

## 实战 Firefox 扩展开发

[成 富](#), 软件工程师, IBM 中国软件开发中心

**简介:** Firefox 浏览器自身提供良好的扩展结构, 使得开发人员可以方便的扩展其行为。很多网站, 比如 [del.icio.us](#), 都提供 Firefox 扩展来提供更好的用户体验。学习这方面的知识不仅对于网站开发人员是有用的, 其他人也可以通过开发扩展来解决一些使用 Firefox 中遇到的具体问题。本文以一个能够批量下载某个 HTML 页面上所有图片的 Firefox 扩展作为案例, 详细的介绍了 Firefox 扩展的开发流程。这其中包括构建开发环境, 使用 XUL 来描述用户界面, 使用 [JavaScript](#) 来为扩展增加行为, 扩展的打包、发布和更新等方面的内容。

本文的标签: [firefox](#), [javascript](#), [web](#), [插件](#)

[标记本文!](#)

发布日期: 2008 年 2 月 28 日

级别: 中级

访问情况 11625 次浏览

### 案例介绍

本文中所要构建的是一个能够批量下载某个 HTML 页面上所有图片的 Firefox 扩展。通常我们在浏览包含许多图片的网页时, 如果想要把自己感兴趣的图片全部下载下来, 需要逐一在图片上点击右键, 然后选择另存为, 再选择文件存放的目录, 最后才能把图片保存在本机上。另外一种做法是把整个网页都保存下来, 不过这样会保存不需要的信息, 包括 JavaScript 脚本和 [CSS](#) 文件等, 会增加所需的磁盘空间, 浏览起来也不方便。该扩展要做的事情就是把网页上所有的图片在一个新窗口中列出来, 用户可以勾选其感兴趣的图片, 并指定需要保存的目录。然后该扩展能够一次性把用户选择的图片都下载下来。用户以后浏览起来也更加方便。

---

[回页首](#)

### 构建开发环境

在动手开发之前, 首先需要构建扩展开发所需的环境。Firefox 把用户的个人信息, 包括设置、已安装的扩展等, 都保存在一个概要文件中, 默认是使用名为 **default** 的概要文件。通过创建一个专门为开发使用的概要文件, 可以不影响正常的使用, 也不会破坏个人信息。为了创建另外一个概要文件, 运行 `firefox -P`, 在弹出的“选择概要文件”的对话框中, 新建一个名为 **dev** 的概要文件, 并使用此概要文件来运行 Firefox。接下来需要安装几个帮助开发的扩展, 分别是 **Venkman**、**Extension Developer's Extension**、**Console<sup>2</sup>**、**Chrome List** 和 **Firebug**。可以在 [参考资源](#) 部分找到这些扩展的下载地址。最后修改 Firefox 的设置使得调试更加容易。在地址栏输入 `about:config` 可以打开 Firefox 的参数设置页面。按照如下的设置修改参数:

#### 清单 1. Firefox 扩展开发环境参数设置

```
javascript.options.showInConsole = true //把 JavaScript 的出错信息显示在错误控制台
nglayout.debug.disable_xul_cache = true //禁用 XUL 缓存, 使得对窗口和对话框的修改不需要重新加载 XUL 文件
browser.dom.window.dump.enabled = true //允许使用 dump() 语句向标准控制台输出信息
javascript.options.strict       = true //在错误控制台中启用严格的 JavaScript 警告信息
```

至此, 开发环境就构建完成了。当需要进行扩展开发时, 运行 `firefox -P dev` 启动 Firefox 即可。

---

[回页首](#)

构建初始的扩展目录结构

接下来正式进行扩展开发。首先介绍一下一个 **Firefox** 扩展的基本目录结构。

图 1. Firefox 扩展目录结构



在图 1 中，**content** 目录下面存放的是扩展的描述界面的 **XUL** 文件和增加行为的 **JavaScript** 文件。**locale** 目录存放的是本地化相关的文件。如果需要支持英文和中文，就可以在 **locale** 目录下面新建 **en-US** 和 **zh-CN** 目录来存放相应的本地化字符串。**skin** 目录存放的是一些 **CSS** 文件，用来定义扩展的外观。**chrome.manifest** 是 **Chrome** 注册的清单文件（参见 [侧栏](#)）。**install.rdf** 分别包含了扩展安装的信息。

什么是 **Chrome**？

**Chrome** 指的是应用程序窗口的内容区域之外的用户界面元素的集合，这些用户界面元素包括工具条，菜单，进度条和窗口的标题栏等。**Chrome** 提供者能为特定的窗口类型（如浏览器窗口）提供 **chrome**。有三种基本的 **chrome** 提供者：

- 内容（Content）：通常是 **XUL** 文件。
- 区域（Locale）：存放本地化信息。
- 皮肤（Skin）：描述 **chrome** 的外观。通常包含 **CSS** 和图像文件。

在构建了初始的目录结构之后，需要让 **Firefox** 能够识别并加载该扩展。首先需要找到当前的概要文件所对应的目录。在 **Microsoft Windows 2000** 和 **XP** 的电脑上面，该目录是 **C:\Documents and Settings\<您的登录用户名>\Application Data\Mozilla\Firefox\Profiles**。找到该目录之后，可以看到以 **dev** 结尾的目录，那就是我们之前构建的开发环境的概要文件所在的目录。在其下的 **extensions** 目录下面，新建一个文件，其文件名为 **install.rdf** 中指定的扩展的 ID，此处为 **batchimagesdownloader@cn.ibm.com**。该文件的内容就是扩展内容所在的实际目录，比如：**C:\FirefoxExtDev\batchimagesdownloader**。**Firefox** 就能识别并加载我们添加的扩展了。在每次对扩展做了一定的修改之后，不需要重新启动 **Firefox**，只需要安装之前介绍的 **Extension Developer's Extension**，并在 **Tools** 菜单中单击 **Extension Developer > Reload all Chrome** 即可。接下来就可以尝试为扩展添加功能了。

[回页首](#)

构建基本的用户界面

我们首先从用户界面入手。如之前所述，我们希望在新的窗口中显示当前 **HTML** 页面中所有的图片，并可以让用户进行选择。**Firefox** 扩展使用 **XUL** 来描述其用户界面。**XUL** 提供了一套基于 **XML** 的描述方式，可以用来描述用户界面的各种组件，比如按钮、菜单和工具条等。最初始的界面包含显示图片的一个表格以及 **OK** 和 **Cancel** 两个按钮。

清单 2. 基本用户界面的 **XUL** 描述

```
<?xml version="1.0"?>
```

```
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<window
  id="batchimagesdownloader-mainwindow" title="Batch Images Downloader"
  orient="horizontal" onload="mainWindowOnLoad();"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <vbox flex="1">
    <vbox flex="1">
      <label value="Images on this web page:" />
      <spacer style="height: 5px"/>
      <hbox height="500" width="750" style="overflow:auto;">
        <grid>
          <columns>
            <column/>
            <column/>
            <column/>
          </columns>
          <rows id="imagesContainer"></rows> <!-- 显示图片的表格 -->
        </grid>
      </hbox>
    </vbox>
    <spacer style="height: 10px"/>
    <hbox>
      <spacer flex="1"/>
      <button id="mainWindow-add-button" label="OK" default="true"
        oncommand="download();" />
      <button id="mainWindow-cancel-button" label="Cancel" oncommand="close();" />
    </hbox>
    <spacer style="height: 5px"/>
  </vbox>
</window>
```

上面就是显示图片的新窗口的界面元素的声明。下面需要添加用户的交互行为。

---

## [回页首](#)

### 添加菜单事件响应

我们希望当用户点击 **Firefox** 上面的一个菜单项时，弹出刚才构建的新窗口。我们这里要做的是向 **Firefox** 自带的 **Tools** 菜单添加一个新的名为 **Batch Images Downloader** 的菜单项。当用户点击此菜单项时，就会弹出 [代码清单2](#)中定义的窗口。在扩展中可以使用覆盖（**Overlay**）来向已有的界面中添加元素。使用覆盖可以在运行时向一个 XUL 文档添加新的组件。我们在 **chrome.manifest** 中定义了名为 **overlay.xul** 的文件，会对 **Firefox** 已有的用户界面进行一定的修改。只需要在 **overlay.xul** 中添加下面的内容即可：

### 清单 3. 增加菜单项的 XUL 描述

```
<menupopup id="menu_ToolsPopup">
  <menuitem id="batchimagesdownloader-show" label="Batch Images Downloader"
    oncommand="BatchImagesDownloader.show(event);" />
</menupopup>
```

上面定义了当点击菜单项时，会调用 `BatchImagesDownloader.show`方法，这是在 **overlay.js** 中定义的一个 **JavaScript** 方

法, 用来处理新窗口的弹出。`overlay.js` 由 `overlay.xul` 包含进来。

#### 清单 4. 菜单项的事件响应方法

```
var BatchImagesDownloader = {
  show : function() {
    var doc = window.getBrowser().selectedBrowser.contentDocument;
    var imageNodes = doc.getElementsByTagName("img"); //获取所有的 img 节点
    var params = {"imageNodes" : imageNodes};
    this.openWindow("BatchImagesDownloader.mainWindow",
      "chrome://batchimagesdownloader/content/mainWindow.xul",
      "chrome=yes,centerscreen", params);
  },

  //打开一个新的窗口, 或是使得已经创建的窗口获得焦点
  openWindow : function(windowName, url, flags, params) {
    var windowsMediator = Components.classes["@mozilla.org/appshell/window-mediator;1"]
      .getService(Components.interfaces.nsIWindowMediator);
    var aWindow = windowsMediator.getMostRecentWindow(windowName);
    if (aWindow) {
      aWindow.focus();
    }
    else {
      aWindow = window.openDialog(url, windowName, flags, params);
    }
    return aWindow;
  }
};
```

添加上述的代码之后, 可以通过点击 **Tools** 菜单项下来的 **Batch Images Downloader** 菜单项来弹出新的窗口。

#### [回页首](#)

#### 显示图片

可以在 `BatchImagesDownloader.show`方法中看到, 当弹出新窗口的时候, 会把当前页面上的所有 `img` 节点都作为参数传递给新打开的窗口。这些 `img` 节点就是需要展现给用户并供其选择的。接下来要做的就是在新窗口中显示这些图片。在 `JavaScript` 的方法中, 可以像在 `HTML` 中的 `DOM` 操作一样, 对 `XUL` 定义的 `DOM` 树进行修改。这其中包括使用 `document.createElementNS` 来创建新的 `XUL` 元素, 同样也可以使用 `CSS` 来修改 `XUL` 元素的外观。

#### 清单 5. 显示图片的 `JavaScript` 方法

```
const COLUMNS_PER_ROW = 3; //每行显示3张图片
function mainWindowOnLoad() {
  var params = window.arguments[0];
  var imageNodes = params.imageNodes;
  displayImages(imageNodes);
}

function displayImages(imageNodes) {
```

```

imageNodes = imageNodes || [];
var cols = COLUMNS_PER_ROW, row, image, hbox, checkbox;
var rows = document.getElementById("imagesContainer");
for (var i = 0, n = imageNodes.length; i < n; i++) {
    var imageNode = imageNodes[i];
    var imageSrc = imageNode.getAttribute("src");
    if (imageSrc == "") {
        continue;
    }
    if (cols >= COLUMNS_PER_ROW) {
        row = document.createElementNS(XUL_NS, "row"); //开始新的一行
        row.setAttribute("align", "center");
        rows.appendChild(row);
        cols = 0;
    }
    else {
        hbox = document.createElementNS(XUL_NS, "hbox");
        hbox.setAttribute("style", "padding:5px 5px 5px 5px;");
        image = document.createElementNS(XUL_NS, "image");//创建 XUL 图像元素来显示图片
        image.setAttribute("src", imageSrc);
        checkbox = document.createElementNS(XUL_NS, "checkbox");//创建 XUL 复选框元素以供用户选择
        checkbox.setAttribute("imageUrl", imageSrc);
        hbox.appendChild(checkbox);
        hbox.appendChild(image);
        row.appendChild(hbox);
        cols++;
    }
}
}
}

```

我们需要在新窗口加载完成之后，就显示当前页面的所有图片。因此需要注册新窗口的 **onload** 事件的响应方法。这里是 `mainWindowOnLoad`。在 `mainWindowOnLoad` 中，通过 `window.arguments[0]` 可以获得作为打开新窗口的参数传进来的 `img` 节点列表。然后根据这些 `img` 节点的 `src` 属性，创建相应的 XUL 图像元素并显示在表格中，并在每个图片下面创建一个复选框以供用户选择。

[回页首](#)

[下载图片](#)

为了让用户能够下载所选的图片，需要添加新的界面元素让用户可以指定下载图片存放的目录，并提供一个进度条来显示当前的下载进度。

## 清单 6. 支持下载的用户界面 XUL 描述

```

<hbox>
    <label value="Save images to" />
    <textbox id="mainWindow-save-path" readonly="true" style="min-width: 15em;" flex="1"/>
    <button label="Browse..." oncommand="selectSaveDirectory();" />
</hbox>
<spacer style="height: 10px"/>
<progressmeter mode="determined" id="downloadProgress"

```

```
value="0" style="visibility:hidden;"/>
```

该扩展提供了一个默认的图片保存路径，那就是当前用户的根目录。用户也可以选择他想要的保存图片的目录。

## 清单 7. 用户选择图片保存目录的 JavaScript 方法

```
var saveDirectory = getDefaultSaveDirectory();
function selectSaveDirectory() {
    const nsIFilePicker = Components.interfaces.nsIFilePicker;
    var fp = Components.classes["@mozilla.org/filepicker;1"].createInstance(nsIFilePicker);
    fp.init(window, "", nsIFilePicker.modeGetFolder);
    var result = fp.show(); //显示目录选择对话框
    if (result == nsIFilePicker.returnOK) {
        var file = fp.file;
        saveDirectory = file;
        byId("mainWindow-save-path").value = file.path;//把目录的路径显示在文本框中
    }
}
```

//获得默认的图片保存目录，也就是当前用户的根目录

```
function getDefaultSaveDirectory() {
    var file = Components.classes["@mozilla.org/file/directory_service;1"]
        .getService(Components.interfaces.nsIProperties)
        .get("Home", Components.interfaces.nsIFile);
    return file;
}
```

当用户对图片进行了选择，并点击 OK 之后，需要执行图片下载的任务。在下载图片中，会使用 Firefox 的 XPCOM 的实现，请参看 [侧栏](#) 和 [参考资料](#)，获得关于 XPCOM 的更多信息。

## 清单 8. 下载单张图片的 JavaScript 方法

```
function downloadSingleImage(uri, callback) {
    var ios = Components.classes["@mozilla.org/network/io-service;1"]
        .getService(Components.interfaces.nsIIOService);
    var imageURI = ios.newURI(uri, null, null); //创建图像的 URI
    var imageFileName = uri.substring(uri.lastIndexOf("/") + 1);
    var channel = ios.newChannelFromURI(imageURI); //创建读取 URI 指定的数据流的通道
    var observer = {
        onStreamComplete : function(loader, context, status, length, result) {
            var file = Components.classes["@mozilla.org/file/local;1"]
                .createInstance(Components.interfaces.nsILocalFile);
            file.initWithFile(saveDirectory); //图片保存的目录
            file.appendRelativePath(imageFileName);
            var stream = Components.classes["@mozilla.org/network/safe-file-output-stream;1"]
                .createInstance(Components.interfaces.nsIFileOutputStream);
            stream.init(file, -1, -1, 0);
            var bstream = Components.classes["@mozilla.org/binaryoutputstream;1"]
                .createInstance(Components.interfaces.nsIBinaryOutputStream);
            bstream.setOutputStream(stream);
            bstream.writeByteArray(result, length); //把图片流的全部字节写入输出文件流中
            if (stream instanceof Components.interfaces.nsISafeOutputStream) {
```

```

        stream.finish();
    }
    else {
        stream.close();
    }
    if (typeof callback == "function") {
        callback();
    }
}
};
var streamLoader = Components.classes["@mozilla.org/network/stream-loader;1"]
    .createInstance(Components.interfaces.nsIStreamLoader);
streamLoader.init(channel, observer, null);
}

```

## 什么是 XPCOM?

XPCOM 是一种跨平台的组件对象模型，类似于微软的 COM。它有多种的语言绑定，可以在 JavaScript, Java, Python 和 C++ 中使用和实现 XPCOM 的组件。XPCOM 本身提供了一系列核心组件和类，比如文件和内存管理，线程，基本的数据结构（字符串，数组）等。

这里的实现方式是读取从远程获取的图片数据流，并把相应的数据写入到本地磁盘指定的目录中。为了实现以异步的方式读取和保存数据，使用了 nsIStreamLoader 接口的实现。它从指定的通道读取数据，当数据读取完成之后，会通知相应的监听器。在这里，我们用图片的URL地址来初始化一个通道，同时创建了一个监听器。在 onStreamComplete 的方法中，把得到的图片字节流写入到本地文件存储中。最后，如果注册了回调函数，就执行此回调函数。

## 清单 9. 下载用户选择的全部图片的 JavaScript 方法

```

function download() {
    var rows = document.getElementById("imagesContainer");
    var checkboxes = rows.getElementsByTagName("checkbox");
    var imageUrls = [];
    for (var i = 0, n = checkboxes.length; i < n; i++) {
        if (checkboxes[i].checked) {
            imageUrls.push(checkboxes[i].getAttribute("imageUrl")); //用户选择的图片的 URL
        }
    }
    var progressmeter = byId("downloadProgress");
    progressmeter.style.visibility = "visible";
    var total = imageUrls.length, step = 100 / total, current = 0;
    for (var i = 0; i < total; i++) {
        downloadSingleImage(imageUrls[i], function() {
            var value = parseInt(progressmeter.value); //更新进度条
            progressmeter.value = value + step;
        });
    }
    close();
}

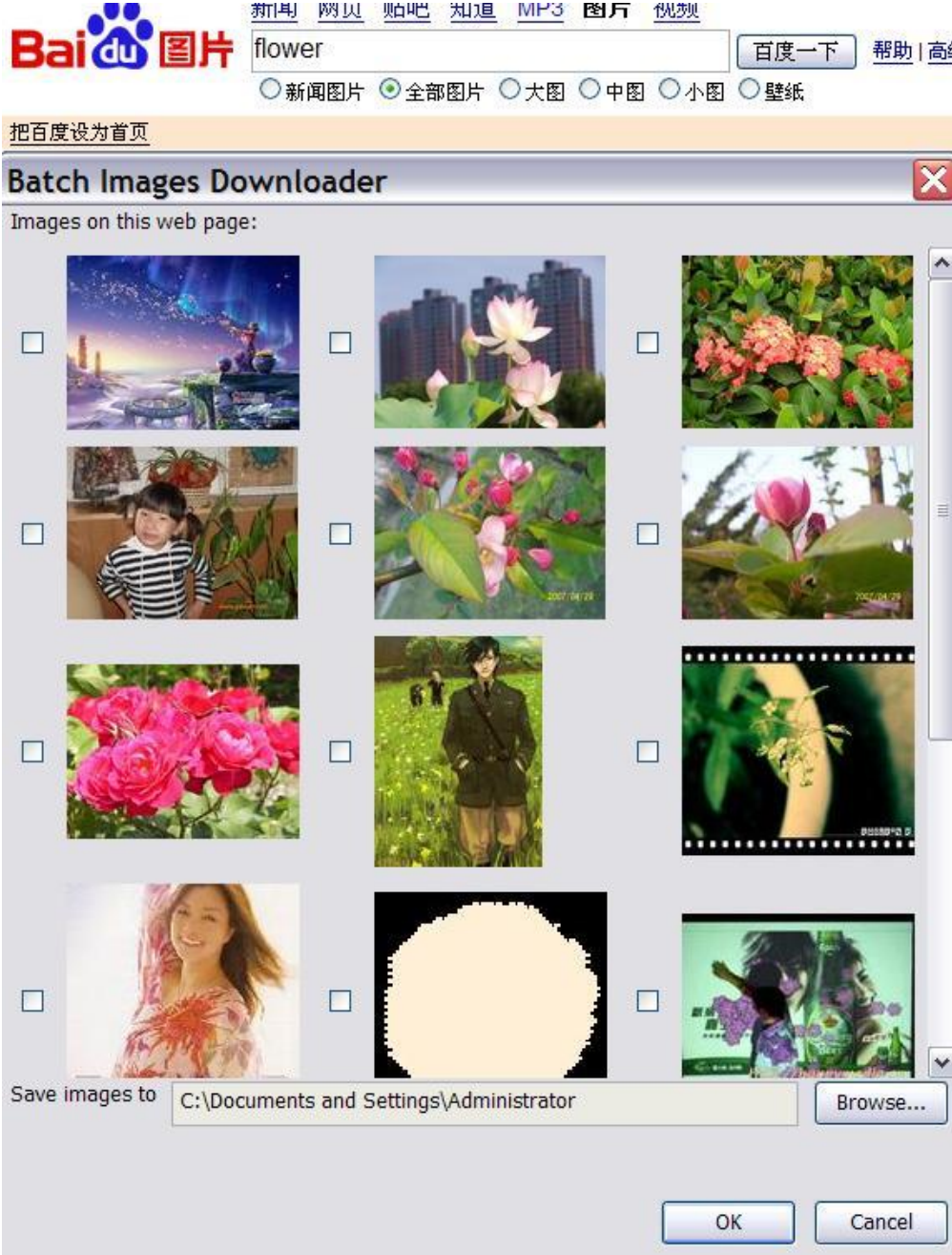
```

下载全部图片时，会逐个检查复选框的状态，把用户选择的图片的 URL 记录下来。对每张图片，都会调用 downloadSingleImage 以异步的方式来下载。在单张图片下载完成之后，会有回调函数来通知主窗口，更新进度条的状态。当所有图片都下载完成之后，关闭当前窗口。

至此，整个扩展就开发完成了。实际的扩展的截图如下：



图 2. 该 Firefox 扩展实际使用的截图



在 图 2 中，HTML 页面的内容是使用 “flower”作为关键字来访问百度图片搜索。

## 回页首

## 打包、发布和更新

### 打包

扩展打包的过程非常简单。只需要把整个目录的内容打包成一个 ZIP 格式的文件，并把文件的扩展名改为 xpi 即可。需要注意的是 install.rdf 要在 ZIP 文件的根目录下面，这样扩展才能安装到 Firefox 中。

### 发布



如果您想让别人也来使用您开发的扩展，一种方式是直接把打包好的 `xpi` 文件发给对方，他只需要用 **Firefox** 打开这个文件，就会自动提示安装。另外一种方式是该 `xpi` 文件存放在某个公开的 **HTTP** 服务器中，对方只需要用 **Firefox** 访问该 `xpi` 文件，同样会自动提示安装。这里需要注意的是要正确的设置 `xpi` 文件的 **MIME** 类型。**Firefox** 能识别的 `xpi` 文件的 **MIME** 类型是 `application/x-xpinstall`。您可能需要配置您的 **HTTP** 服务器。

## 更新

如果您的扩展的开发周期较长，需要发布多个版本的话，可以利用扩展的自动更新的能力。这样当有新的版本发布时，**Firefox** 会自动提示用户去获取最新的版本。要实现自动更新，需要在扩展的 `install.rdf` 中指定描述更新信息的 `rdf` 文件的位置，该 `rdf` 文件通常命名为 `update.rdf`。在 `update.rdf` 中声明了当前最新的版本号 and 最新版本的下载地址。如果用户安装的扩展的版本低于 `update.rdf` 中声明的版本，则 **Firefox** 会提示用户是否更新。请参考本文附带的 [源代码](#) 中的 `update.rdf` 文件。

## 回页首

## 声明

本文章仅代表作者本人观点，与 **IBM** 公司无关。

## 回页首

## 下载

描述	名字	大小	下载方法
批量下载图片的 <b>Firefox</b> 扩展的源代码	<code>batchimagesdownloader.zip</code>	4KB	<a href="#">HTTP</a>

## 关于下载方法的信息

## 参考资料

## 学习

- 在 [Mozilla 开发者中心的扩展开发专题](#) 中学习更多关于扩展开发的知识。
- 在 [Mozilla 开发者中心的 XPCOM 专题](#) 中学习更多关于 XPCOM 的知识。
- 在 [XUL Planet](#) 中学习更多关于 XUL 的知识。
- 阅读“[XUL — 快速开发跨平台易用用户接口的新途径](#)”：介绍 XML User-interface Language - 基于 XML 的用户接口语言。
- 阅读“[XML 用户界面语言 \(XUL\) 开发入门](#)”：在这个教程中，您将使用 XUL 进行编程。
- 阅读“[使用 XUL 实现浏览器扩展，第 1 部分: 使用用户界面特性创建一个 Firefox 浏览器扩展](#)”：介绍如何创建超越 Web 浏览器内置功能的扩展。
- 阅读“[使用 XUL 实现浏览器扩展，第 2 部分: 组建一个跨平台的 Firefox 扩展](#)”：了解如何构建功能强大的灵活的 Mozilla 浏览器扩展。

- | 阅读 “[创建动态的 Firefox 用户界面](#)”：学习如何使用 [Ajax](#) 从 Web 服务器下载 XML 数据，以及如何使用 XSLT 将 XML 数据动态地转换为用 XUL 表达的 Firefox 用户界面元素。
- | 通过 developerWorks [Web 开发专区](#) 中介绍 Web 技术的文章和教程扩展您的网站开发技巧。

## 获得产品和技术

- | 下载扩展开发者常用的扩展
  - | [Venkman](#)
  - | [Extension Developer's Extension](#)
  - | [Console<sup>2</sup>](#)
  - | [Chrome List](#)
  - | [Firebug](#)

## 讨论

- | [Mozilla 扩展开发的新闻组](#)
- | [Mozilla 扩展开发的 Google 讨论组](#)

# Firefox 扩展开发进阶指南

[成 富](#), 高级软件工程师, IBM

**简介:** Firefox 扩展可以从不同的方面增强 Firefox 浏览器的功能, 方便用户使用。本文在《[实战 Firefox 扩展开发](#)》一文的基础上, 重点介绍了 Firefox 扩展开发中的一些高级话题, 包括高级用户界面元素及其操作、XBL、XUL 数据模板、JavaScript 代码模块、XPCOM、国际化和扩展自动更新等。本文可以帮助开发人员更好的理解这些高级特性, 从而开发出功能更加强大的 Firefox 扩展。

本文的标签: [firefox](#), [中间件开发](#)

[标记本文!](#)

发布日期: 2011 年 8 月 25 日

级别: 中级

访问情况 804 次浏览

在上一篇与 Firefox 扩展开发相关的《[实战 Firefox 扩展开发](#)》中, 笔者介绍了 Firefox 扩展开发的基本内容。这篇文章作为上一篇的后续, 主要会讨论一些高级话题, 而且以 Firefox 4 作为扩展开发平台。本文中的示例代码都在 Firefox 4 下测试通过。在开发的时候, 建议使用 Firefox 便携版作为开发环境, 下载地址见 [参考资料](#)。Firefox 便携版中已安装扩展的文件存放在 Data/profile/extensions 目录中, 方便进行查看。下面首先介绍高级用户界面元素相关的内容。

## 高级用户界面元素及其操作

Firefox 扩展一般使用 XUL 来创建用户界面。相对于 HTML 来说, XUL 提供了更加丰富的用户界面组件, 同时也支持用 CSS 来定义 XUL 元素的外观样式。大部分简单 XUL 元素的使用比较好理解, 下面主要介绍一些复杂的用户界面组件及其用法。

### 窗口

一般的 Firefox 扩展都会打开一个新的窗口来展示界面。通常的做法是在用户点击菜单项或是工具栏上的按钮之后, 弹出来一个新的窗口, 与用户进行交互。通过 XUL 的 window 元素可以创建一个新的窗口, 并在窗口中包含其它的 XUL 用户界面元素作为内容。通过 window.open() 方法就可以打开一个新的窗口。该方法的返回值是窗口对象的一个引用, 可以用来查看窗口的属性和对其进行操作, 比如通过其 document 属性就可以获取到窗口文档的 DOM 对象。

在扩展中经常会需要对窗口进行查询。这里可以用 nsIWindowMediator 服务。该服务可以用来根据窗口类型进行查询。nsIWindowMediator 服务是一个 [XPCOM](#) 组件。关于 XPCOM 组件的细节, 会在下面的章节中介绍。[代码清单 1](#)中给出了 nsIWindowMediator 的使用示例。

#### 清单 1. 使用 nsIWindowMediator 查询窗口

```
var wm = Components.classes["@mozilla.org/appshell/window-mediator;1"]
    .getService(Components.interfaces.nsIWindowMediator);
var enumerator = wm.getEnumerator("navigator:browser");
while(enumerator.hasMoreElements()) {
    var win = enumerator.getNext(); // 迭代器中的每个元素都是一个窗口对象
    alert(win.document.title);
}
```

如 [代码清单 1](#) 所示, navigator:browser 是 Firefox 中浏览网页的窗口的类型。nsIWindowMediator 服务的 getEnumerator() 方法可以得到一个用来遍历所有指定类型窗口的迭代器。nsIWindowMediator 的另一个方法 getMostRecentWindow() 可以用来获取到最近打开的指定类型的窗口。该方法通常用来检查某个窗口是否已经被打开

了。一般的做法是在窗口的 `windowtype` 属性上设置一个自定义的窗口类型，然后就可以用 `getMostRecentWindow()` 方法来查询此类型的窗口是否已经被打开。[代码清单 2](#) 给出了一个在扩展中常用的方法。该方法首先判断某个窗口是否已经被打开，如果没有，就打开此窗口，否则就把焦点移到该窗口上。

### 清单 2. 打开窗口或把焦点移到已有窗口上

```
var wm = Components.classes["@mozilla.org/appshell/window-mediator;1"]
    .getService(Components.interfaces.nsIWindowMediator);
var win = wm.getMostRecentWindow("dwSample-custom-window");
if (win) {
    win.focus(); // 设置窗口为当前焦点
}
else {
    window.open("chrome://dwSample/content/custom-window.xul",
        "dwSample-test-custom-window", "chrome,centerscreen");
}
```

如 [代码清单 2](#) 所示，这里打开的窗口 `custom-window.xul` 中通过 `windowtype` 属性设置了自定义的窗口类型 `dwSample-custom-window`。该类型是 `getMostRecentWindow()` 方法的参数。

在使用窗口时的另一个常见需求是向新打开的窗口传递数据，比如新窗口初始化时所需的数据。当一个新窗口被打开的时候，在新窗口中可以通过 `window.opener` 来访问打开它的窗口的 `window` 对象。如果新窗口总是以固定的方式被打开，也就是说 `window.opener` 的值是确定的话，可以通过 `window.opener` 来进行数据传递。如果 `window.opener` 属性的值不确定的话，可以使用 `nsIWindowWatcher` 服务在打开新窗口的时候传递数据，具体用法见 [代码清单 3](#)。

### 清单 3. 使用 nsIWindowWatcher 传递数据

```
var args = { name : "Alex" };
args.wrappedJSObject = args;
var watcher = Components.classes["@mozilla.org/embedcomp/window-watcher;1"]
    .getService(Components.interfaces.nsIWindowWatcher);
watcher.openWindow(null, "chrome://dwSample/content/window.xul",
    "dwSample-test-window", "chrome,centerscreen", args);
```

如 [代码清单 3](#) 所示，`args` 就是传递给新窗口的参数。在新窗口中，可以通过 `window.arguments` 来获取到参数的值。`window.arguments[0].wrappedJSObject` 的值就是传递过来的 `args` 对象。

## 对话框

Firefox 扩展中可以使用两种类型的对话框：一种是 Firefox 提供的标准对话框，包括消息提示、确认和输入对话框等；另外一种则是由扩展本身使用 `dialog` 元素创建的自定义对话框。

在 HTML 页面中，可以使用 `window.alert()`、`window.confirm()` 和 `window.prompt()` 等方法来弹出浏览器标准的提示、确认和输入对话框。这些方法在扩展的 JavaScript 代码中也是可以使用的。但是在扩展中推荐的方式是使用 `nsIPromptService` 服务。使用 `nsIPromptService` 中的方法创建对话框的一个重要优势是可以设置对话框的标题，另外还可以在对话框中添加一个额外的复选框。这个额外的复选框可以用来实现“下次不要再提示我”这样的功能。[代码清单 4](#) 中给出了 `nsIPromptService` 服务提供的对话框的使用方式。

### 清单 4. nsIPromptService 服务提供的对话框的使用

```
var ps = Components.classes["@mozilla.org/embedcomp/prompt-service;1"]
    .getService(Components.interfaces.nsIPromptService);
ps.alert(null, "Simple Alert", "Hello World!");
var checkValue = {value : true};
ps.alertCheck(null, "Alert with checkbox", "Hello!", "Don't show again.", checkValue);
var result = ps.confirmCheck(null,
    "Confirm", "Confirm", "Don't show again.", checkValue);
var content = {value : "Alex"};
result = ps.prompt(null, "Input something", "Your name :", content, null, {});
if (result) {
    ps.alert(null, "Input value", content.value);
}
```

图 1. 简单提示对话框



图 2. 带复选框的提示对话框



图 3. 带复选框的确认对话框

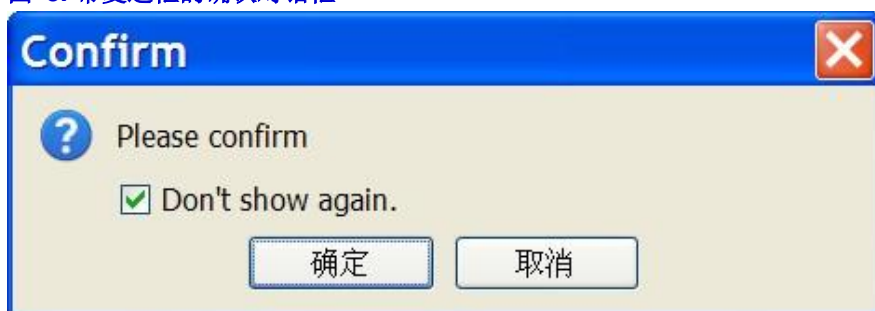


图 4. 输入对话框



图 1、图 2、图 3 和 图 4 分别展示了 [代码清单 4](#) 中定义的四种不同对话框的运行效果。如 [代码清单 4](#) 中所示，`alert()` 和 `alertCheck()` 方法用来打开消息提示对话框，不过后者可以在对话框上添加一个复选框。参数 `checkValue` 用来设置复选框初始时是否选中。类似的，`confirm()` 和 `confirmCheck()` 都是用来打开确认对话框的。从 `prompt()` 方法的用法中可以得出打开对话框的代码和对话框之间的数据传递方式。创建一个包含属性 `value` 的 JavaScript 对象，并作为参数传递给 `prompt()` 方法。这个 JavaScript 对象的属性 `value` 的初始值作为对话框中显示的初始值。当对话框关闭之后，从属性 `value` 就可以获取到用户所输入的值。也就是说这个 JavaScript 对象既做输入，又做输出。

对于由 `dialog` 元素创建的对话框，在外观和作用上类似于由 `window` 元素创建的窗口，不同之处在于 `dialog` 提供了标准的按钮和对应的事件处理能力。这些标准按钮包括确定（`accept`）、取消（`cancel`）、帮助（`help`）、更多信息（`disclosure`）和两个额外的自定义按钮（`extra1` 和 `extra2`）。对每个按钮都可以分别设置其标签、事件处理方法和快捷访问键。例如对确认按钮来说，可以通过 `buttonlabelaccept`、`ondialogaccept` 和 `buttonaccesskeyaccept` 来分别设置其标签、事件处理方法和快捷访问键。通过 `dialog` 元素的 `buttons` 属性可以设置对话框上要包含的按钮名称，如 `buttons="accept, cancel, help"` 就声明了对话框中包含确定、取消和帮助等 3 个按钮。[图 5](#) 中给出了一个自定义对话框的运行效果图。

图 5. 自定义对话框



通过 `window.openDialog()` 可以打开新的对话框，与 `window.open()` 不同的是，在调用 `window.openDialog()` 的时候可以传递额外的参数给对话框，这比打开新窗口时的参数传递要简单得多，如 [代码清单 5](#) 所示。

#### 清单 5. `window.openDialog()` 打开对话框时的数据传递

```
var args = { name : "Alex" };
window.openDialog("chrome://dwSample/content/dialog.xul", "dwSample-test-dialog",
    "chrome,modal,centerscreen", args);
```

在对话框的代码中，可以通过 `window.arguments[0]` 来获取到 `args` 对象。

#### 侧栏

有些扩展选择使用侧栏来作为其主界面，而不是用一般的新窗口。侧栏的界面声明应该被放在一个单独的 XUL 文件中，并且该 XUL 文件的根元素是 `page`。一般来说，侧栏可以通过 Firefox 主菜单的“查看”->“侧栏”来打开和关闭。[图 6](#) 给出了一个侧栏的运行效果图。新的侧栏也需要在覆层（`overlay`）中进行注册。注册的方式如 [代码清单 6](#) 所示。



图 6. 侧栏



清单 6. 在覆层中注册侧栏

```
<menupopup id="viewSidebarMenu">
  <menuitem observes="viewDwSampleSidebar" />
</menupopup>

<broadcaster>
  <broadcaster id="viewDwSampleSidebar"
    label="dwSample 侧栏"
    autoCheck="false"
    type="checkbox"
    group="sidebar"
    sidebarurl="chrome://dwSample/content/sidebar.xul"
    sidebartitle="dwSample 侧栏"
    oncommand="toggleSidebar('viewDwSampleSidebar');" />
</broadcaster>
```

如 [代码清单 6](#) 所示, ID 为 viewSidebarMenu 的菜单上就是 Firefox 主菜单上的“查看”->“侧栏”项。在这个菜单项上多加一个子菜单用来打开和关闭侧栏。通过 toggleSidebar() 方法就可以切换某个侧栏的打开和关闭状态。

如果在扩展的其它地方需要访问侧栏的话, 可以用 [代码清单 7](#) 中的方法。

清单 7. 与侧栏进行交互

```
var sidebarWindow = document.getElementById("sidebar").contentWindow;
if (sidebarWindow.location.href ==
    "chrome://dwsample/content/sidebar.xul") {
  sidebarWindow.sayHi();
}
```

## 表格

在用户界面中常见的表格在 Firefox 扩展中是通过包括 listbox、listhead、listheader、listcols、listcol、listitem 和 listcell 等 XUL 元素来创建的。从与 HTML 表格的相关元素对应的角度来说, listbox 即表格本身, 相当于 table; listhead 是表头, 相当于 thead; listheader 是表头中每一列的标题, 相当于 th; listcol 是表格中的一列; listcols 是 listcol 的集合; listitem 是表格中的一行, 相当于 tr; listcell 是表格中的单元格, 相当于 td。以这种方式进行类比, 就很容易了解在扩展中创建表格的方式。

listbox 的单元格中只能包含文本和图片。如果希望表格的单元格中能包含其它内容, 如 XUL 组件, 就需要使用 richlistbox 和 richlistitem。 [代码清单 8](#) 给出了在 richlistitem 中添加一个按钮的示例。

#### 清单 8. 在 richlistitem 中添加一个按钮

```
<richlistbox>
  <richlistitem><button label="Button"></button></richlistitem>
</richlistbox>
```

#### 布局

在创建扩展的用户界面的时候, 布局是其中一个很重要的方面。XUL 中的布局的基本元素是作为容器来使用的盒子 (box), 分成水平盒子和垂直盒子, 分别由 hbox 和 vbox 元素来创建。hbox 和 vbox 分别在水平和垂直方向上依次排列其内部的元素。比如希望 3 个按钮 (button) 水平排列的话, 只需要用一个 hbox 来包含它们就可以了。hbox 和 vbox 内的 XUL 元素的一个重要属性 flex 用来声明盒子中剩余空间的分配方式。如果包含在盒子内部的组件不能占满 hbox 和 vbox 盒子的全部空间, 就会留出来相应的空白。通过设置内部组件的 flex 属性就可以分配这些空白区域。flex 属性的值是整数, 定义的是空白区域在各个组件之间分配的相对比例。例如, 3 个组件分别声明了 flex 属性的值是 4、2 和 2 的话, 那么 3 个组件分别占据 50%、25% 和 25% 的剩余空白区域。hbox 和 vbox 的属性 align 和 pack 用来声明盒子中间组件的对齐方式。对于 hbox 来说, align 和 pack 分别用来声明在垂直和水平方向上的对齐方式, 而对 vbox 来说则正好相反。

如果需要对多个 XUL 组件进行复杂的布局的话, 可以使用 grid 元素。grid 可以用来实现复杂的表格式布局。在 grid 中可以包含多行和多列。行和列分别用 row 和 column 表示, 而 rows 和 columns 则分别是行和列的集合。row 和 column 的每个子元素都分别占据其所对应的单元格。通常是在其中嵌套使用 hbox 和 vbox 来实现复杂的布局。

#### 使用 HTML 创建界面

虽然 XUL 在使用方式上非常类似 HTML, 但是有一部分扩展开发者还是对 HTML 语言比较熟悉, 对 HTML 本身的布局方式和使用 CSS 来设计样式也比较了解。在 Firefox 扩展中, 同样也是可以使用 HTML 页面来作为用户界面的。XUL 中的 browser 和 iframe 元素可以用来显示一个 XUL 或 HTML 页面。这两个元素都有一个属性叫 type 用来声明所包含页面的内容类型。这个属性的值有安全性方面的作用, 进而会影响包含 browser 或 iframe 的窗口与其中的页面的数据传递方式。属性 type 默认值是 chrome, 指的是 HTML 页面是属于扩展的一部分。对于使用 HTML 来构建界面的扩展来说, 这个值是很合适的。在这种情况下, HTML 页面是可以通过 parent 属性来访问其父窗口的 window 对象的, 如 [代码清单 9](#) 所示。如果属性 type 的值是其它值的话, 从安全的角度考虑, 这种访问方式是不允许的。

#### 清单 9. HTML 页面与其父对话框的数据传递

```
// 包含 HTML 页面的 XUL 对话框
<script>
  window.hobby = "Game";
</script>
<hbox flex="1" width="300" height="300">
  <browser id="content" type="chrome"
    src="chrome://dwSample/content/hobby.html" flex="1" />
</hbox>

// 在 HTML 页面 hobby.html 中
<script type="text/javascript">
  function onLoad() {
    var node = document.getElementById("hobby");
```

```
node.value = window.parent.hobby;
}
```

```
</script>
```

在 [代码清单 9](#) 中, 父对话框的 `window` 对象中定义了一个 `hobby` 变量, 在子 HTML 页面中, 通过 `window.parent.hobby` 来获取这个变量。图 7 给出了使用 HTML 创建界面的对话框的运行效果图, 图中的对话框内容界面是由 `chrome://dwSample/content/hobby.html` 页面来创建的。

图 7. 使用 HTML 创建界面的对话框



在介绍完高级用户界面元素及其操作之后, 下面介绍如何通过 XBL 实现用户界面的组件化。

---

[回页首](#)

## 使用 XBL 实现组件化

在开发 Firefox 扩展的用户界面的时候, 会遇到的一个现实问题就是用户界面的组件化。大多数时候, 开发人员都是使用 XUL 的基本组件来构建用户界面。在有些情况下, 扩展中的某些部分的用户界面可能需要在不同的地方被重复使用。这种代码重复显然是在开发中是要避免的。最直接的解决办法就是创建自己的用户界面组件。这样就只需要在一个地方进行定义和修改, 在其它地方只是引用即可。在 XUL 中, 这种自定义的组件是通过 XBL (XML Binding Language) 来实现的。简单来说, 通过 XBL 可以创建出新的 XUL 元素。这些 XUL 元素可以像基本的 XUL 元素一样在扩展中使用。自定义的 XUL 元素在内部封装了组件的展现和相关的逻辑, 可以有自己的属性、方法和事件等。实际上, 比较复杂的基本 XUL 元素也是通过 XBL 创建出来的。

在扩展中使用 XBL 由两部分组成, 一个部分是自定义 XUL 元素的声明, 另一个部分则是具体的使用这个 XUL 元素。XBL 组件的声明包含在一个 XML 文件中, 而应用部分则通过一个 CSS 文件来指定。本文用一个简单的 XUL 元素作为示例来进行说明。该 XUL 元素是一个地址输入组件, 允许用户选择所在省份和城市, 以及输入详细的地址。完整的示例代码见 [参考资料](#)。代码清单 10 中给出了该 XUL 元素的 XML 文件的部分内容和使用它的 CSS 文件的内容。

**清单 10. XUL 元素声明的 XML 文件与使用它的 CSS 文件**

```
//XML 文件内容
<?xml version="1.0"?>
<bindings xmlns="http://www.mozilla.org/xbl"
  xmlns:xbl="http://www.mozilla.org/xbl"
  xmlns:xul="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <binding id="addressinput">
  </binding>
</bindings>

//CSS 文件内容
dwaddressinput {
  -moz-binding: url("chrome://dwSample/content/addressinput.xml#addressinput");
}
```

如 [代码清单 10](#) 所示, XML 文档中根元素 `bindings` 下面的每个 `binding` 元素表示一个新的 XUL 元素。`binding` 的属性 `id` 是该 XUL 元素的标识符。在 CSS 文件中声明了一条规则, 对 `dwaddressinput` 元素添加了 `-moz-binding` 声明。这条规则的含义是把元素 `dwaddressinput` 关联到 `chrome://dwSample/content/addressinput.xml` 这个 XML 文件所声明的绑定上。由于一个 XML 文件中可以声明多个新的 XUL 元素, URL 后面的 `#dwaddressinput` 用来指定所关联的 XUL 元素的 ID, 即 `binding` 元素的属性 `id` 的值。这样在引用了该 CSS 文件的 XUL 文件中, 就可以创建出 `dwaddressinput` 元素。

**内容**

下面具体说明声明 XUL 元素的 XML 文件中包含的内容。在 `binding` 元素下的 `content` 元素用来定义该 XUL 元素的用户界面内容, 可以包含其它的基本的 XUL 元素。需要注意的是, 由于 XML 名称空间的要求, 在声明基本的 XUL 元素的时候, 需要使用相应的名称空间前缀, 如 `xul:hbox`。[代码清单 11](#) 中给出了示例 XUL 元素的 `content` 元素的内容。

**清单 11. 示例 XUL 元素的 content 元素的内容**

```
<content>
  <xul:hbox>
    <xul:menulist label="省份" anonid="province-select"
      oncommand="document.getBindingParent(this).provinceSelectChanged(event);">
    </xul:menulist>
    <xul:menulist label="城市" anonid="city-select">
    </xul:menulist>
    <xul:textbox anonid="location-input" />
  </xul:hbox>
</content>
```

如 [代码清单 11](#) 所示, 有几个地方值得说明一下。首先是属性 `anonid` 用来声明元素的 ID。在这里不能直接使用 `id`。这是因为一个 XUL 元素可能在一个页面上被多次使用。如果使用 `id` 的话, 会造成页面上多个元素具有相同的 ID。属性 `anonid` 是一种替代 `id` 的标识符。在 XUL 元素的 JavaScript 代码中可以通过 `document.getAnonymousElementByAttribute()` 方法来根据 `anonid` 的值查询元素。代码中通过属性 `oncommand` 为 `menulist` 添加了事件处理代码。其中的 `document.getBindingParent(this)` 用来获取包含它的 XUL 元素, 再调用其中的方法。 `provinceSelectChanged` 是声明的 XUL 元素中添加的自定义方法。

**方法**

为新的 XUL 元素添加一个方法是通过 method 元素来实现的。[代码清单 12](#) 中给出了 provinceSelectChanged() 方法的声明。

### 清单 12. 示例 XUL 元素中的方法声明

```
<method name="provinceSelectChanged">
  <parameter name="aEvent" />
  <body>
    <![CDATA[
      var provinceSelect = document.
        getAnonymousElementByAttribute(this, "anonid", "province-select");
      var province = provinceSelect.selectedItem.label;
      this.selectProvince(province);
    ]]>
  </body>
</method>
```

如 [代码清单 12](#) 所示, method 元素的属性 name 表示的是方法的名称; 子元素 parameter 表示的是该方法的参数, 可以用多个 parameter 元素来表示多个参数; 子元素 body 中包含的是方法体的 JavaScript 代码。在该方法体被执行的时候, 关键词 this 所指向的对象是 XUL 元素本身。在编写 JavaScript 方法体的时候, 需要注意这一点。通过 method 元素来声明的方法可以被外部的 JavaScript 代码所使用。

### 属性

除了方法之外, 还可以声明 XUL 元素的自定义属性。属性声明是通过 property 元素来实现的。[代码清单 13](#) 中给出了属性 province 的声明。

### 清单 13. 示例 XUL 元素中的属性声明

```
<property name="province">
  <getter>
    <![CDATA[
      var provinceSelect = document.
        getAnonymousElementByAttribute(this, "anonid", "province-select");
      return provinceSelect.selectedItem.label;
    ]]>
  </getter>
  <setter>
    <![CDATA[
      var province = val ? val : DEFAULT_PROVINCE;
      this.selectItemByLabel("province-select", province);
      this.selectProvince(province);
      return val;
    ]]>
  </setter>
</property>
```

如 [代码清单 13](#) 所示, property 元素的 name 属性声明了属性的名称。声明属性中最重要的是定义获取和设置属性时的逻辑。这分别是通过 getter 和 setter 子元素来实现的。这两个子元素的文本内容就是获取或设置时要执行的 JavaScript 代码。如果要执行的 JavaScript 代码比较简短的话, 可以省去这两个子元素, 而用 property 元素的属性 onget 和 onset 来替代。在设置属性的 JavaScript 代码中, 可以直接使用变量 val。该变量的值是调用者提供的该属

性的新值。另外设置属性的方法需要把 `val` 作为其返回值，这样就可以实现级联赋值。

与 `property` 元素具备类似功能的是 `field` 元素，它也可以用来定义 XUL 元素中的属性，不过只提供了基本的属性值存储功能，并不能自定义读取和设置时的逻辑。

## 具体使用

下面介绍如何具体的使用新创建出来的 XUL 元素。首先需要在 XUL 文件中引用声明了绑定关系的 [代码清单 10](#) 中给出的 CSS 文件，然后就可以像基本 XUL 元素一样直接声明，或是通过 JavaScript 代码来动态创建。[代码清单 14](#) 中给出了动态创建新声明的 XUL 元素 `dwaddressinput` 的方式。

### 清单 14. 动态创建 `dwaddressinput` 元素

```
//JavaScript 代码
function onLoad() {
    var container = document.getElementById("container");
    var addressInput = createAddressInput(container,
        'dwSample-address-input', "湖南省", "湘潭市", "韶山市");
    window.sizeToContent();
}

function createAddressInput(parentNode, id, province, city, location) {
    var addressInput = document.createElement("dwaddressinput");
    addressInput.setAttribute("id", id);
    parentNode.appendChild(addressInput);
    addressInput.province = province;
    addressInput.city = city;
    addressInput.location = location;
    return addressInput;
}

function showAddress() {
    var addressInput = document.getElementById("dwSample-address-input");
    alert(addressInput.getAddress());
}

//XUL 代码
<vbox flex="1" id="container"></vbox>
```

如 [代码清单 14](#) 所示，在 XUL 中创建了一个 `vbox` 元素作为新创建出来的 XUL 元素的容器。`onLoad()` 方法在加载完成之后被调用。JavaScript 方法 `createAddressInput()` 用来创建 `dwaddressinput` 元素，并通过其属性来设置初始值。新创建的 `dwaddressinput` 元素的 ID 是 `dwSample-address-input`。方法 `showAddress()` 首先通过 ID 来获取到 `dwaddressinput` 元素的引用，再调用其中的方法 `getAddress()`。这个 `getAddress()` 方法也是通过 `method` 元素来声明的。[图 8](#) 中给出了实际的运行效果图。

图 8. XBL 使用示例





在介绍完使用 XBL 创建自定义 XUL 元素之后，下面说明如何使用 XUL 数据模板。

[回页首](#)

## XUL 数据模板

在 Firefox 扩展开发中总是免不了与各种不同类型的数据打交道，不管是扩展本身的内部数据，还是来自远程的数据。典型的处理数据的做法是获取数据之后先进行解析，提取出其中感兴趣的内容，再创建相应的用户界面元素来显示。整个过程繁琐而且容易出错。事实上，如果扩展所使用的数据类型是 RDF、XML 或 [SQLite 数据库](#) 的话，使用 XUL 数据模板技术是更好的选择。

XUL 数据模板是一个强大的系统，在其中封装了数据的获取、查询和用户界面生成等操作。使用该技术可以用简洁的代码实现复杂的功能。下面使用一个简易的 RSS 订阅源阅读器扩展作为示例来进行说明。典型的 RSS 订阅源阅读器需要涉及到数据的获取、XML 格式的解析和查询、以及用 JavaScript 代码动态创建用户界面等操作。[代码清单 15](#) 给出了使用 XUL 数据模板实现的相关代码。

### 清单 15. XUL 数据模板示例

```
<vbox height="600" style="overflow:auto;"
  datasources="http://news.163.com/special/00011K6L/rss_newstop.xml"
  ref="*" querytype="xml" flex="1">
  <template>
    <query expr="channel/item">
      <assign var="?title" expr="./title/text()"/>
      <assign var="?command"
        expr="concat('viewContent(&quot;', ./link/text() , '&quot;);')" />
    </query>
    <action>
      <hbox uri="?">
        <description flex="1" value="?title" />
        <button label="查看" oncommand="?command" />
      </hbox>
    </action>
  </template>
</vbox>
```

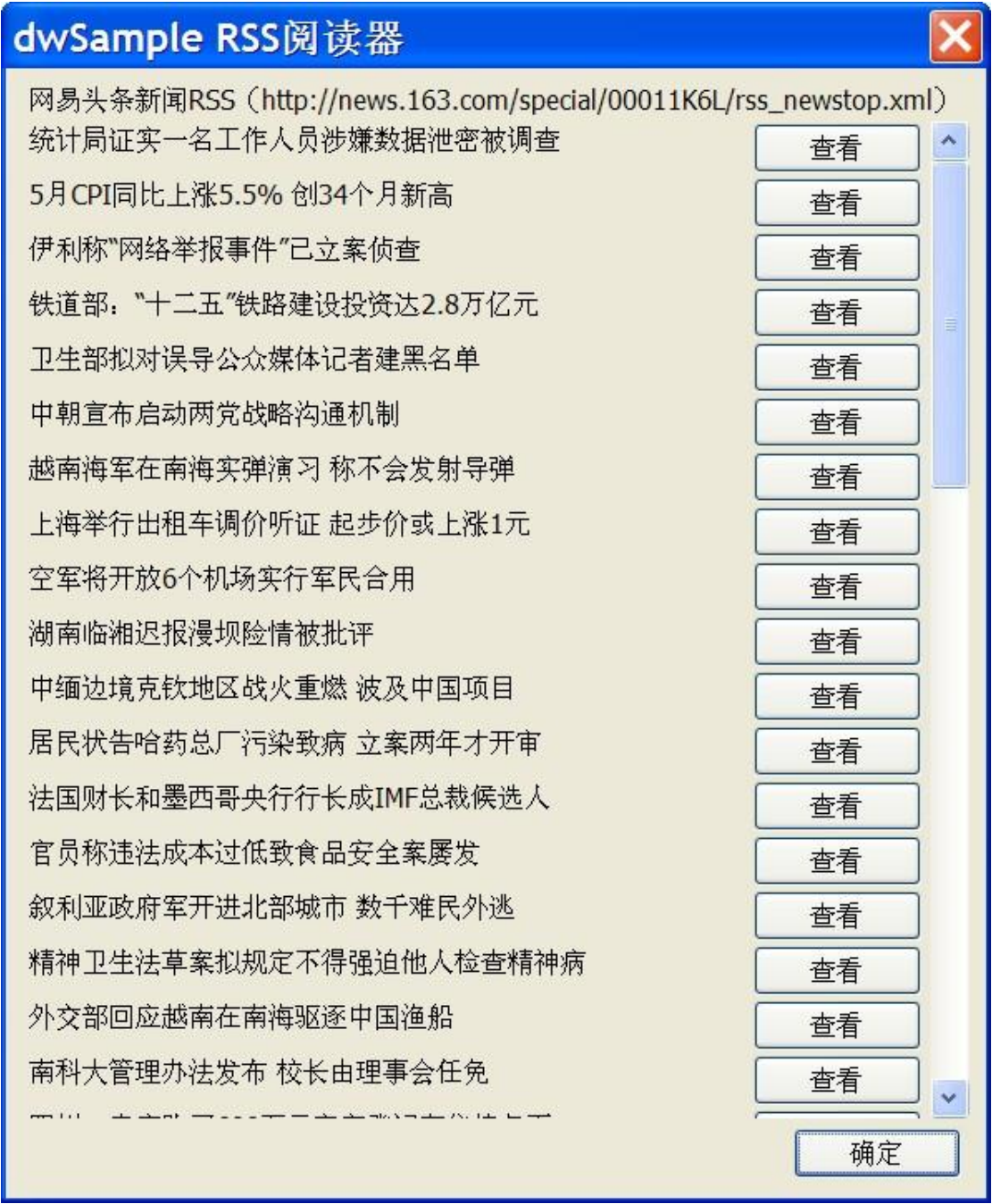
如 [代码清单 15](#) 所示，使用 XUL 数据模板的实现非常简洁易懂，并没有复杂的动态创建用户界面的 JavaScript 代码。下面进行具体的分析和说明。首先是声明要使用的是数据源。vbox 元素的 datasources 属性来声明数据源的 URL。这个属性可以添加在任何 XUL 元素上。对于 RDF 类型的数据来说，可以指定以空格分隔的多个数据源 URL。XUL 数据模板会自动聚合多个 RDF 数据源。属性 querytype 用来指定数据类型，可以是 rdf、xml 和 storage，分别表

示 RDF、XML 和 SQLite 数据类型。属性 `ref` 用来表示进行数据查询时候的起始位置。对于 XML 数据来说，数据查询总是从 XML 文档根节点开始的，因此这个属性并没有使用，一般设成 `*` 即可。

`vbox` 元素的子元素 `template` 中包含的就是 XUL 数据模板的定义。模板定义通常由两个部分组成，分别是查询 `query` 和动作 `action`。查询是用来从数据源中选择感兴趣的内容。对于 XML 数据来说，查询是通过 XPath 来完成的。`query` 元素的 `expr` 属性是查询的 XPath 表达式，这里是选择了 RSS 订阅源中所有的 `item` 元素。`query` 元素的子元素 `assign` 用来定义可以在模板中使用的变量。这主要是为了方便在模板中使用数据。在示例应用中，我们感兴趣的是每个条目的标题和链接。对于标题来说，通过 XPath 表达式 `./title/text()` 获取到了 `title` 元素的文本内容，并赋值给变量 `?title`。变量名称前面的问号 `?` 用来表明这是一个变量。而对于链接的处理，这里创建了一个变量 `?command` 用来表示一个 JavaScript 方法调用，并把条目的链接封装在方法调用中。

`action` 元素里面包含的就是用户界面的模板。这里面可以使用任何基本的 XUL 元素。需要注意的是其中声明了属性 `uri="?"` 的 XUL 元素。对于查询结果中的每条记录，该元素及其子元素在最后的用户界面中都会被重复一次。就示例应用来说，在生成的用户界面中，对于 RSS 订阅源中的每个 `item`，都会有一个 `hbox` 及其子 `description` 和 `button` 元素与之对应。`description` 和 `button` 元素都直接引用了查询中通过 `assign` 定义的变量。[图 9](#) 给出了示例的 RSS 阅读器的运行效果图。

图 9. RSS 阅读器



在介绍完 XUL 数据模板之后，下面介绍如何在扩展中添加偏好设置。

[回页首](#)

偏好设置

有些 **Firefox** 扩展提供了一些配置选项，允许用户进行自定义。这些偏好设置在 **Firefox** 重启之后仍然是生效的。通过 **Firefox** 的“工具”菜单下的“选项”，就可以打开 **Firefox** 自己的偏好设置对话框。另外通过在浏览器地址栏输入 `about:config` 也可以进行修改。

偏好设置中的每个配置项都是一个名值对。名称是由扩展开发者自己指定的，一般是以固定的名称空间作为前缀以避免冲突。而配置项的值可以是字符串、整数和布尔值等简单类型，还可以是 **Unicode** 字符串、本地化字符串和文件路径等复杂类型。定义好扩展中所需的配置项的名称之后，就可以声明它们了。在扩展根目录下的 `defaults/preferences` 目

录下创建一个 JavaScript 文件，用来声明配置项及其默认值。其内容如 [代码清单 16](#) 所示。

#### 清单 16. 声明偏好设置配置项及其默认值

```
pref("extensions.dwSample.locale", "zh_CN");
```

如 [代码清单 16](#) 所示，pref 是一个 JavaScript 方法，用来设置配置项的默认值。示例扩展中的所有配置项都使用 extensions.dwSample 作为前缀。虽然通过这种方式声明配置项及其默认值不是必需的，但大多数情况下，这是一种好的实践。

定义好配置项之后，需要提供相应的界面让用户可以修改这些配置。虽然通过 about:config 可以直接修改，但对普通用户来说，这种用户体验并不友好。一般只适用于高级配置选项。事实上，通过 XUL 元素来创建扩展自己的配置修改界面是很容易的。[代码清单 17](#) 中给出了示例界面。

#### 清单 17. 配置修改界面

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<?xml-stylesheet
  href="chrome://browser/skin/preferences/preferences.css" type="text/css" ?>
<prefwindow xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <prefpane id="basic-options" label="基本设置">
    <preferences>
      <preference id="pref-locale"
        name="extensions.dwSample.locale" type="string"/>
    </preferences>
    <textbox label="语言: " preference="pref-locale"/>
  </prefpane>
</prefwindow>
```

如 [代码清单 17](#) 所示，配置修改界面的根元素是 prefwindow，而不是一般的 window 或 dialog。在一个 prefwindow 中可以包含多个 prefpane。每个 prefpane 会被显示成一个标签页。preferences 元素下的每个 preference 元素用来声明一个配置项，其 name 和 type 属性分别是配置项的名称和类型。在声明了要修改的配置项之后，还需要提供相应的用户界面组件来让用户进行修改，如这里的 textbox 元素。通过属性 preference 可以把一个组件与配置项绑定起来，其值应该是之前声明的 preference 元素的 id。绑定之后，用户所做的修改会被自动同步到配置项中。

配置修改界面创建完成之后，在扩展代码中可以像其它对话框一样，通过 window.openDialog() 来打开它。一般还需要在扩展的 install.rdf 中通过

<em:optionsURL>chrome://dwSample/content/settings.xul</em:optionsURL> 来声明。这样的话，用户就可以通过 Firefox 的扩展管理界面来修改设置。

接下来是如何在扩展代码中使用配置项，见 [代码清单 18](#)。

#### 清单 18. 在 JavaScript 代码中使用配置项

```
var prefs = Components.classes["@mozilla.org/preferences-service;1"]
  .getService(Components.interfaces.nsIPrefService);
prefs = prefs.getBranch("extensions.dwSample.");
var locale = prefs.getCharPref("locale");
prefs.setCharPref("locale", "zh_CN");
```

如 [代码清单 18](#) 所示, 首先获取到配置服务 `nsIPrefService`, 接着通过 `getBranch()` 方法获取到扩展本身的相关分支。最后根据配置项的数据类型, 调用相应的获取和设置的方法即可, 包括 `getCharPref()/setCharPref()`、`getIntPref()/setIntPref()` 和 `getBoolPref()/setBoolPref()`。前面提到过, 扩展的配置项的名称一般都带有固定的名称空间前缀, 使用类似 `x.y.z` 的格式。这使得配置项的名称实际上形成了一种树形结构。通过 `getBranch()` 方法获取到树上的某个分支之后, 再引用配置项名称的时候就可以不用带前缀了。

另外一个常见的需求是在配置项被用户修改之后得到通知, 并执行相应的处理逻辑。这是通过添加配置项监听器来实现的, 见 [代码清单 19](#)。

#### 清单 19. 添加配置项监听器

```
var observer = {
  observe : function(subject, topic, data) {
    if(topic != "nsPref:changed") return;
    switch (data) {
      case "locale":
        sayHello();
        break;
    }
  }
};

function addPrefListener() {
  var prefs = Components.classes["@mozilla.org/preferences-service;1"]
    .getService(Components.interfaces.nsIPrefService);
  prefs = prefs.getBranch("extensions.dwSample.");
  prefs.QueryInterface(Components.interfaces.nsIPrefBranch2);
  prefs.addObserver("", observer, false);
}
```

如 [代码清单 19](#) 所示, 通过 `addObserver()` 方法可以添加一个配置项的监听器。该方法的第一个参数表示的是感兴趣的配置项名称, 可以是一个分支或是具体的配置项。第二个参数则是监听器对象本身。监听器对象需要包含一个 `observe()` 方法。当感兴趣的配置项发生变化的时候, `observe()` 方法会被调用。调用的时候 `observe()` 方法会接受 3 个参数: 第一个参数 `subject` 表示的是配置服务对象, 即上面的变量 `prefs`; 第二个参数 `topic` 表示的是变化的类型; 第三个参数 `data` 表示的是发生变化的配置项的名称。

在介绍完偏好设置相关的内容之后, 下面介绍 **JavaScript** 代码模块。

---

[回页首](#)

#### JavaScript 代码模块

**JavaScript** 代码模块 (**JavaScript code module**) 是 **Firefox 3** 中引入的新概念, 用来在不同的作用域范围内共享 **JavaScript** 代码, 以及创建全局的 **JavaScript** 单例对象。除了在扩展中开发自己的代码模块之外, 还可以使用 **Firefox** 提供的已有模块来简化扩展的开发。

一个 **JavaScript** 代码模块就是一个普通的 **JavaScript** 文件, 惟一的不同是在文件中通过 `EXPORTED_SYMBOLS` 来声明



该模块所暴露出来的对象。当该 JavaScript 模块被引入到其它作用域的时候，这些暴露出来的对象会被混入进去，可以直接使用。[代码清单 20](#) 给了一个简单的 JavaScript 代码模块。

#### 清单 20. JavaScript 代码模块示例

```
var EXPORTED_SYMBOLS = ["dwSampleModule"];

var dwSampleModule = {
    echo : function(message) {
        return "echo ==> " + message;
    }
};
```

如 [代码清单 20](#) 所示，该 JavaScript 代码模块暴露了一个对象 `dwSampleModule`，其中包含了一个 JavaScript 方法 `echo()`。模块文件以 `.jsm` 或 `.js` 为后缀，并放在扩展根目录下的 `modules` 子目录中。除此之外，还需要在扩展的 `chrome.manifest` 文件中为 `modules` 目录添加一个别名，如 `resource dwSample modules/`。这里的 `dwSample` 就是新声明的别名。在需要使用该 JavaScript 代码模块的地方，通过 `Components.utils.import()` 方法来引入，如 `Components.utils.import("resource://dwSample/dwSample.jsm");`。

`Components.utils.import()` 方法有两个参数：第一个参数是模块文件的路径，使用的是 `resource://` 协议，路径中的 `dwSample` 就是之前声明的别名；第二个参数是模块中暴露的对象所混入的作用域，默认是当前的全局对象。如果希望为引入的模块再添加额外的名称空间，可以使用这个参数。[代码清单 21](#) 给出了如何在代码中使用 JavaScript 代码模块。

#### 清单 21. 使用 JavaScript 代码模块

```
Components.utils.import("resource://dwSample/dwSample.jsm");
var message = dwSampleModule.echo("Hello!");
```

如 [代码清单 21](#) 所示，在引入了模块之后，就可以直接引用模块中暴露出来的对象 `dwSampleModule`。需要注意的是，如果一个模块在不同的地方被多次引用，所有这些引用都共享一个对象。因此尽量不要修改引用的对象。

Firefox 也提供了一些标准 JavaScript 代码模块，可供扩展使用。通过这些代码模块，可以简化一些常见的操作。如 `resource://gre/modules/NetUtil.jsm` 和 `resource://gre/modules/FileUtils.jsm` 分别提供了与网络 and 文件操作相关的实用方法。

在介绍完 JavaScript 代码模块相关的内容之后，下面介绍 XPCOM 组件。

[回页首](#)

## XPCOM

XPCOM (cross platform component object model) 是 Mozilla 产品中使用的组件对象模型，类似于微软的 COM 组件。Mozilla 产品底层的很多服务都是用 XPCOM 组件来实现的。在上面的章节中，很多地方都用到了 XPCOM 组件，比如窗口管理、标准对话框和偏好设置等。XPCOM 组件的接口定义用 XPIDL 来描述，一个组件可能实现多个接口。同样的 XPCOM 组件接口，在不同的平台上可能有不同的实现。使用者通过接口来调用组件所提供的服务。XPCOM 组件可以有多种语言绑定，包括 C/C++、JavaScript、Java、Python、Perl 和 Ruby 等。可以用这些语言来编写 XPCOM 组件的实现，也可以访问 XPCOM 组件提供的服务。对于扩展开发来说，最常见的 XPCOM 绑定语言就是 JavaScript。



在 JavaScript 中使用 XPCOM 组件时候的两个重要的对象是 `Components.classes` 和 `Components.interfaces`。`Components.classes` 包含的是系统中当前已经注册的所有 XPCOM 组件类，而 `Components.interfaces` 包含的则是所有的 XPCOM 组件接口。当需要使用某个组件类的时候，通过其标识符从 `Components.classes` 中获取。比如之前介绍过的标准对话框 XPCOM 组件的标识符是 `@mozilla.org/embedcomp/prompt-service;1`，通过 `Components.classes["@mozilla.org/embedcomp/prompt-service;1"]` 就可以获取到组件类。组件类分成一般类和单例对象两种。对于一般类来说，使用之前需要创建出相应的实例，这是通过 `createInstance()` 方法来实现的；对于单例对象来说，只需要通过 `getService()` 获取到这个对象即可。`createInstance()` 和 `getService()` 方法的参数都是 `Components.interfaces` 中包含的接口引用。因为 XPCOM 组件是通过接口来访问的，因此在使用之前，需要把组件对象转换成某个接口，再通过接口的方法来访问组件。对于实现了多个接口的组件，可以通过 `QueryInterface()` 来把组件对象转换成特定的接口。在偏好设置的 [代码清单 19](#) 中展示了这种用法。实际上，在组件类对象上调用 `createInstance()` 和 `getService()` 并传入接口作为参数的时候，就相当于先创建或找到对应的对象，再转换成参数所指定的接口。

当需要使用某个 XPCOM 组件的时候，应该通过相关的文档说明来了解该组件所实现的接口，以及每个接口的声明。在有些时候，可能会需要创建自己的 XPCOM 组件。创建 XPCOM 组件比较复杂，而且使用的场景比较少。在这里就不做论述，相关的内容见 [参考资料](#)。

在介绍完 XPCOM 组件之后，下面介绍扩展国际化相关的内容。

## [回页首](#)

## 国际化

如果你开发的扩展希望被国外的用户使用的话，就需要添加国际化的支持。完整的国际化支持包括很多方面，如界面上文本，数字、日期和货币的格式等。这里只介绍其中最常见的界面文本的国际化。

首先需要把界面上可能出现的文本都提取出来。这些文本一般来自 XUL 文件和 JavaScript 文件。对于 XUL 文件来说，其中的文本应该被提取到文档类型声明（Document Type Definition, DTD）文件中。每个 XUL 文件都应该有多个与之对应的不同语言的 DTD 文件。这些 DTD 文件应该被放在扩展根目录下的 `locale` 目录下对应的语言代码子目录中。如英语和中文的 DTD 文件，应该分别被放在 `en_US` 和 `zh_CN` 目录下。DTD 文件中包含的是实体的声明，每个实体对应于所需被本地化的一段文本。[代码清单 22](#) 给出了一个 DTD 文件的示例。

### 清单 22. 本地化 DTD 文件示例

```
<!ENTITY dwSample.111nMenuItem.label "本地化测试">
```

在 XUL 文件中，需要通过文档类型声明来引用 DTD 文件。如 `<!DOCTYPE overlay SYSTEM "chrome://dwSample/locale/overlay.dtd">`。通过 `&dwSample.111nMenuItem.label;` 的方式来引用这些实体，即在实体名称前面加上 `&`，后面加上 `;`。

DTD 文件适用于包含在 XUL 文件中的文本。如果希望在 JavaScript 代码中显示本地化文本的话，就需要用到属性文件。这里的属性文件与 Java 国际化中使用的属性文件并没有差别，是简单的名值对。为了方便读取属性文件中的内容，可以使用 XUL 中的 `stringbundle` 元素。这个元素提供了一些方法用来从属性文件中读取文本。[代码清单 23](#) 中给出了 `stringbundle` 元素的使用示例。

### 清单 23. stringbundle 元素的使用示例

```
<script>
function onLoad() {
```

```

        var bundle = document.getElementById("dwSample-string-bundle");
        var message = bundle.getFormattedString("dwSample.greeting", ["Alex"]);
        document.getElementById("messageContainer").value = message;
    }
</script>
<stringbundleset>
    <stringbundle id="dwSample-string-bundle"
        src="chrome://dwSample/locale/localized-dialog.properties" />
</stringbundleset>

```

如 [代码清单 23](#) 所示, `stringbundleset` 元素用来包含多个 `stringbundle` 元素。每个 `stringbundle` 元素用来引用一个属性文件。当需要读取文本的时候, 首先获取到 `stringbundle` 元素, 再调用其 `getString()` 或 `getFormattedString()` 方法。`getString()` 方法用来直接根据名称获取对应的文本, `getFormattedString()` 方法则可以传入参数进行替换来生成文本。属性文件中的 `%S` 和 `%n$S` 都可以作为占位符, 不过后者指定了替换时候的参数顺序。比如同样的调用方式 `getFormattedString("message", ["Alex", "Bob"])`, 如果属性文件中的写法是 `message = Hello, %S and %S`, 那么得到的文本是 `Hello, Alex and Bobb`; 如果写法换成 `message = Hello, %2$S and %1$S`, 那么得到的文本就是 `Hello, Bob and Alex`。

在介绍完扩展国际化相关的内容之后, 下面介绍如何实现扩展的自动更新。

## [回页首](#)

### 自动更新

如果你的扩展是一个长期的开发项目, 那么就会不断有版本的更新。这个时候, 使用 **Firefox** 扩展的自动更新功能就是一个很好的选择。**Firefox** 会定期查找扩展的可用更新, 并提示用户进行升级。对于扩展开发人员来说, 要做的就是扩展中启用这项能力, 并满足 **Firefox** 对扩展更新的要求。

为了启用扩展的自动更新能力, 首先需要创建一个 `update.rdf` 文件。这个文件中包含了扩展的最新版本的相关信息, 主要包括新版本的版本号和对应的 XPI 文件的下载地址等。当有新版本发布的时候, 只需要更新这个文件中的相关信息即可。在扩展的 `install.rdf` 文件中通过 `updateURL` 来声明该扩展更新时使用的 `update.rdf` 文件的 URL。`update.rdf` 应该被存放在可访问的 Web 服务器上。从 **Firefox 3** 开始, 从安全的角度出发, **Firefox** 对扩展的更新增加了更加严格的限制: `update.rdf` 的 URL 需要使用 HTTPS 链接, 如果使用 HTTP 的话, 则需要添加额外的数字签名以进行验证; 扩展新版本的 XPI 文件也需要使用 HTTPS 链接, 如果使用 HTTP 的话, 则需要添加额外的 XPI 文件的报文摘要信息。HTTPS 链接的证书失效等问题也会导致更新失败。对于大多数扩展开发者来说, 找到一个支持 HTTPS 链接的服务器存放 `update.rdf` 和相关的 XPI 文件是一件比较困难的事情。因此本文主要介绍如何使用 HTTP 链接进行更新。

首先需要下载和安装工具 **McCoy**。这个工具可以在 `install.rdf` 中添加数字签名所需的公钥, 以及对 `update.rdf` 进行签名。运行 **McCoy** 之后, 首先需要创建数字签名所需的密钥。接着通过 **Install** 功能来处理 `install.rdf`, 这一步会加上密钥的声明。对于新版本的 XPI 文件, 用 SHA 算法计算其报文摘要, 并作为 `updateHash` 的值更新到 `update.rdf` 中。最后再通过 **McCoy** 的 **Sign** 功能处理 `update.rdf` 文件。这一步会加上数字签名。把处理好的 `update.rdf` 和 XPI 文件存放到 Web 服务器上, 就可以实现扩展的自动更新。

## [回页首](#)

## 总结

通过开发 **Firefox** 扩展可以从不同的方面增强 **Firefox** 浏览器的功能。用户也习惯于使用扩展来优化浏览网页时的体验。本文详细介绍了 **Firefox** 扩展开发中的一些高级话题, 包括高级用户界面元素及其操作、使用 **XBL** 创建自定义 **XUL** 元素、**XUL** 数据模板、**JavaScript** 代码模块、**XPCOM**、国际化和实现扩展的自动更新等。理解了本文中的这些内容之后, 开发人员可以更加高效的开发出高质量 **Firefox** 扩展。

---

## 回页首

## 下载

描述	名字	大小	下载方法
包含本文中各个示例的 <b>Firefox</b> 扩展源代码 <sup>1</sup>	dwSample.zip	15KB	<a href="#">HTTP</a>

## 关于下载方法的信息

### 注意:

1. 下载之后重命名为 **dwSample.xpi** 并进行安装。从 **Firefox** 的“工具”->“**dwSample** 菜单”选择各个示例进行查看。

## 参考资料

### 学习

- 1 “[实战 Firefox 扩展开发](#)” (developerWorks 中国, 2008 年 2 月) 介绍了 **Firefox** 扩展开发的基本内容。
- 1 Mozilla [官方扩展开发资料库](#): 包含了来自 Mozilla 官方的大量扩展开发资料。
- 1 Mozilla [XUL School 教程](#) 是一个详实的 **Firefox** 扩展开发教程。
- 1 Mozilla [XUL 文档索引](#) 中包含了与 **XUL** 相关的文档链接。
- 1 [developerWorks Web development 专区](#) : 通过专门关于 **Web** 技术的文章和教程, 扩展您在网站开发方面的技能。
- 1 [developerWorks Ajax 资源中心](#) : 这是有关 **Ajax** 编程模型信息的一站式中心, 包括很多文档、教程、论坛、blog、wiki 和新闻。任何 **Ajax** 的新信息都能在这里找到。
- 1 [developerWorks Web 2.0 资源中心](#) , 这是有关 **Web 2.0** 相关信息的一站式中心, 包括大量 **Web 2.0** 技术文章、教程、下载和相关技术资源。您还可以通过 [Web 2.0 新手入门](#) 栏目, 迅速了解 **Web 2.0** 的相关概念。
- 1 查看 [HTML5 专题](#) , 了解更多和 **HTML5** 相关的知识和动向。

## 获得产品和技术

- I 下载 [Firefox 标准安装版](#) 或 [Firefox 便携版](#) 。
- I 下载 [McCoy](#) 对 Firefox 扩展进行数字签名。
- I 下载 [Hashtab](#) 在 Windows 平台上计算文件的 SHA 报文摘要。

## 讨论

- I [参与论坛讨论](#)。
- I 查看 [developerWorks 博客](#) 的最新信息。
- I 加入 [developerWorks 中文社区](#) 。查看开发人员推动的博客、论坛、组和维基，并与其他 developerWorks 用户交流。