对于<input>和<button>元素，可以动态修改其 type 属性。但<select>元素的 type 属性是只读的。

2. 表单字段的公共方法

每个表单字段都有两个公共方法：focus()和 blur()。focus()方法把浏览器焦点设置到表单字段，这意味着该字段会变成活动字段并可以响应键盘事件。例如，文本框在获得焦点时会在内部显示闪烁的光标，表示可以接收输入。focus()方法主要用来引起用户对页面中某个部分的注意。比如，在页面加载后把焦点定位到表单中第一个字段就是很常见的做法。实现方法是监听 load 事件，然后在第一个字段上调用 focus()，如下所示：

```
window.addEventListener("load", (event) => {
  document.forms[0].elements[0].focus();
});
```

注意，如果表单中第一个字段是 type 为"hidden"的<input>元素，或者该字段被 CSS 属性 display 或 visibility 隐藏了，以上代码就会出错。

HTML5 为表单字段增加了 autofocus 属性，支持的浏览器会自动为带有该属性的元素设置焦点，而无须使用 JavaScript。比如：

```
<input type="text" autofocus>
```

为了让之前的代码在使用 autofocus 时也能正常工作，必须先检测元素上是否设置了该属性。如果设置了 autofocus，就不再调用 focus()：

```
window.addEventListener("load", (event) => {
  let element = document.forms[0].elements[0];

  if (element.autofocus !== true) {
    element.focus();
    console.log("JS focus");
  }
});
```

因为 autofocus 是布尔值属性，所以在支持的浏览器中通过 JavaScript 访问表单字段的 autofocus 属性会返回 true（在不支持的浏览器中是空字符串）。上面的代码只会在 autofocus 属性不等于 true 时调用 focus()方法，以确保向前兼容。大多数现代浏览器支持 autofocus 属性，只有 iOS Safari、Opera Mini 和 IE10 及以下版本不支持。

> **注意**　默认情况下只能给表单元素设置焦点。不过，通过将 tabIndex 属性设置为–1 再调用 focus()，也可以给任意元素设置焦点。只有 Opera 不支持这个技术。

focus()的反向操作是 blur()，其用于从元素上移除焦点。调用 blur()时，焦点不会转移到任何特定元素，仅仅只是从调用这个方法的元素上移除了。在浏览器支持 readonly 属性之前，Web 开发者通常会使用这个方法创建只读字段。现在很少有用例需要调用 blur()，不过如果需要是可以用的。下面是一个例子：

```
document.forms[0].elements[0].blur();
```

3. 表单字段的公共事件

除了鼠标、键盘、变化和 HTML 事件外，所有字段还支持以下 3 个事件。

- blur：在字段失去焦点时触发。
- change：在<input>和<textarea>元素的 value 发生变化且失去焦点时触发，或者在<select>元素中选中项发生变化时触发。
- focus：在字段获得焦点时触发。

blur 和 focus 事件会因为用户手动改变字段焦点或者调用 blur()或 focus()方法而触发。这两个事件对所有表单都会一视同仁。change 事件则不然，它会因控件不同而在不同时机触发。对于<input>和<textarea>元素，change 事件会在字段失去焦点，同时 value 自控件获得焦点后发生变化时触发。对于<select>元素，change 事件会在用户改变了选中项时触发，不需要控件失去焦点。

focus 和 blur 事件通常用于以某种方式改变用户界面，以提供可见的提示或额外功能（例如在文本框下面显示下拉菜单）。change 事件通常用于验证用户在字段中输入的内容。比如，有的文本框可能只限于接收数值。focus 事件可以用来改变控件的背景颜色以便更清楚地表明当前字段获得了焦点。blur 事件可以用于去掉这个背景颜色。而 change 事件可以用于在用户输入了非数值时把背景改为红色。以下代码展示了上述操作：

```
let textbox = document.forms[0].elements[0];
textbox.addEventListener("focus", (event) => {
  let target = event.target;
  if (target.style.backgroundColor != "red") {
    target.style.backgroundColor = "yellow";
  }
});

textbox.addEventListener("blur", (event) => {
  let target = event.target;
  target.style.backgroundColor = /[^\d]/.test(target.value) ? "red" : "";
});

textbox.addEventListener("change", (event) => {
  let target = event.target;
  target.style.backgroundColor = /[^\d]/.test(target.value) ? "red" : "";
});
```

这里的 onfocus 事件处理程序会把文本框的背景改为黄色，更清楚地表明它是当前活动字段。onblur 和 onchange 事件处理程序会在发现非数值字符时把背景改为红色。为测试非数值字符，这里使用了一个简单的正则表达式来检测文本框的 value。这个功能必须同时在 onblur 和 onchange 事件处理程序上实现，以确保无论文本框是否改变都能执行验证。

> **注意** blur 和 change 事件的关系并没有明确定义。在某些浏览器中，blur 事件会先于 change 事件触发；在其他浏览器中，触发顺序则相反。因此不能依赖这两个事件触发的顺序，必须区分时要多加注意。

19.2 文本框编程

在 HTML 中有两种表示文本框的方式：单行使用<input>元素，多行使用<textarea>元素。这两个控件非常相似，大多数时候行为也一样。不过，它们也有非常重要的区别。

默认情况下，<input>元素显示为文本框，省略 type 属性会以"text"作为默认值。然后可以通过

图灵社区会员 aSINKz(1561821892@qq.com) 专享 尊重版权

size 属性指定文本框的宽度，这个宽度是以字符数来计量的。而 value 属性用于指定文本框的初始值，maxLength 属性用于指定文本框允许的最多字符数。因此要创建一个一次可显示 25 个字符，但最多允许显示 50 个字符的文本框，可以这样写：

```
<input type="text" size="25" maxlength="50" value="initial value">
```

<textarea>元素总是会创建多行文本框。可以使用 rows 属性指定这个文本框的高度，以字符数计量；以 cols 属性指定以字符数计量的文本框宽度，类似于<input>元素的 size 属性。与<input>不同的是，<textarea>的初始值必须包含在<textarea>和</textarea>之间，如下所示：

```
<textarea rows="25" cols="5">initial value</textarea>
```

同样与<input>元素不同的是，<textarea>不能在 HTML 中指定最大允许的字符数。

除了标记中的不同，这两种类型的文本框都会在 value 属性中保存自己的内容。通过这个属性，可以读取也可以设置文本模式的值，如下所示：

```
let textbox = document.forms[0].elements["textbox1"];
console.log(textbox.value);

textbox.value = "Some new value";
```

应该使用 value 属性，而不是标准 DOM 方法读写文本框的值。比如，不要使用 setAttribute() 设置<input>元素 value 属性的值，也不要尝试修改<textarea>元素的第一个子节点。对 value 属性的修改也不会总体现在 DOM 中，因此在处理文本框值的时候最好不要使用 DOM 方法。

19.2.1　选择文本

两种文本框都支持一个名为 select()的方法，此方法用于全部选中文本框中的文本。大多数浏览器会在调用 select()方法后自动将焦点设置到文本框（Opera 例外）。这个方法不接收参数，可以在任何时候调用。下面来看一个例子：

```
let textbox = document.forms[0].elements["textbox1"];
textbox.select();
```

在文本框获得焦点时选中所有文本是非常常见的，特别是在文本框有默认值的情况下。这样做的出发点是让用户能够一次性删除所有默认内容。可以通过以下代码来实现：

```
textbox.addEventListener("focus", (event) => {
  event.target.select();
});
```

把以上代码应用到文本框之后，只要文本框一获得焦点就会自动选中其中的所有文本。这样可以极大提升表单易用性。

1. select 事件

与 select()方法相对，还有一个 select 事件。当选中文本框中的文本时，会触发 select 事件。这个事件确切的触发时机因浏览器而异。在 IE9+、Opera、Firefox、Chrome 和 Safari 中，select 事件会在用户选择完文本后立即触发；在 IE8 及更早版本中，则会在第一个字符被选中时触发。另外，调用 select()方法也会触发 select 事件。下面来看一个例子：

```
let textbox = document.forms[0].elements["textbox1"];

textbox.addEventListener("select", (event) => {
```

图灵社区会员 aSINKz(1561821892@qq.com) 专享 尊重版权

```
  console.log(`Text selected: ${textbox.value}`);
});
```

2. 取得选中文本

虽然 select 事件能够表明有文本被选中，但不能提供选中了哪些文本的信息。HTML5 对此进行了扩展，以方便更好地获取选中的文本。扩展为文本框添加了两个属性：selectionStart 和 selectionEnd。这两个属性包含基于 0 的数值，分别表示文本选区的起点和终点（文本选区起点的偏移量和文本选区终点的偏移量）。因此，要取得文本框中选中的文本，可以使用以下代码：

```
function getSelectedText(textbox){
  return textbox.value.substring(textbox.selectionStart,
                                 textbox.selectionEnd);
}
```

因为 substring()方法是基于字符串偏移量的，所以直接传入 selectionStart 和 selectionEnd 就可以取得选中的文本。

这个扩展在 IE9+、Firefox、Safari、Chrome 和 Opera 中都可以使用。IE8 及更早版本不支持这两个属性，因此需要使用其他方式。

老版本 IE 中有一个包含整个文档中文本选择信息的 document.selection 对象。这意味着无法确定选中的文本在页面中的什么位置。不过，在与 select 事件一起使用时，可以确定是触发这个事件文本框中选中的文本。为取得这些选中的文本，必须先创建一个范围，然后再从中提取文本，如下所示：

```
function getSelectedText(textbox){
  if (typeof textbox.selectionStart == "number"){
    return textbox.value.substring(textbox.selectionStart,
                                   textbox.selectionEnd);
  } else if (document.selection){
    return document.selection.createRange().text;
  }
}
```

这个修改后的函数兼容在 IE 老版本中取得选中文本。注意 document.selection 是根本不需要 textbox 参数的。

3. 部分选中文本

HTML5 也为在文本框中选择部分文本提供了额外支持。现在，除了 select()方法之外，Firefox 最早实现的 setSelectionRange()方法也可以在所有文本框中使用。这个方法接收两个参数：要选择的第一个字符的索引和停止选择的字符的索引（与字符串的 substring()方法一样）。下面是几个例子：

```
textbox.value = "Hello world!"

// 选择所有文本
textbox.setSelectionRange(0, textbox.value.length);  // "Hello world!"

// 选择前 3 个字符
textbox.setSelectionRange(0, 3);   // "Hel"

// 选择第 4~6 个字符
textbox.setSelectionRange(4, 7);   // "o w"
```

如果想看到选择，则必须在调用 setSelectionRange()之前或之后给文本框设置焦点。这个方法在 IE9、Firefox、Safari、Chrome 和 Opera 中都可以使用。

IE8 及更早版本支持通过范围部分选中文本。这也就是说，要选择文本框中的部分文本，必须先使用 IE 在文本框上提供的 createTextRange()方法创建一个范围，并使用 moveStart()和 moveEnd()范围方法把这个范围放到正确的位置上。不过，在调用这两个方法前需要先调用 collapse()方法把范围折叠到文本框的开始。接着，moveStart()可以把范围的起点和终点都移动到相同的位置，再给 moveEnd()传入要选择的字符总数作为参数。最后一步是使用范围的 select()方法选中文本，如下面的例子所示：

```
textbox.value = "Hello world!";

var range = textbox.createTextRange();

// 选择所有文本
range.collapse(true);
range.moveStart("character", 0);
range.moveEnd("character", textbox.value.length);   // "Hello world!"
range.select();

// 选择前 3 个字符
range.collapse(true);
range.moveStart("character", 0);
range.moveEnd("character", 3);
range.select();  // "Hel"

// 选择第 4~6 个字符
range.collapse(true);
range.moveStart("character", 4);
range.moveEnd("character", 6);
range.select();  // "o w"
```

与其他浏览器一样，如果想要看到选中的效果，则必须让文本框获得焦点。

部分选中文本对自动完成建议项等高级文本输入框是很有用的。

19.2.2　输入过滤

不同文本框经常需要保证输入特定类型或格式的数据。或许数据需要包含特定字符或必须匹配某个特定模式。由于文本框默认并未提供什么验证功能，因此必须通过 JavaScript 来实现这种**输入过滤**。组合使用相关事件及 DOM 能力，可以把常规的文本框转换为能够理解自己所收集数据的智能输入框。

1. 屏蔽字符

有些输入框需要出现或不出现特定字符。例如，让用户输入手机号的文本框就不应该出现非数字字符。我们知道 keypress 事件负责向文本框插入字符，因此可以通过阻止这个事件的默认行为来屏蔽非数字字符。比如，下面的代码会屏蔽所有按键的输入：

```
textbox.addEventListener("keypress", (event) => {
  event.preventDefault();
});
```

运行以上代码会让文本框变成只读，因为所有按键都被屏蔽了。如果想只屏蔽特定字符，则需要检查事件的 charCode 属性，以确定正确的回应方式。例如，下面就是只允许输入数字的代码：

```
textbox.addEventListener("keypress", (event) => {
  if (!/\d/.test(String.fromCharCode(event.charCode))){
    event.preventDefault();
```

图灵社区会员 aSINKz(1561821892@qq.com) 专享 尊重版权

```
  }
});
```

这个例子先用String.fromCharCode()把事件的charCode转换为字符串，再用正则表达式/\d/来测试。这个正则表达式匹配所有数字字符，如果测试失败就调用 preventDefault()屏蔽事件默认行为。这样就可以让文本框忽略非数字输入。

虽然 keypress 事件应该只在按下字符键时才触发，但某些浏览器会在按下其他键时也触发这个事件。Firefox 和 Safari（3.1 之前）会在按下上、下箭头键、退格键和删除键时触发 keypress 事件。Safari 3.1 及之后版本对这些键则不会再触发 keypress 事件。这意味着简单地屏蔽所有非数字字符还不够好，因为这样也屏蔽了上述这些非常有用的且必要的键。好在我们可以轻松检测到是否按下了这些键。在 Firefox 中，所有触发 keypress 事件的非字符键的 charCode 都是 0，而在 Safari 3 之前这些键的 charCode 都是 8。综合考虑这些情况，就是不能屏蔽 charCode 小于 10 的键。为此，上面的函数可以改进为：

```
textbox.addEventListener("keypress", (event) => {
  if (!/\d/.test(String.fromCharCode(event.charCode)) &&
      event.charCode > 9){
    event.preventDefault();
  }
});
```

这个事件处理程序可以在所有浏览器中使用，屏蔽非数字字符但允许同样会触发 keypress 事件的所有基础按键。

还有一个问题需要处理：复制、粘贴及涉及 Ctrl 键的其他功能。在除 IE 外的所有浏览器中，前面代码会屏蔽快捷键 Ctrl+C、Ctrl+V 及其他使用 Ctrl 的组合键。因此，最后一项检测是确保没有按下 Ctrl 键，如下面的例子所示：

```
textbox.addEventListener("keypress", (event) => {
  if (!/\d/.test(String.fromCharCode(event.charCode)) &&
      event.charCode > 9 &&
      !event.ctrlKey){
    event.preventDefault();
  }
});
```

最后这个改动可以确保所有默认的文本框行为不受影响。这个技术可以用来自定义是否允许在文本框中输入某些字符。

2. 处理剪贴板

IE 是第一个支持剪贴板相关事件及通过 JavaScript 访问剪贴板数据的浏览器。IE 的实现成为了事实标准，这是因为 Safari、Chrome、Opera 和 Firefox 都实现了相同的事件和剪贴板访问机制，后来 HTML5 也增加了剪贴板事件 。以下是与剪贴板相关的 6 个事件。

- beforecopy：复制操作发生前触发。
- copy：复制操作发生时触发。
- beforecut：剪切操作发生前触发。
- cut：剪切操作发生时触发。
- beforepaste：粘贴操作发生前触发。
- paste：粘贴操作发生时触发。

这是一个比较新的控制剪贴板访问的标准，事件的行为及相关对象会因浏览器而异。在 Safari、Chrome 和 Firefox 中，beforecopy、beforecut 和 beforepaste 事件只会在显示文本框的上下文菜单（预期会发生剪贴板事件）时触发，但 IE 不仅在这种情况下触发，也会在 copy、cut 和 paste 事件之前触发。无论是在上下文菜单中做出选择还是使用键盘快捷键，copy、cut 和 paste 事件在所有浏览器中都会按预期触发。

通过 beforecopy、beforecut 和 beforepaste 事件可以在向剪贴板发送或从中检索数据前修改数据。不过，取消这些事件并不会取消剪贴板操作。要阻止实际的剪贴板操作，必须取消 copy、cut 和 paste 事件。

剪贴板上的数据可以通过 window 对象（IE）或 event 对象（Firefox、Safari 和 Chrome）上的 clipboardData 对象来获取。在 Firefox、Safari 和 Chrome 中，为防止未经授权访问剪贴板，只能在剪贴板事件期间访问 clipboardData 对象；IE 则在任何时候都会暴露 clipboardData 对象。为了跨浏览器兼容，最好只在剪贴板事件期间使用这个对象。

clipboardData 对象上有 3 个方法：getData()、setData()和 clearData()，其中 getData()方法从剪贴板检索字符串数据，并接收一个参数，该参数是要检索的数据的格式。IE 为此规定了两个选项："text"和"URL"。Firefox、Safari 和 Chrome 则期待 MIME 类型，不过会将"text"视为等价于"text/plain"。

setData()方法也类似，其第一个参数用于指定数据类型，第二个参数是要放到剪贴板上的文本。同样，IE 支持"text"和"URL"，Safari 和 Chrome 则期待 MIME 类型。不过，与 getData()不同的是，Safari 和 Chrome 不认可"text"类型。只有在 IE8 及更早版本中调用 setData()才有效，其他浏览器会忽略对这个方法的调用。为抹平差异，可以使用以下跨浏览器的方法：

```
function getClipboardText(event){
  var clipboardData =  (event.clipboardData || window.clipboardData);
  return clipboardData.getData("text");
}

function setClipboardText (event, value){
  if (event.clipboardData){
    return event.clipboardData.setData("text/plain", value);
  } else if (window.clipboardData){
    return window.clipboardData.setData("text", value);
  }
}
```

这里的 getClipboardText()函数相对简单，它只需要知道 clipboardData 对象在哪里，然后便可以通过"text"类型调用 getData()。相应的，setClipboardText()函数则要复杂一些。在确定 clipboardData 对象的位置之后，需要根据实现以相应的类型（Firefox、Safari 和 Chrome 是"text/plain"，而 IE 是"text"）调用 setData()。

如果文本框期待某些字符或某种格式的文本，那么从剪贴板中读取文本是有帮助的。比如，如果文本框只允许输入数字，那么就必须检查粘贴过来的值，确保其中只包含数字。在 paste 事件中，可以确定剪贴板上的文本是否无效，如果无效就取消默认行为，如下面的例子所示：

```
textbox.addEventListener("paste", (event) => {
  let text = getClipboardText(event);

  if (!/^\d*$/.test(text)){
    event.preventDefault();
```

```
    }
});
```

这个 onpaste 事件处理程序确保只有数字才能粘贴到文本框中。如果剪贴板中的值不符合指定模式，则取消粘贴操作。Firefox、Safari 和 Chrome 只允许在 onpaste 事件处理程序中访问 getData() 方法。

因为不是所有浏览器都支持剪贴板访问，所以有时候更容易屏蔽一个或多个剪贴板操作。在支持 copy、cut 和 paste 事件的浏览器（IE、Safari、Chrome 和 Firefox）中，很容易阻止事件的默认行为。在 Opera 中，则需要屏蔽导致相应事件的按键，同时阻止显示相应的上下文菜单。

19.2.3 自动切换

JavaScript 可以通过很多方式来增强表单字段的易用性。最常用的是在当前字段完成时自动切换到下一个字段。对于要收集数据的长度已知（比如电话号码）的字段是可以这样处理的。在美国，电话号码通常分为 3 个部分：区号、交换局号，外加 4 位数字。在网页中，可以通过 3 个文本框来表示这几个部分，比如：

```
<input type="text" name="tel1" id="txtTel1" maxlength="3">
<input type="text" name="tel2" id="txtTel2" maxlength="3">
<input type="text" name="tel3" id="txtTel3" maxlength="4">
```

为增加这个表单的易用性并加速数据输入，可以在每个文本框输入到最大允许字符数时自动把焦点切换到下一个文本框。因此，当用户在第一个文本框中输入 3 个字符后，就把焦点移到第二个文本框，当用户在第二个文本框中输入 3 个字符后，把焦点再移到第三个文本框。这种自动切换文本框的行为可以通过如下代码实现：

```
<script>
  function tabForward(event){
    let target = event.target;

    if (target.value.length == target.maxLength){
      let form = target.form;

      for (let i = 0, len = form.elements.length; i < len; i++) {
        if (form.elements[i] == target) {
          if (form.elements[i+1]) {
            form.elements[i+1].focus();
          }
          return;
        }
      }
    }
  }

  let inputIds = ["txtTel1", "txtTel2", "txtTel3"];
  for (let id of inputIds) {
    let textbox = document.getElementById(id);
    textbox.addEventListener("keyup", tabForward);
  }

  let textbox1 = document.getElementById("txtTel1");
  let textbox2 = document.getElementById("txtTel2");
  let textbox3 = document.getElementById("txtTel3");
</script>
```

这个 tabForward()函数是实现自动切换的关键。它通过比较用户输入文本的长度与 maxlength 属性的值来检测输入是否达到了最大长度。如果两者相等（因为浏览器会强制最大字符数，所以不可能出现多的情况），那么就要通过循环表单中的元素集合找到当前文本框，并把焦点设置到下一个元素。这个函数接着给每一个文本框都指定了 onkeyup 事件处理程序。因为 keyup 事件会在每个新字符被插入到文本框中时触发，所以此时应该是检测文本框内容长度的最佳时机。在填写这个简单的表单时，用户不用按 Tab 键切换字段和提交表单。

不过要注意，上面的代码只适用于之前既定的标记，没有考虑可能存在的隐藏字段。

19.2.4　HTML5 约束验证 API

HTML5 为浏览器新增了在提交表单前验证数据的能力。这些能力实现了基本的验证，即使 JavaScript 不可用或加载失败也没关系。这是因为浏览器自身会基于指定的规则进行验证，并在出错时显示适当的错误消息（无须 JavaScript）。这些能力只有支持 HTML5 这部分的浏览器才有，包括所有现代浏览器（除了 Safari）和 IE10+。

验证会根据某些条件应用到表单字段。可以使用 HTML 标记指定对特定字段的约束，然后浏览器会根据这些约束自动执行表单验证。

1. 必填字段

第一个条件是给表单字段添加 required 属性，如下所示：

```
<input type="text" name="username" required>
```

任何带有 required 属性的字段都必须有值，否则无法提交表单。这个属性适用于<input>、<textarea>和<select>字段（Opera 直到版本 11 都不支持<select>的 required 属性）。可以通过 JavaScript 检测对应元素的 required 属性来判断表单字段是否为必填：

```
let isUsernameRequired = document.forms[0].elements["username"].required;
```

还可以使用下面的代码检测浏览器是否支持 required 属性：

```
let isRequiredSupported = "required" in document.createElement("input");
```

这行代码使用简单的特性检测来确定新创建的<input>元素上是否存在 required 属性。

注意，不同浏览器处理必填字段的机制不同。Firefox、Chrome、IE 和 Opera 会阻止表单提交并在相应字段下面显示有帮助信息的弹框，而 Safari 什么也不做，也不会阻止提交表单。

2. 更多输入类型

HTML5 为<input>元素增加了几个新的 type 值。这些类型属性不仅表明了字段期待的数据类型，而且也提供了一些默认验证，其中两个新的输入类型是已经得到广泛支持的"email"和"url"，二者都有浏览器提供的自定义验证。比如：

```
<input type="email" name="email">
<input type="url" name="homepage">
```

"email"类型确保输入的文本匹配电子邮件地址，而"url"类型确保输入的文本匹配 URL。注意，浏览器在匹配模式时都存在问题。最明显的是文本"-@-"会被认为是有效的电子邮件地址。浏览器厂商仍然在解决这些问题。

要检测浏览器是否支持这些新类型，可以在 JavaScript 中新创建一个输入元素并将其类型属性设置为"email"或"url"，然后再读取该元素的值。老版本浏览器会自动将未知类型值设置为"text"，而支

持的浏览器会返回正确的值。比如：

```
let input = document.createElement("input");
input.type = "email";
let isEmailSupported = (input.type == "email");
```

对于这两个新类型，除非应用了 required 属性，否则空字段是有效的。另外，指定一个特殊输入类型并不会阻止用户输入无效的值。新类型只是会应用一些默认验证。

3. 数值范围

除了"email"和"url"，HTML5 还定义了其他几种新的输入元素类型，它们都是期待某种数值输入的，包括："number"、"range"、"datetime"、"datetime-local"、"date"、"month"、"week"和"time"。并非所有主流浏览器都支持这些类型，因此使用时要当心。浏览器厂商目前正致力于解决兼容性问题和提供更逻辑化的功能。本节内容更多地是介绍未来趋势，而不是讨论当前就能用的功能。

对上述每种数值类型，都可以指定 min 属性（最小可能值）、max 属性（最大可能值），以及 step 属性（从 min 到 max 的步长值）。例如，如果只允许输入 0 到 100 中 5 的倍数，那么可以这样写：

```
<input type="number" min="0" max="100" step="5" name="count">
```

根据浏览器的不同，可能会也可能不会出现旋转控件（上下按钮）用于自动增加和减少。

上面每个属性在 JavaScript 中也可以通过对应元素的 DOM 属性来访问和修改。此外，还有两个方法，即 stepUp()和 stepDown()。这两个方法都接收一个可选的参数：要从当前值加上或减去的数值。（默认情况下，步长值会递增或递减 1。）虽然浏览器还没有实现这些方法，但可以先看一下它们的用法：

```
input.stepUp();       // 加 1
input.stepUp(5);      // 加 5
input.stepDown();     // 减 1
input.stepDown(10);   // 减 10
```

4. 输入模式

HTML5 为文本字段新增了 pattern 属性。这个属性用于指定一个正则表达式，用户输入的文本必须与之匹配。例如，要限制只能在文本字段中输入数字，可以这样添加模式：

```
<input type="text" pattern="\d+" name="count">
```

注意模式的开头和末尾分别假设有^和$。这意味着输入内容必须从头到尾都严格与模式匹配。

与新增的输入类型一样，指定 pattern 属性也不会阻止用户输入无效内容。模式会应用到值，然后浏览器会知道值是否有效。通过访问 pattern 属性可以读取模式：

```
let pattern = document.forms[0].elements["count"].pattern;
```

使用如下代码可以检测浏览器是否支持 pattern 属性：

```
let isPatternSupported = "pattern" in document.createElement("input");
```

5. 检测有效性

使用 checkValidity()方法可以检测表单中任意给定字段是否有效。这个方法在所有表单元素上都可以使用，如果字段值有效就会返回 true，否则返回 false。判断字段是否有效的依据是本节前面提到的约束条件，因此必填字段如果没有值就会被视为无效，而字段值不匹配 pattern 属性也会被视为无效。比如：