File 对象，FileList 对象和 FileReader 对象

1. File 对象

`File` 对象代表一个文件，用来读写文件信息。它继承了 `Blob` 对象，或者说是一种特殊的 `Blob` 对象，所有可以使用 `Blob` 对象的场合都可以使用它。

最常见的使用场合是表单的文件上传控件（`<input type="file">`），用户选中文件以后，浏览器就会生成一个数组，里面是每一个用户选中的文件，它们都是 `File` 实例对象。

```
// <input id="fileItem" type="file">
let file = document.getElementById('fileItem').files[0];
file instanceof File; // true
```

上面代码中，`file` 是用户选中的第一个文件，它是 `File` 的实例。

1.1. 构造函数

浏览器原生提供一个 `File()` 构造函数，用来生成 `File` 实例对象。

```
new File(array, name [, options])
```

`File()` 构造函数接受三个参数。

- `array`：一个数组，成员可以是二进制对象或字符串，表示文件的内容。
- `name`：字符串，表示文件名或文件路径。
- `options`：配置对象，设置实例的属性。该参数可选。

第三个参数配置对象，可以设置两个属性。

- `type`：字符串，表示实例对象的 `MIME` 类型，默认值为空字符串。
- `lastModified`：时间戳，表示上次修改的时间，默认为 `Date.now()`。

下面是一个例子。

```
let file = new File(['foo'], 'foo.txt', {type: 'text/plain'});
```

1.2. File 对象实例属性

- `File.lastModified`：最后修改时间
- `File.name`：文件名或文件路径
- `File.size`：文件大小（单位字节）
- `File.type`：文件的 MIME 类型

```
let myFile = new File([], 'file.bin', {
  lastModified: new Date(2018, 1, 1),
});
myFile.lastModified; // 1517414400000
myFile.name; // "file.bin"
myFile.size; // 0
myFile.type; // ""
```

上面代码中，由于 myFile 的内容为空，也没有设置 MIME 类型，所以 size 属性等于 0，type 属性等于空字符串。

File 对象没有自己的实例方法，由于继承了 Blob 对象，因此可以使用 Blob 的实例方法 slice()。

2. FileList 对象

FileList 对象是一个类似数组的对象，代表一组选中的文件，每个成员都是一个 File 实例。它主要出现在两个场合。

- 文件控件节点（<input type="file">）的 files 属性，返回一个 FileList 实例。
- 拖拉一组文件时，目标区的 DataTransfer.files 属性，返回一个 FileList 实例。

```
// <input id="fileItem" type="file">
let files = document.getElementById('fileItem').files;
files instanceof FileList; // true
```

上面代码中，文件控件的 files 属性是一个 FileList 实例。

FileList 的实例属性主要是 length，表示包含多少个文件。

FileList 的实例方法主要是 item()，用来返回指定位置的实例。它接受一个整数作为参数，表示位置的序号（从零开始）。但是，由于 FileList 的实例是一个类似数组的对象，可以直接用方括号运算符，即 myFileList[0] 等同于 myFileList.item(0)，所以一般用不到 item() 方法。

3. FileReader 对象

FileReader 对象用于读取 File 对象或 Blob 对象所包含的文件内容。

浏览器原生提供一个 FileReader 构造函数，用来生成 FileReader 实例。

```
let reader = new FileReader();
```

3.1. FileReader 实例属性

- FileReader.error：读取文件时产生的错误对象
- FileReader.readyState：整数，表示读取文件时的当前状态。一共有三种可能的状态，0 表示尚未加载任何数据，1 表示数据正在加载，2 表示加载完成。

- `FileReader.result`：读取完成后的文件内容，有可能是字符串，也可能是一个 `ArrayBuffer` 实例。
- `FileReader.onabort`：`abort` 事件（用户终止读取操作）的监听函数。
- `FileReader.onerror`：`error` 事件（读取错误）的监听函数。
- `FileReader.onload`：`load` 事件（读取操作完成）的监听函数，通常在这个函数里面使用 `result` 属性，拿到文件内容。
- `FileReader.onloadend`：`loadend` 事件（读取操作结束）的监听函数。
- `FileReader.onprogress`：`progress` 事件（读取操作进行中）的监听函数。

下面是监听 `load` 事件的一个例子。

```
// <input type="file" onchange="onChange(event)">

function onChange(event) {
  let file = event.target.files[0];
  let reader = new FileReader();
  reader.onload = function (event) {
    console.log(event.target.result);
  };

  reader.readAsText(file);
}
```

上面代码中，每当文件控件发生变化，就尝试读取第一个文件。如果读取成功（load 事件发生），就打印出文件内容。

3.2. FileReader 实例方法

- `FileReader.abort()`：终止读取操作，`readyState` 属性将变成 2。
- `FileReader.readAsArrayBuffer()`：以 `ArrayBuffer` 的格式读取文件，读取完成后 `result` 属性将返回一个 `ArrayBuffer` 实例。
- `FileReader.readAsBinaryString()`：读取完成后，`result` 属性将返回原始的二进制字符串。
- `FileReader.readAsDataURL()`：读取完成后，`result` 属性将返回一个 `Data URL` 格式（`Base64` 编码）的字符串，代表文件内容。对于图片文件，这个字符串可以用于 `<img>` 元素的 `src` 属性。**这个字符串不能直接进行 Base64 解码，必须把前缀 data:/;base64,从字符串里删除以后，再进行解码。**
- `FileReader.readAsText()`：读取完成后，`result` 属性将返回文件内容的文本字符串。该方法的第一个参数是代表文件的 `Blob` 实例，第二个参数是可选的，表示文本编码，默认为 `UTF-8`。

下面是一个例子。

```
// <input type="file" onchange="previewFile()">
// <img src="" height="200">

function previewFile() {
  let preview = document.querySelector('img');
  let file = document.querySelector('input[type=file]').files[0];
  let reader = new FileReader();

  reader.addEventListener(
```

```
      'load',
      function () {
        preview.src = reader.result;
      },
      false
    );

    if (file) {
      reader.readAsDataURL(file);
    }
  }
```

上面代码中，用户选中图片文件以后，脚本会自动读取文件内容，然后作为一个 `Data URL` 赋值给 `<img>` 元素的 `src` 属性，从而把图片展示出来。