附录 E

jQuery 加载并解析 XML

E.1 简述

XML（eXtensible Markup Language）即可扩展标记语言，与 HTML 一样，都是属于 SGML 标准通用语言。在 XML 中，采用了如下语法。

（1）任何起始标签都必须有一个结束标签。

（2）可以采用另一种简化语法，即在一个标签中同时表示起始和结束标签。这种语法是在右边闭合尖括号之前紧跟一个斜线（/），例如<tag/>。XML 解析器会将其翻译成<tag></tag>。

（3）标签必须按照合理的顺序进行嵌套，因此结束标签必须按镜像顺序匹配起始标签，例如 this is a sample string。这相当于将起始和结束标签看作是数学中的左右括号，在没有关闭所有的内部括号之前，不能关闭外面的括号。

（4）所有的属性都需要有值，并且需要在值的周围加上双引号。

E.2 Content-Type

很多情况下 XML 文件不能正常解析都是由于 Content-Type 没有设置好。如果 Content-Type 本身就是一个 XML 文件则不需要设置；如果是由后台程序动态生成的，那么就需要设置 Content-Type 为“text/xml”，否则 jQuery 会以默认的“text/html”方式处理，导致解析失败。以下是几种常见语言中设置 Content-Type 的方式。

```
header("Content-Type:text/xml");          //PHP
response.ContentType="text/xml"           //ASP
response.setContentType("text/xml");      //JSP
```

E.3 XML 结构

作为一个标准的 XML，必须要遵循严格的格式规定，其中最重要的一条规则就是 XML

必须是封闭的。例如如下代码就是错误的，因为它并没有闭合。

```
<?xml version="1.0" encoding="UTF-8"?>
   <name>zhangsan
```

另外 XML 文档只能有一个顶层元素。例如以下代码就是错误的，原因是它有多个顶层元素。

```
<?xml version="1.0" encoding="UTF-8"?>
<name>zhangsan</name>
<id>1</id>
<name>lisi</name>
<id>2</id>
```

一个正确的 XML 应该是下面这样的形式。

```
<?xml version="1.0" encoding="UTF-8"?>
<stulist>
     <student  email="1@1.com">
          <name>zhangsan</name>
          <id>1</id>
     </student>
     <student  email="2@2.com">
          <name>lisi</name>
          <id>2</id>
     </student>
</stulist>
```

E.4 获取 XML

利用上面提到的正确的 XML，通过 jQuery 的 Ajax 函数进行读取，jQuery 代码如下：

```
$.ajax({
     url:'ajax.xml',
     type: 'GET',
     dataType: 'xml',
     timeout: 1000,                    //设定超时
     cache:false,                      //禁用缓存
     error: function(xml){
          alert('加载 XML 文档出错');
```

```
        },
        success: function(xml){
            //这里用于解析 XML
        }
});
```

这样就可以很容易地从后台读取到一段 XML，当然也可以用简单的$.get()方法和$.post()方法来去获取。代码如下：

```
$.get('ajax.xml',function(xml){
    //这里用于解析 XML
});
```

E.5 解析 XML

解析 XML 文档与解析 DOM 一样，也可以用 find()、children()等函数来解析和用 each()方法来进行遍历，另外也可以用 text()和 attr()方法来获取节点文本和属性。例如在 success 回调里解析 XML。代码如下：

```
success: function(xml){
    $(xml).find("student").each(function(i){//查找所有 student 节点并遍历
    var id=$(this).children("id");          //取得子节点
        var id_value=id.text();             //取节点文本
    alert(id_value);                        //这里就是 id 的值了
    alert($(this).attr("email"));           //这里能显示 student 下的 email 属性
    });
}
```

通过上面的代码，能成功获取到相应的数据。接下来就可以将解析出来的数据添加到已有的 HTML 文件中。通常可以先生成一个 DOM 元素片段，然后将数据用 appendTo()函数添加进这个元素片段中，最后将这个片段添加进 HTML 文档中。success 回调代码如下：

```
success: function(xml){
    var frag=$("<ul/>");                          //建立一个代码片段
    $(xml).find("student").each(function(i){//遍历所有 student 节点
        var id=$(this).children("id");            //获取 id 节点
        id_value=id.text();                       //获取节点文本
        email=$(this).attr("email");              //获取 student 下的 email 属性
```

```
        //构造 HTML 字符串，通过 append()方法添加之前建立的代码片段
        frag.append("<li>"+id_value+"-"+email+"</li>");
    });
    frag.appendTo("#load");                    //最后将得到的 frag 添加 HTML 文档中
}
```

E.6 禁用缓存

在项目中经常会遇到一个问题，即数据已经更新了，但传递的还是以前的数据。要避免这种情况，就应当禁用缓存。禁用缓存的方式有很多种。如果是通过$.post()方法获取的数据，那么默认就是禁用缓存的。如果是用了$.get()方法，可以通过设置时间戳来避免缓存。可以在 URL 的后面加上+(+new Date)，代码如下：

```
$.get('ajax.xml?'+(+new Date),function(xml){
    //...
});
// (+new Date)等价于 new Date().getTime()
```

之所以不用随机数，是因为随机数对于同一台电脑来说，在大量使用之后出现重复的概率会很大，而用时间戳则不会出现这种情况。

此外，如果使用了$.ajax()方法来获取数据，只需要设置 cache:false 即可。但要注意，false 是布尔值而不是一个字符串，在这一点上初学者很容易犯错。

掌握了以上内容后，读者就可以顺利地写出符合 XML 语法规范并能正确解析的 XML 文件了。