

ArrayBuffer 对象，Blob 对象

ArrayBuffer 对象

ArrayBuffer 对象表示一段二进制数据，用来模拟内存里面的数据。通过这个对象，JavaScript 可以读写二进制数据。这个对象可以看作内存数据的表达。

这个对象是 ES6 才写入标准的，普通的网页编程用不到它，为了教程体系的完整，下面只提供一个简略的介绍，详细介绍请看《ES6 标准入门》里面的章节。

浏览器原生提供`ArrayBuffer()`构造函数，用来生成实例。它接受一个整数作为参数，表示这段二进制数据占用多少个字节。

```
```javascript
var buffer = new ArrayBuffer(8);
```
```

上面代码中，实例对象`buffer`占用8个字节。

ArrayBuffer 对象有实例属性`byteLength`，表示当前实例占用的内存长度（单位字节）。

```
```javascript
var buffer = new ArrayBuffer(8);
buffer.byteLength // 8
```
```

ArrayBuffer 对象有实例方法`slice()`，用来复制一部分内存。它接受两个整数参数，分别表示复制的开始位置（从0开始）和结束位置（复制时不包括结束位置），如果省略第二个参数，则表示一直复制到结束。

```
```javascript
var buf1 = new ArrayBuffer(8);
var buf2 = buf1.slice(0);
```
```

上面代码表示复制原来的实例。

Blob 对象

简介

Blob 对象表示一个二进制文件的数据内容，比如一个图片文件的内容就可以通过 Blob 对象读写。它通常用来读写文件，它的名字是 Binary Large Object（二进制大型对象）的缩写。它与 ArrayBuffer 的区别在于，它用于操作二进制文件，而 ArrayBuffer 用于操作内存。

浏览器原生提供`Blob()`构造函数，用来生成实例对象。

```
```javascript
new Blob(array [, options])
```
```

`Blob`构造函数接受两个参数。第一个参数是数组，成员是字符串或二进制对象，表示新生成的`Blob`实例对象的内容；第二个参数是可选的，是一个配置对象，目前只有一个属性`type`，它的值是一个字符串，表示数据的 MIME 类型，默认是空字符串。

```
```javascript
var htmlFragment = ['<b id="b">hey!'];
var myBlob = new Blob(htmlFragment, {type : 'text/html'});
```
```

上面代码中，实例对象`myBlob`包含的是字符串。生成实例的时候，数据类型指定为`text/html`。

下面是另一个例子，Blob 保存 JSON 数据。

```
```javascript
var obj = { hello: 'world' };
var blob = new Blob([JSON.stringify(obj)], {type : 'application/json'});
```
```

实例属性和实例方法

`Blob`具有两个实例属性`size`和`type`，分别返回数据的大小和类型。

```
```javascript
var htmlFragment = ['<b id="b">hey!'];
var myBlob = new Blob(htmlFragment, {type : 'text/html'});

myBlob.size // 32
myBlob.type // "text/html"
```
```

`Blob`具有一个实例方法`slice`，用来拷贝原来的数据，返回的也是一个`Blob`实例。

```
```javascript
myBlob.slice(start, end, contentType)
```
```

`slice`方法有三个参数，都是可选的。它们依次是起始的字节位置（默认为0）、结束的字节位置（默认为`size`属性的值，该位置本身将不包含在拷贝的数据之中）、新实例的数据类型（默认为空字符串）。

获取文件信息

文件选择器`<input type="file">`用来让用户选取文件。出于安全考虑，浏览器不允许脚本自行设置这个控件的`value`属性，即文件必须是用户手动选取的，不能是脚本指定的。一旦用户选好了文件，脚本就可以读取这个文件。

文件选择器返回一个 `FileList` 对象，该对象是一个类似数组的成员，每个成员都是一个 `File` 实例对象。`File` 实例对象是一个特殊的 `Blob` 实例，增加了`name`和`lastModifiedDate`属性。

```
```javascript
// HTML 代码如下
// <input type="file" accept="image/*" multiple onchange="fileinfo(this.files)"/>

function fileinfo(files) {
 for (var i = 0; i < files.length; i++) {
 var f = files[i];
 console.log(
 f.name, // 文件名，不含路径
 f.size, // 文件大小，Blob 实例属性
 f.type, // 文件类型，Blob 实例属性
 f.lastModifiedDate // 文件的最后修改时间
);
 }
}
```
```

除了文件选择器，拖放 API 的`dataTransfer.files`返回的也是一个`FileList`对象，它的成员因此也是 `File` 实例对象。

下载文件

AJAX 请求时，如果指定`responseType`属性为`blob`，下载下来的就是一个 `Blob` 对象。

```
```javascript
function getBlob(url, callback) {
 var xhr = new XMLHttpRequest();
 xhr.open('GET', url);
 xhr.responseType = 'blob';
 xhr.onload = function () {
 callback(xhr.response);
 }
 xhr.send(null);
}
```
```

上面代码中，`xhr.response`拿到的就是一个 `Blob` 对象。

生成 URL

浏览器允许使用`URL.createObjectURL()`方法，针对 Blob 对象生成一个临时 URL，以便于某些 API 使用。这个 URL 以`blob://`开头，表明对应一个 Blob 对象，协议头后面是一个识别符，用来唯一对应内存里面的 Blob 对象。这一点与`data://URL`（URL 包含实际数据）和`file://URL`（本地文件系统里面的文件）都不一样。

```
```javascript
var droptarget = document.getElementById('droptarget');

droptarget.ondrop = function (e) {
 var files = e.dataTransfer.files;
 for (var i = 0; i < files.length; i++) {
 var type = files[i].type;
 if (type.substring(0,6) !== 'image/')
 continue;
 var img = document.createElement('img');
 img.src = URL.createObjectURL(files[i]);
 img.onload = function () {
 this.width = 100;
 document.body.appendChild(this);
 URL.revokeObjectURL(this.src);
 }
 }
}
```
```

上面代码通过为拖放的照片文件生成一个 URL，产生它们的缩略图，从而使得用户可以预览选择的文件。

浏览器处理 Blob URL 就跟普通的 URL 一样，如果 Blob 对象不存在，返回404状态码；如果跨域请求，返回403状态码。Blob URL 只对 GET 请求有效，如果请求成功，返回200状态码。由于 Blob URL 就是普通 URL，因此可以下载。

读取文件

取得 Blob 对象以后，可以通过`FileReader`对象，读取 Blob 对象的内容，即文件内容。

FileReader 对象提供四个方法，处理 Blob 对象。Blob 对象作为参数传入这些方法，然后以指定的格式返回。

- `FileReader.readAsText()`：返回文本，需要指定文本编码，默认为 UTF-8。
- `FileReader.readAsArrayBuffer()`：返回 ArrayBuffer 对象。
- `FileReader.readAsDataURL()`：返回 Data URL。
- `FileReader.readAsBinaryString()`：返回原始的二进制字符串。

下面是`FileReader.readAsText()`方法的例子，用来读取文本文件。

```
```javascript
```

```
// HTML 代码如下
// <input type='file' onchange='readfile(this.files[0])'></input>
// <pre id='output'></pre>
function readfile(f) {
 var reader = new FileReader();
 reader.readAsText(f);
 reader.onload = function () {
 var text = reader.result;
 var out = document.getElementById('output');
 out.innerHTML = '';
 out.appendChild(document.createTextNode(text));
 }
 reader.onerror = function(e) {
 console.log('Error', e);
 };
}

```

上面代码中，通过指定 FileReader 实例对象的`onload`监听函数，在实例的`result`属性上拿到文件内容。

下面是`FileReader.readAsArrayBuffer()`方法的例子，用于读取二进制文件。

```
```javascript
// HTML 代码如下
// <input type="file" onchange="typefile(this.files[0])"></input>
function typefile(file) {
    // 文件开头的四个字节，生成一个 Blob 对象
    var slice = file.slice(0, 4);
    var reader = new FileReader();
    // 读取这四个字节
    reader.readAsArrayBuffer(slice);
    reader.onload = function (e) {
        var buffer = reader.result;
        // 将这四个字节的内容，视作一个32位整数
        var view = new DataView(buffer);
        var magic = view.getUint32(0, false);
        // 根据文件的前四个字节，判断它的类型
        switch(magic) {
            case 0x89504E47: file.verified_type = 'image/png'; break;
            case 0x47494638: file.verified_type = 'image/gif'; break;
            case 0x25504446: file.verified_type = 'application/pdf'; break;
            case 0x504b0304: file.verified_type = 'application/zip'; break;
        }
        console.log(file.name, file.verified_type);
    };
}
```

```