第 27 章 表单处理

学习要点：

1.表单介绍
2.文本框脚本
3.选择框脚本

主讲教师：李炎恢
合作网站：http://www.ibeifeng.com
讲师博客：http://hi.baidu.com/李炎恢

为了分担服务器处理表单的压力，JavaScript 提供了一些解决方案，从而大大打破了处处依赖服务器的局面。

一．表单介绍

在 HTML 中，表单是由<form>元素来表示的，而在 JavaScript 中，表单对应的则是 HTMLFormElement 类型。HTMLFormElement 继承了 HTMLElement，因此它拥有 HTML 元素具有的默认属性，并且还独有自己的属性和方法：

HTMLFormElement 属性和方法

属性或方法	说明
acceptCharset	服务器能够处理的字符集
action	接受请求的 URL
elements	表单中所有控件的集合
enctype	请求的编码类型
length	表单中控件的数量
name	表单的名称
target	用于发送请求和接受响应的窗口名称
reset()	将所有表单重置
submit()	提交表单

获取表单<form>对象的方法有很多种，如下：

```
document.getElementById('myForm');              //使用 ID 获取<form>元素
document.getElementsByTagName('form')[0];       //使用获取第一个元素方式获取
document.forms[0];                              //使用 forms 的数字下标获取元素
document.forms['yourForm'];                     //使用 forms 的名称下标获取元素
document.yourForm;                              //使用 name 名称直接获取元素
```

PS：最后一种方法使用 name 名称直接获取元素，已经不推荐使用，这是向下兼容的早期用法。问题颇多，比如有两个相同名称的，变成数组；而且这种方式以后有可能会不兼容。

提交表单

通过事件对象，可以阻止 submit 的默认行为，submit 事件的默认行为就是携带数据跳转到指定页面。

```
addEvent(fm, 'submit', function (evt) {
    preDef(evt);
});
```

我们可以可以使用 submit()方法来自定义触发 submit 事件，也就是说，并不一定非要点击 submit 按钮才能提交。

```
if (e.ctrlKey && e.keyCode == 13) fm.submit();  //判断按住了 ctrl 和 enter 键触发
```

PS：在表单中尽量避免使用 name="submit"或 id="submit"等命名，这会和 submit()方法发生冲突导致无法提交。

提交数据最大的问题就是重复提交表单。因为各种原因，当一条数据提交到服务器的时候会出现延迟等长时间没反映，导致用户不停的点击提交，从而使得重复提交了很多相同的请求，或造成错误、或写入数据库多条相同信息。

```
addEvent(fm, 'submit', function (evt) {              //模拟延迟
    preDef(evt);
    setTimeout(function () {
        fm.submit();
    }, 3000);
});
```

有两种方法可以解决这种问题：第一种就是提交之后，立刻禁用点击按钮；第二种就是提交之后取消后续的表单提交操作。

```
document.getElementById('sub').disabled = true; //将按钮禁用

var flag = false                                 //设置一个监听变量
if (flag == true) return                         //如果存在返回退出事件
flag = true;                                     //否则确定是第一次，设置为 true
```

PS：在某些浏览器，F5 只能起到缓存刷新的效果，有可能获取不到真正的源头更新的数据。那么使用 ctrl+F5 就可以把源头给刷出来。

重置表单

用户点击重置按钮时，表单会被初始化。虽然这个按钮还得以保留，但目前的 Web 已经很少去使用了。因为用户已经填写好各种数据，不小心点了重置就会全部清空，用户体验极差。

有两种方法调用 reset 事件，第一个就是直接 type="reset"即可；第二个就是使用 fm.reset()方法调用即可。

```
<input type="reset" value="重置" />              //不需要 JS 代码即可实现
addEvent(document,'click', function () {
```

```
    fm.reset();                                    //使用 JS 方法实现重置
});
addEvent(fm,'reset', function () {                 //获取重置按钮
    //
});
```

表单字段

如果想访问表单元素，可以使用之前章节讲到的 DOM 方法访问。但使用原生的 DOM 访问虽然比较通用，但不是很便利。表单处理中，我们建议使用 HTML DOM，它有自己的 elements 属性，该属性是表单中所有元素的集合。

```
fm.elements[0];                                    //获取第一个表单字段元素
fm.elements['user'];                               //获取 name 是 user 的表单字段元素
fm.elements.length;                                //获取所有表单字段的数量
```

如果多个表单字段都使用同一个 name，那么就会返回该 name 的 NodeList 表单列表。

```
fm.elements['sex'];                                //获取相同 name 表单字段列表
```

PS：我们是通过 fm.elements[0]来获取第一个表单字段的，但也可以使用 fm[0]直接访问第一个字段。因为 fm[0]访问方式是为了向下兼容的，所以，我们建议大家使用 elements 属性来获取。

共有的表单字段属性

除了<fieldset>元素之外，所有表单字段都拥有相同的一组属性。由于<input>类型可以表示多种表单字段，因此有些属性只适用于某些字段。以下罗列出共有的属性：

属性或方法	说明
disabled	布尔值，表示当前字段是否被禁用
form	指向当前字段所属表单的指针，只读
name	当前字段的名称
readOnly	布尔值，表示当前字段是否只读
tabIndex	表示当前字段的切换
type	当前字段的类型
value	当前字段的值

这些属性其实就是 HTML 表单里的属性，在 XHTML 课程中已经详细讲解过，这里不一个个赘述，重点看几个最常用的。

```
fm.elements[0].value;                              //获取和设置 value
fm.elements[0].form == fm;                         //查看当前字段所属表单
fm.elements[0].disabled = true;                    //禁用当前字段
fm.elements[0].type = 'checkbox';                  //修改字段类型，极不推荐
```

除了<fieldset>字段之外，所有表单字段都有 type 属性。对于<input>元素，这个值等于 HTML 属性的 type 值。对于非<input>元素，这个 type 的属性值如下：

元素说明	HTML 标签	type 属性的值
单选列表	<select>...</select>	select-one
多选列表	<select multiple>...</select>	select-multiple
自定义按钮	<button>...</button>	button
自定义非提交按钮	<button type="button">...</button>	button
自定义重置按钮	<button type="reset">...</button>	reset
自定义提交按钮	<button type="submit">...</button>	submit

PS：<input>和<button>元素的 type 属性是可以动态修改的，而<select>元素的 type 属性则是只读的。(在不必要的情况下，建议不修改 type)。

共有的表单字段方法

每个表单字段都有两个方法：foucs()和 blur()。

方法	说明
focus()	将焦点定位到表单字段里
blur()	从元素中将焦点移走

```
fm.elements[0].focus();                    //将焦点移入
fm.elements[0].blur();                     //将焦点移出
```

共有的表单字段事件

表单共有的字段事件有以下三种：

事件名	说明
blur	当字段失去焦点时触发
change	对于<input>和<textarea>元素，在改变 value 并失去焦点时触发；对于<select>元素，在改变选项时触发
focus	当前字段获取焦点时触发

```
addEvent(textField, 'focus', function () {          //缓存 blur 和 change 再测试一下
    alert('Lee');
});
```

PS：关于 blur 和 change 事件的关系，并没有严格的规定。在某些浏览器中，blur 事件会先于 change 事件发生；而在其他浏览器中，则恰好相反。

二．文本框脚本

在 HTML 中，有两种方式来表现文本框：一种是单行文本框<input type="text">，一种是多行文本框<textarea>。虽然<input>在字面上有 value 值，而<textarea>却没有，但通过都

可以通过 value 获取他们的值。

```
var textField = fm.elements[0];
var areaField = fm.elements[1];
alert(textField.value + ',' + areaField.value);        //得到 value 值
```

PS：使用表单的 value 是最推荐使用的，它是 HTML DOM 中的属性，不建议使用标准 DOM 的方法。也就是说不要使用 getAttribute()获取 value 值。原因很简单，对 value 属性的修改，不一定会反映在 DOM 中。

除了 value 值，还有一个属性对应的是 defaultValue，可以得到原本的 value 值，不会因为值的改变而变化。

```
alert(textField.defaultValue);                    //得到最初的 value 值
```

选择文本

使用 select()方法，可以将文本框里的文本选中，并且将焦点设置到文本框中。

```
textField.select();                               //选中文本框中的文本
```

选择部分文本

在使用文本框内容的时候，我们有时要直接选定部分文本，这个行为还没有标准。Firefox 的解决方案是：setSelectionRange()方法。这个方法接受两个参数：索引和长度。

```
textField.setSelectionRange(0,1);                 //选择第一个字符
textField.focus();                                //焦点移入

textField.setSelectionRange(0, textField.value.length);    //选择全部
textField.focus();                                //焦点移入
```

除了 IE，其他浏览器都支持这种写法(IE9+支持)，那么 IE 想要选择部分文本，可以使用 IE 的范围操作。

```
var range = textField.createTextRange();          //创建一个文本范围对象
range.collapse(true);                             //将指针移到起点
range.moveStart('character', 0);                  //移动起点，character 表示逐字移动
range.moveEnd('character', 1);                    //移动终点，同上
range.select();                                   //焦点选定
```

PS：关于 IE 范围的详细讲解，我们将在今后的课程中继续讨论。并且 W3C 也有自己的范围。

```
//选择部分文本实现跨浏览器兼容
function selectText(text, start, stop) {
    if (text.setSelectionRange) {
        text.setSelectionRange(start, stop);
        text.focus();
    } else if (text.createTextRange) {
        var range = text.createTextRange();
```

```
            range.collapse(true);
            range.moveStart('character', start);
            range.moveEnd('character', stop - start);       //IE 用终点减去起点得到字符数
            range.select();
        }
    }
```

使用 select 事件，可以选中文本框文本后触发。

```
addEvent(textField, 'select', function () {
    alert(this.value);                              //IE 事件需要传递 this 才可以这么写
});
```

取得选择的文本

如果我们想要取得选择的那个文本，就必须使用一些手段。目前位置，没有任何规范解决这个问题。Firefox 为文本框提供了两个属性：selectionStart 和 selectionEnd。

```
addEvent(textField, 'select', function () {
    alert(this.value.substring(this.selectionStart, this.selectionEnd));
});
```

除了 IE，其他浏览器均支持这两个属性（IE9+已支持）。IE 不支持，而提供了另一个方案：selection 对象，属于 document。这个对象保存着用户在整个文档范围内选择的文本信息。导致我们需要做浏览器兼容。

```
function getSelectText(text) {
    if (typeof text.selectionStart == 'number') {  //非 IE
        return text.value.substring(text.selectionStart, text.selectionEnd);
    } else if (document.selection) {                //IE
        return document.selection.createRange().text;      //获取 IE 选择的文本
    }
}
```

PS：有一个最大的问题，就是 IE 在触发 select 事件的时候，在选择一个字符后立即触发，而其他浏览器是选择想要的字符释放鼠标键后才触发。所以，如果使用 alert()的话，导致跨浏览器的不兼容。我们没有办法让浏览器行为保持统一，但可以通过不去使用 alert()来解决。

```
addEvent(textField, 'select', function () {
    //alert(getSelectText(this));                   //导致用户行为结果不一致
    document.getElementById('box').innerHTML = getSelectText(this);
});
```

过滤输入

为了使文本框输入指定的字符，我们必须对输入进的字符进行验证。有一种做法是判断字符是否合法，这是提交后操作的。那么我们还可以在提交前限制某些字符，还过滤输入。

```
addEvent(areaField, 'keypress', function (evt) {
    var e = evt || window.event;
```

```
    var charCode = getCharCode(evt);            //得到字符编码
    if (!/\d/.test(String.fromCharCode(charCode)) && charCode > 8) {    //条件阻止默认
        preDef(evt);
    }
});
```

PS：前半段条件判断只有数字才可以输入，导致常规按键，比如光标键、退格键、删除键等无法使用。部分浏览器比如 Firfox，需要解放这些键，而非字符触发的编码均为 0；在 Safari3 之前的浏览器，也会被阻止，而它对应的字符编码全部为 8，所以最后就加上 charCode > 8 的判断即可。

PS：当然，这种过滤还是比较脆落的，我们还希望能够阻止裁剪、复制、粘贴和中文字符输入操作才能真正屏蔽掉这些。

如果要阻止裁剪、复制和粘贴，那么我们可以在剪贴板相关的事件上进行处理，JavaScript 提供了六组剪贴板相关的事件：

事件名	说明
copy	在发生复制操作时触发
cut	在发生裁剪操作时触发
paste	在发生粘贴操作时触发
beforecopy	在发生复制操作前触发
beforecut	在发生裁剪操作前触发
beforepaste	在发生粘贴操作前触发

由于剪贴板没有标准，导致不同的浏览器有不同的解释。Safari、Chrome 和 Firefox 中，凡是 before 前缀的事件，都需要在特定条件下触发。而 IE 则会在操作时之前触发带 before 前缀的事件。

如果我们想要禁用裁剪、复制、粘贴，那么只要阻止默认行为即可。

```
addEvent(areaField, 'cut', function (evt) {          //阻止裁剪
    preDef(evt);
});
addEvent(areaField, 'copy', function (evt) {         //阻止复制
    preDef(evt);
});
addEvent(areaField, 'paste', function (evt) {        //阻止粘贴
    preDef(evt);
});
```

当我们裁剪和复制的时候，我们可以访问剪贴板里的内容，但问题是 FireFox，Opera 浏览器不支持访问剪贴板。并且，不同的浏览器也有自己不同的理解。所以，这里我们就不在赘述。

最后一个问题影响到可能会影响输入的因素就是：输入法。我们知道，中文输入法，它

的原理是在输入法面板上先存储文本，按下回车就写入英文文本，按下空格就写入中文文本。

有一种解决方案是通过 CSS 来禁止调出输入法：

```
style="ime-mode:disabled"                    //CSS 直接编写
areaField.style.imeMode = 'disabled';        //或在 JS 里设置也可以
```

PS：但我们也发先，Chrome 浏览器却无法禁止输入法调出。所以，为了解决谷歌浏览器的问题，最好还要使用正则验证已输入的文本。

```
addEvent(areaField, 'keyup', function (evt) {      //keyup 弹起的时候
    this.value = this.value.replace(/[^\d]/g, ");  //把非数字都替换成空
});
```

自动切换焦点

为了增加表单字段的易用性，很多字段在满足一定条件时(比如长度)，就会自动切换到下一个字段上继续填写。

```
<input type="text" name="user1" maxlength="1" />   //只能写 1 个
<input type="text" name="user2" maxlength="2" />   //只能写 2 个
<input type="text" name="user3" maxlength="3" />   //只能写 3 个

function tabForward (evt) {
    var e = evt || window.event;
    var target = getTarget(evt);
    //判断当前长度是否和指定长度一致
    if (target.value.length == target.maxLength) {
        //遍历所有字段
        for (var i =0; i < fm.elements.length; i ++) {
            //找到当前字段
            if (fm.elements[i] == target) {
                //就把焦点移入下一个
                fm.elements[i + 1].focus();
                //中途返回
                return;
            }
        }
    }
}
```

三．选择框脚本

选择框是通过<select>和<option>元素创建的，除了通用的一些属性和方法外，HTMLSelectElement 类型还提供了如下的属性和方法：

HTMLSelectElement 对象

属性/方法	说明
add(new,rel)	插入新元素，并指定位置
multiple	布尔值，是否允许多项选择

options	<option>元素的 HTMLColletion 集合
remove(index)	移除给定位置的选项
selectedIndex	基于 0 的选中项的索引，如果没有选中项，则值为-1
size	选择框中可见的行数

在 DOM 中，每个<option>元素都有一个 HTMLOptionElement 对象，以便访问数据，这个对象有如下一些属性：

HTMLOptionElement 对象

属性	说明
index	当前选项在 options 集合中的索引
label	当前选项的标签
selected	布尔值，表示当前选项是否被选中
text	选项的文本
value	选项的值

```
var city = fm.elements['city'];                 //HTMLSelectElement
alert(city.options);                            //HTMLOptionsCollection
alert(city.options[0]);                         //HTMLOptionElement
alert(city.type);                               //select-one
```

PS：选择框里的 type 属性有可能是：select-one，也有可能是：select-multiple，这取决于 HTML 代码中有没有 multiple 属性。

```
alert(city.options[0].firstChild.nodeValue);    //上海 t，获取 text 值，不推荐的做法
alert(city.options[0].getAttribute('value'));   //上海 v，获取 value 值，不推荐的做法

alert(city.options[0].text);                    //上海 t，获取 text 值，推荐
alert(city.options[0].value);                   //上海 v，获取 value 值，推荐
```

PS：操作 select 时，最好使用 HTML DOM，因为所有浏览器兼容的很好。而如果使用标准 DOM，会因为不同的浏览器导致不同的结果。

PS：当选项没有 value 值的时候，IE 会返回空字符串，其他浏览器会返回 text 值。

选择选项

对于只能选择一项的选择框，使用 selectedIndex 属性最为简单。

```
addEvent(city, 'change', function () {
    alert(this.selectedIndex);                      //得到当前选项的索引，从 0 开始
    alert(this.options[this.selectedIndex].text);   //得到当前选项的 text 值
    alert(this.options[this.selectedIndex].value); //得到当前选项的 value 值
});
```

PS：如果是多项选择，他始终返回的是第一个项。

```
city.selectedIndex = 1;                    //设置 selectedIndex 可以定位某个索引
```

通过 option 的属性(布尔值)，也可以设置某个索引，设置为 true 即可。

```
city.options[0].selected = true;           //设置第一个索引
```

而 selected 和 selectedIndex 在用途上最大的区别是，selected 是返回的布尔值，所以一般用于判断上；而 selectedIndex 是数值，一般用于设置和获取。

```
addEvent(city, 'change', function () {
    if (this.options[2].selected == true) {        //判断第三个选项是否被选定
        alert('选择正确！');
    }
});
```

添加选项

如需动态的添加选项我们有两种方案：DOM 和 Option 构造函数。

```
var option = document.createElement('option');
option.appendChild(document.createTextNode('北京 t'));
option.setAttribute('value', '北京 v')
city.appendChild(option);
```

使用 Option 构造函数创建：

```
var option = new Option('北京 t', '北京 v');
city.appendChild(option);                  //IE 出现 bug
```

使用 add()方法来添加选项：

```
var option = new Option('北京 t', '北京 v');
city.add(option, 0);                       //0，表示添加到第一位
```

PS：在 DOM 规定，add()中两个参数是必须的，如果不确定索引，那么第二个参数设置 null 即可，即默认移入最后一个选项。但这是 IE 中规定第二个参数是可选的，所以设置 null 表示放入不存在的位置，导致失踪，为了兼容性，我们传递 undefined 即可兼容。

```
city.add(option, null);                    //IE 不显示了
city.add(option, undefined);               //兼容了
```

移除选项

有三种方式可以移除某一个选项：DOM 移除、remove()方法移除和 null 移除。

```
city.removeChild(city.options[0]);         //DOM 移除
city.remove(0);                            //remove()移除，推荐
city.options[0] = null;                    //null 移除
```

PS：当第一项移除后，下面的项，往上顶，所以不停的移除第一项，即可全部移除。

移动选项

如果有两个选择框，把第一个选择框里的第一项移到第二个选择框里，并且第一个选择框里的第一项被移除。

```
var city = fm.elements['city'];                    //第一个选择框
var info = fm.elements['info'];                    //第二个选择框
info.appendChild(city.options[0]);                 //移动，被自我删除
```

排列选项

选择框提供了一个 index 属性，可以得到当前选项的索引值，和 selectedIndex 的区别是，一个是选择框对象的调用，一个是选项对象的调用。

```
var option1 = city.options[1];
city.insertBefore(option1, city.options[option1.index - 1]);        //往下移动移位
```

单选按钮

通过 checked 属性来获取单选按钮的值。

```
for (var i = 0; i < fm.sex.length; i ++) {          //循环单选按钮
    if (fm.sex[i].checked == true) {                //遍历每一个找出选中的那个
        alert(fm.sex[i].value);                     //得到值
    }
}
```

PS：除了 checked 属性之外，单选按钮还有一个 defaultChecked 按钮，它获取的是原本的 checked 按钮对象，而不会因为 checked 的改变而改变。

```
if (fm.sex[i].defaultChecked == true) {
    alert(fm.sex[i].value);
}
```

复选按钮

通过 checked 属性来获取复选按钮的值。复选按钮也具有 defaultChecked 属性。

```
var love = '';
for (var i = 0; i < fm.love.length; i ++) {
    if (fm.love[i].checked == true) {
        love += fm.love[i].value;
    }
}
alert(love);
```

感谢收看本次教程！

本课程是由北风网(ibeifeng.com)

瓢城 Web 俱乐部(yc60.com)联合提供：

本次主讲老师：李炎恢

我的博客：hi.baidu.com/李炎恢/

我的邮件：yc60.com@gmail.com