

第11章 表单编程

在HTML开发中，有一个地方是我们无法绕过的，也是我们获取用户信息，与用户进行交互的最重要的地方，那就是表单。不管是用户的登录，还是用户的个人信息，甚至是我们常见的各种社交网站，都离不开表单的操作。本章就将介绍DOM中有关表单的相关操作。

本章将学习如下内容：

- 获取表单元素和表单字段
- 表单属性介绍
- 表单相关事件
- 表单验证
- HTML5表单控件
- 下拉列表和多选列表的使用
- 表单常见操作

11-1 获取表单元素和表单字段

表单由包含输入字段，选择菜单和按钮等表单控件的 `<form>` 元素组成。这些输入字段可以填充信息，一旦表单被提交了，这些信息就可以得到处理。

传统上，当表单被提交时，表单上的数据会被发送到服务器上面进行处理。一般来讲，服务器端上使用服务器端脚本语言进行处理，例如PHP或者Ruby等服务器脚本语言。

不过，在当年拨号上网的时代，将所有的表单信息验证都放在服务器端，用户体验是非常之低的。所以，当初JavaScript被设计出来其中也有为了解决这个问题。在将表单里面的信息发送到服务器之前，我们经常会通过JavaScript来处理表单中的信息。

但是在利用JavaScript来对表单信息进行处理之前，首先第一步就是要获取到表单以及表单的内容。

1. 什么是表单元素

在HTML中表单就是指form标签，它的作用是将用户输入或选择的数据提交给指定的服务器。比如：

```
<form action="#" method="post" enctype="application/x-www-form-urlencoded">
</form>
```

- action - 提交的地址
- method - 提交方式
- enctype - 数据传递的方式，这是默认的方式，即以键值对的形式提交。

2. 如何获取表单元素

获取表单元素的方式有很多：

```
<form id="form1" name="form1">
</form>
<script>
// 方法一：
let fm = document.getElementById("fm");
// 方法二：
let fm = document.forms[0];
// 方法三：
let fm = document.forms["form1"];// 其中的form1可以是id或name的值
// 方法四：
let fm = document.form1;// 其中的form1只能是name的值
</script>
```

3. 什么是表单字段(域)

首先表单字段应该包含在form元素中，但并不意味着form中的所有元素都是表单字段。实际上表单字段主要是指六个元素：

- input - 类型众多，主要是定义输入域
- textarea - 多行文本
- select - 定义下拉或多选列表
- fieldset - 将相关的表单用外框包含起来
- button - 默认带提交功能的按钮
- output - 显示输出结果

还有些元素虽然不属于表单字段，但是也具备一些和表单字段交互的功能：

- label - 为input元素定义标注
- datalist - 为input元素提供选项列表

4. input的type类型

input元素比较特殊，它有个属性type，可以将input呈现出不同的效果。如：

- text - 文本框
- password - 密码框

- radio - 单选框
- checkbox - 多选框
- file - 文件上传控件
- hidden - 隐藏表单
- submit - 提交按钮
- image - 带图片的提交按钮
- reset - 重置按钮
- button - 普通按钮

注意：submit、image和button标签都具备提交功能。

HTML5新增的type类型：

- email
- url
- number
- range
- date
- time
- datetime-local
- month
- week
- search
- tel
- color

后面小节会具体介绍HTML5新增的相关字段

5. 获取表单字段

获取表单字段的方法也有很多，如：

```
<form id="form1" name="form1">
  <input type="text" id="phone" name="phone" />
</form>
<script>
// 获取表单元素：
let fm = document.getElementById("fm");
// 方法一：
let field = document.getElementById("phone");
// 方法二：
let field = fm.elements[0]; // elements是获取表单元素中所有表单域对象，通过下标值访问
某一个表单字段
```

```
// 方法三：  
let field = fm.elements["phone"];// 其中的phone可以是id或name的值  
// 方法四：  
let field = fm.phone;// 其中的phone可以是id或name的值  
// 方法五：  
let field = fm["phone"];// 其中的phone可以是id或name的值  
// 方法六：  
let field = fm[0];// 其中的下标值表示表单字段在表单元素中的序号  
</script>
```

11-2 表单属性介绍

HTML中每个标签都有很多属性，不同功能的标签拥有的属性也是不一样的。比如表单相关的标签拥有name属性，该属性的作用可以在JS中利用它来对标签进行引用。同时还可以在表单提交后，对表单数据进行引用。而其他非表单标签就无此功能。下面总结一下表单元素中比较常见的属性：

注意：以下的属性并不是全部，只是表单特有且较为常见的属性，像id之类每个标签都有的属性就不在罗列范围内。更多的属性可以参考相应的API文档。

1. form

- action - 提交的地址
- autocomplete - 是否启用表单的自动完成功能，默认为启用(on)
- enctype - 数据传递的方式
- method - 提交方法
- name - 表单名
- target - 规定在何处打开action指定的地址

2. input

- autocomplete - 规定input元素输入字段是否应该启用自动完成功能。默认on
- autofocus - 规定当页面加载时input元素应该自动获得焦点
- checked - 规定type=checkbox/radio时是否为选中状态
- disabled - 禁用该元素
- list - 指向引用的datalist，值为datalist的id
- maxlength - 规定input元素中允许的最大字符数
- name - 表单字段的名称
- placeholder - 规定可描述输入input字段预期值的简短的提示信息
- readonly - 规定输入字段是只读的
- type - 规定要显示的input元素的类型
- value - 指定input元素value的值

3. textarea

- autofocus - 规定当页面加载时input元素应该自动获得焦点
- disabled - 禁用该元素
- maxlength - 规定input元素中允许的最大字符数
- name - 表单字段的名称

- placeholder - 规定可描述输入input字段预期值的简短的提示信息
- readonly - 规定输入字段是只读的
- rows - 规定文本区域内可见的行数
- cols - 规定文本区域内可见的列数

textarea是通过cols和rows属性来规定textarea的尺寸大小，不过更好的办法是使用CSS的height和width属性。

4. select

- autofocus - 规定当页面加载时input元素应该自动获得焦点
- disabled - 禁用该元素
- name - 表单字段的名称
- multiple - 当指定了该属性时下拉列表变多选列表

5. option

注意：option只能包含在select或datalist中。

- disabled - 规定此选项应在首次加载时被禁用
- selected - 规定选项（在首次显示在列表中时）表现为选中状态
- value - 定义送往服务器的选项值

6. button

- autofocus - 规定当页面加载时自动获得焦点
- disabled - 规定此选项应在首次加载时被禁用
- type - 只有三个值,button表示普通按钮；submit表示提交按钮；reset表示重置按钮；
- value - 按钮中的文本值，可以写在开始和结束标签之间

7. 特殊属性

上面介绍的属性中有些是比较特殊的，它们特殊之处在于可以不需要属性值，只需要有属性名即可生效。比如：

```
<form id="fm">
  <input type="button" value="ok" disabled>
  <input type="checkbox" name="language" checked>
</form>
```

当然也可以定义属性值，比如

```
<form id="fm">
  <input type="button" value="ok" disabled="disabled">
  <input type="checkbox" name="language" checked="true">
</form>
```

其实，它的值无论是什么都无所谓，只要有这个属性存在功能就会生效。不过如果是通过JS来设置这些属性，值就需要通过boolean类型来设定，true表示生效、false反之。

```
<form id="fm">
  <input type="button" id="diabileBtn" value="禁用/取消" >
  <input type="text" id="phone" >
</form>
<script>
  let phone = document.getElementById("phone");
  document.getElementById("diabileBtn").onclick = function(){
    phone.disabled = !phone.disabled;
  }
</script>
```

效果：点击按钮可以切换文本框的禁用状态

在表单元素中这类属性有：

- autofocus
- readonly
- disabled
- multiple
- checked
- selected

11-3 表单相关事件

表单元素中有很多专属的事件类型，比如提交、重置、获取焦点、失去焦点等等。下面咱们来具体看看这些事件的用法及特性。

1. 提交事件

表单元素中有三种类型的标签具备提交功能：

```
<input type="submit" >
<input type="image" >
<button type="submit">提交</button>
```

当点击提交按钮后，会触发form元素上的onsubmit事件，通过为它绑定事件处理方法，可以在提交到服务器之前做一些操作：比如验证表单。

```
let fm = document.getElementById("form1");
fm.onsubmit = function(){
    console.log("表单提交");
}
```

注意：onsubmit事件是绑定在form元素上，而不是提交按钮上。

如果想要阻止表单提交，可以使用事件对象中的阻止事件默认行为的方法：`preventDefault()`

```
let fm = document.getElementById("form1");
fm.onsubmit = function(e){
    console.log("表单提交");
    e.preventDefault();
}
```

2. 重置事件

重置是指将表单中的字段都还原到默认的状态值，而并不是清空内容。表单元素中有两种类型的标签具备重置功能：

```
<input type="reset" >
<button type="reset">重置</button>
```


当点击重置按钮后，会触发form元素上的onreset事件，通过为它绑定事件处理方法，可以在重置之前做一些操作。

```
let fm = document.getElementById("form1");
fm.onreset = function(){
    console.log("表单重置");
}
```

注意：onreset事件是绑定在form元素上，而不是重置按钮上。

如果想要阻止表单重置，可以使用事件对象中的阻止事件默认行为的方法：`preventDefault()`

```
let fm = document.getElementById("form1");
fm.onreset = function(e){
    console.log("表单重置");
    e.preventDefault();
}
```

3. 其他的提交和重置方法

除了可以使用标签提交表单外，form元素还提供了两个方法：`submit()`和`reset()`，它们也具备提交的功能。比如：

```
<form id="fm">
  <input type="button" id="submitBtn" value="提交">
  <input type="button" id="resetBtn" value="重置">
</form>
<script>
  let fm = document.getElementById("form1");
  let submitBtn = document.getElementById("submitBtn");
  let resetBtn = document.getElementById("resetBtn");
  submitBtn.onclick = function(){
    fm.submit();
  }
  resetBtn.onclick = function(){
    fm.reset();
  }
</script>
```

例子中用了两个普通的按钮，在它们各自绑定的点击事件中使用了form元素的 `submit` 和 `reset` 方法，实现了提交和重置的功能。利用方法提交和重置与标签的提交和重置区别：

`submit()` 方法提交后不会触发`onsubmit`事件，点击`submit`类型的按钮后则会触发。

`reset()` 方法和`reset`类型的按钮都会触发`onreset`事件。

4. 焦点事件

焦点事件，顾名思义就是当表单里面的控件获取到焦点时所触发的事件。点到输入框，会触发焦点事件，当鼠标离开某个控件时，同样可以触发焦点事件。在JavaScript中，对应的焦点事件有如下两个：

- `focus`：获取焦点时触发的事件的名称
- `blur`：失去焦点时触发的事件的名称

示例如下：

```
<input type="text" id="phone">
<script>
  let phone = document.getElementById("phone");
  phone.onfocus = function(){
    console.log("文本框被选中");
  }
  phone.onblur = function(){
    console.log("文本框失去焦点");
  }
</script>
```

效果：当文本被选中时会在控制台显示"文本框被选中"，当点击文本框以外的地方时，会显示"文本框失去焦点"。`focus` 和 `blur` 这两个事件是在实际项目开发中最常用的表单事件，常用来在用户填写完信息触发表单的验证。

5. 改变事件

在实际开发中，表单元素中有两个控件也用的比较多，就是`radio`和`checkbox`控件，与这两个控件经常绑定的有一个事件叫做`change`事件，这个事件会在表单的内容发生变化时被触发，同样适用于`text`，`select`和`textarea`等表单控件。

示例如下：

```
<select id="opt">
  <option value="CD">成都</option>
  <option value="BJ">北京</option>
  <option value="SH">上海</option>
  <option value="GZ">广州</option>
  <option value="SZ">深圳</option>
</select>
```

```
<script>
    let obj = document.getElementById("opt");//获取表单元素对象
    obj.onchange = function(){
        console.log("选项已经被修改");
    }
</script>
```

效果：当下拉列表选项被修改时，就会触发事件，在控制台输出"选项已经被修改"。

下面的例子，演示了复选框上面绑定change事件，如下：

```
<input type="checkbox" id="HTML5" name="language">HTML5
<input type="checkbox" id="PHP" name="language">PHP
<input type="checkbox" id="JS" name="language">JS
<script>
    let languages = document.getElementsByName("language");
    for(let i = 0;i < languages.length;i++){
        languages[i].onchange = function(){
            console.log("复选框内容已经改变");
        }
    }
</script>
```

效果：通过浏览器的运行结果，我们会发现不管是选中还是取消选中，都会触发change事件，这也是change事件的一个特性，只要表单元素的内容有改变，就会触发该事件。

6. input事件

在文本框(text)中使用change事件，需要失去焦点，并改变了输入框中内容时才会触发。但有的时候要求在输入框中每输入或删除一个字符都能响应事件，这就需要input事件。例如：

```
<input type="text" id="phone" >
<script>
    let phone = document.getElementById("phone");
    phone.oninput = function(){
        console.log("文本框的内容：",this.value);
    }
</script>
```

效果：在输入框中每输入或删除一个字符，控制台都能会显示当前输入框的内容。这在有些要求实时反馈校验结果的表单中会用到。

11-4 表单验证

大部分时候，我们期望用户输入的数据是我们理想中的格式，尤其是在用户注册的时候，一些重要信息我们是不允许用户乱写的，比如手机号，电子邮箱等。那么我们就需要在表单提交前对用户输入的数据进行一个验证。

11-4-1 验证长度

验证长度是比较简单的，甚至表单元素本身就提供了与长度相关的属性。例如 `<input>` 标签元素提供了 `maxlength` 属性可以设置可输入的最大长度，如下：

```
<form action="" id="myForm">
  <input type="text" maxlength="8">
</form>
```

除了使用 `maxlength` 属性以外，我们也可以利用JavaScript来进行长度的验证，示例如下：

```
<body>
  <form action="" id="myForm">
    <input type="text">
    <button>提交</button>
  </form>
  <script>
    let obj = document.getElementById("myForm");//获取到表单对象
    obj.onsubmit = function(){
      //获取文本框内容的长度
      let length = document.getElementsByTagName("input")[0].value.length;

      console.log(`长度为${length}`);
      return false;
    }
  </script>
</body>
```

效果：点击提交按钮时会打印出当前文本框内容的长度

11-4-2 正则表达式验证

还有更加复杂的情况，我们甚至需要对用户输入的数据格式来进行验证。比如用户注册的时候，我们会要求用户的注册信息完全按照网站需要的格式来填写，这时候就需要使用正则匹配验证

了。

```
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <style>
    span{
      font-size: 12px;
      color: red
    }
  </style>
</head>
<body>
  <form action="" id="myForm">
    <h3>正则表达式验证表单数据</h3>
    <div>
      用户名:
      <input type="text" onchange="userNameCheck()">
    </div>
    <span></span>
    <div><button>提交信息</button></div>
  </form>
  <script>
    let userNameCheck = function(){
      //获取<span>标签
      let span = document.getElementsByTagName("span")[0];
      //创建正则表达式
      let reg = /^[a-z]{1}\w{5,9}$/i;
      //获取到文本框里面的内容
      let value = document.getElementsByTagName("input")[0].value;
      if(value.length === 0)
      {
        span.innerText = "";
      }
      else if(reg.test(value))
      {
        span.innerText = "验证通过";
      }
      else{
        span.innerText = "首字符为字母，长度为6-10个字符的数字字母和下划线"
      }
    }
  </script>
</body>
```

效果：当输入内容不符合要求时，`` 标签的内容会被填充为"首字符为字母，长度为6-10

个字符的数字字母和下划线"

正则表达式验证表单数据

用户名:

首字符为字母，长度为6-10个字符的数字字母和下划线

当输入的内容符合要求时， `` 标签的内容会被填充为"验证通过"

正则表达式验证表单数据

用户名:

验证通过

当文本框中无任何内容时，没有提示信息

正则表达式验证表单数据

用户名:

这里，我们就实现了使用JavaScript来对表单内容进行验证，这也是JavaScript刚诞生时最被人津津乐道的一个场景应用，通过客户端来对表单内容进行验证，可以节省用户的时间，大大提升了用户的体验度。

当然，利用JavaScript实现的表单验证只能简单的验证数据的格式，并不能百分百的保证数据传到服务器端时就是正确的格式，因为我们有各种各样的办法绕过JavaScript的验证，比如模拟表单验证等。但是无论如何，在客户端使用JavaScript对表单进行验证都是非常有必要的。这极大程度的提升了用户的体验。

11-5 HTML5中的表单控件

在前面一小节中所介绍的表单验证，大多都是通过正则表达式来实现的。而在HTML5中，则简化了表单验证的操作，为许多常用的信息验证添加了默认的验证规则，大大提高了开发人员的开发效率。如果不是对表单信息验证有非常特殊的要求的话，HTML5所提供的默认的验证规则完全能够胜任。这一小节，就让我们一起来看一下在HTML5中新添加的这些自带验证规则的表单控件以及新增的表单属性。

11-5-1 新增的表单控件

1. email类型

主要用于用户输入email地址的，在提交表单时，会自动验证email输入框的值，如果不是一个有效的email，则会报错

```
<body>
  <form action="">
    <input type="email" name="" id="">
    <button>提交</button>
  </form>
</body>
```

效果：

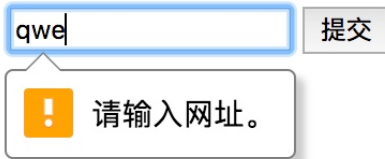
对于不支持 `type=email` 的浏览器，则会将其作为普通文本框来处理，这里也体现了HTML5中平稳退化的设计原则。(包括后面新增的类型，不支持的浏览器也是作为普通文本框来显示)

2. url类型

主要用于输入url地址的，在提交表单的时候，会自动验证url输入框里面的值，如果不是一个有效的url地址，则会报错

```
<body>
  <form action="">
    <input type="url" name="" id="">
```

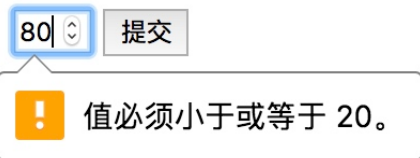
```
<button>提交</button>
</form>
</body>
```



3. number类型

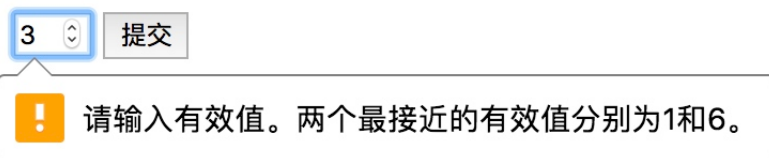
该类型控件只允许输入数值，并且我们还可以设置能够接受数字的范围

```
<body>
  <form action="">
    <input type="number" name="" id="" min="1" max="20">
    <button>提交</button>
  </form>
</body>
```



除此之外，还可以通过 `step` 属性来设置数值的间隔

```
<body>
  <form action="">
    <input type="number" name="" id="" min="1" max="20" step="5">
    <button>提交</button>
  </form>
</body>
```



4. range类型

用于输入包含一定数字范围的文本框，和 `number` 控件的作用大致一致，只不过表现形式是以滚

动条的形式来展现的。和 `number` 控件一样，同样存在 `min`，`max` 以及 `step` 属性。

```
<body>
  <form action="">
    <input type="range" name="" id="" min="1" max="20" step="5">
    <button>提交</button>
  </form>
</body>
```



5. 日期检查类型类型

以前检测日期需要利用插件来实现，现在在HTML5里面也出现了和日期相关的控件，一共有5个，介绍如下

日期控件 `date`

```
<body>
  <form action="">
    <input type="date" name="" id="">
    <button>提交</button>
  </form>
</body>
```



该控件类型支持设置最值属性，如下：

```
<body>
  <form action="">
    <input type="date" min="2015-03-23" max="2015-03-30">
    <button>提交</button>
```

```
</form>
</body>
```

2015 / 03 / 25

2015年03月

周日	周一	周二	周三	周四	周五	周六
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

时间控件 time

```
<body>
  <form action="">
    <input type="time" name="" id="">
    <button>提交</button>
  </form>
</body>
```

上午 01 : 59

本地日期事件控件 datetime-local

```
<body>
  <form action="">
    <input type="datetime-local">
    <button>提交</button>
  </form>
</body>
```

年 / 月 / 日 ----:-- 提交

2018年01月

周日	周一	周二	周三	周四	周五	周六
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3

月控件 month

```
<body>
  <form action="">
    <input type="month">
    <button>提交</button>
  </form>
</body>
```


----年--月 提交




2018年01月

2018			
1月	2月	3月	4月
5月	6月	7月	8月
9月	10月	11月	12月

周控件 week

```
<body>
  <form action="">
    <input type="week">
    <button>提交</button>
  </form>
</body>
```

----- 年第 -- 周  ▼

2018年01月 ▼   

周	周日	周一	周二	周三	周四	周五	周六
1	31	1	2	3	4	5	6
2	7	8	9	10	11	12	13
3	14	15	16	17	18	19	20
4	21	22	23	24	25	26	27
5	28	29	30	31	1	2	3

6. search类型

提供用于搜索关键字的文本框，虽然外观看起来和 `text` 控件差不多，但是却带来了语义上的不同。

```
<body>
  <form action="">
    <input type="search">
    <button>提交</button>
  </form>
</body>
```

123 

实际上和普通文本还是有细微的区别，当我们在搜索框中输入内容后，可以通过右边的小叉进行删除，但是普通文本框则没有这一功能。(当然这取决于不同浏览器的实现)

7. tel类型

tel类型主要用于输入电话号码

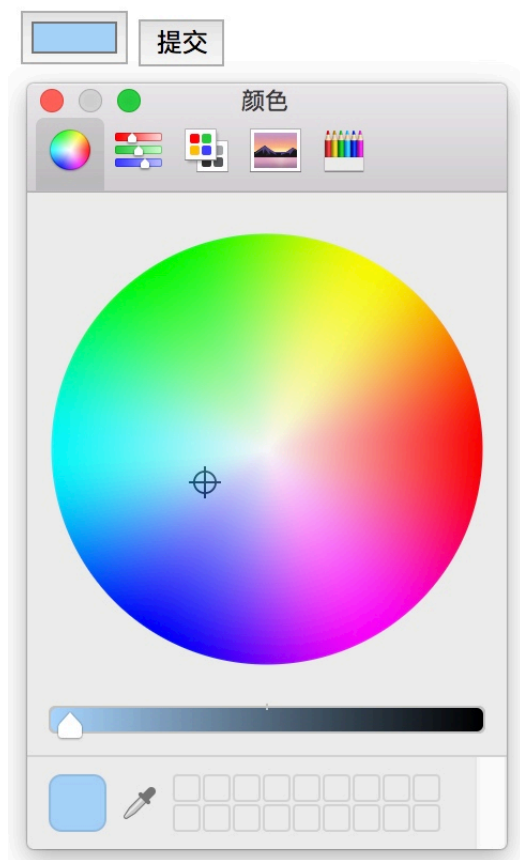
```
<body>
  <form action="">
    <input type="tel">
    <button>提交</button>
  </form>
</body>
```

效果虽然看上去和普通文本框是一样的，但是却有了语义的不同。

8. color类型

专门用于设置颜色的控件

```
<body>
  <form action="">
    <input type="color">
    <button>提交</button>
  </form>
</body>
```



11-5-2 新增的表单属性

`input` 元素不仅新增了控件类型，还新增了几个属性，用于指定输入类型的行为和限制。下面依次介绍这些属性。

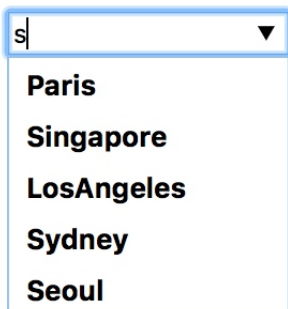
1. autocomplete属性

新增的 `autocomplete` 属性可以帮助用户在 `input` 类型的输入框中实现自动完成内容的输入。支持的控件包

括：`text`，`search`，`url`，`tel`，`email`，`password`，`datepickers`，`range` 以

及 `color`。`autocomplete` 属性的值有2种：`on` 和 `off`，可以将该属性设置到form表单上，因为该属性是可以继承的。可以将 `autocomplete` 属性和 `datalist` 配合着使用，例子如下：

```
<body>
  <form action="" autocomplete="on">
    <input type="text" list="city">
      <datalist id="city">
        <option value="Tokyo">Tokyo</option>
        <option value="NewYork">NewYork</option>
        <option value="BeiJing">BeiJing</option>
        <option value="ChengDU">ChengDU</option>
        <option value="London">London</option>
        <option value="Paris">Paris</option>
        <option value="Singapore">Singapore</option>
        <option value="HongKong">HongKong</option>
        <option value="LosAngeles">LosAngeles</option>
        <option value="Chicago">Chicago</option>
        <option value="Sydney">Sydney</option>
        <option value="Seoul">Seoul</option>
      </datalist>
    </form>
  </body>
```



2. autofocus属性

该属性可以让某些控件自动的获取光标焦点，常用于某些需要自动获取焦点的控件，例如向用户展示许可协议时，默认的焦点就聚焦在同意这个按钮上面，如下：

```
<body>
  <h3>请认真阅读许可协议</h3>
  <textarea name="" id="" cols="30" rows="10">
    本许可协议允许其他人发行、再混合、调整、以您的作品为基础进行创作。
    即使出于商业性目的，只要他们指明您的原创身份。
    这是我们提供的最具有适应性的许可协议。推荐用于最大程度散布和使用被授权作品。
  </textarea>
  <div>
```

```
<button autofocus="on">同意</button>
<button>不同意</button>
</div>
</body>
```

请认真阅读许可协议

本许可协议允许其他人发行、再混合、调整、以您的作品为基础进行创作。

即使出于商业性目的，只要他们指明您的原创身份。

这是我们提供的最具有适应性的许可协议。推荐用于最大程度散布和使用被授权作品。

属性值有两个，分别是 `on` 和 `off`，设置为 `on` 代表开启自动焦点，`off` 代表不开启。当然不开启自动焦点的话直接不书写该属性就可以了。

3. form属性

在以前提交表单的时候，只能提交位于 `<form>` 标签以内的表单控件的内容，如果是处于 `<form>` 标签以外的表单控件，内容是无法被提交到的。通过 `form` 属性，我们可以采集到处于 `<form>` 以外的表单控件的内容，只需要在表单的 `form` 属性里面填写表单的 `id` 便可以和该表单进行绑定，示例如下：

```
<body>
  <form action="#" id="myForm">
    <input type="text" name="test1">
    <button>提交</button>
  </form>
  <input type="text" name="test2" form="myForm">
</body>
```

效果：点击提交后 `name` 值为 `test2` 的表单，内容也被一起提交了。

← → ↺ 🔍 file:///Users/Jie/Desktop/1.html?test1=123&test2=456#

如果一个 `form` 属性要引用两个或者两个以上的表单，只需要用空格将表单的 `id` 间隔开即可

```
<input type="text" name="test2" form="myForm1 myForm2 myForm3">
```

4. 表单重写属性

新增的表单重写属性是一套属性，包括下面的属

性：`formaction`、`formenctype`、`formmethod`、`formnovalidate`、`formtarget`。这里讲一个 `formaction`，其他的属性用法是一样的。之前的form表单，`action` 属性表示将表单内容提交到哪一个页面，但是有一个缺点就是所有的信息都会被提交到一个固定的页面，有了 `formaction` 以后，就可以将不同的信息提交到不同的页面。示例如下：

```
<body>
  <form action="1.php">
    <input type="text" name="test1" id="test1">
    <input type="text" name="test2" id="test2" formaction="2.php">
    <input type="text" name="test3" id="test3" formaction="3.php">
    <input type="text" name="test4" id="test4" formaction="4.php">
    <button>提交</button>
  </form>
</body>
```

这里除了 `test1` 文本框被提交给 `1.php` 以外，其他的文本框内容就被提交给了不同的页面。

5. list属性

该属性主要是和HTML5新增的 `<datalist>` 标签配合使用的，`list` 属性里面写上 `<datalist>` 标签的 `id` 即可，前面在演示 `autocomplete` 属性时已经演示过了，这里不再做过多介绍。

6. 最值属性

这个在前面介绍HTML5新增控件的时候也已经见到过了，主要用于 `number`，`range`，`date` 等控件里面。

max：输入框允许的最大值

min：输入框允许的最小值

step：输入框输入数字时的数字间隔

7. multiple属性

这个属性可以用于设置下拉列表显示多个选项，或者上传文件时上传多个文件。


```
<body>
  <form action="">
    <input type="file" name="" id="" multiple>
    <button>提交</button>
  </form>
</body>
```

选择文件 2 个文件

提交

8. pattern属性

这个属性是相当方便的一个属性，简化了表单验证中正则表达式的书写方式，直接将正则表达式作为该属性的属性值即可，示例如下：

```
<body>
  <form action="">
    <input type="text" pattern="/^\d{6}$/">
    <button>提交</button>
  </form>
</body>
```

123|

提交



请与所请求的格式保持一致。

9. 占位符属性

placeholder 属性用于给文本框一个默认的内容

```
<body>
  <form action="">
    <input type="text" placeholder="请填写您的用户名">
    <button>提交</button>
  </form>
</body>
```

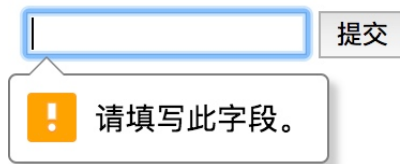
请填写您的用户名

提交

10. required属性

为表单控件书写上该属性表示此项目为必须填写项目

```
<body>
  <form action="">
    <input type="text" required>
    <button>提交</button>
  </form>
</body>
```



提交

请填写此字段。

11. novalidate属性

该属性用于在提交表单时取消整个表单的验证，关闭对表单内所有元素的检查，如果只想取消一个，那么就可以使用前面所讲的 `formnovalidate` 属性单独用于表单里的某一个控件里面。示例如下：

```
<body>
  <form action="" novalidate>
    <input type="email" name="email">
    <button>提交</button>
  </form>
</body>
```



提交

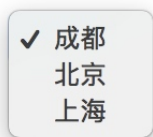
哪怕是不符合email格式要求，也一样能够被提交。

11-6 下拉列表和多选列表的使用

在表单元素中，下拉列表和多选列表也是较为常用的元素之一。

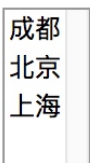
```
<select>
  <option>成都</option>
  <option>北京</option>
  <option>上海</option>
</select>
```

这是下拉列表，一次只能选择一项。



在select中加上 `multiple` 属性就变成多选列表。

```
<select multiple>
  <option>成都</option>
  <option>北京</option>
  <option>上海</option>
</select>
```



这两种效果的列表元素前面已经介绍过，这里我主要为大家介绍如何在JS中动态为它们添加、删除和获取选中的节点。这些操作实际就是对元素option的增删改。使用之前介绍过的DOM操作是可行的，但除此以外，还可以使用DOM中专门为 `select/option` 元素提供的方法。

1. 增加元素

```
let newOption = new Option("深圳", "SZ");
```

其中第一个参数是option标签中间的文本，第二个参数是option的value值。

```
<select id="citySelect">
  <option>成都</option>
  <option>北京</option>
  <option>上海</option>
</select>
<script>
  let citySelect = document.getElementById("citySelect");
  let newOption = new Option("深圳","SZ");
  citySelect.options.add(newOption);
  console.log(citySelect.options);
</script>
```

citySelect.options获取的是列表中option的集合，这个集合不是数组，它是重新封装的集合对象。所以增加元素的方法是add，而不是push。

► *HTMLOptionsCollection(4) [option, option, option, option, selectedIndex: 0]*

2. 修改元素

修改元素可以通过指定的下标找到要修改的option，然后通过value或text修改其中的内容。

```
citySelect.options[0].value = "GZ";
citySelect.options[0].text = "广州";
```

3. 删除元素

删除使用的是 remove 方法，同样通过下标指定要删除的项。

```
citySelect.options.remove(0);
```

如果要删除所有，可以直接将options.length属性设置为0。

```
citySelect.options.length = 0;
```

4. 获取选中的元素

如果是下拉列表，由于只能选中一项，可以使用select元素的value属性来获取选中项的value值，如：

```
console.log("你选中的是：",citySelect.value);
```

这个方法只能获取选中项的value，如果想要获取当中的文本，可以使用selectedIndex属性，它得到的是获取选中项的下标，然后再利用该下标到option集合中找到对应的项，如：

```
console.log("你选中的是：",citySelect.options[citySelect.selectedIndex].text);
```

如果是多选列表，由于选中的可能有多个，那还是需要循环集合，然后判断option的selected属性是否为真。

```
let options = citySelect.options;
for(let i = 0;i < options.length;i++){
    if(options[i].selected){
        console.log("你选中的是：",options[i].text);
    }
}
```

11-7 表单常见操作

本小结我们将结合前面学习到的知识，来看一下项目开发中常见的表单相关操作。

11-7-1 全选和反选

全选，全不选以及反选是网页中最常见的应用，实现代码如下：

```
<body>
  苹果<input type="checkbox" name="fruit" value="苹果">
  香蕉<input type="checkbox" name="fruit" value="香蕉">
  橘子<input type="checkbox" name="fruit" value="橘子">
  榴莲<input type="checkbox" name="fruit" value="榴莲">
  石榴<input type="checkbox" name="fruit" value="石榴">
  甘蔗<input type="checkbox" name="fruit" value="甘蔗">
  葡萄<input type="checkbox" name="fruit" value="葡萄">
  <div style="margin-top:5px;">
    <button id="all">全选</button>
    <button id="not">全不选</button>
    <button id="reverse">反选</button>
  </div>
  <script>
    let obj = document.getElementsByName("fruit");
    //全选绑定的事件
    all.onclick = function(){
      for(let i=0;i<obj.length;i++){
        {
          obj[i].checked = true;
        }
      }
    }
    //全不选绑定的事件
    not.onclick = function(){
      for(let i=0;i<obj.length;i++){
        {
          obj[i].checked = false;
        }
      }
    }
    //反选绑定的事件
    reverse.onclick = function(){
      for(let i=0;i<obj.length;i++){
        {
          if(obj[i].checked === true)
          {
            obj[i].checked = false;
          }
        }
      }
    }
  </script>
```

```

    }
    else{
        obj[i].checked = true;
    }
    //更加简便的写法是直接取反, true变为false, false变为true
    //obj[i].checked = !obj[i].checked;
}
}
</script>
</body>

```

苹果 ☒ 香蕉 ☒ 橘子 ☐ 榴莲 ☒ 石榴 ☐ 甘蔗 ☐ 葡萄 ☐

这里，我们使用for循环来遍历所有的复选框，通过设置复选框的 `checked` 来控制复选框有没有被选中。

11-7-2 下拉框特效

有些时候，我们需要将一个框里面的内容移动到另外一个框里面，下面的代码演示了这一效果：

```

<body>
  <h3>队伍配置</h3>
  <select name="" id="sel1" size="8" multiple>
    <option value="">赵信</option>
    <option value="">泰达米尔</option>
    <option value="">希瓦娜</option>
    <option value="">金克丝</option>
    <option value="">索拉卡</option>
  </select>
  <button id="toRight">>></button>
  <button id="toLeft"><<<</button>
  <select name="" id="sel2" size="8" multiple>
    <option value="">菲奥娜</option>
    <option value="">伊芙琳</option>
    <option value="">卡西奥佩娅</option>
    <option value="">艾希</option>
    <option value="">娑娜</option>
  </select>
  <script>
    //获取两个下拉列表
    let sel1 = document.getElementById("sel1");
    let sel2 = document.getElementById("sel2");
    //为两个按钮添加事件

```

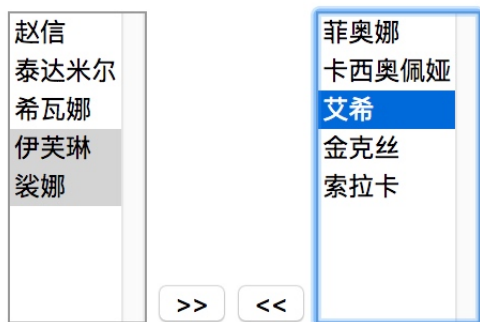
```

toRight.onclick = function(){
    let childs = sel1.childNodes;
    for(let i=0;i<childs.length;i++)
    {
        if(childs[i].selected)
        {
            sel2.appendChild(childs[i]);
        }
    }
}
toLeft.onclick = function(){
    let childs = sel2.childNodes;
    for(let i=0;i<childs.length;i++)
    {
        if(childs[i].selected)
        {
            sel1.appendChild(childs[i]);
        }
    }
}
</script>
</body>

```

效果：可以自由的移动两个下拉列表里面的选项

队伍配置



当然，这个特效在我们的实际项目中可能很少用到，这里使用这个实例是为了让大家知道，如果我们没有创建新的节点，而是直接获取到HTML中的节点往其他节点中进行添加，那么会是一个移动的效果，就是把节点从原来的位置移动到我们要添加的那个节点下面。

11-7-3 下拉列表联动

下拉列表联动也是页面中比较常见的一个效果。例如当我们选择中国时，就会出现与中国相关的城市，而当我们选择日本时，就会显示和日本相关的城市，也就是说选择不同的国家，后面的下拉列表里面的内容就不一样，实现如下：


```
<body>
  <!-- 准备三个下拉列表 -->
  <select name="" id="province">
    <option value="">请选择国家</option>
  </select>
  <select name="" id="city">
    <option value="">请选择城市</option>
  </select>
  <select name="" id="county">
    <option value="">请选择区县</option>
  </select>
  <script>
    //模拟从服务器端获取到的数据
    let provinceData = ["中国","日本"];
    let cityData = [
      ["北京","上海","广州","深圳","成都"],
      ["东京","大阪","京都","名古屋","北海道"],
    ];
    let countyData = [
      [
        ["东城区","西城区","朝阳区","丰台区","石景山区"],
        ["黄浦区","徐汇区","长宁区","静安区","虹口区"],
        ["越秀区","荔湾区","海珠区","花都区","南沙区"],
        ["福田区","罗湖区","南山区","龙华区","宝安区"],
        ["锦江区","武侯区","青羊区","金牛区","高新区"]
      ],
      [
        ["千代田区","新宿区","墨田区","中野区","品川区"],
        ["大正区","港区","鹤见区","旭区","天王寺区"],
        ["右京区","左京区","上京区","下京区","京都市"],
        ["热田区","北区","昭和区","中村区","守山区"],
        ["深川市","北广岛市","江别市","千岁市","惠庭市"]
      ]
    ];
    //获取三个下拉列表的对象
    let provinceObj = document.getElementById("province");
    let cityObj = document.getElementById("city");
    let countyObj = document.getElementById("county");
    //首先将国家的信息新添加到第一个下拉列表里面
    for(let i=0;i<provinceData.length;i++)
    {
      //创建空的option选项
      let newOption = document.createElement("option");
      newOption.value = i;//设置option的属性值 从0开始
      newOption.innerText = provinceData[i];//设置option的文本值
      provinceObj.appendChild(newOption);//将option添加到第一个下拉列表
    }
    //监测第一个下拉列表，一旦选项有改变，做如下的操作
```

```
provinceObj.onChange = function(){
    //清空后面两个下拉列表的内容
    cityObj.innerHTML = "";
    countyObj.innerHTML = "";
    //将第一个下拉列表的value值作为id号
    let provinceID = provinceObj.value;
    if(provinceID === "")//如果id号为空，则后面两个下拉列表显示如下内容
    {
        let newOption = document.createElement("option");
        newOption.innerText = "请选择城市";
        cityObj.appendChild(newOption);
        let newOption2 = document.createElement("option");
        newOption2.innerText = "请选择区县";
        countyObj.appendChild(newOption2);
    }
    //否则显示相应的城市和区域信息
    else{
        //将城市添加到第二个下拉列表里面
        let citys = cityData[provinceID];
        for(let i=0;i<citys.length;i++)
        {
            let newOption = document.createElement("option");
            newOption.value = i;
            newOption.innerText = citys[i];
            cityObj.appendChild(newOption);
        }
        //因为默认是第一个城市，所以默认出现第一个城市所对应的区县
        let counties = countyData[provinceID][0];
        for(let i=0;i<counties.length;i++)
        {
            let newOption = document.createElement("option");
            newOption.value = i;
            newOption.innerText = counties[i];
            countyObj.appendChild(newOption);
        }
    }
}

//监测第二个下拉列表
cityObj.onChange = function(){
    //清空第三个下拉列表的内容
    countyObj.innerHTML = "";
    //得到当前选择的国家ID和城市ID
    let provinceID = provinceObj.value;
    let cityID = cityObj.value;
    //通过前面两个ID定位到对应的区县数组
    let counties = countyData[provinceID][cityID];
    //遍历然后添加节点
    for(let i=0;i<counties.length;i++)
```

```
        {
            let newOption = document.createElement("option");
            newOption.value = i;
            newOption.innerText = counties[i];
            countyObj.appendChild(newOption);
        }
    }
</script>
</body>
```

效果：

请选择国家 ▾ 请选择城市 ▾ 请选择区县 ▾

选择"中国"以后，自动出现第一个城市和与第一个城市相关的区县

中国 ▾ 北京 ▾ 东城区 ▾

主要就是对前面两个下拉列表的内容变化进行监听，然后动态的给每个下拉列表添加上 `<option>` 元素。

总结

1. 操作表单的第一步，就是要获取到表单。我们可以通过多种方式来获取到表单。
2. 表单提供了相应的事件，方便我们把握用户填写表单的状态。例如焦点事件，内容改变事件等。
3. 用户在填写表单时，一些重要信息我们不允许用户随意乱写，这个时候我们需要对表单进行一个验证。一般表单验证需要使用到正则表达式。
4. HTML5中新增了很多表单控件。其中一部分控件自带内容的验证功能。
5. 掌握表单常见的操作有助于我们在开发时事半功倍。