第 11 章 用Ajax改进blog

11

Ajax 技术使我们获得的能力之一是，在静态的网页上提供额外的用户交互。这意味着你可以在提供无中断的用户体验的同时修改静态页面的工作方式。

可以接受这类改进的领域之一是网络日志（简称 blog，又称博客）。若仅从纯数据的角度看，网络日志不过是日志项的一个列表，其中每项都包含一个正文段、标题和到全文的链接。不过浏览旧日志或检查新日志的功能就非常有限了。

在本章里，你能看到两种用基于 Ajax 技术的 JavaScript 代码来改进传统网络日志的方法。其一用于快速滚动一长串日志项列表，而无需离开当前页面；其二在不需要反复刷新页面的情况下就能看到新日志项的出现。

11.1 永不终止的 blog

要改进 blog 的第一部分是，能在甚至不需要点击导航链接的情况下回滚到所有存档。blog 或其他以时间为单位记录的内容站点的常见特性之一是导航查阅旧的日志。通常在页面底部会有一个“下一页”或“上一页”链接，给用户提供存档的导航。

你将在这里看到如何用 Ajax 把整个过程环绕一种方法来实现。在实现之前，需要作如下假定：

- 你有一个网页，其中有许多按时间排序的条目。
- 当用户滚动到页面底部时，意味着他们希望读到更多先前的条目。
- 你有一个可以读取所有条目的数据源。在这里我们要用的WordPress blog软件(http://wordpress.org/)就对此支持得很好。

这个脚本的效果是，只要用户滚动到了页面底部附近，就会自动载入其他文章，让用户能继续滚动并浏览存档，使他们产生一种页面永不终止的错觉。它将使用 WordPress 网志软件提供的功能来构建。如果你用的就是基于 WordPress 的 blog，可以简单地把这个脚本放入其中，给它添加这个新功能。

11.1.1 blog 的模板

你将使用默认 WordPress 安装时提供的一个称作 Kubrik 的基本模板来起步，这个模板很受欢迎。图 11-1 展示了一个基本 Kubrik 主题页面。

图11-1 WordPress的默认Kubrik主题

你可以发现，这个页面有一个主栏位、一个标题栏和一个侧栏。主栏位[①]是最重要的区域，你可以在此看到网志文章，并添加新的信息。让我们看看构成这个blog结构化、简化后的HTML，如代码清单 11-1 所示。

代码清单11-1 Kubrik主题和WordPress生成的HTML的简化版本

```
<html>
<head>
    <title>Never-ending Wordpress</title>
    <script>
    <!-- 我们的脚本放在这里 -->
    </script>
</head>

<body>
    <div id="page">
        <div id="header">
            <!-- 标题栏内容 -->
```


① 原文为标题栏，应该是主栏位。——译者注

```
        </div>
        <div id="content">

            <!-- 第一篇文章 -->
            <div class="post">
                <!-- 文章的标题 -->
                <h2><a href="/test/?p=1">Test Post</a></h2>
                <small>October 24th, 2006</small>

                <div class="entry">
                <!-- 文章的内容 -->
                </div>

                <p class="postmetadata">
                    <a href="/test/?p=1#comments">Comments</a>
                </p>
            </div>

            <!-- 更多文章... -->

        </div>
    </div>
</body>
</html>
```

注意，所有的网志项目都包含在ID为"content"的 中。此外，所有文章的结构都用一种专门的格式规划。然后，你需要构建一套简单的DOM函数来提取这个网志页面的数据。代码清单 11-2 展示了完成这个页面里需要的DOM操作。

代码清单11-2 添加HTML以完成页面的DOM操作

```
// 我们要把新文章载入到 id 为 "content" 的 <div> 中
var content = document.getElementById("content");

// 我们将遍历 RSS feed 中所有的文章
var items = rss.getElementsByTagName("item");
for (var i = 0; i < items.length; i++) {

    // 让我们从每篇 feed 文章中解析出链接、标题和描述数据
    var data = getData( items[i] );

    // 创建一个新的 <div> 用来包裹这篇文章
    var div = document.createElement("div");
    div.className = "post";

    // 创建文章标题
    var h2 = document.createElement("h2");

    // 这将包含 feed 的标题和到文章的链接
    h2.innerHTML = "<a href='" + data.link + "'>" + data.title + "</a>";

    // 将它添加到封装它的 <div> 中
    div.appendChild( h2 );
```

```
    // 现在创建一个 <div> 来存放比较长的部分，即文章内容
    var entry = document.createElement("div");
    entry.className = "entry";

    // 将内容添加到 <div> 内部
    entry.innerHTML = data.desc;
    div.appendChild( entry );

    // 最后，让我们添加一个有返回链接的底部
    var meta = document.createElement("p");
    meta.className = "postmetadata";

    var a = document.createElement("a");
    a.href = data.link + "#comments";
    a.innerHTML = "Comment";
    meta.appendChild( a );

    div.appendChild( meta );

    // 将这个新项目插入文档
    content.appendChild( div );

}
```

然而，如果你不了解处理的是什么数据，所有这些 DOM 操作都没什么意义。下一节里你将看到从服务器传给你的数据，以及如何使用这些 DOM 操作将它插入文档中。

11.1.2 数据源

WordPress 提供了访问网志文章的一套简单方法，即可以通过一个默认 RSS feed 阅读 10 篇最新的文章。不过仅通过 RSS feed 是不够的，因为你可能还需要访问所有的文章，甚至返回到网站本身。这时你可以用一个隐藏特性来实现这个功能。

236

在 WordPress 中，RSS feed 的 URL 通常类似这样：/blog/?feed=rss，不过只要再加上一个参数，/blog/?feed=rss&paged=N，你就可以访问到 blog 任意时间以前的历史记录。当 N 等于 1 时得到最新的 10 篇文章，等于 2 时得到它们之前的 10 篇。代码清单 11-3 展示了包含文章数据的 RSS feed 是什么样子的。

代码清单11-3 WordPress返回的XML RSS feed，包含10篇格式化好的文章

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0">

<channel>
    <title>Test Wordpress Web log</title>
    <link>http://someurl.com/test/</link>
    <description>Test Web log.</description>
    <pubDate>Fri, 08 Oct 2006 02:50:23 +0000</pubDate>
    <generator>http://wordpress.org/?v=2.0</generator>
```

```
    <language>en</language>

    <item>
        <title>Test Post</title>
        <link>http://someurl.com/?p=9</link>
        <pubDate>Thu, 07 Sep 2006 09:58:07 +0000</pubDate>
        <dc:creator>John Resig</dc:creator>
        <category>Uncategorized</category>
        <description><![CDATA[ 这里存放文章内容……]]></description>
    </item>

    <!-- 大量其他项目 -->
</channel>
</rss>
```

通过分析 RSS feed，你获得了一个格式良好的 XML 文件可供使用（而且 JavaScript 非常适于遍历）。代码清单 11-4 展示了用于遍历这份 RSS XML 文档并从中解析相关数据的必要代码。

代码清单11-4 从XML RSS Feed中解析文章信息

```
// 我们将遍历 RSS feed 中的每篇文章
var items = rss.getElementsByTagName("item");

for (var i = 0; i < items.length; i++) {
    // 从 RSS feed 的 item 元素中解析出标题、描述和链接
    var title = elem.getElementsByTagName("title")[0].firstChild.nodeValue;
    var desc = elem.getElementsByTagName("description")[0].firstChild.nodeValue;
    var link = elem.getElementsByTagName("link")[0].firstChild.nodeValue;
}
```

237

在保证数据来源完整，而插入 HTML 文档的结构也稳定的情况下，现在可以用 Ajax 请求和一些基本的事件检测方法把代码黏合在一起了。

11.1.3 事件检测

要激活你的脚本，用户需要完成的主要交互是滚动到页面的底部，所以不管浏览器的视口（viewport）移动到哪里，你必须能判断出它是否在页面的底部。

处理这个的脚本相对简单，你只需要给窗口的滚动事件绑定一个简单的事件处理器。每当用户移动页面的视口（有可能是移动到了页面底部附近），这个事件处理器都能让你知道。所以你只需要使用第 7 章定义的一套简单方法来判断用户的视口到底在哪里。`pageHeight`（判断整个页面有多高），`scrollY`（获知当前视口的顶部滚动到了哪里）和`windowHeight`（获知视口有多高）。如代码清单 11-5 所示。

代码清单11-5 判断用户视口的位置

```
// 我们要根据当前用户在页面中所处的位置判断是否应该载入更多内容
window.onscroll = function(){
    // 检查视口在页面中的位置
    if ( curPage >= 1 && !loading &&
        pageHeight() - scrollY() - windowHeight() < windowHeight() ) {
```

```
            // 用 Ajax 请求来获取 RSS XML feed
        }
    };
```

现在你拥有所需的所有组件了，最后一步是使用 Ajax 请求来获取数据、无缝衔接所有部分。

11.1.4 请求

整个程序的核心要和一段 Ajax 请求结合在一起，并动态载入一部分文章或条目，以便插入页面中。要发送的请求很简单：建立到某个 URL（指向下一部分文章的 URL）的 HTTP GET 连接，并获取这个 URL 指定的 XML 文档。代码清单11-6使用第10章的完整 Ajax 函数就实现了这一点。

238

代码清单11-6 载入一部分新文章的Ajax请求

```
// 用我们方便的 ajax() 函数来载入文章
ajax({

    // 我们只不过是请求一个简单网页，所以就用 GET
    type: "GET",

    // 要获得的 RSS feed 就是一个 XML 文件
    data: "xml",

    // 获得第 N 个页面的 RSS feed。在我们受此载入这个页面时处在第
    // 1 页，所以从 2 开始往回追溯
    url: "./?feed=rss&paged=" + ( ++curPage ),

    // 等待成功获得 RSS feed
    onSuccess: function( rss ){
        // 通过 DOM 来遍历 RSS XML 文档
    }

});
```

构建好发出页面请求的方法后，你就可以把所有功能结合在一起成为一个紧凑的程序包，简单地放在 WordPress blog 里。

11.1.5 结果

将 DOM 构建代码和 RSS XML 遍历代码合在一起，就得到了这个程序最简单的形式，而一旦加上滚动事件的检测和 Ajax 请求，你就得到了对 blog 的一项很吸引人的改进——不离开当前页面的情况下持续滚动所有 blog 文章的能力。代码清单 11-7 展示了用这个功能改进 WordPress blog 所需的完整代码。

代码清单11-7 WordPress blog里添加永不终止的页面功能所需的JavaScript代码

```
// 记录我们目前所在的页面编号
```

```
var curPage = 1;

// 确保我们不会同一时间重复载入同一个页面
var loading = false;

// 我们要根据当前用户在页面中所处的位置判断是否应该载入更多内容
window.onscroll = function(){

    // 我们要在尝试载入新内容前验证几件事情：
    // 1) 必须确保不在内容的最后一页。
    // 2) 必须确保现在不是正在载入新文章。
    // 3) 只在滚动到靠近页面底部才尝试载入新文章。
    if ( curPage >= 1 && !loading &&
        pageHeight() - scrollY() - windowHeight() < windowHeight() ) {

        // 记住我们现在开始载入新文章了。
        loading = true;

        // 用我们方便的 ajax() 函数来载入文章
        ajax({

            // 我们只不过是请求一个简单网页，所以就用 GET
            type: "GET",

            // 要获得的 RSS feed 就是一个 XML 文件
            data: "xml",

            // 获得第 N 个页面的 RSS feed。在我们受此载入这个页面时处在第
            // 1 页，所以从 2 开始往回追溯
            url: "./?feed=rss&paged=" + ( ++curPage ),

            // 等待成功获得 RSS feed
            onSuccess: function( rss ){

                // 我们要把新文章载入到 ID 为 "content" 的 <div> 中
                var content = document.getElementById("content");

                // 我们将遍历 RSS feed 中所有的文章
                var items = rss.getElementsByTagName("item");

                for (var i = 0; i < items.length; i++) {
                    // 将这个新项目插入文档
                    content.appendChild(makePost( items[i] ) );
                }

                // 如果 XML 文档中已无任何项目，我们必须后退到最早的位置
                if (items.length == 0) {
                    curPage = 0;
                }
            },

            // 只要请求完成，我们就可以允许再次尝试载入新的内容了
            onComplete:function(){
                loading = false;
```

```
                }
            });
        }
    };

    // 为一篇文章创建复杂 DOM 结构的函数
    function makePost( elem ) {
        // 让我们从每篇 feed 文章中解析出链接、标题和描述数据
        var data = getData( elem );

        // 创建一个新的 <div> 用来包裹这篇文章
        var div = document.createElement("div");
        div.className = "post";

        // 创建文章标题
        var h2 = document.createElement("h2");

        // 这将包含 feed 的标题和到文章的链接
        h2.innerHTML = "<a href='" + data.link + "'>" + data.title + "</a>";

        // 将它添加到包裹它的 <div> 中
        div.appendChild( h2 );

        // 现在创建一个 <div> 来存放比较长的部分，文章内容
        var entry = document.createElement("div");
        entry.className = "entry";

        // 将内容添加到 <div> 内部
        entry.innerHTML = data.desc;
        div.appendChild( entry );

        // 最后，让我们添加一个有返回链接的底部
        var meta = document.createElement("p");
        meta.className = "postmetadata";
        meta.innerHTML = "<a href='" + data.link + "#comments'>Comment</a>";
        div.appendChild( meta );

241     return div;
    }

    // 从 DOM 元素中解析数据的简单函数
    function getData( elem ) {
        // 返回数据是格式化良好的对象
        return {
            // 从 RSS feed 的 <item> 元素中解析出标题、描述和链接
            title:elem.getElementsByTagName("title")[0].firstChild.nodeValue,
            desc:elem.getElementsByTagName("description")[0].firstChild.nodeValue,
            link:elem.getElementsByTagName("link")[0].firstChild.nodeValue
        };
    }
```

将这些代码添加到你的 WordPress 的 header 模板中应该足以达到类似图 11-2 的效果了。注意滚动条会变得很小，这说明已经动态载入了额外的文章。

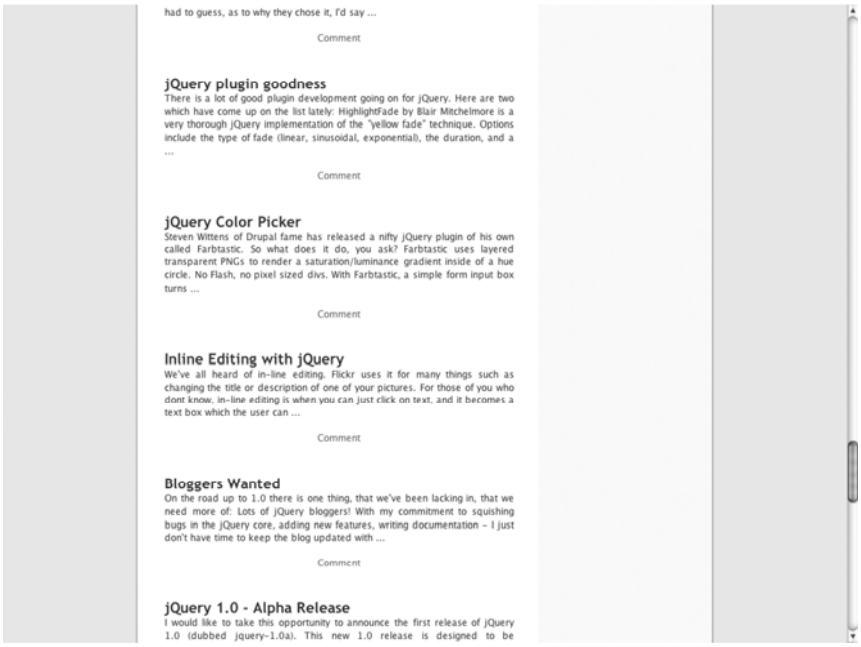

图11-2 额外的内容随向下滚动页面被载入到视口

242

动态内容载入是 Ajax 的常见用途之一。可以肯定的是，这种技巧能简化用户浏览页面的过程，还能减轻服务器的负荷。请求额外信息的应用程序当然并不限于网络日志。基于 Web 的应用程序都能够从这种技术中大大获益，你可以在第 12 章看到更多关于这方面的内容。

在下一节里，你会见到使用同一种 Ajax 技巧来动态载入内容的另一个例子，在这个例子里，能给浏览者提供实时的网志体验。

11.2 实时网志

已经完成了所有比较困难的工作：动态构建文章、获取文章数据和解析数据，现在可以直接在 WordPress 网志里体验到这些工作的另一应用了。

新闻和更新的即时性是传统网络日志比较缺乏的。用户看到的是一个静态页面，列出了最新 blog 中的文章，如果他们需要看到更新，就必须在浏览器中刷新页面。不过许多时候，网志的主人（blogger）都希望能即时告知用户一段信息，就算用户当时还在阅读当前页面。理论上说，应该允许用户载入页面，保持开启，偶尔（从别的窗口或者 Tab）切换回来，就能见到新的文章。

这又是一个 Ajax 技术的应用，你可以使用在上一节提及的技巧来从 RSS XML feed 里载入新内容。下面列出了给普通 blog 创建实时网志体验的必要操作：

- 以固定间隔时间（比如每分钟一次），获取网络日志中最近发布的文章列表。
- 找出未曾显示的文章。
- 将这些文章添加到页面的顶部。

这个操作的流程很简单，实现也简单，如代码清单 11-8 所示。

代码清单11-8 从基于XML的RSS feed即时更新blog的实现

```
// 我们将持续地定期载入新的页面
setInterval(function(){

    // 用我们方便的 ajax() 函数来载入文章
    ajax({

        // 我们只不过是请求一个简单网页，所以就用 GET
        type: "GET",

        // 要获得的 RSS feed 就是一个 XML 文件
243     data: "xml",

        // 获取当前的 RSS feed （包含最新的文章）
        url: "./?feed=rss&paged=1",

        // 等待成功获得 RSS feed
        onSuccess: function( rss ){

            // 我们要把新文章载入到 ID 为 "content" 的 <div> 中
            var content = document.getElementById("content");

            // 获得当前 （还未更新的） 页面中，最新文章的 URL （以避免出现冗余）
            var recentURL = content.getElementsByTagName("h2")[0].firstChild.href;

            // 我们要遍历 RSS feed 中的所有文章
            var items = rss.getElementsByTagName("item");

            // 我们将把所有新的项目放入一个单独的数组中
            var newItems = [];

            // 遍历每个项目
            for ( var i = 0; i < items.length; i++ ) {

                // 如果找到了“旧”文章，强制停止循环
                if ( getData(items[i]).link == recentURL )
                    break;

                // 将这个新项目添加到临时数组中
                newItems.push(items[i]);

            }

            // 反向遍历所有新的项目，以保证他们插入页面的顺序正确
            for (var i = newItems.length-1; i >= 0; i--) {
                // 将新项目插入文档
                content.insertBefore(makePost( newItems[i] ), content.firstChild );
            }

        }
    });

// 每分钟载入一次新页面
244 }, 60000 );
```

如果你把这个脚本（和 11.1 节的脚本一起）放入 WordPress 模板中，就能得到类似图 11-3 的结果。

图11-3 在用户没有刷新页面的情况下WordPress自动把新文章插入到旧文章的前面

通过这个改进，可以有效地把一个简单的 blog 转换为一个实时网志平台。如果在参加某个会议时需要即时更新 blog，你就可以把这段脚本加入站点，读者就能迅速获得更新，而不需要手动刷新页面了。

11.3 小结

本章讨论的最重要概念是，Ajax 相关的技术让你能为传统的静态应用程序构想新的运作方式。因为处理 XML 文档并将其转换为可用的 HTML 是如此简单，在你自己的应用程序里实现起来也是完全可能的。 245

本章给传统的 WordPress blog 平台构建了两个新改进。第一，你把传统的链接加分页的导航方法替换为了在用户浏览时动态载入文章的方法。第二，如果在用户阅读页面时发表了新文章，会被即时添加出来，而且并不影响用户的阅读。这两个改进都能使浏览体验变得更具动态性、更平滑，而不受多页间断的影响。

下一章我们将构建一个先进的 Ajax 功能：自动补全的搜索。 246