

第 5 章

HTML5中的文件



视频讲解

本章学习目标

- 掌握选择上传文件的使用方法。
- 了解并理解文件读取与拖放的过程。
- 熟悉文件读取时的错误与异常的处理方法。

5.1 选择文件

与 HTML4 前版本相同,在 HTML5 中,可以创建一个 file 类型的 input 元素实现文件的上传功能,只是在 HTML5 中,该类型的 input 元素新添加了一个 multiple 属性,如果将属性的值设置为 true,那么可以在一个元素中实现多个文件的上传。另外,通过访问 Blob 对象,可以获取上传文件的类型、大小属性。下面分别进行详细的介绍。

5.1.1 选择单个文件

在 HTML5 中,创建一个 file 类型的 input 元素上传文件时,该元素在页面中的展示方式发生了变化,不再有文本框,而是一个选择文件的按钮,按钮的右边显示选择上传文件的名称,初始化页面时,没有上传文件,因此,显示“未选择文件”字样。

下面通过一个简单的实例介绍如何选择单个文件并上传的过程。

实例 5-1 选择单个文件上传

1. 功能描述

在页面中,创建一个 file 类型的 input 元素,单击该元素的“选择文件”按钮,选择一个图片文件,单击“打开”按钮或双击该文件后,在页面中“选择文件”按钮的右侧将显示所选择图片文件的名称,表明已将该文件选中,等待上传。

2. 实现代码

在 WebStorm 中新建一个 HTML 页面 5-1.html, 加入代码如代码清单 5-1 所示。

代码清单 5-1 选择单个文件上传

```
<!DOCTYPE html>
<html>
<head>
<meta charset = "UTF - 8">
<title>选择单个文件上传</title>
<link href = "Css/css5.css" rel = "stylesheet" type = "text/css">
</head>
<body>
<form id = "frmTmp">
<fieldset>
  <legend>上传单个文件: </legend>
  <input type = "file" name = "fileUpload" id = "fileUpload" />
</fieldset>
</form>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 浏览器中执行的页面效果如图 5-1 所示。



图 5-1 使用 file 类型的 input 元素实现单个文件选择时的效果

4. 源码分析

在本实例中, 单击“选择文件”按钮, 选中上传文件后, 没有编写任何的 JavaScript 代码, 在页面中显示所选择的文件名称。当然, 如果该名称太长, 在页面显示时, 将采取两端显示, 中间省略号的形式展示, 例如, 图片名称为“jQuery 权威指南_2010 年作品”, 那么页面显示为“jQuery 权威...10 年作品.jpg”。

5.1.2 选择多个文件

在 HTML5 中, 除了使用 file 类型的 input 元素选择单个文件外, 还可以通过添加元素的 multiple 属性, 并将该属性值设为 true, 实现选择多个文件的功能。

一个文件对应一个 file 对象,该对象中有两个重要的属性,一个是 name,表示不包含路径的文件名称;另外一个属性是 lastModifiedDate,表示文件最后修改的时间。当使用 file 类型的 input 元素选择多个文件时,该元素中就含有多个 file 对象,从而形成了 FileList 对象,即 FileList 对象就是指 file 对象的列表。

下面通过一个简单的实例介绍如何选择多个文件上传的过程。

实例 5-2 选择多个文件上传

1. 功能描述

在页面中,创建一个 file 类型的 input 元素,添加 multiple 属性,并将该属性的值设置为 true,当单击“选择文件”按钮时,同时选择三个文件,单击“打开”按钮后,页面中在“选择文件”按钮的后面将显示“3 个文件”的字样,移动鼠标指针到文字上时,显示这 3 个文件的详细名称与类型。

2. 实现代码

在 WebStorm 中新建一个 HTML 页面 5-2. html,加入代码如代码清单 5-2 所示。

代码清单 5-2 选择多个文件上传

```
<!DOCTYPE html>
<html>
<head>
<meta charset = "UTF-8">
<title>选择多个文件上传</title>
<link href = "Css/css5.css" rel = "stylesheet" type = "text/css">
</head>
<body>
<form id = "frmTmp">
<fieldset>
<legend>上传多个文件: </legend>
<input type = "file" name = "fleUpload"
id = "fleUpload" multiple = "true"/>
</fieldset>
</form>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 浏览器中执行的页面效果如图 5-2 所示。

4. 源码分析

在本实例中,由于 file 类型的 input 元素中添加了 multiple 属性,因此,可以通过该元素选择多个文件进行上传,选择成功后,“选择文件”按钮右侧不再显示文件的名称,而是显示成功选择上传文件的总量,当将鼠标指针移至总量上时,显示全部上传文件的详细列表。

当多个上传文件被选中时,在上传文件元素中,将会产生一个 FileList 对象,用来装载各文件的基本信息,如文件名称、类型、大小等。在上传文件总量的文字上移动鼠标指针时,



图 5-2 使用 file 类型的 input 元素实现多个文件选择时的效果

将调用该对象的列表信息,展示在页面中。

5.1.3 使用 Blob 接口获取文件的类型与大小

Blob 表示二进制数据块。Blob 接口中提供了一个 `slice()` 方法,通过该方法可以访问指定长度与类型的字节内部数据块。此外,该接口还提供了两个属性,一个为 `size`,表示返回数据块的大小;另外一个为 `type`,表示返回数据块的 MIME 类型。如果不能确定数据块的类型,则返回一个空字符串。在实例 5-1 与实例 5-2 中的 `file` 对象,实质上是 Blob 接口的一个实体,完全继承了该接口中的方法与属性。

下面通过一个实例介绍 `file` 对象如何继承 Blob 接口获取文件类型与大小的过程。

实例 5-3 获取上传文件的类型与大小

1. 功能描述

在页面的表单中,创建一个 file 类型的 input 元素,并将该元素的 `multiple` 属性设置为 `true`,表示允许选择多个上传文件。单击“选择文件”按钮,选取多个需要上传的文件后,在页面中,将以列表的方式展示所选文件的名称、类型、大小信息。

2. 实现代码

在 WebStorm 中新建一个 HTML 页面 5-3.html,加入代码如代码清单 5-3-1 所示。

代码清单 5-3-1 获取上传文件的类型与大小

```
<!DOCTYPE html>
<html>
<head>
<meta charset = "UTF-8">
<title>获取上传文件的类型与大小</title>
<link href = "Css/css5.css" rel = "stylesheet" type = "text/css">
<script type = "text/javascript" language = "javascript"
    src = "Js/js3.js">
</script>
</head>
```

```

<body>
<form id="frmTmp">
<fieldset>
  <legend>上传多个文件: </legend>
  <input type="file" name="fileUpload" id="fileUpload"
    onChange="fileUpload_GetFileList(this.files);"
    multiple="true"/>
  <ul id="ulUpload"></ul>
</fieldset>
</form>
</body>
</html>

```

在实例 5-3 中,页面导入一个 JavaScript 文件 js3.js,在该文件中,调用 fileUpload_GetFileList() 方法,以列表的形式展示上传文件的数据信息,其实现的代码如代码清单 5-3-2 所示。

代码清单 5-3-2 实例 5-3 中的 JavaScript 文件 js3.js 的源码

```

// JavaScript Document
function $$ (id) {
  return document.getElementById(id);
}
//选择上传文件时调用的函数
function fileUpload_GetFileList(f) {
  var strLi = "<li class='li'>";
  strLi = strLi + "<span>文件名称</span>";
  strLi = strLi + "<span>文件类型</span>";
  strLi = strLi + "<span>文件大小</span>";
  strLi = strLi + "</li>";
  for (var intI = 0; intI < f.length; intI++) {
    var tmpFile = f[intI];
    strLi = strLi + "<li>";
    strLi = strLi + "<span>" + tmpFile.name + "</span>";
    strLi = strLi + "<span>" + tmpFile.type + "</span>";
    strLi = strLi + "<span>" + tmpFile.size + " KB</span>";
    strLi = strLi + "</li>";
  }
  $$ ("ulUpload").innerHTML = strLi;
}

```

3. 页面效果

该页面在 Chrome 浏览器中执行的页面效果如图 5-3 所示。

4. 源码分析

在页面加粗代码中,file 类型的 input 元素,选择上传文件时,将触发 onChange 事件,在该事件中,调用自定义的函数 fileUpload_GetFileList(this.files),其中,实参 this.files 表示所选择的上传文件集合,即 FileList 对象;在函数 fileUpload_GetFileList()中,遍历传回的



图 5-3 使用 file 类型的 input 元素获取上传文件类型与大小时的效果

FileList 文件集合, 获取单个的 file 对象, 该对象通过继承 Blob 接口的属性, 返回文件的名称、类型、大小信息, 并将这些信息以叠加的方式保存在变量 strLi 中; 最后, 将变量的内容赋值给 id 为 ulUpload 的列表元素, 通过列表元素, 将上传文件的信息展示在页面中。详细实现过程见代码中加粗部分。

5.1.4 通过类型过滤选择的文件

在实例 5-3 中, 通过 file 对象可以获取每个上传文件的名称、类型、大小, 根据这个特征, 可以过滤上传文件的类型。具体流程是: 当选择上传文件后, 遍历每一个 file 对象, 获取该对象的类型, 并将该类型与设置的过滤类型进行匹配, 如果不符合, 则提示上传文件类型出错或拒绝上传, 从而实现在选择上传文件时, 就通过类型过滤了需要上传的文件。

下面通过一个实例介绍 file 对象通过类型过滤选择文件的过程。

实例 5-4 通过类型过滤上传文件

1. 功能描述

在页面表单中, 创建一个 file 类型的 input 元素, 并设置 multiple 属性为 true, 用于上传多个文件; 当单击“选择文件”按钮, 并选取了需要上传的文件后, 如果选取的文件中存在不符合“图片”类型的文件, 将在页面中显示总数量与文件名称。

2. 实现代码

在 WebStorm 中新建一个 HTML 页面 5-4.html, 加入代码如代码清单 5-4-1 所示。

代码清单 5-4-1 通过类型过滤上传文件

```
<!DOCTYPE html>
<html>
<head>
<meta charset = "UTF-8">
<title>通过类型过滤上传文件</title>
<link href = "css/css5.css" rel = "stylesheet" type = "text/css">
<script type = "text/javascript" language = "jscript">
```

```

        src = "Js/js4.js">
</script>
</head>
<body>
<form id="frmTmp">
<fieldset>
    <legend>上传过滤类型后的文件: </legend>
    <input type="file" name="fileUpload" id="fileUpload"
        onChange="fileUpload_CheckType(this.files);"
        multiple="true" />
    <p id="pTip"/>
</fieldset>
</form>
</body>
</html>

```

在实例 5-4 中,页面导入一个 JavaScript 文件 js4.js,在该文件中,调用 fileUpload_CheckType() 方法,并且按照设置的类型格式过滤需要上传的文件,其实现的代码如代码清单 5-4-2 所示。

代码清单 5-4-2 实例 5-4 中的 JavaScript 文件 js4.js 的源码

```

// JavaScript Document
function $$ (id) {
    return document.getElementById(id);
}
//选择上传文件时调用的函数
function fileUpload_CheckType(f) {
    var strP = "",
    strN = "",
    intJ = 0;
    var strFileType = /image.*/;
    for (var intI = 0; intI < f.length; intI++) {
        var tmpFile = f[intI];
        if (!tmpFile.type.match(strFileType)) {
            intJ = intJ + 1;
            strN = strN + tmpFile.name + "<br>";
        }
    }
    strP = "检测到(" + intJ + ")个非图片格式文件.";
    if (intJ > 0) {
        strP = strP + "文件名如下: <p>" + strN + "</p>";
    }
    $$("pTip").innerHTML = strP;
}

```

3. 页面效果

该页面在 Chrome 浏览器中执行的页面效果如图 5-4 所示。

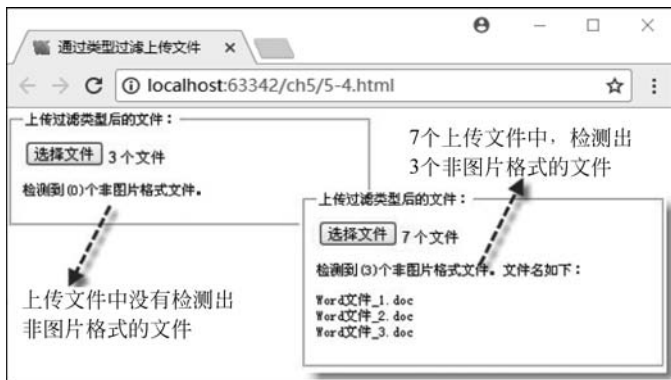


图 5-4 使用 file 类型的 input 元素过滤上传文件类型时的效果

4. 源码分析

在实例 5-3 中,我们知道,如果上传文件是图片类型,则 file 对象返回的类型均以 image/ 开头,后面添加“*”表示所有的图片类型或添加 gif 表示某种类型图片;因此,如果是一个图片文件,那么,该文件返回的类型必定以 image/ 字样开头。

根据这一特点,在本实例中,当遍历传回的文件集合时,通过 match() 方法检测每个文件返回的类型中是否含有 image/* 字样,如果没有,则说明是非图片类型文件,分别将总量与文件名称以叠加的形式保存在变量中;然后,将变量的内容赋值给 id 为 pTip 的元素,最后,通过该元素显示全部过滤文件的总量与名称表。详细实现过程见代码中加粗部分。

5.1.5 通过 accept 属性过滤选择文件的类型

在选择上传文件后,根据文件返回的类型过滤所选择的文件,是一种不错的方法,但需要编写不少的代码,除此方法之外,在 HTML5 中,还可以设置 file 类型 input 元素的 accept 属性为文件的过滤类型;设置完 accept 属性值后,在打开窗口选择文件时,默认的文件类型就是所设置的过滤类型。

下面通过一个实例简单介绍 file 类型 input 元素通过属性过滤选择文件类型的过程。

实例 5-5 通过 accept 属性过滤上传文件的类型

1. 功能描述

在页面表单中,创建一个 file 类型的 input 元素,并在元素中添加一个 accept 属性,属性值设置为 image/gif,当用户单击“选择文件”按钮时,在打开的文件选择窗口中,文件类型为 accept 属性所设置的值。

2. 实现代码

在 WebStorm 中新建一个 HTML 页面 5-5.html,加入代码如代码清单 5-5 所示。

代码清单 5-5 通过类型过滤上传文件

```
<!DOCTYPE html>
<html>
<head>
<meta charset = "UTF - 8">
<title>通过 accept 属性过滤某类型上传文件</title>
<link href = "Css/css5.css" rel = "stylesheet" type = "text/css">
</head>
<body>
<form id = "frmTmp">
  <fieldset>
    <legend>选择某类型上传文件: </legend>
    <input type = "file" name = "fileUpload"
      id = "fileUpload" accept = "image/gif" />
  </fieldset>
</form>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 浏览器中执行的页面效果如图 5-5 所示。



图 5-5 使用 file 类型的 input 元素通过设置 accept 属性过滤上传文件类型时的效果

4. 源码分析

在本实例中,由于对文件元素添加了 accept 属性,并设置 image/gif 类型作为该属性的值,因此,当单击“选择文件”按钮打开窗口时,其默认的选择文件类型就是所设置 accept 的属性值。详细实现过程见代码中加粗部分。

通过简单设置元素的一个属性,就可以在文件选择前过滤所选文件的类型,这种方法代码简单,操作方便。但是在目前的浏览器中,该方法不是很有效,原因在于,即便通过属性设置了文件选择的类型,但不是该类型的文件同样也可以被选中,也能被元素所接受。因此,使用这种过滤上传文件类型方法时,还需要谨慎。

5.2 读取与拖放文件

使用 Blob 接口可以获取上传文件的相关信息,如文件名称、大小、类型,但如果想要读取或浏览文件,则需要通过 FileReader 接口,该接口不仅可以读取图片文件,还可以读取文本或二进制文件。同时,根据该接口提供的事件与方法,可以动态侦察文件读取时的详细状态,接下来详细介绍 FileReader 接口的使用方法。

5.2.1 FileReader 接口

FileReader 接口提供了一个异步的 API,通过这个 API 可以从浏览器主线程中异步访问文件系统中的数据,基于此原因,FileReader 接口可以读取文件中的数据,并将读取的数据放入内存中。

当访问不同文件时,必须重新调用 FileReader 接口的构造函数,因为每调用一次,FileReader 接口都将返回一个新的 FileReader 对象,只有这样,才能实现访问不同文件的数据。

FileReader 接口提供了一整套完整的事件处理机制,用于侦察 FileReader 对象读取或返回数据时的各种进程状态。FileReader 接口的常用事件如表 5-1 所示。

表 5-1 FileReader 接口的常用事件

事件名称	说明
onloadstart	当读取数据开始时,触发该事件
onprogress	当正在读取数据时,触发该事件
onabort	当读取数据终止时,触发该事件
onerror	当读取数据失败时,触发该事件
onload	当读取数据成功时,触发该事件
onloadend	当请求操作成功时,无论操作是否成功,都触发该事件

在 FileReader 接口中,除提供了常用事件外,还拥有许多常用的方法,用于读取文件或响应事件,如 onabort 事件触发时,就要调用 abort() 方法。FileReader 接口的常用方法如表 5-2 所示。

表 5-2 FileReader 接口的常用方法

方法名称	参数	功能说明
readAsBinaryString()	file	以二进制的方式读取文件内容
readAsArrayBuffer()	file	以数组缓冲的方式读取文件内容
readAsText()	file,encoding	以文本编码的方式读取文件内容
readAsDataURL()	file	以数据 URL 的方式读取文件内容
Abort()	无	读取数据终止时,将自动触发该方法

针对 FileReader 接口中的方法,使用说明如下。

- 调用 readAsBinaryString() 方法时,将 file 对象返回的数据块,作为一个二进制字符

串的形式,分块读入内存中。

- 调用 `readAsArrayBuffer()` 方法时,将 `file` 对象返回的数据字节数,以数组缓冲的方式读入内存中。
- 调用 `readAsText()` 方法时,其中 `encoding` 参数表示文本文件编码的方式,默认值为 UTF-8,即以 UTF-8 编码格式将获取的数据块按文本方式读入内存中。
- 调用 `readAsDataURL()` 方法时,将 `file` 对象返回的数据块,以一串数据 URL 字符的形式展示在页面中,这种方法一般读取数据块较小的文件。

5.2.2 使用 `FileReader()` 方法预览图片文件

在前面的实例中,通过 `Blob` 接口可以访问文件数据块,获取文件相关信息。但如果想要读取文件,还需要通过 `FileReader` 接口中的方法,将数据读入内存或页面中。例如,尺寸较小的图片文件,可以通过 `FileReader` 接口中的 `readAsDataURL()` 方法,获取 API 异步读取的文件数据,另存为数据 URL,将该 URL 绑定 `img` 元素的 `src` 属性值,就可以实现图片文件预览的效果。

下面通过一个实例介绍使用 `readAsDataURL()` 方法预览图片的过程。

实例 5-6 使用 `readAsDataURL()` 方法预览图片

1. 功能描述

在页面表单中,添加一个 `file` 类型的 `input` 元素,用于选择上传文件,并设置属性 `multiple` 的值为 `true`,表示允许上传多个文件,单击“选择文件”按钮后,如果选择的是图片文件,将在页面中显示,实现图片文件上传的预览功能。

2. 实现代码

在 WebStorm 中新建一个 HTML 页面 5-6.html,加入代码如代码清单 5-6-1 所示。

代码清单 5-6-1 使用 `readAsDataURL()` 方法预览图片

```
<!DOCTYPE html>
<html>
<head>
<meta charset = "UTF - 8">
<title>使用 readAsDataURL() 方法预览图片</title>
<link href = "Css/css5.css" rel = "stylesheet" type = "text/css">
<script type = "text/javascript" language = "jscript"
    src = "Js/js6.js">
</script>
</head>
<body>
<form id = "frmTmp">
<fieldset>
<legend>预览图片文件: </legend>
<input type = "file" name = "fileUpload" id = "fileUpload"
    onChange = "fileUpload_PrevImageFile(this.files);"
    multiple = "true" />
<ul id = "ulUpload"></ul>
```

```

</fieldset>
</form>
</body>
</html>

```

在实例 5-6 中,页面导入一个 JavaScript 文件 js6.js,在该文件中,调用 fileUpload_PrevImageFile()方法,该方法访问 FileReader 接口,将文件以数据 URL 的方式返回页面,其实现的代码如代码清单 5-6-2 所示。

代码清单 5-6-2 实例 5-6 中的 JavaScript 文件 js6.js 的源码

```

// JavaScript Document
function $$ (id) {
    return document.getElementById(id);
}
//选择上传文件时调用的函数
function fileUpload_PrevImageFile(f) {
    //检测浏览器是否支持 FileReader 对象
    if (typeof FileReader == 'undefined') {
        alert("检测到您的浏览器不支持 FileReader 对象!");
    }
    var strHTML = "";
    for (var intI = 0; intI < f.length; intI++) {
        var tmpFile = f[intI];
        var reader = new FileReader();
        reader.readAsDataURL(tmpFile);
        reader.onload = function(e) {
            strHTML = strHTML + "<span>";
            strHTML = strHTML + "<img src = '" + e.target.result
                + "' alt = '/'>";
            strHTML = strHTML + "</span>";
            $$("ulUpload").innerHTML = "<li>" + strHTML + "</li>";
        }
    }
}

```

3. 页面效果

该页面在 Chrome 浏览器中执行的页面效果如图 5-6 所示。

4. 源码分析

在本实例中,图片预览的过程实质上是图片文件被读取后展示在页面中的过程,为了实现这一过程,需要引用 FileReader 接口中提供的读取文件方法 readAsDataURL(),在引用接口前,考虑到各浏览器对接口的兼容性不一样,JavaScript 代码首先检测用户的浏览器是否支持 FileReader 对象,如果不支持,则提示出错信息。

接下来,在 JavaScript 代码中,遍历传回的上传文件集合,获取每个 file 对象,由于每个文件返回的数据块都不同,因此每次在读取文件前,必须先重构一个新的 FileReader 对象,然后将每个文件以数据 URL 的方式读入页面中,当读取成功时,触发 onload 事件,在该事



图 5-6 使用 readAsDataURL() 方法预览图片时的效果

件中,通过 result 属性获取文件读入页面中的 URL 地址,并将该地址与 img 元素进行绑定,最后通过列表 id 为 ulUpload 的列表元素展示在页面中,从而实现图片文件上传预览的效果。详细实现过程见代码中加粗部分。

5.2.3 使用 FileReader () 方法读取文本文件

除使用 FileReader 接口中的 readAsDataURL() 方法读取图片文件、实现图片预览外,还可以借助接口提供的 readAsText() 方法,将文件以文本编码的方式进行读取,即可以读取上传文本文件的内容,其实现的方法与读取图片基本相似,仅是读取文件的方式不一样。

下面通过一个实例介绍使用 readAsText () 方法读取文本文件内容的过程。

实例 5-7 使用 readAsText () 方法读取文本文件内容

1. 功能描述

在页面表单中,新建一个 file 类型的 input 元素,用于获取上传的文本文件,当单击“选择文件”按钮,挑选了一个文本文件后,在页面中将显示该选择文本文件的内容。

2. 实现代码

在 WebStorm 中新建一个 HTML 页面 5-7. html,加入代码如代码清单 5-7-1 所示。

代码清单 5-7-1 使用 readAsText () 方法读取文本文件内容

```
<!DOCTYPE html>
<html>
<head>
<meta charset = "UTF - 8">
<title>使用 readAsText() 方法读取文本文件</title>
<link href = "Css/css5.css" rel = "stylesheet" type = "text/css">
<script type = "text/javascript" language = "javascript"
    src = "Js/js7.js">
</script>
</head>
<body>
```

```

< form id = "frmTmp">
  <fieldset>
    < legend>读取文本文件: </legend>
    < input type = "file" name = "fileUpload" id = "fileUpload"
      onChange = "fileUpload_ReadTxtFile(this.files);" />
    < article id = "artShow"></article>
  </fieldset>
</form>
</body>
</html>

```

在实例 5-7 中,页面导入一个 JavaScript 文件 js7.js,在该文件中,调用 fileUpload_ReadTxtFile() 方法,该方法将文件以文本编码方式读取并返回页面,其实现的代码如代码清单 5-7-2 所示。

代码清单 5-7-2 实例 5-7 中的 JavaScript 文件 js7.js 的源码

```

// JavaScript Document
function $$ (id) {
  return document.getElementById(id);
}
//选择上传文件时调用的函数
function fileUpload_ReadTxtFile(f) {
  //检测浏览器是否支持 FileReader 对象
  if (typeof FileReader == 'undefined') {
    alert("检测到您的浏览器不支持 FileReader 对象!");
  }
  var tmpFile = f[0];
  var reader = new FileReader();
  reader.readAsText(tmpFile);
  reader.onload = function(e) {
    $$ ("artShow").innerHTML = "<pre>" +
    e.target.result + "</pre>";
  }
}

```

3. 页面效果

该页面在 Chrome 浏览器中执行的页面效果如图 5-7 所示。



图 5-7 使用 readAsText()方法读取文本文件时的效果

4. 源码分析

在本实例中,由于 file 类型的 input 文件上传元素,没有添加 multiple 属性,因此,单击“选择文件”按钮后,将返回单个 file 文件。

与实例 5-6 一样,JavaScript 代码中,首先检测浏览器是否支持 FileReader 对象,如果支持,则重构一个新的 FileReader 对象,调用该对象的 readAsText()方法,将文件以文本编码的方式读入页面中,然后通过 result 属性获取读入的内容,并将该内容赋值给 id 为 artShow 的元素,最后通过该元素将文本文件的内容显示在页面上。

5.2.4 监听 FileReader 接口中的事件

在 FileReader 接口中,提供了很多常用的事件,通过这些事件的触发,可以清晰地监听 FileReader 对象读取文件的详细过程,掌握这一过程中按顺序都触发了哪些事件,从而可以更加精确地定位每次读取文件时的事件先后顺序,为编写事件代码提供有力的支持。

经过反复测试,一个文件通过 FileReader 接口中的方法正常读取时触发事件的先后顺序如图 5-8 所示。

针对文件读取时,事件触发先后顺序示意图,说明如下。

- 大部分的文件读取过程,都集中在 onprogress 事件中,该事件耗时最长。
- 如果文件在读取过程中出现异常或中止,那么, onprogress 事件将结束,直接触发 onerror 或 onabort 事件,而不会触发 onload 事件。
- onload 事件是文件读取成功时触发,而 onloadend 虽然也是文件操作成功时触发,但该事件不论文件读取是否成功,都将触发,因此,想要正确获取文件数据,必须在 onload 事件中编写代码。

下面通过一个实例介绍文件读取时触发事件的先后顺序。

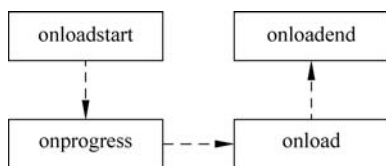


图 5-8 使用 readAsText() 方法读取顺序

实例 5-8 展示文件读取时触发事件的先后顺序

1. 功能描述

在页面的表单中,添加一个 file 类型的 input 元素,当用户单击“选择文件”按钮,并通过打开的窗口选取了一个文件后,页面中将展示读取文件过程中所触发事件的内容。

2. 实现代码

在 WebStorm 中新建一个 HTML 页面 5-8.html,加入代码如代码清单 5-8-1 所示。

代码清单 5-8-1 展示文件读取时触发事件的先后顺序

```

<!DOCTYPE html>
<html>
<head>
<meta charset = "UTF - 8">

```

```

<title>展示文件读取时触发事件的先后顺序</title>
<link href = "Css/css5.css" rel = "stylesheet" type = "text/css">
<script type = "text/javascript" language = "jscript"
    src = "Js/js8.js">
</script>
</head>
<body>
<form id = "frmTmp">
    <fieldset>
        <legend>文件读取事件的顺序:</legend>
        <input type = "file" name = "fileUpload" id = "fileUpload"
            onChange = "fileUpload_ShowEvent(this.files);" />
        <p id = "pStatus"></p>
    </fieldset>
</form>
</body>
</html>

```

在实例 5-8 中, 页面导入一个 JavaScript 文件 js8.js, 在该文件中, 调用 fileUpload_ShowEvent() 方法, 该方法列出文件在正常读取过程中的全部事件, 其实现的代码如代码清单 5-8-2 所示。

代码清单 5-8-2 实例 5-8 中的 JavaScript 文件 js8.js 的源码

```

// JavaScript Document
function $$ (id) {
    return document.getElementById(id);
}
//选择上传文件时调用的函数
function fileUpload_ShowEvent(f) {
    if (typeof FileReader == 'undefined') {
        alert("检测到您的浏览器不支持 FileReader 对象!");
    }
    var tmpFile = f[0];
    var reader = new FileReader();
    reader.readAsText(tmpFile);
    reader.onload = function(e) {
        $$ ("pStatus").style.display = "block";
        $$ ("pStatus").innerHTML = "数据读取成功!";
    }
    reader.onloadstart = function(e) {
        $$ ("pStatus").style.display = "block";
        $$ ("pStatus").innerHTML = "开始读取数据...";
    }
    reader.onloadend = function(e) {
        $$ ("pStatus").style.display = "block";
        $$ ("pStatus").innerHTML = "文件读取成功!";
    }
    reader.onprogress = function(e) {

```

```

    $$ ("pStatus").style.display = "block";
    $$ ("pStatus").innerHTML = "正在读取数据...";
}
}

```

3. 页面效果

该页面在 Chrome 浏览器中执行的页面效果如图 5-9 所示。



图 5-9 显示文件读取过程中各事件执行的先后顺序

4. 源码分析

在本实例的页面中,单击“选择文件”按钮后,触发一个自定义的函数 fileUpload_ShowEvent(),在该函数中,首先检测浏览器是否支持 FileReader 对象,如果不支持,则弹出错误提示信息,然后重新构造一个新的 FileReader 对象,并对传回的文件以文本编码的方式读入页面;最后列出文件在正常读取过程中将触发的 4 个事件,在每个事件中,先显示 id 为 pStatus 的元素,后将事件的状态内容设置为该元素的文本内容;当 FileReader 对象执行 readAsText()方法读取文件时,各个不同事件将按执行顺序被触发,设置的状态内容以动态的方式显示在 id 为 pStatus 的页面元素中。

5.2.5 使用 DataTransfer 对象拖放上传图片文件

在 HTML5 中,借助于 DataTransfer 接口提供的方法,可以实现浏览器与其他应用程序之间文件间的拖动,实现拖动数据的操作。虽然在 HTML 4 之前的版本中也支持拖放数据的操作,但该操作仅局限在整个浏览器中,而非浏览器之外的数据。

下面通过一个实例介绍使用 DataTransfer 对象拖放上传图片文件的过程。

实例 5-9 使用 DataTransfer 对象拖放上传图片文件

1. 功能描述

在页面的表单中,创建一个 ul 元素,用于接收并预览拖放过来的图片文件,当用户从计算机的文件夹中选择图片文件后,可以以拖动的方式将文件放入该元素内,并以预览的方式进行显示。

2. 实现代码

在 WebStorm 中新建一个 HTML 页面 5-9. html, 加入代码如代码清单 5-9-1 所示。

代码清单 5-9-1 使用 DataTransfer 对象拖放上传图片文件

```
<!DOCTYPE html>
<html>
<head>
<meta charset = "UTF-8">
<title>拖放选择上传文件</title>
<link href = "Css/css5.css" rel = "stylesheet" type = "text/css">
<script type = "text/javascript" language = "jscript"
    src = "Js/js9.js">
</script>
</head>
<body>
<form id = "frmTmp">
    <fieldset>
        <legend>拖动选择文件: </legend>
        <ul id = "ulUpload" ondrop = "dropFile (event)"
            ondragenter = "return false"
            ondragover = "return false">
        </ul>
    </fieldset>
</form>
</body>
</html>
```

在实例 5-9 中, 页面导入一个 JavaScript 文件 js9. js, 在该文件中, 调用 fileUpload_MoveFile() 方法, 将拖动的文件数据放入 DataTransfer 对象, 然后调取。其实现的代码如代码清单 5-9-2 所示。

代码清单 5-9-2 实例 5-9 中的 JavaScript 文件 js9. js 的源码

```
// JavaScript Document
function $$ (id) {
    return document.getElementById(id);
}
//选择上传文件时调用的函数
function fileUpload_MoveFile(f) {
    //检测浏览器是否支持 FileReader 对象
    if (typeof FileReader == 'undefined') {
        alert("检测到您的浏览器不支持 FileReader 对象!");
    }
    for (var intI = 0; intI < f.length; intI++) {
        var tmpFile = f[intI];
        var reader = new FileReader();
        reader.readAsDataURL(tmpFile);
        reader.onload = (function(f1) {
            return function(e) {
```

```

        vareleSpan = document.createElement('span');
        eleSpan.innerHTML = ['<img src = "',
            e.target.result,
            '" title = "',
            f1.name, '" />'].join('');
        $$ ('ulUpload').insertBefore(eleSpan, null);
    }
})(tmpFile);
}
}
function dropFile(e) {
    //调用预览上传文件的方式
    fileUpload_MoveFile(e.dataTransfer.files);
    //停止事件的传播
    e.stopPropagation();
    //阻止默认事件的发生
    e.preventDefault();
}

```

3. 页面效果

该页面在 Chrome 浏览器中执行的页面效果如图 5-10 所示。

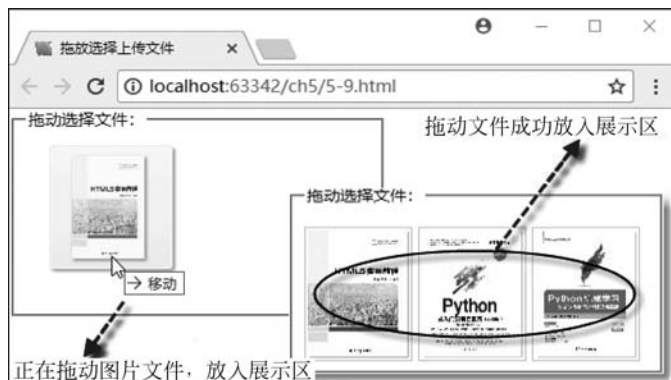


图 5-10 显示拖放选择上传图片文件的效果

4. 源码分析

在本实例中,文件在从文件夹拖入页面目标元素的过程中,通过 `DataTransfer` 对象中的 `setData()` 方法保存数据,页面中的目标元素为了接收被保存的数据,在调用元素的拖放事件 `ondrop` 中调用了一个自定义的函数 `dropFile()`,同时,为了确保目标元素顺利接收拖放文件,必须将目标元素的 `ondragenter` 与 `ondragover` 两个事件都返回 `false`。详细实现过程见代码中加粗部分。

在自定义的函数 `dropFile()` 中,先调用另一个自定义的函数 `fileUpload_MoveFile()`,同时,要实现文件的拖放过程,还要在目标元素的拖放事件中,停止其他事件的传播并且关闭默认事件,其实现的过程如 JavaScript 代码中自定义的函数 `dropFile()` 所示。

在自定义的函数 `fileUpload_MoveFile()` 中,先从 `DataTransfer` 对象中获取被保存的文

件集合,然后,遍历整个集合中的文件成员,获取每一个单独的文件,通过重构一个 FileReader 对象,调用该对象中的 readAsDataURL(),将文件以数据地址的形式读入页面中,同时,创建页面元素 span,将数据地址与 img 元素绑定,通过 join()方法,写入 span 元素的内容中,最后,将全部获取的内容写入 id 为 ulUpload 的列表元素中,通过该元素展示在页面中。其实现的过程如 JavaScript 代码中自定义的函数 fileUpload_MoveFile ()所示。

有关 DataTransfer 对象更多的属性与方法,将在本书的第 11 章中有详细的介绍与运用。

小结

文件在 HTML5 中占有很重要的地位,在本节中,首先介绍选择单个与多个文件的实现方法,然后结合详细实例阐述选择文件后,以各种方式读取数据的过程。同时,讲解了如何在文件读取过程中,捕获出现的错误与异常的方法,为读者完整掌握 HTML5 中文件的使用打下扎实的理论与实践基础。