历史上，JavaScript无法处理二进制数据。如果一定要处理的话，只能使用charCodeAt()方法，一个个字节地从文字编码转成二进制数据，还有一种办法是将二进制数据转成Base64编码，再进行处理。这两种方法不仅速度慢，而且容易出错。ECMAScript5引入了Blob对象，允许直接操作二进制数据。

Bolb对象是一个代表二进制数据的基本对象，在它的基础上，又衍生出一系列相关的API，用来操作文件。

* File对象：负责处理那些以文件形式存在的二进制数据，也就是操作本地文件
* FileList对象：File对象的网页表单接口
* FileReader对象：负责将二进制数据读入内存
* URL对象：用于对二进制数据生成URL

**1、Blob对象**

Blob（Binary Large Object）对象代表了一段二进制数据，提供了一系列操作接口。其他操作二进制数据的API（比如File对象），都是建立在Blob对象基础上的，继承了它的属性和方法。

生成Blob对象有两种方法：一种是使用Blob构造函数，另一种是对现有的Blob对象使用slice方法切出一部分。

（1）Blob构造函数，接受两个参数。第一个参数是一个包含实际数据的数组，第二个参数是数据的类型，这两个参数都不是必需的。

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

var myBlob = new Blob(htmlParts, {'type': 'text\/xml'});

下面是一个利用Blob对象，生成可下载文件的例子。

var blob = new Blob(['Hello World']);

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

a.href = window.URL.createObjectURL(blob);

a.donwload = 'hello-world.txt';

a.textContent = 'Download Hello World';

body.appendChild(a);

 上面的代码生成了一个超级链接，点击后提示下载文本文件hello-world.txt，文件内容为"Hello World"。

（2） Blob对象的slice方法，将二进制数据按照字节分块，返回一个新的Blob对象。

var newBlob = oldBlob.slice(startingByte, endindByte);

下面是一个使用XMLHttpRequest对象，将大文件分割上传的例子。

function upload(blobOrFile) {

var xhr = new XMLHttpRequest();

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

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

xhr.send(blobOrFile);

}

document.querySelector('input[type="file"]').addEventListener('change', function(e) {

var blob = this.files[0];

var BYTES\_PER\_CHUNK = 1024 \* 1024; // 1MB chunk sizes.

var SIZE = blob.size;

var start = 0;

var end = BYTES\_PER\_CHUNK;

while(start < SIZE) {

upload(blob.slice(start, end));

start = end;

end = start + BYTES\_PER\_CHUNK;

}

}, false);

（3）Blob对象有两个只读属性：

* size：二进制数据的大小，单位为字节。
* type：二进制数据的MIME类型，全部为小写，如果类型未知，则该值为空字符串。

在Ajax操作中，如果xhr.responseType设为blob，接收的就是二进制数据。

**2、FileList对象**

FileList对象针对表单的File控件。当用户通过file控件选取文件后，这个控件的files属性值就是FileList对象。它在结构上类似于数组，包含用户选取的多个文件。

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

当用户选取文件后，就可以读取该文件。

var selected\_file = document.getElementById('input').files[0];

采用拖放方式，也可以得到FileList对象。

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

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

function handleFileSelect(evt) {

evt.stopPropagation();

evt.preventDefault();

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

// ...

}

上面代码的 handleFileSelect 是拖放事件的回调函数，它的参数evt是一个事件对象，该参数的dataTransfer.files属性就是一个FileList对象，里面包含了拖放的文件。

**3、File对象**

File对象是FileList对象的成员，包含了文件的一些元信息，比如文件名、上次改动时间、文件大小和文件类型。它的属性值如下：

* name：文件名，该属性只读
* size：文件大小，单位为字节，该属性只读
* type：文件的MIME类型，如果分辨不出类型，则为空字符串，该属性只读。
* lastModifiedDate：文件的上次修改时间，该属性只读。

var selected\_file = document.getElementById('input').files[0];

var fileName = selected\_file.name;

var fileSize = selected\_file.size;

var fileType = selected\_file.type;

**4、FileReader 对象**

FileReader对象接收File对象或Blob对象作为参数，用于读取文件的实际内容，即把文件内容读入内存。对于不同类型的文件，FileReader使用不同的方法读取。

* FileReader.readAsBinaryString(Blob|File) ：读取结果为二进制字符串，每个字节包含一个0到255之间的整数。
* FileReader.readAsText(Blob|File, opt\_encoding) ：读取结果是一个文本字符串。默认情况下，文本编码格式是'UTF-8'，可以通过可选的格式参数，指定其他编码格式的文本。
* FileReader.readAsDataURL(Blob|File) ： 读取结果是一个基于Base64编码的 data-uri 对象。
* FileReader.readAsArrayBuffer(Blob|File) ：读取结果是一个 ArrayBuffer 对象。

FileReader采用异步方式读取文件，可以为一系列事件指定回调函数。

* onabort：读取中断或调用reader.abort()方法时触发。
* onerror：读取出错时触发。
* onload：读取成功后触发。
* onloadend：读取完成后触发，不管是否成功。触发顺序排在 onload 或 onerror 后面。
* onloadstart：读取将要开始时触发。
* onprogress：读取过程中周期性触发。

下面的代码是如何展示文本文件的内容。

var reader = new FileReader();

reader.onload = function(e){

console.log(e.target.result);

}

reader.readAsText(blob);

onload事件的回调函数接受一个事件对象，该对象的target.result就是文件的内容。

下面是一个使用readAsDataURL方法，为img元素添加src属性的例子。

var reader = new FileReader();

reader.onload = function(e) {

document.createElement('img').src = e.target.result;

};

reader.readAsDataURL(f);

下面是一个onerror事件回调函数的例子。

var reader = new FileReader();

reader.onerror = errorHandler;

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;

default:

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

};

}

下面是一个onprogress事件回调函数的例子，主要用来显示读取进度。

var reader = new FileReader();

reader.onprogress = updateProgress;

function updateProgress(evt) {

if (evt.lengthComputable) {

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

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

if (percentLoaded < 100) {

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

progress.textContent = percentLoaded + '%';

}

}

}

读取大文件的时候，可以利用Blob对象的slice方法，将大文件分成小段，逐一读取，这样可以加快处理速度。

**5、URL对象**

URL对象用于生成指向File对象或Blob对象的URL。

var objecturl = window.URL.createObjectURL(blob);

上面的代码会对二进制数据生成一个URL，类似于“blob:http%3A//test.com/666e6730-f45c-47c1-8012-ccc706f17191”。这个URL可以放置于任何通常可以放置URL的地方，比如img标签的src属性。需要注意的是，即使是同样的二进制数据，每调用一次URL.createObjectURL方法，就会得到一个不一样的URL。

这个URL的存在时间，等同于网页的存在时间，一旦网页刷新或卸载，这个URL就失效。除此之外，也可以手动调用URL.revokeObjectURL方法，使URL失效。

window.URL.revokeObjectURL(objectURL);

下面是一个利用URL对象，在网页插入图片的例子。

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

img.src = window.URL.createObjectURL(files[0]);

img.height = 60;

img.onload = function(e) {

window.URL.revokeObjectURL(this.src);

}

body.appendChild(img);

var info = document.createElement("span");

info.innerHTML = files[i].name + ": " + files[i].size + " bytes";

body.appendChild(info);

还有一个本机视频预览的例子。

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

var obj\_url = window.URL.createObjectURL(blob);

video.src = obj\_url;

video.play()

window.URL.revokeObjectURL(obj\_url);

**6、参考链接**

[1] [W3C Working Draft](http://www.w3.org/TR/FileAPI/)

[2] Andrew Dodson, [Get Loaded with the File API](http://msdn.microsoft.com/en-gb/magazine/jj835793.aspx)

[3] Mozilla Developer Network, [Using files from web applications](https://developer.mozilla.org/en-US/docs/Using_files_from_web_applications)

[4] [HTML5 download attribute](http://javascript-reverse.tumblr.com/post/37056936789/html5-download-attribute)

[5] Eric Bidelman, [Reading files in JavaScript using the File APIS](http://www.html5rocks.com/en/tutorials/file/dndfiles/)

注：本文转自<http://javascript.ruanyifeng.com/htmlapi/file.html>，有修改。