9 Перетаскивание

В ЭТОЙ ГЛАВЕ...

- Основы перетаскивания
- Списки перетаскивания
- Перетаскивание с помощью ICEfaces

Все мы знакомы с концепцией "перетаскивания" в настольных приложениях; фактически, перетаскивание — основной фактор придания духа реальности нашим взаимодействиям с компьютерами. ("Мои данные — это не просто набор нулей и единиц — у меня имеются файлы и папки, которые можно перемещать одним лишь прикосновением; они вполне реальны.") Вероятно, наиболее знакомое приложение — проводник файлов. Пользователям предоставляются пиктограммы файлов и папок; пользователи могут открывать папки, чтобы их содержимое отображалось в окне и — это представляет наибольший интерес для разработчиков — они могут перетаскивать пиктограммы (нажимая кнопку мыши и перемещая мышь) из одного места и оставлять их (отпуская кнопку мыши) в другом. Это очень удобный способ сказать: "С этим нужно сделать то-то и там-то". Другие технологии интерфейсов пользователей (такие как щелчок на кнопке или выбор меню), как правило, позволяют лишь сказать: "С этим нужно сделать то-то". Взгляните на рис. 9.1.

Мы пояснили технологию перетаскивания в понятиях примитивных событий мыши, но это вовсе не означает, что она точно так же выглядит для разработчика настольного приложения. Напротив, как правило, она проявляется в виде операции передачи данных посредством абстрактных событий перетаскивания и в виде концепции данных, перемещаемых из одного объекта в другой. Иными словами, перетаскивание часто тесно связано с операциями копирования и вставки. В этой главе мы не станем изучать перетаскивание слишком глубоко, но мы ознакомимся с событиями

перетаскивания, которые достаточно абстрактны, чтобы с их помощью можно было строить некоторые очень интересные приложения.

Удивление вызывает тот факт, что несмотря на существование перетаскивания с момента появления графических интерфейсов настольных ПК, множество приложений все еще не находит ему должного применения. Иначе говоря, технологии, с которыми мы ознакомимся в этой главе, предлагают возможность создания Ajax-приложений, которые действительно превосходят свои настольные аналоги. Как и можно было предположить, реализация перетаскивания в браузере требует использования определенной "магии" JavaScript. К счастью, это та часть JavaScript, которую различные альтруисты уже реализовали. Поэтому приступим и посмотрим, какие каркасы перетаскивания JavaScript доступны для использования.

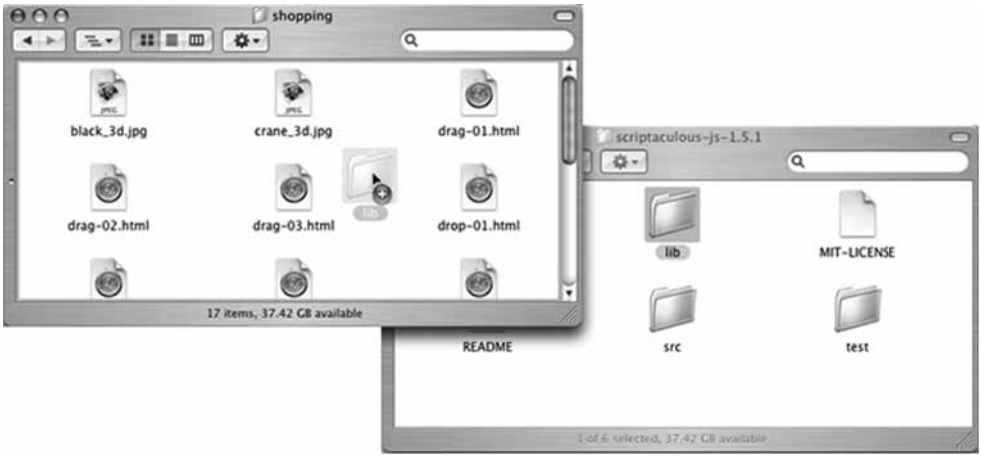

Рис. 9.1. Перетаскивание в настольном приложении

9.1. Каркасы перетаскивания JavaScript

Существует ряд реализаций перетаскивания JavaScript с открытым исходным кодом, но важно выбрать ту реализацию, которая активно поддерживается (чтобы иметь уверенность в ее надежной работе во всех современных браузерах) и обладает API-интерфейсом, успешно взаимодействующим с Ajax. Двумя наиболее популярными каркасами являются OpenRico (`http://openrico.org`) и Script.aculo.us (`http://script.aculo.us/`).

OpenRico можно применять для создания сложных Internet-приложений, и он предоставляет полную поддержку Ajax, возможности управления посредством перетаскивания и библиотеку кинематических эффектов. Script.aculo.us также предоставляет кинематические визуальные эффекты и программный интерфейс приложений для выполнения перетаскивания. Одно из действий, которое они не позволяют выполнить — перетаскивание между другими приложениями и браузером. Несмотря на большие возможности, предоставляемые этими библиотеками, любые перетаскиваемые элементы строго ограничены окном браузера.

Использование обоих названных программных интерфейсов приложений в значительной мере аналогично. Для любого из них объекты, доступные для перетаскивания, и элементы, в которых их можно помещать — это просто элементы `<div>` HTML. В HTML-коде нам требуется всего лишь нечто наподобие следующего фрагмента:

```
<div id="dragster">
  Перетащите это
</div>

<div id="dropster">
  Оставьте где-то здесь
</div>
```

Но пока, ни один элемент не доступен для перетаскивания. Вначале элементы `<div>` необходимо зарегистрировать с помощью выбранного каркаса, передав в него их идентификаторы. При использовании OpenRico эта регистрация выглядела бы так:

```
<script type="text/javascript">
  dndMgr.registerDraggable(
    new Rico.Draggable(
      "test-draggable", "dragster"));
  dndMgr.registerDropZone(new Rico.Dropzone("dropster"));
</script>
```

А при использовании Script.aculo.us:

```
<script type="text/javascript">
  new Draggable("dragster")
  Droppables.add("dropster");
</script>
```

Как видите, оба каркаса очень просты в использовании. Поскольку технологии применения одного каркаса могут быть легко адаптированы для другого, а Script.aculo.us несколько лучше сочетается с браузерами, в примерах этой главы мы будем использовать Script.aculo.us. Теперь посмотрим, как внедрить Ajax в созданные нами перетаскиваемые (т.е. доступные для перетаскивания) и пригодные для помещения перетаскиваемых элементов объекты.

9.2. Перетаскивание с использованием Ajax

Понятно, что мы не собираемся реализовывать функциональные возможности перетаскивания в приложении JavaScript с нуля. Используя такие библиотеки, как Script.aculo.us, можно гарантировать не только правильность работы функций перетаскивания в приложении, но и переносимость приложения между различными браузерами. Для нас задача сводится к увязке функций перетаскивания с Ajax. Без Ajax мы можем предоставить пользователям лишь не слишком функциональную игрушку: они могут перетаскивать те или иные элементы в среде используемого браузера и восторгаться новизной этих действий, но полностью лишены возможности поделиться результатами своей работы с другими пользователями или воздействовать на реальные данные, хранящиеся на сервере. Чтобы приложение стало реальным, события перетаскивания нужно связать с Ajax-обращениями к серверу. Посмотрим, как это можно сделать, на примере тележки для покупок, использующей каркас Script.aculo.us.

9.2.1. Тележка для покупок с поддержкой перетаскивания Ajax

Перетаскивание позволяет предоставить пользователям интуитивно понятную тележку для покупок, поскольку они получают возможность перетаскивать элементы, присутствующие на экране, в тележку для покупок. Как показано на рис. 9.2, для при-

обретения будут доступны три элемента: две книги и обычный камень. Если пользователь не может приобрести конкретный элемент (по причине недостачи денег или, как в нашем случае, поскольку камень не является книгой), тележка отклонит элемент, возвращая его в исходную позицию.

Концепция проста, но нам придется с помощью Ajax вдохнуть жизнь в ряд событий интерфейса пользователя: `onHover`, `onDrop` и `revert`. Поскольку наше приложение имеет дело с только одним типом объектов, который можно перетаскивать (элементами для приобретения), и только одним местом, где объекты можно оставлять, указанные события имеют следующий смысл.

- `onHover`. Пользователь знакомится с элементом; мы можем также призывать его к покупке данного элемента. Это событие может также указывать, что элемент находится в процессе удаления из тележки для покупок.
- `onDrop`. Пользователь выбрал элемент для приобретения.
- `revert`. Функция обратного вызова `revert` предоставляет приложению возможность указать, должен ли объект быть оставлен в выбранном месте или возвращен туда, откуда он поступил. Если элемент отклонен из тележки для покупок, `revert` возвращает значение `true`.

Итак, мы определили перетаскиваемые элементы (книги и камень), области, пригодные для оставления (только тележка для покупок), и события (`onHover`, `onDrop` и `revert`). Теперь рассмотрим реализацию.

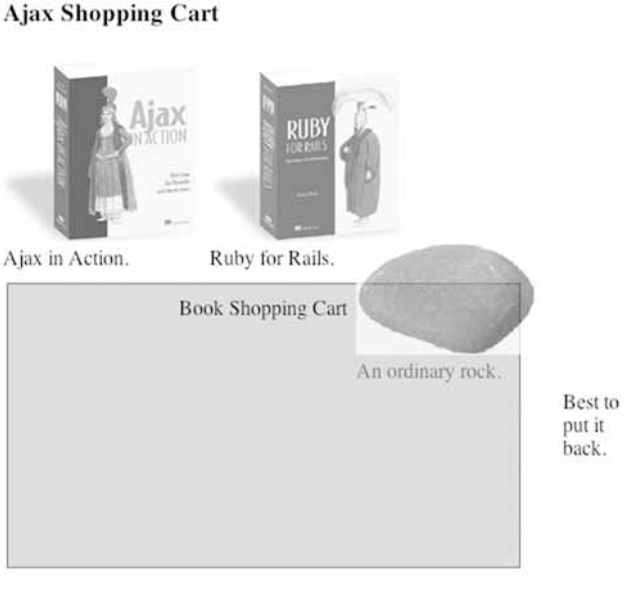

Рис. 9.2. Тележка для покупок с поддержкой перетаскивания

Проблема

События перетаскивания браузера должны быть переданы серверу.

Решение

Чтобы понять этот пример, вначале нужно рассмотреть HTML-код. Как только мы в нем разберемся, роль JavaScript станет ясна. Приведенный ниже HTML-код содержит ряд основных фрагментов:

- загрузку библиотеки JavaScript;
- определения стилей;
- элементы для доступных для перетаскивания и оставления областей;
- элемент для сообщения о состоянии;
- регистрацию пригодной для оставления области с помощью каркаса Script.aculo.us.

Следует иметь в виду только еще один нюанс: каждая доступная для перетаскивания область будет регистрироваться с помощью Script.aculo.us при ее определении. Если бы мы не нуждались в выполнении каких-то Ajax-запросов и собирались всего лишь предоставить пользователю возможность перетаскивания объектов в окне браузера без отправки какой-либо информации серверу, этим можно было бы ограничиться. Как легко предположить, мы применим Ajax, когда дело дойдет до создания кода JavaScript для этого примера. Мы начнем с рассмотрения HTML-кода, приведенного в листинге 9.1.

Листинг 9.1. HTML-код тележки для покупок

```
<html>
<head>
<script type="text/javascript" src="lib/prototype.js"></script>
<script type="text/javascript" src="lib/scriptaculous.js"></script>
</head>
<style type="text/css">
  div.carthoverclass {
    border:1px solid blue;
  }
  div.cart {
    z-index:100;
    text-align:center;
    height:200px;
    padding:10px;
    background-color:#abf;
  }
</style>
<body>
  <h3>Ajax Shopping Cart</h3>
  <table><tr>
    <td>
      <div alt="Product1" id="product_1"
        itemid="01" style="z-index:500" />
        <img alt="Product1" src="crane_3d.jpg" />
        <br />
        Ajax in Action.
      </div>
```

Загружает библиотеки JavaScript

Указывает стиль выделения рамки тележки

Указывает стиль тележки для покупок

Определяет первый доступный для перетаскивания элемент

```
      <script type="text/javascript">
        new Draggable('product_1',
          {revert:handleRevert});
      </script>
    </td>

    <td>
      <div alt="Product2" id="product_2"
          itemid="02" style="z-index:500" />
        <img alt="Product2" src="black_3d.jpg" />
        <br />
        Ruby for Rails.
      </div>
      <script type="text/javascript">
      new Draggable('product_2', {revert:handleRevert});
      </script>
    </td>

    <td>
      <div alt="Product3" id="product_3"
          itemid="03" style="z-index:500" />
        <img alt="Product3" src="rock.jpg" />
        <br />
        An ordinary rock.
      </div>
      <script type="text/javascript">
      new Draggable('product_3', {revert:handleRevert});
      </script>
    </td>
  </tr></table>

  <table><tr>

    <td width="400px">
      <div id="cart" class="cart" >
        Shopping Cart
      </div>
    </td>

    <td width="25">
    </td>

    <td width="50">
      <span id="cartinfo"></span>
    </td>

  </tr></table>

<script type="text/javascript">
  cartinfoDiv = $("cartinfo");
  Droppables.add('cart',
    { hoverclass:'carthoverclass',
      onHover:
      function(dragged, dropon, event) {
        handleHover(dragged, dropon,
          event, cartinfoDiv);
      }
```

❶ Регистрирует первый доступный для перетаскивания элемент

Определяет второй доступный для перетаскивания элемент

Определяет третий доступный для перетаскивания элемент

Определяет область, доступную для оставления элемента

Определяет область сообщения о состоянии

❷ Регистрирует область тележки для покупок, доступную для оставления элемента

```
      onDrop: :
      function(dragged, dropon, event) {
        handleDrop(dragged, dropon,
          event, cartinfoDiv);
      }
    }
  )
</script>

</body>
</html>
```

❷ **Регистрирует область тележки для покупок, доступную для оставления элемента**

Мы передаем `id` доступного для перетаскивания элемента `<div>` каркасу Script.aculo.us ❶ и регистрируем функцию обратного вызова для события `revert`, чтобы можно было управлять тем, должен ли элемент оставаться в тележке или же возвращаться на место. Мы также передаем Script.aculo.us `id` элемента `<div>` доступной для оставления элементов тележки ❷ и регистрируем анонимные функции обратного вызова `onDrop` и `onHover`, чтобы можно было передать `id` области сообщения о состоянии.

Как видите, доступные для перетаскивания и оставления элементы `<div>` могут содержать все что угодно — текст, изображения и т.п. — поэтому, чтобы объекты выглядели нужным образом, достаточно создать соответствующий HTML-код. Как только элементы `<div>` созданы, их нужно всего лишь зарегистрировать с помощью Script.aculo.us в качестве перетаскиваемых или доступных для оставления, и Script.aculo.us будет их анимировать соответствующим образом при перетаскивании пользователем с помощью мыши. Конечно, мы регистрируем также ряд функций обратного вызова, являющихся обработчиками событий перемещения перетаскиваемого элемента над областями оставления (`onHover`) или его оставления в этой области (`onDrop`). Мы зарегистрировали также функцию обратного вызова `revert`, которая позволяет возвращать значение `true` или `false` для указания того, должен ли элемент оставаться в тележке для покупок или быть возвращен в исходную позицию. Какова же будет роль Ajax в этих функциях обратного вызова? Две основных функции Ajax в сценарии — загрузка новых сообщений о состоянии в зависимости от того, как пользователь переместил перетаскиваемый элемент (например, выражение благодарности за покупку при оставлении элемента в тележке), и загрузка состояния возврата перетаскиваемого элемента (камень не может быть приобретен, поэтому он должен быть возвращен в исходное положение). Остальная часть сценария посвящена всего лишь проверке правильности доставки событий интерфейса пользователя — и это оказывается несколько затруднительным для событий "плавающего перемещения" элемента над областью оставления. Результат работы HTML-кода показан на рис. 9.2, а сам HTML-код приведен в листинге 9.2.

Листинг 9.2. Код JavaScript тележки для покупок

```
<script type="text/javascript">
function AjaxHTML(url, target) {
  AjaxOperation(url, function(req) {replaceHTML(req, target);});
}

function AjaxOperation(url, func, asynch) {
  var req;
```

Обновляет HTML-код посредством Ajax-запроса

```
  if (window.XMLHttpRequest) {
    req = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    req = new ActiveXObject("Microsoft.XMLHTTP");
  }
  if (req) {
    req.onreadystatechange =
      function (){onReady(req, func);};
    req.open("GET", url, true);
    req.send("");
  }
}

function onReady(req, func) {
  if (req.readyState == 4) {
    if (req.status == 200) {
      func(req);
    }
  }
}

function replaceHTML(req, target) {
  var content = req.responseText;
  target.innerHTML = content;
}

var revertFlag = true;
function checkRevert(req) {
  var content = req.responseText;
  revertFlag = (1 == (content - 0));
}

var lastHover;
function handleHover(dragged, dropon, event, target) {
  itemid = dragged.getAttribute("itemid");
  if (lastHover == itemid) {
    return;
  }
  lastHover = itemid;
  AjaxOperation("revert-" + itemid + ".txt",
    function(req) {checkRevert(req)});
  AjaxHTML("hover-" + itemid + ".html", target);
}

function handleDrop(dragged, dropon, event, target) {
  lastHover = null;
  itemid = dragged.getAttribute("itemid");
  AjaxHTML("drop-" + itemid + ".html", target);
}

function handleRevert(dragged) {
  return revertFlag;
}
</script>
```

❶

❷

Дожидается полностью асинхронного ответа

Обновляет HTML-код на основе Ajax-ответа

❸

❹

❺

Асинхронно отображает состояние "плавающего перемещения" посредством функций Ajax

❻

Асинхронно отображает состояние оставления элемента посредством функций Ajax

Возвращает флаг revert для Script.aculo.us

Обсуждение

Теперь, после просмотра кода, несколько подробнее остановимся на некоторых из его наиболее интересных фрагментов. Поскольку нам требуется быстро выполнить несколько Ajax-запросов, при создании функции обратного вызова ❷ мы поддерживаем отдельные объекты запроса ❶. Это предотвращает их взаимную перезапись. Важно также отметить, что флаг `revert` кодируется значением `0` для `false` и `1` для `true`. В этом фрагменте ❸ мы извлекаем его из Ajax-ответа и преобразуем в булевское значение JavaScript. Имейте также в виду, что каждое перемещение мыши генерирует событие "плавающего перемещения", поэтому мы отфильтровываем ❹ все такие события, кроме первого. После фильтрации мы синхронно извлекаем флаг `revert` ❺ перед загрузкой сообщения о событии "плавающего перемещения". Это делается для предотвращения взаимного влияния двух запросов. И, наконец, мы сбрасываем значение `lastHover` ❻. Это гарантирует полное выполнение функции `handleHover()` при последующем ее вызове. Это необходимо, поскольку сразу после оставления элемента пользователь может перетащить его снова.

В основном код JavaScript, приведенный в листинге 9.2, просто отвечает на события и с помощью Ajax-запросов загружает данные обновления страницы с сервера. Определенные проблемы связаны только с функцией `handleHover()`. Первая из них состоит в том, что `handleHover()` вызывается не просто в начале состояния "плавающего перемещения" элемента. Она вызывается для каждого перемещения мыши во время зависания перетаскиваемого элемента над областью оставления. Существует некоторая вероятность того, что возникнет желание передавать по сети любое перемещение мыши, но это маловероятно. Поэтому мы добавили функцию для обеспечения обработки только первого события "плавающего перемещения". Это достигается отслеживанием перетаскиваемого элемента, который в текущий момент находится в состоянии "плавающего перемещения". Если им является тот же элемент, что и в последний раз, функция `handleHover()` осуществляет возврат без дальнейшей обработки.

Вторая проблема связана с тем, что в функции `handleHover()` нужно выполнять две операции: обновлять отображаемую информацию соответствующим сообщением, призывающим пользователя произвести покупку, и проверять, должен ли элемент быть возвращен из тележки для покупок. Часто Ajax-приложения могут ограничиваться работой с единственным глобальным объектом запроса, но в данном случае необходимо гарантировать правильную обработку как флага `revertFlag`, так и сообщения для состояния "плавающего перемещения". Решение заключается в передаче объекта `XMLHttpRequest` в качестве параметра, чтобы каждый отдельный запрос мог действовать независимо от других.

Реальное приложение выполнения покупок, скорее всего, будет предоставлять для выбора более трех элементов, и, вероятно, эти элементы будут генерироваться динамически. К счастью, решение задачи сводится к включению кода регистрации вслед за каждым элементом:

```
<script type="text/javascript">
  new Draggable('product_1', {revert:handleRevert});
</script>
```

9.2.2. Манипулирование данными в списках

Базовые примитивы перетаскивания, такие как перетаскиваемые элементы и области оставления, предлагают очень большие возможности, поэтому легко предста-

вить себе, как можно было бы строить разнообразные приложения. Например, как быть при наличии ряда элементов, скажем фруктов и овощей, которые нужно организовать в виде двух списков: по типу (фрукт или овощ) и по предпочтениям (по их порядку следования в списках). В этом случае имеет смысл организовать интерфейс перетаскивания. Все элементы должны быть перетаскиваемыми (чтобы их можно было перемещать по странице), а оба списка должны быть доступными для оставления элементов. При перетаскивании элемента из одного списка в другой, он удаляется из первого списка и добавляется во второй. При перетаскивании элемента внутри списка другие элементы должны сдвигаться на другую позицию, чтобы элемент можно было оставить к конкретной позиции. Реализация этих возможностей требовала бы действительно сложного фрагмента кода JavaScript в обработчике события `onHover`. К счастью, Script.aculo.us предоставляет встроенные примитивы для манипулирования списками посредством перетаскивания. Все что от нас требуется, это создать списки в HTML-коде, используя дескрипторы списков HTML, а затем указать библиотеке Script.aculo.us, какие элементы могут использоваться в каждом из списков и что делать при изменении списков. При изменении списков у нас появляется повод применить Ajax и вызвать приложение на сервере, как показано на рис. 9.3.

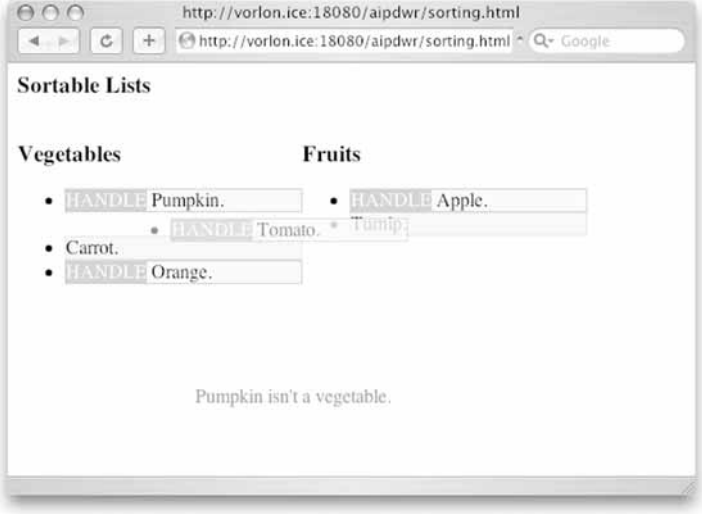

Рис. 9.3. Проверка принадлежности к овощам

При решения этой задачи мы будем лишь информировать пользователя о первом неверно помещенном фрукте или овоще. Поскольку серверу нужно передавать сравнительно сложную структуру (два упорядоченных списка), для выполнения маршализации данных мы используем DWR. Применение двух таких мощных каркасов, как Script.aculo.us и DWR, позволит нам сосредоточить свои усилия на коде Java серверной стороны. Как вы убедитесь, самая сложная часть задачи — определение того, какой элемент является фруктом, а какой овощем.

Проблема

Нам необходимо предоставить пользователю списки данных, которые можно изменять перетаскиванием.

Решение

Как и в большинстве Ajax-приложений, в этом примере код будет разбит на три основные части: HTML, JavaScript и код приложения. Поскольку мы используем библиотеки DWR и Script.aculo.us, наши фрагменты HTML- и JavaScript-кода будут очень простыми — по сути, они представляют собой код регистрации и обработчики событий, а код приложения будет написан на Java. Итак, приступим.

Листинг 9.3. HTML-код списка

```
<html>
<head>
<style type="text/css">
li.green {
  background-color: #ECF3E1;
  border:1px solid #C5DEA1;
  cursor: move;
}

li.orange {
  border:1px solid #E8A400;
  background-color: #FFF4D8;
}

span.handle {
  background-color: #E8A400;
  color:white;
  cursor: move;
}
</style>
</head>
<body>

<h3>Sortable Lists</h3>

<div style="height:200px;" >
<div style="float:left;" >
<h3>Vegetables</h3>
  <ul id="vegetables"
    style="height:150px;width:200px;">
    <li class="orange" id="left_Pumpkin">
      <span class="handle">HANDLE</span> Pumpkin.
    </li>
    <li class="green" id="left_Carrot">Carrot.</li>
    <li class="orange" id="left_Orange">
      <span class="handle">HANDLE</span> Orange.
    </li>
  </ul>
</div>

<div style="float:left;" >
  <h3>Fruits</h3>
  <ul id="fruits" style="height:150px;width:200px;" >
    <li class="orange" id="right_Apple">
      <span class="handle">HANDLE</span> Apple.
    </li>
```

Специфицирует стиль овощей

Специфицирует стиль фруктов

Специфицирует стиль дескриптора фруктов

Содержит левый список систематизированных элементов

Содержит правый список систематизированных элементов

```
    <li class="orange" id="right_Tomato">
      <span class="handle">HANDLE</span> Tomato.</li>
    <li class="green" id="right_Turnip">Turnip.</li>
  </ul>
</div>

<br>

<div style="clear: both; margin-left: 150px;" >
  <span id="plant-error"
    style="color: red;" ></span>
</div>
</div>

<script type="text/javascript">
  Sortable.create("vegetables",
    {dropOnEmpty:true,
    containment:["vegetables","fruits"],
    constraint:false,
    onUpdate:handleUpdate});
  Sortable.create("fruits",
    {dropOnEmpty:true,handle:'handle',
    containment:["vegetables","fruits"],
    constraint:false,
    onUpdate:handleUpdate});
</script>

</body>
```

Указывает область сообщения приложения

Регистрирует левый список

Регистрирует правый список

В результате мы получаем списки элементов, место для отображения сообщений и регистрационные функции Script.aculo.us, обеспечивающие возможность перетаскивания и оставления элементов.

Теперь нужно применить ряд функций DWR, чтобы связать все эти возможности со своим приложением на сервере. Для этого достаточно включить необходимые библиотеки DWR и реализовать функции обратного вызова Script.aculo.us с помощью функций DWR (см. листинг 9.4).

Листинг 9.4. Код JavaScript списка

```
<head>
<script type="text/javascript" src="lib/prototype.js">
  </script>
<script type="text/javascript" src="lib/scriptaculous.js">
  </script>
<script type="text/javascript" src="dwr/engine.js"></script>
<script type="text/javascript" src="dwr/util.js"></script>
<script type="text/javascript" src="dwr/interface/Demo.js"></script>

<script type="text/javascript">
function handleUpdate(list) {
  Demo.checkPlant(
    Sortable.serialize("vegetables")
    + "&" + Sortable.serialize("fruits"),
    showPlantMessage);
}
```

Загружает библиотеки Script.aculo.us

Загружает базовые библиотеки DWR

Загружает библиотеки для приложений DWR

❶ Кодирует измененные списки для приложения

```
function showPlantMessage(message) {
  DWRUtil.setValue("plant-error", message);
}
</script>
</head>
```

❷ Отображает сообщение приложения

У нас имеются два списка, и любой из них может изменяться, поэтому при любом изменении мы отправляем серверу текущие состояния обоих списков ❶. На обновление списка сервер будет отвечать ❷ сообщением, которое мы будем вставлять в страницу с помощью функции `DWRUtil.setValue()`. Как видно из функций сценария DWR, необходимо реализовать только один метод серверной стороны: в зависимости от содержимого обоих списков мы будем возвращать строку, указывающую на наличие ошибки (см. листинг 9.5).

Листинг 9.5. Java-код списка

```
package aip;
import java.util.HashMap;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
public class Demo {
  Map plants;
  public Demo() {
    initPlantData();
  }
  private void initPlantData() {
    plants = new HashMap();
    List fruits = new ArrayList();
    List vegetables = new ArrayList();
    fruits.add("Apple");
    fruits.add("Pumpkin");
    fruits.add("Orange");
    fruits.add("Tomato");
    vegetables.add("Carrot");
    vegetables.add("Turnip");
    plants.put("vegetables", vegetables);
    plants.put("fruits", fruits);
  }
  public String checkPlant(String listUpdate) {
    String[] items = listUpdate.split("&");
    for (int i = 0; i < items.length; i++) {
      if ("".equals(items[i])) {
        continue;
      }
      String[] pair = items[i].split("\\[\\]=");
      List plant = (ArrayList) plants.get(pair[0]);
      if (!plant.contains(pair[1])) {
        return pair[1] + " isn't a " +
          pair[0].substring(0, pair[0].length() - 1) + ".";
      }
    }
    return "";
  }
}
```

Инициализирует экспертную систему овощей/фруктов

Разбивает закодированный список на элементы

Разбивает элементы на пары список/запись

Визуализирует вывод относительно принадлежности к фруктам/овощам

Рассмотрим метод `checkPlant()` подробнее, поскольку он имеет очень мало дела с растительными продуктами, но выполняет значительный объем работы по декодированию сериализованных списков Script.aculo.us. В функции обратного вызова `handleUpdate()` библиотеки Script.aculo.us мы дописываем последовательную форму второго списка к первому и отправляем его серверу посредством DWR. Параметр `listUpdate` может выглядеть подобно следующему:

```
vegetables[]=Pumpkin&vegetables[]=Carrot&vegetables[]=Orange
      &fruits[]=Apple&fruits[]=Tomato&fruits[]=Turnip
```

Отдельные элементы разделяются символом `&` (подобно значениям HTML-формы) и кодируются следующим образом:

имя_списка`[]=`*id*

Поэтому, чтобы выполнить декодирование, мы разбиваем всю строку на элементы у каждого символа `&`, а затем каждый элемент, помеченный символами `[]=`, разбиваем на его имя списка и идентификатор. Остальная часть метода `checkPlant()` всего лишь просматривает элементы в базе данных (которая в данном случае не слишком велика) для обнаружения первой ошибки. Найдя ошибку, метод генерирует информационное сообщение, исходя из имеющихся данных.

Обсуждение

Каркас Script.aculo.us облегчает разработку приложений, которые позволяют пользователям работать со списками. Однако, в отличие от программных интерфейсов приложений настольных ПК, изменения в списках не передаются нам в виде объектов. Вместо этого мы получаем текущее состояние списка в виде последовательности идентификаторов `id`. Поэтому элементам списков важно присваивать понятные идентификаторы, а не просто исходить из их позиции в списке. Точнее говоря, используемые `id` должны уникально идентифицировать элемент списка в данном приложении. Существует множество возможностей, но, вероятно, целесообразно использовать ключ базы данных или хеш-код объекта. Идентификатором может быть любое имя, которое может быть уникально идентифицировано на сервере (но оно должно быть достаточно кратким, поскольку пересылаться будет весь список идентификаторов `id`).

Хотя Script.aculo.us — прекрасное средство, оно не единственное в нашем арсенале. Рассмотрим еще один популярный каркас Ajax, ICEfaces, который можно использовать для реализации возможностей перетаскивания в Ajax-приложениях.

9.2.3. Тележка для покупок Ajax, использующая ICEfaces

ICEfaces — комплект инструментальных средств с открытым исходным кодом, предназначенный для разработки Ajax-приложений в виде стандартных приложений JavaServer Faces (JSF) (`http://www.icefaces.org`). Отличительные характеристики ICEfaces — методология разработки и естественная возможность обновления, инициируемая приложением посредством Ajax Push. (Метод Ajax Push тесно связан с “Comet” или “Reverse Ajax” и может использоваться для реализации внутри Web-приложений уведомлений или функций сотрудничества множества пользователей.) Разработка приложения ICEfaces сводится к разработке стандартного приложения JSF. Приложение может становиться Ajax-приложением без каких-либо изменений кода, тем самым, сохраняя четкое разделение “модель-представление-контроллер”, на которое делается упор в JSF. Для получения всех преимуществ, предоставляемых инициируемыми приложением обновлениями, требуется единственный метод программного интерфейса

приложений ICEfaces. Приложение просто вызывает на сервере метод `render()` при необходимости обновления страницы, а каркас ICEfaces определяет и выталкивает в браузер минимум необходимых обновлений страницы.

Чтобы получить представление о разработке приложения в каркасе ICEfaces, применим его для реализации примера тележки для покупок с поддержкой перетаскивания, которую мы ранее создали с помощью JavaScript и Script.aculo.us. Это приложение просто: пользователь может перетащить любой из трех элементов, а приложение обновит сообщение, отображаемое сбоку от тележки, в зависимости от перетаскиваемого элемента и от того, оставлен ли он в тележке.

Проблема

Нужно реализовать тележку для покупок с поддержкой перетаскивания, используя каркас ICEfaces.

Решение

Но вначале взгляните на тележку для покупок Ajax, показанную на рис. 9.4. Если ее внешний вид и поведение покажутся вам знакомыми, не удивляйтесь. ICEfaces использует библиотеки Script.aculo.us для предоставления возможностей перетаскивания — интерфейс JavaScript к каркасу Script.aculo.us всего скрыт для разработчика Java за компонентной моделью JSF, предоставляемой каркасом ICEfaces.

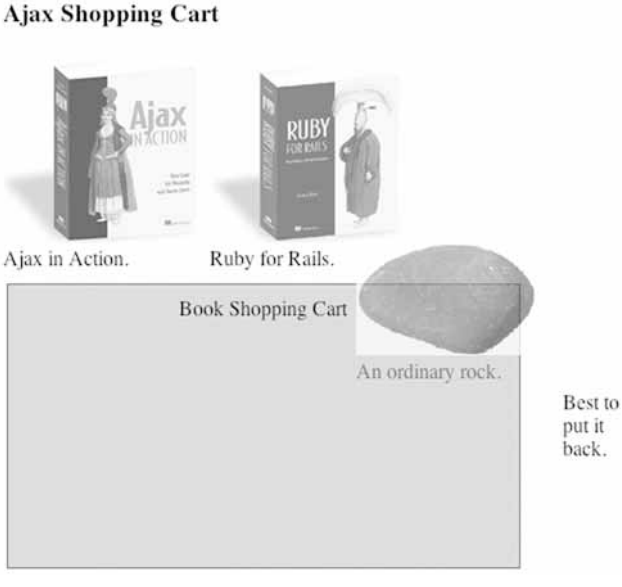

Рис. 9.4. Тележка для покупок с поддержкой перетаскивания, на этот раз реализованная с помощью ICEfaces

Наша реализация будет состоять из трех основных разделов: объявления интерфейса пользователя (который в листинге 9.6 представлен в виде документа JSP, но очень напоминает код разметки HTML), двух простых JavaBean-компонентов и XML-файла конфигурации, содержащего вставленные данные приложения. Начнем с подробного рассмотрения кода разметки, который содержит небольшой объем кода конфигу-

рирования JSF и стиля каскадной таблицы стилей, но основное внимание мы уделим конфигурации перетаскиваемых компонентов. Еще один заслуживающий внимания компонент – отображение сообщения тележки для покупок. Он настолько прост, что его легко упустить, поскольку он представляет собой всего лишь компонент текстового вывода, связанный с JavaBean. ICEfaces берет на себя заботу о создании кода Ajax. Когда элемент оставляется в тележке и сообщение изменяется, каркас определяет минимально необходимое обновление страницы и автоматически применяет его. Нам нужно лишь указать способ привязки частей окна к данным в используемой модели.

Листинг 9.6. ICEfaces JSF

```
<f:view xmlns:f="http://java.sun.com/jsf/core"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:ice="http://www.icesoft.com/icefaces/component">
```

Объявляет пространство имен дескриптора JSF

```
<html>
<head>
   <style type="text/css">
      div.cart {
         z-index:100;
         text-align:center;
         height:200px;
         padding:10px;
         background-color:#abf;
      }
   </style>
</head>
<body>
```

Содержит код разметки HTML и стиль каскадной таблицы стилей

```
<h3>Ajax Shopping Cart</h3>

<ice:form>
```

<- Специфицирует форму, содержащую компоненты ввода

```
 <table><tr>

  <td>
    <ice:panelGroup
       style="z-index:500;cursor:move;"
       draggable="true"
       dragListener=
          "#{dndBean.dragListener}"
       dragMask=
          "dragging,drag_cancel,hover_end"
       dragOptions=
          "#{craneItem.dragOptions}"
       dragValue="#{craneItem}" >
    <img alt="Product1" src="crane_3d.jpg" />
    <br/>
      Ajax in Action.
    </ice:panelGroup>
  </td>
  <td>
    <ice:panelGroup
```

Определяет контейнер для перетаскиваемой книги

<- Указывает перетаскиваемый элемент

❶

❷

❸

<- ❹

Специфицирует HTML-код для перетаскиваемой книги

<- Закрывает контейнер перетаскиваемого элемента

```
      style="z-index:500; cursor:move;"
      draggable="true"
      dragListener=
         "#{dndBean.dragListener}"
      dragMask=
         "dragging,drag_cancel,hover_end"
      dragOptions=
         "#{blackItem.dragOptions}"
      dragValue="#{blackItem}">
   <img alt="Product2" src="black_3d.jpg" />
   <br/>
    Ruby for Rails.
   </ice:panelGroup>
  </td>

  <td>
   <ice:panelGroup
      style="z-index:500; cursor:move;"
      draggable="true"
         dragListener=
         "#{dndBean.dragListener}"
      dragMask=
         "dragging,drag_cancel,hover_end"
      dragOptions=
         "#{rockItem.dragOptions}"
      dragValue="#{rockItem}">
   <img alt="Product3" src="rock.jpg" />
   <br/>
    An ordinary rock.
   </ice:panelGroup>
  </td>
</tr></table>
<table><tr>

<td width="400px">
   <ice:panelGroup style="z-index:0;"
      dropTarget="true">
     <div id="cart" class="cart" >
        Book Shopping Cart
     </div>
   </ice:panelGroup>
  </td>

<td width="25">
</td>

<td width="50">
   <ice:outputText
     value="#{dndBean.dragMessage}"/>
</td>
</tr></table>
</ice:form>
</body>
</html>
</f:view>
```

Специфицирует вторую перетаскиваемую книгу

Специфицирует перетаскиваемый камень

Специфицирует HTML-код для области оставления тележки

Определяет цель оставления

Содержит текст сообщения, полученный из JavaBean

`dndBean` ❶ — имя объекта JavaBean, моделирующего наше приложение тележки для покупок. Он содержит метод слушателя `dragListener()`, вызов которого это выражение связывания требует от JSF, когда что-либо интересное происходит с данным перетаскиваемым элементом.

Нас интересуют не все события, связанные с перетаскиванием. Каркас ICEfaces обрабатывает события на сервере ❷, и мы не стремимся к пересылке серверу буквально каждого перемещения мыши, возникающего при перетаскивании. Поэтому мы используем эту маску для блокирования не интересующих нас событий.

Нам требуется, чтобы различные элементы по-разному вели себя при перетаскивании (например, в зависимости от того, должен ли элемент возвращаться на место после его оставления в тележке). Связывание этого поведения с контейнером элемента с помощью выражения ❸ позволяет осуществлять индивидуальное конфигурирование для каждого элемента.

`dragItem` — действительный объект JavaBean, связанный с перетаскиваемым элементом, которым может оперировать метод `dragListener()` при перетаскивании. Это позволяет Java-приложению работать с объектами, которые имеют смысл на уровне приложения ❹.

Удалось ли вам выявить код Ajax в объявлении страницы? Он умышленно скрыт — одна из целей каркаса ICEfaces заключается в прозрачном предоставлении кода Ajax. Разработчик сосредоточивает свое внимание на приложении: на том, какие компоненты помещаются на страницу, и как эти компоненты связаны с динамической моделью данных. Все низкоуровневые аспекты способа обновления этих компонентов по сети с помощью XHR вынесены за рамки приложения и обрабатываются каркасом.

Мы создали страницу, которая объявляет компоненты, отображаемые для пользователя. Теперь нам требуется исполняемая часть приложения. Она будет полностью реализована на сервере в объектах JavaBean. Начнем с простейшей части модели: покупаемых элементов. Объект `ShoppingItem` (листинг 9.7) полностью ориентирован на данные. JavaBean обеспечивает ему только возможность содержания информации об элементе. В более сложном приложении пришлось бы включать такие сведения, как цена, вес и доступность, но для этого простого примера наш элемент будет содержать только сообщения, указывающие о его перетаскивании и оставлении пользователем, и параметры, используемые для определения поведения при перетаскивании (только камень может быть возвращен назад вместо его оставления в тележке).

Listing 9.7. `ShoppingItem`

```
package aip;
import java.io.Serializable;

public class ShoppingItem implements Serializable {

   String hoverMessage = "";
   public void setHoverMessage(String message) {
      this.hoverMessage = message;
   }

   public String getHoverMessage() {
      return this.hoverMessage;
   }

   String dropMessage = "";
   public void setDropMessage(String message) {
      this.dropMessage = message;
   }
```

Устанавливает/получает сообщение о "плавающем перемещении"

Устанавливает/получает сообщение об оставлении

```
  public String getDropMessage(){
    return this.dropMessage;
  }

  String dragOptions = "";
  public void setDragOptions(String options) {
    this.dragOptions = options;
  }

  public String getDragOptions() {
    return this.dragOptions;
  }
}
```

Устанавливает/получает параметры перетаскивания

Легко заметить, что `ShoppingItem` может представлять свойства покупаемого элемента в нашем простом приложении, но как создаются и конфигурируются отдельные элементы? Интересная технология — использование внедрения зависимостей или инверсия возможностей элемента управления с помощью управляемых контейнеров. В этом примере (листинг 9.8) мы применяем управляемые контейнеры просто для создания экземпляров и указания элементов в приложении, внедряя контейнеры в сеанс пользователя извне (это простое приложение не имеет никаких осмысленных зависимостей).

Листинг 9.8. Конфигурирование ICEfaces JSF

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE faces-config PUBLIC
  "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
  "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">

<faces-config>
```

Объявляет файл конфигурации JSF

```
  <managed-bean>
    <managed-bean-name>dndBean</managed-bean-name>
    <managed-bean-class>aip.DragDropBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
```

Объявляет контейнеры перетаскивания в сеансе

```
  <managed-bean>
    <managed-bean-name>craneItem</managed-bean-name>
    <managed-bean-class>aip.ShoppingItem</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
        <managed-property>
          <property-name>hoverMessage</property-name>
          <value>Good choice.</value>
        </managed-property>
        <managed-property>
          <property-name>dropMessage</property-name>
          <value> Thank you for your purchase.</value>
        </managed-property>
        <managed-property>
          <property-name>dragOptions</property-name>
          <value></value>
        </managed-property>
  </managed-bean>
```

Присваивает имя первому элементу книги

Указывает область определения сеанса

❶ **Объявляет сообщение “плавающего перемещения” для элемента**

Объявляет сообщение опускания для элемента

Объявляет параметры перетаскивания элемента (по умолчанию)

```
  <managed-bean>
    <managed-bean-name>blackItem</managed-bean-name>
    <managed-bean-class>aip.ShoppingItem</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
        <managed-property>
          <property-name>hoverMessage</property-name>
          <value>Excellent selection.</value>
        </managed-property>
        <managed-property>
          <property-name>dropMessage</property-name>
          <value>You are sure to enjoy.</value>
        </managed-property>
        <managed-property>
          <property-name>dragOptions</property-name>
          <value></value>
        </managed-property>
  </managed-bean>
```

Объявляет второй элемент книги

```
  <managed-bean>
    <managed-bean-name>rockItem</managed-bean-name>
    <managed-bean-class>aip.ShoppingItem</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
        <managed-property>
          <property-name>hoverMessage</property-name>
          <value>Put it back.</value>
        </managed-property>
        <managed-property>
          <property-name>dropMessage</property-name>
          <value>You really don't want this one.</value>
        </managed-property>
        <managed-property>
          <property-name>dragOptions</property-name>
          <value>revert</value>
        </managed-property>
  </managed-bean>
</faces-config>
```

Объявляет третий элемент (камень)

Объявляет параметры перетаскивания элемента (возврат на место)

Свойство `hoverMessage` соответствует непосредственно полю `hoverMessage` в контейнере `ShoppingItem`. JSF ищет метод с именем `setHoverMessage()` ❶ и вызывает его для экземпляра контейнера с помещенным в него значением.

В результате мы получаем интерфейс пользователя и ряд данных, но пока наше приложение не способно выполнять какие-либо действия. Мы еще не реализовали никакие функции, выполняемые в ответ на инициированные пользователем события. Что же делает наше приложение? Когда пользователь перетаскивает элемент, сообщение обновляется в зависимости от выполняемого действия. Как видно в объявлении страницы, посредством функции обратного вызова `dragListener()` наш объект JavaBean выясняет, что элемент был перетащен. Благодаря компоненту текстового вывода, связанному с контейнером, сообщение обновляется (ICEfaces обеспечивает, чтобы при необходимости сообщение обновлялось на странице). Посмотрим, как эти функции реализованы в контейнере (листинг 9.9).

Листинг 9.9. `DragDropBean`

```
package aip;

import com.icesoft.faces.component
      .dragdrop.DragEvent;
import com.icesoft.faces.component.ext.HtmlPanelGroup;

public class DragDropBean {

   private String dragMessage = "";

   public void setDragMessage(String message) {
      this.dragMessage = message;
   }

   public String getDragMessage() {
      return this.dragMessage;
}

   public void dragListener(
         DragEvent dragEvent) {
      ShoppingItem item = (ShoppingItem)
        ((HtmlPanelGroup)
              dragEvent.getComponent())
                    .getDragValue();

      if (null != item) {
         if ( dragEvent.getEventType()
              == dragEvent.HOVER_START ) {
            this.dragMessage =
              item.getHoverMessage();
         } else if ( dragEvent.getEventType()
              == dragEvent.DROPPED ) {
            this.dragMessage =
              item.getDropMessage();
         }
      }

   }
}
```

Использует события перетаскивания каркаса ICEfaces

❶ Устанавливает/получает сообщения перетаскивания

Обрабатывает события перетаскивания

Извлекает объект покупаемого элемента

❷ Отображает сообщение "плавающего перемещения" при таком перемещении

❸ Отображает сообщение оставления, если элемент оставлен

Метод `setDragMessage()` ❶, скорее всего, никогда не будет вызываться, но с точки зрения стилистики лучше включить методы установки свойств контейнера. При наступлении события "плавающего перемещения" мы извлекаем сообщение "плавающего перемещения" из элемента. Элемент — это объект, которому известно, что должно отображаться при его "плавающем перемещении" над тележкой для покупок ❷. Если элемент оставлен в тележке ❸, мы извлекаем из него сообщение оставления.

Обсуждение

С философской точки зрения наибольшего внимания заслуживает свободная связь между моделью и представлением. Событие перетаскивания не вызывает непосредственного отображения сообщения, за исключением случаев, когда тележка должна быть изменена соответствующим образом. Событие принимается приложением, в результате чего его внутреннее состояние изменяется. Дизайнер может отражать это внутреннее состояние в представлении страницы любым удобным для него способом,

например, путем отображения сообщения рядом с тележкой или во всплывающем окне. Но при этом ему не нужно знать, что перетаскивание элемента в тележку изменяет внутреннее состояние. Изменение состояния, которое относится к области бизнес-логики, благополучно скрыто в коде приложения.

9.3. Резюме

Мы рассмотрели три способа использования перетаскивания с помощью Ajax: во-первых, выяснили, как применять эту возможность к произвольным объектам, во-вторых, ознакомились с изменениями, требуемыми для списков, и, в-третьих, рассмотрели каркас ICEfaces. Единственными изменениями, которые потребовались для списков, заключались в автоматическом смещении элементов в сторону и использовании подходящей функции сериализации (которая учитывает позицию элементов списка, а не только то, в какую область они помещаются). В любом случае основной вывод состоит в том, что доступны библиотеки JavaScript (такие как Script.aculo.us или ICEfaces), которые представляют перетаскивание событиями высокого уровня, легко связываемыми с функциями Ajax. Теперь достаточно применить эти методики, чтобы удивить пользователей настольных приложений приложениями браузера, которые предоставляют даже больше возможностей, чем можно было ожидать.