**理解DOMString、Document、FormData、Blob、File、ArrayBuffer数据类型**

by [zhangxinxu](http://www.zhangxinxu.com/) from [http://www.zhangxinxu.com](http://www.zhangxinxu.com/)  
本文地址：<http://www.zhangxinxu.com/wordpress/?p=3725>

**一、XMLHttpRequest 2.0的家臣们**

我大学那会儿，一个称为Ajax的东西对前端行业造成了深远影响，不仅是JS语言，而包括前端地位、职位兴起以及工作分工等。抛开IE6浏览器不谈，其他浏览器的Ajax实际上都是借助XMLHttpRequest实现的。

然后，好多年过去了，XMLHttpRequest带着两位家臣，DOMString和Document数据类型攻城略地，几乎一统天下。

然时代是发展的，人们群众的需求是旺盛的，HTML5犹如冉冉升起的新星开始普照大地，;恩泽大众。XMLHttpRequest由于就两个家臣DOMString和Document，且并不是100%听话。因此，其已经开始hold不住HTML5的耀眼光芒了。为了顺应时代的潮流，XMLHttpRequest凹凸曼变身升级到2.0，变化诸多，其中一个很重要的变化就是广招家臣，扩张实力，与HTML5一起完成千秋万载之大业。

这些家臣有：DOMString、Document、FormData、Blob、File、ArrayBuffer这些类型。也就是在XMLHttpRequest Level 2背景下，我们Ajax可以发送任意这些类型的数据。有了诸多忠实可靠的家臣，XMLHttpRequest Level 2犹如织田信长般势不可挡，前途无量！

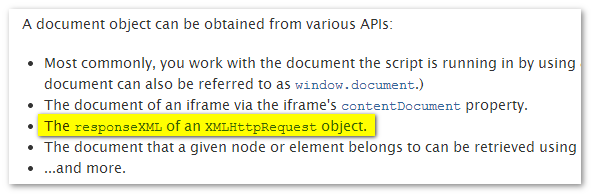
织田信长家臣有：羽柴秀吉、柴田胜家、明智光秀、竹中半兵卫、黑田官兵卫、织田信忠、泷川一益、丹羽长秀、前田利家、池田恒兴、佐久间信盛、森兰丸、九鬼嘉隆

**二、家臣之DOMString**

跟着XMLHttpRequest闯南走北很多年，看名字似乎很嚣张且高深莫测。实际上，在JavaScript中，DOMString就是String。规范解释说DOMString指的是UTF-16字符串，而JavaScript正是使用了这种编码的字符串，因此，在Ajax中，DOMString就等同于JS中的普通字符串。

大家应该都与XMLHttpRequest中数据返回属性之responseText打过交道吧，按照我的理解，这厮就是与DOMString数据类型发生关系的，表明返回的数据是常规字符串。

**三、家臣之Document数据类型**

如果单纯看Document对象，则解释很多，在这里，我们只要关注下图标注的这一个：  


可以看到，实际上就是XMLHttpRequest中数据返回属性之responseXML，也就是可以解析为XML的数据。因此，这里的Document数据类似你就可以近似看成XML数据类型。

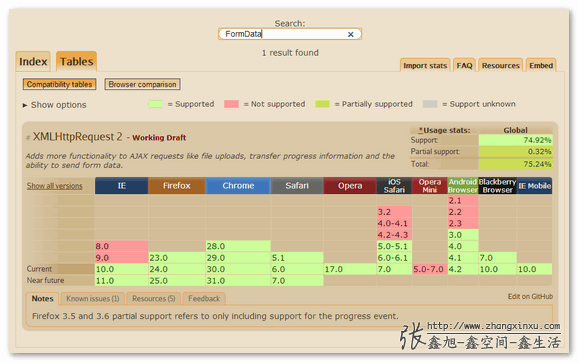
DOMString和Document都是XMLHttpRequest时代就跟随的数据类型，元老级。下面这些数据类型都是XMLHttpRequest 2.0新增的，新招的家臣，各怀绝技哦！

**四、家臣之FormData对象**

XMLHttpRequest Level 2添加了一个新的接口FormData. 利用FormData对象，我们可以通过JavaScript用一些键值对来模拟一系列表单控件，我们还可以使用XMLHttpRequest的send()方法来异步的提交这个”表单”。比起普通的ajax, 使用FormData的最大优点就是我们可以异步上传一个二进制文件。

以上为官方口吻的解释，略抽象。我们应该都用过jQuery，其中有个方法叫做serialize(), 作用就是表单序列化，也就是以查询字符串形式获得类表单post/get的数据给Ajax请求，例如：userid=123&username=zxx.

FormData对象的作用就类似于这里的serialize()方法，不过FormData是浏览器原生的，且支持二进制文件，是个一眼就会让人喜欢的很赞的东西！

**兼容性**如下：  


IE10+浏览器已经良好支持了，下面要介绍的其他家臣也都是IE10+支持。

实际使用是作为构造函数，如下：

new FormData ([可选]HTMLFormElement)

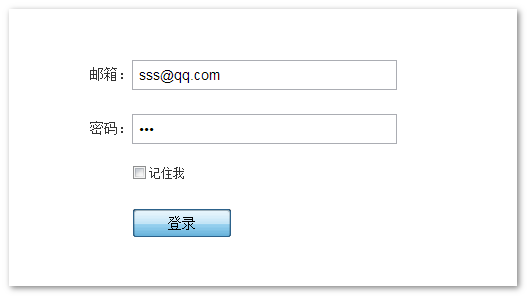
HTMLFormElement这个参数可选，可有可无。表示form表单元素，就是我们要序列化，要提交的那个表单元素。

例如：

var newFormData = new FormData(someFormElement);

newFormData就是someFormElement这个表单元素中所有键值对数据了。

您可以狠狠地点击这里：[FormData对象与表单数据获取demo](http://www.zhangxinxu.com/study/201310/formdata-form-data-get.html)

demo页面为一个普通的登录表单，截图如下：  


点击登录执行Ajax登录，不过这里是采用FormData格式发送的。

相关JS代码如下：

document.querySelector("#formData").addEventListener("submit", function(event) {

var myFormData = new FormData(this);

var xhr = new XMLHttpRequest();

xhr.open(this.method, this.action);

xhr.onload = function(e) {

if (xhr.status == 200 && xhr.responseText) {

// 显示：'欢迎你，' + xhr.responseText;

this.reset();

}

}.bind(this);

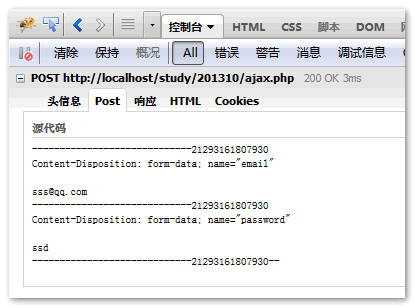
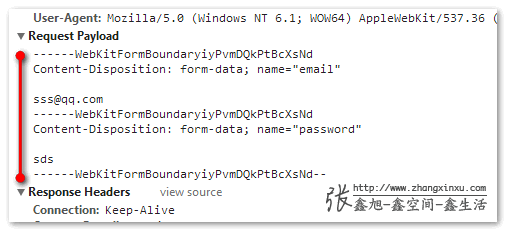
// 发送FormData对象数据

xhr.send(myFormData);

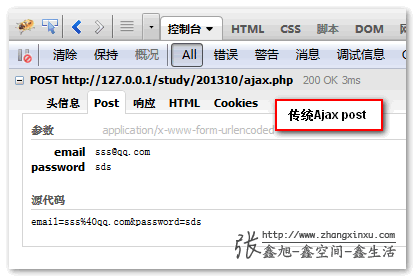
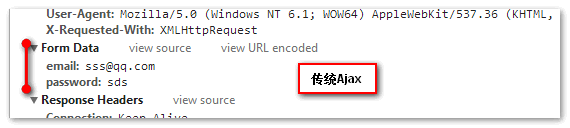
// 阻止默认的表单提交

event.preventDefault();

}, false);

我们打开工具查看下请求：  
  


以上分别是Firebug和Chrome开发者工具查看的结果。

我们再看下传统Ajax请求：  
  


差异还是比较大的。  
FormData提交格式的每个数据分三部分：

* 第一部分也就是第一行，表示“分界线(boundary)”，我尚未深入研究这个分界线，不过，我没估计错的话，二进制大文件分隔传输时候，就是使用这个分界线。在webkit核心中，使用*“——WebKitFormBoundary”加16位随机Base64位编码的字符串*作为分隔边界。根据Firebug的显示，Firefox中，似乎是使用*很多个"-"加时间戳*进行边界分隔的。这里的边界的作用比较单纯，可能就是把表单的这两个字段作为两个独立数据流传输。
* 第二部分也就是第二行，表示内容配置，这里都是统一的form-data（因为是FormData对象格式提交的），然后紧跟着name键值。
* 第三部分就是第三行，表示传输的值。

虽然前台传输差异较大，但是，后台的处理是可以一致的，例如，我这里的PHP代码就非常简单：

<?php

$username = $\_POST['email'];

if (isset($username) == true) {

echo $username;

} else {

echo '';

}

?>

FormData对象还有一个方法，为append()方法，可以人为的给当前FormData对象添加一个键/值对。

语法如下：

void append(DOMString 键, Blob 值, [可选] DOMString 文件名);

void append(DOMString 键, DOMString 值);

语法第一行出现了Blob, 这是我们下面要介绍的家臣之一，您可以先记住，这是用来表示二进制文件的，后面的文件名可选，据说，如果缺省，且传输的是Blob对象，则会使用"blob"代替。  
第二行就是比较常规的用法，DOMString这个家臣已经介绍了，在JavaScript中就是普通字符串的意思。因此，比方说我们要额外提交个token值，可能就是：

myFormData.append("token", "ce509193050ab9c2b0c518c9cb7d9556");

于是，后台就可以get token这个值了。

大家自行补脑，我就不再撑篇幅了。

**五、家臣之Blob数据对象**

一个Blob对象就是一个包含有只读原始数据的类文件对象。Blob对象中的数据并不一定得是JavaScript中的原生形式。File接口基于Blob, 继承了Blob的功能，并且扩展支持了用户计算机上的本地文件。

创建Blob对象的方法有几种，可以调用Blob构造函数，还可以使用一个已有Blob对象上的slice()方法切出另一个Blob对象，还可以调用canvas对象上的toBlob方法。

以上为MDN上官方口吻的解释。实际上，Blob是计算机界通用术语之一，全称写作：BLOB (binary large object)，表示二进制大对象。MySql/Oracle数据库中，就有一种Blob类型，专门存放二进制数据。

在实际Web应用中，Blob更多是图片二进制形式的上传与下载，虽然其可以实现几乎任意文件的二进制传输。

举个例子，使用Blob从服务器上GET某mm的图片（只要关心标红的部分）：

var xhr = new XMLHttpRequest();

xhr.open("get", "mm1.jpg", true);

xhr.responseType = "blob";

xhr.onload = function() {

if (this.status == 200) {

var blob = this.response; // this.response也就是请求的返回就是Blob对象

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

img.onload = function(e) {

window.URL.revokeObjectURL(img.src); // 清除释放

};

img.src = window.URL.createObjectURL(blob);

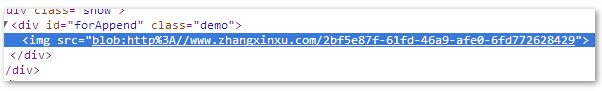
eleAppend.appendChild(img);

}

}

xhr.send();

您可以狠狠地点击这里：[Blob获取图片并二进制显示demo](http://www.zhangxinxu.com/study/201310/blob-get-image-show.html)

我们查看demo页面这个mm图片元素，会发现其URL地址既不是传统HTTP，也不是Base64 URL，而是Blob形式~如下截图示意：  
  


这就是Blob在Web开发中非常重要的一个功能——创建Blob网址。上述代码涉及XMLHttpRequest 2一些重要知识点，以及window.URL相关技术，都是可以深入挖掘学习的部分，但，不是本文重点，以后有机会会细致阐述。

但是，并不是所有的图片都能以Blob形式请求，因为，毕竟是Ajax请求嘛，还是有一定的跨域限制。XMLHttpRequest 2虽然支持跨源资源共享(CORS)，但是，还是需要对Access-Control-Allow-Origin的设置，允许来自那个域名的这类请求，例如，允许本人的站点Blob请求你服务器上的图片资源，你可以设置：

Access-Control-Allow-Origin: http://zhangxinxu.com

要允许任何域向您提交请求，可以设置：

Access-Control-Allow-Origin: \*

我们都知道CSS3的font-face属性，在Firefox浏览器下，如果字体文件跨域（包括跨子域），是显示不出来的，也是通过

Access-Control-Allow-Origin: \*

设置解决。其实，本质是一样的。

由于权限原因，我的个人站点无法配置Access-Control-Allow-Origin，我测试了下，新浪微博的图片是无法二进制请求的，不过我的前东家，[xiaomishu.com](http://www.xiaomishu.com)的图片都是可以Ajax请求并Blob显示的，悄悄告诉大家，是我当初动的手脚，(\*^\_\_^\*) 嘻嘻……

**属性**  
Blob对象有两个属性，参见下表：

|  |  |  |
| --- | --- | --- |
| **属性名** | **类型** | **描述** |
| **size** | unsigned long long(表示可以很大的数值) | Blob对象中所包含数据的大小。字节为单位。 **只读。** |
| **type** | DOMString | 一个字符串，表明该Blob对象所包含数据的MIME类型。例如，上demo图片MIME类似就是”image/jpeg“. 如果类型未知，则该值为空字符串。 **只读。** |

今天在微博上看到一个表单提交之前判断文件大小并作阻止的tip，实际上，就是使用的Blob对象的size属性。

**构造函数**  
与FormData对象类似，Blob也有一个构造函数用法。语法如下：

Blob Blob(

[可选] Array parts,

[可选] BlobPropertyBag properties

);

例如：

var myBlob= new Blob(arrayBuffer);

其中，两个参数的含义是：

parts

一个数组，包含了将要添加到Blob对象中的数据。数组元素可以是任意多个的ArrayBuffer, ArrayBufferView(typed array), Blob, 或者DOMString对象。

properties

一个对象，设置Blob对象的一些属性。目前仅支持一个type属性，表示Blob的类型。

**方法**  
Blob对象有个很重要的方法-slice()，作用是，可以实现文件的分割！

这个slice()有一段不堪回首的历史，不过现在大家不要关心。目前的slice()方法已经跟JS中数组啊，字符串的slice方法用法一致了。如下：

Blob slice(

[可选] long long start,

[可选] long long end,

[可选] DOMString contentType

};

参数释义：

start

开始索引，可以为负数，语法类似于数组的slice方法。默认值为0.

end

结束索引，可以为负数，语法类似于数组的slice方法。默认值为最后一个索引。

contentType

新的Blob对象的MIME类型，这个值将会成为新的Blob对象的type属性的值，默认为一个空字符串。

显然，此方法返回的数据格式还是Blob对象，不过是指定范围复制的新的Blob对象。注意，如果start参数的值比源Blob对象的size属性值还大，则返回的Blob对象的size值为0，也就是不包含任何数据。

**六、家臣之File对象**

File顾名思意就是“文件”，通常而言，表示我们使用file控件(<input type="file">)选择的[FileList](https://developer.mozilla.org/en-US/docs/Web/API/FileList)对象，或者是使用拖拽操作搞出的[DataTransfer](https://developer.mozilla.org/en-US/docs/DragDrop/DataTransfer)对象。

这里的File对象也是二进制对象，因此，从属于Blob对象，Blob对象的一些属性与方法，File对象同样适合，且推荐使用Blob对象的属性与方法。

File对象自身也有一些属性与方法，但是，有些已经过时——不推荐使用，因此，当前很多HTML5 Ajax文件上传下载的教程中出现是属性和方法都是过时的，不要盲目Copy，请大家明辨！

**属性**

File.lastModifiedDate[只读]

文件对象最后修改的日期

File.name[只读]

文件对象的名称

File.fileName[只读] [过时不推荐使用]

文件对象的名称（请使用File.name代替）

File.fileSize[只读] [过时不推荐使用]

文件对象的大小（请使用Blob.size代替）

Blob.size[只读]

Blob对象包含数据的字节大小

Blob.type[只读]

一个字符串，表明该Blob对象所包含数据的MIME类型

**方法**

File.getAsBinary()[过时不推荐使用]

二进制形式返回文件数据（请使用FileReader对象的FileReader.readAsBinaryString()方法代替）

File.getAsDataURL()[过时不推荐使用]

返回文件data:URL编码字符串数据（请使用FileReader对象的FileReader.readAsDataURL()方法代替）

File.getAsText(string encoding)[过时不推荐使用]

以给定的字符串编码返回文件数据解释后的文本（请使用FileReader对象的FileReader.readAsText()方法代替）

Blob.size[只读]

Blob对象包含数据的字节大小

Blob.type[只读]

一个字符串，表明该Blob对象所包含数据的MIME类型。

上面有提到FileReader对象，这货是相当的有货，之前有人曾问我，如何将图片转换成Data base64 url格式，其中一个方法就是FileReader.readAsDataURL()方法（还有就是canvas元素的toDataURL()和toDataURLHD()方法），然与本文主旨无关，暂不赘述；如您有兴趣，页面底部有其相关知识点链接，可自行概览。

**七、家臣之ArrayBuffer对象**

//zxx:ArrayBuffer对象牵扯知识点非常多，这里仅接触肌肤，深入接触下次会专门再说下。

很术语的解释有：

ArrayBuffer表示二进制数据的原始缓冲区，该缓冲区用于存储各种类型化数组的数据。

ArrayBuffer是二进制数据通用的固定长度容器。

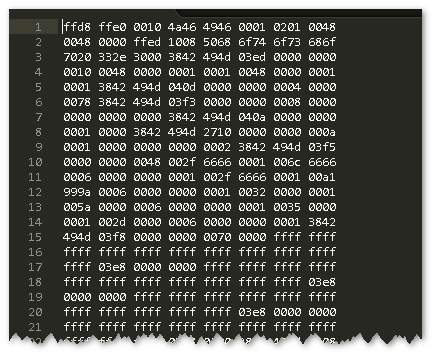
所谓术语，就是小白看不懂的解释语。我再用通俗语解释下，希望大家可以有点感性的认识：

术语中，提到“二进制”，我们脑中应该会出现01010111之类；提到“缓冲”，会联想到在线视频提前加载一部分视频的那个缓冲。但是，两个合起来，“二进制数据缓冲区”，脑补就不连贯了，焦虑产生~~

现在，听我的，上面概念全部扔掉。所谓ArrayBuffer就是个装着2进制数据的对象。或者想象成带了个名叫“缓冲”帽子的二进制数据。然后直接关联：ArrayBuffer ＝ 2进制。

上面＝表示关联，不是相等，诸位。

例如，我们设置Ajax请求的responseType为”arraybuffer“，我们去请求某mm图片，返回的response就是ArrayBuffer，就是个二进制对象。什么缓冲不缓冲的，千万别补脑这个。

如果还觉得概念抽象，可以看下面的具体认知：  
大家可能玩过神器编辑器Sublime Text, 我们随便找张图片拖进去，会发现是类似下面这样子的代码：  


Sublime Text以16进制的形式显示图片资源，ArrayBuffer的差别在于是二进制，因此，我们可以把ArrayBuffer的形体脑补成——上图的数字全是的0101 1000 1101之类的。Get it否？

上面提到的Blob对象也是二进制，那Blob和ArrayBuffer有啥区别呢？

Blob可以append ArrayBuffer数据，也就是Blob是个更高一级的大分类，类似领导的感觉。ArrayBuffer则是具有某种恶魔果实的尖兵。

ArrayBuffer存在的意义就是作为数据源提前写入在内存中，就是提前钉死在某个区域，长度也固定，万年不变。于是，当我们要处理这个ArrayBuffer中的二进制数据，例如，分别8位，16位，32位转换一遍，这个数据都不会变化，3种转换共享数据。

So，ArrayBuffer就是缓冲出来的打死不动的二进制对象。

注意，ArrayBuffer本身是不能读写的，需要借助类型化数组或DataView对象来解释原始缓冲区（宰割原始二进制数据）。

**类型化数组**  
类型化数组(Typed Arrays)是JavaScript中新出现的一个概念，专为访问原始的二进制数据而生。

类型数组的类型有：

|  |  |  |
| --- | --- | --- |
| **名称** | **大小 (以字节为单位)** | **说明** |
| **Int8Array** | 1 | 8位有符号整数 |
| **Uint8Array** | 1 | 8位无符号整数 |
| **Int16Array** | 2 | 16位有符号整数 |
| **Uint16Array** | 2 | 16位无符号整数 |
| **Int32Array** | 4 | 32位有符号整数 |
| **Uint32Array** | 4 | 32位无符号整数 |
| **Float32Array** | 4 | 32位浮点数 |
| **Float64Array** | 8 | 64位浮点数 |

本质上，类型化数组和ArrayBuffer是一样的。不过一个可读写（脱掉buffer限制），一个当数据源的命。

举一些代码例子，看看本质一致在何处：

// 创建一个8字节的ArrayBuffer

var b = new ArrayBuffer(8);

// 创建一个指向b的视图v1，采用Int32类型，开始于默认的字节索引0，直到缓冲区的末尾

var v1 = new Int32Array(b);

// 创建一个指向b的视图v2，采用Uint8类型，开始于字节索引2，直到缓冲区的末尾

var v2 = new Uint8Array(b, 2);

// 创建一个指向b的视图v3，采用Int16类型，开始于字节索引2，长度为2

var v3 = new Int16Array(b, 2, 2);

上面代码里变量的数据结构如下表所示：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **变量** | **索引** | | | | | | | |
|  | 字节（不可索引） | | | | | | | |
| b= | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|  | 类型数组 | | | | | | | |
| v1= | 0 | | | | 1 | | | |
| v2= |  |  | 0 | 1 | 2 | 3 | 4 | 5 |
| v3= |  |  | 0 | | 1 | |  |  |

由于类型化数组直接访问固定内存，因此，速度很赞，比传统数组要快！因为普通Javascript数组使用的是Hash查找方式。同时，类型化数组天生处理二进制数据，这对于XMLHttpRequest 2、canvas、webGL等技术有着先天的优势。

**DataView对象**  
DataView对象在可以在ArrayBuffer中的任何位置读取和写入不同类型的二进制数据。

用法语法如下：

var dataView = new DataView(DataView(buffer, byteOffset[可选], byteLength[可选]);

其中，buffer表示ArrayBuffer；byteOffset指缓冲区开始处的偏移量（以字节为单位）；byteLength指缓冲区部分的长度（以字节为单位）。

**属性**

buffer

表示ArrayBuffer

byteOffset

指缓冲区开始处的偏移量

byteLength

指缓冲区部分的长度

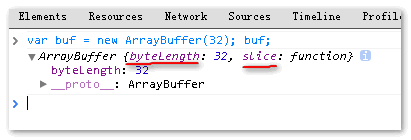
方法有很多，实际上，是有规律的，篇幅原因，也不是重点，就单纯露个脸：  
getInt8, getUint8, getInt16, getUint16, getInt32, getUint32, getFloat32, getFloat64, setInt8, setUint8, setInt16, setUint16, setInt32, setUint32, setFloat32, setFloat64.

下面回到**ArrayBuffer对象**，ArrayBuffer对象自身也可以构造，跟上面的FormData, Blob对象类似，例如：

var buf = new ArrayBuffer(32);

语法为：

ArrayBuffer ArrayBuffer(length[可以很大数值]);

我们在控制台运行下new ArrayBuffer(32)，看看结果：  


可以看到，其有一个byteLength属性，表示ArrayBuffer的长度，也可以说是大小；还有一个slice方法，语法如下：

ArrayBuffer slice(

begin

end[可选]

);

begin表示起始，end表示结束点。据说，Internet Explorer 10 以及iOS6-是没有该方法的。

综上，举个ArrayBuffer的实例吧，发送使用XMLhttpRequest发送ArrayBuffer数据：

function sendArrayBuffer() {

var xhr = new XMLHttpRequest();

xhr.open('POST', '/server', true);

xhr.onload = function(e) { ... };

var uInt8Array = new Uint8Array([1, 2, 3]);

xhr.send(uInt8Array.buffer);

}

使用了类型化数组，发送的是类型化数组(uInt8Array)的buffer属性，也就是ArrayBuffer对象。

over~

**八、结束语**

新技术层出不穷，我觉得吧，以后，行业的分支可能要更细了。比方说JS开发吧，可能就有JS UI交互开发工程师；JS Web开发工程师。因为，一个人想要完全hold住这么多的知识点，还真不是一般人能做到的。

刚开始写的时候，还想最后举个文件分割上传的例子，只可惜内容实在太多，加上去也会被湮没，于是作罢，决定有机会，专门讲下这个。还有FileReader可以独立讲一下，还有类型化数组也可以专门讲一下等。

学路漫漫，任重道远。文中若有致命的结论错误或疏忽的文字书写错误，都欢迎指正，不甚感谢。欢迎讨论，欢迎交流！

**参考链接**

* https://developer.mozilla.org/zh-CN/docs/DOM/DOMString
* https://developer.mozilla.org/en-US/docs/Web/API/document
* https://developer.mozilla.org/zh-CN/docs/DOM/XMLHttpRequest/FormData
* https://developer.mozilla.org/zh-CN/docs/DOM/Blob
* https://developer.mozilla.org/en-US/docs/Web/API/File
* https://developer.mozilla.org/en-US/docs/Web/API/FileReader
* http://technet.microsoft.com/zh-cn/ie/br212474
* https://developer.mozilla.org/en-US/docs/Web/API/ArrayBuffer
* http://blog.csdn.net/hfahe/article/details/7421203

原创文章，转载请注明来自[张鑫旭-鑫空间-鑫生活](http://www.zhangxinxu.com/)[[http://www.zhangxinxu.com](http://www.zhangxinxu.com/)]  
本文地址：<http://www.zhangxinxu.com/wordpress/?p=3725>

# [DataURL与File,Blob,canvas对象之间的互相转换的Javascript](http://blog.csdn.net/cuixiping/article/details/45932793)

分类： [JS/DHTML/CSS](http://blog.csdn.net/cuixiping/article/category/16769) 2015-05-27 01:14 3160人阅读 [评论](http://blog.csdn.net/cuixiping/article/details/45932793#comments)(1) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/cuixiping/article/details/45932793#report)

[blob](http://www.csdn.net/tag/blob)[file](http://www.csdn.net/tag/file)[canvas](http://www.csdn.net/tag/canvas)[dataurl](http://www.csdn.net/tag/dataurl)[javascript](http://www.csdn.net/tag/javascript)

目录[(?)[+]](http://blog.csdn.net/cuixiping/article/details/45932793)

# canvas转换为dataURL (从canvas获取dataURL)

var dataurl = canvas.toDataURL('image/png');

var dataurl2 = canvas.toDataURL('image/jpeg', 0.8);

* 1
* 2

# File对象转换为dataURL、Blob对象转换为dataURL

File对象也是一个Blob对象，二者的处理相同。

function readBlobAsDataURL(blob, callback) {

var a = new FileReader();

a.onload = function(e) {callback(e.target.result);};

a.readAsDataURL(blob);

}

//example:

readBlobAsDataURL(blob, function (dataurl){

console.log(dataurl);

});

readBlobAsDataURL(file, function (dataurl){

console.log(dataurl);

});

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12

# dataURL转换为Blob对象

function dataURLtoBlob(dataurl) {

var arr = dataurl.split(','), mime = arr[0].match(/:(.\*?);/)[1],

bstr = atob(arr[1]), n = bstr.length, u8arr = new Uint8Array(n);

while(n--){

u8arr[n] = bstr.charCodeAt(n);

}

return new Blob([u8arr], {type:mime});

}

//test:

var blob = dataURLtoBlob('data:text/plain;base64,YWFhYWFhYQ==');

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11

# dataURL图片数据绘制到canvas

先构造Image对象，src为dataURL，图片onload之后绘制到canvas

var img = new Image();

img.onload = function(){

canvas.drawImage(img);

};

img.src = dataurl;

* 1
* 2
* 3
* 4
* 5

# File,Blob的图片文件数据绘制到canvas

还是先转换成一个url，然后构造Image对象，src为dataURL，图片onload之后绘制到canvas

利用上面的 readBlobAsDataURL 函数，由File,Blob对象得到dataURL格式的url，再参考 **dataURL图片数据绘制到canvas**

readBlobAsDataURL(file, function (dataurl){

var img = new Image();

img.onload = function(){

canvas.drawImage(img);

};

img.src = dataurl;

});

* 1
* 2
* 3
* 4
* 5
* 6
* 7

不同的方法用于构造不同类型的url (分别是 dataURL, objectURL(blobURL), filesystemURL)。这里不一一介绍，仅以dataURL为例。

filesystemURL不是指本地文件URL的形式(file:///….), 而是格式类似于 filesystem:http://... 的一种URL，支持沙盒文件系统的浏览器支持(目前仅Chrome)支持。

# Canvas转换为Blob对象并使用Ajax发送

转换为Blob对象后，可以使用Ajax上传图像文件。

先从canvas获取dataurl, 再将dataurl转换为Blob对象

var dataurl = canvas.toDataURL('image/png');

var blob = dataURLtoBlob(dataurl);

//使用ajax发送

var fd = new FormData();

fd.append("image", blob, "image.png");

var xhr = new XMLHttpRequest();

xhr.open('POST', '/server', true);

xhr.send(fd);

你尚未登录，仅允许查看本站部分内容。请[登录](http://div.io/dlogin)或[使用邀请码注册](http://div.io/reg)。

[](http://div.io/user/barretlee)

# JavaScript多文件下载 3个回复 专栏 @ [Javascript](http://div.io/node/javascript)

[小胡子哥](http://div.io/user/barretlee) 发布于 11 月前

以前写过的一些[文章](http://www.barretlee.com/entry/)，转载于此，交流学习。  
作者：[小胡子哥 ( Barret Lee )](http://weibo.com/173248656)

对于文件的下载，可以说是一个十分常见的话题，前端的很多项目中都会有这样的需求，比如 highChart 统计图的导出，在线图片编辑中的图片保存，在线代码编辑的代码导出等等。而很多时候，我们只给了一个链接，用户需要右键点击链接，然后选择“另存为”，这 个过程虽说不麻烦，但还是需要两步操作，倘若用户想保存页面中的多个链接文件，就得重复操作很多次，最常见的就是英语听力网站上的音频下载，手都要点麻！

本文的目的是介绍如何利用 javascript 进行多文件的下载，也就是当用户点击某个链接或者按钮的时候，同时下载多个文件。这里的“同时”用的不是很准确，在现代浏览器中可以实现多文件的并行下 载，而在一些老版本浏览器，如IE8-，此类的浏览器就进行只能单个文件的下载，但是我们可以让多个文件依次保存下来，算是串行下载吧~

## 文件类型介绍及其特点

### 一般类型

平时比较常见的有 txt、png、jpg、zip、tar 等各种文件格式，这些文件格式中，一部分浏览器是会直接打开链接显示内容的，而另外一部分，浏览器不识别响应头，或者不能解析对应的格式，于是当做文件直接下载下来了。如：

<a href="http://barretlee.github.io/test.rar">file</a>

这句代码，若直接点开链接，浏览器将会直接下载该文件。

### dataURL类型

dataURL 也是十分常见的类型，他可以作为 src 或者 url() 的参数送进去。比较常见的有如下几种：

文本： data:text/plain;这里是正文内容。

图片： data:image/jpg;base64,/9j/4AAQSkZJRgABAQEA....

data:image/png;base64,/9j/4AAQSkZJRgABAQEA....

base64 是用的比较广泛的一种数据格式。

Base64格式

data:[][;charset=][;base64],

Base64 在CSS中的使用：

.demoImg{ background-image: url("data:image/jpg;base64,/9j/4QMZRXhpZgAASUkqAAgAAAAL...."); }

Base64 在HTML中的使用：

<img width="40" height="30" src="data:image/jpg;base64,/9j/4QMZRXhpZgAASUkqAAgAAAAL...." />

### Blob 流

Blob 对象表示不可变的、包含原始数据的类文件对象。具体的内容可以参阅[MDN文档](https://developer.mozilla.org/en-US/docs/Web/API/Blob)。

他的使用也是特别的方便，如：

var aFileParts = ['<a id="a"><b id="b">hey!</b></a>'];

var oMyBlob = new Blob(aFileParts, {type : 'text/html'}); // the blob

Blob 接收两个参数，一个是数组类型的数据对象，他可以是 ArrayBuffer、ArrayBufferView、Blob、String 等诸多类型；第二个参数是 MINE 类型设置。而本文我们要用到的是 URLcreateObjectURL() 这个函数，他的作用是将一个 URL 所代表的内容转化成一个 [DOMString](https://developer.mozilla.org/en-US/docs/Web/API/DOMString)，产生的结果是一个 文件对象 或者 Blob 对象。

### 二进制流

我们利用 File API 读取文件的时候，拿到的是数据的二进制流格式，这些类型可以直接被 ArrayBuffer 等接收，本文中没有用到，就不细说了。

## JavaScript 多文件下载

HTML5 中 a 标签多了一个属性——download，用户点击链接浏览器会打开并显示该链接的内容，若在链接中加了 download 属性，点击该链接不会打开这个文件，而是直接下载。虽说是比较好用，但低版本浏览器不兼容，这个在本届的 2 和 3 中将会讲到解决方案。

在这里，我们可以利用属性检测来判断浏览器类型：

h5Down = document.createElement("a").hasOwnProperty("download");

### a 标签 download 属性的使用

利用 download 属性可以直接下载单个文件，若想点击一次下载多个文件，就得稍加处理下了：

function downloadFile(fileName, content){

var aLink = document.createElement("a"),

evt = document.createEvent("HTMLEvents");

evt.initEvent("click");

aLink.download = fileName;

aLink.href = content;

aLink.dispatchEvent(evt);

}

download 属性的作用除了让浏览器忽略文件的 MIME 类型之外，还会把该属性的值作为文件名。你可以在 chrome 控制台运行这句程序：

downloadFile("barretlee.html", "./");

浏览器会提示是否保留（下载）该 html 文件。之前我们提到文件类型还可能是 dataURL 或者是 Blob 流，为了让程序也支持这些数据类型，稍微修改下上面的函数：

function downloadFile(fileName, content){

var aLink = document.createElement('a');

, blob = new Blob([content])

, evt = document.createEvent("HTMLEvents");

evt.initEvent("click");

aLink.download = fileName;

aLink.href = URL.createObjectURL(blob);

aLink.dispatchEvent(evt);

}

new Blob([content])，现将文件转换成一个 Blog 流，然后，使用 URL.createObjectURL() 将其转换成一个 DOMString。这样我们就支持 data64 和其他数据类型的 content 了~

### window.open 之后 execCommand("SaveAs")

上面也提到了，尽管 download 属性是十分便利的 H5 利器，但低版本 IE 根本不赏脸，要说方法，IE 还是有很多方式去转换的，比如 ADOBE.STREAM 的 activeX 对象可以把文件转换成文件流，然后写入到一个要保存的文件中。这里要谈到的是略微方便一点的方式：先把内容写到一个新开的 window 对象中，然后利用 execCommand 执行保存命令，就相当于我们在页面上按下 Ctrl+S，这样页面内的信息都会 down 下来。

// 将文件在一个 window 窗口中打开，并隐藏这个窗口。

var win = window.open("path/to/file.ext", "new Window", "width=0,height=0");

// 在 win 窗口中按下 ctrl+s 保存窗口内容

win.document.execCommand("SaveAs", true, "filename.ext");

// 使用完了，关闭窗口

win.close();

这个过程十分明了，不过这里会存在一个问题，并不是程序的问题，而是浏览器的问题，如果我们用 搜狗浏览器 或者 360浏览器 打开新窗口的话，他会新开一个标签页，而不是新开一个窗口，更可恶的时候部分浏览器不允许 window.open （这个可以设置）。所以只好另觅他法了。

### iframe 中操作

既然新开一个窗口那么麻烦，我就在本窗口下完成工作~

function IEdownloadFile(fileName, contentOrPath){

var ifr = document.createElement('iframe');

ifr.style.display = 'none';

ifr.src = contentOrPath;

document.body.appendChild(ifr);

// 保存页面 -> 保存文件

ifr.contentWindow.document.execCommand('SaveAs', false, fileName);

document.body.removeChild(ifr);

}

一般的链接我们可以直接给 iframe 添加 src 属性，然后执行 saveAs 命令，倘若我们使用的是 data64 编码的文件，这个怎么办？

var isImg = contentOrPath.slice(0, 10) === "data:image"；

// dataURL 的情况

isImg && ifr.contentWindow.document.write("<img src='" +

contentOrPath + "' />");

这个也比较好处理，直接把文件写入到 iframe 中，然后在执行保存。

## 代码的封装与接口介绍

### 代码的封装以及相关 DEMO

封装：[lib.js](https://github.com/barretlee/javascript-multiple-download/blob/master/lib.js)  
DEMO：[javascript-multiple-download](http://rawgithub.com/barretlee/javascript-multiple-download/master/test/test.html)

### 接口的调用

提供了三个接口，支持单文件下载，多文件下载，多文件下载自定义命名。

1）单文件下载

Downer("./file/test.txt");

2）多文件下载

Downer(["./file/test.txt","./file/test.txt"]);

3）多文件下载自定义命名

Downer({

"1.txt":"./file/test.txt",

"2.jpg":"./file/test.jpg"

});

文件的 URL 如 ./file/test.txt 都可以改成 base64 或者其他格式。

## 服务器支持与后端实现

### 后端实现

后端实现的原理，就是在响应头中加入一些特殊的标记，如前端发送这样的请求：

function download(path) {

var ifrm = document.getElementById(frame);

ifrm.src = "download.php?path="+path;

}

后端的响应为

<?php

header("Content-Type: application/octet-stream");

header("Content-Disposition: attachment; filename=".$\_GET['path']);

readfile($\_GET['path']);

?>

告诉浏览器这是一个流文件，作为附件方式发送给你，请忽略 MINE type，直接保存。

### 服务器配置

若后台是 apche 作为服务器，可以配置 htaccess 文件：

<filesmatch "\.(zip|rar)$"="">

Header set Content-Disposition attachment

</filesmatch>

意思是只要请求的是 zip 或者 rar 类型的文件，那么久添加一个 Content-Disposition:attachment 的响应头。这样就可以在 php 代码中省略麻烦的操作。

## 小结

由于行文仓促，文中会有不少错误，对多文件下载有更好的提议，希望提出来共同分享！

## 参考资料

* [在浏览器端用JS创建和下载文件](http://www.alloyteam.com/2014/01/use-js-file-download/) AlloyTeam
* [Starting file download with Javascript](http://thezedienblog.blogspot.com/2013/05/starting-file-download-with-javascript.html) Ahzaz's Blog
* [Blob 流](https://developer.mozilla.org/en-US/docs/Web/API/Blob) MDN

<!DOCTYPE HTML>  
<html>  
<head>  
</head>  
<body>  
<div id="aabb"></div>  
<script>

var URL = URL || window;  
  
function saveAs(blob, filename) {  
var type = blob.type;  
var force\_saveable\_type = 'application/octet-stream';  
if (type && type != force\_saveable\_type) {  
var slice = blob.slice || blob.webkitSlice || blob.mozSlice;  
blob = slice.call(blob, 0, blob.size, force\_saveable\_type);  
}  
  
var url = URL.createObjectURL(blob);  
console.log(url);  
var save\_link = document.createElementNS('http://www.w3.org/1999/xhtml', 'a');  
save\_link.href = url;  
save\_link.download = filename;  
save\_link.text = 'ffff';  
  
/\*  
// in firefox 33+  
var m = document.getElementById('aabb');  
m.innerHTML = '<a href="' +url+ '">FFF</a>';  
\*/

// in chrome 38+  
var event = document.createEvent('MouseEvents');  
event.initMouseEvent('click', true, false, window, 0, 0, 0, 0, 0, false, false, false, false, 0, null);  
save\_link.dispatchEvent(event);  
URL.revokeObjectURL(url);

}

function binaryToBlob(data) {  
var l = data.length, arr = new Uint8Array(l);  
  
for(var i = 0; i < l; i++) {  
arr[i] = data.charCodeAt(i);  
}  
  
return new Blob([arr], { type: 'image/octet-stream'})  
};

(function(){  
var r = new XMLHttpRequest();   
r.open("GET", 'a.tif', false);  
**r.overrideMimeType('text/plain; charset=x-user-defined');** // this line is very important ,the charset must be x-user-defined  
r.send();  
  
var blob = binaryToBlob(r.response);  
blob.fileType = "image/octet-stream";  
saveAs(blob, 'res.tif');

})();

</script></body></html>

# 用JavaScript读取和保存文件

A-A+

[炫意 HTML5](http://www.xyhtml5.com/author/nrain/)2011年10月24日[Javascript](http://www.xyhtml5.com/javascript/)[JavaScript](http://www.xyhtml5.com/tag/javascript-2/) | [拖拽](http://www.xyhtml5.com/tag/drag/)21 次浏览 [1](http://www.xyhtml5.com/read-and-save-files-using-javascript.html#comments)

话说今天只是粗略浏览了一下[Proxy SwitchySharp的源码](https://code.google.com/p/switchysharp/)，就收获了不少东西，其中就包括本文要介绍的读取和保存文件。  
  
因为Google还不提供同步插件数据的功能，所以导入和导出插件配置就必须和文件打交道了。而出于安全原因，只有IE才提供访问文件的API；但随着HTML 5的到来，其他浏览器也纷纷支持了。  
  
  
  
首先说读取文件。W3C提供了一些[File API](http://www.w3.org/TR/FileAPI/)，其中最重要的是[FileReader](https://developer.mozilla.org/en/DOM/FileReader)这个类。  
  
  
  
先列出需要用到的HTML标签：

<input type="file" id="file" onchange="handleFiles(this.files)"/>

当选择了一个文件时，就会把包含这个文件的列表（一个FileList对象）作为参数传给handleFiles()函数了。  
  
这个FileList对象类似一个数组，可以知道文件的数目，而它的元素就是File对象了。  
  
从这个File对象可以获取name、size、lastModifiedDate和type等属性。  
  
把这个File对象传给FileReader对象的读取方法，就能读取文件了。  
  
  
  
FileReader共有4种读取方法：

* readAsArrayBuffer(file)：将文件读取为[ArrayBuffer](https://developer.mozilla.org/en/JavaScript_typed_arrays/ArrayBuffer)。
* readAsBinaryString(file)：将文件读取为二进制字符串
* readAsDataURL(file)：将文件读取为[Data URL](http://www.keakon.net/2010/02/02/%E5%8E%9F%E6%9D%A5%E6%B5%8F%E8%A7%88%E5%99%A8%E8%BF%98%E6%94%AF%E6%8C%81dataURI%E5%8D%8F%E8%AE%AE)
* readAsText(file, [encoding])：将文件读取为文本，encoding缺省值为'UTF-8'

此外，abort()方法可以停止读取文件。  
  
  
  
FileReader对象在读取文件后，还需要进行处理。为了不阻塞当前线程，API采用了事件模型，可以注册这些事件：

* onabort：中断时触发
* onerror：出错时触发
* onload：文件成功读取完毕时触发
* onloadend：文件读取完毕时触发，无论是否失败
* onloadstart：文件开始读取时触发
* onprogress：当文件读取时，周期性地触发

有了这些方法以后，就可以处理文件了。  
  
先来试试读取文本文件：

function handleFiles(files) {

if (files.length) {

var file = files[0];

var reader = new FileReader();

if (/text/w+/.test(file.type)) {

reader.onload = function() {

$('<pre>' + this.result + '</pre>').appendTo('body');

}

reader.readAsText(file);

}

}

}

这里的this.result实际上就是reader.result，也就是读取出来的文件内容。  
  
测试一下你会发现这个文件的内容被添加到网页中了。如果是用Chrome的话，必须把网页放在服务器上或插件里，file协议下会失败。  
  
  
  
再来试试图片，因为浏览器可以直接显示Data URI协议的图片，所以这次就添加图片：

function handleFiles(files) {

if (files.length) {

var file = files[0];

var reader = new FileReader();

if (/text/w+/.test(file.type)) {

reader.onload = function() {

$('<pre>' + this.result + '</pre>').appendTo('body');

}

reader.readAsText(file);

} else if(/image/w+/.test(file.type)) {

reader.onload = function() {

$('<img src="' + this.result + '"/>').appendTo('body');

}

reader.readAsDataURL(file);

}

}

}

其实input:file控件还支持选择多个文件：

<input type="file" id="files" multiple="" onchange="handleFiles(this.files)"/>

这样handleFiles()里就需要遍历处理files了。  
  
  
  
如果只想读取部分数据的话，File对象还有webkitSlice()或mozSlice()方法，用于生成Blob对象。这个对象可以和File对象 一样被FileReader读取。这2个方法接收3个参数：第1个参数是起始位置；第2个是结束位置，省略时则读到文件结尾；第3个是content type。  
  
例子可以参考[《Reading local files in JavaScript》](http://www.html5rocks.com/en/tutorials/file/dndfiles/)。  
  
  
  
当然，除了导入数据和显示文件以外，它还可以用来做AJAX上传，代码可以参考[《Using files from web applications》](https://developer.mozilla.org/en/Using_files_from_web_applications)。  
  
  
  
  
  
接下来说保存文件。  
  
实际上[File API: Writer](http://www.w3.org/TR/file-writer-api/)提供了4个接口，但目前只有部分浏览器（Chrome 8+和Firefox 4+）实现了BlobBuilder，其余接口都不可用。  
  
对于不支持的浏览器，可以使用[BlobBuilder.js](https://github.com/eligrey/BlobBuilder.js)和[FileSaver.js](https://github.com/eligrey/FileSaver.js)来获得支持。  
  
我研究了一下，发现了其中的奥秘。  
  
  
  
BlobBuilder可以创建一个Blob对象。把这个Blob对象传递给[URL.createObjectURL()](https://developer.mozilla.org/en/DOM/window.URL.createObjectURL)方法，就可以拿到一个object URL。而这个object URL就是这个Blob对象的下载地址。  
  
拿到下载地址后，创建一个a元素，将下载地址赋值给href属性，文件名赋值给download属性（Chrome 14+支持）。  
  
然后再创建一个click事件，交给这个a元素处理，就会导致浏览器开始下载这个Blob对象了。  
  
最后，用[URL.revokeObjectURL()](https://developer.mozilla.org/en/DOM/window.URL.revokeObjectURL)来释放这个object URL，通知浏览器可以不必继续引用这个文件了。  
  
  
  
下面就是一段化简的代码：

var BlobBuilder = BlobBuilder || WebKitBlobBuilder || MozBlobBuilder;

var URL = URL || webkitURL || window;

function saveAs(blob, filename) {

var type = blob.type;

var force\_saveable\_type = 'application/octet-stream';

if (type && type != force\_saveable\_type) { // 强制下载，而非在浏览器中打开

var slice = blob.slice || blob.webkitSlice || blob.mozSlice;

blob = slice.call(blob, 0, blob.size, force\_saveable\_type);

}

var url = URL.createObjectURL(blob);

var save\_link = document.createElementNS('http://www.w3.org/1999/xhtml', 'a');

save\_link.href = url;

save\_link.download = filename;

var event = document.createEvent('MouseEvents');

event.initMouseEvent('click', true, false, window, 0, 0, 0, 0, 0, false, false, false, false, 0, null);

save\_link.dispatchEvent(event);

URL.revokeObjectURL(url);

}

var bb = new BlobBuilder;

bb.append('Hello, world!');

saveAs(bb.getBlob('text/plain;charset=utf-8'), 'hello world.txt');

测试时会提示保存一个文本文件。Chrome需要把网页放在服务器上或插件里。

## Introduction

HTML5 finally provides a standard way to interact with local files, via the [File API](http://www.w3.org/TR/file-upload/) specification. As example of its capabilities, the File API could be used to create a thumbnail preview of images as they're being sent to the server, or allow an app to save a file reference while the user is offline. Additionally, you could use client-side logic to verify an upload's mimetype matches its file extension or restrict the size of an upload.

The spec provides several interfaces for accessing files from a 'local' filesystem:

1. File - an individual file; provides readonly information such as name, file size, mimetype, and a reference to the file handle.
2. FileList - an array-like sequence of File objects. (Think <input type="file" multiple> or dragging a directory of files from the desktop).
3. Blob - Allows for slicing a file into byte ranges.

When used in conjunction with the above data structures, the [FileReader](http://dev.w3.org/2006/webapi/FileAPI/#filereader-interface) interface can be used to asynchronously read a file through familiar JavaScript event handling. Thus, it is possible to monitor the progress of a read, catch errors, and determine when a load is complete. In many ways the APIs resemble XMLHttpRequest's event model.

## Selecting files

The first thing to do is check that your browser fully supports the File API:

// Check for the various File API support.

if (window.File && window.FileReader && window.FileList && window.Blob) {

// Great success! All the File APIs are supported.

} else {

alert('The File APIs are not fully supported in this browser.');

}

Of course, if your app will only use a few of these APIs, modify this snippet accordingly.

### Using form input for selecting

The most straightforward way to load a file is to use a standard <input type="file"> element. JavaScript returns the list of selected File objects as a FileList. Here's an example that uses the 'multiple' attribute to allow selecting several files at once:

<input type="file" id="files" name="files[]" multiple />

<output id="list"></output>

<script>

function handleFileSelect(evt) {

var files = evt.target.files; // FileList object

// files is a FileList of File objects. List some properties.

var output = [];

for (var i = 0, f; f = files[i]; i++) {

output.push('<li><strong>', escape(f.name), '</strong> (', f.type || 'n/a', ') - ',

f.size, ' bytes, last modified: ',

f.lastModifiedDate ? f.lastModifiedDate.toLocaleDateString() : 'n/a',

'</li>');

}

document.getElementById('list').innerHTML = '<ul>' + output.join('') + '</ul>';

}

document.getElementById('files').addEventListener('change', handleFileSelect, false);

</script>

**Example**: Using form input for selecting. Try it!

### Using drag and drop for selecting

Another technique for loading files is native drag and drop from the desktop to the browser. We can modify the previous example slightly to include drag and drop support.

<div id="drop\_zone">Drop files here</div>

<output id="list"></output>

<script>

function handleFileSelect(evt) {

evt.stopPropagation();

evt.preventDefault();

var files = evt.dataTransfer.files; // FileList object.

// files is a FileList of File objects. List some properties.

var output = [];

for (var i = 0, f; f = files[i]; i++) {

output.push('<li><strong>', escape(f.name), '</strong> (', f.type || 'n/a', ') - ',

f.size, ' bytes, last modified: ',

f.lastModifiedDate ? f.lastModifiedDate.toLocaleDateString() : 'n/a',

'</li>');

}

document.getElementById('list').innerHTML = '<ul>' + output.join('') + '</ul>';

}

function handleDragOver(evt) {

evt.stopPropagation();

evt.preventDefault();

evt.dataTransfer.dropEffect = 'copy'; // Explicitly show this is a copy.

}

// Setup the dnd listeners.

var dropZone = document.getElementById('drop\_zone');

dropZone.addEventListener('dragover', handleDragOver, false);

dropZone.addEventListener('drop', handleFileSelect, false);

</script>

**Example**: Using drag and drop for selecting. Try it!

Drop files here

Note: Some browsers treat <input type="file"> elements as native drop targets. Try dragging files onto the input field in the previous example.

## Reading files

Now comes the fun part!

After you've obtained a File reference, instantiate a [FileReader](http://dev.w3.org/2006/webapi/FileAPI/#filereader-interface) object to read its contents into memory. When the load finishes, the reader's onload event is fired and its result attribute can be used to access the file data.

FileReader includes four options for reading a file, asynchronously:

* FileReader.readAsBinaryString(Blob|File) - The result property will contain the file/blob's data as a binary string. Every byte is represented by an integer in the range [0..255].
* FileReader.readAsText(Blob|File, opt\_encoding) - The result property will contain the file/blob's data as a text string. By default the string is decoded as 'UTF-8'. Use the optional encoding parameter can specify a different format.
* FileReader.readAsDataURL(Blob|File) - The result property will contain the file/blob's data encoded as a [data URL](http://en.wikipedia.org/wiki/Data_URI_scheme).
* FileReader.readAsArrayBuffer(Blob|File) - The result property will contain the file/blob's data as an [ArrayBuffer](https://www.khronos.org/registry/typedarray/specs/latest/#5) object.

Once one of these read methods is called on your FileReader object, the onloadstart, onprogress, onload, onabort, onerror, and onloadend can be used to track its progress.

The example below filters out images from the user's selection, calls reader.readAsDataURL() on the file, and renders a thumbnail by setting the 'src' attribute to a data URL.

<style>

.thumb {

height: 75px;

border: 1px solid #000;

margin: 10px 5px 0 0;

}

</style>

<input type="file" id="files" name="files[]" multiple />

<output id="list"></output>

<script>

function handleFileSelect(evt) {

var files = evt.target.files; // FileList object

// Loop through the FileList and render image files as thumbnails.

for (var i = 0, f; f = files[i]; i++) {

// Only process image files.

if (!f.type.match('image.\*')) {

continue;

}

var reader = new FileReader();

// Closure to capture the file information.

reader.onload = (function(theFile) {

return function(e) {

// Render thumbnail.

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

span.innerHTML = ['<img class="thumb" src="', e.target.result,

'" title="', escape(theFile.name), '"/>'].join('');

document.getElementById('list').insertBefore(span, null);

};

})(f);

// Read in the image file as a data URL.

reader.readAsDataURL(f);

}

}

document.getElementById('files').addEventListener('change', handleFileSelect, false);

</script>

**Example**: Reading files. Try it!

Try this example with a directory of images!

### Slicing a file

In some cases reading the entire file into memory isn't the best option. For example, say you wanted to write an async file uploader. One possible way to speed up the upload would be to read and send the file in separate byte range chunks. The server component would then be responsible for reconstructing the file content in the correct order.

Lucky for us, the File interface supports a slice method to support this use case. The method takes a starting byte as its first argument, ending byte as its second, and an option content type string as a third.

var blob = file.slice(*startingByte*, *endindByte*);

reader.readAsBinaryString(blob);

The following example demonstrates reading chunks of a file. Something worth noting is that it uses the onloadend and checks the evt.target.readyState instead of using the onload event.

<style>

#byte\_content {

margin: 5px 0;

max-height: 100px;

overflow-y: auto;

overflow-x: hidden;

}

#byte\_range { margin-top: 5px; }

</style>

<input type="file" id="files" name="file" /> Read bytes:

<span class="readBytesButtons">

<button data-startbyte="0" data-endbyte="4">1-5</button>

<button data-startbyte="5" data-endbyte="14">6-15</button>

<button data-startbyte="6" data-endbyte="7">7-8</button>

<button>entire file</button>

</span>

<div id="byte\_range"></div>

<div id="byte\_content"></div>

<script>

function readBlob(opt\_startByte, opt\_stopByte) {

var files = document.getElementById('files').files;

if (!files.length) {

alert('Please select a file!');

return;

}

var file = files[0];

var start = parseInt(opt\_startByte) || 0;

var stop = parseInt(opt\_stopByte) || file.size - 1;

var reader = new FileReader();

// If we use onloadend, we need to check the readyState.

reader.onloadend = function(evt) {

if (evt.target.readyState == FileReader.DONE) { // DONE == 2

document.getElementById('byte\_content').textContent = evt.target.result;

document.getElementById('byte\_range').textContent =

['Read bytes: ', start + 1, ' - ', stop + 1,

' of ', file.size, ' byte file'].join('');

}

};

var blob = file.slice(start, stop + 1);

reader.readAsBinaryString(blob);

}

document.querySelector('.readBytesButtons').addEventListener('click', function(evt) {

if (evt.target.tagName.toLowerCase() == 'button') {

var startByte = evt.target.getAttribute('data-startbyte');

var endByte = evt.target.getAttribute('data-endbyte');

readBlob(startByte, endByte);

}

}, false);

</script>

**Example**: Slicing a file. Try it!

Read bytes:

### Monitoring the progress of a read

One of the nice things that we get for free when using async event handling is the ability to monitor the progress of the file read; useful for large files, catching errors, and figuring out when a read is complete.

The onloadstart and onprogress events can be used to monitor the progress of a read.

The example below demonstrates displaying a progress bar to monitor the status of a read. To see the progress indicator in action, try a large file or one from a remote drive.

<style>

#progress\_bar {

margin: 10px 0;

padding: 3px;

border: 1px solid #000;

font-size: 14px;

clear: both;

opacity: 0;

-moz-transition: opacity 1s linear;

-o-transition: opacity 1s linear;

-webkit-transition: opacity 1s linear;

}

#progress\_bar.loading {

opacity: 1.0;

}

#progress\_bar .percent {

background-color: #99ccff;

height: auto;

width: 0;

}

</style>

<input type="file" id="files" name="file" />

<button onclick="abortRead();">Cancel read</button>

<div id="progress\_bar"><div class="percent">0%</div></div>

<script>

var reader;

var progress = document.querySelector('.percent');

function abortRead() {

reader.abort();

}

function errorHandler(evt) {

switch(evt.target.error.code) {

case evt.target.error.NOT\_FOUND\_ERR:

alert('File Not Found!');

break;

case evt.target.error.NOT\_READABLE\_ERR:

alert('File is not readable');

break;

case evt.target.error.ABORT\_ERR:

break; // noop

default:

alert('An error occurred reading this file.');

};

}

function updateProgress(evt) {

// evt is an ProgressEvent.

if (evt.lengthComputable) {

var percentLoaded = Math.round((evt.loaded / evt.total) \* 100);

// Increase the progress bar length.

if (percentLoaded < 100) {

progress.style.width = percentLoaded + '%';

progress.textContent = percentLoaded + '%';

}

}

}

function handleFileSelect(evt) {

// Reset progress indicator on new file selection.

progress.style.width = '0%';

progress.textContent = '0%';

reader = new FileReader();

reader.onerror = errorHandler;

reader.onprogress = updateProgress;

reader.onabort = function(e) {

alert('File read cancelled');

};

reader.onloadstart = function(e) {

document.getElementById('progress\_bar').className = 'loading';

};

reader.onload = function(e) {

// Ensure that the progress bar displays 100% at the end.

progress.style.width = '100%';

progress.textContent = '100%';

setTimeout("document.getElementById('progress\_bar').className='';", 2000);

}

// Read in the image file as a binary string.

reader.readAsBinaryString(evt.target.files[0]);

}

document.getElementById('files').addEventListener('change', handleFileSelect, false);

</script>

**Example**: Monitoring the progress of a read. Try it!

0%

**Tip**: To really see this progress indicator in action, try a large file or a resource on a remote drive.

## References

* [File](http://www.w3.org/TR/file-upload/) API specification
* [FileReader](http://www.w3.org/TR/file-upload/#dfn-filereader) interface specification
* [Blob](http://www.w3.org/TR/file-upload/#dfn-Blob) interface specification
* [FileError](http://www.w3.org/TR/file-upload/#dfn-fileerror) interface specification
* [ProgressEvent](http://www.w3.org/TR/progress-events/#Progress) specification