On-line-поддержка книги

ЕЛЕНА БЕНКЕН, ГЕННАДИЙ САМКОВ

AJAX

ПРОГРАММИРОВАНИЕ ДЛЯ ИНТЕРНЕТА

Принцип работы AJAX

XML и JSON

Объектная модель документа: DOM в JavaSript и DOM-функции в PHP

Проблема русификации Web-приложений

Использование XML и создание периодических запросов

Запрос данных с сервера MySQL

Библиотеки для создания AJAX-приложений: ExtJS, jQuery

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.

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.

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 $70\times100^{1}/_{16}$. 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

Оглавление

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).

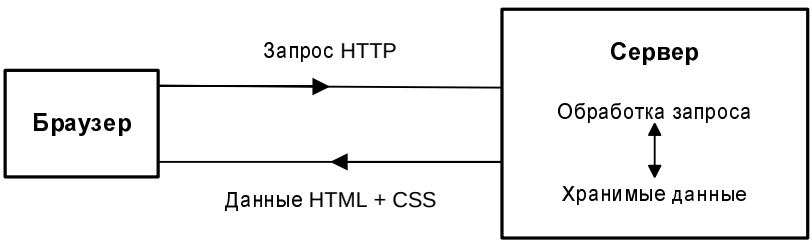

Рис. 1.1. Классическая схема работы Web-приложения

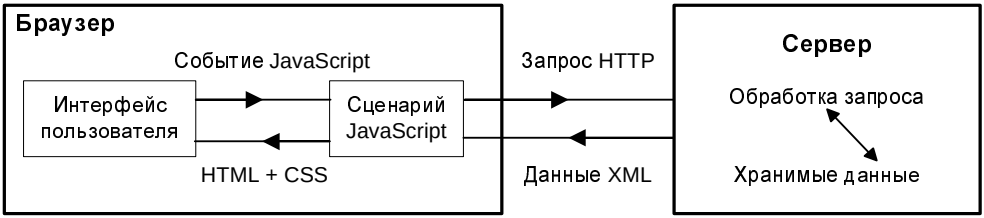

Рис. 1.2. AJAX-модель Web-приложения

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)
{
```