2 Web-приложение QuickGallery

В этой главе...

- Web-приложение QuickGallery
- Оценка классической модели Web-приложения

Prototype и Scriptaculous созданы для упрощения и ускорения разработки приложений, использующих Ajax. Основная функция Ajax — выполнение асинхронных запросов к серверу, и в предыдущей главе мы уже видели, как класс `Ajax.Request` из библиотеки Prototype позволяет упростить организацию асинхронных запросов (см. листинг 1.4). В последующих трех главах мы рассмотрим предоставляемую Prototype поддержку Ajax более подробно, сосредоточившись на ее использовании в более сложных приложениях. Мы подробно изучим класс `Ajax.Request` в главе 3; в данной главе мы подготовим сцену для изучения Ajax и ряда других возможностей, которые будем использовать в дальнейшем.

Рассматривая составные части Prototype и Scriptaculous, мы будем стараться рассмотреть и их применение в реальных приложениях средней сложности. В данной главе мы рассмотрим пример приложения, который будем развивать и улучшать на протяжении всей книги и закончим в главе 12. Это Web-приложение QuickGallery, представляющее собой галерею изображений (каждому Web-приложению требуется название, и многие из этих названий звучат не слишком умно). Версия, которую мы создадим в этой главе, будет весьма простой и ничем не примечательной. В ней не будет использоваться Ajax — она создается просто в качестве отправной точки для изучения преимуществ Ajax в главах 3 и 4. Финальная версия QuickGallery будет готова в главе 12, в которой мы добавим в это приложение целый ряд возможностей, предоставляемых Prototype и Scriptaculous.

Итак, в данной главе не ожидайте создания чего-то особенно выдающегося. Если вы спешите, можете сразу перейти к главе 3 и обращаться к данной главе, только если вам понадобится разобраться в каких-то деталях QuickGallery.

2.1. Дизайн и реализация

Нам нужно сделать две вещи. Во-первых, необходимо составить описание приложения QuickGallery, и в пользовательских, и в технических аспектах. Во-вторых, нужно определить сложности в реализации приложения согласно классической модели Web-приложений. Точнее говоря, нам нужно решить, как повлияют ограничения этой модели на наше приложение. В дальнейшем, в главах 3 и 4, это позволит нам определить, где в нашем приложении можно применить Ajax, чтобы избавиться от этих ограничений.

В настоящем разделе мы кратко опишем приложение QuickGallery. Для начала посмотрим, что наше приложение должно делать и какие требования к нему мы предъявим.

2.1.1. Требования к приложению

Приложение, над которым мы будем работать в этой книге, представляет собой программу просмотра изображений, способную отображать файлы в дереве папок сервера — в режиме предварительного просмотра и в полноэкранном режиме. Вначале наши требования достаточно тривиальны. Нам требуется возможность просмотра файлов с изображениями в папке в виде набора уменьшенных изображений. После щелчка на уменьшенном изображении соответствующий файл должен открываться в полноэкранном режиме. Кроме того, нам нужна возможность перемещаться по дереву папок. Соответственно экран приложения будет поделен на три части, как показано на рис. 2.1.

Список названий открывавшихся папок в верхней части экрана позволит нам вернуться вверх по дереву папок, а в списке в левой части экрана перечислены подпапки открытой в данный момент папки. Оставшаяся часть экрана выделена для вывода уменьшенных копий изображений из текущей папки или вывода выбранного изображения в полноэкранном режиме.

Уменьшенные копии могут быстро скачиваться и отображаться, позволяя пользователю легко просматривать все содержимое папки. Изображения, получаемые с помощью современных цифровых камер, слишком объемисты, чтобы скачивать их целиком в больших количествах, а выполнение масштабирования на лету может вызвать у браузера “несварение”. Таким образом, галерея должна создавать уменьшенные копии изображений по требованию.

Приложение работает с серверной частью, использующей язык PHP. Мы выбрали PHP из-за его простоты и распространенности. Серверная часть относительно проста, так что портировать ее на другие языки не составит особого труда. Серверный код для Ajax-приложения в основном использует те же методики, что и приложения, отличные от Ajax. Однако есть ряд отличий, и мы рассмотрим реализацию этих методик на основных серверных языках (PHP, Java, .NET и Ruby/Rails) в приложении Д.

В реальном мире существуют две категории Ajax-проектов: разработанные с использованием Ajax с нуля и созданные без Ajax, а затем модифицированные. Хотя приложение QuickGallery было написано специально для данной книги, первая его версия не использует Ajax, чтобы вы могли ознакомиться с процессом модификации классических Web-приложений под Ajax.

Итак, вначале наше приложение QuickGallery представляет собой классическое Web-приложение, в котором реакцией на действия пользователя является полное обновление страницы. Серверная часть представляет собой один PHP-файл, генерирующий представление, и файл включения в PHP, содержащий управляющую логику. Сначала мы рассмотрим управляющую логику.

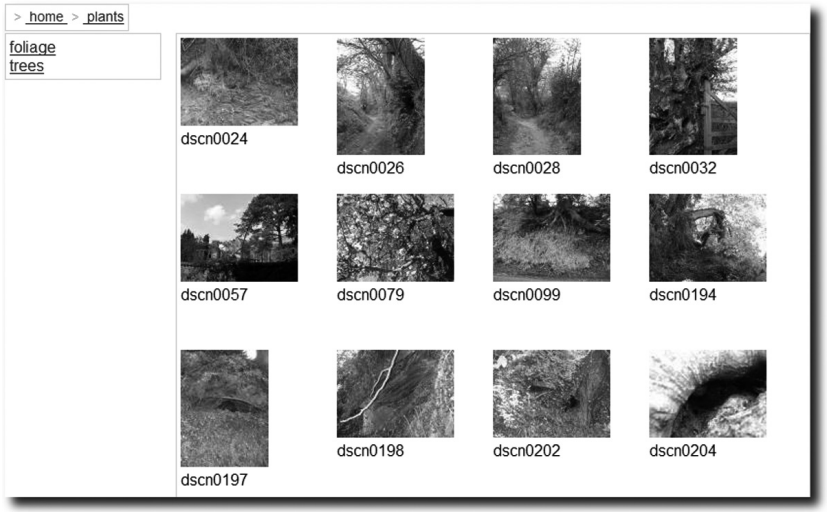


Рис. 2.1. Приложение QuickGallery, отображающее содержимое папки со списком подпапок в левой части экрана и родительских папок – в верхней части

2.1.2. Перемещение по файловой системе

Первое, что мы должны определить, — к какой части файловой системы сервера сможет обращаться наше приложение. В целях упрощения мы предположим, что все наши изображения хранятся в подпапках одной базовой папки, путь к которой мы зададим при настройке приложения. Кроме того, мы зададим неполную URL, которая будет использоваться при составлении URL файлов изображений в браузере.

Наш демонстрационный сервер представляет собой немодифицированный Apache 2.0 под Ubuntu Linux, корневая папка сервера обозначена как `/var/www/`. Мы создали папку `/var/www/dave/album`, в которой хранятся файлы QuickGallery, и добавили символьную ссылку на домашнюю папку, которая называется "сайты", — `/var/www/dave`. Итак, конфигурация нашей галереи выглядит следующим образом:

```
$basedir='/home/dave/sites/album/images';
$img_pre_path='/dave/album/images';
```

Мы будем использовать эту информацию разными способами. Для успешного перемещения по файловой системе нам нужно постоянно знать, где в дереве папок мы находимся в данный момент. Мы передаем приложению переменную запроса, указывая в ней путь относительно базовой папки. Если путь не задан, то мы начинаем с базовой папки. Код, который все это выполняет, весьма прямолинеен:

```
if (isset($_GET['path'])){
  $path=$_GET['path'];
  $fulldir=implode('/',array($basedir,$path));
}else{
  $path="";
  $fulldir=$basedir;
}
```

Функция `implode()` в PHP просто объединяет элементы массива с заданным разделителем. Переменная `$path` указывает текущую позицию в дереве папок, а переменная `$fulldir` устанавливает соответствующую позицию в реальной файловой системе.

Теперь мы знаем, где находимся, и можем генерировать навигационную информацию.

Просмотр содержимого папки

Наша первая задача — просмотреть текущую папку на наличие в ней файлов изображений и подпапок. Эту задачу решает функция `list_dir()`, код которой приведен ниже.

```
function list_dir($dir,$path){
  $dirs=array();
  $imgs=array();
  $others=array();
  $d=dir($dir);                                          <┐
  while (false != ($entry = $d->read())){                <┘ Просматриваем папку
    if (is_dir(implode('/',array($dir,$entry)))){
      if (($entry!='.') && ($entry!='..')){
        $dirs[]=$entry;                                  <┐
      }                                                  <┘ Записываем подпапку
    }else if (is_image_file($entry,$path)){
      $bits=explode('.',$entry);
      $imgs[]=$bits[0];                                  <┐
    }else{                                               <┘ Записываем файл изображения
      $others[]=$entry;
    }
  }
  $results=array(
    'dirs' => $dirs,
    'imgs' => $imgs,
    'others' => $others
  ); #4
  return $results;
}                                                        <— Собираем результаты
```

Эта функция принимает два аргумента: абсолютный путь к папке и относительный путь с точки зрения пользователя. Непосредственно в этой функции нам нужен только абсолютный путь, но относительный путь мы будем использовать позже, при генерации уменьшенных копий изображений.

Функция `list_dir()` перебирает содержимое текущей папки с помощью функций файловой системы языка PHP (1) и записывает каждый найденный элемент в одну из трех категорий: подпапка (2), файл изображения (3) или что-то другое. Затем результаты собираются в массив и возвращаются из функции (4). Мы можем проверять наличие подпапок с помощью встроенной функции PHP `is_dir()`. Заметьте, что мы явным образом игнорируем специальные папки `.` и `..`, обозначающие текущую позицию и родительскую папку соответственно. Записи в верхней части экрана приложения позволят пользователю перемещаться вверх по дереву папок, а перемещаться в текущую позицию нет смысла.

Чтобы определить, содержит ли файл изображение, мы проверим его имя. Для простоты в данном случае мы работаем только с файлами JPEG. Всю работу выполняет функция `is_image_file()`, приведенная ниже.

```
function is_image_file($entry,$path){
  $is_image=false;
  $bits=explode('.',$entry);
  $last=count($bits)-1;
  if ($bits[$last]=='jpg'){
    $is_image=($bits[$last-1]!='thumb');
    if ($is_image){
      ensure_thumbnail($bits[0],$path);
    }
  }
  return $is_image;
}
```

Функция PHP `explode()` выполняет операцию, обратную выполняемой функцией `implode()`, — разбивает на элементы массива строку с заданными символами-разделителями. Здесь мы используем ее для выделения расширения файла — если это `.jpg`, значит, это файл изображения. Из этого правила есть исключение: если окончание файла — `thumb.jpg`, то мы не считаем его файлом изображения, поскольку предполагаем, что это ранее сгенерированная уменьшенная копия изображения. Если мы убеждаемся, что нашли файл изображения, то возвращаем значение `true`, но прежде чем сделать это, проверим, готова ли уменьшенная копия этого изображения.

Вскоре мы разберемся, как создаются уменьшенные копии изображений, а сейчас закончим рассмотрение навигации по файловой системе — посмотрим, как обеспечивается возврат вверх по дереву папок.

Создание пути для возврата

Путь для возврата должен позволять пользователю быстро переходить вверх по дереву папок и предоставлять ему информацию о текущей позиции в этом дереве. Чтобы создать такой путь, нужно разбить путь к текущей папке на составные части:

```
function get_breadcrumbs($path){
  $bits=split('/',$path);
  $crumbs=array();
  $tmp_path='/';
  $crumbs[]=array(
    'name' => 'home',
    'path' => $tmp_path
  );
  foreach ($bits as $i => $value){
    if (strlen($value) > 0){
      $tmp_path.=$value.'/';
      $crumbs[]=array(
        'name' => $value,
          'path' => $tmp_path
        );
      }
    }
```

Добавляем запись о корневой папке

Добавляем запись о папке

```
  return $crumbs;
}
```

Сначала мы добавляем в верхнюю часть экрана запись о корневой папке (1). Затем перебираем элементы пути, занося каждый из этих элементов в верхнюю часть экрана (2). Для каждого элемента нам следует записать имя, которое нужно отображать, и путь, который соответствует этому элементу, т.е. совокупность всех предшествующих элементов. И наконец, мы возвращаем накопленные данные как массив массивов.

Теперь, сгенерировав все навигационные данные, мы можем заняться генерацией уменьшенных копий изображений.

2.1.3. Генерация уменьшенных копий изображений

Определив, что файл является файлом изображения, мы проверили, существует ли для него уменьшенная копия изображения. Генерация такой копии требует открытия исходного файла и выполнения операции масштабирования. По сравнению с генерацией HTML-кода это довольно интенсивная в плане нагрузки операция, и мы хотим, чтобы она выполнялась, только если уменьшенная копия не была создана ранее. Давайте посмотрим, как это делается.

```
function ensure_thumbnail($base_name,$path){
  global $basedir,$thumb_max;
  $thumb_name=join('/',
    array($basedir,$path,
    $base_name.'.thumb.jpg')
  );
  if (!file_exists($thumb_name)){
    $source_name=join('/',
      array($basedir,$path,$base_name.'.jpg')
    );
    $source_img=imagecreatefromjpeg
      ($source_name);
    $source_x=imageSX($source_img);
    $source_y=imageSY($source_img);
    $thumb_x=($source_x > $source_y) ?
    $thumb_max :
    $thumb_max*($source_x/$source_y);
  $thumb_y=($source_x < $source_y) ?
    $thumb_max :
    $thumb_max*($source_y/$source_x);
  $thumb_img=ImageCreateTrueColor($thumb_x,$thumb_y);
  imagecopyresampled(
    $thumb_img,$source_img,
    0,0,0,0,
    $thumb_x,$thumb_y,
    $source_x,$source_y
    );
    imagejpeg($thumb_img,$thumb_name);
    imagedestroy($source_img);
    imagedestroy($thumb_img);
  }
}
```

Проверяем наличие уменьшенной копии

Считываем изображение

Создаем уменьшенную копию

Копируем данные уменьшенной копии

Сохраняем уменьшенную копию

Первое, что мы сделаем в этой копии, — проверим, не существует ли уже уменьшенная копия изображения (1). Если да, мы завершаем выполнение функции — больше делать ничего не надо. Если уменьшенной копии нет, то мы создаем ее.

Здесь мы сталкиваемся с последним элементом конфигурирования программы — мы должны задать размер уменьшенных копий изображений. Не у всех изображений соотношение сторон одинаково, поэтому мы задаем величину одной стороны — большей из двух. Масштабирование выполняется так, чтобы уменьшенная копия сохранила соотношение сторон оригинала. Это гарантирует аккуратное размещение уменьшенных копий на экране в квадратных ячейках, определенных CSS (см. рис. 2.1).

Уменьшенные копии изображений генерируются с помощью функций, предоставляемых GD — модулем, ставшим фактически стандартным для манипулирования изображениями в PHP. До версии 5.2 этот модуль был опциональным, но теперь он входит в состав базовых версий дистрибутивов PHP. На нашей тестовой Linux-системе нам пришлось установить модуль php-gd отдельно, после установки Apache и PHP. Вам это, возможно, не понадобится — все зависит от того, какие дистрибутивы вы используете. В приложении Г рассказывается, как установить PHP с модулем GD на машины под управлением Windows, Linux и Mac. Если вы решите перенести приложение на какой-то другой серверный язык, то вам нужно будет разобраться с библиотеками этого языка для манипулирования изображениями, но базовые принципы останутся такими же.

Установив GD, мы можем создавать уменьшенные копии изображений следующим образом. Во-первых, откроем файл с исходным изображением (2). Во-вторых, создадим новое пустое изображение (3), размер которого соответствует размеру создаваемой уменьшенной копии. Затем выполним непосредственно масштабирование (4). И наконец, сохраним полученную уменьшенную копию в файле, заменяя окончание `.jpg` в имени исходного файла `thumb.jpg`.

Итак, мы рассмотрели все элементы, необходимые для генерации галереи. Последний элемент управляющей логики нашего приложения — объединение всех этих элементов.

2.1.4. Объединение элементов приложения

Мы знаем, как определять, где мы находимся в дереве папок, как генерировать все нужные нам навигационные данные, как генерировать уменьшенные копии изображений и как сохранять их для последующего использования. Все, что осталось сделать, — это объединить эти функции в законченное приложение. Последний фрагмент кода в нашем приложении делает именно это:

```
$crumbs=get_breadcrumbs($path);
$cc=count($crumbs);
if ($cc > 0){
  $closeup=is_close_up($crumbs[$cc-1]['name']);
}else{
  $closeup=false;
}
if ($closeup==false){
  $listings=list_dir($fulldir,$path);
  $subdirs=$listings['dirs'];
  $imgs=$listings['imgs'];
  $others=$listings['others'];
}
```

Сначала мы получаем данные, которые отображаются в верхней части экрана. Если эти данные есть, мы проверяем, просматривается ли в данный момент отдельное изображение или содержимое какой-то папки. Если мы просматриваем папку, то выводим ее содержимое.

Чтобы определить, просматриваем ли мы папку или отдельный файл изображения, проверяем, заканчивается ли последняя запись в верхней части экрана символами `.jpg`. Если да, мы предполагаем, что просматривается файл изображения. Код функции проверки приведен ниже.

```
function is_close_up($name){
  $result=false;
  $bits=explode('.',$name);
  $last=$bits[count($bits)-1];
  if ($last=='jpg'){ $result=true; }
  return $result;
}
```

Здесь для разделения пути на составные компоненты мы снова используем функцию PHP `explode()`. Эта система не является совершенной — ее, например, можно свести с ума, создав папку с расширением `.jpg`, но если мы не ставим себе такой цели, программа будет работать.

Вот и вся управляющая логика нашего приложения. Полный листинг содержимого файла включения приведен ниже. Обратите внимание на то, что пути, определенные в первых двух строках этого листинга, в вашей системе могут быть другими. Как сконфигурировать приложение QuickGallery, рассказывается в разделе 3 приложения Г.

Листинг 2.1. Управляющая логика приложения QuickGallery (файл Images.inc.php)

```
<?php
$basedir='/home/dave/sites/album/images';
$img_pre_path='/dave/album/images';
$thumb_max=120;

function list_dir($dir,$path){
  $dirs=array();
  $imgs=array();
  $others=array();
  $d=dir($dir);
  while (false != ($entry = $d->read())){
    if (is_dir(implode('/',array($dir,$entry)))){
      if (($entry!='.') && ($entry!='..')){
        $dirs[]=$entry;
      }
    }else if (is_image_file($entry,$path)){
      $bits=explode('.',$entry);
      $imgs[]=$bits[0];
    }else{
      $others[]=$entry;
    }
  }
  $results=array(
    'dirs' => $dirs,
```

```
    'imgs' => $imgs,
    'others' => $others
  );
  return $results;
}

function is_image_file($entry,$path){
  $is_image=false;
  $bits=explode('.',$entry);
  $last=count($bits)-1;
  if ($bits[$last]=='jpg'){
    //eaii?e?oai ?aiaa nicaaiiua oiaiuoaiiua eiiee
    $is_image=($bits[$last-1]!='thumb');
    if ($is_image){
      ensure_thumbnail($bits[0],$path);
    }
  }
  return $is_image;
}

function ensure_thumbnail($base_name,$path){
  global $basedir,$thumb_max;
  $thumb_name=join('/',array($basedir,$path,$base_name.'.thumb.jpg'));
  if (!file_exists($thumb_name)){
    $source_name=join('/',array($basedir,$path,$base_name.'.jpg'));
    $source_img=imagecreatefromjpeg($source_name);
    $source_x=imageSX($source_img);
    $source_y=imageSY($source_img);
    $thumb_x=($source_x > $source_y) ?
      $thumb_max :
      $thumb_max*($source_x/$source_y);
    $thumb_y=($source_x < $source_y) ?
      $thumb_max :
      $thumb_max*($source_y/$source_x);
    $thumb_img=ImageCreateTrueColor($thumb_x,$thumb_y);
    imagecopyresampled(
      $thumb_img,$source_img,
      0,0,0,0,
      $thumb_x,$thumb_y,
      $source_x,$source_y
    );
    imagejpeg($thumb_img,$thumb_name);
    imagedestroy($source_img);
    imagedestroy($thumb_img);
  }
}

function get_breadcrumbs($path){
  $bits=split('/',$path);
  $crumbs=array();
  $tmp_path='/';
```

```
  $crumbs[]=array(
    'name' => 'home',
    'path' => $tmp_path
  );
  foreach ($bits as $i => $value){
    if (strlen($value) > 0){
      $tmp_path.=$value.'/';
      $crumbs[]=array(
        'name' => $value,
        'path' => $tmp_path
      );
    }
  }
  return $crumbs;
}

function is_close_up($name){
  $result=false;
  $bits=explode('.',$name);
  $last=$bits[count($bits)-1];
  if ($last=='jpg'){ $result=true; }
  return $result;
}

if (isset($_GET['path'])){
  $path=$_GET['path'];
  $fulldir=implode('/',array($basedir,$path));
}else{
  $path="";
  $fulldir=$basedir;
}
$crumbs=get_breadcrumbs($path);
$cc=count($crumbs);
if ($cc > 0){
  $closeup=is_close_up($crumbs[$cc-1]['name']);
}else{
  $closeup=false;
}
if ($closeup==false){
  $listings=list_dir($fulldir,$path);
  $subdirs=$listings['dirs'];
  $imgs=$listings['imgs'];
  $others=$listings['others'];
}
?>
```

После выполнения этого файла включения у нас будут все данные, необходимые для заполнения HTML-файла. Собственно говоря, выделив управляющий код в независимый файл, мы немало выиграли. Этот файл никак не связан с представлением данных в Web-браузере, поэтому его несложно будет применить в Ajax-приложении, в котором вместо готовых HTML-страниц мы будем отправлять браузеру чистый текст, HTML,

XML или еще какие-то типы данных. Мы еще вернемся к этому коду в главах 3 и 4, но не будем вносить в него никаких особо серьезных изменений.

Однако другую часть приложения, связанную с представлением данных, нам придется модифицировать гораздо серьезнее. В следующем разделе мы посмотрим, как из данных, сгенерированных нашим управляющим кодом, создаются HTML-страницы.

2.1.5. Создание HTML-страниц

Итак, файл включения выполняет для нас солидную часть работы, предоставляя все данные, с которыми должно работать приложение. Задача генерации из этих данных HTML-кода сравнительно проста — нам достаточно определить несколько тегов `<DIV>` для размещения основных элементов пользовательского интерфейса, расположить эти теги на странице с помощью таблиц CSS и просмотреть массивы данных, которые сгенерировал файл включения.

Листинг 2.2. Генерация HTML-кода в приложении QuickGallery (файл `index.php`)

```
<?php
require('images.inc.php');
?>
<html>
<head>
<link rel='stylesheet' type="text/css" href="images.css">
</head>
<body>

<div id='title' class='box'>
<?php foreach ($crumbs as $i => $value){ ?>
  &amp;nbsp;&amp;gt;&amp;nbsp;
  <a href="images.php?path=<?php echo $value['path'] ?>">
  <?php echo $value['name'] ?>
  </a>
<?php } ?>
</div>

<?php if ($closeup){ ?>

<div id='closeup' class='box'>
<img src='<?php echo $img_pre_path.$path ?>'>
</div>
<?php }else{ ?>

<?php if (count($subdirs)>0){ ?>
<div id='folders' class='box'>
<?php foreach ($subdirs as $i => $value){ ?>
<div>
  <a href="images.php?path=<?php
    echo implode('/',array($path,$value))
  ?>">
    <?php echo $value ?>
  </a>
```

Включаем управляющую логику

Перебираем родительские папки

Выводим изображение на экран

Выводим содержимое папки на экран

Перебираем подпапки

```
</div>
<?php } ?>
</div>
<?php } ?>

<?php if (count($imgs)>0){ ?>
<div id='images' class='box'>
<?php foreach ($imgs as $i => $value){
  $full_img=implode('/',array($path,$value));
?>

<div class='img_tile'>
  <a href="images.php?path=<?php echo $full_img ?>.jpg">
  <img border='0'
    src="<?php echo $img_pre_path.$full_img ?>.thumb.jpg"/>
  </a>
  <br/>
  <?php echo $value ?>
  </a>
</div>
<?php } ?>
</div>
<?php } ?>

<?php } ?>

</body>
</html>
```

Перебираем файлы изображений

Генерируем файл изображения

В файле управляющей логики мы работали с единым, цельным блоком PHP. Здесь мы работаем с набором небольших фрагментов PHP и HTML, который должен быть знаком работавшим с JSP или ASP.

Первое, что мы делаем, — включаем файл с управляющей логикой (1). Затем перебираем родительские каталоги, список которых нужно отображать в любом случае (2). Далее нужно определить, будет ли отображаться одно изображение в полномасштабном режиме (3) или все изображения в папке (4). Если последнее, мы перебираем подпапки (5) и файлы изображений (6), выводя каждое изображение в отдельном квадратике в уменьшенном виде (7).

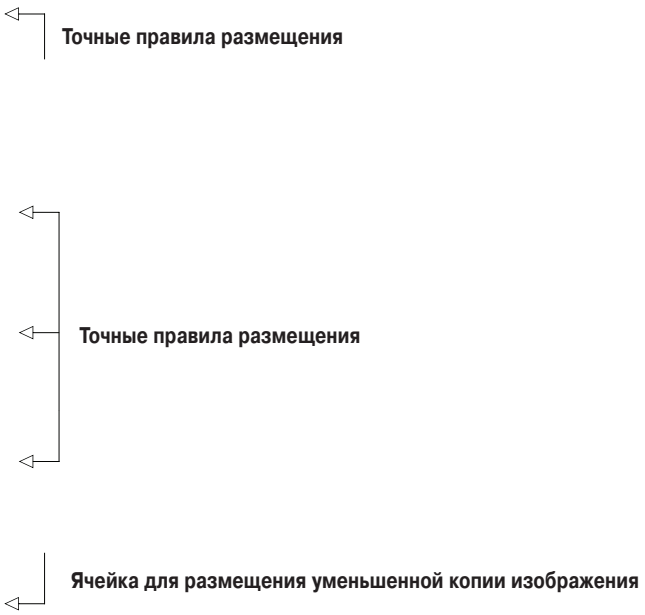
Это практически структурное определение документа. Другими словами, мы только что описали отношения "родитель–потомок" между всеми элементами страницы. Например, уменьшенные копии изображений мы помещаем в элемент с ID `images`. В коде HTML мы не определяем особо подробно, как должно выглядеть наше приложение на экране, — этим должна заниматься таблица стилей CSS. Содержимое файла CSS приведено в листинге 2.3.

Листинг 2.3. Таблица стилей CSS для приложения QuickGallery (файл images.css)

```
*{
  font-face: Arial, Helvetica;
}
.box{
```

Общие элементы стиля

```
  position: absolute;
  border: solid #adf 1px;
  background-color: white;
  padding: 4px;
}
#title{
  top: 5px;
  left: 5px;
  height: 18px;
  font-size: 14px;
  color: #8af;
}
#folders{
  top: 35px;
  left: 5px;
  width: 150px;
}
#images{
  top: 35px;
  left: 180px;
}
#closeup{
  top: 35px;
  left: 180px;
}
.img_tile{
  float: left;
  width: 160px;
  height: 160px;
}
```



Именно эта таблица стилей в основном и определяет внешний вид нашего приложения. Каждый из основных элементов страницы получает в CSS класс `box`, определяющий простые стили границ и, что более важно, координаты расположения на экране (1). Кроме того, разным элементам в CSS соответствуют различные дополнительные правила с разными ID (2). Эти правила определяют расположение соответствующих элементов на странице. В конце таблицы стилей определено правило, касающееся уменьшенных копий изображений (3), использующее свойство CSS `float` для аккуратного размещения этих копий на экране независимо от размеров окна браузера.

PHP-страница, генерирующая HTML-код (см. листинг 2.2), и таблица стилей (см. листинг 2.3) определяют внешний вид нашего приложения — единую монолитную страницу. Если мы захотим изменить что-то на этой странице, то сгенерируем ее заново и с нуля выполним рендеринг всех ее элементов на экране. Таков принцип работы простейших Web-приложений — и этот принцип был практически единственным, существовавшим до появления Ajax. В следующем разделе мы рассмотрим этот момент подробнее.

На этом заканчивается наш обзор классического Web-приложения QuickGallery в варианте, не использующем Ajax. В этом приложении нет ничего особо замысловатого или необычного, но оно дает нам отправную точку для применения Ajax. Кроме того, стоит заметить, что нам пришлось приложить определенные усилия, чтобы разделить наше приложение на ряд компонентов, а именно: отделить управляющую логику от

представления (файла HTML) и таблицы стилей CSS. Как мы вскоре убедимся, такое разделение очень нам пригодится, когда мы попытаемся переделать приложение с использованием Ajax. В следующем разделе мы посмотрим, как может повлиять на структуру приложения использование Ajax и что мы сможем выиграть от этого использования.

2.2. Оценка классической модели Web-приложений

Как уже говорилось в начале этой главы, наше знакомство с Ajax мы начнем с того, что создадим простое приложение, не использующее Ajax, чтобы получить возможность оценить его недостатки. Это позволит понять, где нам может пригодиться Ajax, чтобы мы не "применяли Ajax потому, что это Ajax". Теперь, рассмотрев приложение QuickGallery, посмотрим, как на его структуру повлияла классическая модель функционирования Web-приложений.

2.2.1. Ссылки, формы и обновление страниц

Как говорилось в предыдущем разделе, пользовательский интерфейс классического Web-приложения представляет собой цельную HTML-страницу. На этой странице может применяться ряд вспомогательных элементов, например рисунки или таблицы стилей CSS, но конечным результатом использования этих элементов будет именно цельная, монолитная страница.

Пользователь может взаимодействовать с этой страницей двумя способами: щелкая на гиперссылках и заполняя формы. Оба способа сводятся к генерации запросов к серверу и последующему ожиданию реакции на эти запросы. Реакция должна представлять собой новую цельную страницу. Как мы заметили в главе 1, это приведет к перерывам в активности пользователя, который вынужден ожидать, пока загрузится новая страница. Графически эта ситуация продемонстрирована на рис. 2.2.

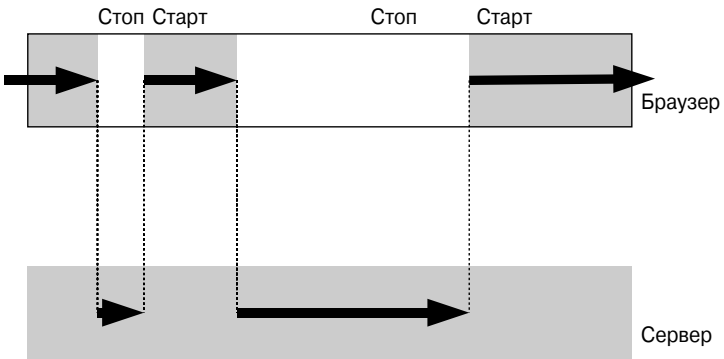


Рис. 2.2. Прерывистая последовательность взаимодействия пользователя с Web-приложением классической модели

Время на этой картинке откладывается по горизонтали, слева направо. Верхняя полоска изображает Web-браузер — пользователь может работать с ним, только когда сервер завершил обработку запроса и ничего не делает. А пока пользователь работает

с приложением, сервер ничего не делает — в это время отображается уже отправленная браузеру страница, которая, возможно, уже устарела в результате действий пользователя. Чем интерактивнее приложение, тем чаще будут встречаться периоды, в течение которых серверу нечего делать.

Теперь, поняв, в чем суть проблемы, рассмотрим, как она проявляет себя в QuickGallery.

2.2.2. Классическая модель Web-приложений и QuickGallery

С приложением QuickGallery мы взаимодействуем только щелчками на гиперссылках. Мы взаимодействуем с сервером двумя способами: перемещаясь между папками (с помощью списка подпапок или родительских папок) или щелкая на уменьшенной копии изображения, чтобы просмотреть его оригинал. Рассмотрим каждый из этих способов по очереди.

Перемещения между папками

Изображения, отображаемые в галерее, располагаются в древовидной структуре, соответствующей структуре папок файловой системы сервера. Мы перемещаемся по этой структуре с помощью списка родительских папок в верхней части окна приложения и списка подпапок в левой части этого окна. Всего в окне приложения три области, и с какой бы из областей мы ни работали, все три области всегда обновляются одновременно, как показано на рис. 2.3.

В правой части этого рисунка затенены области, содержимое которых изменяется (элементы с белым цветом фона не изменяются). В данном случае изменилось почти все содержимое окна, так что обновление всей страницы не будет слишком неэффективным решением. Единственное исключение — список родительских папок. В зависимости от того, насколько далеко вверх или вниз по дереву папок мы переместимся, будет изменяться бо́льшая или меньшая часть содержимого верхней части окна. На рис. 2.3 области, которые будут обновлены в ответе сервера, но будут содержать ту же самую информацию, что и раньше, заштрихованы. Это области, представляющие собой потенциальные проблемы при обновлении страниц целиком, — их можно было бы обновлять частично, экономя сетевой трафик и процессорное время, необходимое для перерисовки экрана. А теперь посмотрим, что произойдет, когда мы щелкнем на уменьшенной копии изображения, чтобы увидеть полноразмерный оригинал.

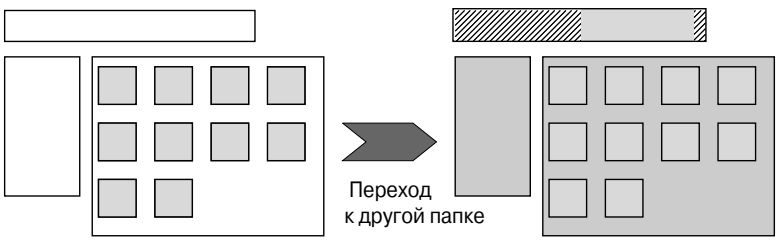


Рис. 2.3. Влияние перехода к другой папке на информацию, отображаемую приложением QuickGallery

Просмотр оригинала изображения

Когда мы просматриваем оригинал изображения, он отображается вместо набора уменьшенных копий изображений. Списки подпапок и родительских папок остаются неизменными, просто в списке родительских папок отображается имя открытого файла изображения. Это позволит нам вернуться к просмотру уменьшенных копий, щелкнув на имени папки в списке родительских папок. На рис. 2.4 показаны изменения в приложении, связанные с просмотром оригинала изображения и возвратом к набору уменьшенных копий.

При переходе к просмотру оригинала мы выполняем ненужное обновление списков подпапок и родительских папок (мы не переходили в другую папку — открыли лишь один из файлов в текущей). Следовательно, обновление всей страницы — отнюдь не оптимальный вариант. А что еще важнее, когда мы возвращаемся к просмотру уменьшенных копий, мы заново загружаем всю страницу, которую пользователь уже видел, прежде чем открыл для просмотра какой-то из файлов (разумеется, если содержимое папки не изменилось со времени предыдущего к ней обращения). Единственное исключение из этого правила — переход из папки в одну из ее подпапок прямо в режиме просмотра оригинала изображения. Невозможность сохранения информации о предыдущем состоянии приложения дорого обходится нам в данном случае — и с точки зрения затрат трафика, и с точки зрения загрузки процессора.

Как видите, недостатки классической модели Web-приложений становятся критичными, если нам нужно создать высокоинтерактивные приложения, быстро реагирующие на действия пользователя. Наше приложение, QuickGallery, тоже страдает от этих недостатков. Пути их устранения мы обсудим в главах 3 и 4, когда применим Ajax в QuickGallery.

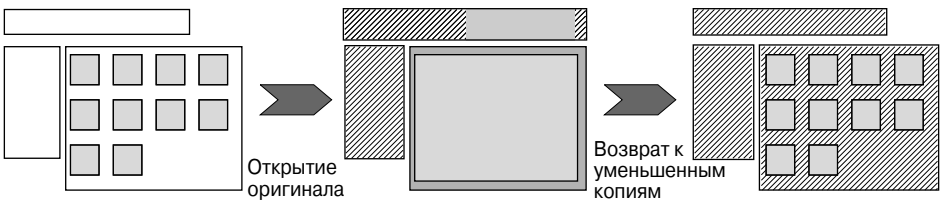


Рис. 2.4. Влияние перехода между просмотрами уменьшенных копий и оригиналов на информацию, отображаемую приложением QuickGallery

2.3. Резюме

В этой главе мы рассмотрели наше приложение-галерею в варианте, не использующем Ajax. Данный вариант послужит базой для последующих усовершенствований. QuickGallery — это простое приложение, которое делает то, что должно делать, максимально простым способом. Мы рассмотрели классическую модель создания Web-приложений и убедились, что наше приложение, как и большинство других, страдает от определенных недостатков этой модели.

Применение Ajax для устранения этих недостатков на первый взгляд может показаться рискованным предприятием, но с помощью Prototype и Scriptaculous мы справимся с ним без чрезмерных усилий. Мы приступим к этой операции в следующей главе, перейдя от обновлений страницы к взаимодействию с сервером с помощью Ajax.