# 第 11 章 集成 Ajax

在 Web2.0 的应用中，经常碰到诸如在客户端交互作用、复杂视觉效果及异步通讯之类的问题，这些功能都要用 javascript 去实现。但是，用 javascript 编程是很繁琐且难以测试的。为了自动完成许多要用 javascript 编写的常用功能，symfony 在模板中提供了一组完整的辅助函数。许多客户端功能甚至连一行 javascript 语句都不用写就可以开发出来，开发人员只需要考虑他们想要达到的效果，而复杂的语法和兼容性问题则交给 symfony 去处理。

本章介绍以下内容，这些内容将会帮助你了解如何用 symfony 提供的工具来编写客户端代码：

- 在 symfony 模板中,基本的 javascript 辅助函数输出与标准兼容的<script>标记，用于更新 DOM（文档对象模型）元素或用链接触发一段代码。
- symfony 中集成了一个 javascript 库 Prototype，它为 javascript 核心增加了新的函数和方法，以便加速客户端代码的开发。
- ajax 辅助函数让用户可以通过点击一个链接，提交一个表单或修改一个表单元素来更新一个页面的某些部分。
- 辅助函数的选项可以提供更大的灵活性和能力，回调函数的运用就是最典型的例子。
- symfony 中还集成了另一个 javascript 库 Script.aculo.us，利用它可以增强动态视觉效果，最终提高用户体验。
- JSON（Javascript 对象标识）是一种用于在客户端和服务器端相互通信的标准。
- 结合了前述各种技术的复杂的客户端交互，都可以在 symfony 的应用中加以实现。只要写一行 PHP 语句去调用一个 symfony 辅助函数，就可以实现诸如自动完成、拖拽、可排序列表及可编辑文本等功能。

## 基本的 javascript 辅助函数

过去，由于 javascript 存在浏览器兼容性问题，所以在专业的 web 应用中很少用到它。但是现在，兼容性问题已经基本解决，利用许多质量稳定的 javascript 库，无需写大量代码和进行耗时的测试，就可以开发出复杂的 javascript 交互应用。ajax 就是进展最快和最普及的 javascript 应用，本章后面的“ajax 辅助函数”将会介绍它。

你可能会奇怪为什么在这一章里只有极少的 javascript 代码。这正是 symfony 的独特之处，因为 symfony 已经将 javascript 行为封装和抽象进辅助函数中了，所以在你的模板中可以完全不用编写 javascript 代码。开发人员只需要用一行 PHP 语句就可以为页面中的一个元素增加一种行为，但是被调用的辅助函数会

输出 javascript 代码，只要分析一下生成的响应就可以揭示出封装的复杂性，因为辅助函数处理了包括浏览器一致性、复杂的限制条件和扩展性等问题，所以 javascript 代码的总量非常重要。本章将告诉你怎样不使用 javascript 语句达到用 javascript 才能产生的效果。

这里介绍的所有辅助函数都用于模板，只要你事先声明使用 Javascript 辅助函数组即可。

```
<?php use_helper('Javascript') ?>
```

你很快会看到，一些辅助函数只输出 HTML 代码，而另一些会输出 javascript 代码。

## 模板中的 javascript

在 XHTML 中，javascript 代码块必须包含在 CDATA 声明中。但是如果在页面中需要写多个 javascript 代码块时，这样做就变得非常繁琐。所以 symfony 提供了 javascript_tag()辅助函数，用于将一个字符串转化为一个符合 XHTML 规范的<script>标记。例 11-1 是这个辅助函数的例子：

例 11-1 用 javascript_tag()辅助函数插入 javascript 代码

```
<?php echo javascript_tag("
  function foobar()
  {
  ...
  }
") ?>
=> <script type="text/javascript">
    //<![CDATA[
      function foobar()
      {
        ...
      }
    //]]>
    </script>
```

但是 javascript 的最主要应用并不是编写代码块，而是用超链接去触发某个具体脚本。例 11-2 显示了 link_to_function()辅助函数的使用。

例 11-2 利用 link_to_function()辅助函数，一个链接可以触发 javascript 代码。

```
<?php echo link_to_function('Click me!', "alert('foobar')") ?>
=>  <a href="#" onClick="alert('foobar'); return none;">Click me!</a>
```

和 link_to()辅助函数一样，第三个参数可以为标记加入选项。

NOTE 与 link_to()相似的还有 button_to()，你可以调用 button_to_function()辅助函数通过一个按钮(<input type="button">) 触发 javascript 代码。如果你还想要一个可点击的图片，只要调用 link_to_function(image_tag(“myimage”), “alert(‘foobar`)”)即可。

## 更新一个 DOM 元素

更新页面中的一个元素是动态界面经常要解决的问题。例 11-3 是为此而经常编写的代码。

例 11-3 用 javascript 更新一个元素

```
<div id="indicator">Data processing beginning</div>
<?php echo javascript_tag('
  document.getElementById("indicator").innerHTML =
    "<strong>Data processing complete</strong>";
') ?>
```

symfony 为此提供了一个名为 update_element_function()的辅助函数，用于生成 javascript 代码而不是 html 代码。例 11-4 是一个示例。

例 11-4 在 javascript 代码块中用 update_element_function()辅助函数更新一个元素

```
<div id="indicator">Data processing beginning</div>
<?php echo javascript_tag('
  update_element_function("indicator", array(
    "content"  => "<strong>Data processing complete</strong>",
  '))
) ?>
```

你也许在想：这条辅助函数语句和真正的 javascript 代码一样长，那使用它有什么特别的好处呢？好处在于它的可读性。例如，如果你想根据某种条件在一个元素之前或之后插入内容，或者不是更新元素内容而是删除一个元素，或者不做任何处理时，javascript 代码将会变得相当混乱，但是利用 update_element_function()，却可以像例 11-5 那样保持清晰易读。

例 11-5 update_element_function()辅助函数的选项

```
// 在 indicator 元素之后插入内容
update_element_function('indicator', array(
  'position' => 'after',
  'content'  => "<strong>Data processing complete</strong>",
));

// 如果$condition 成立，则删除 indicator 之前的元素
update_element_function('indicator', array(
  'action'   => $condition ? 'remove' : 'empty',
  'position' => 'before',
))
```

可以看出，这个辅助函数让你的模板比任何 javascript 代码都要容易理解，而且你只要使用一种语法就可以处理相似的行为。这也是为什么这个辅助函数名字这样长的原因：无需额外的说明，它就可以充分地解释自身的用途。

### 轻松地降级（Graceful Degradation）

用<noscript> 指明的 html 代码，仅在不支持 javascript 的浏览器中显示。symfony 用一个辅助函数从另一方面进行补充这个功能，也就是利用它可以让代码仅在支持 javascript 的浏览器中执行。例 11-6 中，显示了用 if_javascript()和 end_if_javascript()实现这种降级的用法：

例 11-6 利用 if_javascript()辅助函数轻松地降级

```
<?php if_javascript(); ?>
  <p>You have JavaScript enabled.</p>
<?php end_if_javascript(); ?>

<noscript>
  <p>You don't have JavaScript enabled.</p>
</noscript>
```

NOTE 调用 if_javascript()和 end_if_javascript()时，不需要用 echo.

## Prototype

Prototype 是一个优秀的 javascript 库，它扩展了客户端脚本的能力，增加了开发者想要的功能，并且提供了新的机制去操作 DOM，该项目的网站是 http://prototypejs.org/。

symfony 框架中绑定了 Prototype 文件，在每个项目的 web/sf/prototype 目录中可以找到它的文件。这样只要在你的 action 中增加下列代码就可以使用 Prototype：

```
$prototypeDir = sfConfig::get('sf_prototype_web_dir');
$this->getResponse()->addJavascript($prototypeDir.'/js/prototype');
```

或者在 view.yml 文件中加入：

```
all:
  javascripts: [%SF_PROTOTYPE_WEB_DIR%/js/prototype]
```

**NOTE** 因为 symfony Ajax 辅助函数(下一节介绍)需要用到 Prototype，所以只要你用到 Prototype 库，它就会自动被包含进来。也就是说，如果你的模板调用一个_remote 辅助函数，你无需在你的响应里手工添加 Prototype Javascript。

一旦你载入了 Prototype 库，就可以利用它为 javascript 核心增加的新函数。本书的主要目的不是讲述这些函数，你可以很容易在互联网上找到所需的文档，以下是几个有关的网站：

- Particletree: http://particletree.com/features/quick-guide-to-prototype/
- Sergio Pereira: http://www.sergiopereira.com/articles/prototype.js.html
- Script.aculo.us: http://wiki.script.aculo.us/scriptaculous/show/Prototype

Prototype 新增的 Javascript 函数之一是$()函数。简单地说，这个函数可以看成是 document.getElementById()函数的缩写，但它有更强的功能。例 11-7 给出了一个应用的例子。

例 11-7 利用$()函数根据 DOM 的元素 ID 取得元素值。

```
node = $('elementID');

// 相当于
node = document.getElementById('elementID');

// 也可以一次检索多个元素
// 在本例中返回值是一个由 DOM 元素组成的数组。
nodes = $('firstDiv', 'secondDiv');
```

Prototype 还提供了 javascript 核心真正缺乏的一个方法，这个方法返回所有 CSS className 属性等于它接收的参数的 DOM 元素组成的数组：

```
nodes = document.getElementByClassName('myclass');
```

当然你不太会用到这个函数，因为 Prototype 提供了一个更强大的$$()函数。这个函数根据 CSS 选择器返回一个由 DOM 元素组成的数组。所以前面的调用可以写成如下形式：

```
nodes = $$('.myclass');
```

凭借 CSS 选择器的作用，你可以根据 class、ID、父子关系和前后关系去解析 DOM，这比通过 XPath 表达式去解析更为简单。你甚至可以用一个混合了所有这些选择器的复杂选择器去访问对应的元素。

```
nodes = $$('body div#main ul li.last img > span.legend');
```

Prototype 增强 Javascript 语法功能的最后一个例子是数组迭代。它为 Javascript 定义匿名函数和闭包（closure）（译者注：如果在一个 javascript 函数体中又出现一个函数定义时，称此函数为闭包（closure））功能提供了和 PHP 同样的简化形式。如果你编写 javascript 代码，可能会大量用到这个功能。

```
var vegetables = ['Carrots', 'Lettuce', 'Garlic'];
vegetables.each(function(food) { alert('I love ' + food); });
```

因为用 Prototype 编写 Javascript 比纯手工编写更为有趣，并且因为 Prototype 也是 symfony 的一个组成部分，所以你应该花些时间去研究它的文档。

# Ajax 辅助函数

如果你想在服务器端用 PHP 脚本去更新页面中的元素内容，而不想用 javascript 去更新（如例 11-5 所示），这样你可以根据一个服务器的响应来改变页面的某个部分，那该怎么做呢？remote_function()辅助函数就可以完成这个任务，如例 11-8 所示：

例 11-8 应用 remote_function() 辅助函数

```
<div id="myzone"></div>
<?php echo javascript_tag(
  remote_function(array(
```

```
    'update'  => 'myzone',
    'url'     => 'mymodule/myaction',
  ))
) ?>
```

**NOTE** url 参数既可以包含一个内部 URI(module/action?key1=value1&…)，也可以包含一个路由规则名，就像在一个标准的 url_for()中那样。

当你调用这个函数时，这段脚本就会根据 mymodule/myaction 动作的请求或响应，去更新 id 为 myzone 的元素。这种交互就是 Ajax，也正是高度可交互的 web 应用的核心。Wikipedia (http://en.wikipedia.org/wiki/AJAX) 描述了 Ajax 的特点：

Ajax 让页面只和服务器交换很少量的数据，因而每当用户改变页面时，不需要重新导入整个网页，而使得页面的响应更快。也就是说增强了网页的交互性，速度和可用性。

Ajax 依赖于 javascript 对象 XMLHttpRequest，该对象的行为如同一个隐藏的帧，你可以从一个服务器请求更新它并且重用它去操纵页面的剩余部分。这是一个相当底层的对象，不同的浏览器用不同的方法去处理它。所幸的是，Prototype 封装了所有 Ajax 需要的代码并提供了一个更为简化的 Ajax 对象，symfony 就借助于这个对象。这也是为什么当你在一个模板中使用 Ajax 辅助函数时，Prototype 就会自动装入的原因。

**CAUTION** 如果远程动作的 URL 不属于当前页所在的域，Ajax 辅助函数将不工作。这既是出于安全考虑，也受到浏览器禁止远程动作通过的限制。

一个 Ajax 交互由三个部分组成：一个调用者（链接、按钮、表单、时钟或其它用户可以操纵以启动动作的任何控件），一个服务器动作和页面中的一个显示动作响应的区域。如果远程动作返回的数据还要被客户端的 Javascript 函数处理，你可以创建更复杂的交互。symfony 提供了多个名字中包含 remote 的辅助函数，以便你在模板中插入 Ajax 交互。它们使用共同的语法，可以将所有的 Ajax 参数放进一个关联数组中。注意，Ajax 辅助函数输出的是 HTML，而不是 Javascript。

**SIDEBAR** Ajax 动作如何工作？

远程函数被调用的动作就是一个通常的动作。与其它动作一样，它们可以被路由，可以确定用哪个视图提交它们返回的响应，也可以向模板传递参数以及改变模型等。

但是，当通过 Ajax 调用动作时，动作将返回 true 给下面的调用：

```
$isAjax = $this->isXmlHttpRequest();
```

symfony 知道动作处于 Ajax 环境中，因而能够对响应做相应的处理。因此，在默认情况下，开发环境中的 Ajax 动作不包含 web 调试工具栏，而且也会跳过装饰处理（默认情况下，模板不会被包含在布局中）。如果你想装饰 Ajax 的视图，你需要在模块的 view.yml 文件中，为这个视图明确设置 has_layout 的值为 true。

还有一点：因为响应性在 Ajax 的交互中至关重要，所以如果响应不是非常复杂，最好不要创建视图，而是直接从动作返回响应。这样你可以在动作中用 renderText()方法，直接跳过模板而加速 Ajax 请求。

## Ajax 链接

在 Web 2.0 应用中，Ajax 链接是 Ajax 交互应用的主要内容。显而易见，link_to_remote()辅助函数就可以输出一个调用远程函数的链接。除了第二个参数是一个由 Ajax 选项组成的关联数组以外，其他语法类似于 link_to()。例 11-9 是一个示例。

例 11-9 利用 ink_to_remote() 辅助函数得到 Ajax 链接

```
<div id="feedback"></div>
<?php echo link_to_remote('Delete this post', array(
    'update' => 'feedback',
    'url'    => 'post/delete?id='.$post->getId(),
)) ?>
```

在这个例子中，点击 Delete this post 链接就会在后台发出一个对 post/delete 的调用。从服务器返回的响应将出现在 id 为 feedback 的元素中。图 11-1 显示了运行的过程。

图 11-1 用链接触发一个远程更新


对于链接，你可以用图片代替字符串，用规则名代替内部的模块/动作 URL，还可以将选项加进标记的第三个参数中。如例 11-10 所示。

例 11-10 link_to_remote() 辅助函数的选项

```
<div id="emails"></div>
<?php echo link_to_remote(image_tag('refresh'), array(
```

```
    'update' => 'emails',
    'url'    => '@list_emails',
), array(
    'class'  => 'ajax_link',
)) ?>
```

## Ajax 驱动的表单

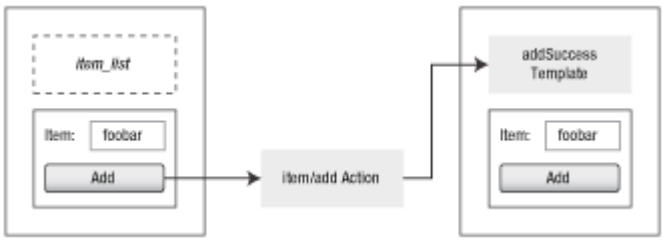
Web 表单一般要调用另一个动作，但这会导致整个页面被刷新。对表单来说，类似于 link_to_function()可以在表单提交后，仅用服务器的响应去更新页面中的一个元素，form_remote_tag()辅助函数就是完成这个任务的，例 11-11 展示了它的语法：

例 11-11 利用 form_remote_tag() 辅助函数得到 Ajax 表单

```
<div id="item_list"></div>
<?php echo form_remote_tag(array(
    'update'   => 'item_list',
    'url'      => 'item/add',
)) ?>
  <label for="item">Item:</label>
  <?php echo input_tag('item') ?>
  <?php echo submit_tag('Add') ?>
</form>
```

就像 form_tag()辅助函数一样，form_remote_tag()也打开一个<form>。提交这个表单会在后台向 item/add 动作发出一个 POST 请求，请求参数就是 item 字段的内容。响应会替换 item_list 元素的内容（如图 11-2 所示）。最后用通常的</form>标记关闭 Ajax 表单。

图 11-2 利用表单触发远程更新.




CAUTION 由于 XMLHttpRequest 对象的限制，Ajax 表单不可以分为多个部分。这意味着你不能通过 Ajax 表单上传文件。不过你可以用其他方法解决这个问题—比如，用隐式的 iframe 代替 XMLHttpRequest（参看 http://www.air4web.com/files/upload/给出的一个实现）。

如果你想让一个表单同时工作在页面模式和Ajax模式，最好的方法是定义一个通常的表单，然后除了提供通用的提交按钮，再增加一个按钮（<input type="button"/>）用于以Ajax方式提交表单。~~Symfony~~symfony用submit_to_remote()调用这个按钮。这可以帮助你建立一个可以轻松降级的Ajax交互表单，即可以与不支持Javascript的浏览器兼容。见例11-12所示。

例11-12 具有普通提交方式和Ajax提交方式的表单

```
<div id="item_list"></div>
<?php echo form_tag('@item_add_regular') ?>
  <label for="item">Item:</label>
  <?php echo input_tag('item') ?>
  <?php if_javascript(); ?>
    <?php echo submit_to_remote('ajax_submit', 'Add in Ajax', array(
        'update'   => 'item_list',
        'url'      => '@item_add',
    )) ?>
  <?php end_if_javascript(); ?>
  <noscript>
    <?php echo submit_tag('Add') ?>
  </noscript>
</form>
```

另一个混合了普通提交和Ajax提交标记的例子是编辑文章的表单。它可以提供一个实现了Ajax的预览按钮和一个用普通提交实现的发布按钮。

NOTE 当用户按下回车键时，表单会用定义在主<form>标记中的动作去提交，在本例中，就是普通提交动作。

现代表单不仅仅在提交时作出回应，在用户改变某个域的值时也会有回应。在symfony中，你可以用observe_field()辅助函数来实现这个功能。例11-13应用这个辅助函数建立一个具有建议特性的页面，也就是在item字段中每输入一个字符，都会触发一次Ajax调用去刷新页面中的item_suggestion元素。

例11-13 当字段值变化时，observe_field()调用一个远程函数

```
<?php echo form_tag('@item_add_regular') ?>
  <label for="item">Item:</label>
  <?php echo input_tag('item') ?>
  <div id="item_suggestion"></div>
  <?php echo observe_field('item', array(
      'update'   => 'item_suggestion',
      'url'      => '@item_being_typed',
  )) ?>
```

```
  <?php echo submit_tag('Add') ?>
</form>,
```

每当用户改变字段 item（称为“发现域”）的值时，即使没有提交表单，也会调用记录在@item_being_typed 规则中的模块/动作。这个动作将从 value 请求参数中获得当前的 item 值。如果你想传递非发现域的值，你可以在 with 选项中用 javascript 表达式来指明。例如，如果你想让动作得到 param 参数，可以将 observe_field()辅助函数写成例 11-14 中的形式：

例 11-14 用 with 选项将你自己的参数传递给远程动作

```
<?php echo observe_field('item', array(
    'update'   => 'item_suggestion',
    'url'      => '@item_being_typed',
    'with'     => "'param=' + value",
)) ?>
```

注意，这个辅助函数并不输出一个 HTML 元素，而是输出作为参数传递的元素的行为。在本章中你会看到更多用 javascript 辅助函数指定行为的例子。

如果你想发现一个表单中的所有域，你可以使用 observe_form() 辅助函数，每当表单中的任意一个域发生变化时，它都会调用一个远程动作。

## 周期性调用远程函数

symfony 还有一个 periodically_call_remote()辅助函数用于每隔几秒钟触发一次 Ajax 交互。它并不与某个 HTML 控件结合，而是作为整个页面的一个行为在后台透明地运行。在实现追踪鼠标位置、自动保存大块文本输入区内容等功能时，这个函数非常有用。例 11-15 是一个示例。

例 11-15 用 periodically_call_remote()周期性调用一个远程函数

```
<div id="notification"></div>
<?php echo periodically_call_remote(array(
    'frequency' => 60,
    'update'    => 'notification',
    'url'       => '@watch',
    'with'      => "'param=' + $('mycontent').value",
)) ?>
```

如果你不想指明两次调用远程函数之间的间隔秒数(frequency)，则默认值是 10 秒。注意，with 参数是 javascript 计算出来的，所以你可以在其中用 Prototype 函数，比如$()。

# 远程调用参数

除了参数update和url以外，前面介绍的所有Ajax 辅助函数还可以带另外一些参数。由Ajax参数组成的关联数组可以调整和改变远程调用的行为和对它们的响应的处理。

## 根据响应状态更新确定的元素

如果远程动作失败了，远程辅助函数可以选择更新另一个元素，而不是更新由成功的响应更新的元素，为了达到这个目的，只要将update参数的值分开存放进一个关联数组中，并且为success和failure两种情况设置要更新的不同的元素即可。如果一个页面有许多Ajax交互和一个错误反馈区，就可以利用这个功能。例11-16展示了这种有条件更新的用法。

例11-16 处理一个有条件更新

```
<div id="error"></div>
<div id="feedback"></div>
<p>Hello, World!</p>
<?php echo link_to_remote('Delete this post', array(
    'update'   => array('success' => 'feedback', 'failure' =>
'error')
    'url'      => 'post/delete?id='.$post->getId(),
)) ?>
```

TIP 只有HTTP错误代码(500,400 及所有不在2XX范围内的代码)才会触发失败更新，返回sfView::ERROR的动作并不会触发失败更新。所以如果你想写一个返回Ajax失败的动作，你必须调用类似$this->getResponse()->setStatusCode(404)的函数。

## 根据元素位置更新元素

通过加入position参数，你可以更新相对于某个具体元素位置的元素，就像update_element_function()辅助函数一样。例11-17是一个示例。

例11-17 用位置参数改变响应位置

```
<div id="feedback"></div>
<p>Hello, World!</p>
<?php echo link_to_remote('Delete this post', array(
    'update'   => 'feedback',
    'url'      => 'post/delete?id='.$post->getId(),
    'position' => 'after',
```

)) ?>

这个例子将在 feedback 元素之后插入 Ajax 调用的响应，也就是在 和<p>之间。有了这个方法，你可以调用多个 Ajax 调用，并在 update 参数指定的元素之后聚集所有的响应。

position 参数可以取以下值：

- before：在元素之前
- after：在元素之后
- top：在元素的内容顶部
- bottom：在元素的内容底部

## 根据条件更新元素

远程调用还可以设置 confirm 参数，以便在真正发出 XMLHttpRequest 之前用户可以确认。如例 11-18 所示：

例 11-18 在远程函数中加入确认参数以便在调用之前取得确认

```
<div id="feedback"></div>
<?php echo link_to_remote('Delete this post', array(
    'update'   => 'feedback',
    'url'      => 'post/delete?id='.$post->getId(),
    'confirm'  => 'Are you sure?',
)) ?>
```

这样，当用户点击了链接后，就会弹出一个显示了“Are you sure?”的 javascript 对话框，只有当用户点击 OK 表示确认后，post/delete 动作才会被调用。

如果你还设置了 condition 参数，远程调用将根据在浏览器端执行的条件测试来确定是否被执行，例 11-19 给出一个例子：

例 11-19 根据客户端的测试来确定是否调用远程函数

```
<div id="feedback"></div>
<?php echo link_to_remote('Delete this post', array(
    'update'    => 'feedback',
    'url'       => 'post/delete?id='.$post->getId(),
    'condition' => "$('elementID') == true",
)) ?>
```

## 确定 Ajax 请求方法

Ajax 请求默认采用 POST 方法。如果你调用一个并不改变数据的 Ajax 函数，或者，你想显示一个含有内置验证方法的表单作为 Ajax 调用的结果，你可能想将 Ajax 请求方法改为 GET。例 11-20 显示了用 method 选项改变 Ajax 请求方法的例子。

例 11-20 改变 Ajax 的请求方法

```
<div id="feedback"></div>
<?php echo link_to_remote('Delete this post', array(
    'update'    => 'feedback',
    'url'       => 'post/delete?id='.$post->getId(),
    'method'    => 'get',
)) ?>
```

## 授权脚本运行

如果一个 Ajax 函数的响应代码（即插入在 update 元素中的由服务器发出的代码）中含有 Javascript，在缺省情况下这些代码并不会执行。这样做的目的是为了减少远程攻击的风险，并且只有在开发者确认了响应中的代码后才执行脚本。

这样为了能执行远程响应中的脚本，你需要设置 script 参数来明确声明可以执行。例 11-21 的 Ajax 调用指明了可以执行从远程响应中得到的 Javascript。

例 11-21 授权执行 Ajax 响应中的脚本

```
<div id="feedback"></div>
// 如果 post/delete 动作的响应中含有 JavaScript，
// 允许其在浏览器中执行。
<?php echo link_to_remote('Delete this post', array(
    'update' => 'feedback',
    'url'    => 'post/delete?id='.$post->getId(),
    'script' => true,
)) ?>
```

如果远程模板包含像 remote_function()之类的 Ajax 辅助函数，要记住这些 PHP 函数会生成 Javascript 代码，如果你不加上'script' => true 选项，这些 Javascript 代码就不会被执行。

NOTE 对于远程响应，虽然你允许脚本执行，但如果你想用某种工具去查看生成的代码，你是看不到远程代码中的脚本的。这些脚本只会执行却不会出现在代

码中。虽然看起来这有些奇怪，但这种处理却是很正常的。

## 创建回调函数

Ajax 交互的一个严重缺点就是在要更新的区域被更新之前，用户感受不到这种交互。也就是说，如果网络很慢或服务器运行失败，用户以为动作已经被处理了，而实际上，动作根本就没有被执行。所以，将 Ajax 交互中发生的事件传达给用户是非常重要的。

每个远程请求都被默认为一个异步过程，在这个过程中，可以触发各种 javascript 回调函数，诸如进度条之类。所有的回调函数都可以访问 request 对象，该对象暗含着 XMLHttpRequest。回调函数对应于 Ajax 交互中发生的以下事件：

- before：在初始化请求之前
- after：在初始化请求之后并在导入之前
- loading：当远程响应正被浏览器导入时
- loaded：当远程响应被浏览器导入完成时
- interactive：当用户可以和远程响应交互时，即使该响应还没有完全导入
- success：当 XMLHttpRequest 已完成，而且 HTTP 状态码在 2XX 的范围内
- failure：当 XMLHttpRequest 已完成，而且 HTTP 状态码不在 2XX 的范围内
- 404：当请求返回 404 状态时
- complete：当 XMLHttpRequest 完成时（如果存在的话，将会在 success 或 failure 之后触发）

例如，我们常会在初始化远程调用时显示一个导入进度条，而在收到响应后隐去进度条。要达到这个目的，只要在 Ajax 调用中简单地加入 loading 和 complete 参数即可，如例 11-22 所示。

例 11-22 用 Ajax 回调函数显示和隐藏一个活动进度条

```
<div id="feedback"></div>
<div id="indicator">Loading...</div>
<?php echo link_to_remote('Delete this post', array(
    'update'   => 'feedback',
    'url'      => 'post/delete?id='.$post->getId(),
    'loading'  => "Element.show('indicator')",
    'complete' => "Element.hide('indicator')",
)) ?>
```

show 和 hide 方法以及 Javascrip Element 对象都是 Prototype 中有用的工具。

# 创建视觉效果

~~Symfony~~symfony 中集成了 script.aculo.us 库的视觉效果，这个库可以帮助你完成不仅仅是在页面中显示和隐藏 元素的功能。在 http://script.aculo.us/中可以了解有关文档。简单地讲，这个库提供了许多为了获得复杂视觉效果而操纵 DOM 的 Javascript 对象和功能。例 11-23 中列出了一些示例。因为结果是 web 页中某个区域的视觉模拟，所以建议你测试一下它们的效果，以便理解它们到底是如何运行的。Script.aculo.us 网站提供了动态效果的汇总，你可以从中选择。

例 11-23 在 Javascript 中用 script.aculo.us 实现视觉效果

```
// 加亮 my_field 元素
Effect.Highlight('my_field', { startcolor:'#ff99ff',
endcolor:'#999999' })

// 为元素添加下拉百叶窗效果
Effect.BlindDown('id_of_element');

// 为元素添加渐隐效果
Effect.Fade('id_of_element', { transition:
Effect.Transitions.wobble })
```

~~Symfony~~symfony 用 visual_effect()辅助函数封装了 Javascript 的 Effect 对象，这个辅助函数是 Javascript 辅助函数组的成员，它将输出通常的链接会用到的 Javascript 代码，如例 11-24 所示。

例 11-24 用 visual_effect()辅助函数在模板中实现视觉效果

```
<div id="secret_div" style="display:none">I was here all along!</div>
<?php echo link_to_function(
  'Show the secret div',
  visual_effect('appear', 'secret_div')
) ?>
// 将调用 Effect.Appear('secret_div')
```

visual_effects()辅助函数还可以用在 Ajax 回调函数中，例 11-25 实现了一个类似例 11-22 的活动进度条，但是视觉感受更舒适。当启动 Ajax 调用时，indicator 元素渐渐显现出来，而当响应到达时，indicator 元素又渐渐隐去。另外，为了吸引用户注意 feedback 元素，当远程调用更新它以后，它会被加亮显示。

例 11-25 Ajax 回调函数的视觉效果

```
<div id="feedback"></div>
<div id="indicator" style="display: none">Loading...</div>
<?php echo link_to_remote('Delete this post', array(
    'update'   => 'feedback',
    'url'      => 'post/delete?id='.$post->getId(),
    'loading'  => visual_effect('appear', 'indicator'),
    'complete' => visual_effect('fade', 'indicator').
                  visual_effect('highlight', 'feedback'),
)) ?>
```

注意观察是如何在回调函数中通过链接方式将各种视觉效果混合起来的。

# JSON

Javascript 对象表示模型（JavaScript Object Notation，JSON）是一个轻量级的数据交换格式。简单地讲，它就是用于传送对象信息的 Javascript 数组，如例 11-26 中的例子所示。它对 Ajax 交互来说有两个主要的好处，一个是 Javascript 易于读取它，二是它可以减少 web 响应的数据量。

例 11-26 Javascript 中的 JSON 对象

```
var myJsonData = {"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

如果一个 Ajax 动作需要返回结构化的数据给调用页面，以便 javascript 进一步处理的话，JSON 是一种比较适合作为响应的格式。例如，当一个 Ajax 调用想更新调用页面的多个元素时，JSON 就非常有用。

设想有一个像例 11-27 那样的调用页面，其中含有两个需要更新的元素。一个远程辅助函数只能更新页面中的一个元素（要么是 title，要么是 name），而不能同时更新两个。

例 11-27 用于多个 Ajax 更新的模板示例

```
<h1 id="title">Basic letter</h1>
<p>Dear <span id="name">name_here</span>,</p>
<p>Your e-mail was received and will be answered shortly.</p>
<p>Sincerely,</p>
```

要更新两者，把 Ajax 响应想象成一个包括以下数组的 JSON 头：

```
 [["title", "My basic letter"], ["name", "Mr Brown"]]
```

远程调用就能轻易地解析这个响应，并且稍稍利用 Javascript 就可以更新一行中的几个域。为达到这个效果，将例 11-28 中的代码加进例 11-27 的模板即可。

例 11-28 从一个远程响应更新多个元素

```
<?php echo link_to_remote('Refresh the letter', array(
  'url'      => 'publishing/refresh',
  'complete' => 'updateJSON(request, json)'
)) ?>

<?php echo javascript_tag("
function updateJSON(request, json)
{
  var nbElementsInResponse = json.length;
  for (var i = 0; i < nbElementsInResponse; i++)
  {
     Element.update(json[i][0], json[i][1]);
  }
}
") ?>
```

complete 回调函数可以访问响应的 json 头并把它传递给第三方函数。这个 updateJSON()函数对 JSON 头进行迭代，对数组的每个成员，用第二个参数设置的元素名称去更新第一个参数设置的元素名称。例 11-29 显示了 publishing/refresh 动作如何返回一个 JSON 响应。

例 11-29 返回 JSON 头的动作示例

```
class publishingActions extends sfActions
{
  public function executeRefresh()
  {
    $output = '[["title", "My basic letter"], ["name", "Mr Brown"]]';
    $this->getResponse()->setHttpHeader("X-JSON", '('.$output.')');
```

```
    return sfView::HEADER_ONLY;
  }
```

HTTP 协议允许在响应头中存放 JSON。因为响应不包含任何内容，动作只是立即将它作为一个头发送出去。这样它就完全跳过了视图层，不仅和->renderText()一样快，而且更小。

**CAUTION** 例 11-29 中的方法有一个服务器的限制，即 HTTP 头的最大容量。虽然没有正式的规定，但浏览器有时不能很好地转换和解释大的 HTTP 头。也就是说，如果你的 JSON 数组比较大，那么远程动作应该将 JSON 数组放在一个标准的响应中，而不是将 JSON 数组放在 HTTP 头中。

JSON 已经成为 web 应用的一个标准。Web 服务经常建议用 JSON 而不是 XML 去响应，以便将服务集成到客户端(mashup)而不是服务器端。所以如果你要选择用哪种格式在服务器和 javascript 函数间进行通信，JSON 也许是最好的选择。

**TIP** 从 PHP5.2 开始，PHP 提供了 json_encode()和 json_decode()两个函数，以便你在 PHP 语法和 JSON 语法之间进行数组转换。这有助于 JSON 数组和 Ajax 的集成，请参考(http://www.php.net/manual/en/ref.json.php) 的说明。

# 用 Ajax 完成复杂的交互

在 symfony 的 Ajax 辅助函数里，有些工具只需一个简单的调用，你就可以完成复杂的交互功能。有了类似桌面应用的交互功能（拖拽，自动完成和实时编辑），你无需编写复杂的 javascript 代码就可以提高用户体验。下面几节将描述这些用于复杂交互的辅助函数，并给出简单的例子。其他的参数和技巧请参考 script.aculo.us 的文档。

**CAUTION** 如果你要提供复杂的交互功能，界面显示将花费更多的时间才能达到自然的效果。只有你确信这样有助于提高用户体验时，才使用复杂的交互功能。否则应该避免使用。

## 自动完成

用户在一个文本录入控件中输入字符时，如果能列出与用户输入匹配的一些词汇，这就称为自动完成。如果远程动作以例 11-30 那样的 HTML 列表形式返回响应时，利用 input_auto_complete_tag()辅助函数，你就可以达到这个效果。

例 11-30 能够使用自动完成标记的响应示例

```
<ul>
  <li>suggestion1</li>
  <li>suggestion2</li>
  ...
```

</ul>

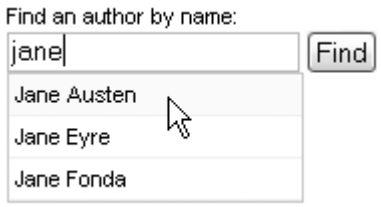
你可以仿照例 11-31 中的例子，在进行常规文本输入时，将辅助函数插入模板。

例 11-31 在模板中使用自动完成标记辅助函数

```
<?php echo form_tag('mymodule/myaction') ?>
  Find an author by name:
  <?php echo input_auto_complete_tag('author', 'default name',
    'author/autocomplete',
    array('autocomplete' => 'off'),
    array('use_style'    => true)
  ) ?>
  <?php echo submit_tag('Find') ?>
</form>
```

当用户每次在 author 域中输入一个字符，它就会调用 author/autocomplete 动作。你可以设计动作，以便根据 author 请求参数确定一个可能的匹配列表，并且以类似例 11-30 的格式返回。然后辅助函数就会在 author 标记下显示这个列表，当用户点击其中的一个提示或用键盘选择一个提示时，输入就可以完成。图 11-3 是它的图示。

图 11-3 一个自动完成的例子




input_auto_complete_tag() 辅助函数的第三个参数可以取以下参数：

- use_style：自动控制响应列表的形式。
- frequency：周期性调用的频率（默认值为 0.4 秒）。
- tokens：开通标记化的递增自动完成功能。例如，如果你设置了这个参数为，而用户输入了 jane,george，则动作只接受'george'值。

NOTE 和其它辅助函数一样，input_auto_complete_tag() 辅助函数也接受本章前面描述的常用的远程辅助函数选项。特别是设置 loading 和 complete 视觉效果，可以得到更好的用户体验。

## 拖放

在桌面应用中用鼠标将一个元素拖放到某个地方是很常见的，但在浏览器中却

极少见。这是因为用 Javascript 编写这些行为是非常复杂的。但是在 symfony 中却可以方便地只用一行代码就能实现。

symfony 框架提供了两个辅助函数: draggable_element()和 drop_receiving_element()，它们可以被看成是行为修改器(modifier)，他们为相应的元素加上发现器(observer)和拖动能力，可以声明一个元素是可拖动的或是可拖动元素的释放接收元素。一个可拖动元素就是能用鼠标点击抓取的元素，只要鼠标按钮未松开，这个元素就可以在整个窗口中移动。当这个可拖动元素被释放，一个接收元素就会去调用一个远程函数。例 11-32 显示了用一个购物车接收元素进行交互的例子。

例 11-32 购物车中的可拖放元素和释放接收元素

```
<ul id="items">
  <li id="item_1" class="food">Carrot</li>
  <?php echo draggable_element('item_1', array('revert' => true)) ?>
  <li id="item_2" class="food">Apple</li>
  <?php echo draggable_element('item_2', array('revert' => true)) ?>
  <li id="item_3" class="food">Orange</li>
  <?php echo draggable_element('item_3', array('revert' => true)) ?>
</li>
<div id="cart">
  <p>Your cart is empty</p>
  <p>Drag items here to add them to your cart</p>
</div>
<?php echo drop_receiving_element('cart', array(
  'url'        => 'cart/add',
  'accept'     => 'food',
  'update'     => 'cart',
)) ?>
```

无序列表中的每一项都可以被鼠标拖动并在整个窗口中移动。鼠标松开时，它们将返回开始的位置。如果在 cart 元素上松开鼠标，它就触发一个调用 cart/add 动作的远程函数。这个动作根据 id 请求参数确定该将哪个项目放进 cart 元素。例 11-32 模拟了一次真正的购物会话：你可以抓取所需项目放进购物车，然后结帐。

**TIP** 在例 11-32 中，辅助函数正好写在要修改的元素的后面，但你也可以不这样安排，你可以在模板的最后将 draggable_element()和 drop_receiving_element()分组列出。重要的是辅助函数调用的第一个参数，它指明要接受行为的元素标识符。

draggable_element()辅助函数接受以下参数：

- revert：如果为真，当释放鼠标时，元素将返回原来位置。它也可以是任意的函数指针，用于拖动结束时调用。
- ghosting：克隆一个元素并且拖动这个复制品，放下这个复制品之前，原先的元素留在原地不动。
- snap：如果为假，则不会出现快照。否则，可拖动元素只能被拖拽到由 xy 或 [x,y] 或 function(x,y){ return [x,y] }指定的 xy 交叉点。

drop_receiving_element() 辅助函数接受以下参数：

- accept：描述 CSS 类的一个字符串或一个字符串数组。只有当可拖放元素有一个或多个 CSS 类时，才能被接受。
- hoverclass：当用户在一个元素上拖动一个被接受的可拖动元素，则将 CSS 类加到这个元素上。

## 可排序列表

可拖动元素还能用鼠标移动列表的选项，来对一个列表进行排序。
sortable_element()辅助函数将可排序行为加给选项，例 11-33 是实现这个特性的一个例子。

例 11-33 可排序列表

```
<p>What do you like most?</p>
<ul id="order">
  <li id="item_1" class="sortable">Carrots</li>
  <li id="item_2" class="sortable">Apples</li>
  <li id="item_3" class="sortable">Oranges</li>
  // Nobody likes Brussel sprouts anyway
  <li id="item_4">Brussel sprouts</li>
</ul>
<div id="feedback"></div>
<?php echo sortable_element('order', array(
  'url'    => 'item/sort',
  'update' => 'feedback',
  'only'   => 'sortable'
)); ?>
```

借助于 sortable_element() 辅助函数的神奇能力，<ul>元素成为可排序的，也就是说，它的子元素可以通过拖放被重新排序。每当用户通过拖动并释放一个选项来重新排序时，就会产生一个带有以下参数的 Ajax 请求：

```
POST /sf_sandbox/web/frontend_dev.php/item/sort HTTP/1.1
  order[]=1&order[]=3&order[]=2&_=
```

整个有序列表作为一个数组传送，格式是order[$rank]=$id，$order从0开始，$id 则是列表元素中id属性中下划线（“_”）后的值。

sortable_element()辅助函数接受以下参数：

- only：描述CSS类的一个字符串或一个字符串数组。只有具有这个类的可排序元素的子元素才可以被移动。
- hoverclass：当鼠标在元素上浮动时，将CSS类加到元素上。
- overlap：如果多个选项显示在同一行，则设为horizontal；如果每行显示一个选项（如例中所示），则设为vertical。
- tag：如果要排序的列表不是一组<li>元素，则你必须指定可排序元素中的哪些子元素是可以被拖放的（例如div 或dl）。

## 就地编辑

越来越多的互联网应用程序允许用户在页面中直接编辑页面内容，而无需在表单中重新显示内容。这种交互的原理很简单。当用户将鼠标浮动到一块文本上时，这块文本将被加亮。如果用户在这个块中点击，普通文本就转换为填写了文本的文本区，并且出现一个保存按钮。用户就可以在文本区中编辑，保存之后，文本区消失，普通文本重新出现。在symfony中，用input_in_place_editor_tag()辅助函数就可以将这个可编辑行为加给一个元素。例11-34显示了这个辅助函数的使用。

例11-34 可编辑的文本

```
<div id="edit_me">You can edit this text</div>
<?php echo input_in_place_editor_tag('edit_me', 'mymodule/myaction',
array(
  'cols'         => 40,
  'rows'         => 10,
)) ?>
```

当用户点击可编辑文本时，它就被替换为一个填写了文本的文本输入区，这个输入区是可编辑的。当提交表单时，Ajax将编辑过的值作为value参数调用mymodule/myaction动作，动作完成的结果就是更新了可编辑元素。写这样的代码非常快而且很有用。

input_in_place_editor_tag()辅助函数接受以下参数：

- cols和rows：指明用于编辑的文本输入区的大小（如果rows大于1，就成为<textarea>）。
- loadTextURL：指明动作的URI，调用该动作可以显示要编辑的文本。如果可编辑元素的内容使用特殊的格式，并且你想让用户不用格式化就可以编辑该文本时，这个参数非常有用。

- save_text 和 cancel_text：保存链接（默认值是 ok）和取消链接（默认值是 cancel）上的文本。

# 总结

如果你不喜欢在模板中用 Javascript 来实现客户端的行为，你可以用 Javascript 辅助函数。它们不仅能自动实现基本的链接和元素更新，还提供一种快速开发 Ajax 交互的方法。Prototype 有力地增强了 Javascript 语法，而 script.aculo.us 则提供了强大的视觉效果，有了它们的帮助，你只需要几行代码就可以写出复杂的交互程序。

因为用 symfony 生成高度交互性的应用就像写静态页面那样容易，所以你可以在 web 应用中实现大多数桌面应用程序的交互功能。