**使用 JavaScript File API 实现文件上传**

File API 是 Mozilla 向 W3C 提出的一个草案，旨在用标准 JavaScript API 实现本地文件的读取。File API 将极大地方便 Web 端的文件上传等操作，并有望成为未来的 HTML 5 规范的一部分。本文将介绍 File API 的概况，并用两个实例展示 File API 的应用。

3http://dw1.s81c.com/developerworks/i/v17/dw-cmts-arrow.png [评论](http://www.ibm.com/developerworks/cn/web/1101_hanbf_fileupload/#icomments)

[韩 冰峰](http://www.ibm.com/developerworks/cn/web/1101_hanbf_fileupload/#authorN10038), 软件工程师, IBM

[张 顺](http://www.ibm.com/developerworks/cn/web/1101_hanbf_fileupload/#authorN1004D), 软件工程师, IBM

[莫 映](http://www.ibm.com/developerworks/cn/web/1101_hanbf_fileupload/#authorN10062), 高级工程师, IBM

2011 年 1 月 24 日

* [expand](http://www.ibm.com/developerworks/cn/web/1101_hanbf_fileupload/#toggle)**内容**



在 IBM Bluemix 云平台上开发并部署您的下一个应用。

[**开始您的试用**](https://developer.ibm.com/sso/bmregistration?lang=zh_CN&ca=dwchina-_-bluemix-_-1101_hanbf_fileupload-_-sidebar)

**概述**

以往对于基于浏览器的应用而言，访问本地文件都是一件头疼的事情。虽然伴随着 Web 2.0 应用技术的不断发展，JavaScript 正在扮演越来越重要的角色，但是出于安全性的考虑，JavaScript 一直是无法访问本地文件的。于是，为了在浏览器中能够实现诸如拖拽并上传本地文件这样的功能，我们就不得不求助于特定浏览器所提供的各种技术了。比如对于 IE，我们需要通过 ActiveX 控件来获取对本地文件的访问能力，而对于 Firefox，同样也要借助插件开发。由于不同浏览器的技术实现不尽相同，为了让程序能够支持多浏览器，我们的程序就会变得十分复杂而难于维护。不过现在，这一切都因为 File API 的出现而得到了彻底的改变。

File API 是 Mozilla 向 W3C 提交的一个草案，旨在推出一套标准的 JavaScript API，其基本功能是实现用 JavaScript 对本地文件进行操作。出于安全性的考虑，该 API 只对本地文件提供有限的访问。有了它，我们就可以很轻松的用纯 JavaScript 来实现本地文件的读取和上传了。目前，FireFox 3.6 是最先支持这一功能的浏览器。除此以外，最新版本的 Google Chrome 浏览器和 Safari 浏览器也有相应的支持。File API 有望成为 W3C 目前正在制定的未来 HTML 5 规范当中的一部分。

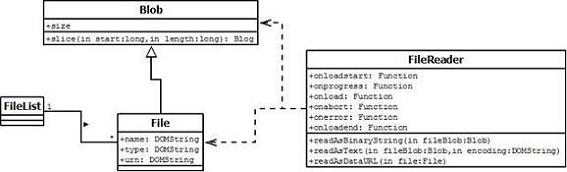
本文后续章节将对 File API 做一个基本的介绍。并为读者演示：如何利用 File API 实现基于 file input 控件的本地文件读取与上传；以及利用拖拽实现从用户系统批量上传文件的功能。通过本文，读者可以了解最新的 File API 的概况以及使用，从而为构建具有更好用户体验的 Web 2.0 应用提供参考。

[**回页首**](http://www.ibm.com/developerworks/cn/web/1101_hanbf_fileupload/#ibm-pcon)

**File API 概览**

File API 由一组 JavaScript 对象以及事件构成。赋予开发人员操作在 <input type=”file” … /> 文件选择控件中选定文件的能力。图 1 展示了 File API 所有的 JavaScript 的组合关系。

**图 1. File API 相关类图**



类型 FileList 包含一组 File 对象。通常 FileList 对象可以从表单中的文件域（<input type=”file” .../>）中拿取。Blob 对象代表浏览器所能读取的一组原始二进制流。Blob 对象中，属性 size 表示流的大小。函数 slice() 可以将一个长的 Blob 对象分割成小块。File 对象继承自 Blob 对象，在 Blob 对象基础上增加了和 File 相关的属性。其中，属性 name 表示文件的名字，这个名字去掉了文件的路径信息，而只保留了文件名。属性 type 表示文件的 MIME 类型。属性 urn 则代表这个文件的 URN 信息。为完成文件读取的操作，一个 FileReader 对象实例会关联 File 或 Blob 对象，并提供三种不同的文件读取函数以及 6 种事件。参见表 1 及表 2。

**表 1. 文件读取函数**

|  |  |
| --- | --- |
| **函数名称** | **功能** |
| readAsBinaryString() | 读取文件内容，读取结果为一个 binary string。文件每一个 byte 会被表示为一个 [0..255] 区间内的整数。函数接受一个 File 对象作为参数。 |
| readAsText() | 读取文件内容，读取结果为一串代表文件内容的文本。函数接受一个 File 对象以及文本编码名称作为参数。 |
| readAsDataURL | 读取文件内容，读取结果为一个 data: 的 URL。DataURL 由 RFC2397 定义，具体可以参考 <http://www.ietf.org/rfc/rfc2397.txt>。 |

当我们通过代码调用FILE API的时候，浏览器将在一个安全的环境中(沙箱)建立一个模拟的文件系统，所有的文件读写操作都是在这个安全的环境中进行的。

**表 2. 文件读取事件**

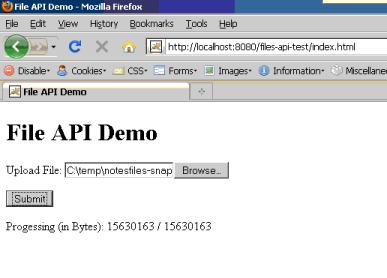
|  |  |
| --- | --- |
| **事件名称** | **事件说明** |
| Onloadstart | 文件读取开始时触发。 |
| Progress | 当读取进行中时定时触发。事件参数中会含有已读取总数据量。 |
| Abort | 当读取被中止时触发。 |
| Error | 当读取出错时触发。 |
| Load | 当读取成功完成时触发。 |
| Loadend | 当读取完成时，无论成功或者失败都会触发。 |

[**回页首**](http://www.ibm.com/developerworks/cn/web/1101_hanbf_fileupload/#ibm-pcon)

**File API 简单示例**

接下来我们用一个简单的例子展示 File API 的基本用法。这个示例包含两个代码文件，index.html 包含 Web 端的 HTML 代码和处理上传的 JavaScript 代码；upload.jsp 包含服务器端接收上传文件的代码。请参见附件中的 sourcecode.zip。在这个例子中，我们将显示一个传统的带有 File 选择域的表单。当用户选择文件，点击提交后，我们使用 File API 读取文件内容，并通过 XMLHttpRequest 对象，用 Ajax 的方式将文件上传到服务器上。图 2 展示了运行中的演示截图。

**图 2 File API 演示**



我们逐步展示其中的代码。清单 1 给出了代码的 HTML 部分。

**清单 1 示例代码的 HTML 部分**

<body>

<h1>File API Demo</h1>

<p>

<!-- 用于文件上传的表单元素 -->

<form name="demoForm" id="demoForm" method="post" enctype="multipart/form-data"

action="javascript: uploadAndSubmit();">

<p>Upload File: <input type="file" name="file" /></p>

<p><input type="submit" value="Submit" /></p>

</form>

<div>Progessing (in Bytes): <span id="bytesRead">

</span> / <span id="bytesTotal"></span>

</div>

</p>

</body>

可以看到，我们用普通的 <form> 标签来包含一个传统的 <input type=”file” … /> 元素。在 <form> 中还有一个 submit 元素。在 <form> 之外有一些 <span> 元素用来表示已读取和总共的数据量。<form> 的 action 属性指向了一个 JavaScript 函数 uploadAndSubmit()。这个函数完成了读取文件并上传的过程。函数代码见清单 2。

**清单 2 读取文件并上传的 JavaScript 函数**

function uploadAndSubmit() {

var form = document.forms["demoForm"];

if (form["file"].files.length > 0) {

// 寻找表单域中的 <input type="file" ... /> 标签

var file = form["file"].files[0];

// try sending

var reader = new FileReader();

reader.onloadstart = function() {

// 这个事件在读取开始时触发

console.log("onloadstart");

document.getElementById("bytesTotal").textContent = file.size;

}

reader.onprogress = function(p) {

// 这个事件在读取进行中定时触发

console.log("onprogress");

document.getElementById("bytesRead").textContent = p.loaded;

}

reader.onload = function() {

// 这个事件在读取成功结束后触发

console.log("load complete");

}

reader.onloadend = function() {

// 这个事件在读取结束后，无论成功或者失败都会触发

if (reader.error) {

console.log(reader.error);

} else {

document.getElementById("bytesRead").textContent = file.size;

// 构造 XMLHttpRequest 对象，发送文件 Binary 数据

var xhr = new XMLHttpRequest();

xhr.open(/\* method \*/ "POST",

/\* target url \*/ "upload.jsp?fileName=" + file.name

/\*, async, default to true \*/);

xhr.overrideMimeType("application/octet-stream");

xhr.sendAsBinary(reader.result);

xhr.onreadystatechange = function() {

if (xhr.readyState == 4) {

if (xhr.status == 200) {

console.log("upload complete");

console.log("response: " + xhr.responseText);

}

}

}

}

}

reader.readAsBinaryString(file);

} else {

alert ("Please choose a file.");

}

}

在这个函数中，首先我们找到含有 <input type=”file” … /> 元素的 <form>，并找到含有上传文件信息的 <input> 元素。如 <input> 元素中不含有文件，说明用户没有选择任何文件，此时将报错。

**清单 3 找到 <input> 元素**

var form = document.forms["demoForm"];

if (form["file"].files.length > 0)

{

var file = form["file"].files[0];

… …

}

else

{

alert ("Please choose a file.");

}

这里，从 form[“file”].files 返回的对象类型即为提到的 FileList。我们从中拿取第一个元素。之后，我们构建 FileReader 对象：

var reader = new FileReader();

在 onloadstart事件触发时，填充页面上表示读取数据总量的 <span> 元素。参见清单 4

**清单 4 onloadstart 事件**

reader.onloadstart = function()

{

console.log("onloadstart");

document.getElementById("bytesTotal").textContent = file.size;

}

在 onprogress 事件触发时，更新页面上已读取数据量的 <span> 元素。参见清单 5

**清单 5 onprogress 事件**

reader.onprogress = function(p) {

console.log("onloadstart");

document.getElementById("bytesRead").textContent = p.loaded;

}

最后的 onloadend 事件中，如果没有错误，我们将读取文件内容，并通过 XMLHttpRequest 的方式上传。

**清单 6 onloadend 事件**

reader.onloadend = function()

{

if (reader.error)

{

console.log(reader.error);

}

else

{

// 构造 XMLHttpRequest 对象，发送文件 Binary 数据

var xhr = new XMLHttpRequest();

xhr.open(/\* method \*/ "POST",

/\* target url \*/ "upload.jsp?fileName=" + file.name

/\*, async, default to true \*/);

xhr.overrideMimeType("application/octet-stream");

xhr.sendAsBinary(reader.result);

… …

}

}

按照 File API 的规范，我们也可以将事件 onloadend 的处理拆分为事件 error 以及事件 load 的处理。

在这个示例中，我们后台使用一个 JSP 来处理上传。JSP 代码如清单 7。

**清单 7 处理上传的 JSP 代码**

<%@ page import="java.io.\*" %><%

BufferedInputStream fileIn = new

BufferedInputStream(request.getInputStream());

String fn = request.getParameter("fileName");

byte[] buf = new byte[1024];

//接收文件上传并保存到 d:\

File file = new File("d:/" + fn);

BufferedOutputStream fileOut = new BufferedOutputStream(new

FileOutputStream(file));

while (true) {

// 读取数据

int bytesIn = fileIn.read(buf, 0, 1024);

System.out.println(bytesIn);

if (bytesIn == -1)

{

break;

}

else

{

fileOut.write(buf, 0, bytesIn);

}

}

fileOut.flush();

fileOut.close();

out.print(file.getAbsolutePath());

%>

在这段 JSP 代码中，我们从 POST 请求中接受文件名字以及二进制数据。将二进制数据写入到服务器的“D:\”路径中。并返回文件的完整路径。以上代码可以在最新的 Firefox 3.6 中调试通过。

[**回页首**](http://www.ibm.com/developerworks/cn/web/1101_hanbf_fileupload/#ibm-pcon)

**使用拖拽上传文件**

前面我们介绍了怎样通过 HTML5 的 File API 来读取本地文件内容并上传到服务器，通过这种方式已经能够满足大部分用户的需求了。其中一个不足是用户只能通过点击“浏览”按钮来逐个添加文件，如果需要批量上传文件，会导致用户体验不是很友好。而在桌面应用中，用户一般可以通过鼠标拖拽的方式方便地上传文件。拖拽一直是 Web 应用的一个软肋，一般浏览器都不提供对拖拽的支持。虽然 Web 程序员可以通过鼠标的 mouseenter，mouseover 和 mouseout 等事件来实现拖拽效果，但是这种方式也只能使拖拽的范围局限在浏览器里面。一个好消息是 HTML5 里面不仅加入了 File API，而且加入了对拖拽的支持，Firefox 3.5 开始已经对 File API 和拖拽提供了支持。下面我们先简要介绍一下拖拽的使用，然后用一个例子来说明如何通过拖拽上传文件。

**拖拽简介**

拖拽一般涉及两个对象：拖拽源和拖拽目标。

拖拽源：在 HTML5 草案里如果一个对象可以作为源被拖拽，需要设置 draggable 属性为 true 来标识该对象可作为拖拽源。然后侦听源对象的 dragstart 事件，在事件处理函数里设置好 DataTransfer。在 DataTransfer 里可以设置拖拽数据的类型和值。比如是纯文本的值，可以设置类型为"text/plain"，url 则把类型设置为"text/uri-list"。 这样，目标对象就可以根据期望的类型来选择数据了。

拖拽目标：一个拖拽目标必须侦听 3 个事件。

dragenter：目标对象通过响应这个事件来确定是否接收拖拽。如果接收则需要取消这个事件，停止时间的继续传播。

dragover：通过响应这个事件来显示拖拽的提示效果。

drop：目标对象通过响应这个事件来处理拖拽数据。在下面的例子里我们将在 drop 事件的处理函数里获取 DataTransfer 对象，取出要上传的文件。

由于本文主要介绍 File API，对这部分不作详细解释，感兴趣的读者可以参考 HTML5 草案（见参考资料）。

**拖拽上传文件实例**

下面以一个较为具体的例子说明如何结合拖拽和 File API 来上传文档。由于直接和桌面交互，所以我们不需要处理拖拽源，直接在目标对象里从 DataTransfer 对象获取数据即可。

首先，我们需要创建一个目标容器用来接收拖拽事件，添加一个 div 元素即可。然后用一个列表来展示上传文件的缩略图，进度条及文件名。参见清单 8 的 HTML 代码和图 3 的效果图。详细代码请参见附件中的 dnd.html 文件。

**清单 8 拖曳目标的 HTML 代码**

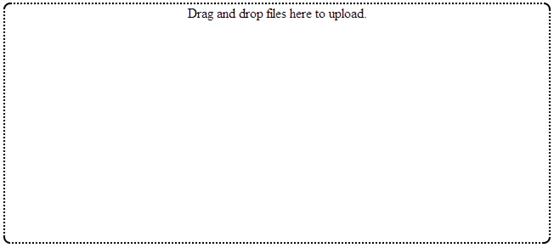
<div id="container">

<span>Drag and drop files here to upload.</span>

<ul id="fileList"></ul>

</div>

**图 3 拖曳目标**



拖拽目标创建好之后，我们需要侦听其对应的事件 dragenter，dragover 和 drop。在 dragenter 事件处理函数里，我们只是简单地清除文件列表，然后取消 dragenter 事件的传播，表示我们接收该事件。更加妥当的作法是判断 DataTransfer 里的数据是否为文件，这里我们假设所有拖拽源都是文件。dragover 事件里我们取消该事件，使用默认的拖拽显示效果。在 drop 事件里我们注册了 handleDrop 事件处理函数来获取文件信息并上传文件。清单 9 展示了这些事件处理函数。

**清单 9 设置事件处理函数**

function addDNDListeners()

{

var container = document.getElementById("container");

var fileList = document.getElementById("fileList");

// 拖拽进入目标对象时触发

container.addEventListener("dragenter", function(event)

{

fileList.innerHTML ='';

event.stopPropagation();

event.preventDefault();

}, false);

// 拖拽在目标对象上时触发

container.addEventListener("dragover", function(event)

{

event.stopPropagation();

event.preventDefault();

}, false);

// 拖拽结束时触发

container.addEventListener("drop", handleDrop, false);

}

window.addEventListener("load", addDNDListeners, false);

**处理 drop 事件**

用户在拖拽结束时松开鼠标触发 drop 事件。在 drop 事件里，我们可以通过 event 参数的 DataTransfer 对象获取 files 数据，通过遍历 files 数组可以获取每个文件的信息。然后针对每个文件，创建 HTML 元素来显示缩略图，进度条和文件名称。File 对象的 getAsDataURL 可以将文件内容以 URL 的形式返回，对图片文件来说可以用来显示缩略图。要注意的一点是，在 drop 事件处理函数里要取消事件的继续传播和默认处理函数，结束 drop 事件的处理。清单 10 展示了 drop 事件的处理代码。

**清单 10 drop 事件的处理**

function handleDrop(event)

{

// 获取拖拽的文件列表

var files = event.dataTransfer.files;

event.stopPropagation();

event.preventDefault();

var fileList = document.getElementById("fileList");

// 展示文件缩略图，文件名和上传进度，上传文件

for (var i = 0; i < files.length; i++)

{

var file = files[i];

var li = document.createElement('li');

var progressbar = document.createElement('div');

var img = document.createElement('img');

var name = document.createElement('span');

progressbar.className = "progressBar";

img.src = files[i].getAsDataURL();

img.width = 32;

img.height = 32;

name.innerHTML = file.name;

li.appendChild(img);

li.appendChild(name);

li.appendChild(progressbar);

fileList.appendChild(li);

uploadFile(file, progressbar)

}

}

**上传文件**

我们可以通过 XMLHttpRequest 对象的 sendAsBinary 方法来上传文件，通过侦听 upload 的 progress，load 和 error 事件来监测文件上传的进度，成功完成或是否发生错误。在 progress 事件处理函数里我们通过计算已经上传的比例来确定进度条的位置。参见清单 11。图 4 展示了上传文件的效果图。

**清单 11 上传文件**

function uploadFile(file, progressbar)

{

var xhr = new XMLHttpRequest();

var upload = xhr.upload;

var p = document.createElement('p');

p.textContent = "0%";

progressbar.appendChild(p);

upload.progressbar = progressbar;

// 设置上传文件相关的事件处理函数

upload.addEventListener("progress", uploadProgress, false);

upload.addEventListener("load", uploadSucceed, false);

upload.addEventListener("error", uploadError, false);

// 上传文件

xhr.open("POST", "upload.jsp?fileName="+file.name);

xhr.overrideMimeType("application/octet-stream");

xhr.sendAsBinary(file.getAsBinary());

}

function uploadProgress(event)

{

if (event.lengthComputable)

{

// 将进度换算成百分比

var percentage = Math.round((event.loaded \* 100) / event.total);

console.log("percentage:" + percentage);

if (percentage < 100)

{

event.target.progressbar.firstChild.style.width = (percentage\*2) + "px";

event.target.progressbar.firstChild.textContent = percentage + "%";

}

}

}

function uploadSucceed(event)

{

event.target.progressbar.firstChild.style.width = "200px";

event.target.progressbar.firstChild.textContent = "100%";

}

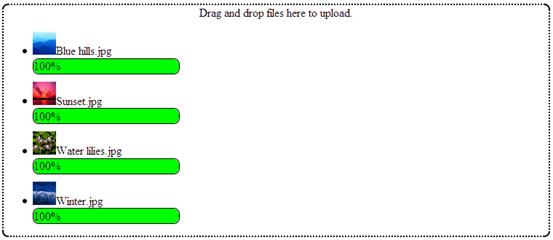
function uploadError(error)

{

alert("error: " + error);

}

**图 4 上传文件的效果图**



[**回页首**](http://www.ibm.com/developerworks/cn/web/1101_hanbf_fileupload/#ibm-pcon)

**小结**

本文通过对 File API 规范的讲解，以及两个展示其使用方法的例子，为大家提前揭示了作为未来 HTML5 重要组成部分的 JavaScript File API 的全貌。利用它，结合其他 HTML5 的新特性，比如 Drag&Drop，我们可以利用纯 JavaScript 方案，为用户提供更好使用体验的 Web 应用，与此同时，这样的一致化方案也使我们避免了以往跨浏览器支持所花费的巨大代价。相信 File API 的出现和广泛应用，将会是未来的 Web 2.0 应用的大势所趋。

[**回页首**](http://www.ibm.com/developerworks/cn/web/1101_hanbf_fileupload/#ibm-pcon)