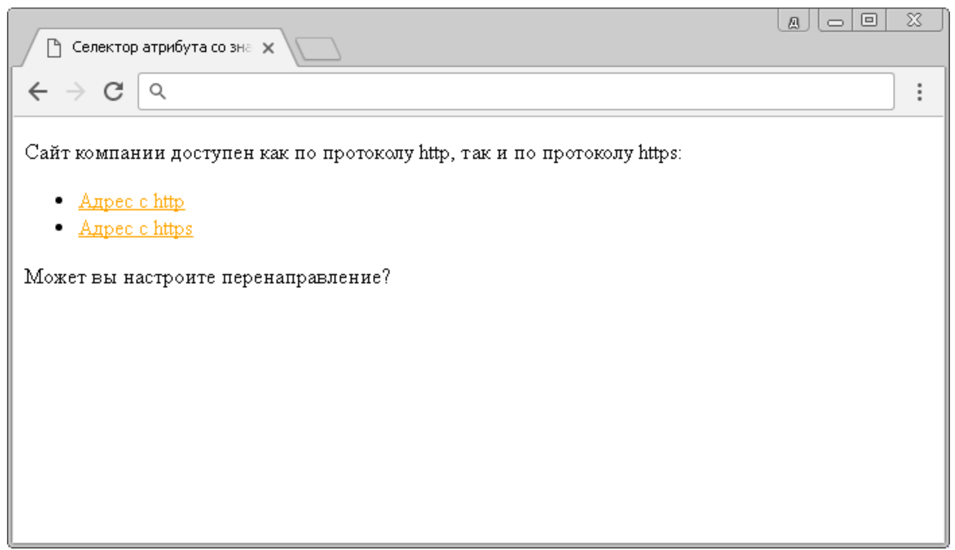


Рис. 17е Пример использования селектора атрибута со значением, начинающимся с определённых символов.

В CSS 3 был введен не только селектор атрибутов, который позволяет выбирать элементы, чьи атрибуты начинаются с определённых символов, но и элементы, чьи атрибуты заканчиваются определёнными символами.

Давайте рассмотрим пример в котором мы выберем гиперссылки, которые заканчиваются определённым разрешением файла.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset = "UTF-8">
  <title>Селектор атрибута со значением, заканчивающимся определёнными символами</title>
<style>
a[href$=".doc"]{/* выбираем все элементы с атрибутом href, значение которого закачивается на .doc */
```

```
color: green; /* устанавливаем цвет текста */
background-color: lightblue; /* устанавливаем цвет заднего фона */
}
a[href$=".mp3"] {/* выбираем все элементы с атрибутом href, значение которого закачивается на .mp3 */
background-color: khaki; /* устанавливаем цвет заднего фона */
}
</style>
</head>
  <body>
          <ahref ="http:/path.to/test.doc">Инструкция</a><br>
          <ahref ="http:/path.to/test.mp3">Песняпрозайцев</a>
  </body>
</html>
```

Обратите внимание, что значение атрибута в этом случае мы указываем в кавычках, чтобы браузер не интерпретировал это как разрешение файла.

Результат нашего примера:

Рис. 18 Пример использования селектора атрибутов со значением, заканчивающимся определёнными символами.

Давайте рассмотрим с Вами какие еще выборки можно сделать с использованием селектора атрибутов и перейдем к изучению такого полезного инструмента как псевдокласс отрицания.

Следующий селектор позволит нам, к примеру, выбрать изображения, которые содержат в названии **IMG_** (как правило такое наименование файлов использует компания *Canon*):

```
img [src*="IMG_"]{/* выбирает все элементы img, атрибут src, которых содержит символы "IMG_" */
блок объявлений;
}
/* селектор выбирает элементы с такими значениями как "xxxIMG_", "IMG_xxxx" и "xxxIMG_xxxx" */
```

Браузер интерпретирует это как необходимость выбора всех изображений, атрибут **src** которых содержит **IMG_** (в любом месте). Зачастую такой способ позволяет быстрее отформатировать необходимые элементы, по сравнению с созданием и присвоением стилевого класса для каждого элемента.

Следущий селектор позволяет выбрать элемент, значение атрибута которого содержит определённое слово (не зависимо от позиции):

```
p[title ~="home"]{/* выбирает элементы <p>, которые содержат определенное слово */
блок объявлений;
}
<ptitle = "gohome">Абзац title="gohome"</p><!-- абзац будет стилизован (содержит home) -->
<p title = "home home">Абзац title="home home"</p><!-- абзацбудетстилизован (содержит home) -->
<p title = "home-1">Абзац title="home1"</p><!-- абзацнебудетстилизован -->
```

```
<p title = "homes">Абзац title="homes"</p><!-- абзацнебудетстилизован -->
<p title = "shome">Абзац title="shome"</p><!-- абзацнебудетстилизован -->
```

Ну и заключительный селектор атрибута, который имеется в современном стандарте, позволяет выбрать элемент значение атрибута которого начинается с определенного слова:

```
[title | = home]{/* выбирает все элементы с атрибутом, значение которого начинается с определённого слова (после него не должно быть никаких символов, либо допускается продолжать значение через дефис, иначе выборка не будет произведена) */
блок объявлений;
}
<ptitle = "home">Абзац title="home"</p><!-- абзац будет стилизован (начинается с home) -->
<ptitle = "home-1">Абзац title="home-1"</p><!-- абзац будет стилизован (начинается с home после которого следует дефис) -->
<p title = "home home">Абзац title="home home"</p><!-- абзацнебудетстилизован -->
<p title = "not home">Абзац title="not home"</p><!-- абзацнебудетстилизован -->
<p title = "homes">Абзац title="homes"</p><!-- абзацнебудетстилизован -->
<p title = "shome">Абзац title="shome"</p><!-- абзацнебудетстилизован -->
```

Обратите внимание, что условие выборки будет соблюдено если атрибут содержит значение, которое содержит только указанное слово, или если после указанного слова сразу следует дефис (значение продолжается через дефис).

Псевдоклассотрицания :not()

Селектор **:not()** или псевдокласс отрицания, позволяет выбрать элементы, или селекторы отличные от указанных.

Что нельзя использовать с псевдоклассом **:not()**:

- Использовать в одном селекторе несколько псевдоклассов **:not()**.
- Использовать с псевдоэлементами(**::first-letter**, **::first-line** и так далее).
- Нельзя использовать с селекторами потомков (например, **divul a**).
- Использовать в групповых селекторах (комбинации из селекторов).

Давайте рассмотрим пример в котором по разному стилизуем изображения. Допустим у нас есть изображения фиксированного размера со следующими значениями:

```
img{/* выбираем все изображения */
width: 100px; /* ширина элемента в пикселях */
height: 100px; /* высота элемента в пикселях */
}
```

Создадим селектор класса **.photo** и применим его к необходимым изображениям, чтобы они получили оранжевую границу.

```
.photo{/* выбираем все элементы с классом photo */
border: 2px solidorange; /* сплошная граница размером 2 пикселя оранжевого цвета */
}
```

Перед Вами стоит задача изменить стиль для всех изображений (предположим, что их сотни), но при этом у Вас ограниченное количество времени и необходимо сделать так, чтобы эти изменения затронули изображения с классом **.photo**. Для этого Вам необходимо создать селектор совместно с псевдоклассом отрицания:

```
img:not(.photo) {/* выбираем все изображения, которые не имеют класса photo */
border: 2px dashedgreen; /* пунктирная граница размером 2 пикселя зеленого цвета */
}
```

Всё вместе и результат:

```
<!DOCTYPEhtml>
<html>
<head>
  <meta charset = "UTF-8">
  <title>Псевдоклассотрицания :not()</title>
<style>
img{/* выбираем все изображения */
width: 100px; /* ширина элемента в пикселях */
height: 100px; /* высота элемента в пикселях */
}
.photo{/* выбираем все элементы с классом photo */
border: 2px solidorange; /* сплошная граница размером 2 пикселя оранжевого цвета */
}
img:not(.photo) {/* выбираем все изображения, которые не имеют класса photo */
border: 2px dashedgreen; /* пунктирная граница размером 2 пикселя зеленого цвета */
}
</style>
</head>
  <body>
          <img src = "nich.jpg"alt = "nich"class = "photo">
          <img src = "nich.jpg"alt = "nich" class = "photo"><br>
```

```
            <img src = "nich.jpg"alt = "nich">
            <img src = "nich.jpg" alt = "nich">
            <img src = "nich.jpg"alt = "nich">
    </body>
  </html>
```

Как вы можете заметить на изображении ниже, мы справились с поставленной задачей и стилизовали с использованием псевдоклассаотрицания **:not()** все изображения, отлично от изображений с классом **photo**:

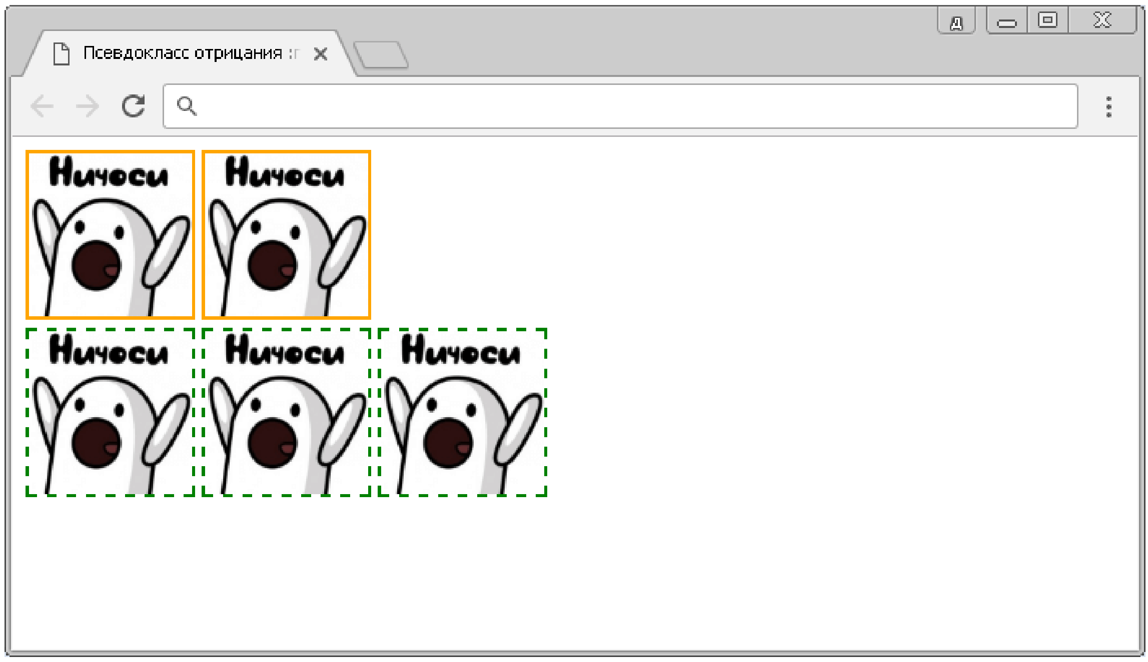


Рис. 19 Пример использования псевдоклассаотрицания :not() в CSS.

Селекторы дочерних элементов

Селектор дочерних элементов позволяет форматировать элементы, вложенные внутрь других элементов (выбирает все дочерние элементы внутри элемента родителя).

Элемент, подчиненный другому элементу более высокого уровня, является **дочерним**. На изображении ниже оба элемента <h2> и <p>являются **дочерними** по отношению к <body>, но элемент <i> при этом **не является дочерним** для элемента <body>, так как он расположен внутри тега <p>, и **является дочерним** именно для него.

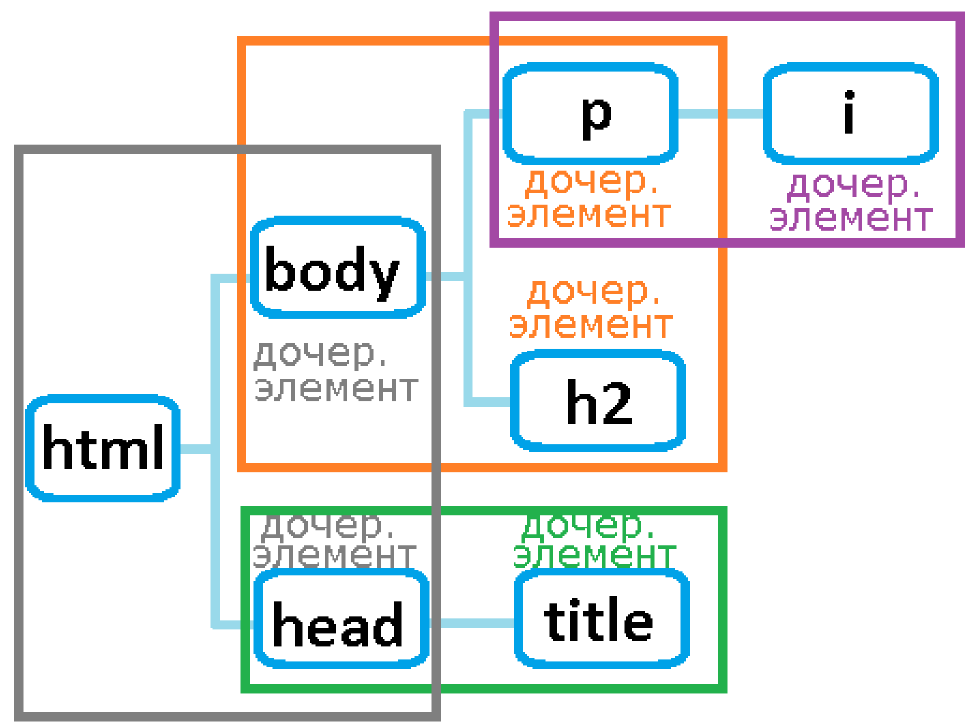


Рис. 20 Дочерние элементы в HTML документе.

Перед нами стоит задача стилизовать гиперссылку (элемент <a>), который выделен *оранжевым* цветом на изображении:

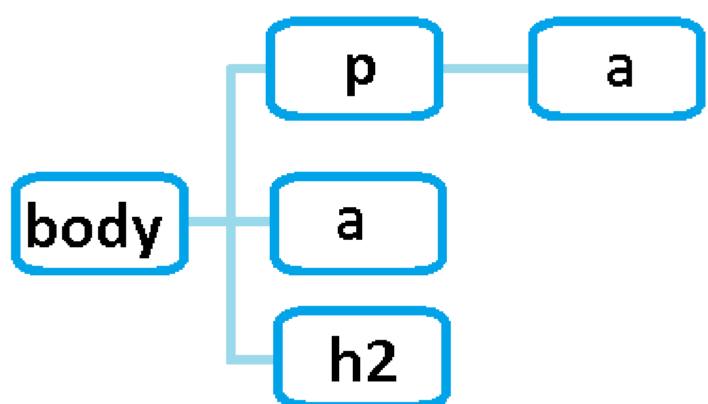


Рис. 20а Задача выбора элемента на странице.

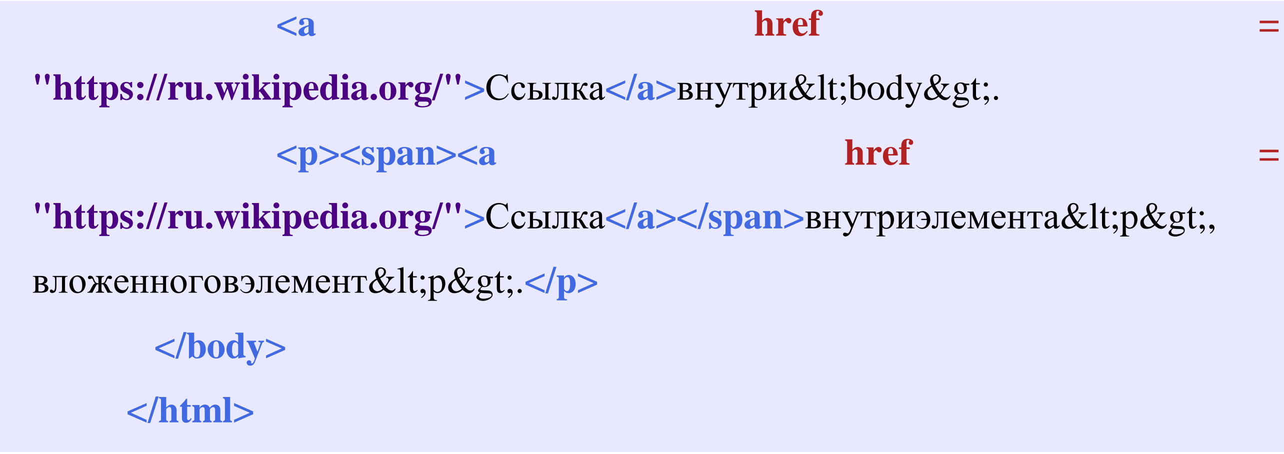
При использовании **селектора потомков body a** приведет к выбору всех элементов <a>, так как они являются вложенными по отношению к элементу <body>, если мы используем **селектор потомков p a**, то это приведет к выбору двух элементов <a>, которые вложены внутри элементов <p>, так как они оба являются его потомками. **Селекторы потомков** мы с Вами рассматривали в статье "**Селекторы. Часть 2**".

В нашем случае, как никогда лучше подойдет **селектор дочерних элементов**, благодаря которому мы можем добраться до необходимого нам элемента <a> самым простым способом.

```
p > a {/* выбирает любой элемент <a> дочерний по отношению к <p> */
блок объявлений;
}
```

Рассмотрим пример:

```
<!DOCTYPEhtml>
<html>
<head>
  <meta charset = "UTF-8">
  <title>Селектор дочерних элементов</title>
<style>
p>a{/* селектор дочерних элементов (выбираем дочерние <a>, вложенные в <p>) */
color: orange; /* устанавливаме цвет текста */
font-size: /* устанавливаме размер текста текста */
}
p a{/* селектор потомков (выбираем все потомки <a>, вложенные в <p>) */
text-decoration: none; /* убираем декорирование текста (нижнее подчеркивание) */
}
</style>
</head>
  <body>
      <p><a href = "https://ru.wikipedia.org/">Ссылка</a>внутри&lt;p&gt;.</p>
```



```
<a href = "https://ru.wikipedia.org/">Ссылка</a>внутри&lt;body&gt;.
<p><span><a href = "https://ru.wikipedia.org/">Ссылка</a></span>внутриэлемента&lt;p&gt;, вложенноговэлемент&lt;p&gt;.</p>
    </body>
  </html>
```

В этом примере с использованием **селектора дочерних элементов** мы выбрали все дочерние <a>, вложенные в <p> (один элемент), а с помощью **селектора потомков** стилизовали все потомки <a>, вложенные в <p> (убрали декорирование текста у двух элементов).

Результат нашего примера:

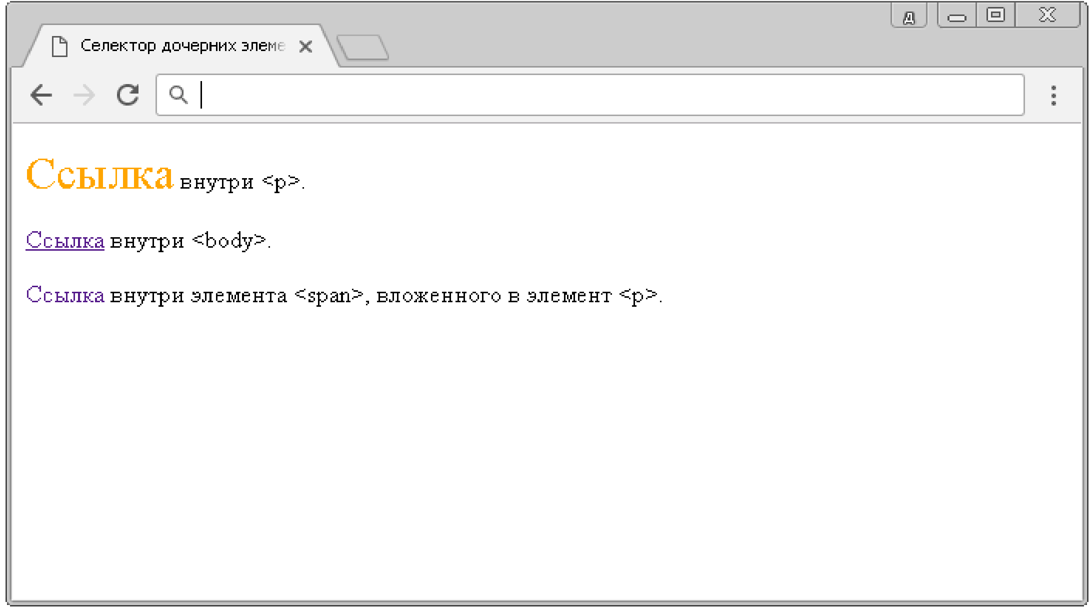


Рис. 206 Пример использования селектора дочерних элементов.

Псевдоклассы дочерних элементов

Псевдокласс :first-child

Псевдокласс **:first-child** применяет стиль к элементу в том случае, если элемент является первым дочерним элементом своего родителя.

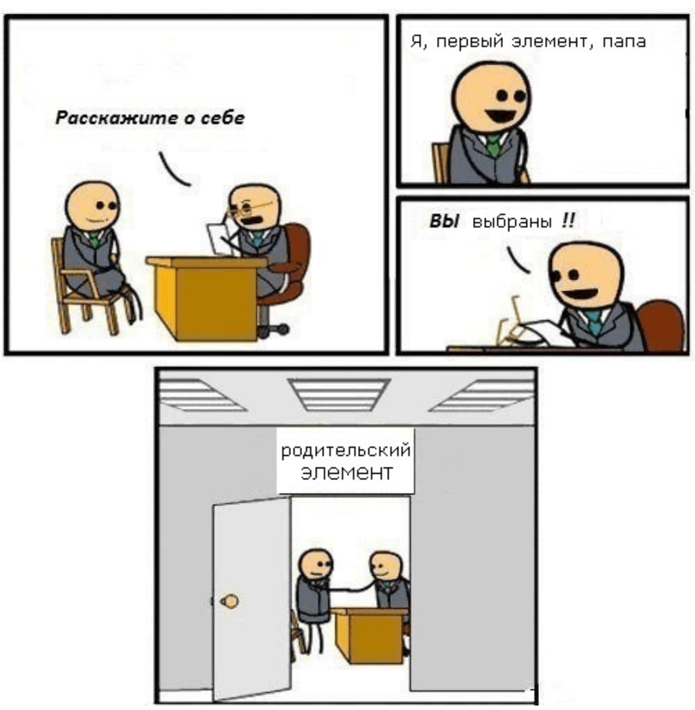


Рис. 21 Выборка с использованием псевдокласса дочерних элементов.

Давайте рассмотрим пример, в котором перед Вами стоит задача изменить стиль для всех элементов, которые выделены оранжевым цветом на изображении:

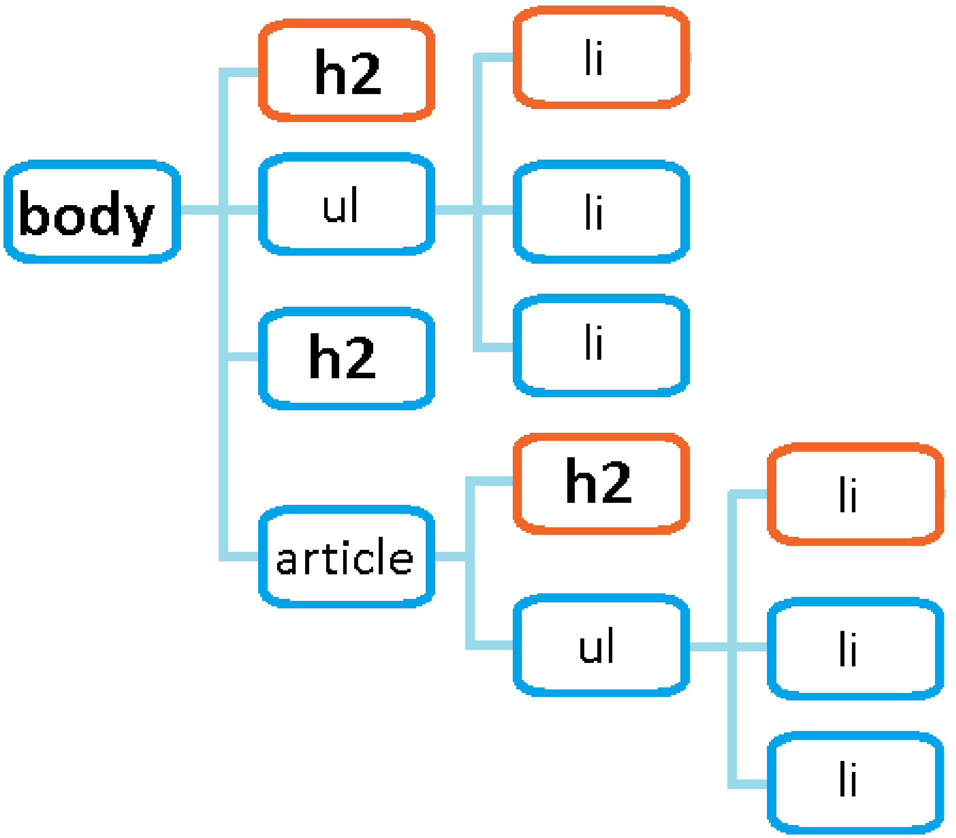


Рис. 21а Пример выбора селектора дочерних элементов.

Что общего у элементов, выделенных на изображении? А общее у них то, что выделенные элементы <h2> и <li> являются первыми дочерними элементами своих родителей, и чтобы нам их стилизовать необходимо использовать псевдокласс **:first-child**.

Перейдем к примеру:

```
<!DOCTYPEhtml>
<html>
<head>
  <meta charset = "UTF-8">
  <title>Псевдокласс :first-child</title>
<style>
h2:first-child{/* выбираем каждый элемент <h2>, который является первым дочерним элементом своего родителя */
color: blue; /* устанавливаем цвет текста */
}
li:first-child{/* выбираем каждый элемент <li>, который является первым дочерним элементом своего родителя */
color: red; /* устанавливаем цвет текста */
font-size: 24px; /* устанавливаем размер шрифта */
}
</style>
</head>
  <body>
        <h2>Первый заголовок h2 тега body</h2>
        <ul>
              <li>один</li>
              <li>два</li>
              <li>три</li>
        </ul>
        <h2>Второй заголовок h2 тега body</h2>
```

```
            <article>
                  <h2>Первый заголовок h2 тега article</h2>
                  <ul>
                        <li>один</li>
                        <li>два</li>
                        <li>три</li>
                  </ul>
            </article>
      </body>
</html>
```

В этом примере с использованием псевдокласса **:first-child** мы стилизовали элементы <h2> и <li>, которые являются первыми дочерними элементами своих родителей.

Результат нашего примера:

Рис. 22 Пример использования псевдокласса :first-child.

Псевдокласс :last-child

Псевдокласс **:last-child** применяет стиль к элементу в том случае, если элемент является последним дочерним элементом своего родителя.

Этот псевдокласс в отличие от псевдокласса **:first-child** выбирает последний дочерний элемент своего родителя, а не первый.

Давайте рассмотрим пример, в котором перед Вами стоит задача изменить стиль для всех элементов, которые выделены оранжевым цветом на изображении:

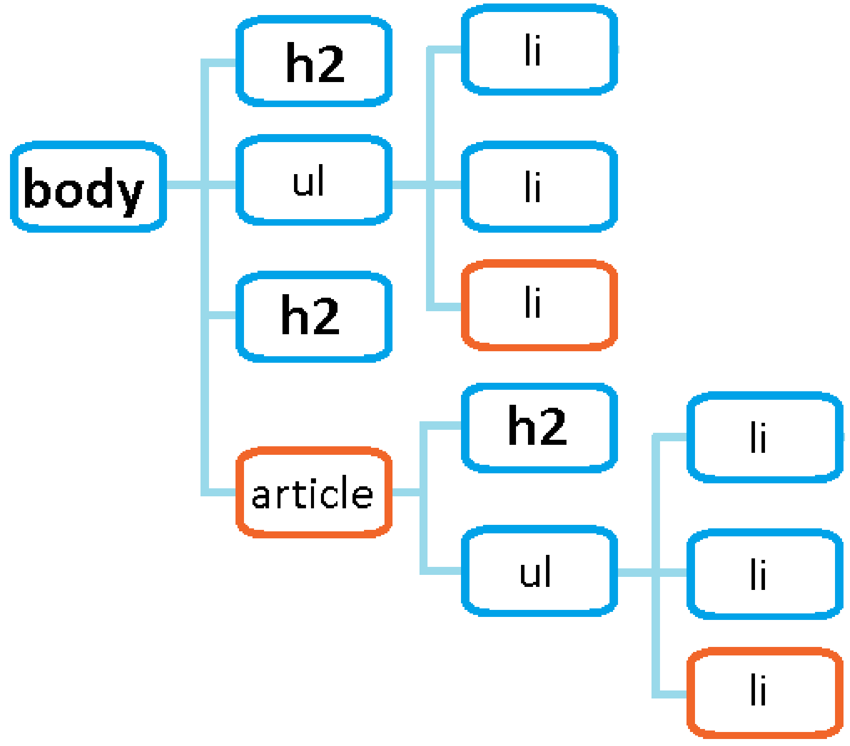


Рис. 23 Пример выбора селектора дочерних элементов.

Что общего у элементов, выделенных на изображении? А общее у них то, что выделенные элементы <article> и <li> являются последними дочерними элементами своих родителей, и чтобы нам их стилизовать необходимо использовать псевдокласс **:last-child**.

Если вы сходу сможете ответить почему ни один элемент <h2> на изображении выше нельзя стилизовать с использованием псевдокласса **:last-child**, то можете сразу перейти к примеру, если нет, то внимательно изучите следующее изображение, оно поможет Вам до конца понять как работает псевдокласс **:last-child**:

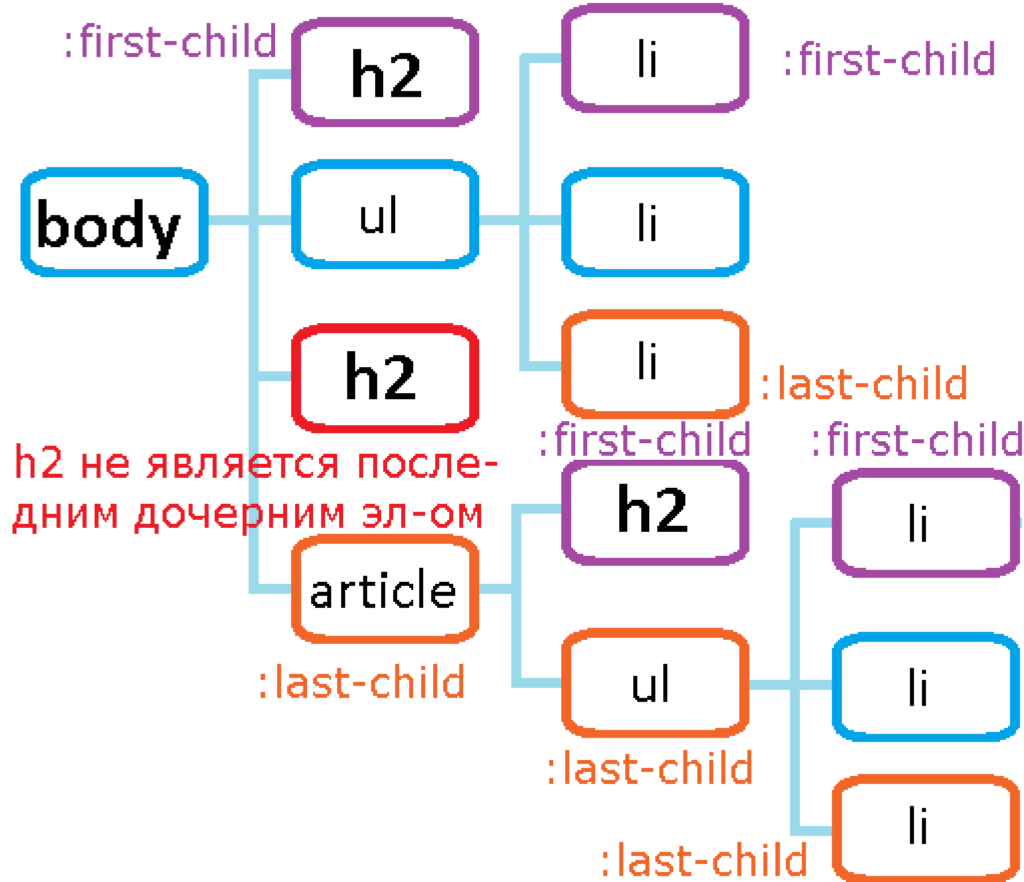


Рис. 23а Схема работы псевдокласса :last-child.

Еще раз поясню, если вы создадите селектор **h2:last-child**, то браузер не найдет этот элемент по той причине, что нет элементов <h2>, которые являются последними дочерними элементами своего родителя, важно это понять, так как подобные вещи иногда ставят людей в ступор.

Перейдем к примеру:

```
<!DOCTYPEhtml>
<html>
<head>
  <meta charset = "UTF-8">
  <title>Псевдокласс :first-child</title>
<style>
article:last-child{/* выбираем каждый элемент <article>, который является последним дочерним элементом своего родителя */
color: blue; /* устанавливаем цвет текста */
```

```
}
li:first-child{/* выбираем каждый элемент <li>, который является последним дочерним элементом своего родителя */
color: red; /* устанавливаем цвет текста */
font-size: 24px; /* устанавливаем размер шрифта */
}
</style>
</head>
  <body>
        <h2>Первый заголовок h2 тега body</h2>
        <ul>
              <li>один</li>
              <li>два</li>
              <li>три</li>
        </ul>
        <h2>Второй заголовок h2 тега body</h2>
        <article>
              <h2>Первый заголовок h2 тега article</h2>
              <ul>
                    <li>один</li>
                    <li>два</li>
                    <li>три</li>
              </ul>
        </article>
  </body>
</html>
```

В этом примере с использованием псевдокласса **:last-child** мы стилизовали элементы **<article>** и **<li>**, которые являются последними дочерними элементами своих родителей.

Результат нашего примера:

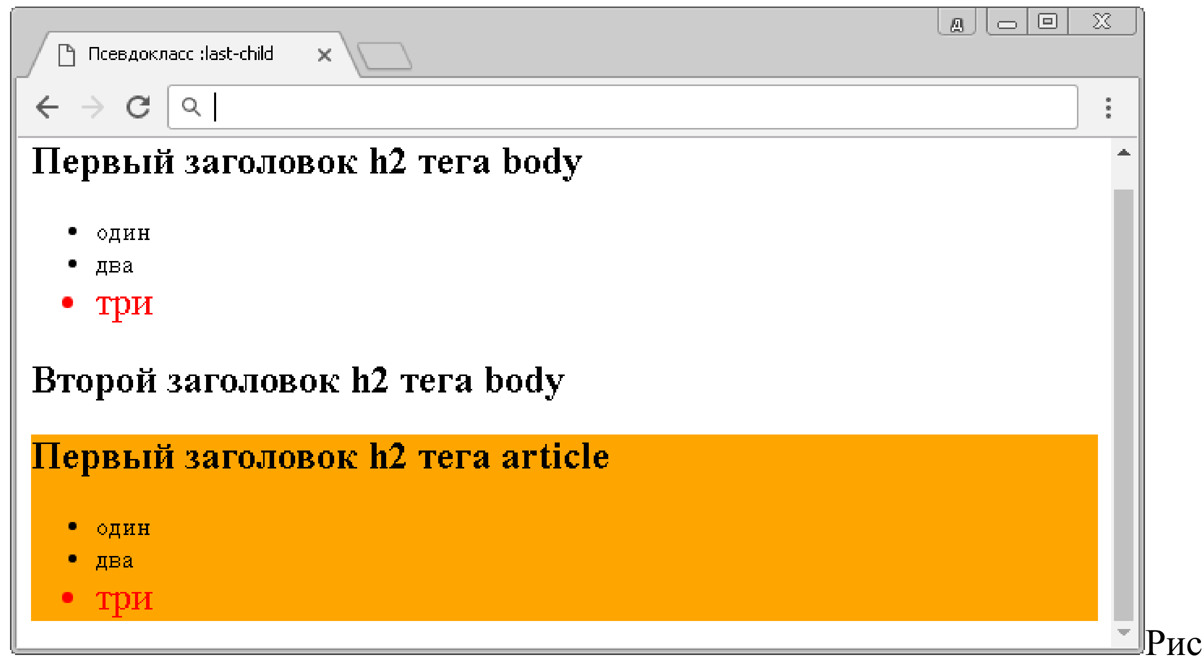


Рис. 236 Пример использования псевдокласса :last-child.

Псевдокласс :nth-child

Стилизация по порядковому номеру

Ну что сказать, у нас остались неоконченные дела, которые касаются стилизации того самого неуловимого элемента <h2>, а на помощь в этом нам придет псевдокласс **:nth-child**.

Псевдокласс **:nth-child** позволяет выбрать дочерние элементы внутри родительского элемента в зависимости от их размещения (порядкового номера). Применение данного псевдокласса широко распространено, он позволяет чередовать стили строк в таблицах, списках, придать стиль сочетанию дочерних элементов и так далее.

Давайте рассмотрим пример, в котором перед Вами стоит задача изменить стиль для всех элементов, которые выделены оранжевым цветом на изображении:

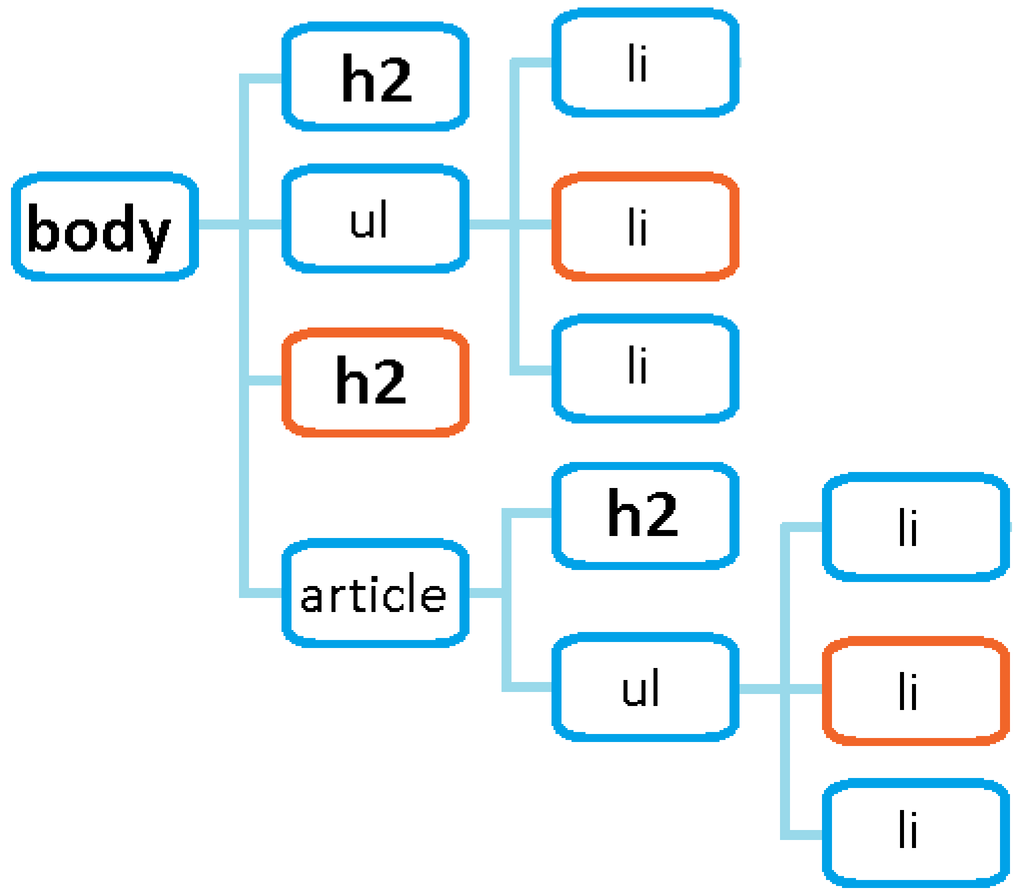


Рис. 24 Пример выбора селектора дочерних элементов.

Что общего у элементов, выделенных на изображении? А общее у них то, что они выделены оранжевым цветом. Смешно? Не думаю. Общее у них все же есть, элементы <li> являются вторыми дочерними элементами своих родителей, а элемент <h2> тоже можно посчитать, и его порядковый номер будет третьим (третий дочерний элемент своего родителя <body>). Чтобы стилизовать эти элементы, нам необходимо использовать псевдокласс **:nth-child**.

Перейдем к примеру:

```
<!DOCTYPEhtml>
<html>
<head>
  <meta charset = "UTF-8">
  <title>Псевдокласс :first-child</title>
<style>
```

```
/* групповой селектор, который выбирает каждый третий дочерний элемент <h2> своего родителя
и каждый второй элемент <li> своего родителя */
h2:nth-child(3), li:nth-child(2){
background-color: orange; /* устанавливаем цвет заднего фона */
}
</style>
</head>
  <body>
        <h2>Первый заголовок h2 тега body</h2>
        <ul>
              <li>один</li>
              <li>два</li>
              <li>три</li>
        </ul>
        <h2>Второй заголовок h2 тега body</h2>
        <article>
              <h2>Первый заголовок h2 тега article</h2>
              <ul>
                    <li>один</li>
                    <li>два</li>
                    <li>три</li>
              </ul>
        </article>
  </body>
</html>
```

В этом примере с использованием псевдокласса **:nth-child** мы стилизовали элементы <h2> и <li>, которые имеют определённый порядковый номер дочернего элемента внутри своих родительских элементов.

Результат нашего примера:

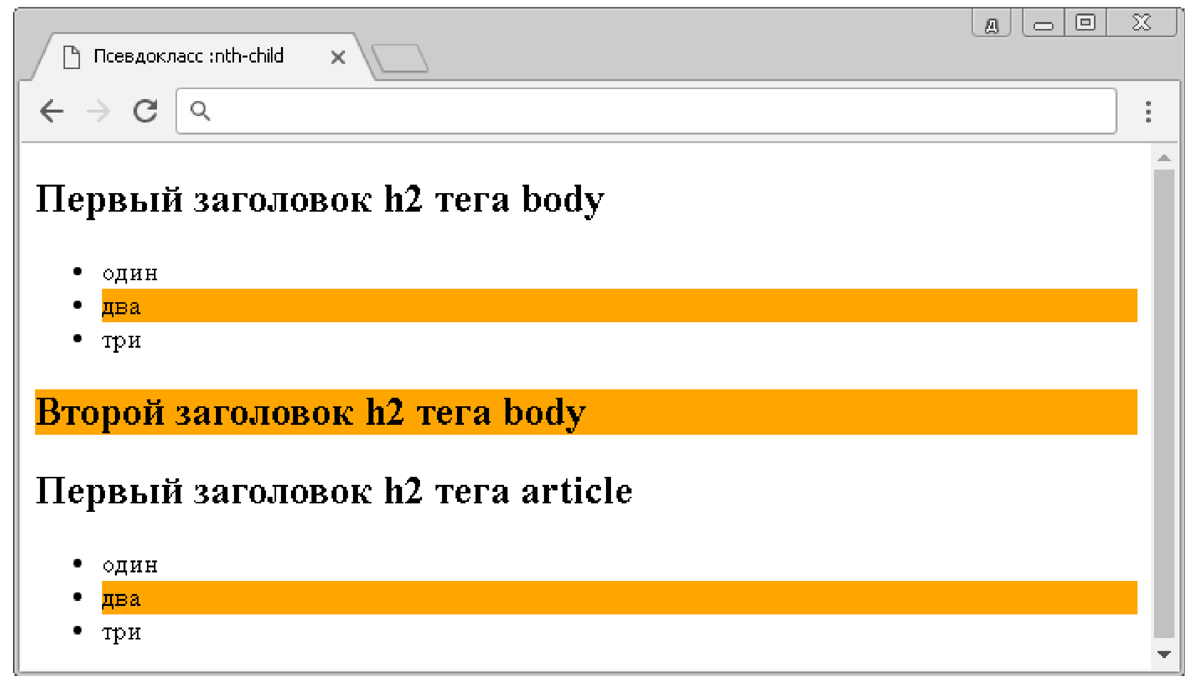


Рис. 24а Пример использования псевдоэлемента :last-child.

Продвинутое использование псевдокласса :nth-child

Стилизация по ключевому слову

В качестве значения псевдокласса **:nth-child** может выступать не только порядковый номер дочерних элементов, которые необходимо стилизовать, но и ключевые слова, которые могут определять целую группу элементов. В качестве ключевого слова можно использовать два значения:

- **even** (четные элементы)
- **odd** (нечетные элементы)

Стилизация элементов с использованием ключевых слов имеет очень широкое применение, так как вы с легкостью можете выбрать и стилизовать четные, либо нечетные дочерние элементы в документе.

Давайте для примера создадим две простые таблицы с разными стилевыми классами и рассмотрим наглядно разницу в применении значений ключевых слов псевдокласса **:nth-child** для HTML элемента <tr>, который определяет строку таблицы:

```
<!DOCTYPE html>
<html>
```

```
<head>
  <meta charset = "UTF-8">
  <title>Стилизация четных и нечетных дочерних элементов</title>
<style>
.primer1 tr:nth-child(even) {/* стилизация четных дочерних элементов */
background-color: #AAA; /* устанавливаем цвет заднего фона */
}
.primer2 tr:nth-child(odd) {/* стилизация нечетных дочерних элементов */
background-color: #AAA; /* устанавливаем цвет заднего фона */
}
caption{/* селектор типа (выбираем HTML элемент <caption>) */
color: red; /* устанавливаем цвет текста */
}
</style>
</head>
  <body>
        <table class = "primer1">
              <caption>Значение even (четные)</caption>
              <tr>
                    <th>1 строка</th><th>Позиция</th><th>Количество</th>
              </tr>
              <tr>
                    <td>2 строка</td><td></td><td></td>
              </tr>
              <tr>
                    <td>3 строка</td><td></td><td></td>
              </tr>
```

```
				<tr>
					<td>4 строка</td><td></td><td></td>
				</tr>
				<tr>
					<td>5 строка</td><td></td><td></td>
				</tr>
			</table>
			<table class = "primer2">
				<caption>Значение odd (нечетные)</caption>
				<tr>
					<th>1
строка</th><th>Позиция</th><th>Количество</th>
				</tr>
				<tr>
					<td>2 строка</td><td></td><td></td>
				</tr>
				<tr>
					<td>3 строка</td><td></td><td></td>
				</tr>
				<tr>
					<td>4 строка</td><td></td><td></td>
				</tr>
				<tr>
					<td>5 строка</td><td></td><td></td>
				</tr>
			</table>
		</body>
	</html>
```

В этом примере с использованием псевдокласса **:nth-child** мы стилизовали *четные* строки первой таблицы (элементы <tr>) и *нечетные* во второй таблице.

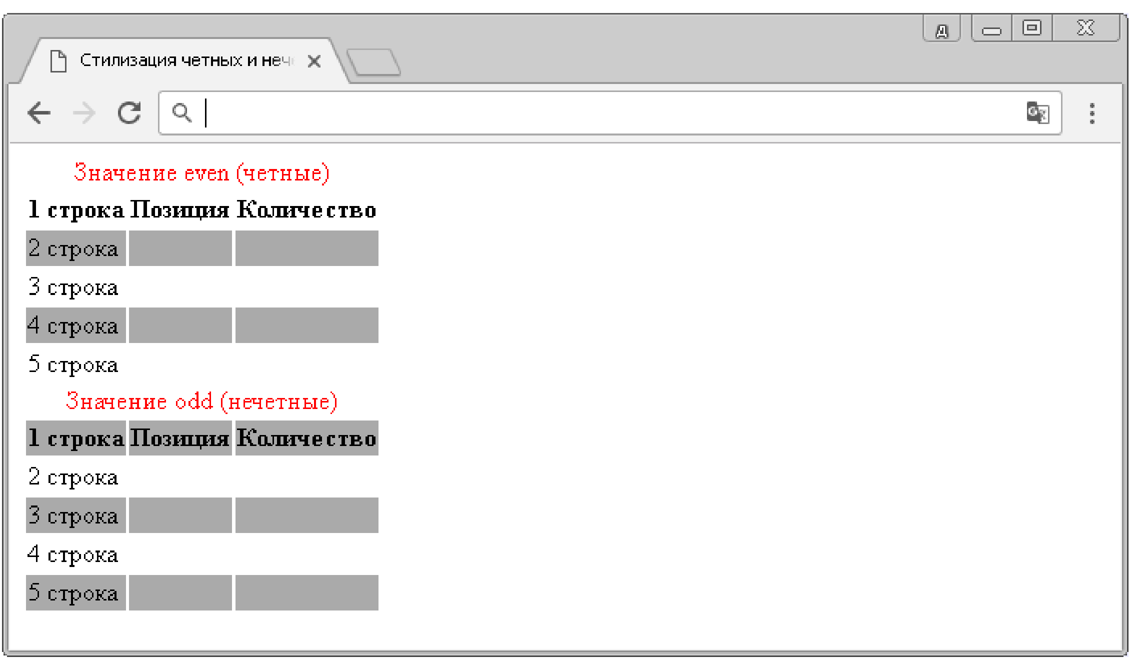


Рис. 246 Пример стилизации четных и нечетных дочерних элементов.

Стилизация по простой математической формуле

Псевдокласс **:nth-child** позволяет выбрать не только чётные, нечетные, или дочерние элементы с определённым порядковым номером, но и дочерние элементы, заданные по элементарной математической формуле. Давайте рассмотрим следующий селектор и разберем, что значит эта запись:

```
td:nth-child(4n+2) {
background-color: lightblue; /* устанавливаем цвет заднего фона */
}
```

Этот селектор означает, что каждая четвёртая ячейка таблицы (<td>) внутри строки, начиная со второй ячейки таблицы, будет стилизована:

- **4n** – каждый четвертый элемент.
- **2** – с какого элемента начинать.

В формулах допускается использование значений со знаком вычитания, но в этом как правило нет необходимости:

```
td:nth-child(4n-1) {
background-color: lightblue; /* устанавливаем цвет заднего фона */
```

```
}
```

Этот селектор означает, что каждая четвёртая ячейка таблицы (<td>) внутри строки, начиная с третьей ячейки таблицы (-1 ячейки нет по объективным причинам, поэтому происходит сдвиг влево), будет стилизована:

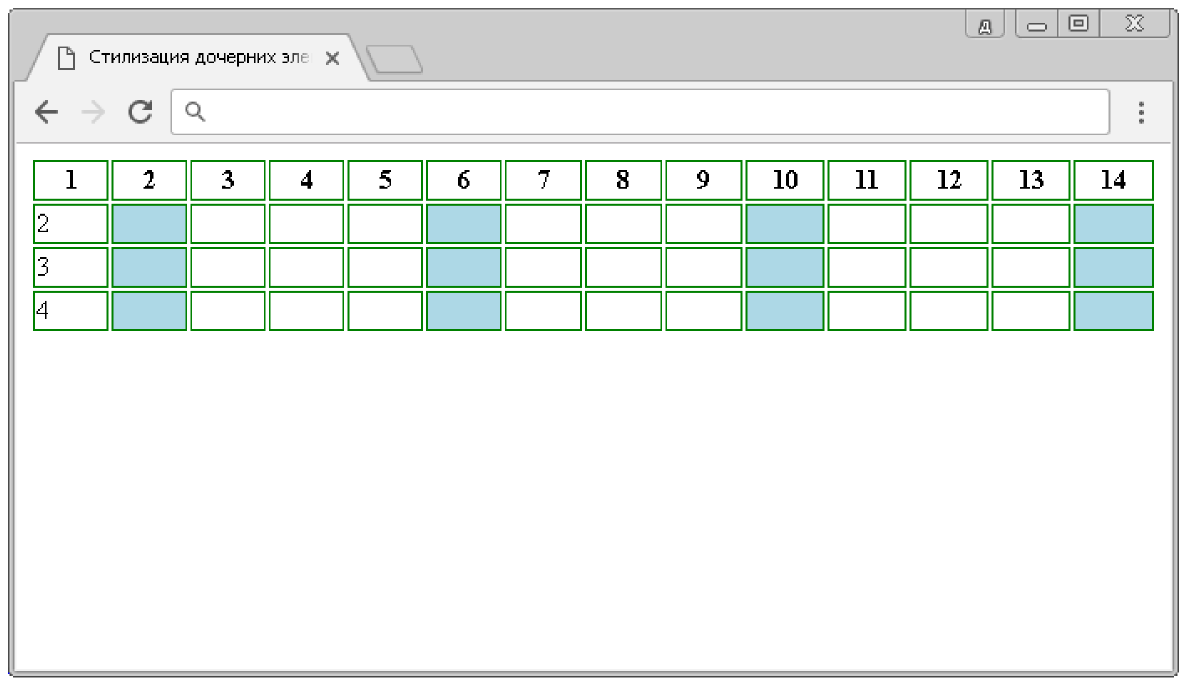
- **4n** – каждый четвертый элемент.
- **-1** – с какого элемента начинать.

Давайте рассмотрим пример использования:

```
<!DOCTYPEhtml>
<html>
<head>
  <metacharset = "UTF-8">
  <title>Стилизация дочерних элементов по математической формуле</title>
<style>
td, th{/* групповым селектором выбираем заголовочные ячейки и ячейки данных */
border: 1px solidgreen; /* задаём сплошную границу размером 1 пиксель зеленого цвета */
width: 50px;  /* устанавливаем ширину заголовочным ячейкам и ячейкам данных */
}
td:nth-child(4n+2) {
background-color: lightblue; /* устанавливаем цвет заднего фона */
}
</style>
</head>
  <body>
        <table>
              <tr>
```

```
			<th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th>
		</tr>
		<tr>
			<td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td>
		</tr>
		<tr>
			<td>3</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td>
		</tr>
		<tr>
			<td>4</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td>
		</tr>
	</table>
</body>
</html>
```

В этом примере с использованием псевдокласса **:nth-child** мы выбрали и стилизовали каждую четвёртую ячейку таблицы (<td>) внутри строки, начиная со второй ячейки таблицы. Результат нашего примера:



1	2	3	4	5	6	7	8	9	10	11	12	13	14
2													
3													
4													

Рис. 25 Пример стилизации дочерних элементов по математической формуле.

Глава 15. Наследование и каскадность

Наследование

Что касается наследования в CSS, то это не что иное, как **метод тиражирования различных CSS свойств, относящихся к одному элементу страницы на вложенные в него элементы (потомки).**

Давайте сразу перейдем к примеру и рассмотрим наследование стилей на примере HTML элемента <body>, который определяет видимое содержимое страницы.

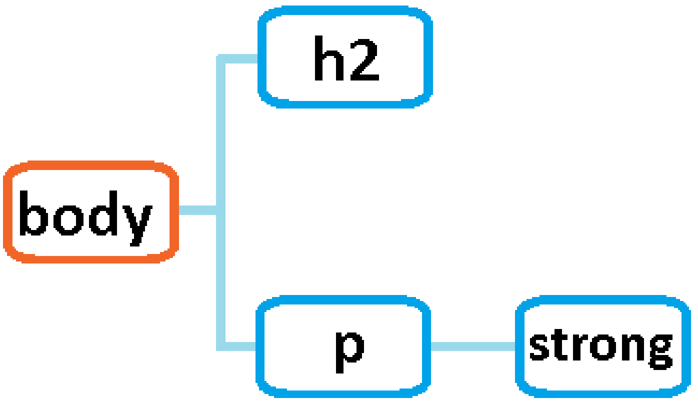


Рис.28 Схема наследования стиля в CSS.

Создадим стиль для элемента **<body>**, который будет изменять цвет и тип шрифта:

```
<!DOCTYPE html>
<html>
<head>
  <metacharset = "UTF-8">
  <title>Пример наследования стиля в CSS</title>
<style>
body{/* используем селектор типа */
color: green; /* устанавливаем цвет текста */
font-family: arial; /* устанавливаем тип шрифта */
}
</style>
</head>
<body>
  <h2>Заголовок второго уровня</h2>
  <p>Полный<strong>абзац</strong></p>
</body>
</html>
```

В этом примере для элемента **<body>** мы установили *зеленый* цвет текста и тип шрифта *Arial*. CSS свойства **color** и **font-family**наследуются, а это означает, что эти свойства будут применяться и на вложенные элементы внутри **<body>** (на все его потомки).

Обращаю Ваше внимание на то, что вы всегда можете посмотреть наследуется или нет конкретное свойство в полном **справочнике CSS**.

Результат нашего примера:

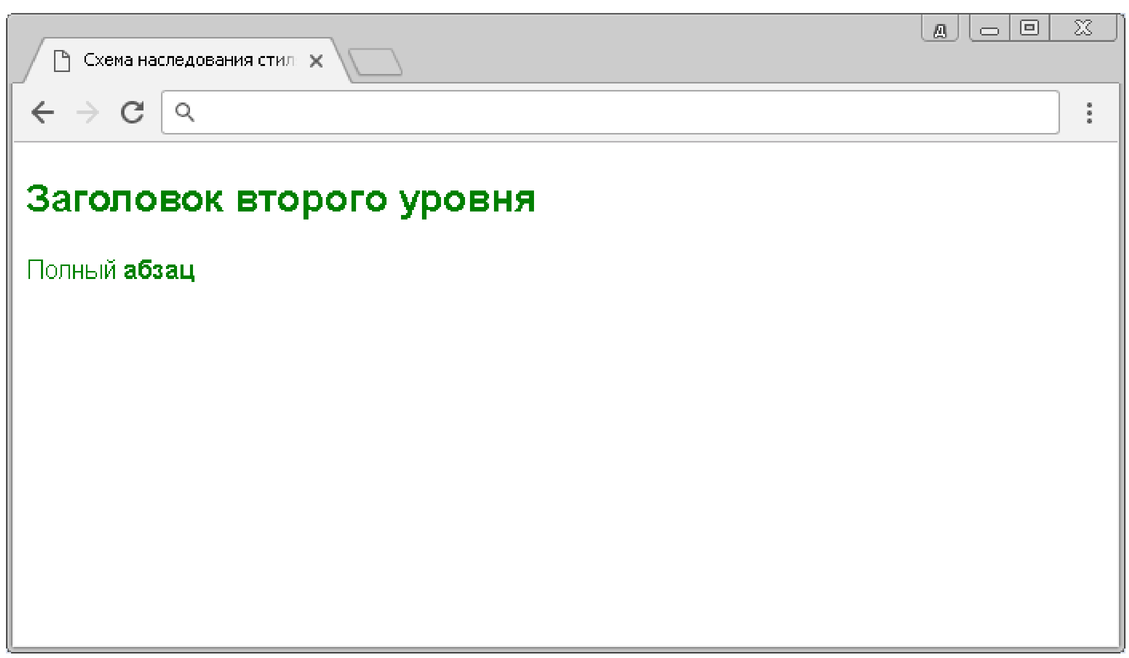


Рис. 29 Пример наследования стиля в CSS.

В выше рассмотренном примере все элементы, расположенные внутри <body> (его потомки) унаследовали его свойства. Механизм наследования имеет многоуровневую систему и распространяется не только на прямых потомков элемента, но и переносится на все вложенные элементы. В нашем примере к такому элементу относится элемент <strong>, который по аналогии с другими элементами унаследовал все свойства стиля, заданного для элемента <body>. В этом заключается основной смысл наследования, который используется в CSS.

Механизм наследования значительно сокращает код CSS, например, если бы наш элемент <strong> получил цвет по умолчанию – чёрный, то нам пришлось бы отдельно для этого элемента устанавливать стиль, который бы определял как цвет, так и тип шрифта, что значительно увеличивало бы трудозатраты на разработку конкретной страницы.

Еще один момент, который обязательно необходимо понять, это то, что аналогично действует механизм наследования не только для селекторов типа, но и для всех типов селекторов, рассмотренных в предыдущих статьях, посвященной этой тематике. Допустим, мы создали селектор класса с аналогичными CSS свойствами и применили его к элементу <body>, то в этом случае все элементы, вложенные в него также унаследуют эти свойства.

Вы будете в дальнейшем использовать эти особенности CSS для упрощения Вашей работы по созданию стилей для элементов документа.

Ограничения и нюансы наследования

Ограничения:

- Не наследуются свойства, которые влияют на размещение элементов на странице, свойства отвечающие за внешние и внутренние отступы элемента, свойства отвечающие за границы элементов. Все эти свойства мы подробно рассмотрим далее в учебнике. Повторюсь, что вы всегда можете посмотреть наследуется или нет конкретное свойство на сайте в **справочнике CSS**.
- Я хочу, чтобы вы поняли, что многие свойства не наследуются по объективным причинам, представьте, что мы создаем границу для родительского элемента и после этого все потомки по этой логике должны унаследовать это свойство, что выглядело бы абсурдно и напротив увеличивало бы работу по созданию стилей (их отмене, или сбросу).

Нюансы:

Все современные браузеры используют собственные встроенные CSS стили для HTML элементов, эти стили, при необходимости, вы можете посмотреть у конкретного элемента на сайте в **справочнике HTML** (значение CSS по умолчанию). В следующей статье мы научимся обнулять встроенные стили для отображения ваших страниц одинаково во всех популярных браузерах.

А сейчас на примере элемента <a>, определяющего гиперссылку, мы рассмотрим пример в котором рассмотрим почему некоторые элементы не наследуют некоторые свойства своего предка:

```
<!DOCTYPE html>
<html>
<head>
  <metacharset = "UTF-8">
  <title>Нюансы наследования стилей</title>
```

```
<style>
body{/* используем селектор типа */
color: green; /* устанавливаем цвет текста */
text-decoration: none; /* убираем декорирование текста (нижнее подчеркивание) */
}
</style>
</head>
<body>
<p>Абзац, который содержит внутри себя <a href = "#">гиперссылку</a>.</p>
</body>
</html>
```

В этом примере для элемента <body> мы установили следующие стили: *зеленый* цвет текста и отсутствие декорирования текста (убрали нижнее подчеркивание снизу). Обратите внимание на элемент <a> на изображении, он полностью не изменился. Давайте разберемся ниже (после просмотра изображения) почему так происходит.

Результат нашего примера:

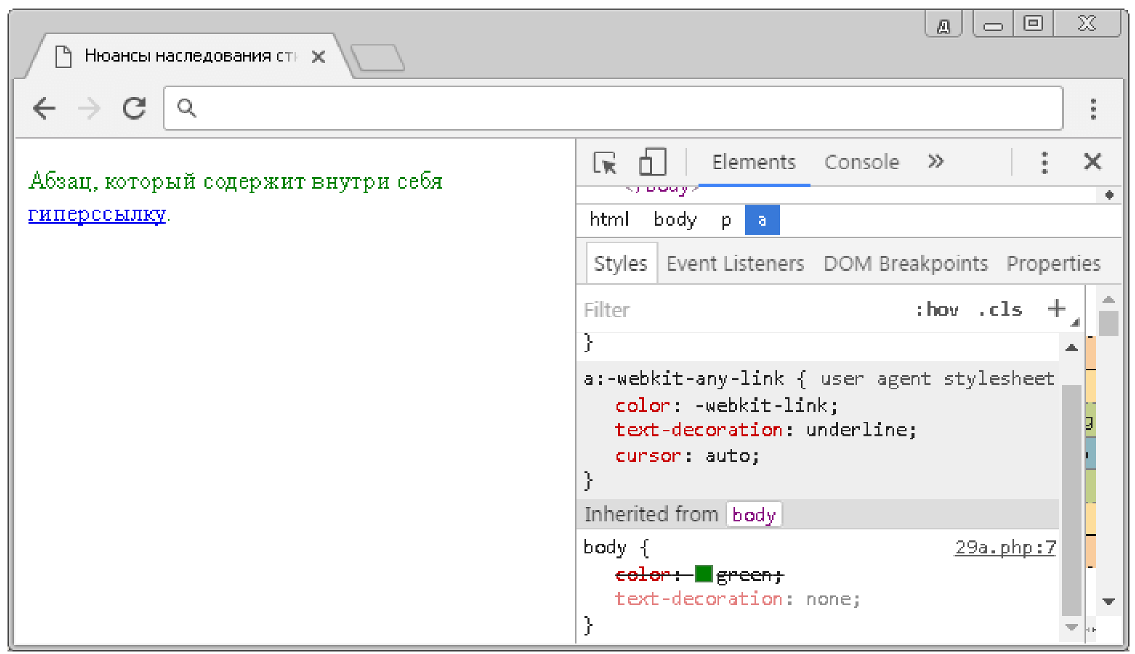


Рис. 29а Нюансы наследования стилей.

При возникновении конфликта побеждает сильнейший, в CSS это, как правило, явно определенный стиль. Откроем инструменты разработчика (для Chrome это **F12**). Обратите внимание какие встроенные в браузер CSS свойства (*useragentstylesheet*) имеет любая ссылка (*anylink*) в документе. Для неё заданы следующие стили: цвет текста *синий* (*-webkit-link* - значение браузера по умолчанию), декорирование текста (нижнее подчеркивание) и определено, что браузер устанавливает курсор автоматически (значение **auto** CSS свойства **cursor**).

Ниже отображаются свойства, которые были унаследованы (*Inheritedfrom*) элементом <a> от <body>. Как вы можете заметить, браузер отбросил все стили элемента <body>, по той причине, что у элемента <a> стиль, который определяет цвет текста явно определен (встроенный стиль по умолчанию), а свойство, которое определяет декорирование текста **не наследуется** и отображается с небольшой прозрачностью в инструментах разработчика, но даже если бы оно наследовалось, то не применилось бы по аналогии с цветом теста (у элемента <a> это свойство тоже явно определно).

Такие конфликты между стилями всегда разрешает браузер, а по каким правилам определяет, кому отдает приоритет и как он в том, или ином случае себя должен повести, на чью сторону встать, определяется правилами **каскадности**. Подробнее о правилах каскадности мы поговорим в следующей статье учебника.

Каскадность

Прежде всего, что такое каскадность? **Каскадность** – это правила, по которым определяется какие стилевые свойства задаются браузером элементам на странице (последовательность применения стилей к определённым элементам и разрешение при необходимости, возникающих конфликтов).

Давайте рассмотрим следующее изображение, на нем отображены основные источники информации о стилях, которые образуют каскад:

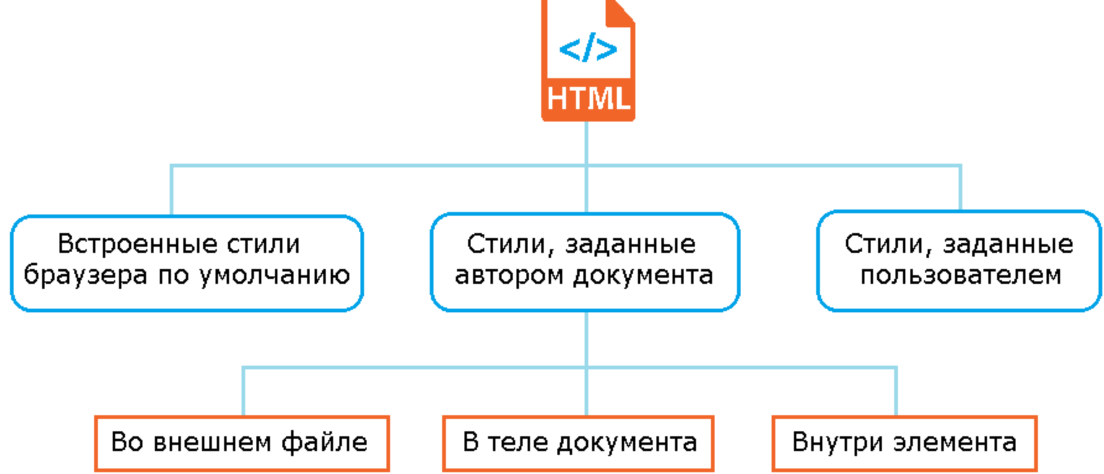


Рис. 30 Основные источники информации о стилях.

К основным источникам информации о стилях относятся:

- **Встроенные стили браузера** по умолчанию.
- **Стили, заданные автором документа**, их в свою очередь можно указать в трех местах:
    - *Внешний CSS файл* (с использованием элемента <link>, который размещается внутри элемента <head>).
    - *В теле документа* (в настоящее время разрешается размещать стили только в начале документа - стили помещаются внутри парного тега <style>, который в свою очередь должен находиться внутри элемента <head>).
    - *Внутри элемента* (стили, которые задаются с использованием глобального HTML атрибута style).
- **Стили, заданные пользователем** (в некоторых современных браузерах возможность определить свои стили для конкретного сайта доступна встроенными средствами в настройках, для других требуется расширение для браузера). Это позволит стилизовать какой-то сайт по своему усмотрению и вкусу, или убрать с него какие-то блоки, которые Вам не хотелось бы видеть.

Наименьшим приоритетом из этого списка обладают встроенные стили браузера, а для пользовательских устанавливается **наивысший приоритет**, но это нас не должно интересовать, ведь кто как хочет, так и стилизует, и это не предмет этой статьи учебника.

Главное надо понять, что если на странице применяется несколько стилей к одному элементу, то браузер объединит свойства этих стилей, при условии, что они не конфликтуют между собой (имеют различные значения для однотипных свойств). А если конфликтуют?

Давайте перейдем к практической части.

Предположим, что у нас есть абзац (элемент <p>), в котором указана определенная гиперссылка (элемент <a>). HTML код может выглядеть следующим образом:

```
<pclass = "main"> Для перехода к основной статье нажмите <ahref = "#"> вот на это место. </a></p>
```

На нашу страницу добавим следующие стили:

```
a{/* используем селектор типа */
color: brown; /* устанавливаем цвет текста */
}
p a{/* используем селектор потомков */
font-weight: bold; /* устанавливаем жирное начертание шрифта */
}
.main a{/* используем селектор потомков */
background-color: orange; /* задаем цвет заднего фона */
text-decoration: none; /* убираем декорирование текста (нижнее подчеркивание) */
}
```

Создадим разметку и добавим стили в наш документ:

```
<!DOCTYPE html>
```

```
<html>
<head>
  <metacharset = "UTF-8">
  <title>Каскадностьв CSS</title>
<style>
a{/* используем селектор типа */
color: brown; /* устанавливаем цвет текста */
}
p a{/* используем селектор потомков */
font-weight: bold; /* устанавливаем жирное начертание шрифта */
}
.main a{/* используем селектор потомков */
background-color: orange; /* задаем цвет заднего фона */
text-decoration: none; /* убираем декорирование текста (нижнее подчеркивание) */
}
</style>
</head>
  <body>
        <pclass = "main">Для перехода к основной статье нажмите <a href = "#">вот на это место. </a></p>
  </body>
</html>
```

Результат нашего примера:

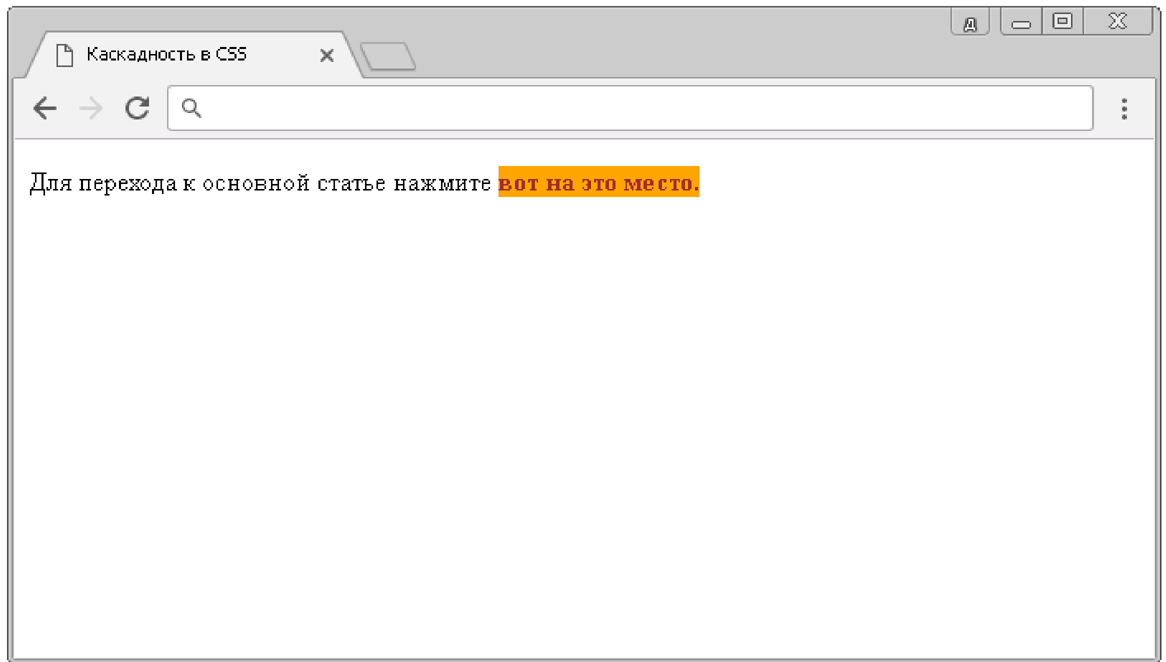


Рис. 30а Каскадность в CSS.

Давайте рассмотрим какие стили были применены к элементу <a>:

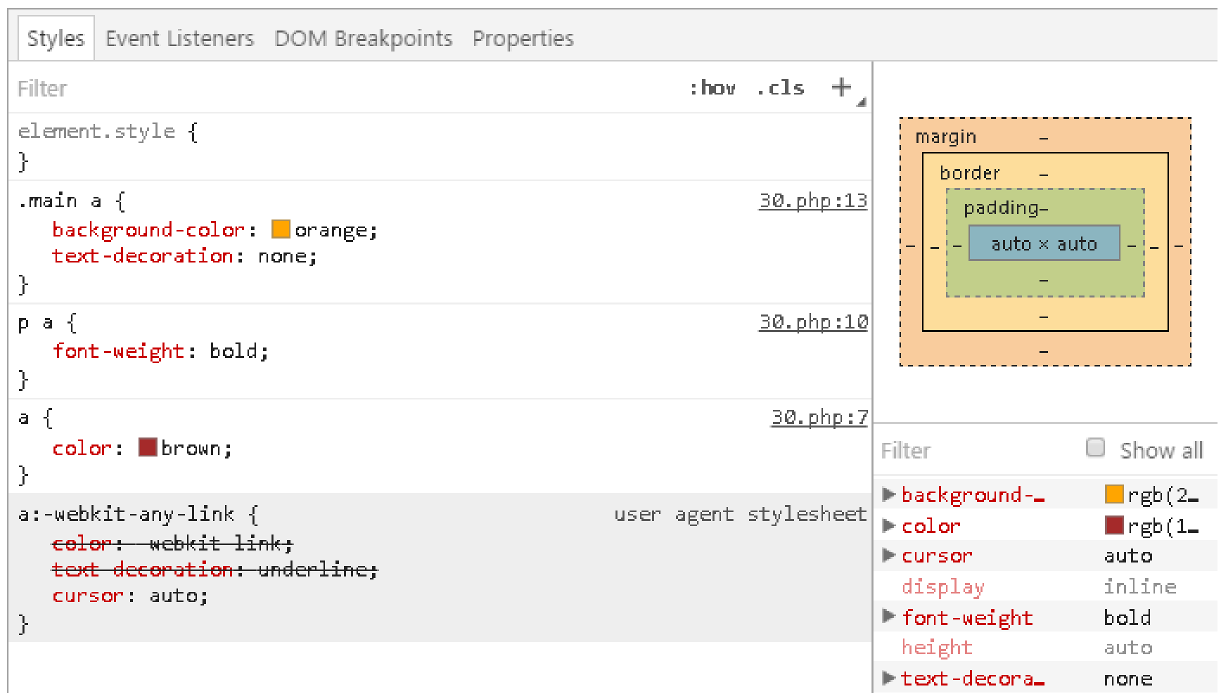


Рис. 30б Применение стилей к элементу.

В данном примере вы можете увидеть, что все три наших селектора с различными стилями были применены к элементу <a>, а встроенные стили элемента - цвет и декорирование текста (нижнее подчеркивание) были заменены на авторские стили (стили автора документа) по той причине, что они имеют больший приоритет.

Система приоритетов в CSS

Давайте рассмотрим пример, где не всё так очевидно и однозначно. К примеру, с использованием свойства **font-family** зададим элементу **<a>** различный тип шрифта с использованием аналогичных селекторов:

```
<!DOCTYPE html>
<html>
<head>
  <metacharset = "UTF-8">
  <title>Конфликты стилей в CSS</title>
<style>
a{/* используем селектор типа */
color: brown; /* устанавливаем цвет текста */
font-family: Arial; /* устанавливаемтипшрифта Arial */
}
p a{/* используем селектор потомков */
font-weight: bold; /* устанавливаем жирное начертание шрифта */
font-family: Helvetica; /* устанавливаемтипшрифта Helvetica */
}
.main a{/* используем селектор потомков */
background-color: orange; /* задаем цвет заднего фона */
text-decoration: none; /* убираем декорирование текста (нижнее подчеркивание) */
font-family: Courier; /* устанавливаемтипшрифта Courier */
}
</style>
</head>
  <body>
        <pclass = "main">Для перехода к основной статье нажмите <a href = "#">вот на это место. </a></p>
  </body>
```

```
</html>
```

Результат нашего примера:

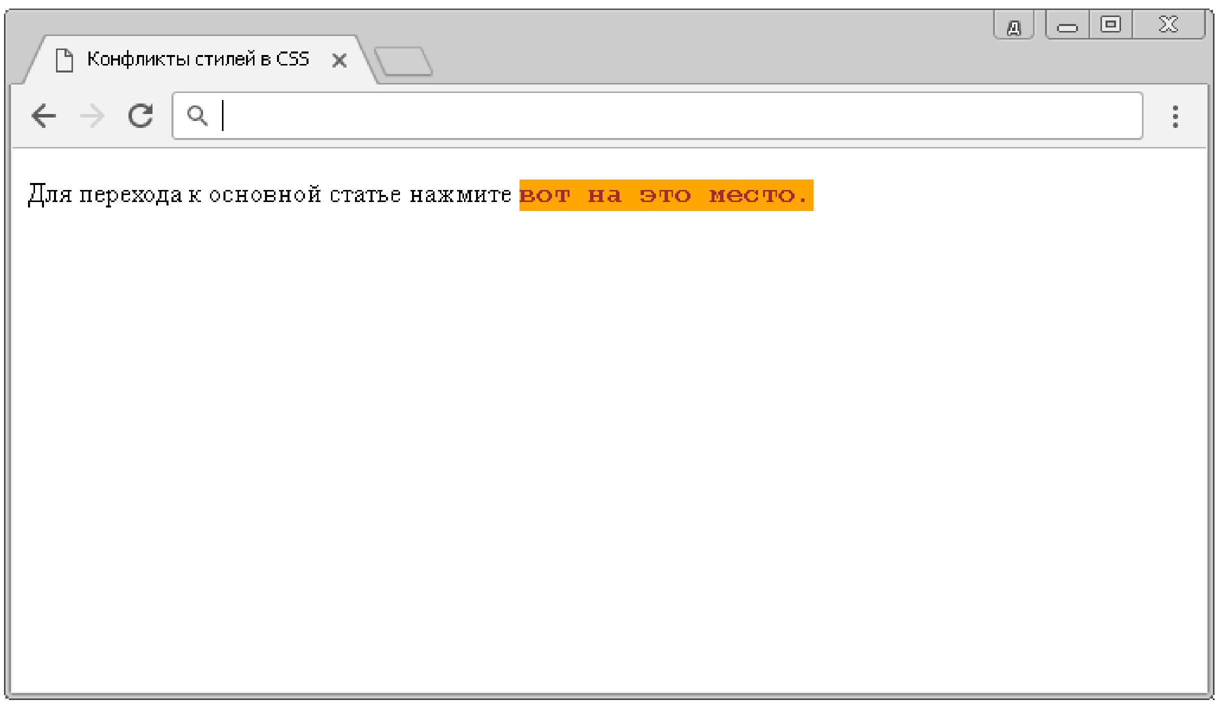


Рис. 31 Пример конфликтования стилей.

Давайте рассмотрим какой шрифт получил элемент <a>:

Рис. 31а Выбор типа шрифта при конфликте стилей.

Обратите внимание, что для нашего элемента был установлен шрифт **Courier**. Как мы видим в "инструментах разрабочика" браузера *Chrome* этот шрифт выбран из селектора потомков в котором используется как селектор

класса, так и селектор типа (**.main a**), а в остальных селекторах тип шрифта для элемента перечеркнут. Но почему?

Я уже обращал Ваше внимание на тот факт, что основное правило механизма каскадности - побеждает тот, у кого установлен самый явно определенный стиль. Для этих целей в CSS существует своя система приоритетов, которая основана на присвоении значений в пунктах для каждого типа селекторов, чем выше это значение, тем выше его значимость:

- Селектор типа, псевдоэлементы — **1 пункт**.
- Селектор класса, псевдоклассы — **10 пунктов**.
- Id селектор — **100 пунктов.**
- Встроенный (*inline*) стиль (стиль задается в самом элементе с использованием глобального HTML атрибута **style**) — **1000 пунктов**.

Давайте перейдем к следующему примеру в котором рассмотрим как работает система приоритетов:

```
<!DOCTYPE html>
<html>
<head>
  <metacharset = "UTF-8">
  <title>Работа системы приоритетов в CSS</title>
<style>
#id_invite{/* id селектор — 100 пунктов */
color: red; /* устанавливаем цвет текста красный */
}
.class_invite{/* селектор класса — 10 пунктов */
color: blue; /* устанавливаем цвет текста синий */
}
p{/* селектор типа — 1 пункт */
color: green; /* устанавливаем цвет текста зеленый */
```

```
        }
        </style>
        </head>
          <body>
                    <!-- устанавливаем цвет текста внутри элемента span
зеленый (inline стиль) -->
                    <pclass = "class_invite"id = "id_invite">Просто<spanstyle =
"color: green;">добавь</span>воды.</p>
          </body>
        </html>
```

В данном примере для элемента <p> был установлен красный цвет текста благодаря **id селектору**, который имеет более высокое значение в пунтах чем другие селекторы (**100**). Кроме того, для демонстрации системы приоритетов мы применили встроенный (**inline**) стиль для элемента <span> и установили для него зеленый цвет шрифта. Обратите внимание, что на изображении, все значения селекторов перечеркнуты, так как встроенный стиль имеет самое высокое значение в пунктах (**1000**).

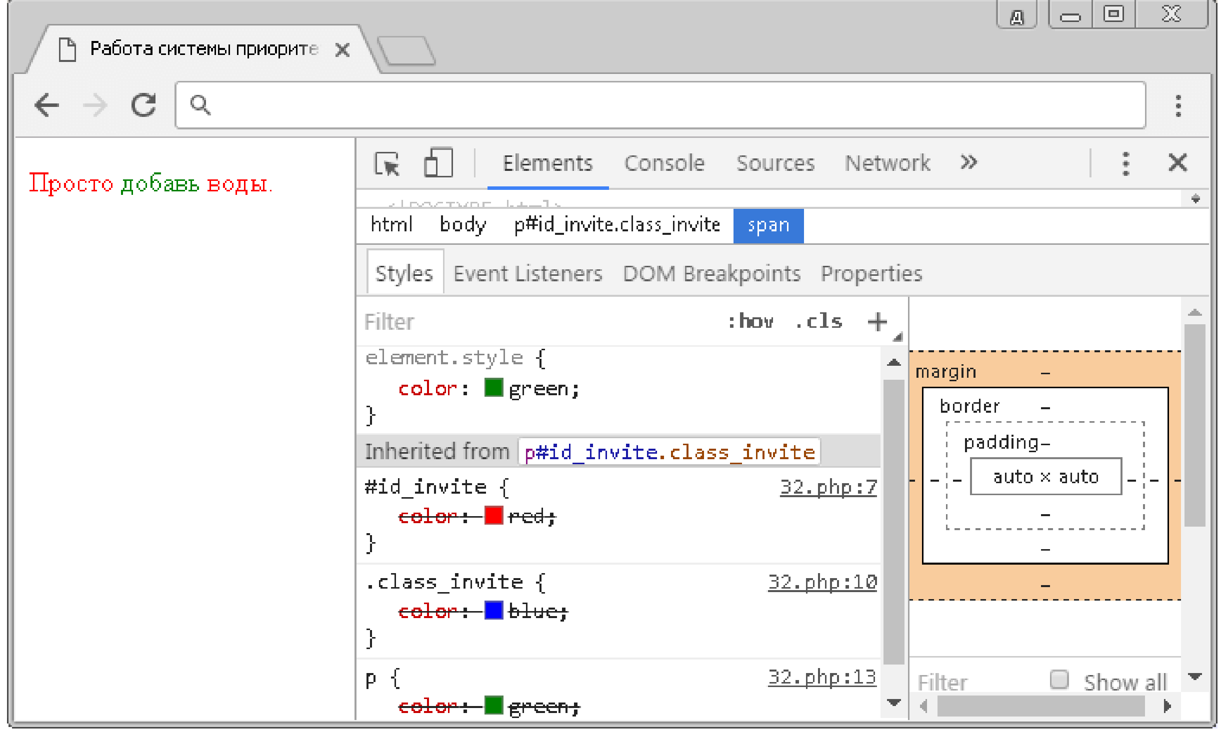


Рис 32 Пример системы приоритетов в CSS.

Чтобы подсчитать специфичность комбинированных селекторов необходимо их просто сложить между собой, например:

```
#id_invite a{/* id селектор (100 пунктов) + селектор типа (1 пункт) = 101 пункт */
блок объявлений;
}
p:first-letter{/* селектор типа (1 пункт) + псевдоэлемент (1 пункт) = 2 пункта */
блок объявлений;
}
.main:first-child{/* селектор класса (10 пунктов) + псевдокласс (10 пунктов) = 20 пунктов */
блок объявлений;
}
```

Отмена значимости стилей

В некоторых случаях, два стиля могут иметь одинаковый приоритет (одинаковое количество пунктов), что доставляет сильную головную боль по той причине, что браузер в этом случае выбирает **последний определенный стиль** в таблице стилей, что может не соответствовать вашим ожиданиям. Если вы поменяете местами конфликтующие стили, то действовать будет тот, который будет указан ниже в таблице стилей. Старайтесь избегать подобных ситуаций и запомните этот момент. Эти знания помогут Вам сэкономить большое количество времени.

В CSS предусмотрена возможность отменить значимость стилей (не учитывать количество пунктов). Для этого необходимо добавить к значению CSS свойства ключевое слово **!important**.

Давайте рассмотрим пример использования ключевого слова **!important**:

```
<!DOCTYPE html>
<html>
```

```
<head>
  <metacharset = "UTF-8">
  <title>Пример отмены значимости стилей</title>
<style>
a.rtfm{/* селектор типа (1 пункт) + селектор класса (10 пунков) = 11 пунктов  */
color: green; /* устанавливаем цвет текста зеленый */
}
a{/* селектор типа (1 пункт)  */
color: red!important; /* отменяем значимость стилей и устанавливаем цвет текста красный */
}
</style>
</head>
  <body>
        <aclass = "rtfm" href = "http://google.com">Найти</a>
  </body>
</html>
```

В этом примере с использованием ключевого слова **!important** мы отменили значимость стилей и установили цвет текста для гиперссылки *красный*. В данном случае если бы мы не использовали ключевое слово **!important**, то цвет бы остался *зеленым* по той причине, что он имеет большую значимость (задан с использованием селектора класса).

Результат нашего примера:

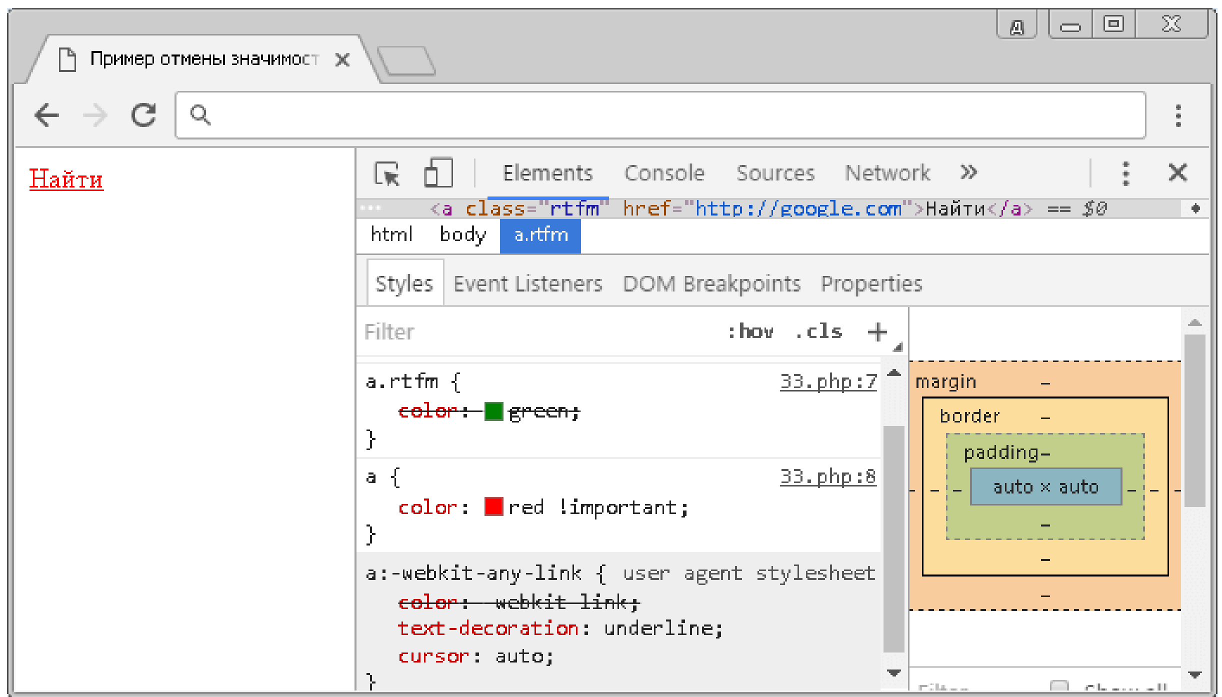


Рис. 33 Пример отмены значимости стилей.

Обратите внимание, что !important указывается в конце каждого свойства и действует только на одно свойство, а не на весь блок объявлений!

Если вы указали значение **!important** для двух однотипных свойств различных стилей, то в этом случае браузер рассматривает их по принципу правил значимости (приоритет отдаётся более значимому свойству). Старайтесь избегать подобных ситуаций.

Сброс и нормализация встроенных стилей.

Ещё один необходимый метод для грамотного построения кроссбраузерных страниц - это необходимость сброса встроенных в браузер внутренних стилей (сброс стандартных стилей браузера). Другими словами, независимо какой браузер у пользователей, которые посещают Ваш сайт, отображение должно быть в одном стиле - с одинаковыми отступами, границами и форматированием.

Для этого существуют различные шаблоны, которые сбрасывают внутренние стили (**reset.css**). Как правило, шаблоны адаптируются под себя

(под Ваши нужды) и служат для удаления полей и отступов, устанавливая например 100% размер шрифта, убирая выделения полужирным шрифтом, устанавливают пространство между строками в абзаце, убирают маркеры в списках и тому подобное.

В настоящее время часто в своей работе используют альтернативу традиционному **reset.css** (сбросу внутренних стилей), нормализовывая таблицы стилей - **normalize.css**.

Данный проект (**normalize.css**) появился после глубокого исследования различий между изначальными стилями браузера под руководством *Николаса Галахера*. Основные задачи **normalize.css** заключаются в том, чтобы сохранить полезные настройки браузера, а не стирать их, и при этом нормализовать стили для широкого круга HTML элементов.

Как вы понимаете, **normalize.css** значительно отличается от **reset.css**. Впоследствии, Вам рекомендую попробовать в своей работе оба метода, чтобы определиться, соответствует ли конкретный метод вашим предпочтениям в разработке.

Глава 16. Работа со шрифтами

Для привлечения внимания посетителей Вашего сайта вы можете придать привлекательный вид текстовому содержимому страниц. Для этих целей в CSS существует большое количество разнообразных свойств форматирования: шрифт текста, его цвет, размер, межстрочный интервал и так далее. В первую очередь рассмотрим методы работы с существующими шрифтами (безопасные веб-шрифты).

Безопасные веб-шрифты

В CSS стиле для выбора типа шрифта применяется свойство **font-family**, в котором указывается интересующий Вас шрифт. Предположим, что вы хотите применить для абзацев страницы шрифт *Courier*. В этом случае

Вам необходимо будет создать, например, селектор типа и воспользоваться свойством **font-family**:

```
p{
font-family :Courier;/* устанавливаемтипшрифта – Courier */
}
```

Главная особенность данного способа заключается в том, что он будет работать, при условии, что у посетителя установлен подобный шрифт, иначе, страница будет отображена с использованием шрифта "встроенного" в браузер.

Так как вы заранее не знаете, есть у пользователя тот, или иной шрифт, то рекомендуется указывать не только основной шрифт, но и пару запасных (альтернативных) шрифтов, для того случая если у пользователя отсутствует основной шрифт.

Рекомендуется последним в списке шрифтов указывать и семейство шрифта (**generic-family**). Если у пользователя по каким-то причинам отсутствуют все перечисленные Вами шрифты, то в этом случае страница будет отображена хотя бы шрифтом того же семейства, а не шрифтом "встроенным" в браузер.

Давайте рассмотрим следующий пример:

```
<!DOCTYPEhtml>
<html>
<head>
  <metacharset = "UTF-8">
  <title>Свойство font-family</title>
<style>
.times{
```

```
font-family: "Times New Roman", serif;/* определяемосновнойшрифт "Times New Roman", альтернативный serif (сзасечками)*/
}
.courier{
font-family: Courier, monospace;/* определяемосновнойшрифт "Courier", альтернативныйmonospace (семействомоноширинныхшрифтов) */
}
</style>
</head>
  <body>
    <pclass = "times">Параграф, отображаемыйшрифтом "Times New Roman".</p>
    <pclass = "courier">Параграф, отображаемыйшрифтом "Courier".</p>
  </body>
</html>
```

В данном примере для первого абзаца браузер проверит, есть ли в наличии у пользователя основной шрифт, если нет, то установит шрифт из семейства *serif* (с засечками). Для второго абзаца был задействован моноширинный шрифт *Courier*, а как альтернатива семейство *моноширинных* шрифтов (буквы имеют одинаковую ширину).

Шрифты, которые содержат в названии более одного слова, либо цифры, необходимо обязательно помещать в кавычки.

Результат нашего примера:

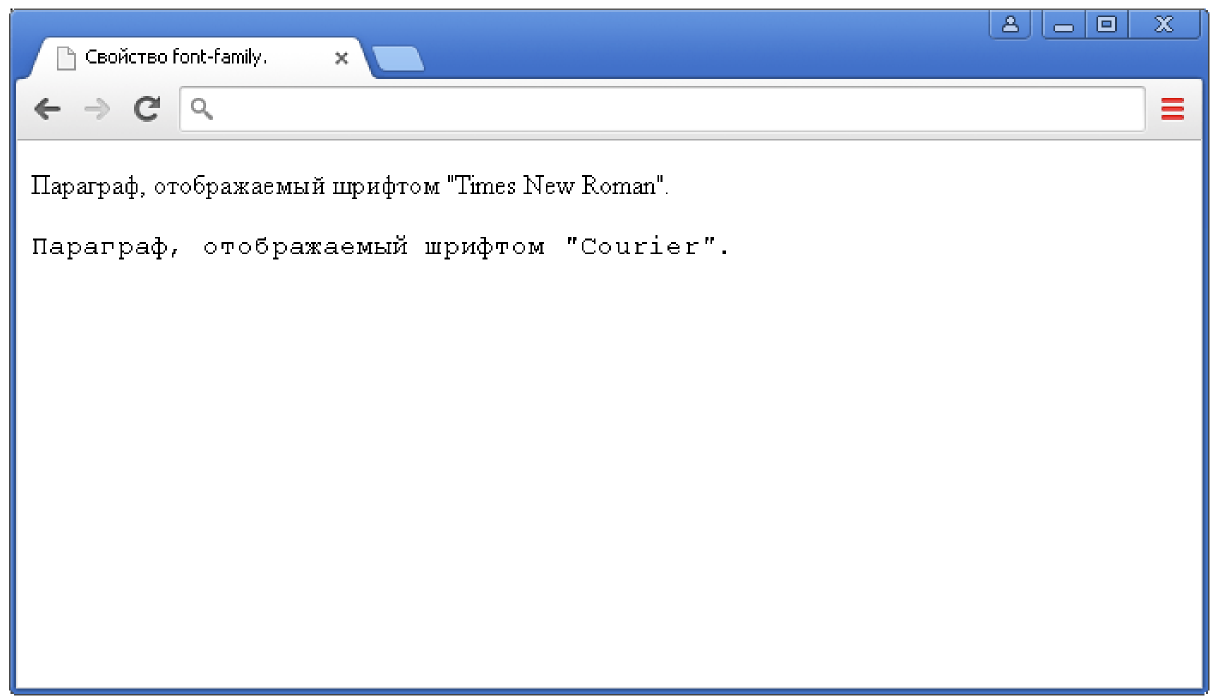


Рис. 34 Пример использования свойства font-family.

Ниже перечислены некоторые часто используемые комбинации безопасных веб-шрифтов, которые с большой вероятностью присутствуют на любом компьютере:

sans-serif (без засечек)

Семейство шрифта (font-family)	**Пример**
Arial, Helvetica, sans-serif	Съешь же еще этих сочных мандаринов.
"Arial Black", Gadget, sans-serif	Съешь же еще этих сочных мандаринов.
"Comic Sans MS", cursive, sans-serif	Съешь же еще этих сочных мандаринов.
Impact, Charcoal, sans-serif	Съешь же еще этих сочных мандаринов.
"Lucida Sans Unicode", "Lucida Grande", sans-serif	Съешь же еще этих сочных мандаринов.

Tahoma, Geneva, sans-serif	Съешь же еще этих сочных мандаринов.
"Trebuchet MS", Helvetica, sans-serif	Съешь же еще этих сочных мандаринов.
Verdana, Geneva, sans-serif	Съешь же еще этих сочных мандаринов.

serif (с засечками)

Семейство шрифта (font-family)	**Пример**
Georgia, serif	Съешь же еще этих сочных мандаринов.
"Palatino Linotype", "Book Antiqua", Palatino, serif	Съешь же еще этих сочных мандаринов.
"Times New Roman", Times, serif	Съешь же еще этих сочных мандаринов.

monospace (моноширинные)

Семейство шрифта (font-family)	**Пример**
"CourierNew", Courier, monospace	Съешь же еще этих сочных мандаринов.
"LucidaConsole", Monaco, monospace	Съешь же еще этих сочных мандаринов.

Типы веб-шрифтов и их поддержка браузерами

Все современные браузеры поддерживают использование определённых веб-шрифтов. Происходит это следующим образом: браузер пользователя загружает шрифт с указанного сервера и применяет его для

отображения текущей страницы. В настоящее время существуют следующие виды веб-шрифтов:

- **TTF/OTF** (TrueType и OpenTypeFonts) - эти шрифты имеют широкую поддержку. Разработаны Microsoft совместно с Adobe, с целью применения в различных операционных системах.
- **WOFF** (WebOpenFontFormat) - сжатая версия шрифтов TTF/OTF. Формат включает в себя метаданные, в которые автор шрифта может добавить информацию об использовании шрифта. WOFF-формат имеет широкую поддержку со стороны браузеров.
- **WOFF2** (WebOpenFontFormat 2) - спецификация была разработана, чтобы обеспечить улучшенное сжатие и тем самым снизить использование пропускной способности сети, в то же время, позволяя быстро производить декомпрессию даже на мобильных устройствах.
- **SVG** (ScalableVectorGraphic) – способ создания векторной графики. SVG-формат имеет очень ограниченную поддержку (IOS/Safari). Планируется, что он перестанет использоваться в Chrome.
- **EOT** (EmbeddedOpenType) – шрифты, которые поддерживаются только в InternetExplorer/Edge (разработаны компанией Microsoft для использования в качестве встроенных шрифтов на веб-страницах).

Формат шрифта	**Chrome**	**Firefox**	**Opera**	**Safari**	**IExplorer**	**Edge**
TTF/OTF (True Type и Open Type Fonts)	**4.0**	**3.5**	**10.0**	**3.1**	**9.0***	**12.0**
WOFF (Web Open Font Format)	**5.0**	**3.6**	**11.1**	**5.1**	**9.0**	**12.0**
WOFF2 (Web Open Font	**36.0**	**39.0***	**26.0**	**Нет**	**Нет**	**Нет**

Format 2)						
SVG (ScalableVectorGraphic)	**4.0**	**Нет**	**9.0**	**3.2**	**Нет**	**Нет**
EOT (EmbeddedOpenType)	**Нет**	**Нет**	**Нет**	**Нет**	**6.0**	**12.0**

Ответственность и поиск веб-шрифтов

Сразу хочу обратить Ваше внимание на то, что многие шрифты создаются с целью заработать на них деньги и соответственно распространяются на коммерческой основе.

Чтобы не сталкиваться с любыми правовыми вопросами, связанными с использованием «платных» шрифтов на бесплатной основе, рекомендуем Вам, пользоваться службами шрифтов, например, такой как **GoogleFonts** или, крупными коллекционными порталами, например, **Webfont.ru**. На начальном этапе Вам этого хватит «за глаза».

Добавление веб-шрифта на страницу

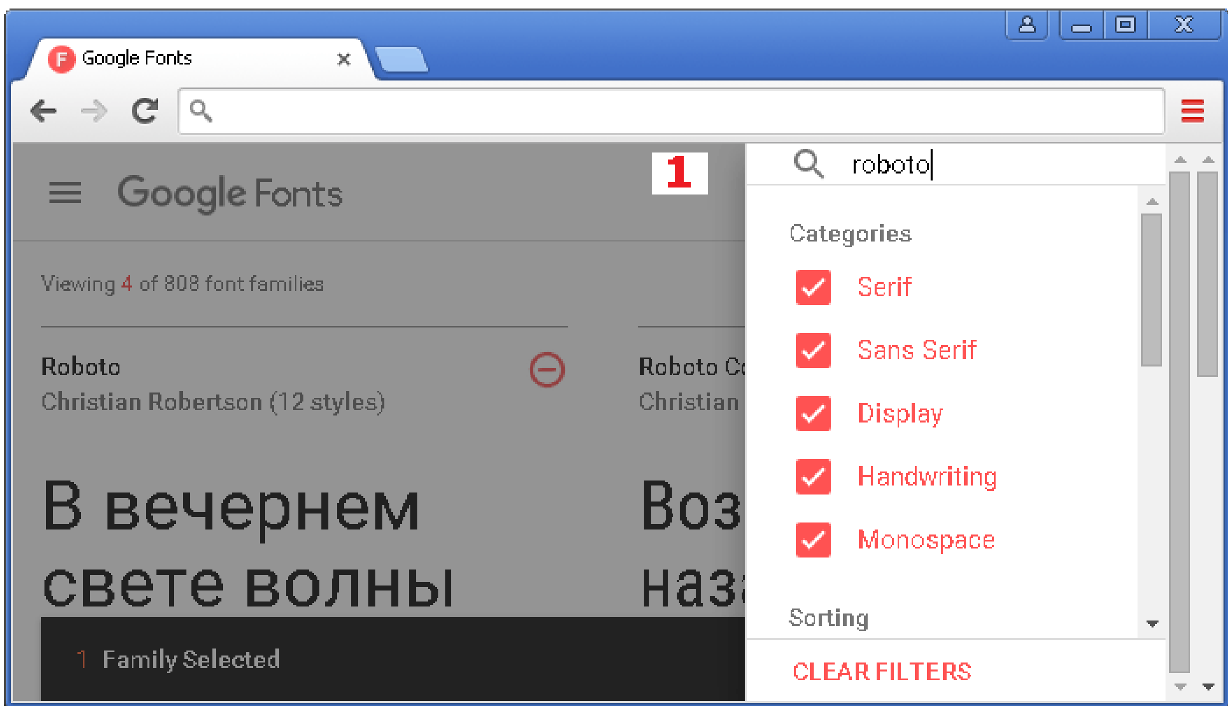
Для добавления шрифта на страницу Вам необходимо:

- использовать правило CSS **@font-face**, которое сообщает браузеру пользователя, откуда необходимо загружать шрифт и какое имя шрифта при этом используется. При работе с правилом **@font-face** важным моментом является размещение его в начале вашей таблицы стилей, это позволит вашему браузеру сразу преступить к обработке необходимого шрифта.
- использовать CSS свойство **font-family**, чтобы указать имя задействованного шрифта и применить к интересующему Вас фрагменту текста (по аналогии работы с локальными шрифтами).

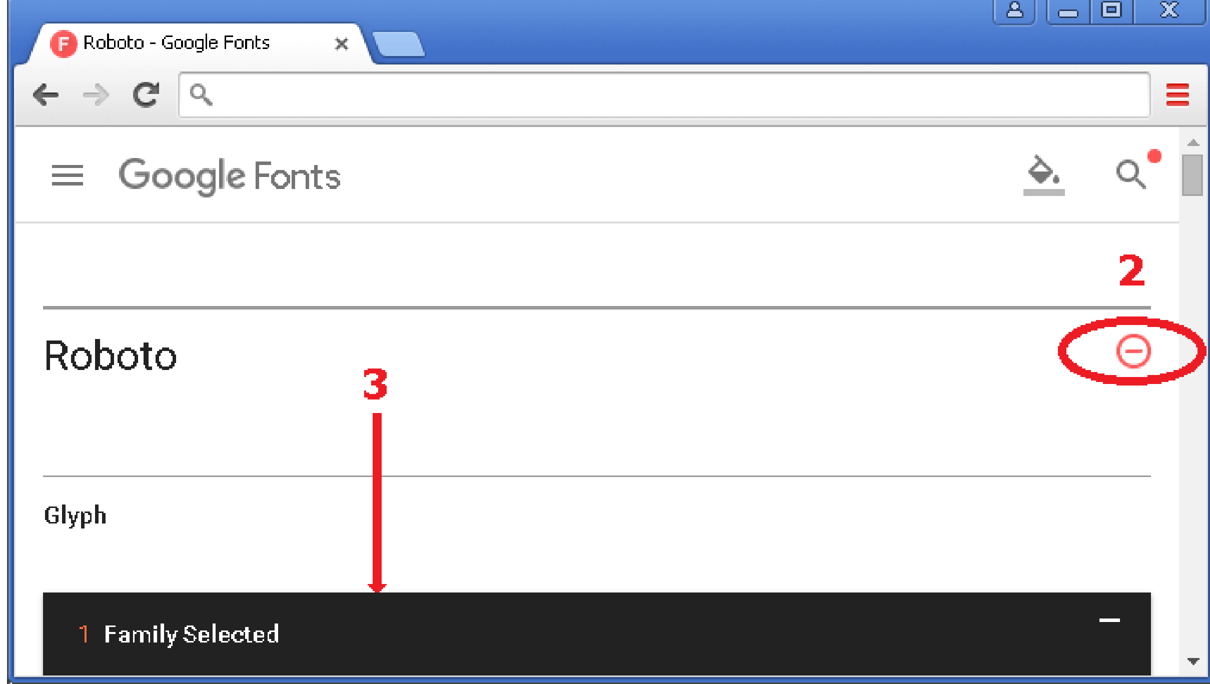
Давайте рассмотрим пошаговое подключение сторонних шрифтов, на примере использования службы **GoogleFonts**.

1. Переходим на сайт службы **GoogleFonts**.

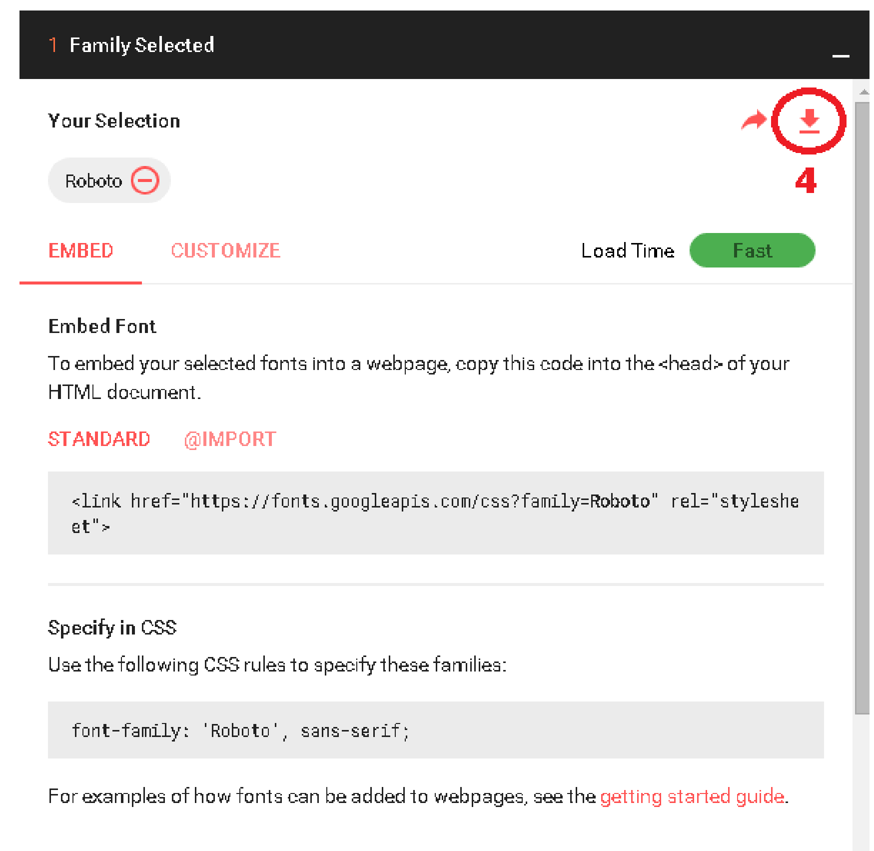
2. Выбираем понравившийся нам шрифт, рекомендуем остановиться на шрифте без засечек **Roboto**, Вы можете найти его в поиске, либо выбрать любой другой:



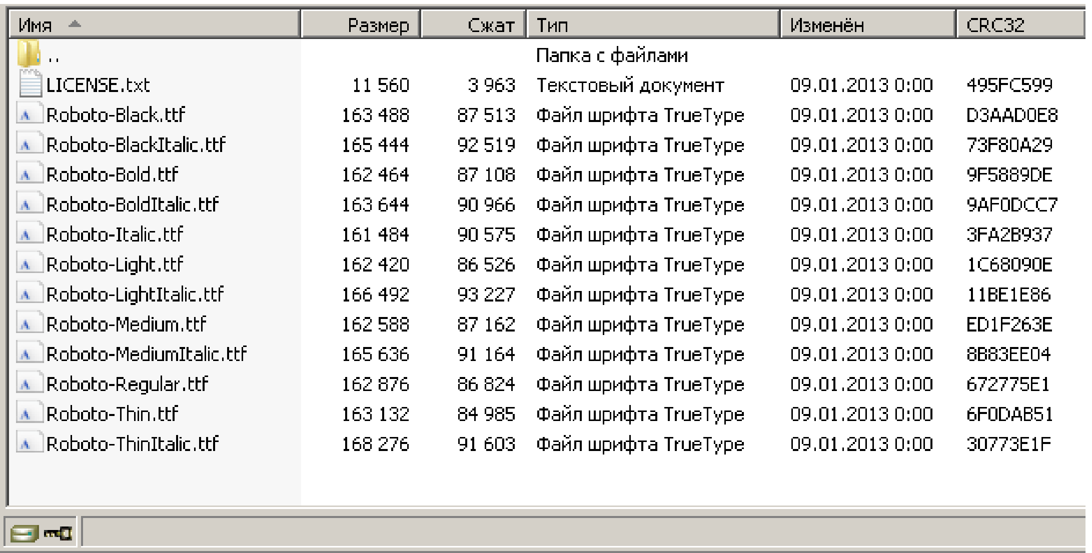
3. После этого необходимо добавить его в коллекцию (у Вас должен быть создан аккаунт Google):



4. После добавления шрифта в коллекцию, для Вас будет доступна возможность скачать его:



5. Распакуйте архив со шрифтами в директорию, из которой они будут подключаться к Вашей веб-странице, либо страницам:



Имя	Размер	Сжат	Тип	Изменён	CRC32
..			Папка с файлами		
LICENSE.txt	11 560	3 963	Текстовый документ	09.01.2013 0:00	495FC599
Roboto-Black.ttf	163 488	87 513	Файл шрифта TrueType	09.01.2013 0:00	D3AAD0E8
Roboto-BlackItalic.ttf	165 444	92 519	Файл шрифта TrueType	09.01.2013 0:00	73F80A29
Roboto-Bold.ttf	162 464	87 108	Файл шрифта TrueType	09.01.2013 0:00	9F5889DE
Roboto-BoldItalic.ttf	163 644	90 966	Файл шрифта TrueType	09.01.2013 0:00	9AF0DCC7
Roboto-Italic.ttf	161 484	90 575	Файл шрифта TrueType	09.01.2013 0:00	3FA2B937
Roboto-Light.ttf	162 420	86 526	Файл шрифта TrueType	09.01.2013 0:00	1C68090E
Roboto-LightItalic.ttf	166 492	93 227	Файл шрифта TrueType	09.01.2013 0:00	11BE1E86
Roboto-Medium.ttf	162 588	87 162	Файл шрифта TrueType	09.01.2013 0:00	ED1F263E
Roboto-MediumItalic.ttf	165 636	91 164	Файл шрифта TrueType	09.01.2013 0:00	8B83EE04
Roboto-Regular.ttf	162 876	86 824	Файл шрифта TrueType	09.01.2013 0:00	672775E1
Roboto-Thin.ttf	163 132	84 985	Файл шрифта TrueType	09.01.2013 0:00	6F0DAB51
Roboto-ThinItalic.ttf	168 276	91 603	Файл шрифта TrueType	09.01.2013 0:00	30773E1F

Как Вы можете заметить, в архиве содержится 12 различных шрифтов. Исходя из названий можно установить, что, например, **Roboto-Italic** предназначен для курсивного стиля шрифта, **Roboto-Bold** для жирного начертания шрифта, **Roboto-BoldItalic** для курсивного жирного начертания и так далее.

Еще раз обращаю Ваше внимание, что один файл шрифта содержит:

- одну плотность шрифта.
- один стиль для этого шрифта.

Обратите внимание на важный момент - если Вам необходимо получить полужирные и курсивные шрифты, то необходимо подгружать их отдельно (использовать правило CSS @font-face для данного типа шрифта)!

Конечно в том случае, если они существуют для понравившегося Вам варианта шрифта, обязательно учтите эти моменты при работе с веб-шрифтами!

Приступим к подключению, загруженных нами шрифтов к нашему документу. Для заголовков второго уровня (элемент <h2>) мы будем использовать шрифт - **Roboto-Bold**. Для абзацев (элемент <p>) будем использовать шрифт **Roboto-Regular**, а для курсивного начертания (элемент <i>) - **Roboto-Italic**.

```
<!DOCTYPE html>
<html>
<head>
  <metacharset = "UTF-8">
  <title>Пример использования правила @font-face</title>
<style>
@font-face{
font-family: "Roboto"; /* задаем произвольное имя для шрифта, которое будет использоваться в свойстве font-family при указании стилей для конкретных элементов */
src: url("/fonts/Roboto-Regular.ttf") format('truetype'); /* задаем путь относительно корня сайта к шрифту (url) и тип шрифта (format). Тип
```

шрифта выступает подсказкой для браузера (в идеале ускоряет процесс обработки) */

font-style: normal; /* указываем, что стиль шрифта обычный - это значение по умолчанию */

font-weight: normal; /* определяет нормальное начертание символов - это значение по умолчанию */

}

@font-face{

font-family: "Roboto"; /* задаем произвольное имя для шрифта, которое будет использоваться в свойстве font-family при указании стилей для конкретных элементов */

/* для упрощения работы со шрифтами допускается использовать одинаковое имя, но при этом начертание или стиль шрифта должен отличаться. IE 8 и ниже не поддерживают такой подход и если Вы хотите поддерживать эти браузеры, то Вам необходимо называть каждый шрифт по разному и более детально стилизовать каждый селектор */

src: url("/fonts/Roboto-Bold.ttf") format('truetype'); /* задаем путь относительно корня сайта к шрифту (url) и тип шрифта (format). Тип шрифта выступает подсказкой для браузера (в идеале ускоряет процесс обработки) */

font-style: normal; /* указываем, что стиль шрифта обычный - это значение по умолчанию */

font-weight: bold; /* определяет жирное начертание символов */

}

@font-face{

font-family: "Roboto"; /* задаем произвольное имя для шрифта, которое будет использоваться в свойстве font-family при указании стилей для конкретных элементов */

src: url("/fonts/Roboto-Italic.ttf") format('truetype'); /* задаем путь относительно корня сайта к шрифту (url) и тип шрифта (format). Тип

```
шрифта выступает подсказкой для браузера (в идеале ускоряет процесс обработки) */
        font-style: italic;  /* указываем, что стиль шрифта курсивный */
        font-weight:  normal;    /* определяет нормальное начертание символов - это значение по умолчанию */
        }
        h2, p, b, i{/* задаем групповой селектор для элементов <h2>, <p>, <b>, <i> */
        font-family: "Roboto", sans-serif;/* устанавливаем шрифт Roboto, если он не будет загружен по какой-то причине, то указываем, чтобы браузер установил шрифт из семейства шрифтов без засечек (sans-serif) */
        }
        </style>
        </head>
        <body>
          <h2>Немного о пандах</h2>
          <p><b>Большая панда</b> (<i>Ailuropodamelanoleuca</i>) — бамбуковый медведь, одно из редчайших вымирающих животных, занесённых в международную Красную книгу; единственный современный представитель рода <b>большие панды</b> (<i>Ailuropoda</i>).</p>
        </body>
        </html>
```

И так, что мы сделали в этом примере:

- Добавили *три* правила **@font-face** **в начало наших CSS стилей** (это важно).
- В каждом правиле с использованием CSS свойства (**font-family**) мы указали одно название шрифта, вы можете использовать своё название для всех правил, но будет лучше и понятнее если оно будет совпадать с наименованием шрифта.

- В каждом правиле мы указали путь к файлу, который содержит шрифт с разрешением **TTF** (*TrueTypeFont*).
- Далее мы для всех правил с использованием свойства (**font-style**) указали стиль шрифта: для *двух* правил шрифт отображается обычным стилем - **normal** (это значение по умолчанию), а для *одного* указали, что он отображается курсивом (***italic***).
- Кроме того, для всех правил с использованием свойства (**font-weight**) указали жирность шрифта: для *двух* правил шрифт отображается обычной жирности - **normal** (это значение по умолчанию), а для *одного* указали что он отображается жирным шрифтом (**bold**).
- Нам осталось только применить наши шрифты к элементам. Для этого мы создали групповой селектор в котором с использованием ранее рассмотренного свойства **font-family** указали тип нашего шрифта и как альтернативу через запятую указали семейство шрифта, это сделано для того, что если по какой-то причине браузер пользователя не сможет загрузить наши шрифты, он смог использовать шрифты из указанного семейства, а не встроенные в браузер.

Результат нашего примера:

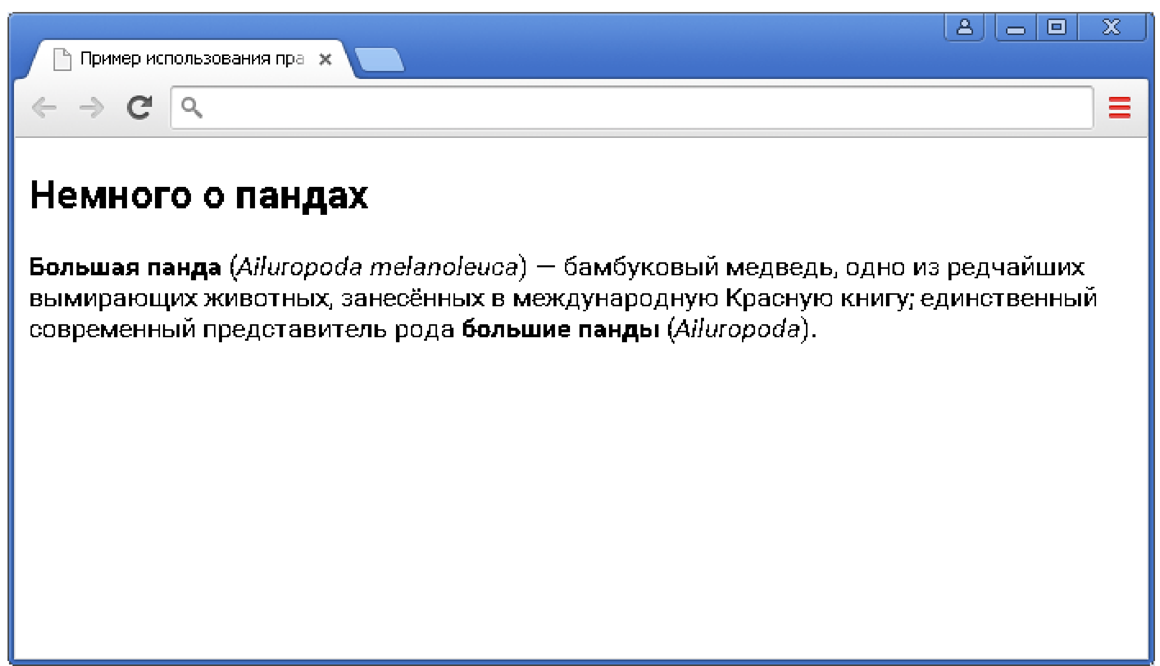


Рис.38 Пример использования правила @font-face.

Для полноты картины, давайте рассмотрим, как добавлять несколько вариантов шрифтов (для поддержки современных более сжатых версий шрифтов):

```
@font-face{
font-family: "Roboto";  /* задаем произвольное имя для шрифта, которое будет использоваться в свойстве font-family при указании стилей для конкретных элементов */
src:
  local("font");   /* проверяется есть ли шрифт на локальном компьютере пользователя по имени, если нет то поиск осуществляется в других указанных источниках */
  url("/fonts/font.woff2") format('woff2');  /* задаем путь к шрифту (url) и тип шрифта (format) */
  url("/fonts/font.woff") format('woff');  /* задаем путь к шрифту (url) и тип шрифта (format) */
  url("/fonts/font.ttf") format('truetype');  /* задаем путь к шрифту (url) и тип шрифта (format) */
}
```

Как вы можете заметить, для того, чтобы добавить несколько вариантов шрифтов необходимо указать несколько путей к файлам, которые содержат шрифты с определенным разрешением.

Обращаю Ваше внимание, что порядок, в котором вы указываете варианты шрифтов, имеет важное значение, так как Ваш браузер выбирает первый поддерживаемый шрифт. К примеру, если Вы хотите использовать шрифты на сайте WOFF2, которые поддерживаются не всеми браузерами, то необходимо их указать до WOFF.

В настоящее время с некоторыми браузерами могут возникать проблемы при использовании директивы **local**, которая служит для проверки

наличия шрифта (по определенном имени) на локальном компьютере пользователя, по этой причине рекомендуется Вам использовать её в своих проектах, для этих целей более надежным является использование скриптов.

Добавление веб-шрифта со стороннего ресурса

Давайте рассмотрим на примере службы **GoogleFonts** варианты подключения шрифтов к Вашим страницам без загрузки их на Ваш сервер.

1. Переходим на сайт службы **GoogleFonts**.
2. Выбираем понравившийся нам шрифт, рекомендуем на том же шрифте, который мы использовали в прошлый раз - шрифт без засечек **Roboto**. Вы можете использовать поиск по наименованию для его нахождения.
3. После добавления шрифта необходимо нажать на вкладку *«Family-selected»* (выбранные вами шрифты):

Roboto

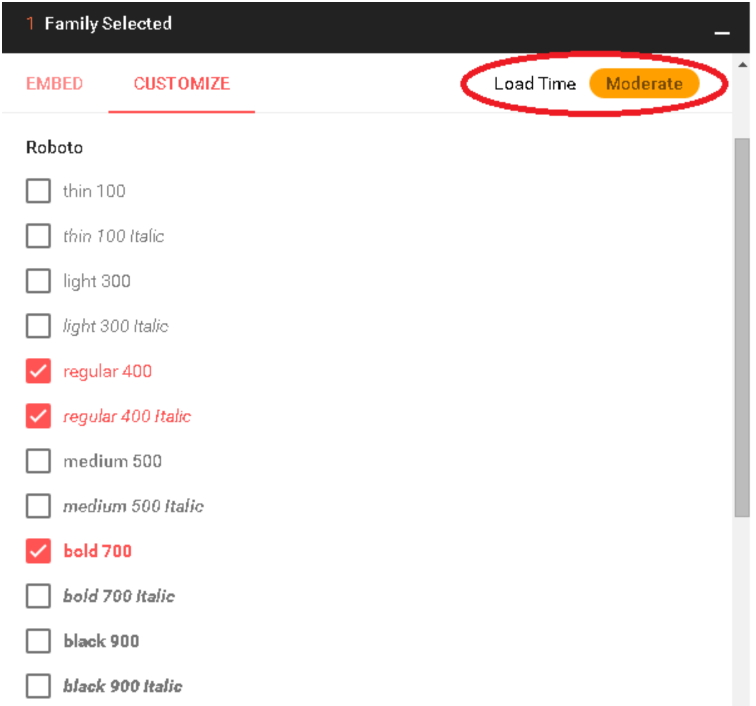
Glyph

1 Family Selected

4. Далее необходимо выбрать те стили шрифтов, которые мы будем использовать на нашем сайте. Для этого необходимо перейти на вкладку *«Customize»*

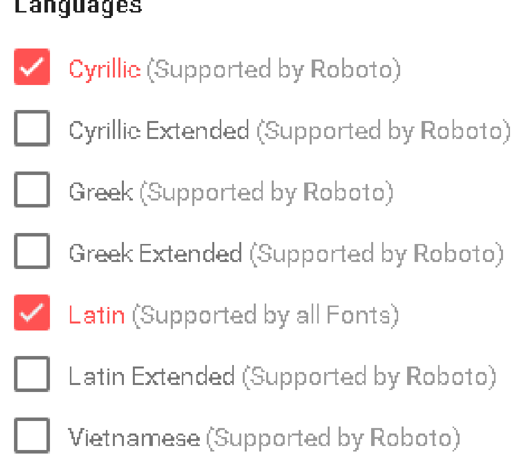
Например, меня интересуют следующие:

- Для заголовков (элемент <h2>) и для жирного начертания будем использовать шрифт – **Bold 700** (RobotoBold в наборе).
- Для абзацев (элемент <p>) будем использовать шрифт **Normal 400** (RobotoRegular в наборе), а для курсивного начертания **Normal 400 Italic** (RobotoItalic в наборе).



При выборе Вам отобразят влияние тех или иных шрифтов на время загрузки страницы. **Используя множество стилей шрифтов, может привести к замедлению загрузки Вашей страницы, поэтому рекомендуется выбирать только те шрифты, которые вам действительно необходимы на вашем сайте.**

5. Следующим шагом необходимо выбрать те наборы символов (языки), которые будете использовать на своем сайте (выбор языков находится в том же вкладке - *«Customize»* немного ниже). В данном случае выбраны кириллические и латинские символы. Выбирайте только те наборы, которые вам необходимы, чтобы Ваши страницы загружались быстрее:



6. Далее на вкладке *«Embed»* нам будет предложено два варианта подключения (**Standart** и **@import**):

Первый вариант подключения (Standard). Создание ссылки на внешнюю таблицу стилей, используя HTML тег <link>. В адресе ссылки указывается веб-сервер Google и информация, которая необходима Google для загрузки необходимых шрифтов (как правило, это формат *woff2* для современных браузеров, определение типа поддерживаемого шрифта происходит на стороне сервера Google в зависимости от Вашего браузера).

Как вы видите, в ссылке указывается наименование шрифта, толщина шрифта и какой набор символов используется. Если вы внимательно читали статью **«Введение в CSS»**, то Вы уже догадались, что необходимо эту ссылку указать на каждой странице, где необходимо использовать данные шрифты.

Обращаю Ваше внимание, что тег **<link>** размещается всегда внутри тега **<head>** (как правило, перед закрывающим тегом **</head>**).

Необходимое наименование шрифта и альтернативный вариант так же указывается в описании:

Specify in CSS

Use the following CSS rules to specify these families:

```
font-family: 'Roboto', sans-serif;
```

For examples of how fonts can be added to webpages, see the getting started guide.

Рассмотрим пример подключения, выбранных нами шрифтов:

```
<!DOCTYPE html>
<html>
<head>
  <metacharset = "UTF-8">
  <title>Пример подключение веб-шрифтов, используя тег <link></title>
<link href = 'https://fonts.googleapis.com/css?family=Roboto:400,700,400italic&subset=latin,cyrillic'rel = 'stylesheet'type = 'text/css'> /* подключаемвнешнюютаблицу стилей для загрузки необходимых шрифтов, в HTML 5 type='text/css' допускается не указывать */
<style>
h2, p, b, i{/* задаем групповой селектор для элементов <h2>, <p>, <b>, <i> */
font-family : "Roboto", sans-serif;/* устанавливаем шрифт Roboto, если он не будет загружен по какой-то причине, то указываем, чтобы браузер установил шрифт из семейства шрифтов без засечек (sans-serif) */
}
</style>
</head>
  <body>
        <h2>Немного о верблюдах</h2>
        <p><b>Верблюды</b> (<i>Camelus</i>) — род млекопитающих семейства верблюдовых (<i>Camelidae</i>) подотряда мозоленогих (<i>Camelidae</i>) отряда парнокопытных (<i>Artiodactyla</i>). Это крупные животные, приспособленные для жизни в засушливых регионах мира — пустынях, полупустынях и степях.</p>
  </body>
```

```
</html>
```

Результат:

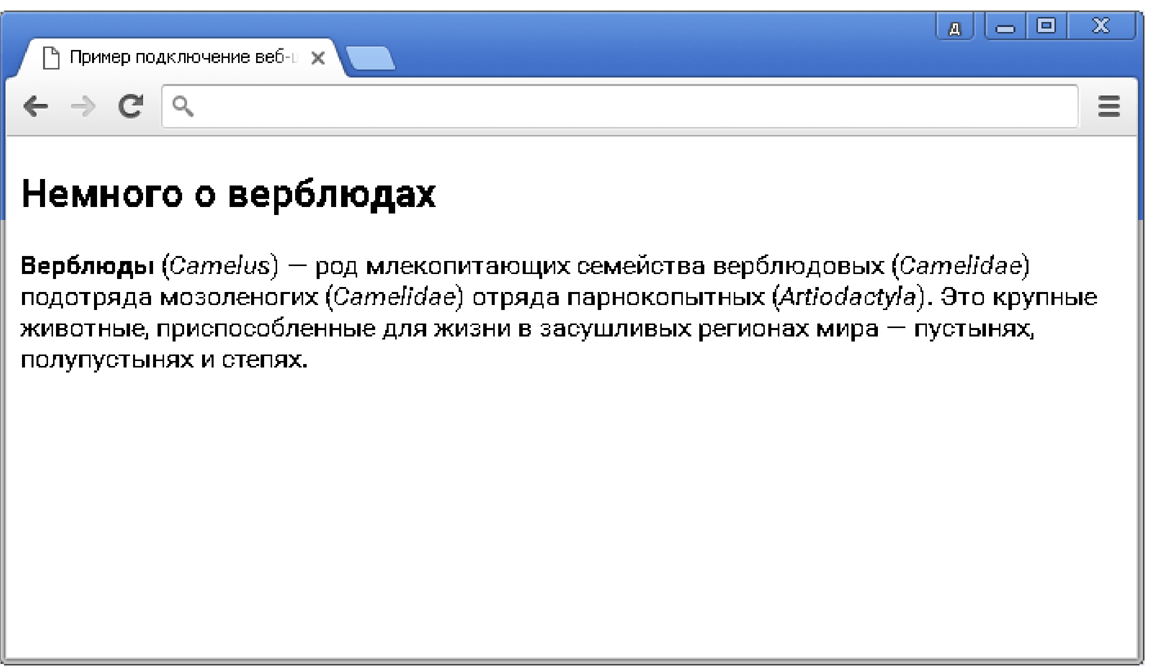


Рис.44 Пример подключение веб-шрифтов, используя тег <link>.

Второй вариант подключения, который мы также рассматривали в статье **«Введение в CSS»** это использование правила **@import**. В отличие от первого метода (используя HTML тег **<link>**), который требует добавления кода к каждой странице Вашего сайта, правило **@import** допускается использовать в начале своей таблицы стилей, которая у Вас уже должна быть подключена к каждой странице.

Чтобы выполнить привязку к внешнему файлу CSS, нужно использовать **url** и заключить путь к CSS файлу в круглые скобки. Допустимо заключить содержимое в скобках в кавычки:

```
@importurl("path/to/file.css");
```

Предлагаемый вариант импортирования на страницу:

STANDARD @IMPORT

```
<style>
@import 'https://fonts.googleapis.com/css?family=Roboto:400,400i,700&subset=cyrillic';
</style>
```

Specify in CSS

Use the following CSS rules to specify these families:

```
font-family: 'Roboto', sans-serif;
```

Обращаю Ваше внимание, что правила @import всегда необходимо указывать перед стилями CSS, иначе, таблицы стилей не импортируются (браузеры просто их проигнорируют).

Вы можете использовать правило **@import** как во внешних таблицах стилей, так и во внутренних. Рассмотрим пример подключения, выбранных нами шрифтов, используя правило **@import** во внутренних таблицах стилей:

```
<!DOCTYPE html>
<html>
<head>
  <metacharset = "UTF-8">
  <title>Пример подключение веб-шрифтов, используя правило @import</title>
<style>
@importurl(https://fonts.googleapis.com/css?family=Roboto:400,700,400italic&subset=latin,cyrillic); /* импортируем внешнюю таблицу стилей */
h2, p, b, i{/* задаем групповой селектор для элементов <h2>, <p>, <b>, <i> */
font-family : "Roboto", sans-serif;/* устанавливаем шрифт Roboto, если он не будет загружен по какой-то причине, то указываем, чтобы
```

```
браузер установил шрифт из семейства шрифтов без засечек (sans-serif)
*/
        }
        </style>
        </head>
          <body>
                    <h2>Немного о пингвинах</h2>
                    <p><b>Пингвиновые</b> или <b>пингвины</b> (лат.
<i>Spheniscidae</i>) —  семейство нелетающих морских птиц, единственное
в отряде <b>пингвинообразных</b> (<i>Sphenisciformes</i>). В семействе 18
современных видов. Все представители этого семейства хорошо плавают и
ныряют.</p>
          </body>
        </html>
```

Результат нашего примера:

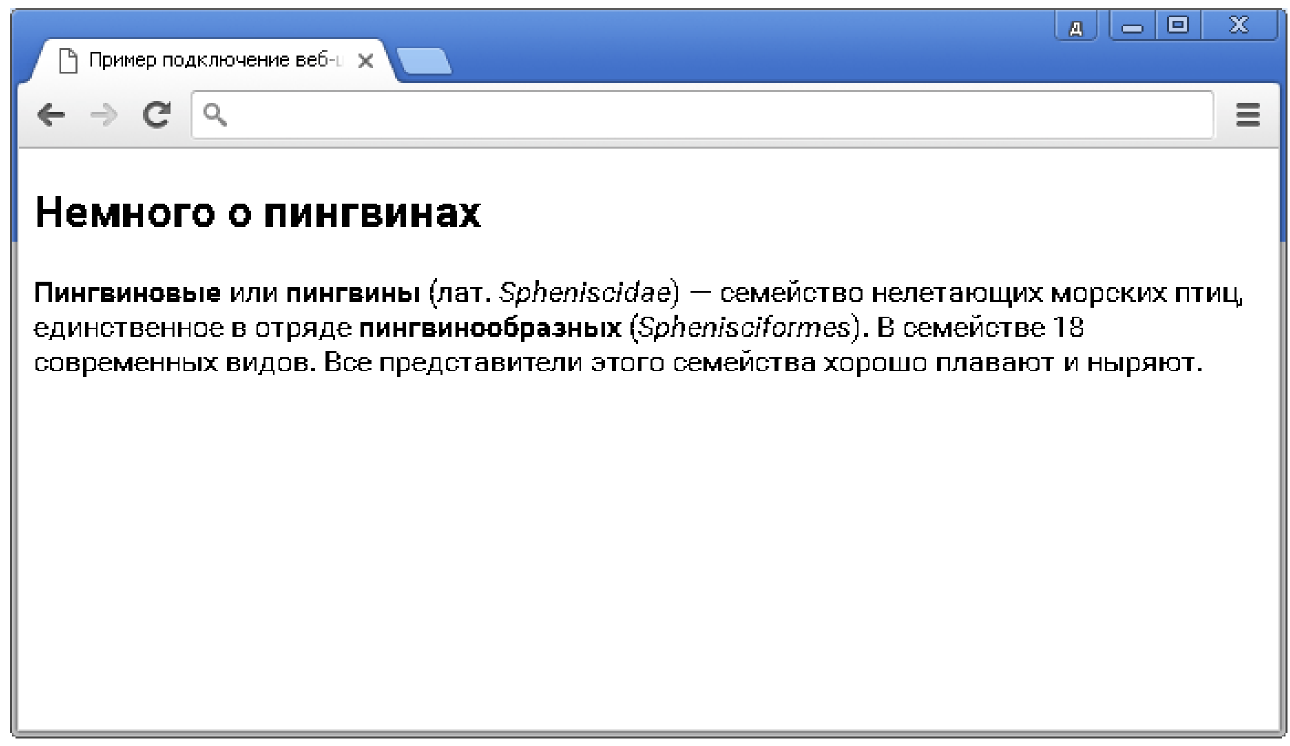


Рис.46 Пример подключение веб-шрифтов, используя правило @import.

Прошу Вас учесть тот момент, что правило **@import** увеличивает число соединений с сервером, поскольку файл, на который вы ссылаетесь, тоже нужно скачать и проанализировать. Исходя из **правил и рекомендаций PageSpeedInsight** (Google), связанных с оптимизацией страниц, рекомендуется избегать применения правила **@import**.

Кроме того, в некоторых случаях правило **@import** может замедлять загрузку таблиц стилей, либо загружать их не в заданном порядке, что может быть критично для конечного отображения конкретной страницы. Не используйте его в своих проектах.

Глава 17. Кроссбраузерность сайта

Почему кроссбраузерность сайта так важна. Если Вы ранее не слышали о таком сложном, на первый взгляд, понятии, сейчас самое время восполнить этот пробел в Ваших знаниях. Ведь если Вы являетесь владельцем ресурса, который предположительно должен приносить прибыль, проверка кроссбраузерности сайта должна быть обязательным условием его эффективной работы. Итак, приступаем.

Кроссбраузерность – это способность веб-ресурса к идентичному отображению в большинстве популярных браузеров. При этом особое внимание уделяется сохранению одинакового уровня читабельности и отсутствию разрывов и других визуальных несоответствий верстки.

Возможно, Вам кажется, что Вы уже слышали о чем-то подобном, только название было немного другое – адаптивность сайта. Спешим Вас разочаровать, это разные вещи.

В случае с адаптивностью речь идет о том, что веб-ресурс должен корректно отображаться и функционировать на максимально возможном количестве устройств (ПК, планшеты, смартфоны, телефоны) при всех возможных разрешениях их экранов. О различных браузерах при этом не упоминается.

Поэтому кроссбраузерность – это отдельная и очень важная способность любого «успешного» веб-ресурса.

Но все же почему кроссбраузерная верстка является таким важным условием создания сайта?

Многочисленные исследования говорят, что у каждого браузера есть своя целевая аудитория. И если Chrome или Яндекс.Браузер использует предпочтительно молодежь, то люди старше 35 лет отдают предпочтение Opera, Mozilla и InternetExplorer.

Не все пользуются одним и тем же и **игнорировать другие браузеры означает добровольно сокращать количество своих посетителей**, которые могли бы стать клиентами.

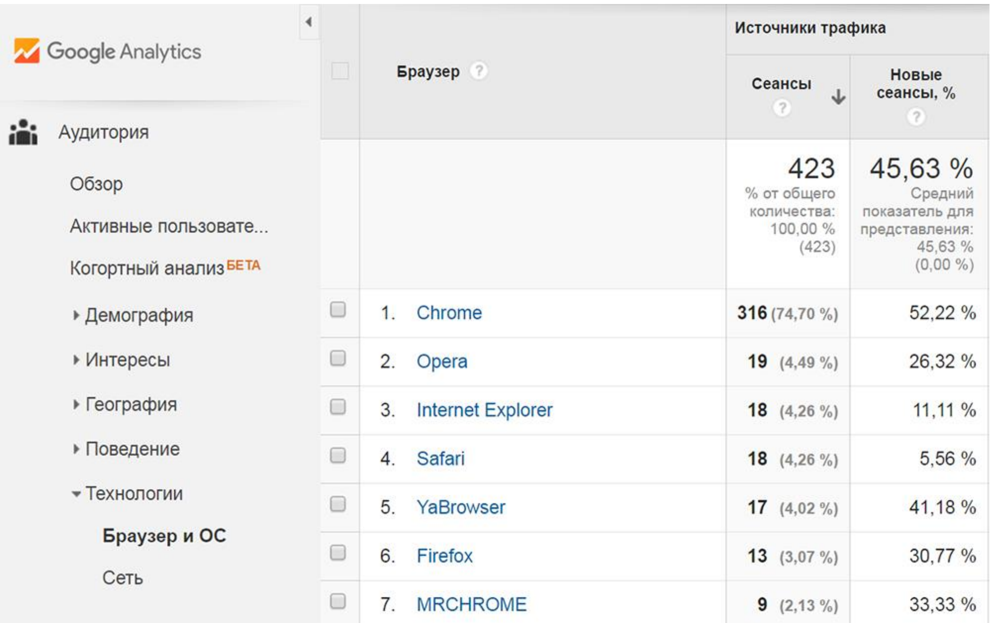
Как проверить кроссбраузерность сайта

Для того чтобы проверить кроссбраузерность сайта можно пойти несколькими путями:

1. В идеале нужно **установить все возможные браузеры и убедиться в правильности отображения сайта и взаимодействии пользователя с ним в каждом случае**. Этот путь самый верный, но достаточно сложен в исполнении, в частности, из-за увеличения трудо- и временных затрат.

2. Иногда случается, что нет возможности (или желания) выполнять проверку в десятках браузеров. В таком случае можно в соответствующем

разделе GoogleAnalytics **ознакомиться со списком браузеров, на которых случались открытия сайта** и ограничить проверку кроссбраузерности в соответствии с ним. Кроме того, там же можно ознакомиться с %-м соотношением заходов из разных браузеров, что поможет сформировать приоритеты и понять, какими из них пользуются чаще.



Проверить кроссбраузерность сайта с помощью упрощенных способов:

а) **Онлайн сервисы**. Бывают как платные так и бесплатные. Например, Browsershots, в котором доступна бесплатная проверка кроссбраузерности сайта для 65 версий популярных браузеров с имитацией ОС, на которых эти браузеры якобы установлены.

Получение результатов происходит довольно медленно в бесплатной версии (от 5 мин. до 2-х часов в зависимости от загруженности сервера запросами). Результаты отображаются в виде скриншотов, по которым можно оценить насколько хорошо была выполнена кроссбраузернаяверстка проверяемого веб-сайта.

Существуют и более мощные иструменты (чаще всего платные) вроде BrowserCam, которые дают возможность проверить кроссбраузерность сайта в режиме реального времени через удаленный доступ к компьютерам, на которых установлены нужные браузеры.

b) **Локальные приложения**. Действуют по тем же принципам, что и описанные выше онлайн сервисы, но зачастую имеют более широкий функционал и возможности. Например, Multi-BrowserViewer позволяет не только увидеть, как будет выглядеть веб-ресурс в разных браузерах, но и полноценно проверить их работу.

Почему проверка кроссбраузерности может провалиться

Вот основные причины, по которым проверка кроссбраузерности сайта может показывать наличие проблем:

1. **Кроссбраузерная верстка выполнена с ошибками в коде и несоответствием стандартам HTML** (когда выполняется валидация сайта, можно узнать, какие именно ошибки присутствуют).
2. **Появление новых библиотек и CSS** (также обновление старых) могут приводить к некорректному отображению в старых версиях браузеров, что довольно часто обнаруживается при проверке кроссбраузероности сайта.
3. **Разные значения атрибутов** (цвета, шрифты, их размер) по умолчанию в различных браузерах. Нужно те, под которые сверстан сайт, прописывать в таблице стилей, иначе каждый браузер будет использовать свои значения, что приведет к изменению отображения.

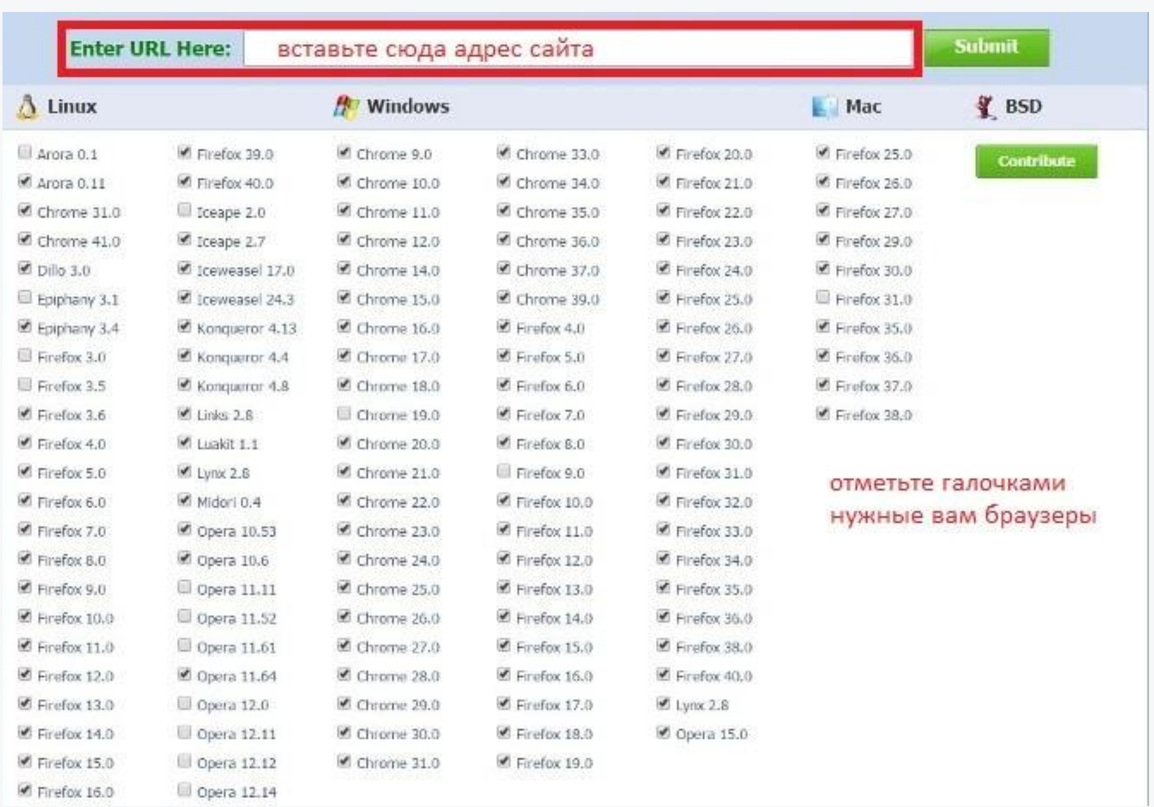
Почему стоит проводить тестирование кроссбраузерности?

Все очень просто – чем лучше ваш сайт смотрится в разных браузерах, тем больше посетителей вам гарантировано. Это очень важный параметр.

Обычно за него отвечают верстальщики, которые занимаются версткой шаблона сайта. Поскольку люди используют разные веб-обозреватели, то для максимального охвата аудитории вам придется делать сайт приятным для использования всем.

Где проверить кроссбраузерность в режиме онлайн

Самый популярный бесплатный сервис для этого — browsershots.org/. Вы можете перейти по ссылке или проверить свой сайт прямо тут. Просто впишите ссылку на ваш сайт и поставьте галочки на те браузеры, в которых нужно выполнить тест.



Совет: не ставьте галочки везде, а только те, которые вам действительно нужны, а то проверка займет много времени.

Единственный минус сервиса – он проверяет долго и почти всегда загружен. Но если вы готовы подождать, то пользуйтесь. Сервис дает вам доступ к скриншотам вашего проекта в разных веб-обозревателях.

Доскональная проверка

Допустим, вам нужно больше возможностей для проверки. Это может пригодиться профессиональным веб-разработчикам или просто веб-мастерам

перфекционистам. Что ж, такие сервисы есть, но сразу предупреждаю, они платные. Хотя попробовать можно бесплатно.

crossbrowsertesting.com/

browserstack.com/

Оба сервиса на английском, так как зарубежные сервисы намного лучше преуспели в плане веб-технологий, чем наши. В принципе, даже без особых знаний английского там можно зарегистрироваться и попробовать. Там же можно проверять и адаптивность, если надо. Пользоваться можно начать сразу после регистрации, но без платных возможностей вам будут доступны лишь некоторые инструменты.

Плюсы платных сервисов очевидны – вы получите проверку кроссбраузерности онлайн максимального качества. Сколько это стоит? Примерно от 10 долларов. Впрочем, если вы веб-разработчик или веб-аналитик, то вам не жалко будет отдать деньги за профессиональную проверку. Например, в crossbrowsertesting можно протестировать свой сайт более чем в 900 браузерах. Причем это реальное тестирование с возможностью покликать, а не просто увидеть скриншот.

Тестирование в IE

IE Tester — это программа, выпущенная специально для просмотра сайта в версиях InternetExplorer от 6 до 9. Хотя программа во многом недоработана, она все же позволяет посмотреть на свой проект в этих браузерах и сделать соответствующие выводы. Так что если вы адаптируете проект под IE, то иметь такой инструмент вам может быть полезно.

Как проверить кроссбраузерность самому?

Проверить можно и самому. Это самый простой вариант. Просто откройте ваш сайт во всех браузерах, которые у вас установлены. Сходите к друзьям, у которых есть другие версии браузеров. Попросите знакомых зайти на ваш сайт с их браузеров и выслать вам скриншот. Заодно наладите деловые контакты

Также хочу вам посоветовать использовать супер полезный англоязычный сервис caniuse. В нем вы можете узнать, какие свойства из новых технологий поддерживаются в тех или иных браузерах. Например, жмем на псевдокласс :selection и видим, что IE8 его не поддерживает, Mozilla поддерживает с префиксом и т.д.

Это очень полезные знания, потому что они помогают вам понять, как действовать дальше.

На этом все методы проверки можно считать рассмотренными. Какой из них выбирать и выбирать ли вообще — дело исключительно ваше. Если вам была интересна статья, то подписывайтесь на этот блог, чтобы узнавать больше о веб-разработке.

РАЗДЕЛ 4. ПРОГРАММИРОВАНИЕ НА JAVASCRIPT

Глава 18. Выражения, переменные, объекты JavaScript.

Язык программирования JavaScript был разработан фирмой Netscape в сотрудничестве с SunMicrosystems и анонсирован в 1995 году. JavaScript предназначен для создания интерактивных html-документов. Основные области использования JavaScript: Создание динамических страниц, т.е. страниц, содержимое которых может меняться после загрузки. Проверка правильности заполнения пользовательских форм.

Решение "локальных" задач с помощью сценариев. JavaScript-код - основа большинства Ajax-приложений. JavaScript позволяет создавать приложения, выполняемые на стороне клиента, т.е. эти приложения выполняются браузером на компьютере пользователя. Программы (сценарии) на этом языке обрабатываются встроенным в браузер интерпретатором. К сожалению, не все сценарии выполняются корректно во всех браузерах, поэтому тестируйте свои javascript-программы в различных браузерах.

Язык JavaScriptрегистрозависимый, т.е. заглавные и прописные буквы алфавита считаются разными символами. Прежде, чем приступить к написанию сценариев, необходимо ознакомиться с основными понятиями, такими как литералы, переменные и выражения.

Логически цельные и обособленные действия программы на JavaScript, будь то присвоение значения переменной, вызов функции и пр., записываются в отдельных строках (эти действия называются операторами, или инструкциями). В конце строк обязательный для многих других языков программирования (для того же Java) символ ; ставить необязательно, за исключением тех случаев, когда нужно явно задать пустой оператор (в тех местах, где оператор требуется синтаксисом языка, но он не нужен программисту) или если все же нужно записать несколько действий в одной строке. Например:

```
var a, b = 123
a = a + 12 + b*2
```

a = a + b; c = a – b / 2

Однако во всех примерах книги строки заканчиваются символом ;, чтобы избежать путаницы (особенно для людей, которые уже имеют опыт программирования на C/C++ или Java). Выражения и операторы программы можно переносить на следующую строку, но только там, где допускается пробел.

Например:

var a, b = «text»

a = b + "=" + "Очень длинная строка текста, которая настолько длинна, что "+" даже не помещается на странице"

12.2. Комментарии

В тексте программы на JavaScript можно помещать два вида комментариев: однострочные и многострочные. Однострочный комментарий обозначается //, а многострочный заключается между /* и */. Примеры обоих видов комментариев:

//Однострочный комментарий

/* Многострочный комментарий */

12.3. Типы данных, переменные, выражения

Основные типы данных, которыми позволяет манипулировать JavaScript, приведены в табл. 12.1 (кроме того, существуют ссылки на объекты и функции, но они будут рассмотрены позже).

Таблица 12.1. Типы данных JavaScript

Тип	Описание
Логический	Может принимать два значения: ИСТИНА (true) и ЛОЖЬ (false)
Целочисленный	Положительное или отрицательное целое число. Можно использовать как десятичные, так и шестнадцатеричные (задаются с 0x в начале числа) значения, например: –1231, 43526412653, 0xFEF5F

Тип	Описание
Вещественный	Положительное или отрицательное число с плавающей точкой. Если нужно обязательно, чтобы число без дробной части воспринималось как вещественное, то точку нужно ставить (например, 432.0 можно записать как 432., но не 432). Может использоваться экспоненциальная запись, например: 1.232e78 или 1.232e+78, –3.232e–89
Строковый	Хранит строки символов. Возможно использование escape-символов (табл. 12.2). Значения должны заключаться либо в двойные, либо в одинарные кавычки. Если используются двойные кавычки, то в строке могут содержаться и одинарные кавычки (например, "Текст в 'одинарных' кавычках"), и наоборот (например, 'Текст в "двойных" кавычках')

Перечисленные выше типы данных применяются в выражениях, например следующее выражение: 524 + 45.23 + «sometext» в результате даст значение «569.23sometext». Из полученного результата можно увидеть, что при расчете значений выражений производится автоматическое преобразование типов значений от менее универсальных типов к более универсальным (сверху вниз по табл. 12.1).

Относительно строковых значений осталось рассказать о том, что такое escapeсимволы. Так вот, escapeсимволы применяются для вставки в строки символов, которые не могут быть введены в текстовом редакторе либо являются недопустимыми (например, двойные кавычки в строке, заключенной в двойные кавычки). Доступные в JavaScriptescape?символы, а также их описания приведены в табл. 12.2.

Таблица 12.2. Escape-символы

Обозначение	Описание
\b	Удаление символа (backspace)
\f	Перевод страницы
\n	Перевод строки
\r	Возврат каретки
\t	Символ табуляции
\'	Одинарная кавычка (апостроф)
\"	Двойная кавычка
\\	Обратный слеш

Чтобы хранить значения выражений, используются переменные. Каждая переменная имеет имя (идентификатор). Имя переменной может содержать символы латинского алфавита, цифры и символ подчеркивания, причем первым символом не должна быть цифра, например: myVar, _any123var. Имена переменных не должны совпадать с зарезервированными

словами языка JavaScript (они далее называются ключевыми, так менее точно, но короче). Ключевыми словами являются также все названия операторов, которые будут рассмотрены далее.

Переменные могут использоваться в выражениях наряду с численными и строковыми значениями (численными и строковыми константами). Например, если в переменной val содержится значение "text = ", то результатом выражения val + "sometext" будет строка "text = sometext".

Внимание!

Регистр символов в именах переменных, а также функций, классов (на будущее) в JavaScript имеет значение. Это значит, что, например, val и Val – это различные переменные.

Перед использованием переменные нужно объявлять. При этом тип переменной указывать не надо (определенных типов просто нет, есть тип, хранящий любые данные). Переменные объявляются при помощи ключевого слова var следующим образом:

var v1, somevar = «asd»;

Как видно, в одной строке можно объявить сразу несколько переменных. При объявлении переменных их можно инициализировать (оператор = используется для присвоения значений переменным). Неинициализированные переменные содержат значение undefined (в данном случае это переменная v1).

Переменные могут также содержать специальное значение null, означающее, что в переменной нет данных. Кроме того, можно использовать специальное значение NaN (Not a Number), сигнализирующее о том, что в переменной содержится не число.

Если необходимо определить тип выражения или тип значения, которое хранится в переменной, то можно использовать оператор typeof(выражение). Этот оператор возвращает строковые значения: number (для численных выражений или значения NaN), string (для строк), object (для значения null и ссылок на объекты).

Простые и составные операторы

Простыми операторами называются единичные конструкции, выполняющие какое?нибудь действие, например вычисление значения одного выражения, присвоение значения, вызов функции и др.

В отличие от простых операторов, составной оператор может содержать в себе любое количество простых или вложенных операторов. Составной оператор задается (часто называемый блоком) при помощи скобок {}. Внутри этих скобок помещаются простые или вложенные составные операторы. Составные операторы используются, если нужно поместить несколько операторов, но ожидается присутствие только одного. В качестве примера рассмотрим оператор if (то, как работает этот оператор, описано далее). В теле оператора if ожидается наличие только одного оператора, выполняющего какое?либо действие, например:

```
if (a == 1) a++;
```

Что же делать, если нужно выполнить последовательность более чем из одного оператора? Ответ: применить составной оператор так, как это сделано ниже:

```
if (a == 1) {
a++;
//Другие действия...
}
```

Операторы языка JavaScript

Арифметические операторы. Инкремент и декремент

В языке JavaScript присутствуют стандартные для языков программирования арифметические операторы, позволяющие производить вычисления с численными и строковыми значениями (для строк только оператор +).

К арифметическим операторам JavaScript относятся: + (сложение), – (вычитание), * (умножение), / (деление). В дополнение к ним присутствует

оператор взятия остатка от деления %. Все указанные операторы являются бинарными (в том смысле, что принимают два значения и возвращают одно). Кроме указанных операторов, существует еще и унарный оператор –, инвертирующий значение аргумента (например –123, –val).

В JavaScript предусмотрена также удобная возможность записи выражений вида i = i + 1, i = i – 1, i = i * j и пр., где i – произвольная переменная, а j – произвольное выражение. Первые два выражения сокращенно записываются как инкремент и декремент: i++ и i–. Третье выражение и подобные ему можно сократить, применив следующие операторы:

• оператор –=, то есть i = i – j эквивалентно i –= j;

• оператор +=, то есть i = i + j эквивалентно i += j;

• оператор *=, то есть i = i * j эквивалентно i *= j;

• оператор /=, то есть i = i / j эквивалентно i /= j;

• оператор %=, то есть i = i % j эквивалентно i %= j.

Кроме того, предусмотрены соответствующие операторы &=, ^=, |= для двоичных операторов и <<=, >>=, >>>= для операторов сдвига.

Логические операторы и операторы сравнения

И логические операторы, и операторы сравнения возвращают результат – логическое значение true или false. Однако логические операторы принимают аргументы логического типа, в то время как операторы сравнения сравнивают значения произвольного типа. И логические операторы, и операторы сравнения языка JavaScript приведены в табл. 12.3.

Таблица 12.3. Логические операторы и операторы сравнения

Оператор	Описание	Оператор	Описание
==	Равно	<	Меньше
!=	Не равно	===	Строго равно (также возвращает false, если операнды несовместимых типов)
>=	Больше или равно	&&	Логическое И. Возвращает true, если оба операнда имеют значение true

Оператор	Описание	Оператор	Описание
>	Больше	||	Логическое ИЛИ. Возвращает true, если хотя бы один из операндов имеет значение true
<=	Меньше или равно	!	Логическое НЕ. Инвертирует значение операнда. Унарный оператор

Двоичные операторы

Двоичные операторы применяют при необходимости осуществления операций над численными (обычно целыми) значениями на уровне их битового представления. Здесь не рассматривается подробно, как представлены числа в памяти компьютера, а описываются только операторы.

Итак, двоичных операторов в JavaScript семь. Эти операторы перечислены в табл. 12.4.

Таблица 12.4. Двоичные операторы JavaScript

Оператор	Описание
&	Выполняет побитовую операцию И для двух аргументов
|	Выполняет побитовую операцию ИЛИ для двух аргументов
~	Выполняет инверсию всех битов аргумента (включая знаковый бит). Унарный оператор
^	Выполняет побитовую операцию ИЛИ-НЕ (XOR) для двух аргументов
<<	Выполняет сдвиг битов числа влево на количество позиций, указанное в правом аргументе (сдвиг на один бит влево эквивалентен умножению числа на 2). При сдвиге младшие разряды заполняются нулями
>>	Выполняет сдвиг битов числа вправо на количество позиций, указанное в правом аргументе (сдвиг на один бит вправо эквивалентен делению числа на 2). При сдвиге старшие разряды заполняются значением знакового бита
>>>	Аналогичен >>, но старшие разряды заполняются нулями (эквивалентен делению на 2 только для положительных чисел)

Приоритет операторов

Все операторы, которые могут быть использованы в выражениях, имеют свой приоритет, который учитывается при определении очередности вычисления значений подвыражений. Ниже приведен список операторов согласно их приоритету (от наивысшего к самому низкому).

1. ++, –, – (унарный), ~, !,typeof.
2. *, /, %.
3. +, -.
4. <<, >>, >>>.
5. >, >=, <, <=.
6. ==, !=, ===.

7. &.

8. ^.

9. |.

10. &&.

11. ||.

12. =, +=, –=, *=, /=, %=, &=, |=, ^=.

Несмотря на приоритет операторов, очередность вычисления подвыражений может быть изменена использованием скобок () любой вложенности. Выражение, заключенное в скобки, всегда вычисляется раньше других на том же уровне вложенности.

Условные операторы

При программировании на JavaScript можно использовать три условных оператора: if, select и оператор ? (именно вопросительный знак). Последний из операторов является самым простым, поэтому рассмотрим его в первую очередь.

Оператор ? используется как сокращенная версия оператора if при необходимости рассчитать значение одного из двух выражений в зависимости от истинности или ложности условия. Оператор имеет следующий формат:

условие ? выражение1 : выражение2

Здесь условие – логическое выражение (результат true или false). Выражение1 вычисляется в случае истинности выражения условие, иначе вычисляется значение выражения выражение2. Оператор ? возвращает значение (подобно любому другому оператору, например = или *), равное значению вычисленного выражения. Ниже приведено несколько примеров использования оператора ? (для большей наглядности выражения часто заключают в скобки):

```
a = (b > 3) ? b : 3; //Значение переменной a будет не меньше 3
a = (text == "continue") ? (a+=2) : a;
```

Следующим рассмотрим оператор if, который позволяет выбрать выполнение одной из двух последовательностей операторов в зависимости от истинности или ложности выражения?условия. Оператор if имеет следующий формат:

```
if (условие) оператор1
else оператор2
```

Если значение выражения условие равно true, то выполняется оператор1 (это может быть как простой, так и составной оператор), в противном случае выполняется оператор2 (также или простой, или составной оператор). Часть else оператор2 является необязательной. Ниже приведено несколько примеров использования оператора if:

```
if (b != 0) a /= b; //Проверяется отсутствие деления на ноль
else {
//Какие-то действия по информированию (в данном случае ничего)
}
if (a > 12)
if (a<25); //Действия при 12 < a < 25
else; //Действия при a > 25
```

В приведенном примере проиллюстрирована одна проблема, с которой часто сталкиваются начинающие программисты на C?подобных языках. Здесь специально проставлены отступы так, чтобы проиллюстрировать тот факт, что ключевое слово else относится к последнему по порядку оператору if. Если бы использовался блок, то принадлежность else была бы очевидной:

```
if (a > 12){
if (a<25) ; //Действияпри 12 < a < 25
else ; //Действия при a > 25
}
```

Напоследок осталось рассмотреть последний из условных операторов – оператор множественного выбора switch. Он позволяет выбрать одну из

многих альтернатив в зависимости от значения заданного выражения. Формат оператора приведен ниже:

```
switch (выражение){
case выражение1:
операторы1
case выражение2:
операторы2
...
default:
операторы_по_умолчанию
}
```

Оператор switch работает следующим образом. Сначала вычисляется значение выражения выражение. Далее это значение сравнивается с выражениями при каждом ключевом слове case сверху вниз. Если, например, значение выражение совпало со значением выражение2, то выполняется последовательность операторов операторы2. Выполнение продолжается до тех пор, пока не будет встречен оператор break либо выполнение не дойдет до конца тела оператора switch (закрывающая скобка }). Если перед следующим ключевым словом case отсутствует оператор break, то выполнится последовательность операторов операторы3 и т. д. Ключевое слово default используется для того, чтобы задать последовательность операторов, которые должны выполниться при несовпадении значения выражение со всеми выражениями при всех ключевых словах case. Для иллюстрации сказанного приводится пример использования оператора switch:

```
switch (var){
case 1:
//Операторы выполнятся при var == 1
break;
case 2:
```

```
//Операторы выполнятся при var == 2
case 3:
//Операторы выполнятся при var == 2 или var == 3
break;
case 4:
//Операторы выполнятся при var == 4
default:
//Операторы выполнятся при var != 1 &&var != 2 &&var != 3
}
```

Циклы

Язык JavaScript поддерживает три вида циклов: for, while и do-while. Начнем с более простых циклов while и do-while. Цикл while позволяет выполнять нужные действия, пока истинно выражение?условие. Формат оператора while следующий:

```
while (условие) оператор
```

Здесь условие – логическое выражение (аналогично операторам ifи ?), а оператор – простой или составной оператор, выполняемый при каждой итерации цикла. Пример использования цикла while:

```
var i = 0;
while (i<10){
//Какие-то действия...
i++; //Не забываем увеличить итератор, чтобы случайно
//не организовать бесконечный цикл
}
```

Следующий оператор цикла do-while имеет следующий формат:

```
do оператор while (условие)
```

Этот оператор цикла аналогичен оператору while с тем лишь отличием, что условие в цикле do-while проверяется после выполнения каждой итерации. Это значит, что оператор выполнится как минимум один раз.

Предыдущий пример можно записать с использованием оператора do-while следующим образом:

```
var i = 0;
do{
//Какие-то действия...
i++; //Не забываем увеличить итератор, чтобы случайно
//не организовать бесконечный цикл
}while (i<10);
```

Теперь рассмотрим оставленный напоследок цикл for. Оператор for имеет следующий формат:

for (выражение1; условие; выражение2) оператор

Значение выражения выражение1 рассчитывается перед первой итерацией цикла. Обычно это инициализация счетчика или другой переменной, нужной в цикле. Операторы в теле цикла (оператор) выполняются до тех пор, пока истинно значение выражения условие. Перед второй и последующей итерациями вычисляется значение выражения выражение2 (обычно это выражение по изменению переменной цикла). Для демонстрации использования цикла for ниже приводится пример (аналог примеров для циклов while и do-while):

```
vari;
for(i=0; i<10; i++){
//Какие-то действия...
}
```

Операторы break и continue

Оператор break, помимо прерывания выполнения последовательности операторов внутри оператора switch, используется для прерывания итерации циклов. В следующем примере выполнение цикла for прерывается как раз с помощью оператора break:

```
vari;
for (i=0; i<10; i++){
```

```
//Действия...
if (i == 5) break;
}
```

Если в теле цикла встречается оператор continue, то остальные операторы игнорируются, а выполнение переходит на проверку условия цикла. Например, в следующем цикле суммируются значения от 1 до 10 (с помощью оператора continue игнорируются значения 5 и 7):

```
vari, sum = 0;
for (i=0; i<10; i++){
if (i == 5 || i == 7) continue;
sum += i;
}
```

Оператор запятая

Оператор полезен в тех случаях, когда нужно одновременно вычислить значение нескольких выражений в том месте, где допускается запись только одного. Рассмотрим этот случай на примере оператора for. Пусть нужно, чтобы в цикле было два итератора, но очень не хочется писать увеличение (уменьшение) одного из них в теле цикла. С использованием оператора , (запятая) можно разрешить проблему следующим образом:

```
var i, j;
for (i=0, j=100; i<j; i++, j–){
//Действия ...
}
```

Выражения, разделенные оператором , (запятая), вычисляются слева направо. При этом возвращаемым значением будет значение самого левого выражения. В следующем примере значение переменной res будет равным 3, а не 6:

```
var res, val = 2;
res = val+=1, val=5;
```

Из этого примера можно подчеркнуть одну особенность всех операторов присваивания языка JavaScript: они тоже возвращают значение. Корректными являются конструкции вида:

a = 1 + (b = c = d = 25);

Здесь значением переменной a будет 26, а остальных переменных – 25. Все сказанное об операторе , (запятая) не касается использования этого оператора при вызове функций.

Функции

Функции применяются в тех случаях, когда недостаточно программы, представляющей собой простую последовательность операторов, выполняющуюся один раз. В этом случае применение функций позволяет сгруппировать операторы и выполнять их произвольное количество раз (вызывая функцию из любого места программы). Упрощенно о функции можно сказать, что это именованная последовательность операторов.

Пользовательские функции

Пользовательская функция – любая функция, созданная программистом (а не встроенная в интерпретатор). Для создания пользовательских функций применяется следующая конструкция:

```
functionимя_функции(параметр1, параметр2, ...){
//Операторы тела функции
return выражение
}
```

Здесь имя_функции – имя, идентификатор функции. На него распространяются те же ограничения, что и на имя любой переменной. В скобках задается список формальных параметров функции (она может и не иметь параметров). Каждый элемент этого списка – идентификатор переменной. Переменные с указанными в списке идентификаторами могут использоваться в функции так, как будто они были объявлены с использованием var (дополнительно эти переменные инициализируются

значениями, переданными в функцию). Для возврата результата выполнения функции используется оператор return.

Рассмотрим пример функции, принимающей два значения и возвращающей сумму переданных ей значений:

```
functionsum (v1, v2){
//Вычисляем сумму двух значений
return v1 + v2;
}
```

Для вызова этой функции в любом месте программы, а также в теле другой функции должна использоваться следующая запись:

```
sum(выражение1, выражение2)
```

Значения двух указанных выражений и будут переданы в функцию в качестве значений переменных v1 и v2. Если результат, возвращаемый функцией, используется, то в этом случае вызов функции записывают в правой части операторов присваивания либо как аргумент оператора в выражении, например:

```
summa = sum(3, 4); //В переменную summa заносится 7
summa = sum(3, sum(4, 5)); //Суммируются три значения
```

Встроенные функции JavaScript

Программист на JavaScript может использовать некоторые встроенные в интерпретатор функции, позволяющие значительно упростить некоторые аспекты программирования, связанные с преобразованиями строковых значений, вычислениями значений выражений. Эти функции приведены в табл. 12.5.

Таблица 12.5. Встроенные функции JavaScript

Функция	Описание
escape(строка)	Преобразует переданную строку к виду, пригодному для использования в качестве URI (все недопустимые для URI символы заменяются своими кодами). Например, строка "http://сайт.com" будет преобразована к виду "http%3A//%u0441%u0430%u0439%u0442.com"
unescape(строка)	Заменяет все коды символов в переданной строке самими символами. Может применяться для декодирования URI
eval(строка)	Вычисляет значение выражения, записанного в строке. Например, eval("3+2") вернет значение 5

Глобальные и локальные переменные

При рассмотрении функций нельзя обойти вниманием такой вопрос, как область видимости переменных. Итак, переменные, объявленные вне функции, являются глобальными переменными. К ним можно получить доступ из любого места программы.

Переменные можно объявлять и внутри функций с использованием все того же ключевого слова var. Такие переменные являются локальными, и доступ к ним можно получить только внутри той функции, в которой они объявлены. Объявление любой локальной переменной может находиться в любом месте функции. При этом переменную, объявленную ниже в тексте программы, можно использовать выше, потому что интерпретатор при входе в функцию создает сразу все объявленные в ней переменные. В месте же объявления переменных происходит только их инициализация. Следующий пример является абсолютно корректным:

```
var j;
i = 1;
j = i;
vari = 2;
```

Рассмотрим также, что происходит, если имеет место такой случай:

```
var i = 1; //Глобальная переменная
function f(){
var i = 2; //Локальная переменная
return i;
```

```
}
```

В данном случае функция возвратит значение 2, то есть в операторе return используется значение локальной переменной.

Ссылки на функции

Кроме значений описанных ранее типов, переменным можно присваивать имена функций. После такого присвоения функцию можно вызвать, используя в качестве имени функции имя переменной, как в приведенном ниже примере:

```
varpfun = sum; //Присваиваем ссылку на функцию
varres = pfun(1,3); //Вызываем функцию sum
function sum(arg1, arg2){
return arg1+arg2;
}
```

Это может оказаться полезным во многих случаях. Например, в функциях сортировки массивов в число параметров функции сортировки часто включают параметр, предназначенный для передачи ссылки на функцию, сравнивающую элементы массива и возвращающую строго определенные значения (см. описание метода sort класса Array далее в тексте главы).

Кстати, оператор typeof возвращает значение function, если его параметром является имя функции или переменная, которой ранее присвоено имя функции.

Массивы

Использование массивов очень удобно, если нужно хранить большое количество однотипных значений. Массив – это список значений, хранящихся под одним именем. Доступ к каждому элементу массива осуществляется по номеру соответствующего элемента. Нумерация элементов массива в JavaScript начинается с нуля.

Массивы создаются в два этапа: объявляется переменная для хранения массива, после чего производится инициализация массива. Например:

```
vararr;
arr = [1,2,3,4,5];
```

или

```
vararr = [1,2,3,4,5];
```

Результатом выполнения приведенного фрагмента кода является создание массива, состоящего из пяти элементов. Доступ к элементам массива осуществляется с помощью оператора []. Операция получения значения третьего элемента массива выглядит следующим образом:

```
val = arr[2];
```

Конструкцию вида имя_массива[номер] можно использовать и в выражениях аналогично обычным переменным. Значения элементов массива можно сразу не указывать, то есть можно объявить пустой массив:

```
vararr;
arr = [];
```

Созданный массив при этом не содержит элементов. Для заполнения массива можно также использовать оператор []. Во время присвоения значений элементам массива проявляется следующая особенность: массив автоматически расширяется, если номер элемента, которому присваивается значение, превосходит номер последнего элемента в массиве. В следующем примере в результате выполнения присваивания массив будет содержать пять элементов, причем четыре первых из них будут иметь значение undefined:

```
vararr;
arr = [];
arr[4] = 5;
```

Для создания многомерного массива нужно элементам уже объявленного массива также присвоить массивы. Создание пустого двумерного массива (четыре строки с неопределенным количеством элементов) может выглядеть следующим образом:

```
vararr;
arr = [[], [], [], []];
```

Для обращения к элементам многомерного массива используется тот же оператор [], например:

arr[0][1] = 124;

arr[2][1] = arr[0][1];

Массивы в JavaScript являются объектами (оператор typeof для переменных массивов возвращает значение object). К тому же отсутствуют ограничения относительно формы, например, двумерных массивов: можно создавать как прямоугольные, так и треугольные или другие массивы произвольной формы. В ряде случаев это весьма полезно, но иногда может и запутать.

Размер массива можно получить с помощью свойства length этого объекта. Что такое свойство объекта и как его использовать, рассказано в следующем разделе.

Глава 19. Иерархия объектов JavaScript. Объект Location

Классификация объектов JavaScript;

- объекты браузера;
- объект **location**;
- свойства и методы объекта **location**;
- объект **history**;
- свойства и методы объекта **history**;
- объект **navigator**;
- свойства объекта **navigator**;
- отступление: определение браузера;
- методы объекта **navigator**;
- объект **screen**;
- свойства объекта **screen**.

«Словарный запас» JavaScript

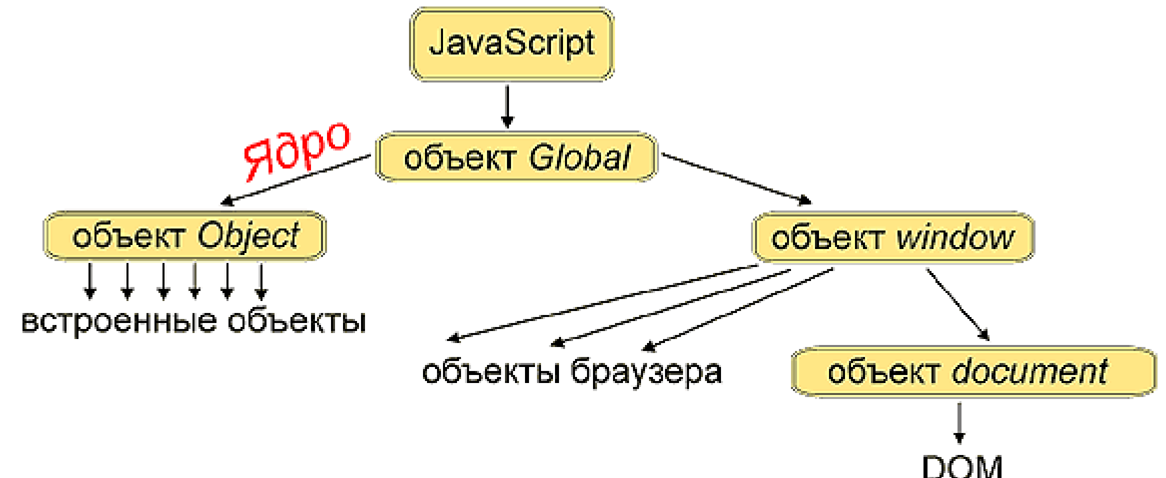
Объекты, с которыми работает JavaScript, можно условно разделить на 2 категории:

1. **Встроенные объекты** (собственно объекты **ядра**), с которыми мы уже познакомились достаточно подробно.

2. **Клиентские объекты**, «родителем» которых является объект **window**. Их тоже можно условно разделить на 2 категории:

а). **Объекты браузера**. К ним относятся элементы интерфейса браузера — окно, показывающее web-страницу, панели инструментов, строка состояния, полосы прокрутки, а такде информация о типе и версии браузера.

б). **Объекты HTML-документа (или XML-документа)** — все те элементы, которые указаны в коде web-страницы.



Таинственный невидимый объект **Global** — прототип всех объектов JavaScript.

Левая ветка — ядро, объект **Object** — прародитель всех **встроенных объектов** (объектов ядра).

Правая ветка — **клиентские объекты**, с которыми работает JavaScript. Мы уже немного прикоснулись к их пращуру — объекту **window** — во второй части (уроки **14**, **15**, **16**, **17**).

Объекты web-страницы, заключённые в объект **document**, представляют собой иерархию **DOM** — DocumentObjectModel (объектная модель документа). Изучению этой огромной, необъятной модели и будет посвящено большинство уроков третьей части.

Но начнём с чего попроще. А именно — с объектов браузера.

Объекты браузера

Существует путаница в классификации клиентских объектов. Часто можно встретить в одном перечне объекты **window**, **location**, **history**, **document**, **navigator**. (Объект **screen** часто вообще не упоминается в перечнях.) Это не совсем точно. Объекты **location**, **history**, **document** и **navigator** (а также **screen**) являются потомками (свойствами) **window**. В свою очередь, **document** имеет так много потомков (свойств), что его нужно рассматривать отдельно и подробно. А более непритязательные объекты — **location**, **history**, **navigator** и **screen** — целесообразно рассмотреть в одной связке, условно назвав её **«объекты браузера»**.

Итак, приступим к изучению этих объектов.

Примечание

У объекта **document** также есть свойства (объекты) *location* и *history*. Прошу это сразу запомнить и постараться не путать «однофамильцев».

Объект location

Свойства объекта описывают и хранят местонахождение текущего документа — например, адрес URL.

При управлении объектом **location** существует возможность изменять адрес URL документа. Объект **location** связан с текущим объектом **window** — окном, в которое загружен документ.

Синтаксис

```
[переменная_окна.]location.свойство;
```

переменная_окна — необязательная переменная, задающая окно, к которому обращается объект. Если переменная опущена, подразумевается свойство текущего окна.

свойство — свойство объекта **location**, к которому обращена инструкция.

Внимание!

Свойство *location* объекта *window* легко перепутать со свойством *location* объекта *document* (значение *document.location* изменить нельзя, тогда как *window.location* — можно. При загрузке страницы значение *document.location* автоманически присваивается объекту *window.location*.

Обращение к фрейму

Можно также изменить содержимое фрейма во фреймосодержащем документе при помощи свойства **parent** — синонима объекта **window**, обозначающего окно верхнего уровня, если окон несколько.

Синтаксис

parent.**frames**[n].**location** = *"адрес_URL"*;

n — число, указывающее номер запрашиваемого фрейма в коллекции **frames**.

"адрес_URL" — URL документа, который инструкция загрузит в этот фрейм.

Свойства объекта location

href

Полный адрес URL текущего документа, например:

location.href = *"http://myhost.ru/aboutme.html"*

Свойство **href** является свойством объекта **location** по умолчанию. То есть отсутствие свойства при указании объекта **location** равносильно определению **location.href**. Например, следующие две инструкции эквивалентны и устанавливают одно и то же значение:

location = *"http://myhost.ru/aboutme.html"*

location.href = *"http://myhost.ru/aboutme.html"*

hash

Часть URL после символа #, которая соответствует местоположению якоря в документе (если таковой имеется).

Символ # не включается в имя свойства при его установке.

location.href = *http://myhost.ru/aboutme.html#ancor1"*

location.hash = *"ancor1"*

pathname

Часть адреса URL, описывающая каталог, в котором находится документ, включая начальный символ косой черты:

location.href = *"http://myhost.ru/images/pict1.jpg"*

location.pathname = *"/images/pict1.jpg"*

protocol

Префикс адреса URL, описывающий протокол обмена. Иными словами, компонент URL до первого двоеточия, включая его. Возможные значения: **"http:"**, **"ftp:"**, **"mailto:"** и **"file:"**.

host

Часть адреса текущего документа, включающая имя хоста и порта сервера (если определён) через двоеточие, например:

location.host = *"myhost.ru:80"*

Что такое порт?

hostname

Первая половина свойства **host** (без порта):

location.hostname = *"myhost.ru"*

port

Вторая половина свойства **host** (только порт):

location.port = *"80"*

Примеры свойств

С помощью свойств объекта location можно манипулировать разными участками полного адреса URL. Полный адрес - то есть location.href (или просто location) складывается из следующих «кубиков» (слэши после протокола необходимо доьавить вручную):

location.protocol + "//" + location.hostname + location.pathname Проверим данную страницу:

```
document.write(location.href)
```

Результат: http://froland.ru/samodel/lsn_js31.html

Примечание

Если страница находится у вас на компьютере, то после протокола (file:) появятся не два, а три слэша:

file:///F:/site/!my2008/lsn_js31.html

Третий слэш «закрывает» пустую строку, которая появляется вместо свойства hostname. В браузере Operahostname будет определяться как localhost:

file://localhost/F:/site/!my2008/lsn_js31.html

```
document.write(location.protocol)
```

Результат: http:

```
document.write(location.hostname)
```

Результат: froland.ru

Примечание

Если страница находится у вас на компьютере, то свойство возвратит пустую строку (Opera выдаст localhost).

```
document.write(location.pathname)
```

Результат: /samodel/lsn_js31.html

Примечание

Если страница находится у вас на компьютере, то свойство возвратит полный путь после протокола file://. При этом в IE слэши после обозначения диска будут обратными:

/F:\site\!my2008\lsn_js31.html

В других браузерах — прямыми:

/F:/site/!my2008/lsn_js31.html

```
document.write(location.protocol + "//" + location.hostname +
location.pathname)
```

Результат: http://froland.ru/samodel/lsn_js31.html

Примечание

Если страница находится у вас на компьютере, то в IE слэши после обозначения диска будут обратными:

file:///F:\site\!my2008\lsn_js31.html

Браузер Opera добавит в качестве хоста localhost:

file://localhost/F:/site/!my2008/lsn_js31.html

Итак, всё вернулось на круги своя.

Методы объекта location

reload()

Синтаксис

location.**reload**([проверка])

проверка — дополнительное Булево значение, по умолчанию — **true**.

Выполняет жесткую перезагрузку текущего документа окна (определённого свойством **location.href**). Собственно, это же делает кнопка браузера **«обновить»** (**«refresh»**).

По умолчанию (без аргумента) или с аргументом **true** метод заставляет браузер проверить время последнего изменения документа и загрузить его либо из кэша(если не был изменён), либо с сервера (соответствует кнопке **«обновить»**).

Если в качестве аргумента указано **false**, то браузер перезагрузит текущий документ с сервера без какой-либо предварительной проверки. Это соответствует нажатию на кнопке **«обновить»** при удержанной кнопке **Shift** (**Shift+Refresh**)или **Ctrl+F5** в InternetExplorer.

replace()

Синтаксис

```
location.replace("URL")
```

URL — строка содержащая полный или относительный URL документа, который будет загружен в текущее окно или фрейм.

Загружает новый документ в текущее окно, не занося предыдущую страницу в список **history** (об объекте **history** — чуть ниже). То есть при нажатии кнопки браузера **«назад»** (**«back»**) пользователь не сможет вернуться на предыдущую страницу.

Это полезно применять при переадресации сайта.

Предположим, сайт переехал, но постоянные пользоыватели помнят прежний адрес. Пока они не привыкли к новому, на старом адресе можно поместить пустую страницу, в <head> которой будет скрипт переадресации.

Допустим, мы сделали это через свойство **href**:

```
<script type="text/javascript">
location.href = "новый_адрес_страницы"
</script>
```

Скрипт приведёт нас на нужную страницу, но если нам зачем-либо понадобится нажать кнопку «назад» мы будем снова попадать на страницу переадресации, которая вновь пошлёт нас по указанному адресу, то есть кнопка «назад» окажется «зацикленной», и последняя страница станет тем самым «отцепленным вагоном» Маршака.

Но можно сделать грамотную переадресацию через метод **replace()**:

```
<script type="text/javascript">
location.replace("новый_адрес_страницы")
</script>
```

Преимущество этого метода в том, что промежуточная страница не записывается в массив **history**, и кнопка «назад» корректно приведёт нас на страницу, на которой мы были до этого.

Объект history

Пока мы сидим в интернете и переходим со страницы на страницу (то есть, научно говоря, выполняем «сессию браузера»), браузер ведёт журнал,

содержащий список адресов страниц (URL), которые мы посетили. Эти адреса записываются в массив, который используется браузером при нажатии кнопок «Назад» («Back») и «Вперёд» («Forward»). Массив содержится в объекте **history**. В многооконных браузерах каждое окно имеет собственный объект **history**.

Свойства объекта history

current

URL текущего окна в массиве **history**. Появляется при загрузке первой страницы. Обновляется при загрузке каждой следующей страницы в то же окно (вкладку).

next

URL следующего окна в массиве **history**. Появляется или обновляется после того, как мы нажали кнопку **«Назад»** (**«Back»**).

previous

URL предыдущего окна в массиве **history**. Появляется или обновляется при каждом переходе на новую страницу.

length

Число элементов в массиве просмотренных страниц (размерность массива **history**).

Примечание

В браузерах IE и Opera начальное значение равно **0** (то есть свойство возвращает не размерность массива, а номер текущего элемента), в браузерах Mozilla, Firefox и Google Chrome — это **1** (то есть именно размерность массива).

Для проверки можете поместить на страницу такой скрипт,

```
<script type="text/javascript">
document.write(history.length)
</script>
```

а потом открыть её в разных браузерах.

Методыобъекта history

back()

Синтаксис

```
history.back()
```

Перейти на предыдущую страницу (эмуляция кнопки «Назад»). Если предыдущей страницы нет, то вызов метода ничего не даст.

forward()

Синтаксис

```
history.forward()
```

Перейти на следующую страницу (эмуляция кнопки «Вперёд»). Вызов метода на последней странице ничего не даст.

go()

Синтаксис

```
history.go(шаг)
/* или */
history.go("URL")
```

Аргументы

шаг — целое число, показывающее, на сколько шагов вперёд/назад надо переместиться. Значение **0**перезагружет текущую страницу; значение **-1** эквивалентно **back()**; значение **1** эквивалентно **forward()**. Если шаг лежит за границами истории посещений (например **-1** для первой страницы), то метод ничего не не даст.

URL — адрес документа, сохранённого в списке хронологии.

Перемещает на определённую позицию в массиве history или на адрес страницы, содержащейся в массиве.

Объект navigator

Объект **navigator** представляет ряд свойств браузера.

Все браузеры имеют несколько общих ключевых свойств, но у каждого браузера есть свои дополнительные свойства. Объект **navigator** позволяет

увидеть, какой браузер и какая его версия выполняют скрипт, и поэтому бывает нужен при создании кроссбраузерных скриптов.

Свойства объекта navigator

Для «общего развития» привожу все свойства этого объекта. Хотя, как вы увидите, некоторые из них на сегодняшний день достаточно бессмысленны. Но языки программирования пишутся «на вырост», и со временем «балластовые» свойства могут оказаться актуальными.

Свойства перечислены по алфавиту, а не по значимости.

appCodeName

Показывает кодовое имя браузера. У всех браузеров (IE, Netscape, Mozilla, Firefox, Opera, GoogleChrome и т.д.) кодовое имя — Mozilla. Так что это свойство пока практического значения не имеет.

appMinorVersion

Всем программам традиционно присваиваются «большие» (major) и «малые» (minor) номера версии. «Большая» версия обозначает принципиально новый вариант программы. «Малая» — улучшенный и исправленный вариант текущей версии программы. Обычно «малая» версия пишется после точки вслед за «большой». Например, 4.03.

По идее свойство **appMinorVersion** должно возвращать номер «малой» версии браузера (тот, что после точки). Однако в большинстве браузеров возвращается **undefined**, т.е. «не определено». Так что свойство опять же чисто номинальное.

appName

Показывает имя браузера.

IE	MicrosoftInternetExplorer
Netscape, Mozilla, Firefox, Google Chrome	Netscape
Opera	Opera*

*** в браузере Opera есть «обманная» настройка, которая позволяет ему «мимикрировать» под другие имена. Зачем — непонятно.**

Свойство **appName** позволяет в общих чертах определить тип браузера. На заре интернета этого вполне хватало. Однако сейчас для точного и корректного определения стоит пользоваться другими приёмами (см. ниже).

appVersion

Показывает полный номер версии браузера и — в скобках — платформу операционной системы и заданный по умолчанию язык — часть информации, возвращаемой свойством **userAgent** (см. ниже). IE и Google Chrome показывают более полную информацию **userAgent**.

В Firefox и Google Chrome возникает путаница с номером версии. Так, и Firefox 3, и Google Chrome показывают номер версии 5.0.

Почему так?

С выходом IE и Netscape 4-х версий произошла «бархатная революция» в «сближении и взаимопонимании» этих браузеров. Поэтому часто в ветвлениях кода для разных браузеров определение версии 4 используется как «индикатор соответствия». Это делается извлечением номера «большой» версии из свойства **appVersion** с помощью метода **parseInt()**, например:

```
if (parseInt(navigator.appVersion) >= 4)
{// код для браузеров 4 версии и выше}
else
{// код для старых браузеров}
```

Отсюда понятно, что «завышенный» номер версии используется в Firefox и Google Chrome для «подстраховки».

Итак, мы выяснили, что при всей громоздкости и «глючности» это свойство может найти полезное применение в кроссбраузерных кодах.

Значения свойства **appVersion** в некоторых браузерах:

IE 6

```
4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; .NET CLR 1.1.4322)
```

Firefox 3.63

5.0 (Windows; ru) *// скромненько и со вкусом*

GoogleChrome*

5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/534.16 (KHTML, like Gecko) Chrome/10.0.648.204 Safari/534.16

*** почему-то русская версия браузера всё равно возвращает en-US.**

Opera 8.54

8.54 (Windows NT 5.1; U; ru)

Ваш браузер

5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36

Что означает абракадабра, похожая на шпионский шифр, рассмотрим в свойстве **userAgent**.

browserLanguage

Если вы обратили внимание, в IE свойство **appVersion** возвращает что угодно, кроме языка.

Свойство **browserLanguage** — язык браузера по умолчанию — специфическое свойство IE (его понимает также Opera 7 и выше). Остальные браузеры возвращают **undefined**.

cookieEnabled

Свойство определяет, позволяет ли данный браузер читать и записывать данные cookie. Оно используется в скриптах, работающих с «куками», например:

```
if (cookieEnabled)
{setCookieData(data)}
```

Возвращает булево значение (если да, то **true**).

cpuClass

Ещё одна «примочка» IE. Возвращает информацию о CPU клиентского компьютера (кто не знает, CPU — это процессор). Процессоры **Intel**

возвращают **x86**, **PowerPCMacintosh** — **PPC**. Ни тебе скорости, ни номера модели. И зачем, однако?..

javaEnabled

Булево значение. Проверяет, включён ли Java в установках браузера (если да, то **true**).

language

То же, что **browserLanguage**, но для браузеров с **appName** Netscape. Opera понимает оба свойства.

onLine

Булево значение. Проверяет, установлен ли браузер для просмотра online или offline (если да, то **true**). Opera не поддерживает.

platform

Возвращает название операционной системы или аппаратной платформы браузера. Для Windows 95/NT, значение — Win32; для Macintosh, на PowerPC CPU, значение — MacPPC. Использование этого свойства для определения базовых средств клиента в условном выражении могут помочь оптимизировать вывод страницы для каждого устройства. Например:

```
if (navigator.platform == "Win32")
{ document.write(// содержание, поддерживаемое компьютерами с операционной системой Windows 95/NT computer )}
```

product

Возвращает название торговой марки браузера. Является частью строки **userAgent**. Браузеры с **appName** «Netscape» возвращают «Gecko». IE и Opera — не поддерживают.

systemLanguage

Код для заданного по умолчанию языка, используемого операционной системой. При многоязычном содержании можно использовать это свойство для ограничения содержания, например:

```
if (navigator.systemLanguage == "nl")
```

```
{document.write(// голландское содержание)}
```

userAgent

При посещении сайта браузер обычно посылает серверу информацию о себе. Это текстовая строка с информацией о названии и версии приложения, операционной системе компьютера и языка. Эту информацию и содержит свойство **userAgent**.

На многих комапьютерах установлены пвкеты .NET Framework — технологии, поставляемой Microsoft. Свойство **userAgent** браузера IE указывает также версии этих пакетов. CLR (CommonLanguageRuntim) — «общеязыковая исполняющая среда» — компонент пакета Microsoft .NET Framework, выполняющий программы, написанные на .NET-совместимых языках программирования.

Поскольку в браузерах «согласья нет», многие сайты для работы с «лебедем, раком и щукой» используют **userAgent** для определения браузера и в зависимости от получаемого значения посылают соответствующее содержимое веб-страниц. Из-за этого многие браузеры стали «прятать» или «подделывать» **userAgent**. Отсюда использование браузером IE, а затем и Google Chrome строки **userAgent**, начинающейся с «Mozilla/». Отсюда и «глючная» опция браузера Opera, позволяющая пользователю выбирать для браузера поддельный «псевдоним».

Из-за этой путаницы для определения браузера лучше использовать не свойство целиком, а его отдельные наиболее характерные элементы, находя их методом **indexOf()**. Например, проверить наличие желаемой строки можно так:

```
if (navigator.userAgent.indexOf("MSIE") != -1)
varisIE = true
```

Кроме популярных «лебедя, рака и щуки», могут встретиться малопопулярные «медвели» и «крокодилы» с каким-нибудь совсем «левым» **userAgent**. И для них опренделение через **userAgent**может не сработать

вообще. Поэтому определение типа браузера через **userAgent** может оказаться таким же некорректным, как и через **appName**.

После перечисления оставшихся свойств приведен способ наиболее корректной проверки (хотя и она не даёт стопроцентной гарантии).

userLanguage

Значение по умолчанию языка браузера, основанное на установке операционной системы **user profile**(если таковая имеется). По умолчанию соответствует свойству **browserLanguage**. Поддерживается только браузерами IE и Opera.

userProfile

Объект, который позволяет непосредственно по запросу сценария обратиться к личной информации, сохраненной в параметрах пользователя (для Win32 версий IE 4+).

В данном обзорном разделе мы не будем рассматривать громоздкий, но малоупотребительный объект **userProfile** с его многочисленными методами, тем более, что он существует в IE 6 и только в IE 6 (это, кстати, может пригодиться для определения версии браузера).

vendor

Возвращает название распространителя браузера. Не поддерживается IE и Opera. Firefox обычно выдаёт пустую строку. Значение для Google Chrome — «GoogleInc.».

На сегодняшний день может использоваться как определитель браузера Google Chrome.

Отступление: способы определения браузера

Определяя тип браузера, логичнее идти не от «вывески», а от его внутренних свойств. Хотя и «вывеска» может сыграть свою подсобную роль.

Самый элементарный способ

Чуть забегая вперёд, обратимся к свойствам документа (подробнее см. в следующем уроке). Принципиальное различие IE и браузеров семейства

Netscape в том, что первый поддерживает коллекцию **document.all**, а последние — нет.

Итак:

```
if (!document.all)
{
/* код для семейства Netscape
(восклицательный знак означает отрицание) */
}
```

Здесь мы определили вариант для браузеров семейства Netscape. В «противный случай» попадают браузеры, поддерживающие **document.all**: IE, Opera и, возможно, какие-то ещё мало популярные. Для их разделения воспользуемся «вывеской» IE.

```
else
{
   if (navigator.appName == "MicrosoftInternetExplorer")
   {
      // код для IE
   }
   else
   {
      // код для всех остальных, поддерживающих document.all
   }
}
```

Часто этого способа бывает достаточно.

Ещё один элементарный способ

Браузер IE поддерживает специальный синтаксис, который интерпретирует код только в IE. Этот код помещается в тэг комментариев, поэтому остальные браузеры видят просто комментарий и игнорируют его.

В дополнение к этому существует тэг <comment></comment>, который IE интерпретирует как комментарий (и, соответственно, игнорирует его

содержимое), а остальные браузеры просто читают его содержимое, как будто самого этого тэга нет.

Из этого можно соорудить следующую конструкцию:

```
<!--[if IE]>
Код для браузера IE
<![endif]-->
<comment>
// Код для других браузеров
</comment>
```

Ключевые слова для IE:

IE — любая версия браузера Internet Explorer;

IE 6 — Internet Explorer 6;

IE 7 — Internet Explorer 7;

IE 8 — Internet Explorer 8;

IE 9 — Internet Explorer 9;

lt — номер версии браузера меньше указанной;

gt — номер версии больше указанной;

lte — номер версии меньше или равен указанной;

gte — номер версии больше или равен указанной.

Например, <!--[ifgte IE 6]> означает любую версию IE, начиная с IE 6.

Более хитрые способы (обзор)

Если поискать, то у каждого браузера можно найти какие-нибудь «фишки», присущие ему одному.

Как мы уже знаем, Opera умеет прикидываться другими браузерами (в 10-й версии это безобразие прекратилось). Но при этом (начиная с 5-й версии) поддерживает объект JavaScript **window.opera**. Вот она и спалилась. Этот же объект может показать и версию:

```
window.opera.version()
```

Допустим:

```
if (window.opera != null)
```

```
{
if (window.opera.version() >= 8)
{
// Коддля Opera 8 иновее
}
else
  {
// Код для Opera старше 8
  }
}
else
{
// Код для других браузеров
}
```

Google Chrome имеет уникальный объект **window.chrome**. Версию определить сложнее, поскольку parseFloat(navigator.appVersion) выдаст «обманку» 5.0. Пока не будем лезть в дебри регулярных выражений (а это делается именно с их помощью).

Уникальные объекты Firefox — **window.sidebar** и **window.globalStorage**. С версией те же проблемы, что и у Google Chrome.

У Safari не обнаружилось уникальных объектов, но у него нет объекта **window.external**. Этого объекта лишена и Opera. Поэтому ветка для определения Safari может выглядеть так:

```
if ((!window.external)&&(!window.opera))
{
// Код для Safari
}
```

IE поддерживает объекты **window.all** и **window.ActiveXObject**.

В IE 6 и только в IE 6, как мы знаем, существует **navigator.userProfile**. В IE 8 появились объекты **window.Storage** и **window.Event**. В IE 7, соответственно, нет ни **userProfile**, ни **Storage**, ни **Event**.

Теперь, когда даны условия задачи, можете сами построить функцию для определения разных браузеров. В 4 части наших уроков с помощью регулярных выражений сделаем более подробную функцию для определения браузеров с версиями.

Методы объекта navigator

preference()

Через маркированные скрипты в Navigator 4 и выше можно обращаться к пользовательским параметрам настроек браузера. Они включают такие детали, как «разрешил ли пользователь загружать изображения» или «позволяются ли таблицы стилей». Большинство этих параметров настройки предназначено для скриптов, используемых сетевыми администраторами, чтобы устанавливать и управлять пользовательскими параметрами настройки Navigator.

Аргументы

name — название настройки как строка, типа general.always_load_images.

value — дополнительное значение, чтобы установить названное предпочтение.

taintEnabled()

Возвращает, включен ли в браузере «datatainting». Этот механизм защиты никогда не был полностью реализован в Navigator, но проверяющий это метод включён в более новые версии Navigator для обратной совместимости. IE 4+ также включает его для совместимости, хотя всегда возвращает **false**. Возвращает булево значение: **true|false**.

Объект screen

Объект **screen** определяет свойства экрана, на котором показан браузер. На их значения влияют многие параметры настройки панели управления.

Методов объект **screen** не имеет.

Свойства объекта screen

Высота и ширина видимой области монитора пользователя в пикселях. Не включает панель задач в 24 пикселя (Windows) или системную строку меню в 20 пикселей (Macintosh). Чтобы использовать эти значения при создании развёрнутого окна, необходимо корректировать верхнюю левую позицию окна.

Пример (окно на полуширины и всю высоту в левой верхней точке экрана, панель задач не заслоняется):

```
window.open("", "", "height=" + screen.availHeight + ", width=" +
screen.availWidth/2 + ", top=0, left=0");
```

Посмотреть

Примечание 1

В Windows XP на панель задач свойство availHeight отводит 30 пикселей (проверено методом тыка).

Примечание 2

IE 4/Macintosh ошибается в расчёте высоты строки меню (отсчитывает 24 пикселя вместо 20).

availLeft, availTop

Координаты в пикселях левой и верхней точек экрана. Всегда нуль.

Эти свойства поддерживают только браузеры семейства Netscape.

bufferDepth

Установка растрового буфера кадра (бит на пиксель).

Тут, видимо, требуется пояснение.

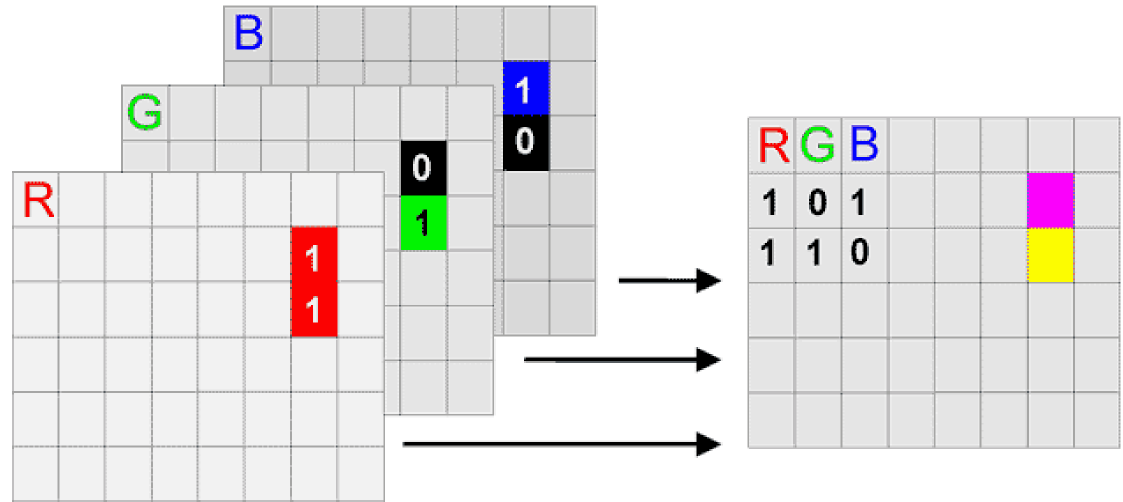
Буфер кадра — это большой непрерывный участок памяти компьютера, в котором для каждой точки (пикселя) в растре отводится как минимум один бит памяти. Эта память называется битовой плоскостью.

Бит памяти имеет только два состояния (двоичное 0 или 1). Поэтому, имея только одну битовую плоскость, можно получить лишь черно-белое изображение.

Применение нескольких плоскостей расширяет цветовые возможности. Количество цветов определяется формулой $\mathbf{2^N}$, где **N** — количество плоскостей. Так, используя простейший буфер из 3 плоскостей для основных цветов (красный, зелёный, синий) мы получим $2^3 = 8$ цветов.

	Красный	**Зелёный**	**Синий**
Чёрный	0	0	0
Красный	1	0	0
Зелёный	0	1	0
Синий	0	0	1
Жёлтый	1	1	0
Голубой	0	1	1
Пурпурный	1	0	1
Белый	1	1	1

Схема простейшего буфера в 3 бита на пиксель:



Если согласовать **bufferDepth** со значениями **colorDepth** (см. ниже), гладкость анимации в некоторых браузерах может улучшиться.

Установка **bufferDepth** на **-1** указывает IE буферизовать по глубине пикселей экрана (как установлено в панели управления), и **colorDepth** также автоматически устанавливается на это значение (если пользователь изменит разрешение, буфер соответственно откорректируется).

Установка на **0** (по умолчанию) ничего не изменяет.

Установка на любое из других разрешённых значений (**1**, **4**, **8**, **15**, **16**, **24** или **32**) буферизует в соответствии с ним и устанавливает **colorDepth** в то же значение.

4	16 цветов
8	256 цветов
15	32768 цветов
16	65536 цветов
24	16777216 цветов
32	4294967296 цветов

Чтобы воспользоваться в скриптах преимуществом более высоких параметров настройки, монитор клиента должен быть установлен на более высокие значения бит-на-пиксель.

colorDepth

Возвращает число битов на пиксель, использованных в мониторе или буфере изображения. Свойство — только для чтения, но его значение может быть изменено под влиянием настройки свойства **bufferDepth** (только IE).

Можно определить разрядность цвета текущего экрана и выбрать цвета в соответствии с ней.

Например:

```
if (screen.colorDepth> 8)
document.getElementById("heading1").color = "cornflowerblue";
else
```

```
document.getElementById("heading1").color = "blue";
```

Тем, кто впервые познакомился с работой цветовой схемы на этом уроке, может показаться непонятным, чем же **bufferDepth** отличается от **colorDepth**.

Поясню ещё одну деталь.

Свойство **colorDepth** считывает настройки монитора, его возможности. Однако при трансляции изображений компьютер может использовать только те возможности монитора, которые необходимы для показа данного изображения. Эта избирательная цветовая схема и записывается в буфер кадра, её-то считывает и может настроить свойство **bufferDepth**. Скажем, показывая картинку gif, оптимизированную на 8 цветов, из всего «супер-пупер-монитора» буфер будет использовать только ту простую схемку, которая показана на моём рисунке (если, конечно, не «извратить» настройки буфера, задав, скажем, для **bufferDepth** настройку **-1**).

height, width

Высота и ширина всей области монитора пользователя в пикселях. Собственно, **width** совпадает с **availWidth**, тогда как **height**, в отличие от **availHeight**, включает всю область экрана вместе с системной панелью.

С помощью этих свойств можно настраивать страницы для различных разрешений монитора:

```
if (screen.height <= 600 && screen.width <= 800)
{
/* Код версии для мониторов с разрешением 800×600 и ниже*/
}
```

pixelDepth

Работает только в браузерах семейства Netscape. Почти то же самое, что **colorDepth**, только его нельзя «извращать» ни с помощью **bufferDepth**, ни каким-либо другим способом (свойство только для чтения и «защищено от дурака»).

Из соображений кроссбраузерности лучше использовать **colorDepth**.

updateInterval

При показе анимации экран всё время обновляется. Свойство **updateInterval** настраивает интервал (в миллисекундах) между обновлениями экрана. Значение **0** позволяет браузеру автоматически выбирать среднее значение. Как правило, это самый оптимальный вариант.

Чем дольше интервал, тем больше шагов анимации, заложенных в буфер, может быть пропущено при показе.

Глава 20. Работа с DOM

Что такое DOM

Аббревиатура **DOM** расшифровывается как *DocumentObjectModel* (объектная модель документа).

DOM — это программный интерфейс доступа к содержимому HTML, XHTML и XML-документов, то есть представление тэгов и атрибутов HTML, XHTML и XML, а также стилей CSS в виде объектов программирования. С этой моделью работает как JavaScript, так и другие языки web-программирования.

Немного истории

Существует 4 уровня **DOM** (0, 1, 2 и 3).

Уровень 0 (1996) включал в себя модели **DOM**, которые существовали до появления уровня 1. В основном это коллекции: document.images, document.forms, document.layers и document.all. Эти модели формально не являются спецификациями **DOM**, опубликованными **W3C**. Скорее они представляют информацию о том, что существовало до начала процесса стандартизации.

Уровень 1 (1997) включал также базовые функциональные возможности по обработке XML-документов: многочисленные способы работы с отдельными узлами, работа с инструкциями обработки XML и т.д.

Кроме того, **DOM** уровня 1 содержит ряд специальных интерфейсов, которые могут обрабатывать отдельные HTML-элементы. Например, можно работать с таблицами HTML, формами, списками выбора и т.д.

В **DOM уровня 2** (2002) было добавлено несколько новых возможностей.

Если в **DOM** уровня 1 отсутствовала поддержка пространств имён, то интерфейсы **DOM** уровня 2 содержат методы для управления пространствами имён, связанными с требованиями к составлению и обработке XML-документов.

Помимо этого, **DOM** уровня 2 поддерживает события.

Уровень 2 является текущим уровнем спецификаций **DOM**, однако **W3C** рекомендует и некоторые разделы спецификаций уровня 3.

DOM уровня 3 — это рабочий проект спецификации, которая расширяет функциональность **DOM**уровня 2. Одна из наиболее важных особенностей этой версии спецификации заключается в возможности работать с многочисленными расширениями **DOM**.

Что означает «программный интерфейс»?

Английское слово **interface** можно перевести как «область контакта». Компьютер, грубо говоря, понимает только две вещи: пустой бит и заполненный бит. Язык, на котором «говорит» компьютер, можно представить как нескончаемую вереницу нулей и единиц, дающих бесконечное количество различных комбинаций.

Любой программный код — это вразумительная для программиста интерпретация этих «нулей и единиц», с которыми работает компьютер. Таким образом, любой язык программирования является интерфейсом человека и машины.

Браузеры работают так же, как и другие компьютерные приложения. Они интерпретируют в «нули и единицы» коды HTML, XML, CSS, скрипты JavaScript, PHP, Perl и т.д. Для работы с этим многоязычием нужна общая платформа. Этой платформой и является **DOM** — спецификация, не

зависящая от конкретного языка программирования или разметки. Это интерфейс, который можно использовать во многих популярных языках программирования, связанных с созданием web-страниц и способных понимать и интерпретировать объекты **DOM**.

DOM и браузеры

Основное содержание **DOM** — это объекты браузера. Все современные браузеры в той или иной мере поддерживают **DOM**, но в отдельных случаях встречаются разночтения. Такие случаи мы будем специально оговаривать.

DOM и JavaScript

В JavaScript вершиной иерархической лестницы объектов **DOM**, своеобразным «проводником» в этот интерфейс служит объект **document**, а объекты **DOM** становятся его свойствами, свойствами его свойств и т.д. Их также называют **узлами DOM**.

Узлы DOM

В **DOM** уровня 2 есть 12 типов узлов. За каждым типом узла **DOM** закреплена константа с уникальным именем. Большинство узлов предназначено для работы с **XML**. В сборке **HTML — JavaScript**, которой мы занимаемся, можно использовать только 5 типов. Но и эта «верхушка айсберга» представляет собой весьма «развесистое дерево», которое не охватить за одно-два занятия.

Полный набор констант типов узлов, определённый в спецификации **W3C DOM** (голубым подсвечены узлы, доступные для **HTML — JavaScript**):

Имя константы	**Значение**	**Описание**
Node.ELEMENT_NODE	1	Узел элемента (возвращает корневой элемент документа, для HTML-документов это элемент HTML)
Node.ATTRIBUTE_NODE	2	Узел атрибута (возвращает

		атрибут элемента XML- или HTML-документа)
Node.TEXT_NODE	3	Текстовый узел (#text)
Node.CDATA_SECTION_NODE	4	Узел секции CDATA (XML: альтернативный синтаксис для отображения символьных данных)
Node.ENTITY_REFERENCE_NODE	5	Узел ссылки на раздел
Node.ENTITY_NODE	6	Узел раздела
Node.PROCESSING_INSTRUCTION_NODE	7	Узел директивы XML
Node.COMMENT_NODE	8	Узел комментария
Node.DOCUMENT_NODE	9	Узел документа (основа доступа к содержанию документа и создания его составляющих)
Node.DOCUMENT_TYPE_NODE	10	Узел типа документа (возвращает тип данного документа, т.е. значение тэга DOCTYPE)
Node.DOCUMENT_FRAGMENT_NODE	11	Узел фрагмента документа (извлечение части дерева документа, создание нового фрагмента документа, вставка фрагмента в качестве дочернего элемента какого-либо узла и т.п.)
Node.NOTATION_NODE	12	Узел нотации*

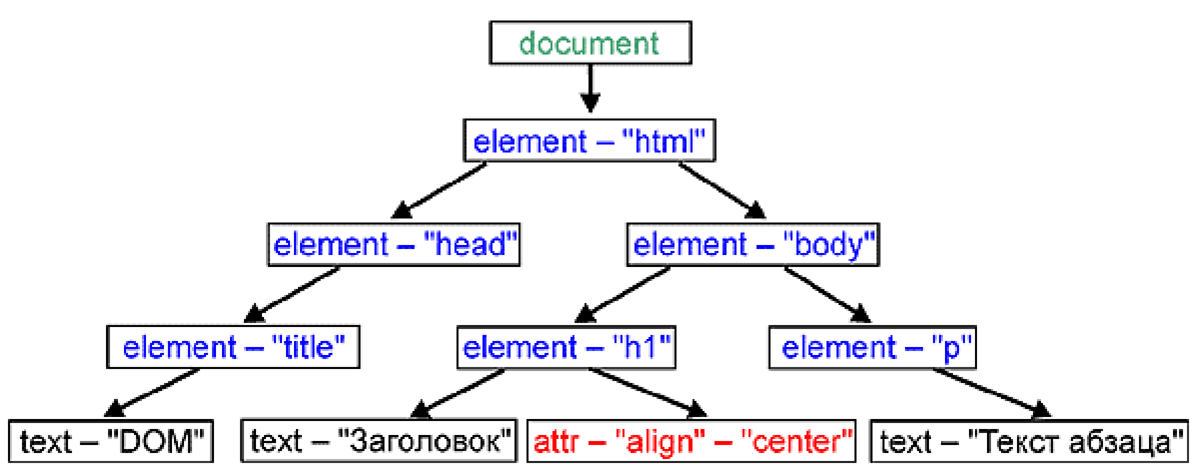
*** Нотации — это имена, идентифицирующие формат неанализируемых разделов, формат элементов, имеющих атрибут нотации, или прикладную программу, которой адресована директива.** (Непонятно? Мне пока тоже не очень.)

Структура документа в модели DOM

Все объекты документа являются узлами DOM. Рассмотримэлементарныйдокумент:

```
<html>
<head><title>DOM</title></head>
<body>
<h1 align="center">Заголовок</h1>
<p>Текстабзаца</p>
</body>
<html>
```

Вот схема его DOM-дерева:



Каждый узел может иметь дочерние узлы (на схеме к ним ведут стрелки). Объект document — основание дерева документа — тоже узел, но у него нет родительского узла и имеется ряд свойств и методов, отсутствующих у других узлов. Он имеет один дочерний узел: элемент <html>.

У элемента <html> два дочерних узла: <head> и <body>, для которых дочерними становятся все элементы, содержащиеся в них.

Внимание!

«Элемент» и «тэг» не синонимы. Тэг — это знак разметки: <p> и </p> — это два разных тэга. А элемент — объект, помеченный этими тэгами: <p>Текст абзаца</p>.

Элементы <title>, <h1> и <p> содержат внутри себя **текст**. Это их дочерние **текстовые узлы**. У элемента <h1> есть также **атрибут**: align="center". **Узлы атрибутов** — это тоже дочерние узлы элементов, которые их содержат.

При работе с узлами DOM-дерева используются их свойства и методы.

Некоторые свойства узлов

Маленькое вступление

Ещё раз повторю: когда мы обращаемся в скриптах к элементам страницы, то имеем дело не только с языком Javascript, но и с внедрённым в него интерфейсом **DOM**. Иногда необходимо отдавать себе в этом отчёт, иногда можно и забыть, «что говорим прозой».

Некоторыми свойствами и методами из объектной модели **DOM** мы уже таким образом пользовались.

В этом уроке мы не будем идти «академическим» путём, рассматривая все свойства всех узлов во всех браузерах. Для начала познакомимся с самыми практичными и «бесконфликтными» из них.

Именно поэтому **не будем** начинать, как принято, с «основных свойств»: **nodeName** и **nodeValue**.

tagName

Возвращает строку с именем тэга элемента. Все значения tagName содержат символы только верхнего регистра.

Синтаксис

элемент.**tagName**

Пример

```
<!-- Вставим <span>, чтобы имя было не из одной буквы. -->
```

```
<p><span id="testTagName">ТестируемсвойствоtagName</span></p>
<p>
<script type="text/javascript">
document.write(document.getElementById("testTagName").tagName)
</script>
</p>
```

Результат

Тестируем свойство tagName

SPAN

innerHTML

С этим свойством мы уже встречались (см. **Урок 10**). А теперь понимаем, откуда оно взялось: «из дома».

Даёт доступ к содержимому элемента. Задаёт не только текстовое содержание, но и все тэги HTML, находящиеся внутри элемента.

Это свойство не только для чтения, но и для изменения содержимого.

Примечание

В IE для ряда элементов **innerHTML** работает только для чтения: это все табличные элементы, за исключением <td> и <th>, а также <title> и <frameset>.

Например, мы создали пустую таблицу без элемента <td> и хотим программно вставить его в <tr>через **innerHTML**:

```
<html>
<head></head>
<body onload="document.getElementById('test').innerHTML =
'<td>тестоваястрока</td>'">
<table>
<tr id="test"></tr>
</table>
</body>
```

</html>

IE выдаст «неизвестную ошибку выполнения», а остальные браузеры произведут вставку.

В то же время, если мы запросим существующее содержимое элемента <tr>, например, через alert(document.getElementById('id').innerHTML), то в IE это сработает.

Синтаксис

элемент.**innerHTML** = *"назначенный текст"*

Пример

```
<!-- читаем текст отсюда
(исходный элемент делаем невидимым) -->
<p style="display: none;" id="testInnerHTML"><b style="color: red;">Тестируем свойство innerHTML</b></p>
<!-- вставляем его сюда -->
<p id="target">Абзац для вставки</p>
<scripttype="text/javascript">
// Эта функция читает текст и вставляет его в заданный абзац.
function testRead() {
document.getElementById("target").innerHTML = document.getElementById("testInnerHTML").innerHTML
}
// Эта функция изменяет текст заданного абзаца.
function testChange() {
document.getElementById("target").innerHTML = "<spanstyle='color: blue;'>Перекрашиваем и меняем текст</span>"
}
// Эта функция возвращает свойство в исходное положение.
function testReset() {
document.getElementById("target").innerHTML = "Абзац для
```

```
вставки"
        }
        </script>
        <form>
        <inputtype="button" value="прочитать innerHTML"
onClick="testRead();">
        <inputtype="button" value="изменить innerHTML"
onClick="testChange();">
        <inputtype="button" value="сброс" onClick="testReset();">
        </form>
```

Абзац для вставки

Примечание

Есть ещё свойство *innerText*, которое включает только текстовое содержимое, очищенное от вложенных тэгов. Но оно работает только в IE.

style

Об этом свойстве мы тоже уже говорили, и даже довольно подробно.

Оно позволяет управлять настройками стилей CSS для данного элемента. Значением свойства является объект **style**, чьи собственные свойства позволяют получать и изменять установки стилей.

Синтаксис

```
элемент.style.свойствоСтиля = "значение"
```

свойствоСтиля — любое свойство стиля CSS.

"значение" — любое существующее значение свойства стиля

Примечание

напомню, что дефисы в значении стилей заменяются в JavaScript на заглавные буквы:

text-align = **textAlign**

background-color = **backgroundColor**

и т.д.

Примеры можно посмотреть в **13 уроке**.

className

Это свойство тоже взаимодействует со стилями CSS, оно задаёт класс элемента из существующего набора таблицы стилей.

Синтаксис

элемент.**className** = *"имя класса"*

"имя класса" — любой класс, существуюший в тэге <style> или в прикреплённом файле.css.

«Паспорта» узлов

Узлы бывают разных типов. В сборке **HTML — JavaScript** используются 5 типов узлов:

Тип	**Значение константы**	**Возвращаемое значение**
Элемент	1	Тэг элемента
Атрибут	2	Имя атрибута
Текст	3	#text
Комментарий	8	#comment
Документ	9	#document

Каждый узел, образно говоря, имеет свой «паспорт», состоящий из трёх свойств: **nodeType**, **nodeName** и **nodeValue**.

Примечание 1

Эти свойства пока работают криво: в каждой избушке (браузере) свои погремушки. По мере «вгрызания» в материал будем знакомиться с глюками и строить обходные функции.

Примечание 2

Эти свойства формально поддерживаются и узлами атрибутов, но с атрибутами они работают не всегда корректно: у атрибутов есть свои особые «паспорта», данные которых не всегда совпадают с данными этих свойств.

nodeType

Возвращает значение константы узла.

nodeName

У элементов возвращает имя тэга прописными буквами, у других узлов — их имена, начинающиеся с «решётки».

nodeValue

У текстовых узлов и комментариев возвращает строку текста, содержащуюся в них, у других — **null**.

Сводная таблица возвращаемых значений

Узел	**nodeType**	**nodeName**	**nodeValue**
Элемент	1	ИМЯ_ТЭГА	null
Текст	3	#text	строка текста
Комментарий	8	#comment	строка текста
Документ	9	#document	null

Работа с узлами элементов DOM

При работе с HTML мы сталкивались с различными коллекциями: **document.images**, **document.links**, **document.all** и т.д. Эти коллекции не принадлежат интерфейсу **DOM 3** и являются своеобразным «атавизмом», оставшимся от **DOM 0**.

Эти атавистические коллекции были затем встроены в интерфейс JavaScript и ведут себя как объекты Array. А коллекции **DOM 3** — это свойства узлов DOM, они не яаляются объектами Array и ведут себя несколько иначе.

Смешение этих понятий может привести к созданию неработающих или «кривых» скриптов.

Преимущество использования **DOM 3** в том, что мы можем сами создавать любые коллекции — как соответствующие, так и не соответствующие встроенным.

Объект nodeList

Когда мы обращаемся к группе узлов — например, к элементам с определённым тэгом, или только к текстовым узлам, или к атрибутам данного элемента, — мы получаем их в виде списка **nodeList**. Он похож на массив JavaScript, но таковым не является.

Как и массив, он имеет свойство **length** и порядковые номера элементов (начиная с нуля), указывающиеся в квадратных скобках. Но другие свойства и методы массивов JavaScript с ним не работают.

Каждый элемент объекта **nodeList** является узлом — объектом, имеющим ряд свойств.

Можно представить себе **nodeList** в виде хэша. Например, в списке атрибутов можно усмотреть массив «ключей» (имена атрибутов: align, color и т.д.) и «значений» (значения атрибутов: center, black и т.д.)

В некоторых случаях для работы с содержимым объекта **nodeList** бывает удобно преобразовать это содержимое в обычные массивы. Например, мы хотим получить массив всех тэгов, находящихся в <body>.

Метод document.**getElementsByTagName**(*"BODY"*)[0] возвращает элемент с тэгом <body>(подробнее о методе см. ниже, как и о свойстве **childNodes**).

Свойство **childNodes** элемента <body> возвращает список всех его дочерних узлов.

Устанавливаем «фильтр», выбирающий из них только узлы элементов (nodeType == 1).

Строим массив, содержащий имена тэгов (nodeName).

```
var allBodyTags = newArray;
var elmBody = document.getElementsByTagName("BODY")[0];
for (var i = 0; i <elmBody.childNodes.length; i++)
{
   // если тип узла - элемент
   if (elmBody.childNodes[i].nodeType == 1)
   {
      // наращиваем возвращаемый массив
      // из имён элементов
      allBodyTags =
allBodyTags.concat(elmBody.childNodes[i].nodeName);
   }
}
```

Свойства доступа к элементам

Рассматриваемые ниже свойства относятся именно к **элементам DOM**. В HTML они интерпретируются с помощью тэгов.

(Ещё раз напомню, что это не синонимы. Если сравнить элемент с беговой дорожкой, то тэги — это старт и финиш. В элементах, не имеющих закрывающего тэга, эти понятия совпадают.)

Примечание

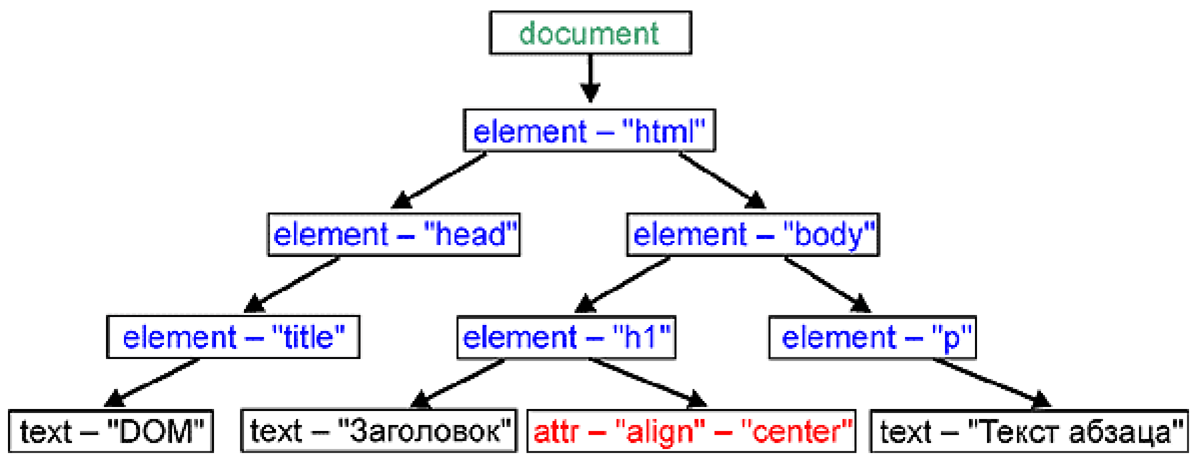
Частично эти свойства доступны и для текстовых узлов, но об этом поговорим в специальном разделе.

Многие свойства доступа к элементам являются массивами, хотя иногда они работают не совсем так, как массивы JavaScript. Нет чёткого разделения между элементами и текстовыми узлами, что создаёт проблемы в разных браузерах.

Каждый элемент содержит свойства, отражающие его место в иерархии DOM. Они позволяют перемещаться по всем элементам дерева. Вот некоторые свойства взаимоотношений элементов:

parentNode	родительский элемент
childNodes (массив)	дочерние элементы
firstChild	первый элемент в массиве childNodes
lastChild	последний элемент в массиве childNodes
prevSibling	предыдущий элемент в массиве childNodes
nextSibling	следующий элемент в массиве childNodes

Ещё раз приведу схему дерева простого документа из прошлого урока.



parentNode

Возвращает родительский элемент. На нашей схеме для узла **HTML** родительским элементом будет сам документ, для **HEAD** и **BODY** — узел **HTML** и т.д.

Текстовые узлы также поддерживают это свойство.

У узла документа, естественно, этого свойства нет.

Возможны вложения «матрёшечного» типа для доступа к «дедушке» или «прадедушке»:

элемент.**parentNode**.**parentNode**

childNodes

Список потомков узла (объект **Nodelist**). При работе с HTML эта функция полезна только для элементов. Текстовые узлы и узлы-атрибуты не имеют потомков.

Синтаксис

Node.**childNodes**[*N*]

Node — родительский узел.

N — номер потомка в списке.

Примечание

Браузеры семейства Netscape воспринимают пробелы в коде (пустые строки или переносы на другую строку) как текстовые узлы.

В нашем примере простейшего документа:

Узел document имеет одного потомка: элемент "html".

Элемент "html" по идее имеет двух потомков: элемент "head" и элемент "body". Но в браузерах семейства Netscape возникнут 4 потомка: перенос строки перед <head>, собственно <head>, перенос строки перед <body> и собственно <body>. Об этом нужно помнить, просматривая список **childNodes**в цикле **for**.

В природе как бы существует метод **normalize()**, которому надлежит устранить проблему, но элементы HTML на него не реагируют. Поэтому могу предложить лишь «кустарный» способ.

У этого способа есть один недостаток: он исключает из обращения все текстовые узлы, а не только «левые». Хотя это как посмотреть: возможно, он делает полезное дело, очищая зёрна от плевел и давая более чёткое разделение элементов и текстовых узлов.

Итак,

«…нормальные бандиты всегда идут в обход»

Смысл этой функции в превращении свойства **childNodes** в обычный массив JavaScript, очищенный от текстовых узлов.

```
functioncleanNodes(elm) {
// в качестве значения для аргумента elm выступает
// родительский элемент, который нам надо просмотреть.
var res = new Array;
```

```
for (vari = 0; i<elm.childNodes.length; i++)
{
// еслитипузла - элемент
if (elm.childNodes[i].nodeType == 1)
// наращиваем возвращаемый массив из имён элементов
res = res.concat(elm.childNodes[i].nodeName);
}
returnres;
}
```

Почему строим массив именно из имён элементов (**nodeName**)?

Потому что **nodeType** у них у всех единица, а **nodeValue**, как мы помним, — **null**. А свойство **length**принимает правильный, нормализованный вид.

***

Есть ещё один кустарный способ избавиться от этого глюка: просто записать весь код HTML на одной длинной строке.

firstChild, lastChild

Указатели на соответствующие элементы в списке **childNodes** (см. таблицу выше).

элемент.firstChild — то же, что *элемент*.childNodes[0].

элемент.lastChild — то же, что *элемент*.childNodes[*элемент*.childNodes.length-1].

Обращаются к родительскому элементу, имеющему «семью» **childNodes**.

prevSibling, nextSibling

Эти указатели обращаются уже к самим «членам семьи» **childNodes**.

```
элемент[n].prevSibling = элемент.parentNode.childNodes[n-1]
элемент[n].nextSibling = элемент.parentNode.childNodes[n+1]
```

Естественно, что у childNodes[0] отсутствует prevSibling,

а у childNodes[childNodes.length-1] бесполезно искать nextSibling.

Примечание

В браузерах семейства Netscape по вышеупомянутой причине узлов несколько больше, чем нужно. Поэтому использовать эти свойства не всегда удобно.

Методы доступа к элементам

В этом обзорном уроке мы рассмотрим лишь несколько основных методов доступа к элементам. Первая группа методов принадлежит корневому объекту **document**, некоторые из них мы уже использовали.

Методы второй группы могут применяться и к самим элементам.

Сейчас мы рассмотрим только те немногие методы, которые нормально работают во всех браузерах.

Метод объекта document

getElementById()

Этим методом мы уже не раз пользовались. Он ссылается на элемент с ID, указанным в аргументе (нужно не забыть задать ID при вёрстке страницы).

Синтаксис

document.**getElementById**(*"ID"*)

Метод для всех элементов

getElementsByTagName()

Возвращает список дочерних элементов по указанному в аргументе имени тэга.

Обратите внимание на правописание имени метода

Часть «Elements» пишется во множественном числе, с буковкой «s» на конце.

Синтаксис

элемент.**getElementsByTagName**(*"ИМЯ_ТЭГА"*)

элемент.**getElementsByTagName**(*"*"*)

элемент — любой элемент **DOM**.

Если в качестве параметра аргумента указана звёздочка, возвращается список всех дочерних элементов.

Этот метод можно применять как к самому документу, так и к любому элементу. Например, его можно применить к элементу <table> для перебора всех <tr>. Или для работы с фрагментом документа, заключённым в тэг <div>.

Для того, чтобы обратиться к конкретному элементу из списка, нужно указать его номер в квадратных скобках.

Например (обращение ко второму абзацу в контейнере с id-"content"):

```
document.geyElementById("content").getElementsByTagName("P")[1]
```

Поскольку метод всегда возвращает список, номер элемента (нулевой) необходимо указывать даже для тех элементов, которые существуют заведомо в единственном числе:

```
document.getElementsByTagName("BODY")[0]
```

Казалось бы, этот метод (с параметром в виде звёздочки) подходит для решения описанной выше проблемы с удалением лишних текстовых узлов. Но есть один нюанс: с этим параметром метод возвращает список не только дочерних элементов, но и всех «внуков» и «пра[прапра]внуков».

Некоторые методы создания элементов

Перечислены только методы, работающие с HTML во всех браузерах.

Метод объекта document

createElement()

Записывает в память образец объекта, связанного с тэгом, указанном в аргументе метода.

Синтаксис

```
document.createElement("имя_тэга")
```

Для того, чтобы созданный этим методом элемент отобразился на странице, нужно задать ему атрибуты и/или содержимое, а затем «привязать» к существующему «родителю». Всё это делается другими методами, которые бюудут рассмотрены ниже.

Методы, применимые к другим элементам

appendChild() — добавляет узел в конец списка дочерних узлов

cloneNode() — создает копию узла

hasChildNodes() — проверяет наличие дочерних узлов

insertBefore() — вставляет новый узел перед заданным дочерним узлом

removeChild() — удаляет заданный дочерний узел

replaceChild() — заменяет заданный дочерний узел новым узлом

appendChild()

Добавляет узел, определённый в аргументе, в конец списка **childNodes** данного узла и возвращает его в качестве результата. Если этот узел уже был в списке, то он сначала удаляется, а затем добавляется.

Синтаксис

```
элемент.appendChild("НОВ_ДОЧ_УЗ")
```

элемент — элемент, из которого вызван метод.

"НОВ_ДОЧ_УЗ" — имя добавляемого дочернего узла

Пример

```
varelem = document.createElement("IMG");
var _body = document.getElementsByTag("BODY")[0];
_body.appendChild(elem);
document.write(_body.lastChild.tagName);
// возвращает "IMG"
```

cloneNode()

Создаёт копию данного узла и возвращает ее.

Синтаксис

```
элемент.cloneNode(уровень)
```

элемент — элемент, из которого вызван метод.

уровень — булево выражение.

Если уровень — **true**, то создаётся копия поддерева документа, начиная с данного узла; если **false**, то копируется только сам узел (и его атрибуты, если это элемент).

hasChildNodes()

Возвращает булево значение: **true**, если элемент имеет хотя бы один дочерний узел; в противном случае — **false**.

Синтаксис

```
элемент.hasChildNodes()
```

элемент — элемент, из которого вызван метод.

insertBefore()

Вставляет новый узел в **nodeList** дочерних узлов данного элемента перед узлом, указанным как тоска входа, и возвращает ссылку на него.

Синтаксис

```
элемент.insertBefore(новый_узел, точка_входа)
```

элемент — элемент, из которого вызван метод.

новый_узел — узел для вставки.

точка_входа — существующий узел, перед которым будет вставлен новый.

Если узел, указанный как новый, уже был в списке, то он сначала удаляется, а затем добавляется.

Если *точка_входа* — **null** (второй аргумент опущен), то узел вставляется в конце списка дочерних узлов элемента.

Пример

```
<divid="test2">
<p>Первый абзац</p>
<pid="entryPoint">Второй абзац</p>
<p>Третий абзац</p>
</div>
<script type="text/javascript">
functioninserting() {
```

```
var parent = document.getElementById("test2");
var point = document.getElementById("entryPoint");
// создаём новый абзац
varelem = document.createElement("P");
// создаём текст для нового абзаца
vartxt = document.createTextNode("А это вставка");
// загоняем текст в абзац
elem.appendChild(txt);
// вставляем абзац в nodeList перед абзацем с id="entryPoint"
returnparent.insertBefore(elem, point);
}
</script>
<form>
<input type="button"value="Вставить"onClick="inserting()">
<input type="button"value="Сбросить"onClick="location.reload()">
</form>
```

Примечание

О методе createTextNode(), создающем новый текстовый узел, подробнее см. в одном из следующих уроков.

Результат

Первый абзац

Второй абзац

Третий абзац

removeChild()

Удаляет узел, указанный в аргументе, из списка дочерних узлов данного элемента.

Синтаксис

```
элемент.removeChild(удаляемый_узел)
```

элемент — элемент, из которого вызван метод.

удаляемый_узел — дочерний узел элемента, подлежащий удалению

Пример

Удаляем первый абзац из предыдущего примера.

```
<script type="text/javascript">
var _div = document.getElementById("test2");
</script>
<form>
<input type="button" value="Удалить"
onClick="_div.removeChild(_div.firstChild)">
<input type="button" value="Восстановить"
onClick="location.reload()">
</form>
```

Примечание

В браузерах семейства Netscape на первом щелчке на странице ничего не произойдёт (в коде удалится перевод строки перед первым тэгом <p>). Поэтому лучше не использовать свойства типа firstChild.

Глава21. Основы ООП на JavaScript.

Объектно-ориентированное программирование (ООП) — это парадигма программирования, которая использует абстракции, чтобы создавать модели, основанные на объектах реального мира. ООП использует несколько техник из ранее признанных парадигм, включая модульность, полиморфизм и инкапсуляция. На сегодняшний день многие популярные языки программирования (такие как Java, JavaScript, C#, C++, Python, PHP, Ruby и Objective-C) поддерживают ООП.

ООП представляет программное обеспечение как совокупность взаимодействующих объектов, а не набор функций или просто список команд (как в традиционном представлении). В ООП, каждый объект может получать сообщения, обрабатывать данные, и отправлять сообщения другим объектам. Каждый объект может быть представлен как маленькая независимая машина с отдельной ролью или ответственностью.

ООП способствует большей гибкости и поддерживаемости в программировании, и широко распространена в крупномасштабном программном инжиниринге. Так как ООП настоятельно подчеркивает модульность, объектно-ориентированный код проще в разработке и проще для понимания впоследствии. Объектно-ориентированный код способствует более точному анализу, кодированию и пониманию сложных ситуаций и процедур, чем методы программирования с меньшей модульностью.[1]

Терминология

Пространство имён

Контейнер, который позволяет разработчикам связать весь функционал под уникальным, специфичным для приложения именем.

Класс

Определяет характеристики объекта. Класс является описанием шаблона свойств и методов объекта.

Объект

Экземпляр класса.

Свойство

Характеристика объекта, например, цвет.

Метод

Возможности объекта, такие как ходьба. Это подпрограммы или функции, связанные с классом.

Конструктор

Метод, вызываемый в момент создания экземпляра объекта. Он, как правило, имеет то же имя, что и класс, содержащий его.

Наследование

Класс может наследовать характеристики от другого класса.

Инкапсуляция

Способ комплектации данных и методов, которые используют данные.

Абстракция

Совокупность комплексных наследований, методов и свойств объекта должны адекватно отражать модель реальности.

Полиморфизм

Поли означает "*много*", а морфизм "*формы*". Различные классы могут объявить один и тот же метод или свойство.

Для более обширного описания объектно-ориентированного программирования, см Объектно-ориентированное_программирование в Wikipedia.

Прототипное программирование

Прототипное программирование — это модель ООП которая не использует классы, а вместо этого сначала выполняет поведение класса и затем использует его повторно (эквивалент наследования в языках на базе классов), декорируя (или расширяя) существующие объекты *прототипы*. (Также называемое бесклассовое, прототипно-ориентированное, или экземплярно-ориентированное программирование.)

Оригинальный (и наиболее каноничный) пример прототипно-ориентированного языка это Selfразработанный Дэвидом Ангаром и Ренделлом Смитом. Однако бесклассовый стиль программирования стал набирать популярность позднее, и был принят для таких языков программирования, как JavaScript, Cecil, NewtonScript, Io, MOO, REBOL, Kevo, Squeak (при использовании фреймворка Viewer для манипуляции компонентами Morphic) и некоторых других.[1]

Объектно-ориентированное программирование в JavaScript

Пространство имён

Пространство имён — это контейнер, который позволяет разработчикам собрать функциональность под уникальным именем приложения. **Пространство имён в JavaScript — это объект, содержащий методы, свойства и другие объекты.**

Важно отметить, что на уровне языка в JavaScript нет разницы между пространством имён и любым другим объектом. Это отличает JS от множества других объектно-ориентированных языков и может стать причиной путаницы у начинающих JS программистов.

Принцип работы пространства имён в JS прост: создать один глобальный объект и все переменные, методы и функции объявлять как свойства этого объекта. Также использование пространств имён снижает вероятность возникновения конфликтов имён в приложении так как каждый объект приложения является свойством глобального объекта.

Давайте создадим глобальный объект MYAPP:

```
// Глобальное пространство имён
var MYAPP = MYAPP || {};
```

Во фрагменте кода выше мы сначала проверяем определён ли объект MYAPP (в текущем файле или другом файле). Если да, то используем существующий глобальный объект MYAPP, иначе создаём пустой объект MYAPP, в котором мы инкапсулируем все методы, функции, переменные и объекты.

Также мы можем создать подпространство имён (учтите, что сначала нужно объявить глобальный объект):

```
// Подпространство имён
MYAPP.event = {};
```

Далее следует пример синтаксиса создания пространства имён и добавления переменных, функций и методов:

```
// Создаём контейнер MYAPP.commonMethod для общих методов и свойств
MYAPP.commonMethod = {
```

```
regExForName: "", // определяет регулярное выражение для валидации имени
regExForPhone: "", // определяет регулярное выражение для валидации телефона
validateName: function(name){
   // Сделать что-то с name, вы можете получить доступ к переменной regExForName
// используя "this.regExForName"
  },
validatePhoneNo: function(phoneNo){
// Сделать что-то с номером телефона
  }
}
// Объект вместе с объявлением методов
MYAPP.event = {
addListener: function(el, type, fn) {
   // код
   },
removeListener: function(el, type, fn) {
   // код
   },
getEvent: function(e) {
   // код
   }
// Можно добавить другие свойства и методы
}
// СинтаксисиспользованияметодаaddListener:
MYAPP.event.addListener("yourel", "type", callback);
```

Стандартные встроенные объекты

В JavaScript есть несколько объектов, встроенных в ядро, например Math, Object, Array и String. Пример ниже показывает как использовать объект Math, чтобы получить случайное число, используя его метод random().

```
console.log(Math.random());
```

Примечание: В данном примере и далее мы будем использовать глобальную функцию console.log(). Если точнее, то функция console.log() не является частью JavaScript, но она поддерживается многими браузерами для облегчения отладки.

Смотрите JavaScriptReference: Standardbuilt-inobjects, чтобы ознакомиться со списком всех встроенных объектов JavaScript.

Каждый объект в JavaScript является экземпляром объекта Object, следовательно наследует все его свойства и методы.

Объекты, создаваемые пользователем

Класс JavaScript — это прототипно-ориентированный язык, и в нём нет оператора class, который имеет место в C++ или Java. Иногда это сбивает с толку программистов, привыкших к языкам с оператором class. Вместо этого JavaScript использует функции как конструкторы классов. Объявить класс так же просто как объявить функцию. В примере ниже мы объявляем новый класс Person с пустым конструктором:

```
var Person = function () {};
```

Объект (экземпляр класса)

Для создания нового экзмепляра объекта obj мы используем оператор newobj, присваивая результат (который имеет тип obj) в переменную.

В примере выше мы определили класс Person. В примере ниже мы создаём два его экземпляра (person1 и person2).

```
var person1 = new Person();
var person2 = new Person();
```

Ознакомьтесь с Object.create(), новым, дополнительным методом инстанцирования, который создаёт неинициализированный экземпляр.

Конструктор

Конструктор вызывается в момент создания экземпляра класса (в тот самый момент, когда создается объект). Конструктор является методом класса. В JavaScript функция служит конструктором объекта, поэтому нет необходимости явно определять метод конструктор. Любое действие определенное в классе будет выполненно в момент создания экземпляра класса.

Конструктор используется для задания свойств объекта или для вызова методов, которые подготовят объект к использованию. Добавление методов и их описаний производится с использованием другого синтаксиса, описанного далее в этой статье.

В примере ниже, конструктор класса Person выводит в консоль сообщение в момент создания нового экземпляра Person.

```
var Person = function () {
console.log('instance created');
};
var person1 = new Person();
var person2 = new Person();
```

Свойство (аттрибут объекта)

Свойства — это переменные, содержащиеся в классе; каждый экземпляр объекта имеет эти свойства. Свойства устанавливаются в конструкторе (функции) класса, таким образом они создаются для каждого экземпляра.

Ключевое слово this, которое ссылается на текущий объект, позволяет вам работать со свойствами класса. Доступ (чтение и запись) к свойствам снаружи класса осуществляется синтаксисом InstanceName.Property, так же как в C++, Java и некоторых других языках. (Внутри класса для получения и изменения значений свойств используется синтаксис this.Property)

В примере ниже, мы определяем свойство firstName для класса Person при создании экземпляра:

```
var Person = function (firstName) {
this.firstName = firstName;
console.log('Person instantiated');
};
var person1 = new Person('Alice');
var person2 = new Person('Bob');
// ВыводитсвойствоfirstNameвконсоль
console.log('person1 is ' + person1.firstName); // выведет "person1 is Alice"
console.log('person2 is ' + person2.firstName); // выведет "person2 is Bob"
```

Методы

Методы — это функции (и определяются как функции), но с другой стороны следуют той же логике, что и свойства. Вызов метода похож на доступ к свойству, но вы добавляете () на конце имени метода, возможно, с аргументами. Чтобы объявить метод, присвойте функцию в именованное свойство свойства prototype класса. Потом вы сможете вызвать метод объекта под тем именем, которое вы присвоили функции.

В примере ниже мы определяем и используем метод sayHello() для класса Person.

```
var Person = function (firstName) {
this.firstName = firstName;
};
Person.prototype.sayHello = function() {
console.log("Hello, I'm " + this.firstName);
};
var person1 = new Person("Alice");
var person2 = new Person("Bob");
// вызываемметодsayHello() класса Person
person1.sayHello(); // выведет "Hello, I'm Alice"
person2.sayHello(); // выведет "Hello, I'm Bob"
```

В JavaScript методы это — обычные объекты функций, связанные с объектом как свойства: это означает, что вы можете вызывать методы "вне контекста". Рассмотримследующийпример:

```
var Person = function (firstName) {
this.firstName = firstName;
};
Person.prototype.sayHello = function() {
console.log("Hello, I'm " + this.firstName);
};
var person1 = new Person("Alice");
var person2 = new Person("Bob");
varhelloFunction = person1.sayHello;
// выведет "Hello, I'm Alice"
person1.sayHello();
// выведет "Hello, I'm Bob"
person2.sayHello();
// выведет "Hello, I'm undefined" (or fails
// with a TypeError in strict mode)
helloFunction();
// выведет true
console.log(helloFunction === person1.sayHello);
// выведет true
console.log(helloFunction === Person.prototype.sayHello);
// выведет "Hello, I'm Alice"
helloFunction.call(person1);
```

Как показывает пример, все ссылки, которые мы имеем на функцию sayHello — person1, Person.prototype, переменная helloFunction и т.д. — ссылаются на одну и ту же функцию. Значение this в момент вызова функции зависит от того, как мы её вызываем. Наиболее часто мы обращаемся к this в выражениях, где мы получаем функцию из свойства

объекта — person1.sayHello() — this устанавливается на объект, из которого мы получили функцию (person1), вот почему person1.sayHello() использует имя "Alice", а person2.sayHello() использует имя "Bob". Но если вызов будет совершён иначе, то this будет иным: вызов this из переменной — helloFunction() — установит this на глобальный объект (window в браузерах). Так как этот объект (вероятно) не имеет свойства firstName, функция выведет "Hello, I'mundefined" (так произойдёт в нестрогом режиме; в strictmode всё будет иначе (ошибка), не будем сейчас вдаваться в подробности, чтобы избежать путаницы). Или мы можем указать this явно с помощью Function#call (или Function#apply) как показано в конце примера.

Примечание: Смотрите подробнее о this в Function#call и Function#apply

Наследование

Наследование — это способ создать класс как специализированную версию одного или нескольких классов (JavaScript поддерживает только одиночное наследование). Специализированный класс, как правило, называют потомком, а другой класс родителем. В JavaScript наследование осуществляется присвоением экземпляра класса родителя классу потомку. В современных браузерах вы можете реализовать наследование с помощью Object.create.

Примечание: JavaScript не обнаружит prototype.constructor класса потомка (смотрите Object.prototype) так что мы должны указать его вручную. Смотритевопрос "Why is it necessary to set the prototype constructor?" на Stackoverflow.

В примере ниже мы определяем класс Student как потомка класса Person. Потом мы переопределяем метод sayHello() и добавляем метод addGoodBye().

```
// Определяем конструктор Person
var Person = function(firstName) {
this.firstName = firstName;
```

```
};
// Добавляем пару методов в Person.prototype
Person.prototype.walk = function(){
console.log("I am walking!");
};
Person.prototype.sayHello = function(){
console.log("Hello, I'm " + this.firstName);
};
// Определяемконструктор Student
function Student(firstName, subject) {
// Вызываем конструктор родителя, убедившись (используя Function#call)
  // что "this" в момент вызова установлен корректно
Person.call(this, firstName);
  // Инициируемсвойствакласса Student
this.subject = subject;
};
// СоздаёмобъектStudent.prototype, которыйнаследуетсяотPerson.prototype.
// Примечание: Рспространённая ошибка здесь, это использование "newPerson()", чтобы создать
// Student.prototype. Это неверно по нескольким причинам, не в последнюю очередь
// потому, что нам нечего передать в Person в качестве аргумента "firstName"
// Правильное место для вызова Person показано выше, где мы вызываем
// его в конструкторе Student.
Student.prototype = Object.create(Person.prototype); // Смотрите примечание выше
```

```
// Устанавливаем свойство "constructor" для ссылки на класс Student
Student.prototype.constructor = Student;
// Заменяемметод "sayHello"
Student.prototype.sayHello = function(){
console.log("Hello, I'm " + this.firstName + ". I'm studying "
        + this.subject + ".");
};
// Добавляемметод "sayGoodBye"
Student.prototype.sayGoodBye = function(){
console.log("Goodbye!");
};
// Примериспользования:
var student1 = new Student("Janet", "Applied Physics");
student1.sayHello();   // "Hello, I'm Janet. I'm studying Applied Physics."
student1.walk();       // "I am walking!"
student1.sayGoodBye(); // "Goodbye!"
// Проверяем, чтоinstanceofработаеткорректно
console.log(student1 instanceof Person);  // true
console.log(student1 instanceof Student); // true
```

Относительно строки Student.prototype = Object.create(Person.prototype);: В старых движках JavaScript, в которых нет Object.create можно использовать полифилл (ещё известный как "shim") или функцию которая достигает тех же результатов, такую как:

```
functioncreateObject(proto) {
functionctor() { }
ctor.prototype = proto;
return new ctor();
}
// Примериспользования:
Student.prototype = createObject(Person.prototype);
```

Примечание: Смотрите Object.create для более подробной информации, и shim для реализации на старых движках.

Инкапсуляция

В примере выше классу Student нет необходимости знать о реализации метода walk() класса Person, но он может его использовать; Класс Student не должен явно определять этот метод, пока мы не хотим его изменить. Это называется **инкапсуляция**, благодаря чему каждый класс собирает данные и методы в одном блоке.

Сокрытие информации распространённая особенность, часто реализуемая в других языках программирования как приватные и защищённые методы/свойства. Однако в JavaScript можно лишь имитировать нечто подобное, это не является необходимым требованием объектно-ориентированного программирования.[2]

Абстракция

Абстракция это механизм который позволяет смоделировать текущий фрагмент рабочей проблемы, с помощью наследования (специализации) или композиции. JavaScript достигает специализации наследованием, а композиции возможностью экземплярам класса быть значениями атрибутов других объектов.

В JavaScript класс Fucntion наследуется от класса Object (это демонстрирует специализацию), а свойство Function.prototype это экземпляр класса Object (это демонстрирует композицию).

```
var foo = function () {};
// выведет "foo is a Function: true"
console.log('foo is a Function: ' + (foo instanceof Function));
// выведет "foo.prototype is an Object: true"
console.log('foo.prototype is an Object: ' + (foo.prototypeinstanceof Object));
```

Полиморфизм

Так как все методы и свойства определяются внутри свойства prototype, различные классы могут определять методы с одинаковыми именами; методы находятся в области видимости класса в котором они определены, пока два класса не имеют связи родитель-потомок (например, один наследуется от другого в цепочке наследований).

Примечания

Это не все способы которыми можно реализовать объектно-ориентированное программирование в JavaScript, который очень гибок в этом отношении. Также способы рассмотренные здесь не отражают всех возможностей JavaScript и не подражают реализации теории объектов в других языках.

Глава22. Разработка мультимедийных веб-приложений

Вместе с новыми элементами audio и video в HTML5 был добавлен новый API в JavaScript для управления этими элементами. С помощью кода JavaScript мы можем получить элементы video и audio (как и любой другой элемент) и использовать их свойства. В JavaScript эти элементы представлены объектом HTMLMediaElement, который с помощью свойств, методов и событий позволяет управлять воспроизведением аудио и видео. Отметим наиболее важные свойства, которые могут нам пригодиться для настройки этих элементов:

- playbackRate: устанавливает скорость воспроизведения. По умолчанию равно 1
- src: возвращает название воспроизводимого ресурса, если он установлен в коде html элемента
- duration: возвращает длительность файла в секундах
- buffered: возвращает длительность той части файла, которая уже буферизирована и готова к воспроизведению
- controls: устанавливает или возвращает наличие атрибута controls. Если он установлен, возвращается true, иначе возвращает false

- loop: устанавливает или возвращает наличие атрибута loop. Если он установлен, возвращается true, иначе возвращает false
- muted: устанавливает или возвращает наличие атрибута muted
- preload: устанавливает или возвращает наличие атрибута preload
- volume: устанавливает или возвращает уровень звука от 0.0 до 1.0
- currentTime: возвращает текущее время воспроизведения

Отдельно для элемента video мы можем использовать ряд дополнительных свойств:

- poster: устанавливает или возвращает атрибут poster
- height: устанавливает или возвращает атрибут height
- width: устанавливает или возвращает атрибут width
- videoWidth, videoHeight: для элемента video возвращают ширину и высоту видео

Следует также отметить два метода, с помощью которых мы можем управлять воспроизведением:

- play(): начинает воспроизведение
- pause(): приостанавливает воспроизведение
- Основные события элементов video и audio:
- canplaythrough: это событие срабатывает после загрузки страницы, если браузер определит, что он может воспроизводить это видео/аудио
- pause: событие срабатывает, когда воспроизведение мультимедиа приостанавливается, и оно переводится в состояние "paused"
- play: событие срабатывает, когда начинается воспроизведение файла
- volumechange: срабатывает при изменении уровня звука мультимедиа
- ended: срабатывает при окончании воспроизведения
- timeupdate: срабатывает при изменении времени воспроизведения

- error: генерируется при возникновении ошибки
- loadeddata: срабатывает, когда будет загружен первый фрейм видеофайла
- loadedmetadata: срабатывает после загрузки метаданных мультимедиа (длительность воспроизведения, размеры видео и т.д.)
- seeking: срабатывает, когда пользователь начинает перемещать курсор по шкале воспроизведения для перемещения к новому месту аудио- или видеофайла
- seeked: срабатывает, когда пользователь завершил перемещение к новому месту на шкале воспроизведения

Теперь используем некоторые из этих свойств, событий и методов для управления элементом video:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Аудио в HTML5</title>
<style>
        .hidden{
display:none;
        }
        #playBtn {
border: solid 1px #333;
padding: 5px;
cursor: pointer;
        }
</style>
</head>
<body>
```

```
<video width="400" height="300">
<source src="cats.mp4" type="video/mp4">
<source src="cats.webm" type="video/webm">
<source src="cats.ogv" type="video/ogg">
</video>
<div id="controls" class="hidden">
<a id="playBtn">Play</a>
<span id="timer">00:00</span>
<input type="range" step="0.1" min="0" max="1" value="0" id="volume"
/>
</div>
<script>
    // получаемвсеэлементы
varvideoEl = document.getElementsByTagName('video')[0],
playBtn = document.getElementById('playBtn'),
vidControls = document.getElementById('controls'),
volumeControl = document.getElementById('volume'),
timePicker = document.getElementById('timer');
// если браузер может воспроизводить видео удаляем класс
videoEl.addEventListener('canplaythrough', function () {
vidControls.classList.remove('hidden');
videoEl.volume = volumeControl.value;
    }, false);
    // запускамилиостанавливаемвоспроизведение
playBtn.addEventListener('click', function () {
if (videoEl.paused) {
videoEl.play();
        } else {
videoEl.pause();
        }
```

```
        }, false);
videoEl.addEventListener('play', function () {
playBtn.innerText = "Pause";
        }, false);
videoEl.addEventListener('pause', function () {
playBtn.innerText = "Play";
        }, false);
volumeControl.addEventListener('input', function () {
videoEl.volume = volumeControl.value;
        }, false);
videoEl.addEventListener('ended', function () {
videoEl.currentTime = 0;
        }, false);
videoEl.addEventListener('timeupdate', function () {
timePicker.innerHTML = secondsToTime(videoEl.currentTime);
        }, false);
        // рассчетотображаемоговремени
functionsecondsToTime(time){
var h = Math.floor(time / (60 * 60)),
dm = time % (60 * 60),
                m = Math.floor(dm / 60),
ds = dm % 60,
                s = Math.ceil(ds);
if (s === 60) {
                s = 0;
                m = m + 1;
            }
if (s < 10) {
                s = '0' + s;
            }
```

```
if (m === 60) {
        m = 0;
        h = h + 1;
    }
if (m < 10) {
        m = '0' + m;
    }
if (h === 0) {
fulltime = m + ':' + s;
    } else {
fulltime = h + ':' + m + ':' + s;
    }
return fulltime;
  }
</script>
</body>
</html>
```

Вначалевкоде JavaScript мыполучаемвсеэлементы. Затем, если браузер поддерживает видео и может его воспроизвести, то обрабатываем событие canplaythrough, устанавливая уровень звука и удаляя класс hidden:

```
videoEl.addEventListener('canplaythrough', function () {
    vidControls.classList.remove('hidden');
    videoEl.volume = volumeControl.value;
}, false);
```

Чтобы запустить воспроизведение, нам надо обработать нажатие ссылки Play:

```
playBtn.addEventListener('click', function () {
    if (videoEl.paused) {  // если видео остановлено, запускаем
        videoEl.play();
    } else {
```

```
            videoEl.pause();
        }
    }, false);
```

Обрабатывая события запуска и остановки воспроизведения, мы можем изменять надпись на ссылке:

```
videoEl.addEventListener('play', function () {

    playBtn.innerText = "Pause";
}, false);

videoEl.addEventListener('pause', function () {

    playBtn.innerText = "Play";
}, false);
```

Обрабатывая событие input, которое возникает при изменении значения ползунка, мы можем синхронизировать изменение ползунка и громкость видео:

```
volumeControl.addEventListener('input', function () {

    videoEl.volume = volumeControl.value;
}, false);
```

Обработка события ended позволит сбросить время воспроизведения:

```
videoEl.addEventListener('ended', function () {

    videoEl.currentTime = 0;
}, false);
```

А обработчик события timeupdate позволит динамически изменять показатель времени воспроизведения:

```
videoEl.addEventListener('timeupdate', function () {
    timePicker.innerHTML = secondsToTime(videoEl.currentTime);
```

}, false);

Для форматирования строки времени применяется вспомогательная функция secondsToTime.

Глава23. Разработка игр, калькуляторов и анимационных приложений

HTML5 вместе с CSS3 и JavaScript дают разработчику широкие возможности создания игр с использованием 3D, анимации, Canvas, математики, цветов, звука, WebGL. Одно из наиболее очевидных преимуществ HTML5 заключается в его независимости и от платформы, и в общем случае от аппаратной начинки.

При детальном рассмотрении можно выявить предоставляемые движками дополнительные возможности: упрощение некоторых часто встречающихся задач или подгрузка ресурсов, оформленный ввод, физика, звук, bitmap'ы (таких, конечно же, немного). Есть и довольно слабо оформленные движки, а есть и те, которые предоставляют в пользование разработчику редактор 2D уровней и инструменты отладки.

Предполагается, что большинство движков служат для сокращения временных затрат на разработку полноценной игры. Однако многие разработчики предпочитают создавать свой проект полностью с нуля, чтобы лучше представлять его устройство. Существует немного JavaScript-HTML5 движков, которые действительно чего-то стоят, однако и у них может быть один большой недостаток: они более не поддерживаются или близки к прекращению поддержки. Поэтому, выбирая движок, остановите свой выбор на тех продуктах, поддержка которых будет длиться достаточно продолжительное время.

Crafty

Легкий модульный игровой движок, включающий множество функций: анимацию, управление событиями, перерисовку регионов, отслеживание пересечений и столкновений, спрайтовую графику и многое другое. Поддерживает все браузеры, в т.ч. IE9. Никаких дополнительных усилий прилагать не требуется.

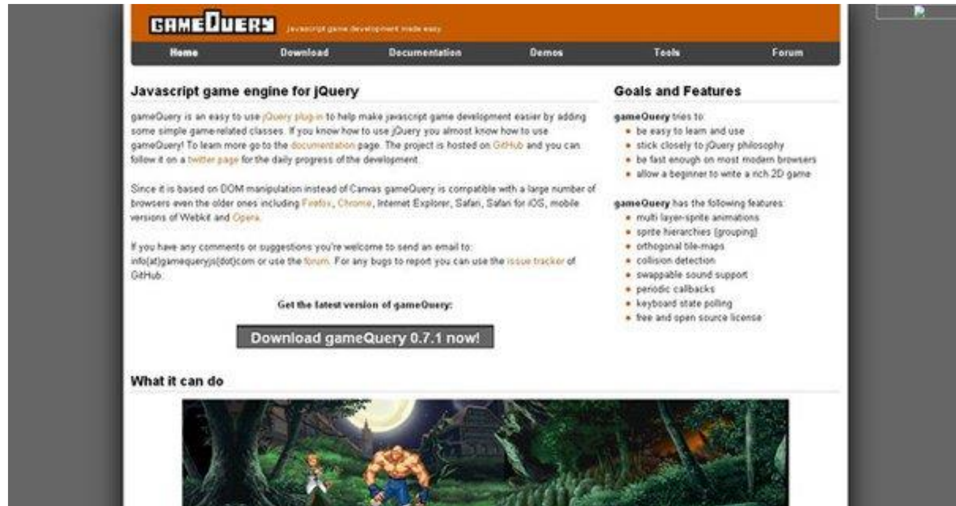
Quintus

Quintus – игровой HTML5-движок, разработанный, чтобы быть модульным и легковесным, с четким JavaScript-подобным интерфейсом. Для

того, чтобы реализовать основные особенности ООП-игрового движка в HTML5-движке, в Quintus в некотором отношении схож с jQuery, а также поддерживает плагины, управление событиями и гибкую модель наследования, чтобы упростить повторное использование реализованных функций.

---

gameQuery



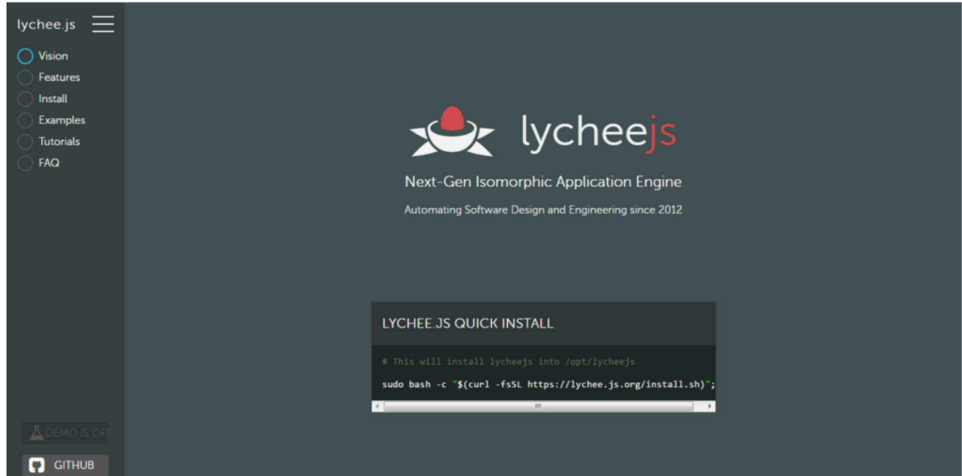
Простой в использовании плагин jQuery, упрощающий разработку игры за счет использования реализованных игровых компонентов. Благодаря особенностям реализации совместим со множеством браузеров, в т.ч. их мобильными версиями.

GMP

Идеально подойдет для реализации 2D спрайтовых аркад в ретро-стиле и головоломок вроде Судоку. Он имеет готовый к использованию самозапускающийся игровой цикл. Поддерживаются мышь и клавиатура. Отлично документирован, и главным недостатком можно считать только отсутствие поддержки звуков.

---

lycheeJS

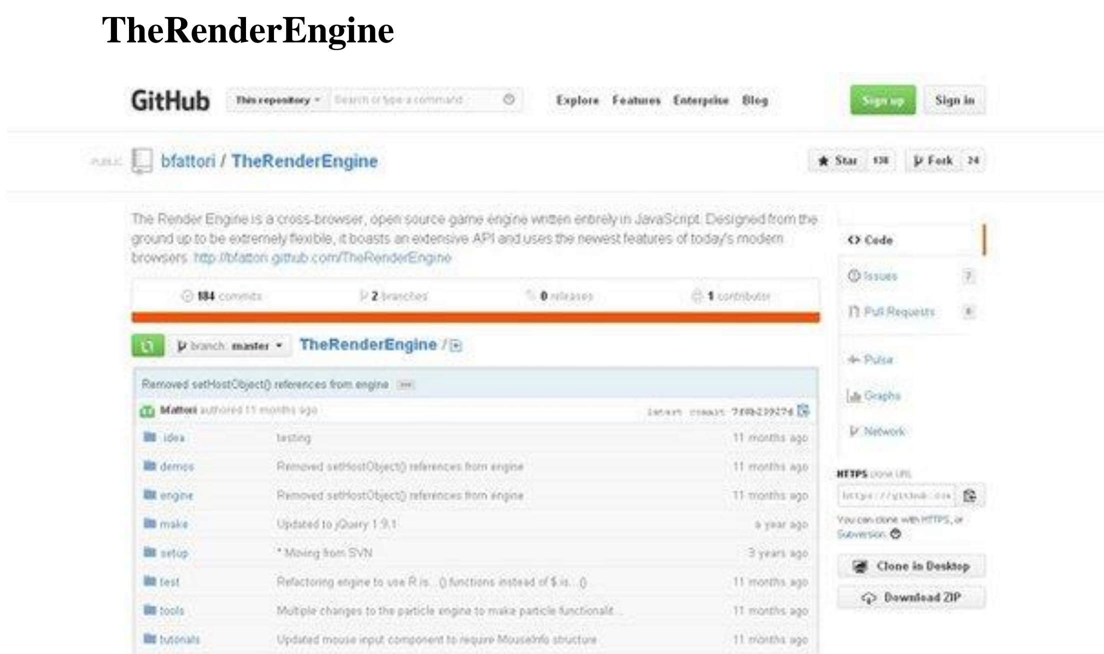


Игровая библиотека JavaScript, которая предлагает готовое решение для проектирования и реализации HTML5 Canvas и WebGL или нативныхOpenGL игр внутри браузера или стационарных платформ. Оптимизирован для GoogleChrome.

Enchant.js

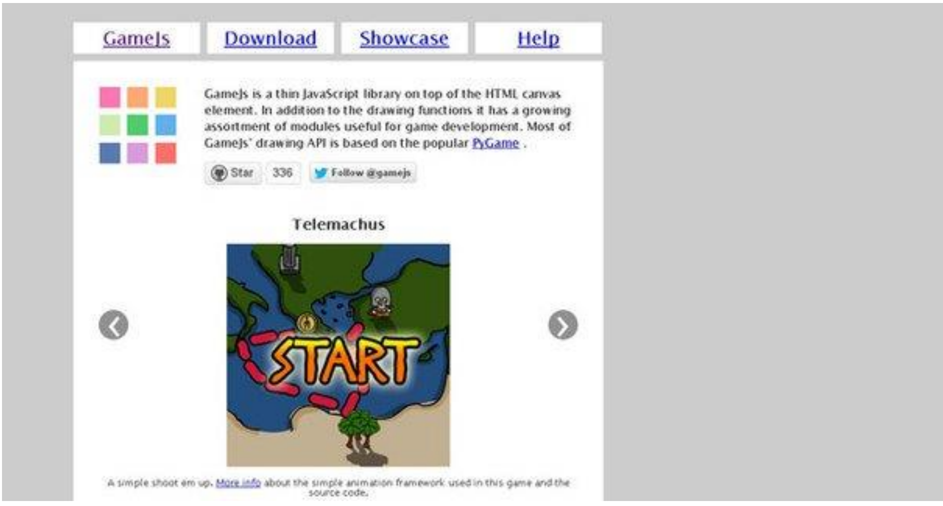
Фреймворк Enchant.js для HTML5+JavaScript игр был разработан в 2011 году, распространяется с открытым исходным кодом (MIT лицензия) и потому бесплатен.

TheRenderEngine



Кросс-браузерныйопенсорсный движок, написанный полностью на JavaScript. Созданный с нуля для того, чтобы быть максимально гибким, он имеет обширный API и использует самые новые фичи современных браузеров. Этот фреймворк предназначен, чтобы делать все за вас: ваша идея – его реализация с помощью самых часто используемых инструментов.

GameJS

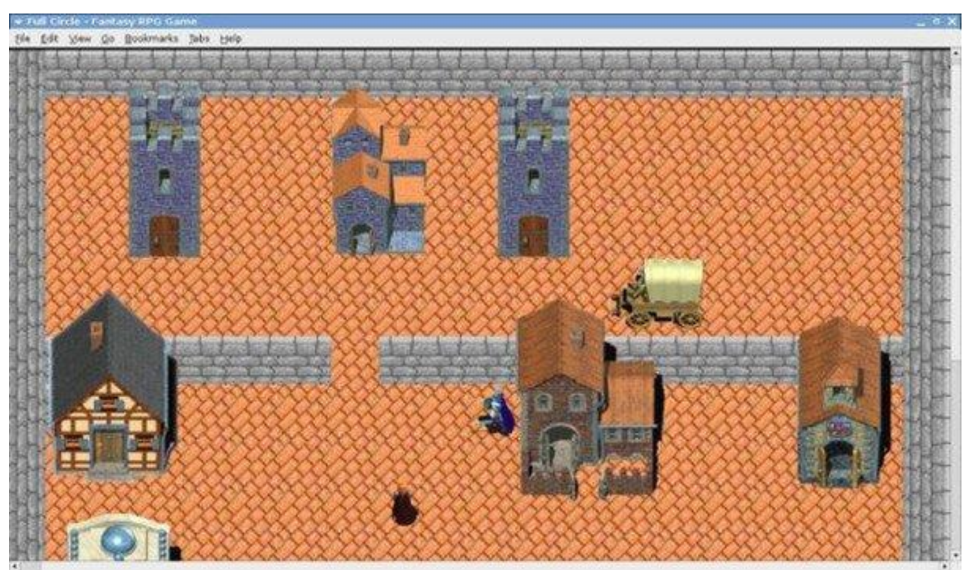


Большая библиотека на верхнем уровне HTML Canvas. В добавок к функциям рисования в ней имеется растущий ассортимент полезных для разработки игр модулей. Большинство имеющегося API основан на популярной PyGame.

CSS GameEngine

Для формирования страницы используются JavaScript и CSS. Вместе они работают достаточно уверенно и слаженно. Разработан для новичков, обучающихся азам программирования видеоигр. Вам будет проще, если у вас уже есть какие-то навыки работы с CSS.

ClanFX



clanfx основан на JavaScript и CSS и использует плиточную графику. Работает на данный момент в Firefox, Epiphany и Opera. Среди реализованных фич: анимированные спрайты, эффекты заклинаний, постройки, плитки/текстуры и базовый искусственный интеллект.

---

gTile

Браузерный движок на чистом JavaScript и DHTML. В gTile плиточная графика была выбрана за ее простоту и доступность. Упор в реализации был сделан на высокий уровень интерактивности и поведении игровых объектов. Меньшее внимание было уделено графике. А потому движок подойдет больше для создания текстовых РПГ, а графических возможностей должно хватить для изображения локаций.

J5g3

Графический JS движок с открытым исходным кодом (GPLv3). Легкий в использовании синтаксис предназначен для того, чтобы сделать фреймворк быстрым и расширяемым.

Jaws

2D игровая библиотека, основанная на HTML5. Использует и Canvas, и средства DOM.

Cocos2D

Портированный с iPhone графический 2D HTML5-движок на JavaScript. Позволяет быстро создавать 2D игры и графические приложения, которые

могут работать на всех современных устройствах без установки дополнительных плагинов.

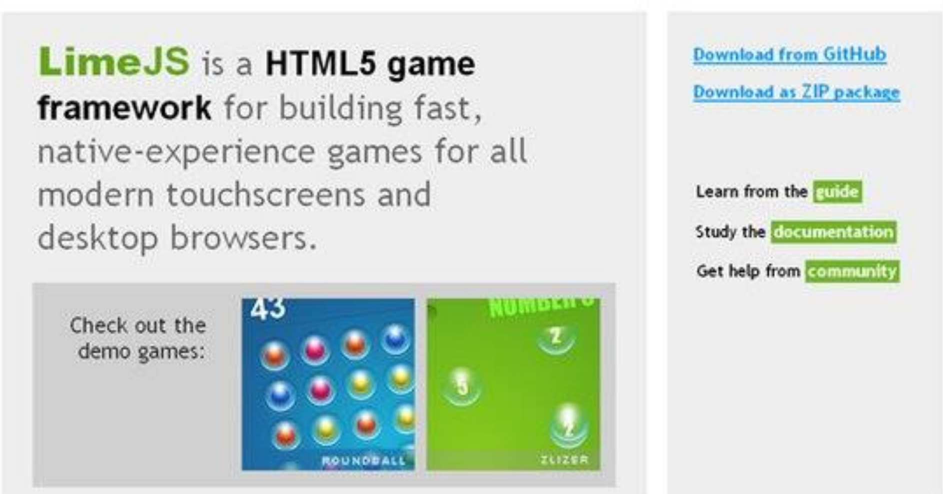
CopperLicht

WebGL библиотека и JavaScript 3D движок для создания браузерных игр и 3D приложений. Использует WebGLCanvas, поддерживаемый современными браузерами и способный поддерживать рендеринг 3D моделей, используя аппаратное ускорение без плагинов.

Aves

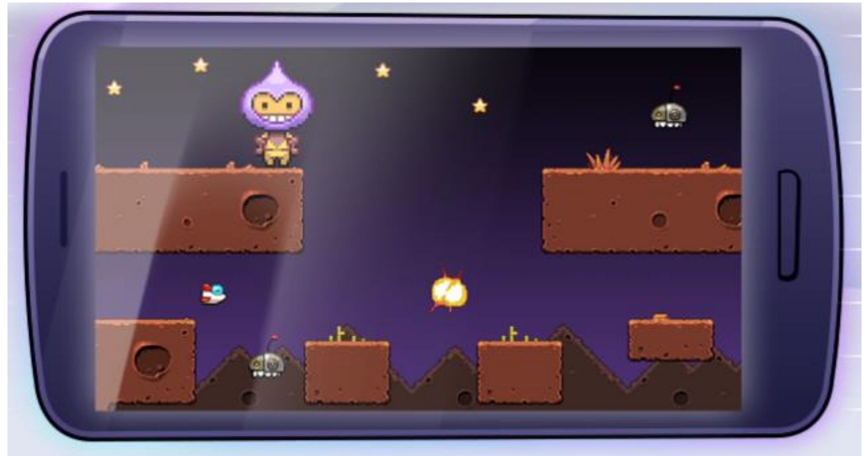
Этот HTML/JavaScript движок – реинкарнация набора инструментов для разработки олдскульных RPG (но с более привлекательной графикой). И все только с помощью HTML и JS. Никаких плагинов. Никакого Flash.

LimeJS



HTML5 движок для разработки игр с поддержкой сенсорного ввода. LimeJS создан с использованием ClosureLibrary, созданной Google, и в нем уже реализованы классы и функции для отслеживания времени, событий, обработки форм и анимации. Также фреймворк поддерживает спрайтовые листы (т.е. все используемые изображения могут быть помещены в один файл).

Phaser



Ещё один фреймворк для создания мобильных и десктопных игр на HTML5 с применением Canvas и WebGL. Бесплатный и с открытым исходным кодом. Есть быстрые гайды для старта на JavaScript и TypeScript.

Глава24. Разработка интернет приложений

Создание интерактивных Интернет сервисов подразумевает широкое применение JavaScript при их разработке. Если 10 лет назад этот язык имел скудным функционал, но в данный момент JS существенно преобразился. Все больше логики сервисов переносится в клиентскую часть. В статье мы поделимся с вами нашим опытом проектирования и реализации крупных веб приложений.

Подход к проектированию

Код больших проектов имеет одну не самую приятную особенность. При превышении его объемом определенного порога, он становится похож на запутанный клубок. В таком случае скорость разработки падает – много времени тратиться на его распутывание. Потому следует разделять код по логике его работы. Использование таких паттернов проектирования, как MVC, MVVP и им подобным, существенно структурирует код, сделает его более гибким и масштабируемым, систематизирует разработку приложения. Выбор конкретного паттерна может быть обусловлен как требованиями к реализации архитектуры сервиса, так и личными предпочтениями разработчиков.

В 99% случаев чтобы не изобретать велосипед, стоит использовать фреймворки, в которых уже реализованы указанные выше паттерны. Наиболее распространенными являются Backbone.js, Ember.js, Knockout.js.

Разработка типового приложения

Предположим, что перед нами встала задача по разработке одностраничного RESTful приложения. (назовем его условно “SampleApp”). Залог успеха при построении масштабируемых и гибких приложений – наличие возможности отключить какой-либо функционал, либо добавить оный не влияя на общий ход работы системы. Для этого все сущности приложения разделяем на модули.

Для обеспечения модульности наших скриптов воспользуемся механизмом пространства имен. В JavaScript нет привычного, скажем для C-

разработчиков, инструмента для их объявления. Их реализация заключается в создании глобального объекта, в поля которого записываются компоненты системы.

```
(function() {
window.sample = {}; // Главный объект приложения
sample.routers = {}; // Объект, содержащий контроллеры для обеспечения
                     // переходов между разделами приложения.
sample.models = {}; // Объект для хранения моделей с данными
sample.ui = {}; // Объект, содержащий контроллеры, отвечающие за
                // построение интерфейсов приложения и их поведения.
sample.core = {state: null}; // Ядро нашего приложения. Содержит в
                             // себе объект, являющуюся «песочницей».
sample.modules = {}; // Объект, содержащий контроллеры других
                     // модулей системы
})();
```

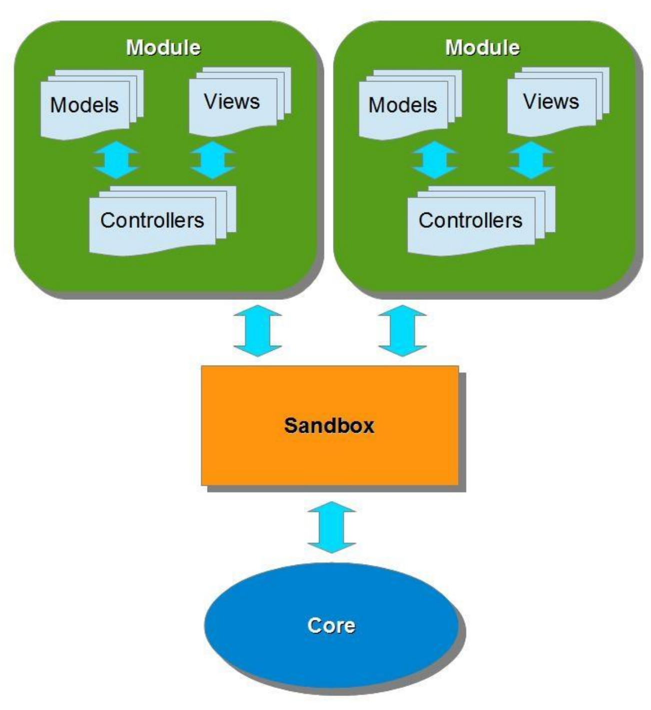
Все создаваемые сущности мы в дальнейшем будем распределять внутрь созданных объектов. Пример на backbone.js (не обращайте внимания, что мы расширяем объект *Backbone.View.* На самом деле данный объект реализует не представление, а контроллер, отвечающий за логику интерфейса):

```
sample.ui.MainPage = Backbone.View.extend({
  anotherField: ‘’,  // public field
initialize: function() {
          // Здесь код, выполняющийся при инициализации объекта
          varsomeField = ‘’;  // private field
  },
     render: function() {
          // рендерингтемплейтовит.п.
this.renderWidjet1();
```

```
this.renderWidjet2();
this.renderWidjet3();
  },
  renderWidjet1: function() {
  },
      renderWidjet2: function() {
  },
      renderWidjet3: function() {
  }
});
sample.ui.mainPage = new sample.ui.mainPage();
sample.ui.mainPage.render();
```

Архитектура

Модули не должны знать о существовании друг друга. Связь между ними происходит через «песочницу». Ею выступает объект, хранящий в себе состояний приложения и методы для взаимодействия между модулями.



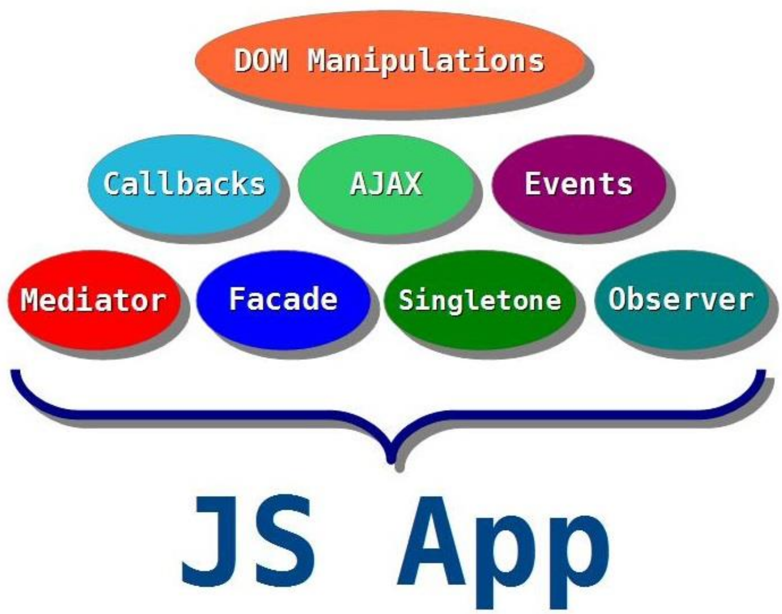

```
// Объект-«песочница»
sample.core.AppState = Backbone.Model.extend({
```

```
message: null,
receiver: null,
  // Методопределенияприемника
setReceiver: function(receiver) {
            this.set('receiver', receiver);
  },
  // Методпересылкисообщения
  proceedMessage: function(message, success, error) {
  var receiver = this.get('receiver');
            if (receiver) {
                      this.set('message', message) ;
                      receiver.receiveMessage(message);
if (success) {
                                success.call();
                      }
            }  else if (error) {
                      error.call();
            }
  }
});
// Объект-отправитель
sample.modules.Sender = Backbone.View.extend({
  message: 'text to send',
initialize: function() {
            this.model = sample.core.state;
  },
  sendMessage: function() {
            // отправляем сообщения в «песочницу»
            this.model.proceedMessage(this.message,
this.successMessage, this.errorMessage);
```

```
  },
  successMessage: function() {
          alert('OK!');
  },
errorMessage: function() {
          alert('THERE IS NO RECEIVER!');
  }
});
// Объект-приемниксообщений
sample.modules.Receiver = Backbone.View.extend({
initialize: function() {
this.model = sample.core.state;
// Сообщаем «песочнице», что пересылать сообщения нужно нам
this.model.setReceiver(this);
  },
  receiveMessage: function(message) {
          alert(message);
  }
});
// Объект-подписчик на событие
sample.modules.Listener = Backbone.View.extend({
initialize: function() {
var self = this;
          this.model = sample.core.state;
          // Подписываемся
this.model.bind(‘change:message’: function() {
  self.listenMessage(self.model.get(‘message’));
          });
  },
     listenMessage: function(message) {
```

```
            console.log(message);
        }
    });
    sample.core.state = new sample.core.AppState();
    sample.modules.receiver = new sample.modules.Receiver();
    sample.modules.sender  = new sample.modules.Sender();
    sample.modules.listener = new sample.modules.Listener();
```

В приведенном примере представлены 3 модуля: отправитель, получатель и подписчик, а также «песочница» для их связи. Несложно заметить, что каждый из объектов представляет собой применение паттерна проектирования Singletone. В тоже время «песочница» построена на основе шаблона Mediator, выступая посредником меду отправителем и получателем. Шаблон подписчика – Observer, который отслеживает изменения поля посредника. Это реализовано через механизм событий. Данный инструмент является ключевой особенностью языка JavaScript. При разработке сложного приложения стоит как можно чаще использовать его. Стандартные события ограничиваются действиями пользователя и манипуляцией DOM, но различные библиотеки и фреймворки расширяют данный инструмент, позволяя подписываться на изменения данных в моделях и т.п.



Построение масштабируемого JS приложения подразумевает широкое использование callback-методов. Функции обратного вызова – один из основных инструментов построения асинхронного кода. В примере выше использование callbacks позволяет привязать обработчики ошибок отправки сообщений к объекту-отправителю.

Backend и модель данных

В RESTful приложениях связь с бэкэндом осуществляется через API, реализующий выдачу требуемых данных в определенном формате. Как правило это JSON. Идея построения связи фронтэнд-бэкэнда заключается в привязке модели на клиенте к определенному серверному методу.

```
sample.models.SomeModel = Backbone.Model.extend({
initialize: function() {
  },
  url: function() {
        return 'some/url';
    }
});
sample.models.someModel = new sample.models.SomeModel;
sample.models.someModel.fetch(); //забираемданныесбэкэнда
sample.models.someModel.save(); //отправляемданныенабэкэнд
```

Маршрутизация запросов

Переход между разделами одностраничного приложения реализуется через специальный объект-маршрутизатор. Такие объекты также часто называют «Машинами состояний» (StateMachines). В Backbone.js такой объект также реализует HTM5 History API.

```
sample.routers.App = Backbone.Router.extend({
routes: {
    "": "initPage",
    "!": "initPage",
"!/": "initPage",
```

```
"!/about": "aboutPage",
"!/chat": "chatPage",
"!/player": "playerPage"
    },
initialize: function() {
        // Создаемобъектымодулей
    },
initPage: function() {
    },
aboutPage: function() {
    },
chatPage: function() {
    },
playerPage: function() {
    }
});
// Запускаеммаршрутизацию
$(document).ready(function() {
sample.routers.app = new sample.routers.App();
Backbone.history.start();
});
```

Другие особенности

Разделы приложения могу содержать общие для всех них элементы. Такие объекты не должны перегружаться при переходе с одной страницы на другую (например видеоплеер). Для этого необходимо для каждого виджета создать на странице отдельный контейнер. Все отображения элементов производятся через манипуляции с деревом DOM. Это даст нам возможность менять лишь нужные блоки раздела. Но помните, что работа с DOM – процесс ресурсоемкий. Старайтесь максимально оптимизировать эти действия.

Если ваш проект содержит сценарии, совершающие тяжелые вычисления – используйте воркеры. Данный механизм позволит вынести сложную логику во внешний скрипт, выполняемый в фоновом потоке. Обмен данных между воркерами и основным кодом ведется при помощи текстовых сообщений, т.е. при необходимости передачи объектов их нужно сериализовать. Сериализация может быть достаточно трудоёмким действием. Поэтому всегда помните: **использование воркеров оправдано лишь когда время сериализации и десериализации меньше времени последующих вычислений.**

Структура размещения кода

При разработке крупного приложения предельно важно четко структурировать разрабатываемый код. Поддерживать проект, в котором все находится в одном файле, крайне тяжело. Нужно принять за правило следующий подход: каждая сущность (объект, модель, контроллер и т.п.) = отдельный файл. Объедините сущности по их назначению и сложите в директории. Например все модели в директорию ‘models’, представления – в ‘templates’ и т.д.

Сборка всех файлов в единый может происходить как простым объединением bash-скриптом, так и использованием специальных библиотек (например require.js).

РАЗДЕЛ 5. ОСНОВЫ PHP

Глава 25. Введение в программирование на стороне сервера

Большинство крупных веб-сайтов используют программирование серверной части чтобы динамично отображать различные данные при необходимости, в основном взятые из базы данных, располагающейся на сервере и отсылаемые клиенту через некоторый код (например, HTML и JavaScript). Возможно, самая значительная польза программирования серверной части в том, что оно позволяет формировать контент веб-сайта под конкретного пользователя. Динамические сайты могут подсвечивать контент, более соответствующий предпочтениям и привычкам пользователя. Это также делает сайты более простыми для использования благодаря хранению личных предпочтений и информации, например, используя сохраненные данные кредитной карты для упрощения последующих платежей. Это также делает возможность взаимодействовать с пользователем сайта, посылая уведомления и обновления по электронной почте или по другим каналам. Все эти возможности создают условия для более глубокого взаимодействия с пользователями.

В современном мире веб-разработки знания о программировании серверной части настоятельно рекомендуемы.

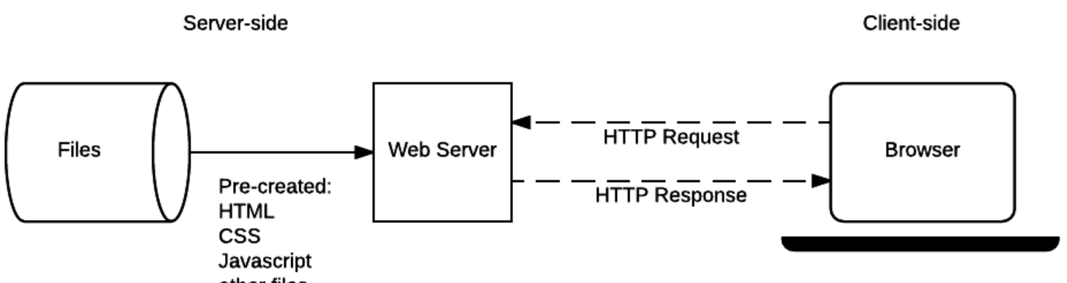
Что такое программирование серверной части сайта?

Веб браузеры взаимодействуют с веб-серверами при помощи гипертекстового транспортного протокола (HTTP). Когда вы нажимаете на ссылку на веб-странице, заполняете форму или запускаете поиск, **HTTP запрос** отправляется из вашего браузера на целевой сервер. Запрос включает в себя URL, определяющий затронутый ресурс, метод, определяющий требуемое действие (например, получить, удалить или опубликовать ресурс) и может включать дополнительную информацию, закодированную в параметрах URL (пары поле-значение, оправленные как строка запроса), как POST запрос (данные, отправленные в формате HTTP POST), или в куки-файлах.

Веб серверы ожидают сообщений с клиентскими запросами, обрабатывают их по прибытию и отвечают веб-браузеру при помощи ответного HTTP сообщения. Ответ содержит строку состояния, показывающую, был ли запрос успешным, или нет (например, "HTTP/1.1 200 OK" в случае успеха. Тело успешного ответ на запрос может содержать запрашиваемые данные (например, новую HTML страницу, или изображение, и т.п), который может отображаться через веб-браузер.

Статические сайты

Схема ниже показывает базовую архитектуру веб-сервера для статического сайта (статический сайт – это такой сайт, который возвращает один и тот же жестко запрограммированный контент с сервера всякий раз, когда определенный запрос поступает). Когда пользователь хочет зайти на страницу, браузер посылает HTTP-запрос “GET”, указывая ее путь. Сервер извлекает запрошенный документ из своей файловой системы и возвращает HTTP-ответ, содержащий документ и успешный статус (обычно 200 OK). Если файл не может быть извлечен по каким-либо причинам, возвращается статус ошибки (смотри ошибки клиента и ошибки сервера).



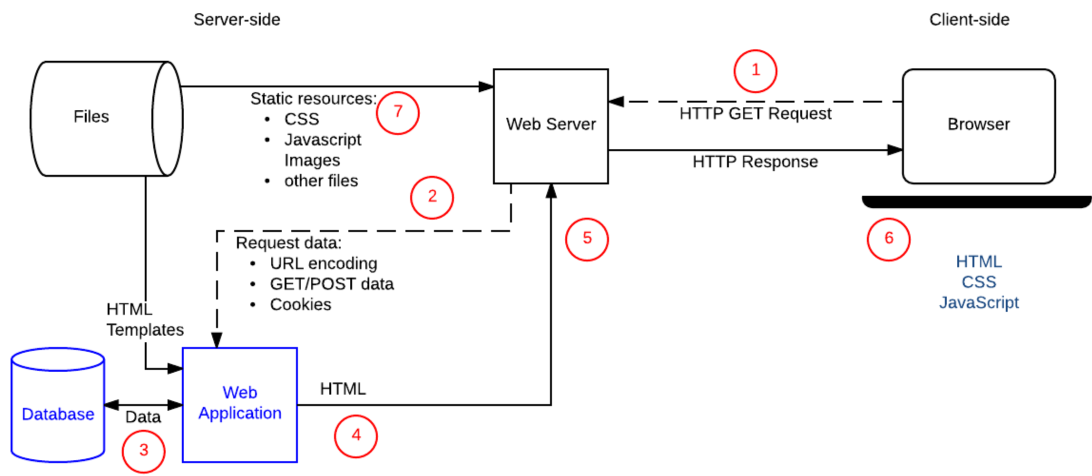
Динамические сайты

Динамический веб-сайт – это сайт, на котором ответный контент генерируется *динамически* только когда это требуется. На динамическом веб-сайте HTML страницы обычно создаются вставлением данных из базы данных в плейсхолдеры в HTML шаблонах (это намного более эффективный путь хранения большого количества контента, чем использование статических сайтов). Динамический сайт может возвращать различные данные в зависимости от информации, основанной на URL, поставляемой

пользователем или сохраненные предпочтения и может производить другие операции как часть ответа (например, отправка уведомлений).

Большая часть кода для поддержки динамического веб-сайта должна запускаться на сервере. Создание этого кода известно, как «**программирование серверной части**» (или иногда «**программирование бэкенда**»).

Схема ниже показывает простую архитектуру *динамического сайта*. Как и на предыдущей схеме, браузеры отправляют HTTP-запросы на сервер, затем сервер обрабатывает запросы и возвращает соответствующие HTTP-ответы. Запросы*статических* данных обрабатываются так же, как и для статических сайтов (статические данные – это любые файлы, которые не изменяются, обычно это: CSS, JavaScript, изображения, заранее созданные PDF-файлы и прочее).



Запросы динамических данных отправляются (2) в код серверной части (показано на диаграмме как *Веб-приложение*). Для «динамических запросов» сервер интерпретирует запрос, читает необходимую информацию из базы данных (3), комбинирует извлеченные данные с шаблонами HTML и возвращает ответ, содержащий сгенерированный HTML (5, 6).

Одинаково ли программирование серверной части и клиентской?

Теперь обратим внимание на код, задействованный в серверной части и клиентской части. В каждом случае код существенно различается:

- Они имеют различные цели и назначение.

- Они обычно не используют один язык программирования (исключение составляет, который может быть использован и в клиентской и в серверной частях).
- Они выполняются в разных средах операционной системы.

Код, который выполняется в браузере, известный как код клиентской части, прежде всего связан с улучшением внешнего вида и поведения генерируемого веб-сайта. Это включает выбор и стилизация UI компонент, вёрстка, навигация, валидация форм и так далее. Серверная часть, напротив, в основном задействована в выборе: *какое именно содержимое* возвращается браузеру в ответ на запросы. Код серверной части имеет дело с такими задачами как валидация отправленных данных и запросов, использование баз данных для хранения и извлечения данных и отправка корректных данных клиенту, как положено.

Код клиентской части написан с использованием HTML, CSS и JavaScript выполняется внутри веб-браузера и имеет ограниченный доступ, или вообще никакого к операционной системе (включая ограниченный доступ к файловой системе). Веб-разработчики не могут контролировать, какой браузер использует каждый пользователь для просмотра сайта: браузеры предоставляют противоречивые уровни совместимости с особенностями кода клиентской части и часть вызова программирования клиентской части – это изящное обращение с различиями в поддержке браузеров.

Код серверной части может быть написан на любом количестве языков программирования – примеры популярных языков серверной части включают в себя PHP, Python, Ruby и C#. Код серверной части имеет полный доступ к операционной системе сервера, и разработчик может выбрать какой язык программирования (и какую версию) он хотел бы использовать.

Разработчики обычно пишут свой код, используя **веб-фреймворки**. Веб-фреймворки – это набор функций, объектов, правил и других конструкций кода, спроектированных для решения типичных задач, ускорить

разработку и упростить разные типы задач, возникающих в конкретной области. И снова, поскольку и клиентская и серверная части используют фреймворки, области очень разные и, следовательно, фреймворки тоже разные. Фреймворки клиентской части упрощают вёрстку и представление данных, тогда как фреймворки серверной части обеспечивают много «обычного» функционала веб-сервера, который вы возможно в противном случае должны были осуществлять самостоятельно (например, поддержка сессий, поддержка пользователей и аутентификация, простой доступ к базе данных, шаблонам библиотек, и т.д.).

Внимание: Фреймворки клиентской части часто используются для ускорения написания кода клиентской части, но вы также можете решить писать весь код руками; на самом деле, написание кода руками может быть более быстрым и эффективным, если вам нужен маленький простой веб-сайт UI. И наоборот, вы практически никогда не посмотрите в сторону написания кода серверной части веб-приложения без фреймворка: осуществление жизненно важной функции, такой как HTTP сервер действительно сложно сделать с нуля скажем на Python, но веб-фреймворки для Python, такие как Django обеспечивают это из коробки вместе с другими полезными инструментами.

Что можно сделать в серверной части?

Программирование серверной части очень полезно поскольку позволяет *эффективно*доставлять информацию, составленную для индивидуальных пользователей и таким образом создавать намного лучший опыт использования.

Компании, такие как Amazon, используют программирование серверной части для построения исследовательских результатов для товаров, формирования целевого предложения, основанного на предпочтениях клиента и предыдущих покупках, упрощения заказов и т.д. Банки используют программирование серверной части, чтобы хранить учетную информацию и

позволять только авторизованным пользователям просматривать и совершать транзакции. Другие сервисы, такие как Facebook, Twitter, Instagram и Wikipedia используют бэкэнд, чтобы выделять, распространять и контролировать доступ к интересному контенту.

Некоторые типичные применения и выгоды бэкэнда перечислены ниже. Вы заметите, что есть некоторое пересечение!

Эффективное хранение и доставка информации

Представьте, сколько товаров доступно на Amazon и представьте, сколько постов было написано на Facebook? Создание статической страницы для каждого товара или поста было бы абсолютно неэффективным.

Программирование серверной части позволяет вместо этого хранить информацию в базе данных и динамически создавать, и возвращать HTML и другие типы файлов (например, PDF, изображения, и т.д.). Также есть возможность просто вернуть данные (JSON, XML, и т.д.) для отображения используя подходящий фреймворк клиентской части (это уменьшает загрузку процессора на сервере и количество передаваемых данных.

Сервер не ограничен в отправке информации из баз данных и может вместо этого возвращать результат инструментов программного обеспечения или данные из сервисов коммуникации. Контент даже может быть целевым относительно устройства клиента, который его получает.

Из-за того, что информация находится в базе данных, её также можно легко передать и обновить через другие бизнес системы (например, отслеживание).

Внимание: Вам не нужно сильно напрягать свое воображение, чтобы увидеть достоинства кода серверной части для эффективного хранения и передачи информации:

Зайдите на Amazon или другой интернет-магазин.

Введите в поиск несколько ключевых слов и заметьте, как структура страницы не изменилась, тогда как результаты изменились.

Откройте два или три разных товара. Заметьте, что они имеют схожую структуру и внешний вид, но содержимое для разных товаров было вставлено из базы данных.

Для обычного поиска (например, «рыба») вы можете увидеть буквально миллионы найденных значений. Использование базы данных позволяет этому храниться и передаваться эффективно позволяет контролировать представление информации всего в одном месте.

Настраиваемый пользовательский опыт взаимодействия

Серверы могут хранить и использовать информацию о клиентах чтобы поставлять удобный и сделанный индивидуально пользовательский опыт взаимодействия. Например, многие сайты хранят данные кредитных карт, чтобы не нужно было вводить их повторно. Сайты наподобие GoogleMaps используют домашний адрес и текущее местоположение для предоставления информации о маршруте и выделяя местные учреждения в результатах поиска.

Более глубокий анализ привычек пользователя может быть использован для предвидения их интересов и дальнейших настроек ответов и уведомлений, например, обеспечение списка ранее посещенных популярных местоположений, которые вы возможно захотите найти на карте.

Внимание: Зайдите в GoogleMaps как анонимный пользователь, выберете кнопку «Как добраться» и введите начальную и конечную точки для проезда. Теперь войдите в систему используя ваш гугл аккаунт, если он у вас есть (информация об этом появляется на панели внизу, когда вы выбираете маршрут). Веб-сайт теперь позволит выбрать местоположение дома и работы и начальные и конечные точки (или сохранить эти детали, если вы их еще не вводили).

Контролируемый доступ к контенту

Программирование серверной части позволяет сайтам ограничивать доступ авторизованным пользователям и предоставлять только ту информацию, которую пользователю разрешено видеть.

Реальные примеры:

- Социальные сети, такие как Facebook позволяют пользователям полностью контролировать свои данные, но только разрешать своим друзьям просматривать и комментировать их. Пользователь определяет, кто может просматривать его данные и более того, чьи данные появляются на их стене. Авторизация — центральная часть опыта взаимодействия.
- Сайт, на котором вы находитесь прямо сейчас контролирует доступ к контенту: статьи видны всем, но только авторизованные пользователи могут редактировать контент. Чтобы попробовать это, нажмите на кнопку «Редактировать» вверху страницы, и, если вы авторизованы, вы увидите редакторский интерфейс, если нет — вас отправит на страницу авторизации.

Внимание: Рассмотрим другие реальные примеры, где доступ к контенту контролируется. Например, что вы можете увидеть, если зайдете на сайт вашего банка? Авторизуйтесь через вашу учетную запись и какую дополнительную информацию вы можете просматривать и редактировать? Что за информацию вы можете увидеть, которую может редактировать только банк?

Хранение информации о сессии/состоянии

Программирование серверной части позволяет разработчикам использовать сессии – изначально это механизм, позволяющий серверу хранить информацию о текущем пользователе сайта и отправлять разные ответы, основанные на этой информации. Это позволяет, например, сайту знать, что пользователь был предварительно авторизован и отображает ссылки на их адрес электронной почты или историю заказов, или возможно сохранить прогресс простой игры, так чтобы пользователь мог при следующем заходе на сайт продолжить оттуда, где он закончил.

Внимание: Посетите новостной сайт, у которого есть подписка и откройте ветку тегов (например, TheAge). Продолжайте посещать сайт в течении нескольких часов/дней. В итоге вас начнет перенаправлять на

страницы, объясняющие, как подписаться и статьи станут вам недоступны. Эта информация является примером сессии, сохраненной в куки-файлах.

Уведомления и средства связи

Серверы могут отправлять общие или пользовательские уведомления непосредственно через сайт или по электронной почте, смс, мгновенные сообщения, видеосвязь или другие средства связи.

Вот несколько примеров:

- Facebook или Twitter отправляет уведомления по электронной почте и смс-сообщения, чтобы уведомить вас о новых разговорах.
- Регулярно отправляет письма на электронную почту, предлагающие товары, похожие на те, которые уже были куплены или просмотрены, которые могут вас заинтересовать.
- Веб-сервер может посылать предупреждения администратору сайта, предупреждая его о том, что на сервере заканчивается память или о подозрительной активности пользователя.

Внимание: Самый распространенный вид уведомлений – это «подтверждение регистрации». Возьмите почти любой интересующий вас большой сайт (Google, Amazon, Instagram, и т.д.) и создайте новую учетную запись используя ваш адрес электронной почты. Вы вскоре получите письмо, подтверждающую вашу регистрацию или с необходимой информацией для активации вашей учетной записи.

Анализ данных

Веб-сайт может собирать много данных о своих пользователях: что они ищут в поиске, что они покупают, что они рекомендуют, как долго они остаются на каждой странице. Программирование серверной части может быть использовано, чтобы усовершенствовать ответы, основанные на анализе этих данных.

Например, и Amazon и Google рекламируют товары на основании предыдущих поисков (и покупок).

Выводы

Теперь вы узнали, что код серверной части выполняется на веб-сервере и его основная роль состоит в контролировании отправляемой пользователю информации (тогда как код клиентской части в основном определяет структуру и способ преподнесения информации пользователю). Вы должны также понимать, что это полезно, так как позволяет создавать веб-сайты, которые эффективно доставляют информацию, собранную для конкретных пользователей и иметь четкое представление о некоторых вещах, которые вы сможете делать, когда станете разработчиком бэкенда.

Наконец, вы должны понимать, что код серверной части может быть написан на ряде языков программирования, и что вам следует использовать веб-фреймворк для упрощения процесса.

Глава 26. Программирование на PHP. Введение в ООП на PHP. Регулярные выражения в PHP. Работа с БД на PHP.

Работа с формами на PHP.**PHP** (*HypertextPreProcessor*) – один из самых популярных инструментов веб-программирования на стороне сервера. Работа PHP в самом простом варианте сводится к обработке http запроса клиента. Обработка запроса, в свою очередь, заключается в программном формировании гипертекста в соответствии с параметрами запроса, после чего полученная разметка возвращается клиенту. Когда клиент (интернет браузер) запрашивает обычную статическую интернет страницу (чаще всего с расширением html), сервер в качестве ответа возвращает ему содержимое этой страницу без изменений “как есть”. Если запрашивается php страница, то в процессе обработки запроса содержимое указанной страницы сначала обрабатывается интерпретатором PHP, и только потом результат этой обработки отправляется клиенту.

Другими словами, **PHP – это препроцессор гипертекста**, что и отражено в его названии. **Пре**процессор потому что окончательной

обработке гипертекст подвергается уже на стороне клиента, результат которой мы видим в окне браузера (процессором гипертекста является уже сам браузер). Можно сказать, что PHP – это генератор гипертекста, поскольку в большинстве случаев его работа – это программная генерация HTML разметки по содержимому базы данных или по любой другой структурированной информации, размещенной на сервере. Аббревиатура выглядит, как PHP, а не как, к примеру, HPP или иначе, поскольку первоначально расшифровывалась как *PersonalHomePageTools* – инструментарий для создания персональных интернет страниц. Таким образом, первый вариант расшифровки PHP отражал его назначение, а нынешний – принцип работы.

PHP – это язык программирования, который поддерживает практически все основные конструкции процедурного программирования: переменные, условные операторы, циклы, функции и т.д. PHP – это *объектно-ориентированный язык программирования* – он поддерживает классы и объекты, а также привычное наследование на уровне классов. PHP – это язык веб-программирования, поскольку он в первую очередь создан для разработки динамических интернет сайтов и поэтому содержит большое количество готовых решений, применяемых в этой сфере, таких как:

- обработка и извлечение параметров *http запросов* GET и POST;
- формирование и отправка *http заголовков*;
- инфраструктура для хранения *данных сеанса*;
- программные сервисы для работы с *cookies*;

cookies - текстовые данные, сохраняемые браузером на компьютере клиента, которые чаще всего содержат параметры доступа (логин и пароль) или персональные настройки пользователя. Cookies формируются браузером и автоматически, в ходе каждого удаленного обращения пересылаются серверу в заголовках http запроса.

- работа с файлами по *FTP протоколу*;
- работа с базами данных посредствам SQL запросов;
- поддержка регулярных выражений;
- поддержка *HTTP авторизации*;
- обмен сообщениями по электронной почте и многое другое.

В этой главе кратко рассмотрим ключевые моменты применения PHP для создания несложных веб-приложений. Материалы будут организованы в несколько разделов, по каждому из которых будут приведены примеры с их исходным кодом. Для самостоятельных экспериментов необходим свой сайт или любой другой сайт, к которому у вас имеется полный доступ, и на сервере которого установлен PHP.

Основы программирования на PHP

1. Добавление PHP в HTML код
2. Приложение из нескольких php файлов
3. Переменные и функции
4. Операторы
5. Массивы, строки и другие элементы программирования на PHP

Добавление в разметку кода PHP и результат работы препроцессора гипертекста

Программный код PHP добавляется непосредственно в любое место HTML разметки. Самой HTML разметки может и не быть вовсе, а исходный код страницы может быть представлен только фрагментом программы на PHP. В любом случае, для вставки PHP нужно использовать специальный тег и внутри него размещать текст программы. Делается это следующим образом:

```
<?php
код программы
?>
```

В ходе работы PHP интерпретатора секции `<?php … ?>` заменяются на разметку, формируемую в результате работы размещенного в них

программного кода. Для вывода результата работы препроцессора используется оператор *echo*, аргументами которого могут быть константы, переменные, функции или различного рода выражения, а результатом – текст. Самый простой вариант может выглядеть следующим образом:

```
<?php
echo "<h1>данная разметка сформирована программно интерпретатором PHP.</h1>";
?>
```

Если вы откроете исходный код полученной страницы в браузере, то никакого PHP там уже не будет (если, конечно, на сервере установлен интерпретатор PHP). Смысла в таком использовании оператора `echo` не особо много. Вся прелесть PHP в том, что формируемый HTML может зависеть от параметров запроса, содержимого базы данных, политики безопасности и многого другого. Анализ и обработка всего этого выполняется с использованием знакомых практически всем конструкций процедурного языка программирования, таких как циклы, условия, функции и т.п. Забегая вперед, приведу небольшой пример программы на PHP с использованием цикла и условного оператора, с тем, чтобы начальное представление о препроцессоре гипертекста стало более полным. Следующая программа выводит значения факториала чисел от 1 до 9.

```
<html>
<head>
<title>Пример программы на PHP</title>
</head>
<body>
<?php
echo "<h1>данная разметка сформирована программно интерпретатором PHP.</h1>";
$f=1;
for ($i=1; $i<10; $i++)
```

```
 {
if ($i>1)
    $f=$f*$i;

echo $i,"!=",$f,"<br/>";
}
 ?>
</body>
</html>
```

Результат ее работы будет выглядеть в браузере примерно следующим образом:

```
1!=1
2!=2
3!=6
4!=24
5!=120
6!=720
7!=5040
8!=40320
9!=362880
```

Организация приложения из нескольких файлов PHP

Многие интернет сайты состоят из страниц, которые созданы по одному и тому же шаблону. Меняется только содержимое основной области. Одним из наиболее важных применений PHP является возможность описания шаблонов (макетов) динамических веб-страниц. Сам макет веб-страницы представляют в виде набора областей: заголовок, левая и правая панели, основная область, подвал и т.д. и для каждой из них создают отдельный php файл, генерирующий соответствующую часть разметки. Для создания страницы сайта по такому шаблону достаточно подключить нужные области, добавив ссылки на созданные php файлы в определенном порядке, а в

качестве содержимого добавить только разметку для основной области, которой страница будет отличаться от всех остальных разделов сайта.

Для подключения внешнего файла в php имеются две инструкции *require* и *include*. Инструкция **require** вставляет содержимое указанного файла **до начала обработки** программного кода интерпретатором PHP, а инструкция **include** включает его **в процессе обработки** – когда интерпретатор дойдет до соответствующей строки в программе. Вот так может выглядеть разметка интернет страницы, создаваемой по шаблону.

```
<?php
require "header.php";
require "side_left.php";
?>
разметка основной области
<?php
require "side_right.php";
require "footer.php";
?>
```

Таким образом, если файл, на который ссылается инструкция require, недоступен, то дальнейшая обработка страницы завершается. В случае с include обработка недоступного файла просто пропускается, и продолжается обработка оставшегося PHP кода. Как следствие того, что ссылки include обрабатываются в процессе интерпретации, подключением php файла с помощью этого оператора можно управлять в зависимости от различных условий, с использованием оператора `if`, многократно в теле цикла `for` и т.д. Вот маленький пример.

```
<?php
if ($value==$pass)
include "login.php";
else
include "fail.php";
```

```
?>
```

Содержимое файлов, ссылки на которые описаны с использованием инструкции require будут вставлены независимо ни от каких условий ровно в те места, где они размещены.

Еще один момент: если у вас сложная структура сайта с большим количеством шаблонов, которые ссылаются один на другой, а вам необходимо, чтобы некоторые файлы были добавлены ровно один раз, то для этого следует использовать специальные варианты описанных инструкций: *require_once* и *include_once*. Эти варианты гарантируют, что указанный файл будет добавлен только один раз в том месте, где соответствующая ссылка встречается впервые. В следующем примере содержимое php файла будет подгружено только один раз.

```
<?php
for ($i=0; $i<3;$i++)
include_once "a.php";
?>
```

А в следующем примере ссылка будет разрешена два раза, поскольку инструкции обрабатываются последовательно, и вторая команда require также добавит содержимое a.php к разметке формируемой страницы.

```
<?php
require "a.php";
require "a.php";
require_once "a.php";
?>
```

Переменные и функции

PHP – это язык программирования с *динамической типизацией* (поздним связыванием), а значит, как и в случае с JavaScript, тип переменной не указывается явно, а определяется в момент присвоения ей значения. Имена переменных, как видно из ранее приведенных примеров, в PHP начинаются с символа `$`. Определение функции в PHP ничем особым от

других процедурных языков не отличается: имя, список аргументов, тело функции и инструкция `return` для возврата результата.

```
<?php
functionname($arg1, $arg2, …)
{
телофункции
returnreturn_value;
}
?>
```

Все переменные, описанные во всех подключенных php файлах и находящихся вне каких либо функций, являются глобальными, а значит, доступны из любого места php скрипта. Любая переменная, описанная в контексте функции, является локальной по отношению к ней. Иногда не все параметры, необходимые для работы функции, в силу различных обстоятельств, определяют в качестве ее аргументов. Такие параметры могут являть собой общие настройки приложения и быть определены через глобальные переменные. Чтобы обратиться из функции к значению любой глобальной переменной, достаточно определить на нее ссылку с использованием ключевого слова `global`. Вот небольшой пример всему вышесказанному:

```
<?php
   //Определение глобальных переменных
   $k=1;
$b=5;

   //y(x)=|kx+b|
function y($x)
{
    //Обращение к глобальным переменным
global $k;
```

```
global $b;

    //Определение локальной переменной
    $y=$k*$x+$b;

if ($y<0)
return -$y;

return $y;
    }

   //Вызовфункции
echo y(10), "<br/>";
echo y(-10), "<br/>";
?>
```

Таким образом, если внутри функции определить локальную переменную (без ключевого слова global) с именем, совпадающим с именем любой глобальной переменной, то значению последней ничего не грозит. Следует понимать, что в любом месте кода текущего php файла вам доступны все глобальные переменные и все функции, которые представлены во всех файлах, подключенных через инструкцию require, и во всех тех файлах, которые подключены через инструкцию include до этого самого места, поскольку последующие include могут быть еще не обработаны интерпретатором.

Операторы PHP

Синтаксис PHP практически во всем заимствует синтаксические конструкции C/C++, C#, Java и других "С-образных" языков программирования. Тем, кто знаком хотя бы с одним из них, все базовые конструкции PHP не вызовут никаких вопросов. Тем, кто видит подобный синтаксис впервые или не знаком с программированием вообще, прошу

пройти по ссылкам ниже и более подробно про все это прочитать. Итак, вот те самые основные операторы или команды препроцессора гипертекста (в квадратных скобках необязательные части конструкций).

Условный оператор IF

if (*выражение*)

оператор “если истина”

[elseif (*альтернативное выражение 1*)

оператор “альтернативный условие 1”]

…

[elseif (*альтернативное выражение n*)

оператор “альтернативный условие n”]

[else*оператор “иначе”*]

Стоит отметить, что логические выражения в PHP имеют практически такой же синтаксис, что и в перечисленных мною в начале этого раздела языках. Разница только в том, что переменные начинаются с символа `$`. Итак, в логических выражениях поддерживаются следующие операторы сравнения: **равенство** `==`, строго **меньше** и строго **больше** `<`, `>`; нестрогие варианты `<=`, `>=`; и **неравенство** `!=` или `<>`. Логические операторы обозначаются следующим образом: **“И”** `and` или `&&`; **“ИЛИ”** `or` или `||`. Есть и некоторые другие. Пример `if` был приведен в первом разделе. Более подробно про условный оператор читаем здесь.

Краткий условный или тернарный оператор

Ниже приведен общий вид условного оператора присваивания с использованием тернарного оператора:

```
$результат = условие ? выражение если true : выражение если false;
```

Пример:

```
$result = ($a>5) ? $a+$b : $a-$b;
```

Если a больше 5, то переменной result присваивается значение a+b, иначе a-b.

Оператор выбора SWITCH

Альтернатива оператора if с большим количеством конструкций elseif. Выполнение операторов начинается с той секции case, со значением которой совпадет значение выражения и продолжается по всем последующим case, пока не встретится команда break - завершить выполнение. Секция default – альтернатива секции else в условном операторе.

```
switch (выражение)
{
caseзначение 1: оператор 1; [break;]
caseзначение 2: оператор 2; [break;]
caseзначение 3: оператор 3; [break;]
default :оператор, выполняемый “по-умолчанию”; [break;]
}
```

Более подробно про оператор выбора читаем здесь.

Цикл перебора FOR

```
for (начальное выражение; конечное выражение; приращение)
тело цикла;
```

Пример с циклом for найдете выше. Более подробно про *цикл for* читаем здесь.

Цикл с предусловием WHILE

```
while (условие)
тело цикла;
```

Пример делает то же самое, что и самый первый пример с циклом for:

```
$i=2; $f=1;
while ($i<10)
 {
  $f=$f*$i;
```

```
$i++;
echo $i,"!=",$f,"<br/>";
 }
```

Более подробно про *цикл while* читаем здесь.

Цикл с постусловием DO ... WHILE

При таком подходе тело цикла выполняется минимум один раз независимо от условия.

```
do
тело цикла;
while (условие);
```

Пример тот же:

```
$i=2; $f=1;
do
 {
  $f=$f*$i;
  $i++;
echo $i,"!=",$f,"<br/>";
}
while ($i<10);
```

Более подробно про *цикл do ... while* читаем здесь.

Перебор элементов в массиве или оператор foreach

Про массивы еще будет сказано несколько слов отдельно, а ниже конструкция для перебора его элементов в цикле:

```
foreach (массивas $элемент)
выражение, где что-то делаем с переменной $элемент;
```

Пример:

```
$values = array("for", "while", "do", "foreach");
echo "Циклывphp:","<br/>";
foreach ($values as $operator)
echo $operator, "<br/>";
```

Операторы BREAK и CONTINUE

Циклы и оператор выбора в PHP, также, как и во многих других языках программирования поддерживают инструкции *break* – досрочного завершения цикла и *continue* – досрочного завершения текущей итерации цикла (переход к началу цикла). Более подробно про них читайте здесь.

Массивы, строки и прочие особенности программирования на PHP, о которых следует знать

Комментарии

Комментарии в PHP добавляются точно так же, как в C# или C++. Для создания комментариев от текущего места и до конца строки достаточно добавить две косых черты:

```
//Комментарий до конца строки
```

Для выделения многострочных комментариев следует использовать сочетания `/*` и `*/`:

```
/*Этот комментарий занимает несколько строк.
 Такой вид комментариев удобен для отключения части программного
 кода или для вставки развернутого описания работы программы*/
```

Массивы

Если вы не знаете что такое массивы, то пройдите по ссылке по узнайте что это такое. Массивы нужны для описания процесса обработки однотипных данных. Если вы в своей программе и не создаете массивов, то как минимум имеете дело с их обработкой, поскольку массивы могут возвращаться различными функциями программного окружения PHP. Чаще всего массивы обрабатываются в циклах. В PHP не нужно заботиться о выделении и об освобождении памяти под переменные, в том числе и под

массивы, поэтому работать с ними крайне легко. Вот несколько примеров создания массивов:

```
$symbols = array("a","c","g","b","d");
```

или

```
$symbols[]="a"; $symbols[]="c"; $symbols[]="g";…
```

Обращение к элементу массивы выполняется указанием его индекса в квадратных скобках, например для вывода значения `echo $symbols[2];` или для изменения значения `$symbols[2]=”h”;` В приведенных выше примерах индексы элементов массива определяются автоматически и начинаются с 0. Индексы массива можно также задать самостоятельно, и они могут быть не только целочисленными, но и строковыми:

```
//Кодировка символов ASCII
$codes["a"]=97;
$codes["c"]=99;
$codes["g"]=103;
```

Для удаления элемента из массива в PHP следует использовать специальную функцию *unset*, например `unset ($symbols[“c”]);`

Многомерные массивы в PHP также поддерживаются, а обращение к элементу многомерного массива – это последовательное указание индексов по всем измерениями, каждое измерение указывается в отдельных квадратных скобках, например `$cell[0][5][2]=”0-5-2”;` Интерпретатор PHP предоставляет большое количество встроенных функций для работы с массивами, которые позволяют разбивать, объединять, сортировать массивы, осуществлять в них поиск элементов и еще делать многое другое. В этом обзоре у меня нет целей описать все и вся, что касается программирования на PHP, поэтому подробное описание всех возможностей этого языка ищите на других сайтах, приведен небольшой пример работы с массивами:

```
<?php
    //Определение массива
```

```
$words = array("массив", "цикл", "условие", "переменная", "комментарий");
//Сортировка элементов массива по алфавиту
sort($words);
//Удаление последнего элемента
unset($words[count($words)-1]);
//Вывод отсортированного массива без последнего элемента
foreach ($words as $word)
echo $word, "<br/>";
?>
```

Строки

PHP также поддерживает большое количество операций, связанных с обработкой строк. Для объединения (конкатенации) нескольких строк в одну следует использовать символ ".". Дляпреобразование любых других типов в строку и наоборот следует использовать оператор преобразования типов, например `$s=(string)3.1415;` или `$f=(float)”3.1415”;` Подробного описания всех функций работы со строками в этом обзоре вы также не найдете, поскольку все это займет довольно много места, а создавать подробный справочник в мои планы не входило. Вот небольшой пример:

```
<?php
//Объявление строки
$s1="abcdef";
//Объединение двух строк
$s2=$s1."ghi";
//Сравнение двух строк
if (strcmp($s2, "abcdefghi")==0)
echo "строка s2 равна 'abcdefghi'","<br/>";
//Преобразование строки в верхний регистр
```

```
echo "Преобразование s1 в верхний регистр: ", strtoupper($s1),"<br/>";
  //Значение длины строки
echo "Длинастроки s1 = ", strlen($s2),"<br/>";
//Преобразование целочисленного типа к строке;
$s3="длина s1=".(string)strlen($s1)."; длина s2=".(string)strlen($s2);
echo $s3,"<br/>";
  //Перевод строки в числовой формат
echo "14.8 - 7.1 = ",(string)((float)"14.8" - (float)"7.1");
?>
```

Передача аргументов по ссылке

В PHP, также как и во многих других языках программирования аргументы функции можно передавать по ссылке. Таким образом, в теле функции вы будете иметь дело не с копией переданной в качестве аргумента переменной, а с самой этой переменной. Т.е. если внутри функции значение аргумента изменяется, то изменяется и значение самой переменной, переданной по ссылке. Для обозначения того, что аргумент передается по ссылке в PHP используется символ `&`. Передача параметров по ссылке на примере функции-инкремента продемонстрирована ниже:

```
<?php
functioninc(&$x) {$x=$x+1;}
$x=1;
inc($x);
   //Будет выведено значение 2
echo($x);
?>
```

Альтернативное обращение к значению переменной

Оказывается, что в PHP к значению переменной можно обратиться не только через ее непосредственное обозначение в программе, но и через другую переменную, если значение последней будет содержать имя запрашиваемой переменной. На следующем примере, пожалуй, становится понятно, зачем разработчикам php понадобился символ $ - он символизирует обращение к значению переменной, идентификатор которой располагается непосредственно после него, будь то лексема или любая адресуемая область памяти. Для обращения к значению переменной по значению другой переменной потребуется уже два символа $ подряд. Пример:

```
<?php
  //Определение обычной текстовой переменной
  $a="переменная a";
  //Переменная ref_a содержит имя переменной
  $ref_a="a";
  //Ссылка на значение переменной 'a' с помощью значения переменной 'ref_a'
echo $a,"=",$$ref_a, "<br/>";
?>
```

Объектно-ориентированное программирование (ООП) на PHP

Объект - это набор специальных переменных - *свойств* и специальных функций - *методов*. То, что в процедурном программировании называлось переменной — в ООП называется свойство. То, что в процедурном программировании называлось функцией — в ООП называется методом класса. Созданные на основе класса объекты называются экземплярами класса или просто объекты.

Обращение из метода к свойствам только через служебное слово $this: $this->name; (обратите внимание на отсутствие знака доллара перед name) Обращение внутри метода к другому методу тоже через $this: $this->foo();

Для доступа к свойствам и методам объекта служит оператор "->": $this->name; (обратите внимание на отсутствие знака доллара перед name)

Обращение внутри метода к другому методу тоже через $this: $this->foo();. Объект создается с помощью оператора **new** на основании шаблона, называемого *классом*. Класс определяется ключевым словом **class**.

Пример 1

```
<html>
<head>
<title>Класс со свойством и методом</title>
</head>
<body>
<?php
class классN1
{
public $имя = "Маша"; // - это свойство класса доступное снаружи класса
private$Private_name;     // - это свойство доступно только методам класса
protected$Protected_name;  // это свойство доступно методам собственного класса, а также методам наследуемых классов
function Привет() // - это метод класса
    {
echo"<H1>".$this->имя."! Привет!</H1>";
    }
functionПока( $a )
    {
$this->имя = $a;
echo"<H1>".$this->имя."! Пока!</H1>";
    }
}
```

```
$obj = newклассN1();
$obj->Привет();
$obj->имя = "Миша";
$obj->Привет();
$obj->Пока("Яша");
$obj->Привет();
?>
</body>
</html>
```

Модификаторы доступа в ООП:

- **public** — позволяет иметь доступ к свойствам и методам из любого места (глобальная область)
- **protected** — доступ к родительскому и наследуемому классу (область класса наследника)
- **private** — доступ только из класса, в котором объявлен сам элемент (область самого класса)

Метод по умолчанию — public. У свойств значения модификатора по умолчанию нет.

Константы класса в ООП

```
const NAME = 2;
```

Таким образом можно создавать константы и вне класса. Это именно константы класса, они не принадлежат ни одному объекту, они общие на все объекты, поэтому использование внутри метода:

```
function printname(){
echo self::NAME;
}
```

self — это сам класс! Обращение вне класса (можно вызывать из глобальной области видимости без инициализации экземпляра класса):

```
echoOurClass::NAME;
```

this и self

Внутри класса использована специальная переменная **this**. Это указатель, с помощью которого объект может ссылаться на самого себя.

Для обращения к статическим методам используется **self::**

Методу **Пока** передан аргумент точно так же, как и обычной функции. При вызове этого метода объект меняет свое свойство **имя**.

Конструктор — это метод, который автоматически вызывается при создании нового объекта: `publicfunction __construct(){}`. При инициализации6 объекта через служебную конструкцию new, PHP ищет `__construct` и если он есть, то вызывается.

Также можно создать метод, имя которого совпадает с именем класса, - такой метод также будет считаться конструктором. Конструктор может принимать аргументы, что значительно упрощает работу с классами.

Пример 2

```
<html>
<head>
<title>Класс с конструктором</title>
</head>
<body>
<?

class классN2
{
private $имя; // - это свойство класса НЕ доступное снаружи класса
function __construct( $a="Кто-тотам" )
    {
```

```
        $this->имя = $a;
    }
functionПривет()
    {
echo "<H1>".$this->имя."! Привет!</H1>";
    }
}

$obj0 = new классN2();
$obj1 = new классN2("Миша");
$obj2 = new классN2("Маша");
$obj0->Привет();
$obj1->Привет();
$obj2->Привет();
?>
</body>
</html>
```

Сложив все, изложенное выше, можно создать более осмысленный класс. Например, класс, который будет располагать данные в виде таблицы с поименнованными столбцами.

Пример 3

```
<html>
<head>
<title>Класс Table</title>
</head>
<body>
<?php
```

```
classTable
{
private$headers = [];
private$data = [];
function Table ( $headers )
    {
$this->headers = $headers;
    }
functionaddRow( $row )
    {
$tmp = [];
foreach ( $this->headers as$header )
        {
if ( ! isset( $row[$header] )) $row[$header] = "";
$tmp[] = $row[$header];
        }
array_push( $this->data, $tmp );
    }
function output ()
    {
echo"<PRE><B>";
foreach ( $this->headers as$header ) echo"$header  ";
echo"</B><BR>";
foreach ( $this->data as$y )
        {
foreach ( $yas$x ) echo"$x  ";
echo"<BR>";
        }
echo"</PRE>";
    }
```

```
}

$test = new Table (array("a","b","c"));
$test->addRow(array("a"=>1,"b"=>3,"c"=>2));
$test->addRow(array("b"=>1,"a"=>3));
$test->addRow(array("c"=>1,"b"=>3,"a"=>4));
$test->output();
?>
</body>
</html>
```

Свойства класса **Table** - массив имен столбцов таблицы и двумерный массив строк данных. Конструктор класса **Table** получает массив имен столбцов таблицы. Метод **addRow** добавляет в таблицу новую строку данных. Метод **output** выводит таблицу на экран.

Скрытые свойства и методы

Свойства и методы класса могут быть как открытыми (public), так и скрытыми (private). Скрытые свойства и методы недоступны извне класса, т.е. из сценария, в котором используется данный класс, или из другого класса.

Наследование

На основе существующих классов можно создавать новые, используя *механизм наследования*. Механизм наследования - это использование определенного ранее класса в качестве родительского. При этом набор свойств и методов родительского класса можно расширять. Имейте в виду, что производный класс имеет только одного родителя.

Чтобы создать новый класс, наследующий поведение существующего класса, надо использовать ключевое слово **extends** в его объявлении. Например:

```
class классN2 extends классN1
  {
  .......
   }
```

Здесь **классN1** - родительский класс, **классN2** - производный.

Если производный класс не содержит собственного конструктора, то при создании его объекта используется конструктор родительского класса. Если в производном класса существует собственный конструктор, то конструктор родительского класса не вызывается. При необходимости вызвать конструктор родительского класса это надо сделать явно. Например:

```
классN1::классN1();
```

Производный класс будет иметь все свойства и методы родительского класса. Но их можно и переопределить в производном классе.

Пример 4

```
<html>
<head>
<title>Переопределение метода родительского класса</title>
</head>
<body>
<?php
classклассN3
{
public $имя = "Маша";
functionПривет()
    {
echo "<H1>".$this->имя."! Привет!</H1>";
    }
}
classклассN4 extends классN3
```

```
{
functionПривет()
{
echo "<H1>".$this->имя."! Какая встреча!</H1>";
    }
}
$obj = new классN4();
$obj->Привет();
?>
</body>
</html>
```

Метод **Привет** переопределен для производного класса. Свойство **имя** наследуется от родительского.

Начиная с 4-й версии PHP, в объекте производного класса можно вызвать метод родительского класса, который был переопределен.

Пример 5

```
<html>
<head>
<title>Вызов метода родительского класса</title>
</head>
<body>
<?php
classклассN5
{
public $имя = "Маша";
functionПривет()
    {
echo"<H1>".$this->имя."! Привет!</H1>";
```

```
    }
functionПока()
    {
echo"<H1>".$this->имя.", пока!</H1>";
    }

}

/**
 * Class классN6
 */
classклассN6extendsклассN5
{
/**
     *
     */
functionПривет()
    {
echo"<H1>".$this->имя."! Какая встреча!</H1>";
        классN5::Привет();
    }
}
$obj = new классN6();
$obj->Привет();
$obj->Пока();
?>
</body>
</html>
```

Итак, производный класс может наследовать, переопределять и дополнять свойства и методы другого класса.

В следующем примере создан класс **HTMLTable**, основанный на классе **Table** из примера 3. Новый класс формирует данные, сохраненные методом **addRow** родительского класса, и выводит их в HTML-таблицу. Свойства **$cellpadding** и **$bgcolor** дают возможность изменять соответствующие аргументы, при этом переменной **$cellpadding** присваивается значение по умолчанию, равное 2.

Пример 6

```
<html>
<head>
<title>Классы Table иHTMLTable</title>
</head>
<body>
<?php

classTables
{
public$headers = [];
public$data = [];
function Tables( $headers )
    {
$this->headers = $headers;
    }
functionaddRow( $row )
    {
$tmp = [];
foreach ( $this->headers as$header )
        {
if ( ! isset( $row[$header] )) $row[$header] = "";
```

```
$tmp[] = $row[$header];
        }
array_push( $this->data, $tmp );
    }
function output ()
    {
echo"<PRE><B>";
foreach ( $this->headers as$header ) echo"$header  ";
echo"</B><BR>";
foreach ( $this->data as$y )
        {
foreach ( $yas$x ) echo"$x  ";
echo"<BR>";
        }
echo"</PRE>";
    }
}

classHTMLTableextendsTables
{
public$cellpadding = "2";
public$bgcolor;
functionHTMLTable( $headers, $bg="FFFFFF" )
    {
        Tables::Tables($headers );
$this->bgcolor = $bg;
    }
functionsetCellpadding( $padding )
    {
$this->cellpadding = $padding;
```

```
    }
function output ()
    {
echo"<table cellpadding='".$this->cellpadding."'><tr>";
foreach ( $this->headers as$header )
echo"<thbgcolor='".$this->bgcolor."'>".$header;
foreach ( $this->data as$y )
        {
echo"<tr>";
foreach ( $yas$x )
echo"<td bgcolor='".$this->bgcolor."'>$x";
        }
echo"</table>";
    }
}

$test = newHTMLTable( array("a","b","c"), "#00FFFF" );
$test->setCellpadding( 7 );
$test->addRow(array("a"=>1,"b"=>3,"c"=>2));
$test->addRow(array("b"=>1,"a"=>3));
$test->addRow(array("c"=>1,"b"=>3,"a"=>4));
$test->output();
?>
</body>
</html>
```

Обратите внимание на то, что значение свойства **cellpadding** меняется с помощью отдельного метода **setCellpadding**. Конечно, значения свойств можно менять непосредственно, вне объекта:

```
$test->cellpadding = 7 ;
```

Но это считается дурным тоном, т.к. в сложных объектах при изменении одного из свойств могут изменяться и другие свойства.

Использовать или нет технику объектного программирования? С одной стороны, проект, интенсивно использующий объектную технику, может занимать слишком много ресурсов во время выполнения. С другой стороны, правильно организованный объектный подход значительно сократит время разработки и сделает программу более гибкой.

Удаление объектов

Удалить ранее созданный объект можно следующим образом:

```
unset($objName);
```

Ниже приведен пример, в котором объект класса Car создается, а затем удаляется.

```
$myCar = new Car;
unset($myCar);
```

После вызова функции unset() объект больше не существует. В PHP имеется специальный метод __destruct(), который автоматически вызывается при удалении объекта. Ниже приведен класс, содержащий этот метод.

```
classBridge
{
function __destruct()
    {
echo"Мост разрушен";
    }
}
$bigBridge = newBridge;
unset($bigBridge);
```

При создании объекта класса Bridge, а затем его удалении отобразится следующее сообщение:

```
Мост разрушен
```

Оно отображается вследствие вызова метода __destruct() при вызове функции unset(). При удалении объекта может потребоваться акрыть некоторые файлы или записать информацию в базу данных.

Копирование (клонирование) объекта

Клонирование объекта:

```
$a = clone$b;
```

Конструктор не вызывается при клонировании, вызывается магический метод **__clone(){}**. Он НЕ принимает аргументов и к нему нельзя обратиться как к методу.

Преобразование объекта в строку

Для конвертации объекта в строку, и обратно, используются следующие функции:

serialize() - принимает объект и возвращает строковое представление его класса и свойств;

unserialize() - принимает строку, созданную при помощи serialize(), и возвращает объект.

serialize() и unserialize() работают со всеми типами данных, но они не работают с ресурсами.

Специальные методы для обслуживания функций serialize() и unserialize():

__sleep() - вызывается строго перед тем, как объект сериализуется с помощью функции serialize(). Функция __sleep() должна будет вернуть список полей класса, которые функция serialize() включит в возвращаемую строку. Вы можете использовать это для того, чтобы исключить ненужные поля из строкового представления объекта. Например:

```
publicfunction __sleep() {   // почистить
```

```
returnarray_keys( get_object_vars( $this ) );
}
```

__wakeup() - вызывается сразу после того, как объект десериализуется с помощью unserialize().

Абстрактный класс

Абстрактный класс - это класс, который не может быть реализован, то есть, вы не сможете создать объект класса, если он абстрактный. Вместо этого вы создаете дочерние классы от него и спокойно создаете объекты от этих дочерних классов. Абстрактные классы представляют собой шаблоны для создания классов.

```
abstractclassPerson {

private$firstName = "";
private$lastName = "";

publicfunctionsetName( $firstName, $lastName ) {
$this->firstName = $firstName;
$this->lastName = $lastName;
    }

publicfunctiongetName() {
return"$this->firstName $this->lastName";
    }

abstractpublicfunctionshowWelcomeMessage();
    /*  абстрактныйметодshowWelcomeMessage().
```

Так как он абстрактный, в нем нет ни строчки кода, это просто его объявление.

```
    Любой дочерний класс обязан добавить и описать метод
showWelcomeMessage() */
}
```

Интерфейс

Интерфейс - это шаблон, который задает поведение одного или более классов. Вот основные отличия между интерфейсами и абстрактными классами:

- Ни один метод не может быть описан в интерфейсе. Они все абстрактны. В абстрактном классе могут быть и не абстрактные методы.
- Интерфейс не может содержать полей - только методы.
- Класс имплементирует интерфейс, и класс наследует или расширяет другой класс.
- Класс может имплементировать несколько интерфейсов одновременно. Этот же класс может наследовать другой класс. Но у дочернего класса может быть только один супер-класс (абстрактный или нет).

```
interfaceMyInterface {
publicfunctionaMethod();
public function anotherMethod();
}

classMyClass implements MyInterface {

publicfunctionaMethod() {
// (имплементацияметода)
    }

publicfunctionanotherMethod() {
```

```
// (имплементацияметода)
}
}
```

Методы-перехватчики (магические методы)

- **__get**($property) - вызывается при обращении к неопределенному свойству
- **__set**($property,$value) - вызывается, когда неопределенному свойству присваивается значение
- **__unset**($property) - вызывается, когда функция unset() вызывается для неопределенного свойства
- **__isset**($property) - вызывается, когда функция isset() вызывается для неопределенного свойства
- **__call**($method,$argarray) - вызывается при обращении к неопределенному методу
- **__callStatic**($method,$argarray) - вызывается при обращении к неопределенному статическому методу
- **__toString**() - Вызывается, если есть попытка вывести объект, как строку.
- **__debugInfo**() - В PHP 5.6 был добавлен новый магический метод, который позволяет менять свойства и значения объекта, когда он печатается с помощью функции var_dump(класс).
- **__invoke**() - для вызова объекта как функции. Пример

Пример использования необъявленных свойств класса

Где и зачем могут быть использованны методы-перехватчики?

Например есть у вас таблица в базе данных, называется user и есть в ней некие поля, например id, name, email, phone, password, avatar И Вы создали класс на для работы с юзерами, так его и назвали - User

Какие свойства будут у данного класса? Если вы сделаете такие же как в БД - id, name, email и так далее, то получается что при каждом изменении

базы данных - вам нужно менять код в классе User, как то не очень удобно. Добавили вы например поле site - значит нужно его добавлять и в класс User, ну и так далее.

Используя же методы __get() и __set() Вы можете это всё автоматизировать. У вас в классе User вообще не будет ни одного свойства из БД, у нас есть допустим только одно $data - мы туда взяли, да и загрузили всё что есть в базе данных на данного пользователя. А потом, когда программист что то запрашивает, например $user->email мы просто в __get() методе можете посмотреть - если мы такую информацию загрузили из БД, и она лежит в $data['email'] - то вот мы её вам и возвращаем. А в __set() наоборот. Есть такое поле в БД? Значит присвоим ему новое значение.

```
/**
 * ClassUser
 * @property-readintegerid текущего пользователя
 * @property-writeStringsite возвращает ссылку на сайт пользователя
*/
classUser
{
private$data;
private$f_write=false;
publicfunction __set($name, $value) {
$this->data[$name] = $value;
$this->f_write=true; // признак, чтонужносохранитьданные
    }

publicfunction __get($name) {
if(empty($data)){
// читаем запись из БД в data
}
return$this->data[$name];
```

```
    }
function __destruct()
    {
if(!empty($data)&&$this->f_write){
// сохраняем изменения в БД
        }
    }
}

$user=newUser();
$user->site='http://kdg.htmlweb.ru/';       //присваеваем переменной
echo$user->site;       //выводим значение переменной
// записываем в БД. Можно это явно не делать, т.к. при окончании
работы скрипта это поизойдет автоматически
unset($user);
```

Пример использование необъявленного свойства класса как элемент массива

Обратите внимание на то, что из __get возвращается ссылка:

```
classFoo {
private$data = [];
publicfunction __set($name, $value) {
$this->data[$name] = $value;
    }

publicfunction& __get($name) {
return$this->data[$name];
    }
}
```

```
$foo = newFoo();
$foo->bar[2] = 'lol';
var_dump($foo->bar);
```

Использоватние перехватчиков обращения к необъявленным методам класса

```
classOurClass
{
publicfunction __call($name,array $params)
{
echo'Вы хотели вызвать $Object->'.$name.', но его не существует,
                и сейчас выполняется '.__METHOD__.'()';
return;
    }

publicstaticfunction __callStatic($name,array $params)
{
echo'Вы хотели вызвать '.__CLASS__.'::'.$name.', но его не существует,
                и сейчас выполняется '.__METHOD__.'()';
return;
    }
}

$Object=newOurClass;
$Object->DynamicMethod();
OurClass::StaticMethod();
```

Пример обхода закрытыхметов класса:

```
class_byCallStatic{
// Пример обхода "закрытых" методов класса,
```

```
// при использовании метода "__callStatic()" для вызова статического
метода.
publicstaticfunction __callStatic($_name, $_param) {
returncall_user_func_array('static::'. $_name, $_param);
  }
privatestaticfunction _newCall(){ echo'Method: '. __METHOD__; }
}
echo _byCallStatic::_newCall(114, 'Integer', 157); # Результат: Method:
_byCallStatic::_newCall
```

Как вызвать через статический метод любой динамический:

```
/**
 * Class o
 * @method static void __f(int $a1 = 1)
 */
classo
{
publicstaticfunction __callStatic($method, $args)
    {
$class = get_called_class();
$obj = new$class($args[0]);
$method = substr($method, 2);
$pass = array_slice($args,1);
$reflection = newReflectionMethod($obj, $method);
return$reflection->invokeArgs($obj, $pass);
    }

publicfunction f($a1 = 1) {
var_dump('oo', func_get_args());
    }
```

```
}
classaextendso
{
publicfunction f($a1 = 1, $a2 = 2) { var_dump('aa', $a1 ); }
}
classbextendso
{
publicfunction f($b1 = 1) { var_dump('bb', $b1); }
}
a::__f(1,2,3);
b::__f(4,5,6);
```

Полезное описание работы с ReflectionClass, когда вы можете проанализировать свойства и методы класса, проверить параметры по шаблонам и т.д.: http://habrahabr.ru/post/139649/

Как использовать объект как функцию?

```
classDog
{
private$name;
publicfunction __construct($dogName = 'Тузик') {
$this->name = $dogName;
    }
publicstaticfunction __invoke() {
$args = func_get_args();
echo'Собака получила: ' . implode(' и ', $args);
    }
}

$dog = newDog('Мухтар');
$dog('кость', 'поводок');
```

Как обращаться к объекту как к массиву?

Для этого необходимо создать такой объект который реализует интерфейс ArrayAccess из SPL. Следующий пример реализует объект доступ к данным которого можно получать как в стиле обращения к массиву, так и через получение свойств:

```
classMyArrayimplementsArrayAccess
{
protected$arr = array();
publicfunctionoffsetSet($key, $value) {
$this->arr[$key] = $value;
    }
publicfunctionoffsetUnset($key) {
unset($this->arr[$key]);
    }
publicfunctionoffsetGet($key) {
return$this->arr[$key];
    }
publicfunctionoffsetExists($key) {
returnisset($this->arr[$key]);
    }
publicfunction __get($key)
    {
return$this->offsetGet($key);
    }
publicfunction __set($key, $val)
    {
$this->offsetSet($key, $val);
    }
}
```

```
$a = newMyArray();
$a['whoam'] = 'Я значение массива, или объекта? <br/ >';
echo$a['whoam'];
echo$a->whoam;
```

Я значение массива, или объекта?

Я значение массива, или объекта?

Автозагрузка классов

Файлы автозагружаемых классов обычно располагаются в общем месте, например в /include/class/. Имя файла формируется в формате ИМЯ_КЛАССА.php. Данный код необходимо подключить во все PHP-скрипты:

```
spl_autoload_register(function($class_name) {
//echo "Autoload ".$class_name;
$file = $_SERVER['DOCUMENT_ROOT'] . "/include/class/"
.strtolower($class_name) . '.php';
if (file_exists($file) == false) {
if($GLOBALS['DEBUG']) echo"Нетфайла ".$file;
returnfalse;
    }
include_once($file);
returntrue;
});
```

Для автоподгрузки классов можно также использовать определение функции __autoload();

Обработка исключений в ООП

Для обработки некритических ошибок используются исключения(Exception).

```
try {
```

```
$a = 1;
$b = 0;
if($b == 0)
thrownewException ("делениенаноль!");
$c = $a/$b;
} catch (Exception$e) {
echo$e->getMessage();
echo$e->getLine();
}
```

Exception — встроенный класс. Если попали в throw, то код ниже не выполняется и осуществляется переход к блоку catch.

Блок try-catch используется как в процедурном, так и в ООП программировании. Он используется для отлова ошибок — большой блок try с множеством throw и все отлавливаются в одном месте — блоке catch.

Exception можно наследовать, желательно при этом перезагрузить конструктор:

```
classMyExceptionextendsException {
function __construct($msg){
parent::__construct($msg);
    }
}
```

Блоков catch может быть несколько — для каждого класса наследника Exception.

Регулярные выражения в PHP.

Регулярные выражения позволяют найти в строке последовательности, соответствующие шаблону. Например шаблон "Вася(.*)Пупкин" позволит

найти последовательность когда между словами Вася и Пупкин будет любое количество любых символов. Если надо найти шесть цифр, то пишем "[0-9]{6}" (если, например, от шести до восьми цифр, тогда "[0-9]{6,8}"). Здесь разделены такие вещи как указатель набора символов и указатель необходимого количества:

<набор символов><квантификатор><жадность>

Вместо набора символов может быть использовано обозначение любого символа - точка, может быть указан конкретный набор символов (поддерживаются последовательности - упоминавшиеся "0-9"). Может быть указано "кроме данного набора символов".

Указатель количества символов в официальной документации по php называется "квантификатор". Термин удобный и не несет в себе кривотолков. Итак, квантификатор может иметь как конкретное значение - либо одно фиксированное ("{6}"), либо как числовой промежуток ("{6,8}"), так и абстрактное "любое число, в т.ч. 0" ("*"), "любое натуральное число" - от 1 до бесконечности ("+": "document[0-9]+.txt"), "либо 0, либо 1" ("?"). По умолчанию квантификатор для данного набора символов равен единице ("document[0-9].txt").

Для более гибкого поиска сочетаний эти связки "набор символов - квантификатор" можно объединять в метаструктуры.

Как всякий гибкий инструмент, регулярные выражения гибки, но не абсолютно: зона их применения ограничена. Например, если вам надо заменить в тексте одну фиксированную строку на другую, фиксированную опять же, пользуйтесь str_replace. Разработчики php слезно умоляют не пользоваться ради этого сложными функциями ereg_replace или preg_replace, ведь при их вызове происходит процесс интерпретации строки, а это серьезно потребляет ресурсы системы. К сожалению, это любимые грабли начинающих php-программистов.

Пользуйтесь функциями регулярных выражений только если вы не знаете точно, какая "там" строка. Из примеров: поисковый код , в котором из

строки поиска вырезаются служебные символы и короткие слова а так же вырезаются лишние пробелы (вернее, все пробелы сжимаются: " +" заменяется на один пробел). При помощи этих функций проверяем email пользователя, оставляющего свой отзыв. Много полезного можно сделать, но важно иметь в виду: регулярные выражения не всесильны. Например, сложную замену в большом тексте ими лучше не делать. Ведь, к примеру, комбинация "(.*)" в программном плане означает перебор всех символов текста. А если шаблон не привязан к началу или концу строки, то и сам шаблон "двигается" программой через весь текст, и получается двойной перебор, вернее перебор в квадрате. Нетрудно догадаться, что еще одна комбинация "(.*)" означает перебор в кубе, и так далее. Возведите в третью степень, скажем, 5 килобайт текста. Получается 125 000 000 000 (прописью: сто двадцать пять миллиардов операций). Конечно же, если подходить строго, там стольких операций не будет, а будет раза в четыре-восемь меньше, но важен сам порядок цифр.

Набор символов

.	точка	любой символ
[<символы>]	квадратные скобки	класс символов ("любое из"). Например [abcdef]
[^<символы>]		негативный класс символов ("любое кроме")
-	тире	обозначение последовательности в классе символов ("[0-9]" — цифры)
\d	[0-9]	Только цифры
\D	[^0-9]	Кроме цифр
\w	[a-z0-9]	Буквы и цифры
\W	[^a-z0-9]	Кроме букв и цифр
\s	[ ]	Пробельные символы: пробел, табуляция, перевод строки
\S	[^]	Кроме пробельных символов

<table>
<tr><td>|</td><td>(одно|другое)</td><td>На этом месте может быть один из перечисленных вариантов, например: (Вася|Петя|Маша). Если Вы не хотите, чтобы это попало в выборку используйте (?: ...)</td></tr>
</table>

Не пользуйтесь классом символов для обозначения всего лишь одного (вместо "[ ]+" вполне сойдет " +"). Не пишите в классе символов точку — это ведь любой символ, тогда другие символы в классе будут просто лишними (а в негативном классе получится отрицание всех символов).

Квантификатор

Квантификатором можно указать как конкретное значение, так и пределы. Если число заданных подпадает под пределы квантификатора, фрагмент выражения считается совпавшим с разбираемой строкой. Синтаксис:

{<*количество*>}

либо

{<*минимум*>, <*максимум*>}

Если нужно указать только необходимый минимум, а максимума нет, просто ставим запятую и не пишем второе число: "{5,}" ("минимум 5"). Для наиболее часто употребляемых квантификаторов есть специальные обозначения:

*	"звёздочка" или знак умножения	{0,}
+	плюс	{1,}
?	вопросительный знак	{0,1}

На практике такие символы используются чаще, чем фигурные скобки.

Якоря

^	привязка к началу строки
$	привязка к концу строки

Эти символы должны стоять соответственно в самом начале и в самом конце строки.

Жадность

Вопросительный знак выступает еще и как минимизатор квантификатора:

```
.*?
```

```
<?
$str = '[b]жирный текст [b]а тут - еще жирнее[/b] вернулись[/b]';
$to  = '<b>$1</b>';
$re1 = '|\[b\] (.*)  \[/b\]|ixs';
$re2 = '|\[b\] (.*?) \[/b\]|ixs';
$result = preg_replace($re1, $to, $str);
echo"Жаднаяверсия: ".htmlspecialchars($result,null,'windows-1251')."<br />";
$result = preg_replace($re2, $to, $str);
echo"Ленивая версия: ".htmlspecialchars($result,null,'windows-1251')."<br />";
?>
```

Результат работы примера:

```
Жадная версия: <b>жирный текст [b]а тут - еще жирнее[/b] вернулись</b>
Ленивая версия: <b>жирный текст [b]а тут - еще жирнее</b> вернулись[/b]
```

Строка шаблона, как вы уже заметили, начинается и заканчивается слэшами. После второго идут параметры:

	регистронезависимый поиск
	многостроковый режим. По умолчанию PCRE ищет совпадения с шаблоном только внутри одной строки, а символы "^" и "$" совпадают

	только с началом и концом всего текста. Когда этот параметр установлен, "^" и "$" совпадают с началом и концом отдельных строк.
	символ "." (точка) совпадает и с переносом строки (по умолчанию — нет)
	привязка к началу текста
	заставляет символ "$" совпадать только с концом текста. Игнорируется, если установленпараметр m.
	Инвертирует "жадность" для каждого квантификатора (если же после квантификатора стоит "?", этот квантификатор перестает быть "жадным").
	Строка замены интерпретитуется как PHP код.

Функции для работы с регулярными выражениями

- preg_grep - Возвращает массив вхождений, которые соответствуют шаблону
- preg_match - Выполняет проверку на соответствие регулярному выражению. Данная функция ищет только первое совпадение!
- preg_match_all - Выполняет глобальный поиск шаблона в строке
- preg_quote - Экранирует символы в регулярных выражениях. Т.е. вставляет слэши перед всеми служебными символами (например, скобками, квадратными скобками и т.п.), чтобы те воспринимались буквально. Если у вас есть какой-либо ввод информации пользователем, и вы проверяете его с помощью регулярных выражений, то лучше перед этим заэкранировать служебные символы в пришедшей переменной
- preg_replace - Выполняет поиск и замену по регулярному выражению
- preg_replace_callback - Выполняет поиск по регулярному выражению и замену
- preg_split - Разбивает строку по регулярному выражению

preg_grep

Функция **preg_grep** - Возвращает массив вхождений, которые соответствуют шаблону

Синтаксис

array **preg_grep** (string pattern, array input [, int flags])

preg_grep() возвращает массив, состоящий из элементов входящего массива input, которые соответствуют заданному шаблону pattern.

Параметр flags может принимать следующие значения:

PREG_GREP_INVERT

В случае, если этот флаг установлен, функция preg_grep(), возвращает те элементы массива, которые не соответствуют заданному шаблону pattern.

Результат, возвращаемый функцией preg_grep() использует те же индексы, что и массив исходных данных. Если такое поведение вам не подходит, примените array_values() к массиву, возвращаемому preg_grep() для реиндексации.

Пример кода:

```
// Возвращает все элементы массива,
// содержащие числа с плавающей точкой
$fl_array = preg_grep("/^(\d+)?\.\d+$/", $array);
```

preg_match

Функция **preg_match** - Выполняет проверку на соответствие регулярному выражению

Синтаксис int **preg_match** (string pattern, string subject [, array matches [, int flags [, int offset]]]) Ищетвзаданномтексте subject совпаденияс шаблоном pattern

В случае, если дополнительный параметр matches указан, он будет заполнен результатами поиска. Элемент $matches[0] будет содержать часть строки, соответствующую вхождению всего шаблона, $matches[1] - часть строки, соответствующую первой подмаске, и так далее.

flags может принимать следующие значения:

PREG_OFFSET_CAPTURE

В случае, если этот флаг указан, для каждой найденной подстроки будет указана ее позиция в исходной строке. Необходимо помнить, что этот флаг меняет формат возвращаемых данных: каждое вхождение возвращается в виде массива, в нулевом элементе которого содержится найденная подстрока, а в первом - смещение.

Поиск осуществляется слева направо, с начала строки. Дополнительный параметр offset может быть использован для указания альтернативной начальной позиции для поиска. Аналогичного результата можно достичь, заменив subject на substr()($subject, $offset).

Функция preg_match() возвращает количество найденных соответствий. Это может быть 0 (совпадения не найдены) и 1, поскольку preg_match() прекращает свою работу после первого найденного совпадения. Если необходимо найти либо сосчитать все совпадения, следует воспользоваться функцией preg_match_all(). Функция preg_match() возвращает FALSE в случае, если во время выполнения возникли какие-либо ошибки.

Рекомендация: Не используйте функцию preg_match(), если необходимо проверить наличие подстроки в заданной строке. Используйте для этого strpos() либо strstr(), поскольку они выполнят эту задачу гораздо быстрее.

Пример кода

```
<?php
// Символ "i" после закрывающего ограничителя шаблона означает
// регистронезависимыйпоиск.
if (preg_match("/php/i", "PHP is the web scripting language of choice.")) {
echo"Вхождение найдено.";
} else {
echo"Вхождение не найдено.";
}
```

```
?>
```

Пример кода

```
<?php
/*
   Специальная последовательность \b в шаблоне означает границу
слова,
   следовательно, только изолированное вхождение слова 'web' будет
соответствовать
   маске, в отличие от "webbing" или "cobweb".
*/
if (preg_match("/\bweb\b/i", "PHP is the web scripting language of
choice.")) {
echo"Вхождениенайдено.";
} else {
echo"Вхождениененайдено.";
}
if (preg_match("/\bweb\b/i", "PHP is the website scripting language of
choice.")) {
echo"Вхождение найдено.";
} else {
echo"Вхождение не найдено.";
}
?>
```

Пример кода

```
<?php
// Извлекаем имя хоста из URL
preg_match("/^(http:\/\/)?([^\/]+)/i",
"http://www.htmlweb.ru/index.html", $matches);
```

```
$host = $matches[2];
// извлекаем две последние части имени хоста
preg_match("/[^\.\/]+\.[^\.\/]+$/", $host, $matches);
echo"domain name is: {$matches[0]}\n";
?>
```

Результат работы примера:

domainnameis: htmlweb.ru

preg_match_all

Функция **preg_match_all** - Выполняет глобальный поиск шаблона в строке

Синтаксис

int **preg_match_all** (string pattern, string subject, array matches [, int flags [, int offset]])

Ищет в строке subject все совпадения с шаблоном pattern и помещает результат в массив matches в порядке, определяемом комбинацией флагов flags.

После нахождения первого соответствия последующие поиски будут осуществляться не с начала строки, а от конца последнего найденного вхождения.

Дополнительный параметр flags может комбинировать следующие значения (необходимо понимать, что использование PREG_PATTERN_ORDER одновременно с PREG_SET_ORDER бессмысленно):

PREG_PATTERN_ORDER

Если этот флаг установлен, результат будет упорядочен следующим образом: элемент $matches[0] содержит массив полных вхождений шаблона, элемент $matches[1] содержит массив вхождений первой подмаски, и так далее.

Примеркода

```
<?php
preg_match_all("|<[^>]+>(.*)</[^>]+>|U",
"<b>example: </b><div align=left>this is a test</div>",
$out, PREG_PATTERN_ORDER);
echo$out[0][0] . ", " . $out[0][1] . "\n";
echo$out[1][0] . ", " . $out[1][1] . "\n";
?>
```

Результатработыпримера:

```
<b>example: </b>, <divalign="left">this is a test</div>
example: , thisis a test
```

Как мы видим, $out[0] содержит массив полных вхождений шаблона, а элемент $out[1] содержит массив подстрок, содержащихся в тегах.

PREG_SET_ORDER

Если этот флаг установлен, результат будет упорядочен следующим образом: элемент $matches[0] содержит первый набор вхождений, элемент $matches[1] содержит второй набор вхождений, и так далее.

Примеркода

```
<?php
preg_match_all("|<[^>]+>(.*)</[^>]+>|U",
"<b>example: </b><div align=\"left\">this is a test</div>",
$out, PREG_SET_ORDER);
echo$out[0][0] . ", " . $out[0][1] . "\n";
echo$out[1][0] . ", " . $out[1][1] . "\n";
?>
```

Результат работы примера:

```
<b>example: </b>, example:
<divalign="left">this is a test</div>, this is a test
```

В таком случае массив $matches[0] содержит первый набор вхождений, а именно: элемент $matches[0][0] содержит первое вхождение всего шаблона, элемент $matches[0][1] содержит первое вхождение первой подмаски, и так далее. Аналогично массив $matches[1] содержит второй набор вхождений, и так для каждого найденного набора.

PREG_OFFSET_CAPTURE

В случае, если этот флаг указан, для каждой найденной подстроки будет указана ее позиция в исходной строке. Необходимо помнить, что этот флаг меняет формат возвращаемых данных: каждое вхождение возвращается в виде массива, в нулевом элементе которого содержится найденная подстрока, а в первом - смещение.

В случае, если никакой флаг не используется, по умолчанию используется PREG_PATTERN_ORDER.

Поиск осуществляется слева направо, с начала строки. Дополнительный параметр offset может быть использован для указания альтернативной начальной позиции для поиска. Аналогичного результата можно достичь, заменив subject на substr()($subject, $offset).

Возвращает количество найденных вхождений шаблона (может быть нулем) либо FALSE, если во время выполнения возникли какие-либо ошибки.

Примеркода

```
<?php
preg_match_all("/\(?  (\d{3})?  \)?  (?(1)  [\-\s] ) \d{3}-\d{4}/x",
"Call 555-1212 or 1-800-555-1212", $phones);
?>
```

Пример кода

```
<?php
// Запись \\2 является примером использования ссылок на подмаски.
```

```
// Она означает необходимость соответствия подстроки строке, зафиксированной
// второй подмаской, в нашем примере это ([\w]+).
// Дополнительный слеш необходим, так как используются двойные кавычки.
$html = "<b>bold text</b><a href=howdy.html>click me</a>";
preg_match_all("/(<([\w]+)[^>]*>)(.*)(<\/\\2>)/", $html, $matches);
for ($i=0; $i< count($matches[0]); $i++) {
echo"matched: " . $matches[0][$i] . "\n";
echo"part 1: " . $matches[1][$i] . "\n";
echo"part 2: " . $matches[3][$i] . "\n";
echo"part 3: " . $matches[4][$i] . "\n\n";
}
?>
```

Результат работы примера:

```
matched: <b>boldtext</b>
part 1: <b>
part 2: bold text
part 3: </b>
matched: <ahref=howdy.html>click me</a>
part 1: <ahref=howdy.html>
part 2: click me
part 3: </a>
```

preg_quote

Функция **preg_quote** - Экранирует символы в регулярных выражениях

Синтаксис

string **preg_quote** (string str [, string delimiter])

Функция preg_quote() принимает строку str и добавляет обратный слеш перед каждым служебным символом. Это бывает полезно, если в

составлении шаблона участвуют строковые переменные, значение которых в процессе работы скрипта может меняться.

В случае, если дополнительный параметр delimiter указан, он будет также экранироваться. Это удобно для экранирования ограничителя, который используется в PCRE функциях. Наиболее распространенным ограничителем является символ '/'.

В регулярных выражениях служебными считаются следующие символы: .\\ + * ? [^] $ ( ) { } = ! <> | :

Примеркода

```
<?php
$keywords = "$40 for a g3/400";
$keywords = preg_quote($keywords, "/");
echo$keywords; // возвращает \$40 for a g3\/400
?>
```

Пример кода

```
<?php
// Выделение курсивом слова в тексте
// В данном примере preg_quote($word) используется, чтобы
// избежать трактовки символа '*' как спец. символа.
$textbody = "This book is *very* difficult to find.";
$word = "*very*";
$textbody = preg_replace ("/" . preg_quote($word) . "/",
"<i>" . $word . "</i>",
$textbody);
echo$textbody;
?>
```

Результатработыпримера:

```
This book is <i>*very*</i> difficult to find.
```

preg_replace

Функция **preg_replace** - Выполняет поиск и замену по регулярному выражению

Синтаксис

mixed **preg_replace** (mixed pattern, mixed replacement, mixed subject [, int limit])

Выполняет поиск в строке subject совпадений с шаблоном pattern и заменяет их на replacement. В случае, если параметр limit указан, будет произведена замена limit вхождений шаблона; в случае, если limit опущен либо равняется -1, будут заменены все вхождения шаблона.

Replacement может содержать ссылки вида \\n либо (начиная с PHP 4.0.4) $n, причем последний вариант предпочтительней. Каждая такая ссылка, будет заменена на подстроку, соответствующую n'нной заключенной в круглые скобки подмаске. n может принимать значения от 0 до 99, причем ссылка \\0 (либо $0) соответствует вхождению всего шаблона. Подмаски нумеруются слева направо, начиная с единицы.

При использовании замены по шаблону с использованием ссылок на подмаски может возникнуть ситуация, когда непосредственно за маской следует цифра. В таком случае нотация вида \\n приводит к ошибке: ссылка на первую подмаску, за которой следует цифра 1, запишется как \\11, что будет интерпретировано как ссылка на одиннадцатую подмаску. Это недоразумение можно устранить, если воспользоваться конструкцией \${1}1, указывающей на изолированную ссылку на первую подмаску, и следующую за ней цифру 1.

Примеркода

```
<?php
$string = "April 15, 2003";
$pattern = "/(\w+) (\d+), (\d+)/i";
$replacement = "\${1}1,\$3";
echo preg_replace($pattern, $replacement, $string);
```

```
?>
```

Результатом работы этого примера будет:

April1,2003.

Если во время выполнения функции были обнаружены совпадения с шаблоном, будет возвращено измененное значение subject, в противном случае будет возвращен исходный текст subject.

Первые три параметра функции preg_replace() могут быть одномерными массивами. В случае, если массив использует ключи, при обработке массива они будут взяты в том порядке, в котором они расположены в массиве. Указание ключей в массиве для pattern и replacement не является обязательным. Если вы все же решили использовать индексы, для сопоставления шаблонов и строк, участвующих в замене, используйте функцию ksort() для каждого из массивов.

```
<?php
// Использование массивов с числовыми индексами в качестве
аргументов
// функции preg_replace()
$string = "The quick brown fox jumped over the lazy dog.";
$patterns[0] = "/quick/";
$patterns[1] = "/brown/";
$patterns[2] = "/fox/";
$replacements[2] = "bear";
$replacements[1] = "black";
$replacements[0] = "slow";
echo preg_replace($patterns, $replacements, $string);
?>
```

The bear black slow jumped over the lazy dog.

Используя ksort(), получаем желаемый результат:

```
<?php
```

```
ksort($patterns);
ksort($replacements);
echo preg_replace($patterns, $replacements, $string);
?>
```

The slow black bear jumped over the lazy dog.

В случае, если параметр subject является массивом, поиск и замена по шаблону производятся для каждого из его элементов. Возвращаемый результат также будет массивом.

В случае, если параметры pattern и replacement являются массивами, preg_replace() поочередно извлекает из обоих массивов по паре элементов и использует их для операции поиска и замены. Если массив replacement содержит больше элементов, чем pattern, вместо недостающих элементов для замены будут взяты пустые строки. В случае, если pattern является массивом, а replacement - строкой, по каждому элементу массива pattern будет осущесвтлен поиск и замена на pattern (шаблоном будут поочередно все элементы массива, в то время как строка замены остается фиксированной). Вариант, когда pattern является строкой, а replacement - массивом, не имеет смысла.

Модификатор /e меняет поведение функции preg_replace() таким образом, что параметр replacement после выполнения необходимых подстановок интерпретируется как PHP-код и только после этого используется для замены. Используя данный модификатор, будьте внимательны: параметр replacement должен содержать корректный PHP-код, в противном случае в строке, содержащей вызов функции preg_replace(), возникнет ошибка синтаксиса.

Пример кода: Замена по нескольким шаблонам

```
<?php
// Замена по нескольким шаблонам
```

```
$patterns = array ("/(19|20)(\d{2})-(\d{1,2})-(\d{1,2})/",
"/^\s*{(\w+)}\s*=/");
$replace = array ("\\3/\\4/\\1\\2", "$\\1 =");
echo preg_replace($patterns, $replace, "{startDate} = \"1999-5-27\";");
?>
```

Этот пример выведет:

$startDate = "5/27/1999";

Пример кода: Использование модификатора /e

```
<?php
// Использование модификатора /e
preg_replace("/(<\/?)(\w+)([^>]*>)/e",
"'\\1'.strtoupper('\\2').'\\3'",
$html_body);
?>
```

Пример кода: Преобразует все HTML-теги к верхнему регистру

```
<?php
// Конвертор HTML в текст
// $document на выходе должен содержать HTML-документ.
// Необходимо удалить все HTML-теги, секции javascript,
// пробельные символы. Также необходимо заменить некоторые
// HTML-сущности на их эквивалент.
$search = array ("'<script[^>]*?>.*?</script>'si",  // Вырезает javaScript
"'<[\/\!]*?[^<>]*?>'si",           // Вырезает HTML-теги
"'([\r\n])[\s]+'",                 // Вырезает пробельные символы
"'&(quot|#34);'i",                 // Заменяет HTML-сущности
"'&(amp|#38);'i",
"'&(lt|#60);'i",
"'&(gt|#62);'i",
```

```
"'&(nbsp|#160);'i",
"'&(iexcl|#161);'i",
"'&(cent|#162);'i",
"'&(pound|#163);'i",
"'&(copy|#169);'i",
"'&#(\d+);'e");                    // интерпретировать как php-код
$replace = array ("",
"",
"\\1",
"\"",
"&",
"<",
">",
" ",
                chr(161),
                chr(162),
                chr(163),
                chr(169),
"chr(\\1)");
$text = preg_replace($search, $replace, $document);
?>
```

preg_replace_callback

Функция **preg_replace_callback** - Выполняет поиск по регулярному выражению и замену с использованием функции обратного вызова

Синтаксис

mixed **preg_replace_callback** (mixed pattern, callback callback, mixed subject [, int limit])

Поведение этой функции во многом напоминает preg_replace(), за исключением того, что вместо параметра replacement необходимо указывать

callback функцию, которой в качестве входящего параметра передается массив найденных вхождений. Ожидаемый результат - строка, которой будет произведена замена.

Пример кода

```
<?php
/* фильтр, подобный тому, что используется в системах Unix
 для преобразования в заглавные начальных букв параграфа */
$fp = fopen("php://stdin", "r") ordie("can't read stdin");
while (!feof($fp)) {
$line = fgets($fp);
$line = preg_replace_callback(
'|<p>\s*\w|',
create_function(
// Использование одиночных кавычек в данном случае принципиально,
// альтернатива - экранировать все символы '$'
'$matches',
'return strtolower($matches[0]);'
          ),
$line
       );
echo$line;
}
fclose($fp);
?>
```

preg_split

Функция **preg_split** - Разбивает строку по регулярному выражению

Синтаксис

array **preg_split** (string pattern, string subject [, int limit [, int flags]])

Возвращает массив, состоящий из подстрок заданной строки subject, которая разбита по границам, соответствующим шаблону pattern.

В случае, если параметр limit указан, функция возвращает не более, чем limit подстрок. Специальное значение limit, равное -1, подразумевает отсутствие ограничения, это весьма полезно для указания еще одного опционального параметра flags.

flags может быть произвольной комбинацией следующих флагов (соединение происходит при помощи оператора '|'):

PREG_SPLIT_NO_EMPTY

В случае, если этот флаг указан, функция preg_split() вернет только непустые подстроки.

PREG_SPLIT_DELIM_CAPTURE

В случае, если этот флаг указан, выражение, заключенное в круглые скобки в разделяющем шаблоне, также извлекается из заданной строки и возвращается функцией. Этот флаг был добавлен в PHP 4.0.5.

PREG_SPLIT_OFFSET_CAPTURE

В случае, если этот флаг указан, для каждой найденной подстроки, будет указана ее позиция в исходной строке. Необходимо помнить, что этот флаг меняет формат возвращаемых данных: каждое вхождение возвращается в виде массива, в нулевом элементе которого содержится найденная подстрока, а в первом - смещение.

Примеры кода

```
<?php
// Получение подстрок из заданного текста
// разбиваем строку по произвольному числу запятых и пробельных символов,
// которые включают в себя  " ", \r, \t, \n и \f
$keywords = preg_split("/[\s,]+/", "hypertext language, programming");
?>
```

```
<?php
// Разбиваем строку на составляющие символы
$str = 'string';
$chars = preg_split('//', $str, -1, PREG_SPLIT_NO_EMPTY);
print_r($chars);
?>
<?php
$str = 'hypertext language programming';
$chars = preg_split('/ /', $str, -1, PREG_SPLIT_OFFSET_CAPTURE);
print_r($chars);
?>
```

В случае, если после открывающей круглой скобки следует "**?:**", захват строки не происходит, и текущая подмаска не нумеруется. Например, если строка "thewhitequeen" сопоставляется с шаблоном the ((?:red|white) (king|queen)), будут захвачены подстроки "whitequeen" и "queen", и они будут пронумерованы 1 и 2 соответственно:

```
$r="/the ((?:red|white) (king|queen))/";
0="the white king";
1="white king";
2="king";
```

Связь с базами данных MySQL

СУБД **MySQL** - одна из множества баз данных, поддерживаемых в PHP. Система MySQL распространяется бесплатно и обладает достаточной мощностью для решения реальных задач.

Краткое введение в MySQL

SQL - это аббревиатура от слов *Structured Query Language*, что означает структурированный язык запросов. Этот язык является стандартным средством для доступа к различным базам данных.

Система MySQL представляет собой сервер, к которому могут подключаться пользователи удаленных компьютеров.

Для работы с базами данных удобно пользоваться средством, входящее в комплект Web-разработчика: Denwer phpMyAdmin. Здесь можно создать новую базу данных, создать новую таблицу в выбранной базе данных, заполнить таблицу данными, а также добавлять, удалять и редактировать данные.

В MySQL определены три базовых типа данных: числовой, дата и время и строчный. Каждая из этих категорий подразделяется на множество типов. Основные из них:

Тип	Описание
INT	Целое число
TINYINT	Маленькое целое число (-127 до 128 или от 0 до 255)
FLOAT	Вещественное число с плавающей точкой
DATE	Дата. Отображается в виде ГГГГ-ММ-ДД
TIME	Время. Отображается в виде ЧЧ:ММ:СС
DATETIME	Дата и время. Отображается в виде ГГГГ-ММ-ДДЧЧ:ММ:СС
YEAR[(2|4)]	Год. Можно определить двух- или четырехциферный формат
CHAR(M)	Строка фиксированной длины М (М<=255)
VARCHAR(M)	Строка произвольной длины до М (М<=255)
TEXT	Длинные текстовые фрагменты (<=65535)
BLOB	Большие двоичные объекты (изображения, звуки)

Каждый столбец после своего типа данных содержит и другие спецификаторы:

Тип	Описание
NOT NULL	Все строки таблицы должны иметь значение в этом атрибуте. Если не указано, поле может быть пустым (**NULL**)
AUTO_INCREMENT	Специальная возможность MySQL, которую можно задействовать в числовых столбцах. Если при вставке строк в таблицу оставлять такое поле пустым, MySQL автоматически генерирует уникальное значение идентификатора. Это значение будет на единицу больше максимального значения, уже существующего в столбце. В каждой таблице может быть не больше одного такого поля. Столбцы с **AUTO_INCREMENT** должны быть проиндексированными
PRIMARY KEY	Столбец является первичным ключом для таблицы. Данные в этом столбце должны быть уникальными. MySQL автоматически индексирует этот столбец
UNSIGNED	После целочисленного типа означает, что его значение может быть либо положительным, либо нулевым
COMMENT	Название столбца таблицы

Создание новой базы данных MySQL осуществляется при помощи SQL-команды **CREATE DATABASE**.

```
CREATE DATABASE IF NOT EXISTS `base`
  DEFAULT CHARACTER SET cp1251 COLLATE cp1251_bin
```

Создание новой таблицы осуществляется при помощи SQL-команды **CREATE TABLE**. Например, таблица **books** для книжного магазина будет

содержать пять полей: ISBN, автор, название, цена и количество экземпляров:

```
CREATE TABLE books (ISBN CHAR(13) NOT NULL,
                PRIMARY KEY (ISBN),
                author VARCHAR(30),
                title VARCHAR(60),
                price FLOAT(4,2),
                quantity TINYINT UNSIGNED);
```

Чтобы избежать сообщения об ошибке, если таблица уже есть необходимо изменить первую строчку, добавив фразу "IF NOT EXISTS":

```
CREATE TABLE IF NOT EXISTS books ...
```

Для создания **автообновляемого поля** с текущей датой типа TIMESTAMP или DATETIME используйте следующую конструкцию:

```
CREATE TABLE t1 (
ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
dt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
);
```

Добавление данных в эту таблицу осуществляется при помощи SQL-команды **INSERT**. Например:

```
INSERT INTO books ( ISBN, author, title, price, quantity )
        VALUES ('5-8459-0184-7', 'ЗандстраМэт',
'Освой самостоятельно PHP4 за 24 часа', '129', '5');
```

Для извлечения данных из таблицы служит оператор **SELECT**. Он извлекает данные из базы, выбирая строки, которые отвечают заданному

критерию поиска. Оператор **SELECT** сопровождает немалое количество опций и вариантов использования.

Символ * означает, что необходимы все поля. Например:

```
SELECT * FROM books;
```

Для получения доступа только к некоторому полю следует указать его имя в инструкции **SELECT**. Например:

```
SELECT author, title, price FROM books;
```

Чтобы получить доступ к подмножеству строк в таблице, следует указать критерий выбора, который устанавливает конструкция **WHERE**. Например, чтобы выбрать имеющиеся в наличии недорогие книги о PHP, надо составить запрос:

```
SELECT * FROM books WHERE
  price< 200 AND title LIKE '%PHP%' AND quantity != 0;
```

% Соответствует любому количеству символов, даже нулевых

_ Соответствует ровно одному символу

Для того, чтобы строки, извлеченные по запросу, перечислялись в определенном порядке, используется конструкция **ORDER BY**. Например:

```
SELECT * FROM books ORDER BY price;
```

По умолчанию **порядок сортировки** идет по возрастанию. Изменить порядок сортировки на обратный можно с помощью ключевого слова **DESC**:

```
SELECT * FROM books ORDER BY price DESC;
```

Сортировать можно и по нескольким столбцам. Вместо названий столбцов можно использовать их порядковые номера:

```
SELECT * FROM books ORDER BY 4, 2, 3;
```

Для изменения ранее записанных в таблицу значений нужно воспользоваться командой **UPDATE**. Например, цену всех книг повысили на 10%:

```
UPDATE books SET price = price * 1.1;
```

Конструкция **WHERE** ограничит работу **UPDATE** определенным строками. Например:

```
UPDATE books SET price = price * 1.05 WHERE price <= 250;
```

Для удаления строк из базы данных используется оператор **DELETE**. Ненужные строки указываются при помощи конструкции **WHERE**. Например, какие-то книги проданы:

```
DELETE FROM books WHERE quantity = 0;
```

Если нужно удалить все записи

```
TRUNCATE TABLE table_name
```

Для полного удаления таблицы используется:

```
DROP TABLE table_name
```

Связь PHP сбазойданных MySQL

Поработав с phpMyAdmin над созданием базы данных, можно приступить к подключению этой базы данных к внешнему Web-интерфейсу.

Чтобы получить доступ к базе данных из Web, используя PHP, надо сделать следующие основные шаги:

- Подключение к серверу MySQL.
- Выбор базы данных.
- Выполнение запроса к базе данных:
  - **добавление**;
  - **удаление**;
  - **изменение**;
  - **поиск**;
  - **сортировка**.
- Получение результата запроса.
- Отсоединение от базы данных.

Для подключения к серверу базы данных в PHP есть функция **mysql_connect( )**. Ее аргументы: имя компьютера, имя пользователя и пароль. Эти аргументы можно опустить. По умолчанию имя компьютера = **localhost**, тогда имя пользователя и пароль не требуется. Если PHP используется в сочетании с сервером Apache, то можно воспользоваться функцией **mysql_pconnect()**. В этом случае соединение с сервером не исчезает после завершения работы программы или вызова функции **mysql_close()**. Функции **mysql_connect()** и **mysql_pconnect()** возвращают идентификатор подключения, если все прошло успешно. Например:

```
$link = mysql_pconnect ();
if ( !$link ) die ("Невозможно подключение к MySQL");
```

После того, как соединение с сервером MySQL установлено, нужно выбрать базу данных. Для этого используется функция **mysql_select_db()**. Ее аргумент: имя базы данных. Функция возвращает **true**, если указанная база данных существует и доступ к ней возможен. Например:

```
$db = "sample";
mysql_select_db ( $db ) or die ("Невозможно открыть $db");
```

Для добавления, удаления, изменения и выбора данных нужно сконструировать и выполнить запрос SQL. Для этого в языке PHP существует функция **mysql_query()**. Ее аргумент: строка с запросом. Функция возвращает идентификатор запроса.

Пример 1

```
<html>
<head>
<title>Добавление записи в таблицу</title>
</head>
<body>
<?php
$db = "sample";
```

```
$link = mysql_pconnect ();
if ( !$link )
die ("Невозможно подключение к MySQL");
mysql_select_db ( $db ) or die ("Невозможнооткрыть $db");
$query = "INSERT INTO books
        VALUES ('966-7393-80-1', 'АлленВайк',
        'PHP. Справочник', '213', '4')";
mysql_query ( $query );
mysql_close ( $link );
?>
</body>
</html>
```

При каждом выполнении примера 1 в таблицу будет добавляться новая запись, содержащая одни и те же данные. Разумеется имеет смысл добавлять в базу данные, введенные пользователем.

В примере 2.1 приведена HTML-форма для добавления новых книг в базу данных.

Пример 2.1

```
<html>
<head>
<title>HTML-форма добавления новых книг</title>
</head>
<body>
<formaction="insert_book.php"method="post">
<table>
<tr><td>ISBN</td><td><inputname="isbn"maxlength=13size=13></td></t
r>
<tr><td>Автор</td><td><inputname="author"maxlength=30size=30></td>
</tr>
```

```
<tr><td>Название</td><td><inputname="title"maxlength=60size=30></td></tr>
<tr><td>Цена</td><td><inputname="price"maxlength=7size=7></td></tr>
<tr><td>Количество</td><td><inputname="quantity"maxlength=3size=3></td></tr>
<tr><tdcolspan=2><inputtype="submit"value="Ввод"></td></tr>
</table>
</form>
</body>
</html>
```

Результаты заполнения этой формы передаются в insert_book.php.

Пример 2.2

```
<html>
<head>
<title>Программа добавления новых книг (файл insert_book.php)</title>
</head>
<body>
<?php
if (!isset($_POST['isbn']) || !isset($_POST['author']) ||
    !isset($_POST['title']) || !isset($_POST['price']) ||
!isset($_POST['quantity'])){
die ("Не все данные введены.<br>
                Пожалуйста, вернитесь назад и закончите ввод");
}
$isbn   = trim ( $_POST['isbn'] );
$author = trim ( $_POST['author'] );
$title  = trim ( $_POST['title'] ) ;
$isbn   = addslashes ( $isbn );
$author = addslashes ( $author );
```

```
$title  = addslashes ( $title ) ;
$db = "sample";
$link = mysql_connect();
if ( !$link ) die ("Невозможно подключение к MySQL");
mysql_select_db ( $db ) ordie ("Невозможнооткрыть $db");
$query = "INSERT INTO books VALUES ('"
    .$isbn."', '".$author."', '".$title."', '"
    .floatval($_POST['price'])."', '".intval($_POST['quantity'])."')";
$result = mysql_query ( $query );
if ($result) echo"Книгадобавленавбазуданных.";
mysql_close ( $link );
?>
</body>
</html>
```

В примере 2.2 введенные строковые данные обработаны функцией **addslashes()**. Эта функция добавляет обратные слеши перед одинарными кавычками ('), двойными кавычками ("), обратным слешем (\) и null-байтом. Дело в том, что по требованиям систаксиса запросов баз данных такие символы дожны заключаться в кавычки.

Для определения количества записей в результате запроса используется функция **mysql_num_rows()**.

Все записи результата запроса можно просмотреть в цикле. Перед этим с помощью функции **mysql_fetch_[]** для каждой записи получают ассоциативный массив.

В примере 3.1 приведена HTML-форма для поиска определенных книг в базе данных.

Пример 3.1

```
<html>
<head>
```

```
<title>HTML-формапоисchildкниг</title>
</head>
<body>
<formaction="search_book.php"method="post">
Ищемпо:<br>
<selectname="searchtype"size=3>
<optionvalue="author"selected>Автору
<optionvalue="title">Названию
<optionvalue="isbn">ISBN
</select><br>
Чтоищем:<br><inputname="searchterm"><br>
<inputtype=submitvalue="Поиск">
</form>
</body>
</html>
```

Результаты заполнения этой формы передаются в search_book.php.

Пример 3.2

```
<html>
<head>
<title>Программа поиска книг (файл search_book.php)</title>
</head>
<body>
<?php
$searchterm = trim ( $_POST['searchterm'] );
if (!$searchterm)
die ("Не все данные введены.<br>
   Пожалуйста, вернитесь назад и закончите ввод");
$searchterm = addslashes ($searchterm);
$link = mysql_pconnect ();
```

```
if ( !$link ) die ("Невозможно подключение к MySQL");
$db = "sample";
mysql_select_db ( $db ) ordie ("Невозможнооткрыть $db");
$query = "SELECT * FROM books WHERE "
   .$_POST['searchtype']." like '%".$searchterm."%'";
$result = mysql_query ( $query );
$n = mysql_num_rows ( $result );
for ( $i=0; $i<$n; $i++ )
{
$row = mysql_fetch_array($result);
echo"<p><b>".($i+1). $row['title']. "</b><br>";
echo"Автор: ".$row['author']."<br>";
echo"ISBN: ".$row['ISBN']."<br>";
echo"Цена: ".$row['price']."<br>";
echo"Количество: ".$row['quantity']."</p>";
}
if( $n == 0 ) echo"Ничего не можем предложить. Извините";
mysql_close ( $link );
?>
</body>
</html>
```

Альтернативный вариант

```
<html>
<head>
<title>Программа поиска книг (файл search_book.php)</title>
</head>
<body>
<?
$searchterm=trim ( $_POST['searchterm'] );
```

```
if (!$searchterm)
die ("Не все данные введены.<br>Пожалуйста, вернитесь назад и
закончите ввод");
$searchterm = addslashes ($searchterm);
mysql_connect() ordie ("Невозможноподключениек MySQL");
mysql_select_db ( "sample" ) ordie ("НевозможнооткрытьБД");
$result = mysql_query ( "SELECT * FROM books WHERE
".$_POST['searchtype']." like '%".$searchterm."%'" );
$i=1;
while($row = mysql_fetch_array($result))
{
echo"<p><b>".($i++) . $row['title']."</b><br>";
echo"Автор: ".$row['author']."<br>";
echo"ISBN: ".$row['ISBN']."<br>";
echo"Цена: ".$row['price']."<br>";
echo"Количество: ".$row['quantity']."</p>";
}
if( $i == 1 ) echo"Ничего не можем предложить. Извините";
mysql_close( );
?>
</body>
</html>
```

Итак, как работает архитектура Web-баз данных:

1. Web-браузер пользователя выдает HTTP-запрос определенной Web-страницы. Например, пользователь, используя HTML-форму, ищет все книги о PHP. Страница обработки формы называется search_book.php.

2. Web-сервер принимает запрос на search_book.php, извлекает этот файл и передает на обработку механизму PHP.

3. PHP выполняет соединение с MySQL-сервером и отправляет запрос.

4. Сервер принимает запрос к базе данных, обрабатывает его и отправляет результат (список книг) обратно механизму PHP.

5. Механизм PHP завершает выполнение сценария, форматирует результат запроса в HTML. После этого результат в виде HTML возвращается Web-серверу.

6. Web-сервер пересылает HTML в браузер, и пользователь имеет возможность просмотреть запрошенный список книг.

Использование механизма транзакций

Использование механизма транзакция на примере как передать деньги от одного человека другому

```
if(
    mysql_query ("BEGIN") &&
    mysql_query ("UPDATE money SET amt = amt - 6 WHERE name = 'Eve'") &&
    mysql_query ("UPDATE money SET amt = amt + 6 WHERE name = 'Ida'") &&
    mysql_query ("COMMIT")
){
echo"Успешно";
}else{
    mysql_query ("ROLLBACK");
echo"Неуспешно";
}
```

SELECT … FOR UPDATE

Если Вы запускаете несколько процессов, которые делают select запрос к одной и той же таблице, то они могут выбрать одну и ту же запись одновременно.

Чтобы избежать вышеупомянутой ситуации необходимо выполнить не просто SELECT запрос, а его расширенную версию, о которой многие и не подозревают: SELECT … FOR UPDATE.

Таким образом, при выполнении данного запроса, все затронутые записи в базе данных будут заблокированы до завершения сеанса работы с БД или до момента обновления данных записей. Другой скрипт не сможет выбрать заблокированные записи до тех пор, пока не наступит одно из упомянутых условий.

Вам нужно выполнить ещё несколько условий. Во-первых, ваша таблица должна быть создана на основе архитектуры InnoDB. В противном случае блокировка просто не будет срабатывать. Во-вторых, перед выполнением выборки необходимо отключить авто-коммит запроса. Т.е. другими словами автоматическое выполнение запроса. После того как вы укажите UPDATE запрос, необходимо будет ещё раз обратиться к базе и закоммитить изменения с помощью команды COMMIT:

```
mysql_query("SET autocommit = 0");
$result = mysql_query("SELECT * FROM table WHERE locked = 0 LIMIT 1 FOR UPDATE");
$row = mysql_fetch_assoc($result);
mysql_query("UPDATE table SET locked = 1 WHERE id = 1;");
mysql_query("COMMIT;");
```

Работа с формами

Для передачи данных от пользователя Web-страницы на сервер используются HTML-формы. Для работы с формами в PHP предусмотрен ряд специальных средств.

Предварительно определенные переменные

В PHP существует ряд предварительно определенных переменных, которые не меняются при выполнении всех приложений в конкретной среде. Их также называют переменными окружения или переменными среды. Они

отражают установки среды Web-сервера Apache, а также информацию о запросе данного браузера. Есть возможность получить значения URL, строки запроса и других элементов HTTP-запроса.

Все предварительно определенные переменные содержатся в ассоциативном массиве $GLOBALS. Кроме переменных окружения этот массив содержит также глобальные переменные, определенные в программе.

Пример 1

```
<html>
<head>
<title>Просмотрмассива $GLOBALS</title>
</head>
<body>
<?php
  $a = "Hello!";
foreach ( $GLOBALS as $key=>$value )
echo "\$GLOBALS[\"$key\"] == $value<br>";
?>
</body>
</html>
```

В результате на экране появится список всех глобальных переменных, включая переменные окружения. Наиболее часто используемые из них:

Переменная	Описание	Содержание
$_SERVER['HTTP_USER_AGENT']	Название и версия клиента	Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36
$_SERVER['REMOTE_A	IP-адрес	89.146.89.40

DDR']		
getenv('HTTP_X_FORWARDED_FOR')	Внутренний IP-адрес клиента	
$_SERVER['REQUEST_METHOD']	Метод запроса (GET или POST)	GET
$_SERVER['QUERY_STRING']	При запросе GETзакодированные данные, передаваемые вместе с URL	
$_SERVER['REQUEST_URL']	Полный адрес клиента, включая строку запроса	
$_SERVER['HTTP_REFERER']	Адрес страницы, с которой был сделан запрос	https://htmlweb.ru/php/php6.php
$_SERVER['PHP_SELF']	Путь к выполняемой программе	/index.php
$_SERVER['SERVER_NAME']	Домен	htmlweb.ru
$_SERVER['REQUEST_URI']	Путь	/php/php_form.php

Обработка ввода пользователя

PHP-программу обработки ввода можно отделить от HTML-текста, содержащего формы ввода, а можно расположить на одной странице.

Пример 2

```
<?php
if ( ! isset( $cardnumber ) )
$m = "Введите номер карточки";
elseif( $cardnumber == "" )
  $m = "Ваш номер пуст!";
else
```

```
  $m = "Ваш номер: $cardnumber";
?>
<html>
<head>
<title>Примеробработкиввода</title>
</head>
<body>
<h1><?php echo $m?></h1>
<formname="form1"action="<?=$_SERVER['PHP_SELF']?>"method="post">
<p>Номеркарточки:
<inputname="cardnumber"type="text"size="20">
</form>
</body>
</html>
```

Здесь отсутствует кнопка передачи данных, т.к. форма, состоящая из одного поля, передается автоматически при нажатии клавиши <Enter>.

При обработки элемента с многозначным выбором для доступа ко всем выбранным значениям нужно к имени элемента добавить пару квадратных скобок. Для выбора нескольких эллементов следует удерживать клавишу Ctrl.

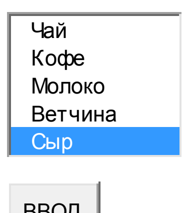
Пример 3.1

Файл ex.htm

```
<html>
<head>
<title>Список</title>
</head>
<body>
<formname="form1"action="ex1.php"method="post">
<selectname="Item[]"size=5multiple>
```

```
<option>Чай
<option>Кофе
<option>Молоко
<option>Ветчина
<option>Сыр
</select>
<inputtype="submit"value="ВВОД">
</form>
</body>
</html>
```

РЕЗУЛЬТАТ ПРИМЕРА 3.1:



Пример 3.2

Файл ex.php

```
<html>
<head>
<title>Обработка списка из файла ex1.htm</title>
</head>
<body>
<?php
echo "Вашзаказ: <p><ul>";
foreach ( $Item as $value )   echo "<li>$value";
echo "</ul>";
?>
</body>
```

```
</html>
```

Пример 4. Прием значений от checkbox-флажков

```
<?php
if (@$_REQUEST['doGo']) {
foreach (@$_REQUEST['known'] as $k=>$v) {
if($v) echo "Вы знаете язык программирования $k!<br>";
elseecho "Вы не знаете языка программирования $k. <br>";
}
}
?>
<formaction="<?=$_SERVER['SCRIPT_NAME']?>"method="post">
Какие языки программирования вы знаете?<br>
<inputtype=hiddenname="known[PHP]"value="0">
<inputtype=checkboxname="known[PHP]"value="1"> PHP<br>
<inputtype=hiddenname="known[Perl]"value="0">
<inputtype=checkboxname="known[Perl]"value="1"> Perl<br>
<inputtype=submitname="doGo"value="Go!">
</form>
```

РЕЗУЛЬТАТ ПРИМЕРА 4:

Какие языки программирования вы знаете?

☐ PHP

☐ Perl

Go!

Пример 5

```
<?php // Модель скрипта, принимающего текст от пользователя.
if (@$_REQUEST['text'])
echohtmlspecialchars($_REQUEST['text'])."<hr>";
```

```
?>
<formaction="<?=$_SERVER['SCRIPT_NAME']?>"method="post">
<textareaname="text"cols="60"rows="10">
< ?=@htmlspecialchars($_REQUEST['text'])?>
</textarea><br>
<inputtype="submit">
</form>
```

Можно обрабатывать формы, не заботясь о фактических именах полей.

Для этого можно использовать (в зависимости от метода передачи) ассоциативный массив $HTTP_GET_VARSили $HTTP_POST_VARS. Эти массивы содержат пары имя/значение для каждого элемента переданной формы. Если Вам все равно, Вы можете использовать ассоциативный массив $_REQUEST.

Пример 6

```
<html>
<head>
<title>Обработка произвольного ввода
      независимо от метода передачи</title>
</head>
<body>
<?php
$params = ( $_SERVER['REQUEST_METHOD'] == "GET" )
  ? $HTTP_GET_VARS : $HTTP_POST_VARS;
foreach ( $params as $key=>$value )
echo "$key == $value<br>";
?>
</body>
</html>
```

Пример 7. Обработка нажатия на кнопку с использованием оператора '@'.

```
<?php
if (@$_REQUEST['submit']) echo "Кнопканажата!"
?>
<formaction="<?=$_SERVER['SCRIPT_NAME']?>">
<inputtype="submit"name="submit"value="Go!">
</form>
```

С помощью функции header(), послав браузеру заголовок "Location", можно перенаправить пользователя на новую страницу.

Например:

```
<? header("Location: ex2.php"); ?>
```

Передача файла на сервер. Залить файл. UpLoad

PHP позволяет передавать на сервер файлы. HTML-форма, предназначенная для передачи файла, должна содержать аргумент enctype="multipart/form-data".

Кроме того в форме перед полем для копирования файла должно находиться скрытое поле с именем max_file_size. В это скрытое поле должен быть записан максимальный размер передаваемого файла.

Само поле для передачи файла - обычный элемент INPUT с аргументом type="file".

Например:

```
<formenctype="multipart/form-data"
   action="<?echo $_SERVER['PHP_SELF']?>"method="post">
<inputtype="hidden"name="max_file_size"value="51200">
<inputtype="file"name="myfile"><br>
<inputtype="submit"value="ВВОД">
</form>
```

После того, как файл передан на сервер, он получает уникальное имя и сохраняется в каталоге для временных файлов. Полный путь к файлу записывается в глобальную переменную, имя которой совпадает с именем поля для передачи этого файла. Кроме этого PHP сохраняет еще некоторую дополнительную информацию о переданном файле в других глобальных переменных:

Переменная	Описание
$_FILES['userfile']['name']	оригинальное имя файла, такое, каким его видел пользователь, выбирая файл
$_FILES['userfile']['type']	mime/type файла, к примеру, может быть image/gif; это поле полезно сохранить, если Вы хотите предоставлять интерфейс для скачивания загруженных файлов
$_FILES['userfile']['size']	размер загруженного файла
$_FILES['userfile']['tmp_name']	полный путь к временному файлу на диске
$_FILES['userfile']['error']	код ошибки, который равен 0, если операция прошла успешно

Пример 8

```
<html>
<head>
<title>Обработка переданного файла</title>
</head>
<?php
if ( isset( $_FILES['userfile']['tmp_name'] ) )
  {
echo "путь: ".$_FILES['userfile']['tmp_name']."<br>";
echo "имя: ".$_FILES['userfile']['name']."<br>";
echo "размер: ".$_FILES['userfile']['size']."<br>";
echo "тип: ".$_FILES['userfile']['type']."<br>";
```

```
  }
?>
<body>
<formenctype="multipart/form-data"
   action="<?echo $_SERVER['PHP_SELF']?>"method="post">
<inputtype="hidden"name="max_file_size"value="51200">
<inputtype="file"name="userfile"><br>
<inputtype="submit"value="ВВОД">
</form>
</body>
</html>
```

Примеры загрузки файлов на сервер

Если возникнут проблеммы с перекодировкой сервером загруженного файла, символ с кодом **0x00** заменен на пробел (символ с кодом **0x20**), допишите в файл **httpd.conf** из каталога Апача (/usr/local/apache) следующие строки.

```
<Location>
CharsetRecodeMultipartFormsOff
</Location>
```

ЛИТЕРАТУРА

1. Sh.Mirziyoyev “Erkin va farovon, demokratik O‘zbekiston davlatini birgalikda barpo etamiz”.Tosh.O ‘zbekiston-2016.

2. O’zbekiston Respublikasining “Ta’lim to’g’risida”gi qonuni// Barkamol avlod O’zbekiston taraqqiyotining poydevori.-T.: “Sharq” nashriyot-matbaa konserni, 1997. 22-32b.

3. O’zbekiston Respublikasi “Kadrlar tayyorlash Milliy dasturi”//Barkamol avlod O’zbekiston taraqqiyotining poydevori.-T.: “Sharq” nashriyot-matbaa kontserni, 1997.-V.31-61.

4. O'zbekiston Respublikasi Prezidentining 2017-yil 7-fevraldagi PF-4947 sonli “O'zbekiston Respublikasini yanada rivojlantirishning Harakatlar strategiyasi to’g’risida”gi farmoni// Xalq so’zi gazetasi. 2017 yil 8-fevral,№ 28 .

5. O’zbekiston Respublikasi Vazirlar Mahkamasining 2002 yil 6 iyundagi 200-sonli “Kompyuterlashtirishni yanada rivojlantirish va axborot kommunikatsiya texnologiyalarini joriy etish chor-tadbirlari to’g’risida”gi qarori.

6. Макфарланд Д., Большая книга CSS3. 3-е изд. — СПб.: Питер, 2014. — 608 с.

7. Goodman D., Morrison M., JavaScript Bible. 12th Edition — Published by Wiley Publishing, Inc., Indianapolis, Indiana, 2014. — 1184 p.

8. Колисниченко, Д. Н. PHP 5/6 и MySQL 6. Разработка Web-приложений / Д.Н. Колисниченко. - М.: БХВ-Петербург, 2011. - 528 с

9. Колисниченко, Денис PHP и MySQL. Разработка веб-приложений / Денис Колисниченко. - М.: БХВ-Петербург, 2015. - 592 с.

10. Машнин, Тимур Google App Engine Java и Google Web Toolkit. Разработка Web-приложений / Тимур Машнин. - М.: БХВ-Петербург, 2014. - 352 с.

12. . Машнин, Тимур JavaFX 2.0. Разработка RIA-приложений / Тимур Машнин. - Москва: Высшая школа, 2012. - 320 с.

13. Мэтью, Дэвид HTML5. Разработка веб-приложений / Дэвид Мэтью. - М.: Рид Групп, 2012. - 320 с.

14. Форсье, Дж. Django. Разработка веб-приложений на Python / Дж. Форсье. - М.: Символ-плюс, 2014. - 607 с.

15. Бенкен, Е.С. PHP, MySQL, XML. Программирование для Интернета (+ CD-ROM) / Е.С. Бенкен. - М.: БХВ-Петербург, 2011. - 250 с.

16. Веллинг, Л. Разработка веб-приложений с помощью PHP и MySQL / Л. Веллинг. - М.: Диалектика / Вильямс, 2016. - 116 с.

17. Гизберт, Дамашке PHP и MySQL / Дамашке Гизберт. - М.: НТ Пресс, 2011. - 663 с.

18. Дунаев, Вадим HTML, скрипты и стили Уцененный товар (№1) / Вадим Дунаев. - М.: БХВ-Петербург, 2015. - 816 с.

19. Жадаев, Александр PHP для начинающих / Александр Жадаев. - М.: Питер, 2014. - 251 с.

20. Клименко, Роман Веб-мастеринг на 100% / Роман Клименко. - М.: Питер, 2013. - 512 с.

21. Колисниченко, Денис PHP и MySQL. Разработка Web-приложений / Денис Колисниченко. - М.: БХВ-Петербург, 2013. - 560 с.

22. Кристиан, Уэнц PHP и MySQL. Карманный справочник / Уэнц Кристиан. - М.: Диалектика / Вильямс, 2016. - 656 с.

23. Кузнецов, Максим Самоучитель PHP 5/6 / Максим Кузнецов , Игорь Симдянов. - М.: БХВ-Петербург, 2009. - 672 с.

24. Локхарт, Джош Современный PHP. Новые возможности и передовой опыт / Джош Локхарт. - М.: ДМК Пресс, 2016. - 304 с.

25. Ляпин, Д.А. PHP — это просто. Начинаем с видеоуроков (+ CD-ROM) / Д.А. Ляпин. - М.: БХВ-Петербург, 2013. - 722 с.

26. Макаров, Александр Yii. Сборник рецептов / Александр Макаров. - М.: ДМК Пресс, 2015. - 372 с.

27. Маклафлин, Б. PHP и MySQL. Исчерпывающее руководство / Б. Маклафлин. - М.: Питер, 2013. - 512 с.

28. Никсон, Робин Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript и CSS / Робин Никсон. - М.: Питер, 2013. - 151 с.

29. Прохоренок, Николай HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера (+ CD-ROM) / Николай Прохоренок. - М.: БХВ-Петербург, 2010. - 912 с.

30. . Ховард, Майкл Как написать безопасный код на C++, Java, Perl, PHP, ASP.NET / Майкл Ховард , Дэвид Лебланк , Джон Виега. - М.: ДМК Пресс, 2014. - 288 с.

31. Бенкен, Е. AJAX. Программирование для Интернета / Е. Бенкен. - М.: БХВ-Петербург, 2012. - 652 с.

32. Вроблевски, Люк Сначала мобильные! / Люк Вроблевски. - М.: Манн, Иванов и Фербер, 2012. - 300 с.

33. Дунаев, Вадим Сценарии для Web-сайта. PHP и JavaScript / Вадим Дунаев. - М.: "БХВ-Петербург", 2012. - 576 с.

34. Книга веб-программиста. Секреты профессиональной разработки веб-сайтов / Б. Хоган и др. - Москва: Мир, 2013. - 288 с.

35. Костин, С. П. Самоучитель создания Web-сайтов / С.П. Костин. - М.: Триумф, 2015. - 176 с.

36. Литвин, Евгений Прибыльный блог. Создай, раскрути и заработай / Евгений Литвин. - М.: Питер, 2011. - 272 с.

37. Марк, Дэйв iOS 6 SDK. Разработка приложений для iPhone, iPad и iPod touch / Дэйв Марк и др. - Москва: СПб. [и др.] : Питер, 2013. - 672 с

38. Мартинес, А. Секреты создания недорогого Web-сайта / А. Мартинес. - М.: Книга по Требованию, 2012. - 405 с.

39. Машнин, Т. С. Eclipse: разработка RCP-, Web-, Ajax- и Android - приложений на Java / Т.С. Машнин. - М.: БХВ-Петербург, 2013. - 384 с.

40. Монтейро, Майк Дизайн – это работа / Майк Монтейро. - М.: Манн, Иванов и Фербер, 2012. - 536 с.

41. Мэтью, Дэвид HTML5. Разработка веб-приложений / Дэвид Мэтью. - М.: Рид Групп, 2012. - 320 с.

42. Нолан, Хестер Как создать превосходный сайт в Microsoft Expression Web 2 и CSS / Хестер Нолан. - М.: ДМК Пресс, 2011. - 118 с.

"Разработка Web приложений"

Учебное пособие

Рассмотрена и рекомендована
к изданию на заседании
научно-методического
совета ТУИТ
от «___» _________ 20___
протокол №____

Составители: Керимов К.Ф., Камалов Ш.К., Мухсинов Ш.Ш., Сиддикова Н.П.

Формат 60х84 1/16

Заказ №____. Тираж –____

Отпечатано в Издательство полиграфическом центре «ALOQACHI» при ТУИТ

Ташкент ул. Амир Темура , 108