```
if (document.forms[0].elements[0].checkValidity()){
  // 字段有效，继续
} else {
  // 字段无效
}
```

要检查整个表单是否有效，可以直接在表单上调用 checkValidity()方法。这个方法会在所有字段都有效时返回 true，有一个字段无效就会返回 false：

```
if(document.forms[0].checkValidity()){
  // 表单有效，继续
} else {
  // 表单无效
}
```

checkValidity()方法只会告诉我们字段是否有效，而 validity 属性会告诉我们字段为什么有效或无效。这个属性是一个对象，包含一系列返回布尔值的属性。

- customError：如果设置了 setCustomValidity()就返回 true，否则返回 false。
- patternMismatch：如果字段值不匹配指定的 pattern 属性则返回 true。
- rangeOverflow：如果字段值大于 max 的值则返回 true。
- rangeUnderflow：如果字段值小于 min 的值则返回 true。
- stepMisMatch：如果字段值与 min、max 和 step 的值不相符则返回 true。
- tooLong：如果字段值的长度超过了 maxlength 属性指定的值则返回 true。某些浏览器，如 Firefox 4 会自动限制字符数量，因此这个属性值始终为 false。
- typeMismatch：如果字段值不是"email"或"url"要求的格式则返回 true。
- valid：如果其他所有属性的值都为 false 则返回 true。与 checkValidity()的条件一致。
- valueMissing：如果字段是必填的但没有值则返回 true。

因此，通过 validity 属性可以检查表单字段的有效性，从而获取更具体的信息，如下面的代码所示：

```
if (input.validity && !input.validity.valid){
  if (input.validity.valueMissing){
    console.log("Please specify a value.")
  } else if (input.validity.typeMismatch){
    console.log("Please enter an email address.");
  } else {
    console.log("Value is invalid.");
  }
}
```

6. 禁用验证

通过指定 novalidate 属性可以禁止对表单进行任何验证：

```
<form method="post" action="/signup" novalidate>
    <!-- 表单元素 -->
</form>
```

这个值也可以通过 JavaScript 属性 noValidate 检索或设置，设置为 true 表示属性存在，设置为 false 表示属性不存在：

```
document.forms[0].noValidate = true;   // 关闭验证
```

如果一个表单中有多个提交按钮，那么可以给特定的提交按钮添加 formnovalidate 属性，指定通过该按钮无须验证即可提交表单：

```
<form method="post" action="/foo">
    <!-- 表单元素 -->
    <input type="submit" value="Regular Submit">
    <input type="submit" formnovalidate name="btnNoValidate"
           value="Non-validating Submit">
</form>
```

在这个例子中，第一个提交按钮会让表单像往常一样验证数据，第二个提交按钮则禁用了验证，可以直接提交表单。我们也可以使用 JavaScript 来设置这个属性：

```
// 关闭验证
document.forms[0].elements["btnNoValidate"].formNoValidate = true;
```

19.3 选择框编程

选择框是使用<select>和<option>元素创建的。为方便交互，HTMLSelectElement 类型在所有表单字段的公共能力之外又提供了以下属性和方法。

- add(newOption, relOption)：在 relOption 之前向控件中添加新的<option>。
- multiple：布尔值，表示是否允许多选，等价于 HTML 的 multiple 属性。
- options：控件中所有<option>元素的 HTMLCollection。
- remove(index)：移除给定位置的选项。
- selectedIndex：选中项基于 0 的索引值，如果没有选中项则为–1。对于允许多选的列表，始终是第一个选项的索引。
- size：选择框中可见的行数，等价于 HTML 的 size 属性。

选择框的 type 属性可能是"select-one"或"select-multiple"，具体取决于 multiple 属性是否存在。当前选中项根据以下规则决定选择框的 value 属性。

- 如果没有选中项，则选择框的值是空字符串。
- 如果有一个选中项，且其 value 属性有值，则选择框的值就是选中项 value 属性的值。即使 value 属性的值是空字符串也是如此。
- 如果有一个选中项，且其 value 属性没有指定值，则选择框的值是该项的文本内容。
- 如果有多个选中项，则选择框的值根据前两条规则取得第一个选中项的值。

来看下面的选择框：

```
<select name="location" id="selLocation">
  <option value="Sunnyvale, CA">Sunnyvale</option>
  <option value="Los Angeles, CA">Los Angeles</option>
  <option value="Mountain View, CA">Mountain View</option>
  <option value="">China</option>
  <option>Australia</option>
</select>
```

如果选中这个选择框中的第一项，则字段的值就是"Sunnyvale, CA"。如果文本为"China"的项被选中，则字段的值是一个空字符串，因为该项的 value 属性是空字符串。如果选中最后一项，那么字段的值是"Australia"，因为该<option>元素没有指定 value 属性。

每个<option>元素在 DOM 中都由一个 HTMLOptionElement 对象表示。HTMLOptionElement 类型为方便数据存取添加了以下属性。

- index：选项在 options 集合中的索引。

- label：选项的标签，等价于 HTML 的 label 属性。
- selected：布尔值，表示是否选中了当前选项。把这个属性设置为 true 会选中当前选项。
- text：选项的文本。
- value：选项的值（等价于 HTML 的 value 属性）。

大多数<option>属性是为了方便存取选项数据。可以使用常规 DOM 功能存取这些信息，只是效率比较低，如下面的例子所示：

```
let selectbox = document.forms[0].elements["location"];

// 不推荐
let text = selectbox.options[0].firstChild.nodeValue;      // 选项文本
let value = selectbox.options[0].getAttribute("value");    // 选项值
```

以上代码使用标准的 DOM 技术获取了选择框中第一个选项的文本和值。下面再比较一下使用特殊选项属性的代码：

```
let selectbox = document.forms[0].elements["location"];

// 推荐
let text = selectbox.options[0].text;      // 选项文本
let value = selectbox.options[0].value;    // 选项值
```

在操作选项时，最好使用特定于选项的属性，因为这些属性得到了跨浏览器的良好支持。在操作 DOM 节点时，与表单控制实际的交互可能会因浏览器而异。不推荐使用标准 DOM 技术修改<option>元素的文本和值。

最后强调一下，选择框的 change 事件与其他表单字段是不一样的。其他表单字段会在自己的值改变后触发 change 事件，然后字段失去焦点。而选择框会在选中一项时立即触发 change 事件。

> **注意**　不同浏览器返回的 value 属性可能会有差异。JavaScript 中的 value 属性始终等于 HTML 中的 value 属性。但在 HTML 中没有指定 value 属性的情况下，IE8 及早期版本会返回空字符串，而 IE9 及之后版本、Safari、Firefox、Chrome 和 Opera 会返回与 text 相同的值。

19.3.1　选项处理

对于只允许选择一项的选择框，获取选项最简单的方式是使用选择框的 selectedIndex 属性，如下面的例子所示：

```
let selectedOption = selectbox.options[selectbox.selectedIndex];
```

这样可以获取关于选项的所有信息，比如：

```
let selectedIndex = selectbox.selectedIndex;
let selectedOption = selectbox.options[selectedIndex];
console.log(`Selected index: ${selectedIndex}\n` +
            `Selected text: ${selectedOption.text}\n` +
            `Selected value: ${selectedOption.value}`);
```

以上代码打印出了选中项的索引及其文本和值。

对于允许多选的选择框，selectedIndex 属性就像只允许选择一项一样。设置 selectedIndex

会移除所有选项，只选择指定的项，而获取 selectedIndex 只会返回选中的第一项的索引。

选项还可以通过取得选项的引用并将其 selected 属性设置为 true 来选中。例如，以下代码会选中选择框中的第一项：

```
selectbox.options[0].selected = true;
```

与 selectedIndex 不同，设置选项的 selected 属性不会在多选时移除其他选项，从而可以动态选择任意多个选项。如果修改单选框中选项的 selected 属性，则其他选项会被移除。要注意的是，把 selected 属性设置为 false 对单选框没有影响。

通过 selected 属性可以确定选择框中哪个选项被选中。要取得所有选中项，需要循环选项集合逐一检测 selected 属性，比如：

```
function getSelectedOptions(selectbox){
  let result = new Array();

  for (let option of selectbox.options) {
    if (option.selected) {
      result.push(option);
    }
  }

  return result;
}
```

这个函数会返回给定选择框中所有选中项的数组。首先创建一个包含结果的数组，然后通过 for 循环迭代所有选项，检测每个选项的 selected 属性。如果选项被选中，就将其添加到 result 数组。最后是返回选中项数组。这个 getSelectedOptions()函数可以用于获取选中项的信息，比如：

```
let selectbox = document.getElementById("selLocation");
let selectedOptions = getSelectedOptions(selectbox);
let message = "";

for (let option of selectedOptions) {
  message += 'Selected index: ${option.index}\n' +
             'Selected text: ${option.text}\n' +
             'Selected value: ${option.value}\n'
}

console.log(message);
```

这个例子先检索了一个选择框的所有选中项。然后通过 for 循环构建包含所有选中项信息的字符串，包括每项的索引、文本和值。以上代码既适用于单选框也适用于多选框。

19.3.2 添加选项

可以使用 JavaScript 动态创建选项并将它们添加到选择框。首先，可以使用 DOM 方法，如下所示：

```
let newOption = document.createElement("option");
newOption.appendChild(document.createTextNode("Option text"));
newOption.setAttribute("value", "Option value");

selectbox.appendChild(newOption);
```

以上代码创建了一个新的<option>元素，使用文本节点添加文本，设置其 value 属性，然后将其

添加到选择框。添加到选择框之后，新选项会立即显示出来。

另外，也可以使用 Option 构造函数创建新选项，这个构造函数是 DOM 出现之前就已经得到浏览器支持的。Option 构造函数接收两个参数：text 和 value，其中 value 是可选的。虽然这个构造函数通常会创建 Object 的实例，但 DOM 合规的浏览器都会返回一个<option>元素。这意味着仍然可以使用 appendChild()方法把这样创建的选项添加到选择框。比如下面的例子：

```
let newOption = new Option("Option text", "Option value");
selectbox.appendChild(newOption);    // 在 IE8 及更低版本中有问题
```

这个方法在除 IE8 及更低版本之外的所有浏览器中都没有问题。由于实现问题，IE8 及更低版本在这种情况下不能正确设置新选项的文本。

另一种添加新选项的方式是使用选择框的 add()方法。DOM 规定这个方法接收两个参数：要添加的新选项和要添加到其前面的参考选项。如果想在列表末尾添加选项，那么第二个参数应该是 null。IE8 及更早版本对 add()方法的实现稍有不同，其第二个参数是可选的，如果要传入则必须是一个索引值，表示要在其前面添加新选项的选项。DOM 合规的浏览器要求必须传入第二个参数，因此在跨浏览器方法中不能只使用一个参数（IE9 是符合 DOM 规范的）。此时，传入 undefined 作为第二个参数可以保证在所有浏览器中都将选项添加到列表末尾。下面是一个例子：

```
let newOption = new Option("Option text", "Option value");
selectbox.add(newOption, undefined);    // 最佳方案
```

以上代码可以在所有版本的 IE 及 DOM 合规的浏览器中使用。如果不想在最后插入新选项，则应该使用 DOM 技术和 insertBefore()。

> **注意**　跟在 HTML 中一样，选项的值不是必需的。Option 构造函数也可以只接收一个参数（选项的文本）。

19.3.3　移除选项

与添加选项类似，移除选项的方法也不止一种。第一种方式是使用 DOM 的 removeChild()方法并传入要移除的选项，比如：

```
selectbox.removeChild(selectbox.options[0]);   // 移除第一项
```

第二种方式是使用选择框的 remove()方法。这个方法接收一个参数，即要移除选项的索引，比如：

```
selectbox.remove(0);   // 移除第一项
```

最后一种方式是直接将选项设置为等于 null。这同样也是 DOM 之前浏览器实现的方式。下面是一个例子：

```
selectbox.options[0] = null;     // 移除第一项
```

要清除选择框的所有选项，需要迭代所有选项并逐一移除它们，如下面例子所示：

```
function clearSelectbox(selectbox) {
  for (let option of selectbox.options) {
    selectbox.remove(0);
  }
}
```

这个函数可以逐一移除选择框中的每一项。因为移除第一项会自动将所有选项向前移一位，所以这样就可以移除所有选项。

19.3.4 移动和重排选项

在 DOM 之前，从一个选择框向另一个选择框移动选项是非常麻烦的，要先从第一个选择框移除选项，然后以相同文本和值创建新选项，再将新选项添加到第二个选择框。DOM 方法则可以直接将某个选项从第一个选择框移动到第二个选择框，只要对相应选项使用 `appendChild()`方法即可。如果给这个方法传入文档中已有的元素，则该元素会先从其父元素中移除，然后再插入指定位置。例如，下面的代码会从选择框中移除第一项并插入另一个选择框：

```
let selectbox1 = document.getElementById("selLocations1");
let selectbox2 = document.getElementById("selLocations2");

selectbox2.appendChild(selectbox1.options[0]);
```

移动选项和移除选项都会导致每个选项的 `index` 属性重置。

重排选项非常类似，DOM 方法同样是最佳途径。要将选项移动到选择框中的特定位置，`insertBefore()`方法是最合适的。不过，要把选项移动到最后，还是 `appendChild()`方法比较方便。下面的代码演示了将一个选项在选择框中前移一个位置：

```
let optionToMove = selectbox.options[1];
selectbox.insertBefore(optionToMove,
                       selectbox.options[optionToMove.index-1]);
```

这个例子首先获得要移动选项的索引，然后将其插入之前位于它前面的选项之前，其中第二行代码适用于除第一个选项之外的所有选项。下面的代码则可以将选项向下移动一个位置：

```
let optionToMove = selectbox.options[1];
selectbox.insertBefore(optionToMove,
                       selectbox.options[optionToMove.index+2]);
```

以上代码适用于选择框中的所有选项，包括最后一个。

19.4 表单序列化

随着 Ajax（第 21 章会进一步讨论）的崭露头角，**表单序列化**（form serialization）已经成为一个常见需求。表单在 JavaScript 中可以使用表单字段的 `type` 属性连同其 `name` 属性和 `value` 属性来进行序列化。在写代码之前，我们需要理解浏览器如何确定在提交表单时要把什么发送到服务器。

- 字段名和值是 URL 编码的并以和号（`&`）分隔。
- 禁用字段不会发送。
- 复选框或单选按钮只在被选中时才发送。
- 类型为`"reset"`或`"button"`的按钮不会发送。
- 多选字段的每个选中项都有一个值。
- 通过点击提交按钮提交表单时，会发送该提交按钮；否则，不会发送提交按钮。类型为`"image"`的`<input>`元素视同提交按钮。
- `<select>`元素的值是被选中`<option>`元素的 `value` 属性。如果`<option>`元素没有 `value` 属性，则该值是它的文本。

表单序列化通常不包含任何按钮，因为序列化得到的字符串很可能以其他方式提交。除此之外其他规则都应该遵循。最终完成表单序列化的代码如下：

```
function serialize(form) {
  let parts = [];
  let optValue;

  for (let field of form.elements) {
    switch(field.type) {
      case "select-one":
      case "select-multiple":
        if (field.name.length) {
        for (let option of field.options) {
            if (option.selected) {
              if (option.hasAttribute){
                optValue = (option.hasAttribute("value") ?
                            option.value : option.text);
              } else {
                optValue = (option.attributes["value"].specified ?
                            option.value : option.text);
              }
              parts.push(encodeURIComponent(field.name)} + "=" +
                         encodeURIComponent(optValue));
            }
          }
        }
        break;
      case undefined:     // 字段集
      case "file":        // 文件输入
      case "submit":      // 提交按钮
      case "reset":       // 重置按钮
      case "button":      // 自定义按钮
        break;
      case "radio":       // 单选按钮
      case "checkbox":    // 复选框
        if (!field.checked) {
          break;
        }
      default:
        // 不包含没有名字的表单字段
        if (field.name.length) {
          parts.push('${encodeURIComponent(field.name)}=' +
                     '${encodeURIComponent(field.value)}');
        }
  }
  return parts.join("&");
}
```

这个 `serialize()`函数一开始定义了一个名为 `parts` 的数组，用于保存要创建字符串的各个部分。接下来通过 `for` 循环迭代每个表单字段，将字段保存在 `field` 变量中。获得一个字段的引用后，再通过 `switch` 语句检测其 `type` 属性。最麻烦的是序列化`<select>`元素，包括单选和多选两种模式。在遍历选择框的每个选项时，只要有选项被选中，就将其添加到结果字符串。单选控件只会有一个选项被选中，多选控件则可能有零或多个选项被选中。同样的代码适用于两种选择类型，因为浏览器会限制可选项的数量。找到选中项时，需要确定使用哪个值。如果不存在 `value` 属性，则应该以选项文本代替，不过 `value` 属性为空字符串是完全有效的。为此需要使用 DOM 合规的浏览器支持的 `hasAttribute()`

方法，而在 IE8 及更早版本中要使用值的 specified 属性。

表单中如果有<fieldset>元素，它就会出现在元素集合中，但应该没有 type 属性。因此，如果 type 属性是 undefined，则不必纳入序列化。各种类型的按钮以及文件输入字段也是如此。（文件输入字段在提交表单时包含文件的内容，但这些字段通常无法转换，因而也要排除在序列化之外。）对于单选按钮和复选框，会检测其 checked 属性。如果值为 false 就退出 switch 语句；如果值为 true，则继续执行 default 分支，将字段的名和值编码后添加到 parts 数组。注意，所有没有名字的表单字段都不会包含在序列化结果中以模拟浏览器的表单提交行为。这个函数的最后一步是使用 join()通过和号把所有字段的名值对拼接起来。

serialize()函数返回的结果是查询字符串格式。如果想要返回其他格式，修改起来也很简单。

19.5 富文本编辑

在网页上编写富文本内容是 Web 应用开发中很常见的需求。富文本编辑也就是所谓的“所见即所得”（WYSIWYG，What You See Is What You Get）编辑。虽然没有规范定义，但源自 IE 的一套事实标准已经被 Opera、Safari、Chrome 和 Firefox 所支持。基本的技术就是在空白 HTML 文件中嵌入一个 iframe。通过 designMode 属性，可以将这个空白文档变成可以编辑的，实际编辑的则是<body>元素的 HTML。designMode 属性有两个可能的值："off"（默认值）和"on"。设置为"on"时，整个文档都会变成可以编辑的（显示插入光标），从而可以像使用文字处理程序一样编辑文本，通过键盘将文本标记为粗体、斜体，等等。

作为 iframe 源的是一个非常简单的空白 HTML 页面。下面是一个例子：

```
<!DOCTYPE html>
<html>
  <head>
    <title>Blank Page for Rich Text Editing</title>
  </head>
  <body>
  </body>
</html>
```

这个页面会像其他任何页面一样加载到 iframe 里。为了可以编辑，必须将文档的 designMode 属性设置为"on"。不过，只有在文档完全加载之后才可以设置。在这个包含页面内，需要使用 onload 事件处理程序在适当时机设置 designMode，如下面的例子所示：

```
<iframe name="richedit" style="height: 100px; width: 100px"></iframe>

<script>
  window.addEventListener("load", () => {
    frames["richedit"].document.designMode = "on";
  });
</script>
```

以上代码加载之后，可以在页面上看到一个类似文本框的区域。这个框的样式具有网页默认样式，不过可以通过 CSS 调整。

19.5.1 使用 contenteditable

还有一种处理富文本的方式，也是 IE 最早实现的，即指定 contenteditable 属性。可以给页面

中的任何元素指定 contenteditable 属性，然后该元素会立即被用户编辑。这种方式更受欢迎，因为不需要额外的 iframe、空页面和 JavaScript，只给元素添加一个 contenteditable 属性即可，比如：

```
<div class="editable" id="richedit" contenteditable></div>
```

元素中包含的任何文本都会自动被编辑，元素本身类似于<textarea>元素。通过设置 contentEditable 属性，也可以随时切换元素的可编辑状态：

```
let div = document.getElementById("richedit");
richedit.contentEditable = "true";
```

contentEditable 属性有 3 个可能的值："true"表示开启，"false"表示关闭，"inherit"表示继承父元素的设置（因为在 contenteditable 元素内部会创建和删除元素）。IE、Firefox、Chrome、Safari 和 Opera 及所有主流移动浏览器都支持 contentEditable 属性。

> **注意**　contenteditable 是一个非常多才多艺的属性。比如，访问伪 URL data:text/html, <html contenteditable>可以把浏览器窗口转换为一个记事本。这是因为这样会临时创建 DOM 树并将整个文档变成可编辑区域。

19.5.2　与富文本交互

与富文本编辑器交互的主要方法是使用 document.execCommand()。这个方法在文档上执行既定的命令，可以实现大多数格式化任务。document.execCommand()可以接收 3 个参数：要执行的命令、表示浏览器是否为命令提供用户界面的布尔值和执行命令必需的值（如果不需要则为 null）。为跨浏览器兼容，第二个参数应该始终为 false，因为 Firefox 会在其为 true 时抛出错误。

不同浏览器支持的命令也不一样。下表列出了最常用的命令。

命　令	值（第三个参数）	说　明
backcolor	颜色字符串	设置文档背景颜色
bold	null	切换选中文本的粗体样式
copy	null	将选中文本复制到剪贴板
createlink	URL 字符串	将当前选中文本转换为指向给定 URL 的链接
cut	null	将选中文本剪切到剪贴板
delete	null	删除当前选中的文本
fontname	字体名	将选中文本改为使用指定字体
fontsize	1~7	将选中文本改为指定字体大小
forecolor	颜色字符串	将选中文本改为指定颜色
formatblock	HTML 标签，如<h1>	将选中文本包含在指定的 HTML 标签中
indent	null	缩进文本
inserthorizontalrule	null	在光标位置插入<hr>元素
insertimage	图片 URL	在光标位置插入图片
insertorderedlist	null	在光标位置插入 元素
insertparagraph	null	在光标位置插入<p>元素

（续）

命　令	值（第三个参数）	说　明
insertunorderedlist	null	在光标位置插入 元素
italic	null	切换选中文本的斜体样式
justifycenter	null	在光标位置居中文本块
justifyleft	null	在光标位置左对齐文本块
outdent	null	减少缩进
paste	null	在选中文本上粘贴剪贴板内容
removeformat	null	移除包含光标所在位置块的 HTML 标签。这是 formatblock 的反操作
selectall	null	选中文档中所有文本
underline	null	切换选中文本的下划线样式
unlink	null	移除文本链接。这是 createlink 的反操作

剪贴板相关的命令与浏览器关系密切。虽然这些命令并不都可以通过 document.execCommand() 使用，但相应的键盘快捷键都是可以用的。

这些命令可以用于修改内嵌窗格（iframe）中富文本区域的外观，如下面的例子所示：

```
// 在内嵌窗格中切换粗体文本样式
frames["richedit"].document.execCommand("bold", false, null);

// 在内嵌窗格中切换斜体文本样式
frames["richedit"].document.execCommand("italic", false, null);

// 在内嵌窗格中创建指向 www.wrox.com 的链接
frames["richedit"].document.execCommand("createlink", false,
                                        "http://www.wrox.com");

// 在内嵌窗格中为内容添加<h1>标签
frames["richedit"].document.execCommand("formatblock", false, "<h1>");
```

同样的方法也可以用于页面中添加了 contenteditable 属性的元素，只不过要使用当前窗口而不是内嵌窗格中的 document 对象：

```
// 切换粗体文本样式
document.execCommand("bold", false, null);

// 切换斜体文本样式
document.execCommand("italic", false, null);

// 创建指向 www.wrox.com 的链接
document.execCommand("createlink", false, "http://www.wrox.com");

// 为内容添加<h1>标签
document.execCommand("formatblock", false, "<h1>");
```

注意，即使命令是所有浏览器都支持的，命令生成的 HTML 通常差别也很大。例如，为选中文本应用 bold 命令在 IE 和 Opera 中会使用标签，在 Safari 和 Chrome 中会使用<b>标签，而在 Firefox 中会使用标签。在富文本编辑中，不能依赖浏览器生成的 HTML，因为命令实现和格式转换都是通过 innerHTML 完成的。