Елена Бенкен

Геннадий Самков

AJAX
ПРОГРАММИРОВАНИЕ ДЛЯ ИНТЕРНЕТА

Санкт-Петербург

«БХВ-Петербург»

2009

УДК 681.3.06
ББК 32.973.26-018.2
Б46

Бенкен, Е. С.

Б46 AJAX: программирование для Интернета / Е. С. Бенкен, Г. А. Самков. — СПб.: БХВ-Петербург, 2009. — 464 с.: ил. + (CD-ROM)

ISBN 978-5-9775-0428-7

Описана технология AJAX и показаны возможности, которые открываются перед разработчиком с ее применением. Рассмотрена объектная модель документа: DOM в JavaSript и DOM-функции в PHP. Изложены основы языка XML и формат JSON. Показан принцип генерации асинхронных запросов к серверу средствами JavaScript. Сделан обзор основных JavaScript-библиотек: Prototype, Scriptaculous, ExtJS и jQuery. Подробно рассмотрены популярные и перспективные библиотеки ExtJS и jQuery: описана объектная модель языка JavaScript, на которой базируются эти библиотеки; применение AJAX-запросов; обработка событий и др. Приведено большое количество практических примеров. Компакт-диск содержит дистрибутивы Web-сервера, модуля PHP и сервера MySQL, исходные коды описываемых библиотек, распространяемых на основании лицензии GPL, а также примеры из книги.

Для Web-программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн обложки	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 04.03.09.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 37,41.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.003650.04.08 от 14.04.2008 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0428-7

© Бенкен Е. С., Самков Г. А., 2009
© Оформление, издательство "БХВ-Петербург", 2009

Оглавление

Введение **3**

Терминология 3
Структура книги 5
Как работать с книгой 6
Источники информации 7
Благодарности 7

ЧАСТЬ I. ТЕХНОЛОГИИ, СОСТАВЛЯЮЩИЕ AJAX **9**

Глава 1. Принцип работы AJAX **11**

Глава 2. Объектно-ориентированное программирование в серверных приложениях **14**

Принципы объектно-ориентированного программирования 14
Объектная модель в PHP 5. Классы и объекты 15
Конструктор класса 16
Создание объекта 17
Деструктор объекта 17
Копирование и клонирование объектов 19
Наследование 20
Финальные классы 22
Доступ к свойствам и методам класса 24
Статические свойства и методы класса 27
Абстрактные классы и интерфейсы 28
Константа класса 29
Ключевое слово *instanceof* 30
Обработка ошибок 30
Автозагрузка класса 32
Итераторы: просмотр всех общедоступных свойств объекта 33
Синглетон 34

Глава 3. Объектно-ориентированное программирование в JavaScript 35

Создание объекта с помощью оператора *new* 36
Создание объектов с помощью объектных литералов 36
Конструктор объекта 37
Функции как объекты 38
Добавление методов при помощи прототипа 38
Наследование при помощи прототипа 40
Создание класса-наследника 41
Полиморфизм 42
Частные элементы классов 43
Пространства имен 44
Обработка ошибок 45
Синглетоны 46
Замыкания 47
Применение замыканий 48

Глава 4. XML и JSON 51

Язык XML 51
Синтаксис XML. Правильно оформленный XML 51
XML-декларация 53
Атрибуты 53
Комментарии 53
Процессуальная инструкция 55
Пространства имен XML 55
Особые символы 56
CDATA 57
JSON 58

Глава 5. Объектная модель документа 61

Объект *Node* 64
Свойства и методы объекта *Document* 65
Доступ к узлу DOM 66
Объект *Element* 66
Объект *NodeList* 67
Объект *NamedNodeMap* 67
Объект *Attr* 68
Объект *Text* 68
Объект *DOMImplementation* 68

Глава 6. DOM в JavaScript 69

Объект *Element* 69

Создание HTML-элемента с помощью методов DOM и включение его в дерево документа 71

Чтение данных из XML-документа 73

Глава 7. DOM-функции в PHP 75

Создание XML-документа с помощью DOM-функций 76

Глава 8. Проблема русификации Web-приложений 81

Кодировки 81

Передача локализованных данных в протоколе HTTP 83

Кодирование символов в сценарии JavaScript 85

Русский язык в PHP 86

Локализация MySQL 89

ЧАСТЬ II. СОЗДАНИЕ AJAX-ПРИЛОЖЕНИЙ 91

Глава 9. Объект *XMLHttpRequest* 93

Глава 10. Использование XML и создание периодических запросов 101

Создание периодических запросов 108

Глава 11. Запрос данных с сервера MySQL 112

Передача данных в формате JSON 118

ЧАСТЬ III. БИБЛИОТЕКИ ДЛЯ РАБОТЫ С AJAX 125

Глава 12. Обзор библиотек для создания AJAX-приложений 127

Глава 13. Библиотека Prototype 131

Полезные методы в Prototype 132

Класс *Element* 135

Класс *Array* 137

AJAX в Prototype 141

Класс *Ajax.Request* 142

Класс *Ajax.Response* 143

Класс *Ajax.Updater* 144

Класс *Ajax.PeriodicalUpdater* 145

Использование AJAX-запросов в Prototype 145

Глава 14. Библиотека script.aculo.us 149

Эффекты 150
Перетаскивание и сортировка (Draggable & Sortable) 153
AJAX в script.aculo.us 157
Автодополнение 157
Класс *Ajax.InPlaceEditor* 161

ЧАСТЬ IV. БИБЛИОТЕКА EXTJS 165

Глава 15. Структура и идеология библиотеки 167

Соглашения об именах 169
Конфигурирование ExtJS и первый пример применения 169
Объект *Ext.Element* 171
Firebug — запаситесь выжигателем жучков 172
Контекст 172
Задание контекста в ExtJS 173
Адаптеры и пространство имен 175
Механизм наследования в ExtJS 175
Вызов метода базового класса 176
Обработка событий в ExtJS 177
События DOM 177
События JavaScript 177
Пользовательские события 179
Xtypes 180
Классы ExtJS 181
Класс *Component* 181
Класс *BoxComponent* 183
Класс *Container* 184
Класс *Panel* 184
Компоновка *(layout)* 184

Глава 16. Поиск элементов: класс *DomQuery* 187

Выбор узлов DOM 187
Селекторы элементов 188
Селекторы атрибутов 188
Отбор элементов CSS Value selectors 188

Глава 17. Панели и компоновка элементов 196

Простая панель 196
Вложенные панели 198

Компоновка панелей: создание аккордеона 199
Панель с несколькими вкладками 203

Глава 18. Формы 209

Создание элемента формы 209
Компоновка формы 210
Передача данных формы на сервер методом *submit* 213
Проверка форм с помощью класса *VTypes*. Календарь-подсказка 217

Глава 19. Визуальные эффекты. Drag & drop 222

Свертывание и развертывание блока 222
Изменение размеров блока 226
Drag & drop 229

Глава 20. Простые виджеты 236

Всплывающие подсказки 236

Глава 21. Создание редактируемых таблиц 240

Создание базы данных 240
Серверный сценарий для запроса к базе и генерации ответа клиенту 241
Клиентская часть: HTML и сценарий JavaScript 243
Разработка динамически редактируемой таблицы 249

ЧАСТЬ V. jQuery 259

Глава 22. Знакомство с jQuery 261

Установка библиотеки 262
Что такое *$()*? 263

Глава 23. Функции ядра jQuery 265

Доступ к объекту jQuery 270

Глава 24. Селекторы jQuery 275

Базовые селекторы 275
Иерархические селекторы 280
Основные фильтры 286
Фильтры содержимого 298
Фильтры видимых и невидимых элементов 302
Фильтры атрибутов 306
Фильтры потомков 316
Селекторы в формах 323
Фильтры состояния элементов форм 326

Глава 25. События в jQuery 331

Помощники при работе с событиями 332

Глава 26. Манипуляции элементами в jQuery 352

Изменение содержимого элементов 352
Вставка содержимого внутрь элементов 358
Вставка содержимого снаружи элементов 364
Обертывание элементов 369
Замещение, удаление, копирование элементов 371

Глава 27. AJAX-запросы в jQuery 378

Загрузка содержимого 378
Реализация *GET*-запросов 385
Реализация *POST*-запросов 392
Полный контроль над AJAX-запросами 395

Глава 28. События AJAX в jQuery 402

Глава 29. Расширения для jQuery 410

Плагин jQuery Form 410
Плагин Live Query 416
Резюме 420

ПРИЛОЖЕНИЯ 421

Приложение 1. Установка Web-сервера Apache, модуля PHP 5 и сервера MySQL в Windows 423

Установка сервера Apache 423
Директивы конфигурации Apache 425
Установка модуля PHP 426
Установка сервера MySQL 5 428

Приложение 2. Отладка JavaScript. Использование Firebug 431

Выполнение и отладка кода JavaScript 432
Просмотр HTTP-заголовков и AJAX-запросов 434

Приложение 3. Описание компакт-диска 436

Литература 437

Предметный указатель 439

Не секрет, что встречаются профессионалы-практики, которые на бумаге не могут структурированно изложить свои знания и опыт, и есть те, кто пишет много книг, но не обладает реальным практическим опытом. К счастью, авторы книги не относятся ни к первым, ни ко вторым. В Елене Сергеевне сочетается профессиональный опыт и умение донести этот опыт до аудитории. Как и предыдущие книги автора, эту отличает от многих других то, что изложенный материал апробирован на сотнях слушателей самых разных компаний, которые проходили обучение у Елены Сергеевны. Для лучшего закрепления материала, в книге приведены различные практические задания для самостоятельной работы.

Мы считаем, что книга является качественным продуктом и заслуживает того, чтобы ее прочитали те, кого интересует данная область. Как и многие другие книги, написанные нашими преподавателями, данная книга выходит в серии "Avalon.ru — советуют профессионалы".

Опыт предыдущих книг показал, как важен контакт между тем, кто написал книгу, и тем, кто ее изучает. Книга не позволяет наладить диалог между автором и читателем, однако, для успешного восприятия материала, важно иметь возможность задать преподавателю вопрос и получить квалифицированный ответ. Если после прочтения книги у вас появились вопросы, на которые вы не нашли ответа, если вы хотите получить еще больше практических заданий, чтобы повысить свою квалификацию, если вы просто хотите сказать спасибо автору данной книги, вы можете сделать это на форуме нашего учебного центра в соответствующем разделе: **http://forums.avalon.ru**.

На ваши вопросы ответят лично авторы книги.

Проректор Санкт-Петербургского государственного
политехнического университета

Александр Витальевич Речинский

Декан факультета переподготовки
специалистов СПбГПУ

Щукин Александр Валентинович

Введение

Предлагаемая книга является учебником в полном смысле этого слова, не больше, но и не меньше.

Предполагается, что читатель знаком с основами JavaScript на среднем уровне, но ему не приходилось вдаваться в детали объектной модели этого языка. Потребуются и знания основ PHP, но изложение объектной модели этого языка приведено.

Кроме того, кратко излагаются основы XML. Затем рассказывается, как именно создается AJAX-запрос к серверу и как на клиентской стороне обработать ответ сервера.

Большое внимание в настоящей книге уделено библиотекам JavaScript, позволяющим упростить и ускорить процесс создания Web-приложений, главным образом рассматриваются библиотеки jQuery и ExtJS.

Терминология

В 2005 г. появился термин, обозначивший основное направление развития Интернета — Web 2.0. Как часто случается в информационных технологиях, термин есть, а точного определения нет. Но явление, которое он обозначил, все-таки можно охарактеризовать некоторыми общими чертами. Главный принцип Web 2.0 — усиление технологий за счет коллективного разума. Ярким примером этого служит проект Wikipedia (**http://wikipedia.org**). С другого ресурса, Google, началась популярность еще одной важной составляющей понятия Web 2.0 — AJAX.

AJAX (Asynchronous JavaScript and XML, асинхронный JavaScript и XML) — это подход к построению пользовательских интерфейсов Web-приложений, при котором данные, запрошенные пользователем, отображаются на уже загруженной странице. При этом не происходит полной перезагрузки страницы, обновляется только ее часть. Использование AJAX стало наиболее популярно после того, как Google начала активно использовать его при создании

сайтов, таких как Gmail и Google Maps, а также Google Suggest — технологии автозаполнения строки поискового запроса на основе общей статистики самых популярных запросов.

AJAX в настоящее время пользуется популярностью горячих пирожков в холодный день. Правда, при ближайшем рассмотрении многие выясняют, что разработчику Web-приложений с использованием AJAX требуется такой объем знаний, что эта модная технология приближается по стоимости к суши, а то и бутербродам с черной икрой. Многие авторы отмечают, что применение термина "технология" по отношению к AJAX не является корректным: ведь он представляет собой просто совокупность нескольких ранее известных технологий.

Для освоения AJAX важно хорошо разбираться в объектной модели документа — DOM (Document Object Model). Тема это обширная, местами занудная, а потому непопулярная среди авторов учебников по Web-технологиям. Но именно описанию и примерам использования DOM уделено особое внимание в данной книге.

Второй важный компонент AJAX — язык XML. Его в наше время надо знать любому Web-мастеру. Понимание XML коренным образом изменяет видение современного Интернета, да и в целом информационных технологий.

Правда, следует сказать, что упоминание XML именно в аббревиатуре AJAX в настоящее время в значительной степени стало архаикой, там, где простой текст недостаточен, XML все больше вытесняется форматом JSON.

Но главной составляющей AJAX все-таки является JavaScript, и данная книга рассчитана на читателя, знакомого с основами этого языка.

Лучшей же доступной русскоязычному читателю книгой по JavaScript является книга Д. Флэнагана "JavaScript. Подробное руководство"[1] — вот где, в частности, много примеров применения DOM в JavaScript.

Конечно, необходимо и знание CSS.

Кроме того, читателю потребуется знание языка PHP 5. Лучшей книги для изучения PHP, чем книга Л. Веллинга, Л. Томсон "Разработка Web-приложений с помощью PHP и MySQL"[2], предложить трудно. Хотя дочитывать ее до конца для изучения AJAX и необязательно. Правда, бросить ее из-за скуки вам не удастся!

---

[1] Флэнаган Д. JavaScript. Подробное руководство. // Пер. с англ. — СПб.: Символ-плюс, 2008.

[2] Веллинг Л., Томсон Л. Разработка Web-приложений с помощью PHP и MySQL. — М.: Вильямс, 2007.

Наконец, надо знать хотя бы основы работы с MySQL. Укажем только любимую книгу Л. Аткинсона "MySQL. Библиотека профессионала"[3]. Опять же там рассказано гораздо больше, чем потребуется для изучения AJAX.

Структура книги

Книга содержит пять частей и три приложения.

В *части I* представлен обзор технологий, составляющих AJAX:

- принцип работы AJAX;
- объектно-ориентированное программирование в интернет-приложениях. Объектная модель PHP 5, работа с объектами в JavaScript;
- объектная модель документа DOM, DOM в JavaScript и PHP;
- основы XML и работа с форматом JSON;
- обработка XML-документов с помощью DOM-функций в PHP;
- проблемы русификации Web-приложений, использующих JavaScript, XML, PHP и MySQL.

В *части II* изложены основы AJAX:

- объект `XMLHTTPRequest`;
- примеры создания Web-приложений: создание, отправка и обработка результатов запроса;
- генерация данных в форматах XML и JSON на сервере, обработка таких данных на клиентской машине;
- создание повторяющихся запросов, особенности разработки AJAX-приложений;
- запрос данных с сервера MySQL.

В *части III* обсуждаются библиотеки для работы с Ajax, в том числе:

- приведен обзор библиотек для ускорения разработки Web-приложений;
- библиотека Prototype: основы и примеры использования;
- Scriptaculous: создание визуальных эффектов и развитие возможностей библиотеки Prototype;
- библиотека Jquery: все, что вы хотели узнать об этой библиотеке;
- библиотека ExtJS, принципы построения и примеры применения.

[3] Аткинсон Л. MySQL. Библиотека профессионала. — М.: Вильямс, 2002.

В *части IV* излагаются основы работы с библиотекой ExtJS:

- архитектура и идеология библиотеки ExtJS;
- реализация объектной модели в ExtJS;
- генерация элементов Web-страницы и компоновка элементов с помощью ExtJS;
- создание элементов интерфейса с помощью ExtJS;
- применение AJAX-запросов в ExtJS для получения данных с сервера;
- создание редактируемой таблицы.

Часть V содержит описание библиотеки jQuery, а именно:

- установка jQuery;
- функции ядра jQuery, доступ к объекту jQuery;
- селекторы jQuery;
- обработка событий в jQuery;
- манипуляции элементами в jQuery;
- AJAX-запросы в jQuery;
- глобальные и локальные события AJAX в jQuery;
- расширения для jQuery, плагины jQuery Form и Live Query.

В *приложении 1* обсуждается порядок установки сервера Apache и модуля PHP в операционной системе Windows.

В *приложении 2* кратко рассматривается Firebug — отладчик для Web-приложений, работающий с Mozilla Firefox.

Приложение 3 — это описание компакт-диска, приложенного к книге.

Как работать с книгой

Книга ориентирована на разработчика, располагающего компьютером с операционной системой Windows, но пользователь UNIX также сможет выполнить на своем компьютере все примеры.

В ходе чтения следует выполнять на компьютере примеры, описываемые в книге. Каждый пример стоит изменять и переделывать самому с тем, чтобы лучше понимать, как он работает или как сделать так, чтобы все перестало работать.

Авторы приложили все усилия, чтобы изложить материал с наибольшей точностью, но не исключают возможности ошибок и опечаток. Авторы также не несут ответственности за последствия использования сведений, изложенных в книге.

Источники информации

Как известно, в книге нельзя охватить все вопросы, и читателю нужно иметь возможность получить дополнительные сведения, например, из Интернета. Вот адреса, которыми вы можете воспользоваться:

- **http://apache.org**;
- **http://php.net**;
- **http://dev.mysql.com**;
- **http://w3schools.com** — школы W3C по XML;
- **http://www.w3.org/** — содержит всеобъемлющую информацию по стандартам Интернета;
- **http://json.org**;
- **http://developer.mozilla.org**.

Можно рекомендовать следующие русскоязычные сайты:

- **http://phpclub.ru**;
- **http://opennent.ru** — этот сайт в основном посвящен операционным системам, в первую очередь семейству UNIX, но посмотрите внимательно — здесь прекрасные статьи, отслеживаются все новости и даются ссылки на оригиналы и переводы статей;
- **http://phpworld.ru** — сайт посвящен PHP 5;
- **http://xml.nsu.ru** — переводы на русский язык школ консорциума W3C по XML;
- **http://zvon.org** — учебник по XML, XSLT.

Благодарности

Авторы приносят свои благодарности Василию Руже за советы, позволившие увидеть логику развития технологии.

Часть I

Технологии, составляющие AJAX

Глава 1

Принцип работы AJAX

Идея AJAX (Asynchronous Javascript and XML) была изложена Джесси Гарретом в его статье "AJAX: новый подход к Web-приложениям" (см. **http://www.adaptivepath.com/ideas/essays/archives/000385.php**, русский перевод — **http://ajax-development.narod.ru/ajax-article.html**).

Web-приложения — это приложения, функциональные возможности которых обеспечиваются сервером и доставляются пользователям по Интернету или интрасети. Классическая модель Web-приложения действует следующим образом. Клиентское приложение отправляет на сервер HTTP-запрос. Сервер производит необходимую обработку: считывает и обрабатывает данные, взаимодействует с различными системами, например, с базами данных или другими серверами, и затем выдает HTML-страницу клиенту (рис. 1.1). Страница может содержать таблицы CSS и сценарии JavaScript.

Существенным недостатком такого алгоритма взаимодействия клиента с сервером является то, что клиенту приходится ждать загрузки каждой последующей страницы.

Суть идеи Гарретта состоит в том, что ожидание клиента сокращается или становится совсем незаметным за счет нескольких усовершенствований (рис. 1.2).

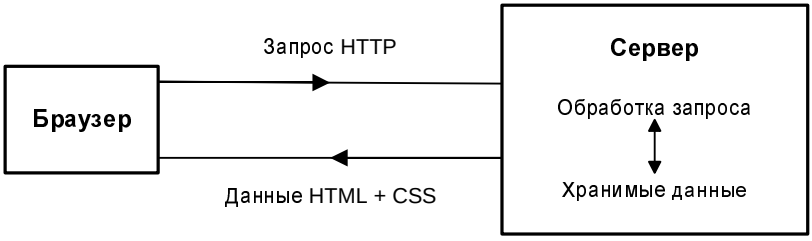

Рис. 1.1. Классическая схема работы Web-приложения

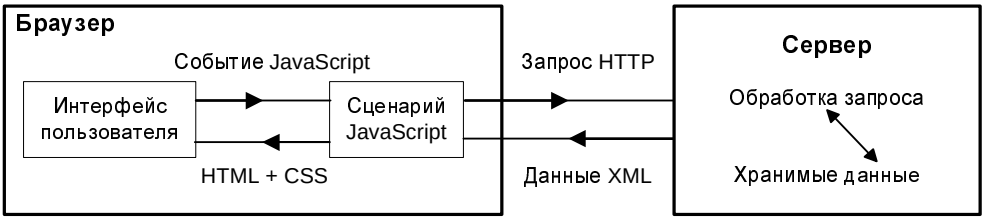

Рис. 1.2. AJAX-модель Web-приложения

AJAX перераспределяет нагрузку между клиентом и сервером, разрешив им общаться между собой, пока пользователь работает со страницей. Клиент загружает в браузер страницу, содержащую сценарий JavaScript. Этот сценарий включает в себя функции обработки событий, которые генерируют HTTP-запрос на сервер. Запрос отправляется незаметно для пользователя. В то время, когда запрос обрабатывается на сервере и происходит передача ответа клиенту, последний продолжает работу, не ожидая полной перезагрузки страницы. Клиентский сценарий отслеживает состояние этого запроса, и, как только все данные, загруженные в качестве ответа сервера, получены браузером, происходит обновление части Web-страницы, уже отображаемой в окне браузера. Такое обновление происходит в результате работы того же сценария JavaScript, который обрабатывает данные, полученные от сервера, и отражает их в определенном фрагменте Web-страницы.

Ответ сервера в AJAX может представлять собой простой текст, текст в XML-формате, в честь которого и добавлена последняя буква в название технологии AJAX, или в формате JSON, который следует признать наиболее удобным для многих Web-приложений.

Основной выигрыш в скорости работы получается за счет того, что запрос к серверу отправляется незаметно для клиента, который продолжает работу, не дожидаясь ответа сервера. Данные же ответа встраиваются в имеющуюся на клиентской стороне страницу.

Итак, AJAX объединяет:

- стандартизованное представление данных с использованием XHTML и CSS;
- динамическое отображение и обработку данных на стороне клиента в сценарии JavaScript при помощи Document Object Model;
- асинхронное получение данных с использованием объекта `XMLHttpRequest`, создаваемого сценарием JavaScript;
- обмен данными XML или данными в других текстовых форматах.

AJAX не является чудесным средством, одним своим появлением украшающим Web-приложения, но разумное использование этой технологии может сделать сайты более дружественными к пользователю. AJAX часто применяют для решения следующих задач:

- проверка правильности заполнения форм с привлечением возможностей сервера;
- подсказки для автодополнения;
- создание динамических таблиц данных (girds), которые на лету обновляют базы данных на сервере;
- разработка приложений, которые требуют обновления информации в режиме реального времени, получая ее из различных источников.

Применение AJAX создает определенные трудности, а именно:

- динамически создаваемые страницы могут иметь один и тот же адрес, поэтому не работает кнопка **Назад**, предоставляющая пользователям возможность вернуться к просмотренным ранее страницам;
- изменение содержимого страницы при постоянном адресе приводит к тому, что сделать закладку на странице непросто;
- поисковые машины не могут проиндексировать все страницы сайта, созданного на основе AJAX;
- на клиентской стороне JavaScript может быть отключен, в результате AJAX-приложения перестанут работать.

Глава 2

Объектно-ориентированное программирование в серверных приложениях

Принципы объектно-ориентированного программирования

В реальной жизни мы повседневно имеем дело с объектами. Любые предметы, окружающие нас, можно представлять как объекты, характеризующиеся некоторыми свойствами. Кроме того, объекты могут совершать действия или над ними можно производить какие-либо действия.

Понятие объекта в программировании предоставляет программисту возможность оперировать данными как объектами реальной жизни. *Объект* представляет собой совокупность свойств и операций, выполняемых объектом или над объектом. Операции принято называть *методами*.

Объектно-ориентированное программирование базируется на трех основных принципах: инкапсуляции, наследовании и полиморфизме.

Инкапсуляция подразумевает обращение с объектом как с черным ящиком: при работе с объектом не видно его внутреннего устройства. Объект дает возможность составить о нем представление по ограниченному числу доступных свойств и методов. Инкапсуляция как раз и означает, что объект представляет собой совокупность данных и операций с ними. Инкапсуляция — это механизм защиты свойств и операций объекта, ограждающий их от неправильного использования. Данные внутри объекта могут быть доступны только для других частей этого объекта, но закрыты от программ, находящихся вне этого объекта.

Полиморфизм позволяет использовать одно и то же имя для решения разных задач. Иначе говоря, одно и то же имя метода для разных объектов может означать разные действия.

Наследование — это процесс, позволяющий объекту приобретать свойства другого объекта. В процессе наследования объект может получить свойства объекта-родителя, добавив к ним свои особенности.

Объектная модель представляет собой удобный инструмент для программиста, ибо позволяет разделять работу над проектом между несколькими участниками. Каждый работает над своим объектом, над его интерфейсами, при этом минимально пересекаясь с другими программистами. Этот подход обеспечивает также наилучшие условия для повторного использования созданного кода.

В данной главе будут рассмотрены основные черты объектных моделей языков PHP и JavaScript. Читатель сможет увидеть сходства и отличия объектов и операций с ними в этих языках. При построении объектной модели PHP за образец была взята широко распространенная классовая модель, реализованная в C или Java. В JavaScript объекты наследуются иным способом — на основе прототипов. В JavaScript нет классов как таковых, но все задачи объектного программирования, тем не менее, решаются. Понимание этих особенностей понадобится читателю в тех главах нашей книги, где обсуждаются библиотеки JavaScript, применяемые при разработке Web-приложений с использованием AJAX-запросов.

Объектная модель в PHP 5. Классы и объекты

Объектная модель PHP базируется на понятии класса. *Класс* определяет совокупность свойств, которые имеет объект данного класса, а также и операций (методов), выполняемых над объектом.

Представьте себе класс млекопитающих. Принадлежность животного к этому классу определяется по наличию у животного некоторых характеристик — свойств. Например, кровь у млекопитающих теплая, при этом все млекопитающие могут совершать действия, например, двигаться или кормить детенышей молоком.

На языке программирования свойство или атрибут — это переменная, имеющая некоторое значение. Действие, совершаемое объектом, — это функция. Мы можем объявить несколько свойств:

```
public $blood, $legs;
```

Ключевое слово `public` указывает, что после него идет объявление элемента класса (т. е. свойства или метода). Кроме того, это ключевое слово определяет

механизм доступа к элементу, о чем пойдет речь далее в этой главе. Дадим свойству значение:

```
$blood="теплая";
```

Определим метод `move()`, который принимает один аргумент `$legs` — количество лап у животного:

```
public function move($legs)
{
  if ($legs) echo "$this->name двигается на $legs ногах <br>";
  else echo "Животное плавает";
}
```

Значение переменной `$legs` надо указать при вызове метода — ничего необычного в этом нет, так мы делали при вызове функции.

Специальный указатель `$this` применяется для обозначения объекта. Аналогично в процедурном программировании в определении функции указывается формальный аргумент. В приведенном здесь фрагменте кода распечатывается значение свойства `name` объекта, для которого вызывается метод `move()`.

Можно сказать, что класс млекопитающих характеризуется совокупностью свойств и методов. Какое-либо животное, относящееся к этому классу, будет обладать этими свойствами и методами, но возможно, что методы будут реализовываться по-разному (ног-то может быть разное количество).

Конструктор класса

При создании объекта — экземпляра класса вызывается функция, которая инициализирует все требуемые переменные, выполнит все действия, нужные для полного определения объекта. Эта функция называется *конструктором*.

В PHP 5 конструктор — это метод, имеющий зарезервированное имя `__construct`, и может быть определен так:

```
function __construct($name)
{
  $this->name = $name;
  $this ->blood='теплая';
  echo "Запущен конструктор класса mammal <br>";
}
```

Свойства класса `name` и `blood` получают значения при вызове конструктора.

Создание объекта

Объект — экземпляр класса, который можно создать с помощью оператора `new`, после которого указывается имя класса и параметры, передаваемые конструктору:

```
$cat = new mammal("кошка");
```

При создании объекта `$cat` для него устанавливаются значения свойств. Получить доступ к ним можно так:

```
echo $cat->name;
```

Вызвать метод объекта `$cat` можно таким образом:

```
$cat->move(4);
```

Напишем определение класса `mammal` (листинг 2.1).

Листинг 2.1. Класс `mammal`

```
<?php
class mammal
{
  public $blood, $legs;
  public function __construct($name)
  {
    $this->name = $name;
    $this->blood="теплая";
    echo "Запущен конструктор класса mammal <br>";
  }
  public function move($legs)
  {
    if ($legs) echo "$this->name двигается на $legs ногах <br>";
    else echo "Животное плавает";
  }
}
?>
```

Деструктор объекта

Объект можно уничтожить в ходе выполнения сценария, вызвав функцию `unset()` и передав ей в качестве параметра имя объекта. Но в любом случае при завершении работы сценария память, занимаемая объектом, высвобож-

дается, и объект из нее удаляется. В объектной модели PHP 5 определена функция `__destruct()`, которая вызывается автоматически при уничтожении объекта.

В листинге 2.2 вы можете увидеть определение конструктора и деструктора класса, а также создание объекта как экземпляра созданного класса.

Листинг 2.2. Класс `mammal` и создание объекта

```
<?php
class mammal
{
  public $blood, $legs;
  public function __construct($name)
  {
    $this->name = $name;
    $this->blood="теплая";
    echo "Запущен конструктор класса mammal <br>";
  }
  public function move($legs)
  {
    if ($legs) echo "$this->name двигается на $legs ногах <br>";
    else echo "Животное плавает";
  }
  function __destruct() {
    echo "Вызван деструктор объекта <br>";
  }
}
$cat = new mammal("кошка");
echo $cat->name."<br>";
$cat->move(4);
unset($cat);
echo "А теперь завершается работа сценария";
?>
```

При выполнении этого примера обратите внимание на то, что деструктор выполняется именно при вызове функции `unset()`. Если же вы удалите из сценария строку с этой функцией, то деструктор будет вызван в самом конце работы после выполнения всех остальных операторов. Деструктор — это подходящее место для действий, которые наводят порядок после выполнения различных работ: закрывают соединения с серверами управления базами данных, очищают память для предотвращения утечек памяти и т. п.

Копирование и клонирование объектов

При копировании объектов не происходит копирование данных — создается только ссылка на область данных так же, как при создании ссылки на переменную.

Попробуем создать два объекта класса млекопитающих — кошку и кита. Дадим кошке 4 лапы, а про кита скажем — нет у него лап (`$legs=0`). Посмотрим, что получилось (листинг 2.3).

Листинг 2.3. Копирование объектов

```
<?php
class simple_mammal
{
  public $legs;
}
$cat = new simple_mammal;
$cat -> legs = 4;
$whale = $cat;
$whale -> legs = 0;
echo $cat -> legs;
echo $whale -> legs;
?>
```

А ничего хорошего не вышло! Поскольку при копировании объекта не копировалась область данных, то и у кошки лап не оказалось.

Чтобы скопировать свойства и методы объекта, надо применить клонирование (листинг 2.4).

Листинг 2.4. Клонирование объектов

```
<?php
class mammal
{
   public $legs;
}
$cat = new mammal;
$cat -> legs = 4;
$whale = clone $cat;
```

```
$whale -> legs = 0;
echo $cat -> legs;
echo $whale -> legs;
?>
```

Вот теперь все работает так, как задумывалось. Последние примеры призваны пояснить отличительные черты работы с объектами в PHP 5: при создании копии объекта с помощью оператора присваивания (`$whale=$cat`) создается ссылка на объект `$cat`, а не копия всех свойств и методов объекта `$cat`.

Наследование

Можно определить класс `beast`, являющийся наследником ранее определенного класса `mammal`:

```
class beast extends mammal
```

Класс-наследник может наследовать свойства и методы класса-родителя, а может их переопределить (это называется *перегрузкой*):

```
public $fur;  // Объявляем новое свойство
```

Метод `move()` перегрузим:

```
function move($legs)
{
  if ($legs) echo "$this->name бегает, лазает по деревьям на ".
                  $legs." лапах <br>";
}
```

Создадим новый метод, присущий только этому классу-наследнику:

```
function description()
{
  $this->fur="мягкая и пушистая";
  echo $this->name, " ", $this->fur, " . ";
  echo "Кровь - ", $this->blood, "<br>";
}
```

Конструктор родительского класса не вызывается автоматически при создании объекта — экземпляра класса-наследника. Его следует вызвать явно, используя символ двойного двоеточия. Конструктор класса `beast` определяется следующим образом:

```
function __construct($name)
{
```

```
  parent::__construct($name);
  echo "Запущен конструктор класса beast <br>";
}
```

В итоге определение класса `beast` выглядит так, как показано в листинге 2.5.

Листинг 2.5. Класс `beast`

```
<?php
class mammal{}
class beast extends mammal
{
  public $fur;
  function __construct($name)
  {
    parent::__construct($name);
    echo "запущен конструктор класса beast <br>";
  }
  function move($legs)
  {
    if ($legs) echo "$this->name бегает, лазает по деревьям на ".
                    $legs." лапах <br>";
  }
  function description()
  {
    $this->fur="мягкая и пушистая";
    echo $this->name, " ", $this->fur, " . ";
    echo "Кровь - ", $this->blood, "<br>";
  }
}
?>
```

Можем теперь создать объекты класса-наследника и посмотреть, как это все работает:

```
$Murka = new beast("кошка");
$Murka-> move(4);
$Murka->description();
```

Финальные классы

Теперь мы хотим создать класс `cat` — потомок класса `beast`. Класс `cat` является также потомком класса `mammal`. Мы хотим указать, что у этого класса `cat` никаких наследников быть не может. Для этого используется ключевое слово `final`:

```
final class cat extends beast
```

Из-за применения этого ключевого слова попытки создать классы-наследники класса `cat` будут вызывать сообщение об ошибке.

Объявим новое свойство `$sound` и определим его значение в конструкторе:

```
public $sound;
function __construct($name)
{
  parent::__construct($name);
  echo "Запущен конструктор класса cat <br>";
  $this->sound="мурр";
}
```

Объявим и новый метод:

```
function speak()
{
  echo $this->name, " говорит ", $this->sound."<br>";
}
```

Объявление класса и создание объекта, экземпляра этого класса, будет выглядеть так, как показано в листинге 2.6.

Листинг 2.6. Класс `cat`

```
<?php
class mammal
{
  public $blood, $legs;
  public function __construct($name)
  {
    $this->name = $name;
    $this->blood="теплая";
    echo "Запущен конструктор класса mammal <br>";
  }
```

```
  public function move($legs)
  {
    if ($legs) echo "$this->name двигается на $legs ногах <br>";
    else echo "Животное плавает";
  }
  function __destruct() {
    echo "Вызван деструктор объекта <br>";
  }
}
class beast extends mammal
{
  public $fur;
  function __construct($name)
  {
    parent::__construct($name);
    echo "запущен конструктор класса beast <br>";
  }
  function move($legs)
  {
    if ($legs) echo "$this->name бегает, лазает по деревьям на ".
                    $legs." лапах <br>";
  }
  function description()
  {
    $this->fur="мягкая и пушистая";
    echo $this->name, " ", $this->fur, " . ";
    echo "Кровь - ", $this->blood, "<br>";
  }
}

final class cat extends beast
{
  public $sound;
  function __construct($name)
  {
    parent::__construct($name);
    echo "Запущен конструктор класса cat <br>";
    $this->sound="мурр";
  }
```

```
  function speak()
  {
    echo $this->name, " говорит ", $this->sound."<br>";
  }
}
// Теперь создадим объект этого класса и вызовем его методы:
$Murka = new cat("кошка");
$Murka-> move(4);
$Murka->description();
$Murka->speak();
?>
```

Объект `$Murka` создается, и все методы работают, но попробуйте добавить в конец этого сценария строки

```
class ChildClass extends cat {
   public function moreTesting() {
       echo "Вызван метод ChildClass::moreTesting()\n";
   }
}
```

и вы увидите на экране сообщение об ошибке:

```
Fatal error: Class ChildClass may not inherit from final class (cat) in
C:\Program Files\Apache Group\Apache2\htdocs\13-6.php on line 64
```

Из приведенных примеров видно, когда стоит использовать объектно-ориентированный подход. Если бы зверей было раз, два и обчелся, никто бы не стал заводить их классификацию, упорядочивать перечень их свойств и методов. Но животных много, потребовалось структурировать информацию о них, вот и завели классы и семейства. Так и в программировании, затраты времени на создание классов окупятся, если надо писать большой проект.

Доступ к свойствам и методам класса

В объектно-ориентированном программировании доступ к свойствам класса осуществляется обычно с определенными ограничениями. Рекомендуется, чтобы работа со свойствами велась через вызов методов — это называется *инкапсуляцией*. Смысл инкапсуляции состоит в том, что внешний пользователь не знает детали реализации объекта, работая с ним путем предоставленного объектом интерфейса.

Для реализации инкапсуляции надо ввести ограничения на доступ к свойствам и методам класса (то и другое можно называть элементом класса) с помощью следующих спецификаторов.

- `public` — общедоступный или открытый, к свойствам и методам, помеченным этим спецификатором, можно получить доступ без каких-либо ограничений. В PHP 4 все свойства и методы объявлялись только как общедоступные, причем перед именем свойства писалось ключевое слово `var`. Объектный код, созданный на языке PHP 4, будет работать и с модулем 5 версии, причем слово `var` будет трактоваться как определение свойства с открытым доступом.
- `protected` — защищенный или с ограничением доступа. Элементы этого типа доступны внутри класса, в котором они объявлены, и в его классах-наследниках. Оборот "внутри класса" означает, что прочесть защищенное свойство можно только с помощью метода, определенного в этом же классе.
- `private` — закрытый или частный. Элементы этого типа доступны только внутри класса, в котором они объявлены.

Если не указывать ни один из спецификаторов, то по умолчанию элемент будет общедоступным. Посмотрим, как все это работает, на примере. Создадим два класса (листинг 2.7).

Листинг 2.7. Типы доступа к свойствам класса

```
<?php
class mammal
{
  public $blood = "теплая";
  protected $legs = "4";
  private $eat = "молоко";
  public function printPrivate()
  {
     echo $this->eat;
  }
}
class cat extends mammal
{
   public function printProtected()
   {
     echo $this->legs;
   }
}
?>
```

Попробуем теперь создать экземпляр класса `mammal` и распечатать значения его свойств:

```
$Murka = new mammal;
echo $Murka->blood;
echo $Murka->classname;
```

Работает. А если так?

```
echo $Murka->legs;
```

Не выходит, получаем сообщение об ошибке:

```
Fatal error: Cannot access protected property mammal::$legs
```

А так?

```
echo $Murka->eat;
```

Нет, нельзя:

```
Fatal error: Cannot access private property mammal::$eat
```

Получить значение частного свойства можно только через вызов метода:

```
$Murka->printPrivate();
```

Определим этот метод так:

```
public function printPrivate()
{
  echo $this->eat;
  echo $this->legs;
}
```

Вызов такого метода предоставит нам значения и частного, и защищенного свойств. А в чем же между ними разница? Посмотрим, как обстоит дело с классом-наследником `cat`. Создадим новый экземпляр класса и вызовем его метод:

```
$Barsik = new cat();
$Barsik->printProtected();
```

Все отлично, значение защищенного свойства `$legs` читается. А теперь попробуем с помощью этого метода получить значение частного свойства `$eat`, добавив его вывод в определение метода `printProtected()`:

```
public function printProtected()
{
  echo $this->legs;
  echo $this->eat;
}
```

Нет, видим в браузере:

```
Notice: Undefined property: cat::$eat
```

Теперь перечитайте определения спецификаторов доступа, и они станут вам понятнее.

Статические свойства и методы класса

В дальнейшем, изучая классы для работы с XML-документами, вы обнаружите в них статические свойства. Мы с вами видели, как определяются статические переменные в процедурном программировании на PHP. Можно объявить статические свойства класса так, как представлено в листинге 2.8.

Листинг 2.8. Статическое свойство класса

```
<?php
class mammal
{
  static $feeding = "молоко";
}
?>
```

Статические свойства едины для всего класса и не могут принадлежать ни одному из объектов класса. Кроме этого, можно обратиться к такому свойству, не создавая объекта:

```
echo mammal::$feeding;
```

Аналогично можно определить статический метод и использовать его без создания объекта такого класса (листинг 2.9).

Листинг 2.9. Статический метод класса

```
<?php
class mammal
{
  static function moving()
  {
    echo "Все звери двигаются ";
  }
}
echo mammal::moving();
?>
```

Однако в статическом методе становится невозможным использовать указатель `$this`, т. к. при вызове статического метода неизвестно, в контексте какого объекта он вызывается.

Абстрактные классы и интерфейсы

Представьте себе, что вы создаете сетевую игру, участники которой ведут борьбу с противниками. Каждый игрок представлен фигуркой того или иного существа, обладающего разными видами оружия, защиты и пр. Для описания этих разнообразных и многочисленных героев вам придется создавать много классов — родителей и наследников, описывать в них методы, которыми могут воспользоваться участники игры. Возможно, в одном из базовых классов вы захотите только указать наличие некоторого метода, а конкретную его реализацию отложить до перегрузки его в классе-наследнике. Это можно сделать с помощью абстрактного метода.

Абстрактным называется метод, который имеет только объявление, но не имеет реализации. Реализация же выполняется в классе-наследнике. Класс, который содержит абстрактный метод, объявляется как абстрактный. При этом в нем могут содержаться и обычные, неабстрактные, методы.

Создать объект, являющийся экземпляром абстрактного класса, невозможно. Но вот после того, как в классе-наследнике бывший абстрактный метод будет переопределен, можно уже создавать экземпляры этого класса-наследника (листинг 2.10).

Листинг 2.10. Абстрактный класс

```
<?php
abstract class voin {
  abstract public function deistvie();
}
class super_voin extends voin {
  public function deistvie() {
    echo "Появилась реализация метода ";
  }
}
$obj = new super_voin;
$obj->deistvie();
?>
```

В PHP невозможно описать класс, являющийся наследником сразу двух классов, даже абстрактных. Для решения этой проблемы существуют интерфейсы, представляющие собой абстрактные классы, не содержащие ни одного неабстрактного метода. Класс может наследовать двум интерфейсам одновременно, переопределяя их методы. Можно создавать объекты — экземпляры такого класса-наследника. В листинге 2.11 представлено объявление двух интерфейсов и класса — их общего наследника.

Листинг 2.11. Интерфейсы

```
<?php
interface voin{
  function shoot();
}
interface artist{
  function paint();
}
class hero implements voin, artist {
  public function shoot() {
    echo "Герой умеет стрелять. ";
  }
  public function paint() {
    echo "Герой умеет рисовать. ";
  }
}
$obj = new hero;
$obj->shoot();
$obj->paint();
?>
```

Константа класса

В PHP 5 введен такой элемент класса, как *константа* (листинг 2.12).

Листинг 2.12. Константа класса

```
<?php
class baza
{
```

```
  const DBNAME = "taxi";
}
echo baza::DBNAME;
?>
```

Из приведенного листинга ясно, что использовать константу можно без создания объекта. Константы класса пригодятся нам для хранения параметров подключения к серверу баз данных или других данных, не подлежащих изменениям в ходе выполнения сценария.

Ключевое слово *instanceof*

Ключевое слово `instanceof` позволяет определить, является ли объект экземпляром определенного класса (назовем класс `country`) или экземпляром класса-наследника (класс-наследник назовем `city`) определенного класса (листинг 2.13).

Листинг 2.13. Ключевое слово `instanceof`

```
<?php
class country { }
$obj1 = new country();
if ($obj1 instanceof country) {
  echo "\$obj1 - объект класса country";
}
class city extends country{ }
$obj2 = new city();
if ($obj2 instanceof country) {
   echo "\$obj2 - объект класса, производного от country";
}
?>
```

Обработка ошибок

Посмотрим теперь, как в объектном коде реализована обработка ошибок. Мы видели, что при написании процедурного кода не рекомендуется выводить сообщения о возможных ошибках, генерируемые самим модулем PHP, на экран. Лучше предусмотреть возможность ошибки и создать свой код ее обработки. Например, может возникнуть проблема с открытием файла, так

не будем заставлять пользователя созерцать неудобочитаемые сообщения, а создадим свое, подходящее к случаю.

Для обработки ошибок в ядро PHP 5 встроен класс `Exception` (в листинге 2.14 представлена структура класса, но не описана реализация, это не рабочий пример, а демонстрация идеи). Конечно, нельзя разрешать пользователю переопределять методы этого класса, поэтому они объявлены как финальные.

Листинг 2.14. Класс `Exception`

```
<?php
class Exception
{
  protected $message = 'Unknown exception';  // сообщение об исключении
  protected $code = 0;   // код исключения, определяемый пользователем
  protected $file;       // имя файла-источника исключения
  protected $line;       // строка, в которой произошло исключение
  function __construct($message = null, $code = 0);
  final function getMessage();      // сообщение об исключении
  final function getCode();         // код исключения
  final function getFile();         // файл источника
  final function getLine();         // строка источника
  final function getTrace();        // массив  backtrace()
  final function getTraceAsString();// форматированная строка трассировки
  /* Overrideable — можно перегружать */
  function __toString();     // Создание строки для отображения на экране
?>
```

Во многих объектно-ориентированных языках реализована схема обработки исключений с помощью конструкции `try/catch/throw`. Она позволяет весь код обработки ошибок локализовать в одном месте сценария.

Попытаемся открыть несуществующий файл:

```
$fp = fopen("file.txt", "r");
```

Нельзя, получаем сообщение об ошибке. Скроем ее от пользователя:

```
@$fp = fopen("file.txt", "r");
```

Теперь напишем объектный код так, чтобы при возникновении ошибки, т. е. в случае, когда `$fp` получила значение `false`, создавался новый экземпляр `$exception` класса `Exception`. Создавать будем, конечно, оператором `new`.

Передадим для его создания в качестве значения свойства `$message` этого класса сообщение об исключении "Невозможно открыть файл!". Не будет ошибки, ну и хорошо, запишем все, что нужно, в файл. Но если возникнет ошибка, то она будет перехвачена конструкцией `catch`, которая напечатает "Ошибка в строке ", затем вызовет метод `getLine()` для нашего объекта и метод `getMessage()`. Эти методы вернут номер строки скрипта, в которой произошла ошибка, и значение свойства `$message`. Синтаксис обработки ошибок таков, как представлено в листинге 2.15.

Листинг 2.15. Обработка ошибок с помощью конструкции `try/catch/throw`

```
<?php
try
{
  @$fp = fopen("file.txt", "w");
  if (!$fp) throw new Exception("Невозможно открыть файл!");
  // Запись данных в файл при отсутствии ошибок
  fclose($fp);
}
catch (Exception $exception)
{
  echo "Ошибка в строке ", $exception->getLine();
  echo $exception->getMessage();
}
?>
```

Автозагрузка класса

Итак, есть у нас большой проект, при реализации которого мы решили использовать объектный подход. Определили классы и записали каждый класс в отдельный файл — так мы сможем соблюсти принцип модульности программного обеспечения и подключать объявления классов к любому скрипту, используя, например, инструкцию `include`. Теперь каждый из описанных ранее классов хранится в файлах mammal.php, beast.php и cat.php. Для их подключения в начале скрипта пишем:

```
include("mammal.php");
```

и т. д. для всех классов. Забудем подключить — получим ошибку при создании объекта. В PHP 5 есть возможность загрузки классов, не подключенных к скрипту, как указано ранее, с помощью глобальной функции `__autoload`.

Эта функция принимает один параметр — имя класса. Действия, которые будут выполняться при вызове функции автозагрузки, придется определить (листинг 2.16).

Листинг 2.16. Автозагрузка класса

```
<?php
function __autoload($class)
{
  include($class.".php");
}
$Barsik = new mammal("кошка");
echo $Barsik->move(4);
?>
```

При создании классов и файлов надо помнить о чувствительности к регистру и при необходимости выполнять соответствующие преобразования имен.

Объекты также могут сохраняться в сессии, но при чтении объекта из сессии для корректной работы необходимо подключить описание класса к тому скрипту, в котором происходит такое чтение. Функция `__autoload` поможет и в этом случае, позволяя загружать коды тех классов, экземплярами которых являются объекты, хранимые в сессии.

Итераторы: просмотр всех общедоступных свойств объекта

В объектной модели определена возможность перебора всех открытых свойств объекта с помощью оператора цикла `foreach()`. Это и выполнено в листинге 2.17.

Листинг 2.17. Использование итератора

```
<?php
class mammal
{
  public $blood, $legs;
  public function __construct($name)
  {
    $this->name = $name;
```

```
    $this->blood = "теплая";
    $this->legs = 4;
  }
}
$Murka = new mammal("Кошка");
foreach ($Murka as $svoistvo => $znachenie) {
  echo "$svoistvo => $znachenie";
}
?>
```

Подытоживая сказанное, следует отметить, что в PHP реализована объектная модель на основе классов со всеми основными принципами объектно-ориентированного программирования.

Синглетон

Синглетон — это шаблон проектирования, который следует применить в том случае, когда в приложении необходим только один экземпляр какого-либо класса. Посмотрите пример в листинге 2.18. Применение статического метода `doAction` позволяет решить поставленную задачу.

Листинг 2.18. Реализация синглетона

```
<?php
class Singleton {
  private static $instance;
  public static function getInstance() {
    if (self::$instance === null) {
      self::$instance = new self;
    }
    return self::$instance;
  }
  public function doAction() {
    echo "Вызов общедоступного метода doAction <br>";
  }
}
Singleton::getInstance()->doAction();
?>
```

Глава 3

Объектно-ориентированное программирование в JavaScript

Объект в JavaScript — это коллекция поименованных свойств. JavaScript относится к языкам прототипного программирования, при котором отсутствует понятие класса, а повторное использование (наследование) производится путем клонирования существующего экземпляра объекта — прототипа.

В JavaScript существуют два метода создания нового объекта: клонирование имеющегося объекта либо создание объекта "с нуля". Для создания объекта с нуля программисту предоставляются средства добавления свойств и методов в объект. В дальнейшем, с получившегося объекта может быть получена полная копия — клон. В процессе клонирования копия наследует все характеристики своего прототипа, но с этого момента она становится самостоятельной и может быть изменена.

Вообще же в JavaScript согласно спецификации ECMA Script определены три типа объектов: базовые (native), объекты браузера (host object) и объекты, создаваемые пользователем (user-defined).

Базовые объекты поддерживаются механизмом JavaScript, например, объекты `Object`, `Math` или `Number`. Имена базовых объектов являются чувствительными к регистру и начинаются с заглавной буквы. Второй тип объектов поддерживается браузером, обеспечивая, таким образом, взаимодействие пользователя с загруженным документом. Имена таких объектов начинаются со строчной буквы, например, `document`, `window`, `frames`.

Наконец, программист может сам создать объект и дать ему имя. Имя может начинаться с любой буквы и также является чувствительным к регистру. Объектом в JavaScript является даже функция.

Следует отметить, что JavaScript позволяет решать задачи объектно-ориентированного программирования, т. е. реализовывать инкапсуляцию,

полиморфизм и наследование. Для демонстрации справедливости последнего утверждения, нам потребуется показать, как в JavaScript можно:

- определить класс;
- определить и вызывать методы класса;
- определить класс-наследник;
- вызывать конструктор родительского класса из класса-наследника;
- переопределить методы родительского класса в классе-наследнике;
- вызывать методы родительского класса из класса-наследника.

Создание объекта с помощью оператора *new*

В JavaScript определен тип данных `Object`, который и применяется при создании объектов. Простейший способ создания объекта состоит в вызове оператора `new`:

```
obj = new Object;
obj.x = 1;
obj.y = 2;
```

При создании объекта к указанным нами явно свойствам добавляется еще одно — `constructor`. Вначале создается объект `obj`, а затем ему добавляются свойство и метод.

Создание объектов с помощью объектных литералов

Другой путь состоит в использовании объектных литералов, как в листинге 3.1.

Листинг 3.1. Создание объекта в JavaScript

```
<html>
<head>
  <script>
    var  nash_object = {
      svoistvo: "значение свойства",
      drugoe_svoistvo: "придумаем еще какое-нибудь значение",
```

```
      message: function(){ alert("Привет!"); },
      nash_metod: function(){ alert("Наш метод!"); }
    };
    alert(nash_object["svoistvo"]);
    nash_object.nash_metod();
  </script>
</head>
<body></body>
</html>
```

Литерал может содержать массивы и выражения JavaScript.

Мы создали объект `nash_object` и теперь можем получить доступ к его свойствам с помощью точки или квадратных скобок:

```
alert(nash_object. svoistvo);
alert(nash_object['svoistvo']);
```

Второй вариант иногда дает более гибкий способ обращения к свойствам, например, зная о том, что к свойствам можно обращаться с помощью квадратных скобок, мы можем определить, например, функцию, меняющую свойства объектов, вот так:

```
function changeValue(property, value) {
    nash_object[property] = value;
}
```

Каков бы ни был способ создания объекта, с помощью оператора `new` или литеральной нотации, в них есть существенный недостаток: результат не годится для повторного использования. Каждый новый объект нужно создавать заново, нельзя сделать просто новую версию старого объекта с новым значением какого-нибудь одного свойства.

Конструктор объекта

Для создания однотипных объектов требуется сначала заполучить объект, который бы обладал нужными чертами, но который можно было бы использовать для того, чтобы по его подобию конструировать другие объекты. В JavaScript для этой цели используется функция конструктора объекта.

Конструктор — это функция языка, отличающаяся тем, что вызывается она с помощью оператора `new`. Ей передается в виде значения ключевого слова `this` ссылка на создаваемый пустой объект. Действие конструктора состоит в наполнении созданного пустого объекта свойствами и методами.

Вот фрагмент кода, который демонстрирует сказанное:

```
function cr()
{
  this.x = 1;
  this.y = 2;
}
Obj = new cr;
```

Можно создать сколько угодно объектов, применяя конструктор. Каждый из таких объектов будет иметь свойства `x` и `y`, имеющие значения 1 и 2 соответственно.

Функции как объекты

Функция в JavaScript является объектом, как и все остальное. Функцию можно создать следующим образом:

```
function a(x) { return 1 + x; }
```

или:

```
var a = function(x) { return 1 + x; }
```

Над функцией можно выполнять те же действия, что и над другими объектами. Например, присвоить ее переменной. Во втором варианте оператор `function` создает новый объект функции, который присваивается переменной `a`.

Как и у любого объекта, у функции могут быть свойства. Допустима, например, следующая запись:

```
b.someVar = 15;
```

Функцию можно присвоить не только переменной, но и свойству объекта. Причем допустимо использовать как нотацию с точкой, так и квадратные скобки:

```
var q = {};
q["someFunc"] = b;
q.someOtherFunc = function(){}
```

Добавление методов при помощи прототипа

Мы видели, что метод можно добавить в функцию-конструктор, просто вписав его внутрь при определении. Но в JavaScript обычно используют другой путь: создание прототипа.