

表单，FormData 对象

表单概述

表单（`<form>`）用来收集用户提交的数据，发送到服务器。比如，用户提交用户名和密码，让服务器验证，就要通过表单。表单提供多种控件，让开发者使用，具体的控件种类和用法请参考 HTML 语言的教程。本章主要介绍 JavaScript 与表单的交互。

```
```html
<form action="/handling-page" method="post">
 <div>
 <label for="name">用户名: </label>
 <input type="text" id="name" name="user_name" />
 </div>
 <div>
 <label for="passwd">密码: </label>
 <input type="password" id="passwd" name="user_passwd" />
 </div>
 <div>
 <input type="submit" id="submit" name="submit_button" value="提交" />
 </div>
</form>
```
```

上面代码就是一个简单的表单，包含三个控件：用户名输入框、密码输入框和提交按钮。

用户点击“提交”按钮，每一个控件都会生成一个键值对，键名是控件的`name`属性，键值是控件的`value`属性，键名和键值之间由等号连接。比如，用户名输入框的`name`属性是`user_name`，`value`属性是用户输入的值，假定是“张三”，提交到服务器的时候，就会生成一个键值对`user_name=张三`。

所有的键值对都会提交到服务器。但是，提交的数据格式跟`<form>`元素的`method`属性有关。该属性指定了提交数据的 HTTP 方法。如果是 GET 方法，所有键值对会以 URL 的查询字符串形式，提交到服务器，比如`/handling-page?user_name=张三&user_passwd=123&submit_button=提交`。下面就是 GET 请求的 HTTP 头信息。

```
```http
GET /handling-page?user_name=张三&user_passwd=123&submit_button=提交
Host: example.com
```
```

如果是 POST 方法，所有键值对会连接成一行，作为 HTTP 请求的数据体发送到服务器，比如`user_name=张三&user_passwd=123&submit_button=提交`。下面就是 POST 请求的头信息。

```
```http
POST /handling-page HTTP/1.1
```

```
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 74
```

```
user_name=张三&user_passwd=123&submit_button=提交
'''
```

注意，实际提交的时候，只要键值不是 URL 的合法字符（比如汉字“张三”和“提交”），浏览器会自动对其进行编码。

点击`submit`控件，就可以提交表单。

```
'''html
<form>
 <input type="submit" value="提交">
</form>
'''
```

上面表单就包含一个`submit`控件，点击这个控件，浏览器就会把表单数据向服务器提交。

注意，表单里面的`<button>`元素如果没有用`type`属性指定类型，那么默认就是`submit`控件。

```
'''html
<form>
 <button>提交</button>
</form>
'''
```

上面表单的`<button>`元素，点击以后也会提交表单。

除了点击`submit`控件提交表单，还可以用表单元素的`submit()`方法，通过脚本提交表单。

```
'''javascript
formElement.submit();
'''
```

表单元素的`reset()`方法可以重置所有控件的值（重置为默认值）。

```
'''javascript
formElement.reset()
'''
```

## ## FormData 对象

### ### 概述

表单数据以键值对的形式向服务器发送，这个过程是浏览器自动完成的。但是有时候，我们希望通过脚本完成过程，构造和编辑表单键值对，然后通过`XMLHttpRequest.send()`方法发送。浏览器原生提供了`FormData`对象来完成这项工作。

`FormData` 首先是一个构造函数，用来生成实例。

```
```javascript
var formData = new FormData(form);
```
```

`FormData()` 构造函数的参数是一个表单元素，这个参数是可选的。如果省略参数，就表示一个空的表单，否则就会处理表单元素里面的键值对。

下面是一个表单。

```
```html
<form id="myForm" name="myForm">
  <div>
    <label for="username">用户名: </label>
    <input type="text" id="username" name="username">
  </div>
  <div>
    <label for="useracc">账号: </label>
    <input type="text" id="useracc" name="useracc">
  </div>
  <div>
    <label for="userfile">上传文件: </label>
    <input type="file" id="userfile" name="userfile">
  </div>
  <input type="submit" value="Submit!">
</form>
```
```

我们用 `FormData` 对象处理上面这个表单。

```
```javascript
var myForm = document.getElementById('myForm');
var formData = new FormData(myForm);
```

```
// 获取某个控件的值
formData.get('username') // ""
```

```
// 设置某个控件的值
formData.set('username', '张三');
```

```
formData.get('username') // "张三"
```
```

### ### 实例方法

FormData 提供以下实例方法。

- `FormData.get(key)`: 获取指定键名对应的键值, 参数为键名。如果有多个同名的键值对, 则返回第一个键值对的键值。
- `FormData.getAll(key)`: 返回一个数组, 表示指定键名对应的所有键值。如果有多个同名的键值对, 数组会包含所有的键值。
- `FormData.set(key, value)`: 设置指定键名的键值, 参数为键名。如果键名不存在, 会添加这个键值对, 否则会更新指定键名的键值。如果第二个参数是文件, 还可以使用第三个参数, 表示文件名。
- `FormData.delete(key)`: 删除一个键值对, 参数为键名。
- `FormData.append(key, value)`: 添加一个键值对。如果键名重复, 则会生成两个相同键名的键值对。如果第二个参数是文件, 还可以使用第三个参数, 表示文件名。
- `FormData.has(key)`: 返回一个布尔值, 表示是否具有该键名的键值对。
- `FormData.keys()`: 返回一个遍历器对象, 用于`for...of`循环遍历所有的键名。
- `FormData.values()`: 返回一个遍历器对象, 用于`for...of`循环遍历所有的键值。
- `FormData.entries()`: 返回一个遍历器对象, 用于`for...of`循环遍历所有的键值对。如果直接用`for...of`循环遍历 FormData 实例, 默认就会调用这个方法。

下面是`get()`、`getAll()`、`set()`、`append()`方法的例子。

```
```javascript
var formData = new FormData();

formData.set('username', '张三');
formData.append('username', '李四');
formData.get('username') // "张三"
formData.getAll('username') // ["张三", "李四"]

formData.append('userpic[]', myFileInput.files[0], 'user1.jpg');
formData.append('userpic[]', myFileInput.files[1], 'user2.jpg');
```
```

下面是遍历器的例子。

```
```javascript
var formData = new FormData();
formData.append('key1', 'value1');
formData.append('key2', 'value2');

for (var key of formData.keys()) {
  console.log(key);
}
// "key1"
```

```

// "key2"

for (var value of formData.values()) {
  console.log(value);
}
// "value1"
// "value2"

for (var pair of formData.entries()) {
  console.log(pair[0] + ': ' + pair[1]);
}
// key1: value1
// key2: value2

// 等同于遍历 formData.entries()
for (var pair of formData) {
  console.log(pair[0] + ': ' + pair[1]);
}
// key1: value1
// key2: value2

```

表单的内置验证

自动校验

表单提交的时候，浏览器允许开发者指定一些条件，它会自动验证各个表单控件的值是否符合条件。

```

<<html
<!-- 必填 -->
<input required>

<!-- 必须符合正则表达式 -->
<input pattern="banana|cherry">

<!-- 字符串长度必须为6个字符 -->
<input minlength="6" maxlength="6">

<!-- 数值必须在1到10之间 -->
<input type="number" min="1" max="10">

<!-- 必须填入 Email 地址 -->
<input type="email">

<!-- 必须填入 URL -->
<input type="URL">

```

如果一个控件通过验证，它就会匹配`:valid`的 CSS 伪类，浏览器会继续进行表单提交的流程。如果没有通过验证，该控件就会匹配`:invalid`的 CSS 伪类，浏览器会终止表单提交，并显示一个错误信息。

checkValidity()

除了提交表单的时候，浏览器自动校验表单，还可以手动触发表单的校验。表单元素和表单控件都有`checkValidity()`方法，用于手动触发校验。

```
```javascript
// 触发整个表单的校验
form.checkValidity()

// 触发单个表单控件的校验
formControl.checkValidity()
```
```

`checkValidity()`方法返回一个布尔值，`true`表示通过校验，`false`表示没有通过校验。因此，提交表单可以封装为下面的函数。

```
```javascript
function submitForm(action) {
 var form = document.getElementById('form');
 form.action = action;
 if (form.checkValidity()) {
 form.submit();
 }
}
```
```

willValidate 属性

控件元素的`willValidate`属性是一个布尔值，表示该控件是否会在提交时进行校验。

```
```javascript
// HTML 代码如下
// <form novalidate>
// <input id="name" name="name" required />
// </form>

var input = document.querySelector('#name');
input.willValidate // true
```
```

validationMessage 属性

控件元素的`validationMessage`属性返回一个字符串，表示控件不满足校验条件时，浏览器显示的提示文本。以下两种情况，该属性返回空字符串。

- 该控件不会在提交时自动校验
- 该控件满足校验条件

```
```javascript
// HTML 代码如下
// <form><input type="text" required></form>
document.querySelector('form input').validationMessage
// "请填写此字段。"
```
```

下面是另一个例子。

```
```javascript
var myInput = document.getElementById('myinput');
if (!myInput.checkValidity()) {
 document.getElementById('prompt').innerHTML = myInput.validationMessage;
}
```
```

setCustomValidity()

控件元素的`setCustomValidity()`方法用来定制校验失败时的报错信息。它接受一个字符串作为参数，该字符串就是定制的报错信息。如果参数为空字符串，则上次设置的报错信息被清除。

如果调用这个方法，并且参数不为空字符串，浏览器就会认为控件没有通过校验，就会立刻显示该方法设置的报错信息。

```
```javascript
/* HTML 代码如下
<form>
 <p><input type="file" id="fs"></p>
 <p><input type="submit"></p>
</form>
*/

document.getElementById('fs').onchange = checkFileSize;

function checkFileSize() {
 var fs = document.getElementById('fs');
 var files = fs.files;
 if (files.length > 0) {
 if (files[0].size > 75 * 1024) {
 fs.setCustomValidity('文件不能大于 75KB');
 return;
 }
 }
 fs.setCustomValidity('');
}
```
```

上面代码一旦发现文件大于 75KB，就会设置校验失败，同时给出自定义的报错信息。然后，点击提交按钮时，就会显示报错信息。这种校验失败是不会自动消除的，所以如果所有文件都符合条件，要将报错信息设为空字符串，手动消除校验失败的状态。

validity 属性

控件元素的属性`validity`属性返回一个`ValidityState`对象，包含当前校验状态的信息。

该对象有以下属性，全部为只读属性。

- `ValidityState.badInput`：布尔值，表示浏览器是否不能将用户的输入转换成正确的类型，比如用户在数值框里面输入字符串。
- `ValidityState.customError`：布尔值，表示是否已经调用`setCustomValidity()`方法，将校验信息设置为一个非空字符串。
- `ValidityState.patternMismatch`：布尔值，表示用户输入的值是否不满足模式的要求。
- `ValidityState.rangeOverflow`：布尔值，表示用户输入的值是否大于最大范围。
- `ValidityState.rangeUnderflow`：布尔值，表示用户输入的值是否小于最小范围。
- `ValidityState.stepMismatch`：布尔值，表示用户输入的值不符合步长的设置（即不能被步长值整除）。
- `ValidityState.tooLong`：布尔值，表示用户输入的字数超出了最长字数。
- `ValidityState.tooShort`：布尔值，表示用户输入的字符少于最短字数。
- `ValidityState.typeMismatch`：布尔值，表示用户填入的值不符合类型要求（主要是类型为 Email 或 URL 的情况）。
- `ValidityState.valid`：布尔值，表示用户是否满足所有校验条件。
- `ValidityState.valueMissing`：布尔值，表示用户没有填入必填的值。

下面是一个例子。

```
```javascript
var input = document.getElementById('myinput');
if (input.validity.valid) {
 console.log('通过校验');
} else {
 console.log('校验失败');
}
```
```

下面是另外一个例子。

```
```javascript
var txt = '';
if (document.getElementById('myInput').validity.rangeOverflow) {
 txt = '数值超过上限';
}
```



```
}
document.getElementById('prompt').innerHTML = txt;
```
```

表单的 novalidate 属性

表单元素的 HTML 属性`novalidate`，可以关闭浏览器的自动校验。

```
```html  
<form novalidate>
</form>
```
```

这个属性也可以在脚本里设置。

```
```javascript  
form.noValidate = true;
```
```

如果表单元素没有设置`novalidate`属性，那么提交按钮（`<button>`或`<input>`元素）的`formnovalidate`属性也有同样的作用。

```
```html  
<form>
 <input type="submit" value="submit" formnovalidate>
</form>
```
```

enctype 属性

表单能够用四种编码，向服务器发送数据。编码格式由表单的`enctype`属性决定。

假定表单有两个字段，分别是`foo`和`baz`，其中`foo`字段的值等于`bar`，`baz`字段的值是一个分为两行的字符串。

```
```  
The first line.
The second line.
```
```

下面四种格式，都可以将这个表单发送到服务器。

** (1) GET 方法**

如果表单使用`GET`方法发送数据，`enctype`属性无效。

```
```html  
<form
 action="register.php"
```

```

 method="get"
 onsubmit="AJAXSubmit(this); return false;"
>
</form>
'''

```

数据将以 URL 的查询字符串发出。

```

'''http
?foo=bar&baz=The%20first%20line.%0AThe%20second%20line.
'''

```

**\*\* (2) application/x-www-form-urlencoded \*\***

如果表单用`POST`方法发送数据，并省略`enctype`属性，那么数据以`application/x-www-form-urlencoded`格式发送（因为这是默认值）。

```

'''html
<form
 action="register.php"
 method="post"
 onsubmit="AJAXSubmit(this); return false;"
>
</form>
'''

```

发送的 HTTP 请求如下。

```

'''http
Content-Type: application/x-www-form-urlencoded

foo=bar&baz=The+first+line.%0D%0AThe+second+line.%0D%0A
'''

```

上面代码中，数据体里面的`%0D%0A`代表换行符（`\r\n`）。

**\*\* (3) text/plain \*\***

如果表单使用`POST`方法发送数据，`enctype`属性为`text/plain`，那么数据将以纯文本格式发送。

```

'''html
<form
 action="register.php"
 method="post"
 enctype="text/plain"
 onsubmit="AJAXSubmit(this); return false;"
>
</form>
'''

```

发送的 HTTP 请求如下。

```
""http
Content-Type: text/plain

foo=bar
baz=The first line.
The second line.
""
```

#### **\*\* (4) multipart/form-data\*\***

如果表单使用`POST`方法, `enctype`属性为`multipart/form-data`, 那么数据将以混合的格式发送。

```
""html
<form
 action="register.php"
 method="post"
 enctype="multipart/form-data"
 onsubmit="AJAXSubmit(this); return false;"
>
</form>
""
```

发送的 HTTP 请求如下。

```
""http
Content-Type: multipart/form-data; boundary=-----314911788813839

-----314911788813839
Content-Disposition: form-data; name="foo"

bar
-----314911788813839
Content-Disposition: form-data; name="baz"

The first line.
The second line.

-----314911788813839--
""
```

这种格式也是文件上传的格式。

#### **## 文件上传**

用户上传文件, 也是通过表单。具体来说, 就是通过文件输入框选择本地文件, 提交表单的时候, 浏览器就会把这个文件发送到服务器。

```
```html
<input type="file" id="file" name="myFile">
```
```

此外，还需要将表单`<form>`元素的`method`属性设为`POST`，`enctype`属性设为`multipart/form-data`。其中，`enctype`属性决定了 HTTP 头信息的`Content-Type`字段的值，默认情况下这个字段的值是`application/x-www-form-urlencoded`，但是文件上传的时候要改成`multipart/form-data`。

```
```html
<form method="post" enctype="multipart/form-data">
  <div>
    <label for="file">选择一个文件</label>
    <input type="file" id="file" name="myFile" multiple>
  </div>
  <div>
    <input type="submit" id="submit" name="submit_button" value="上传" />
  </div>
</form>
```
```

上面的 HTML 代码中，file 控件的`multiple`属性，指定可以一次选择多个文件；如果没有这个属性，则一次只能选择一个文件。

```
```javascript
var fileSelect = document.getElementById('file');
var files = fileSelect.files;
```
```

然后，新建一个 FormData 实例对象，模拟发送到服务器的表单数据，把选中的文件添加到这个对象上面。

```
```javascript
var formData = new FormData();

for (var i = 0; i < files.length; i++) {
  var file = files[i];

  // 只上传图片文件
  if (!file.type.match('image.*')) {
    continue;
  }

  formData.append('photos[]', file, file.name);
}
```
```

最后，使用 Ajax 向服务器上传文件。

```
````javascript
var xhr = new XMLHttpRequest();

xhr.open('POST', 'handler.php', true);

xhr.onload = function () {
  if (xhr.status !== 200) {
    console.log('An error occurred!');
  }
};

xhr.send(formData);
````
```

除了发送 FormData 实例，也可以直接 AJAX 发送文件。

```
````javascript
var file = document.getElementById('test-input').files[0];
var xhr = new XMLHttpRequest();

xhr.open('POST', 'myserver/uploads');
xhr.setRequestHeader('Content-Type', file.type);
xhr.send(file);
````
```