

# 第6章 jQuery的Ajax操作



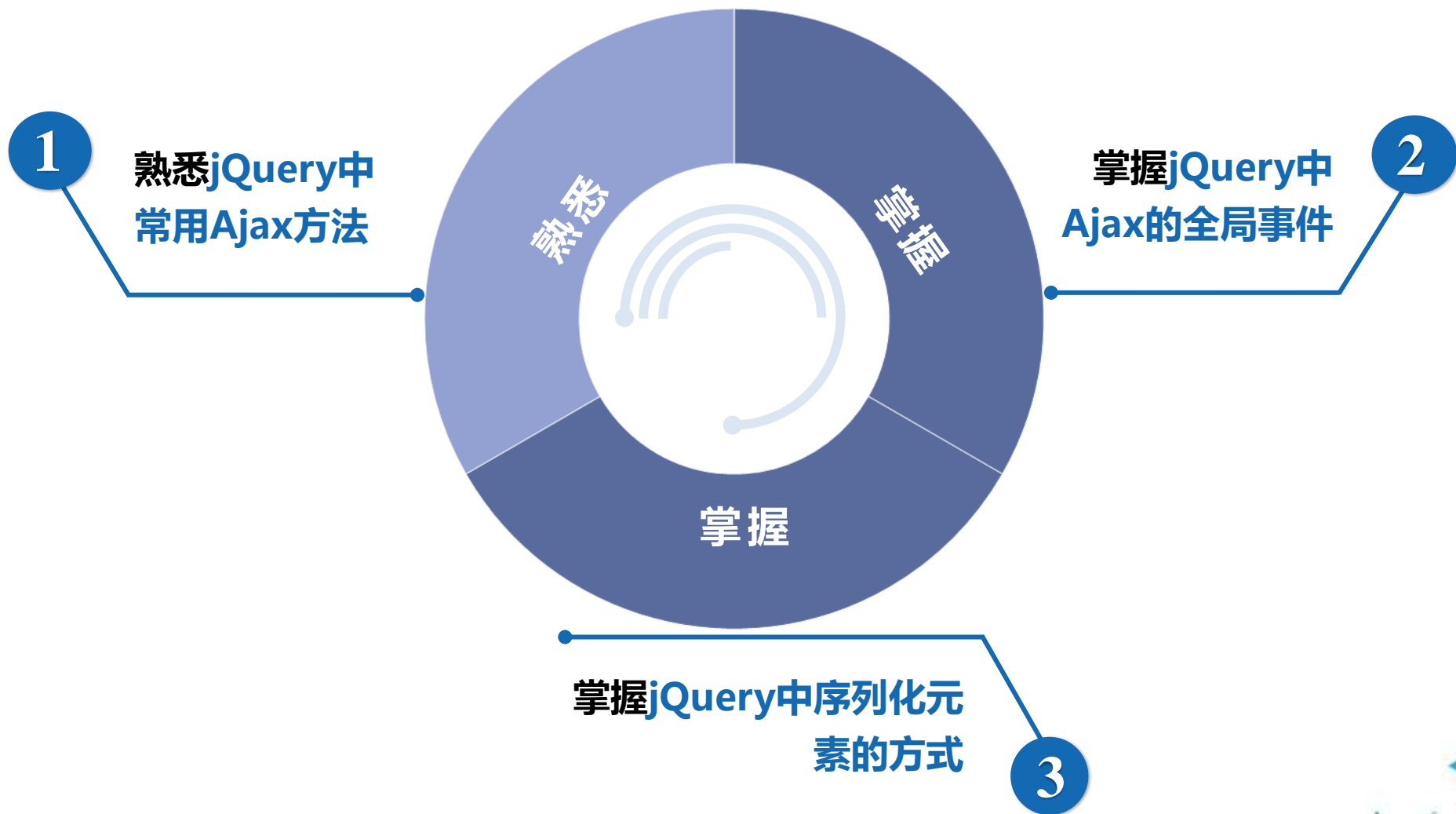
jQuery

- Ajax相关概念
- Ajax事件及相关设置
- 常用Ajax方法
- 图书管理系统




# 学习目标

传智播客·黑马程序员  
改变中国IT教育 我们正在行动





# 目录

 传智播客.黑马程序员  
改变中国IT教育 我们正在行动

6.1

## Ajax简介

 [点击查看本小节知识架构](#)

6.2

## jQuery的Ajax操作

 [点击查看本小节知识架构](#)

6.3

## Ajax底层操作

 [点击查看本小节知识架构](#)



# 目录

传智播客·黑马程序员  
改变中国IT教育 我们正在行动

6.4

## 序列化表单

 [点击查看本小节知识架构](#)

6.5

## 【案例】图书管理系统

 [点击查看本小节知识架构](#)



## 6.1 Ajax简介

### 1. 什么是Ajax

Ajax: 全称是Asynchronous Javascript And XML，即异步的 JavaScript 和 XML。

用途: 在JavaScript、DOM、服务器配合下，浏览器向服务器发出异步请求。

特点:

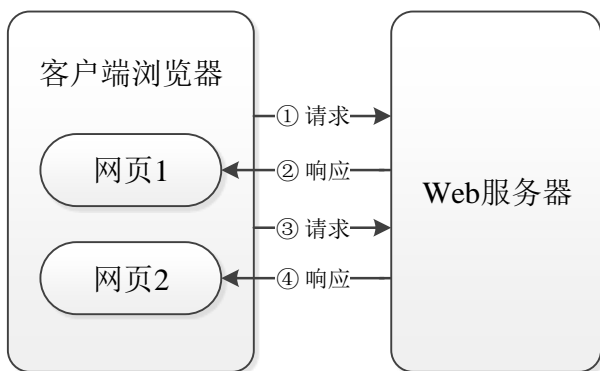
- 异步回调函数；
- Ajax不是一门新的编程语言，而是一种Web应用程序技术



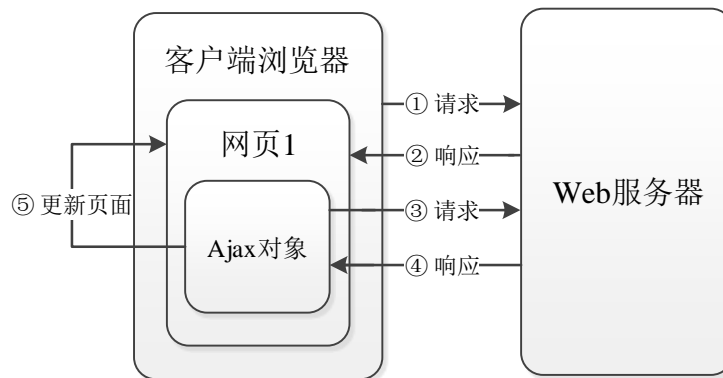
## 6.1 Ajax简介

### 1. 什么是Ajax

传统网页和Ajax技术与服务器通信形式：



传统方式



Ajax方式



## 6.1 Ajax简介

### 1. 什么是Ajax

相较于传统网页，使用**Ajax**技术具有以下几点优势：

- Ajax请求相对请求数据量少
- Ajax请求请求分散，不会集中爆发
- 请求数据时页面不刷新，用户体验良好



## 6.1 Ajax简介

### 2. 搭建WampServer服务器

#### 下载WampServer

下载地址: WampServer官方网站 (<http://www.wampserver.com>)

安装: 按照软件安装提示完成安装工作





## 6.1 Ajax简介

### 2. 搭建WampServer服务器

访问站点目录：通过地址“http://localhost”访问服务器的默认站点

设置站点目录：通过修改文件httpd-vhosts.conf设置默认站点

更改站点目录：将下述配置中的“c:/wamp/www”都替换为站点文件目录

```
DocumentRoot c:/wamp/www  
<Directory "c:/wamp/www/">
```



## 6.1 Ajax简介

### 3. 在HBuilder中配置WampServer服务器

打开HBuilder开发工具，在“外置Web服务器”选项下单击“新建”按钮，即可创建新的Web服务器



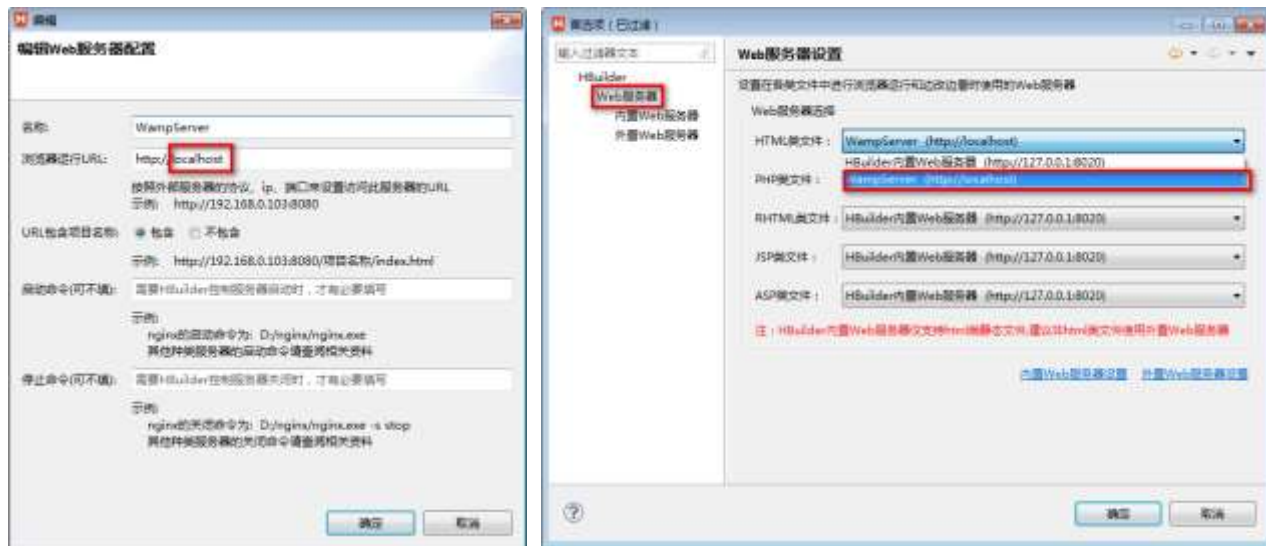


## 6.1 Ajax简介

### 3. 在HBuilder中配置WampServer服务器

#### 设置新的Web服务器:

- ① 在HBuilder窗口中，打开“设置Web服务器”
- ② 单击“新建”按钮，即可创建新的Web服务器





## 6.2 jQuery的Ajax操作

### 1. 加载HTML内容

#### load方法的基本使用:

用途: 请求HTML内容, 并将获得的数据替换到指定元素的内容中

语法: `load(url, [data], [callback]);`

参数:     url - 必需, 规定加载资源的路径; data - 可选, 发送至服务器的数据; callback - 可选, 请求完成时执行的函数。

将路径对应的文件内容插入到`$('#box')`中

```
$('#box').load('target.html');
```



## 6.2 jQuery的Ajax操作

### 1. 加载HTML内容

load方法的url中对文档内容做筛选：如果路径后有选择器，会将文档中的内容按照选择器描述的规则选择出相应的内容插入到执行元素中。

```
$('#box').load( 'target.html h3' );
```



```
$('#box').load( 'target.html' );
```





## 6.2 jQuery的Ajax操作

### 1. 加载HTML内容

#### load方法向服务器发送数据:

如果传入数据作为第二个参数，会将该数据发送至服务器中。服务器端通过获取浏览器发来的数据，经过逻辑判断后返回相应内容。

如下图：

第一个参数为请求路径，第二个参数是发送至服务器数据。

路径

数据

```
$('#box').load('register.php', {username: '小明', password: 18});
```



## 6.2 jQuery的Ajax操作

### 1. 加载HTML内容

服务器端通过获取浏览器发来的数据，经过逻辑判断后返回相应内容，

如下图：

服务器端通过`$_REQUEST`获取浏览器发来的数据，并将数据加工成文档内容返回至服务器。

```
<h3>注册成功</h3>
```

```
<h6>用户名 : <?php echo $_REQUEST['username'];?> </h6>
```

```
<h6>密码 : <?php echo $_REQUEST['password'];?> </h6>
```



## 6.2 jQuery的Ajax操作

### 1. 加载HTML内容

load方法的回调函数：

用途： 在请求成功后执行的函数

参数： responseData - 请求得到的数据； status - 请求状态； xhr - 请求对应的XMLHttpRequest对象；

```
$('#box').load('target.html', function(responseData, status, xhr){  
    console.log(responseData);    // 输出请求得到的数据  
    console.log(status);          // 输出请求状态  
    console.log(xhr);             // 输出XMLHttpRequest对象  
})
```





## 6.2 jQuery的Ajax操作

### 2. 发送GET和POST请求

#### \$.get()方法的基本使用：

用途：按照GET方式与服务器通信

语法：\$.get(url, [data], [function(data, status, xhr)], [dataType]);

参数：

- ❑ url - 必需，规定加载资源的路径；
- ❑ data - 可选，发送至服务器的数据；
- ❑ function(data, status, xhr) - 可选，请求完成时执行的函数；
- ❑ dataType - 可选，预期的服务器响应的数据类型



## 6.2 jQuery的Ajax操作

### 2. 发送GET和POST请求

**function(data, status, xhr):** 请求成功时执行的函数

- ❑ data表示从服务器返回的数据；
- ❑ status表示请求的状态值；
- ❑ xhr表示当前请求相关的XMLHttpRequest对象；

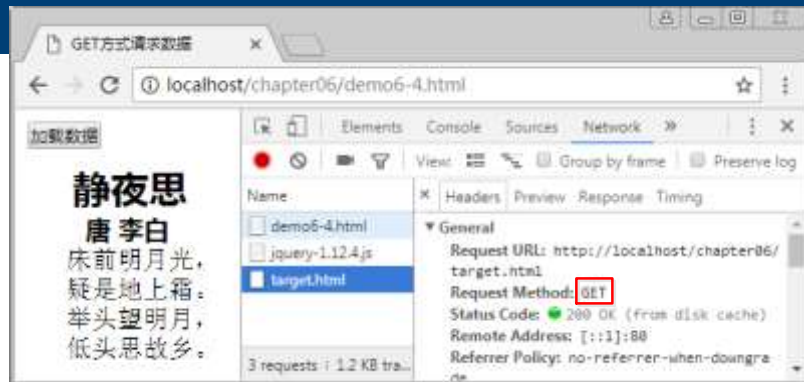


## 6.2 jQuery的Ajax操作

### 2. 发送GET和POST请求

使用\$.get()方法请求数据: 传入路径作为第一个参数, 即向服务器请求数据, 并在回调函数中获取数据。

```
$.get('target.html', function(data) {  
    $('#box').html(data); //通过data参数得到取得的数据  
}, 'html');
```



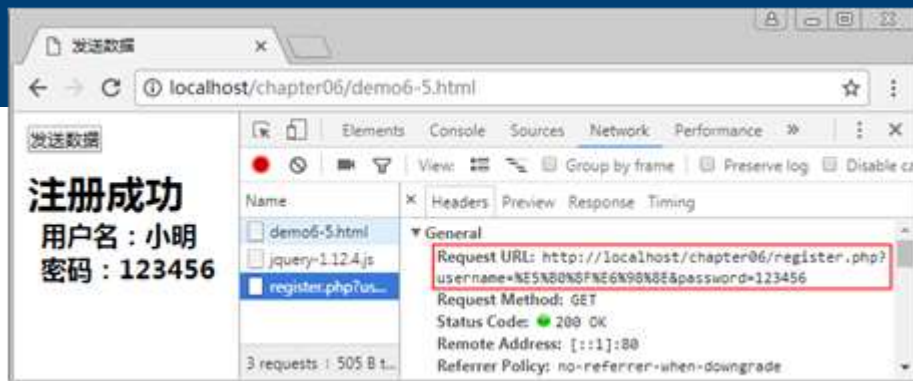


## 6.2 jQuery的Ajax操作

### 2. 发送GET和POST请求

使用\$.get()方法发送数据：需要使用第二参数来传递数据。

```
var userData = {username: '小明', password: 123456};  
$.get('register.php', userData, function(data) {  
    $('#box').html(data);  
}, 'html');
```





## 6.2 jQuery的Ajax操作

### 2. 发送GET和POST请求

#### \$.post()方法的基本使用:

用途: 按照POST方式与服务器通信

语法: \$. post(url, [data], [function(data, status, xhr)], [dataType]);

参数:

- ❑ url - 必需, 规定加载资源的路径;
- ❑ data - 可选, 发送至服务器的数据;
- ❑ function(data, status, xhr) - 可选, 请求完成时执行的函数;
- ❑ dataType - 可选, 预期的服务器响应的数据类型;



## 6.2 jQuery的Ajax操作

### 2. 发送GET和POST请求

**function(data, status, xhr):** 请求成功时执行的函数

参数:

- ❑ data表示从服务器返回的数据
- ❑ status表示请求的状态值
- ❑ xhr表示当前请求相关的XMLHttpRequest对象



## 6.2 jQuery的Ajax操作

### 2. 发送GET和POST请求

**\$.get()**方法和**\$.post()**方法使用方式完全相同，不同的地方有：

- ❑ 请求方式不同，**\$.get()**方法为GET方式，**\$.post()**方法为POST方式
- ❑ 传入路径作为第一个参数，即向服务器请求数据，并在回调函数中获取数据。

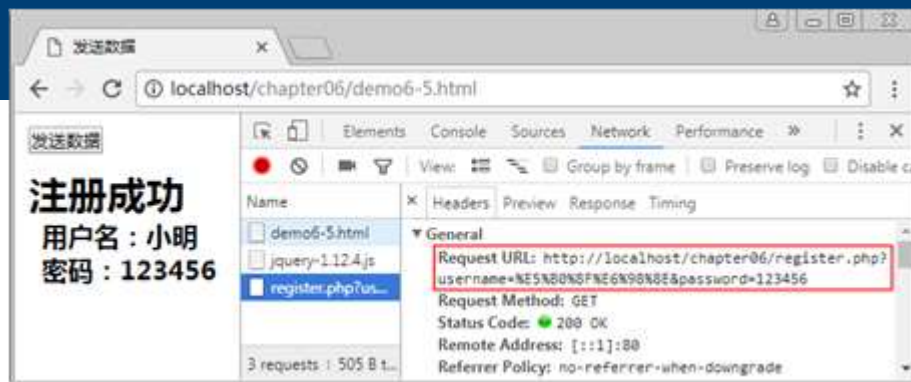


## 6.2 jQuery的Ajax操作

### 2. 发送GET和POST请求

使用\$.get()方法发送数据：需要使用第二参数来传递数据

```
var userData = {username: '小明', password: 123456};  
$.get('register.php', userData, function(data) {  
    $('#box').html(data);  
}, 'html');
```





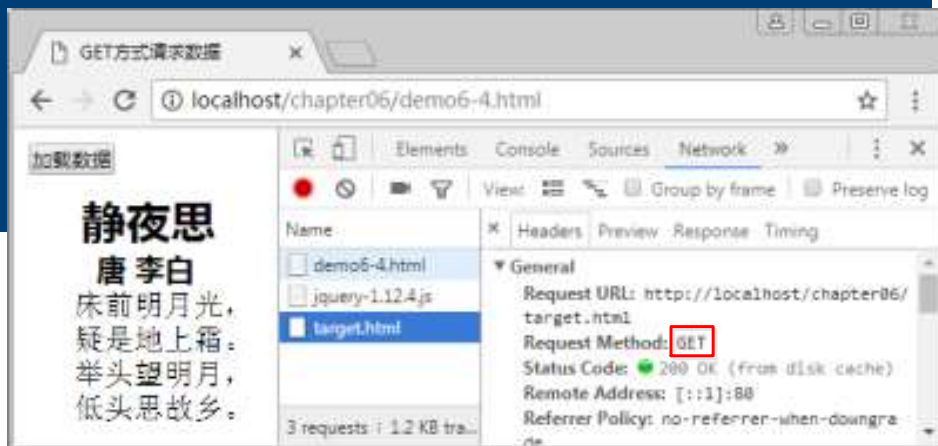


## 6.2 jQuery的Ajax操作

### 2. 发送GET和POST请求

使用\$.get()方法请求数据:

```
<button id="btn">加载数据</button>
<div id="box"></div>
<script>
    $('#btn').click(function() {
        $.get('target.html', function(data) {
            $('#box').html(data);
        }, 'html');
    });
</script>
```





## 6.2 jQuery的Ajax操作

### 2. 发送GET和POST请求

#### \$.get()方法和\$.post()方法的区别:

- 发送数据的方式不同: GET方式作为URL参数发送至服务器, POST方式作为请求实体发送至服务器。
- 发送数据的内容大小不同: GET方式在2KB到8KB之间, POST方式理论上内容大小没有限制。
- 数据的安全性: GET方式将数据作为查询字符串加在了请求地址后面, 较容易被他人读取。POST方式将数据作为请求实体发送, 所以更为安全。



## 6.2 jQuery的Ajax操作

### 3. 数据格式处理

数据格式概述：服务器返回的数据，会遵循一种规范的数据格式，如XML、JSON和TEXT等。通过数据格式来保存数据，可以确保JavaScript程序能够正确解析、识别这些数据。

jQuery中针对不同的数据格式会采取不同的处理方式，接下来将介绍目前最常见的XML、JSON和TEXT这3种数据格式的处理方式。



## 6.2 jQuery的Ajax操作

### 3. 数据格式处理

XML数据格式: XML采用双标签嵌套来记录信息，文件格式类似于HTML文档。

```
<booklist>
  <book>
    <name>JavaScript高级程序设计</name>
    <price>¥78.20</price>
    <author>扎卡斯</author>
  </book>
  <book>....</book>
</booklist>
```

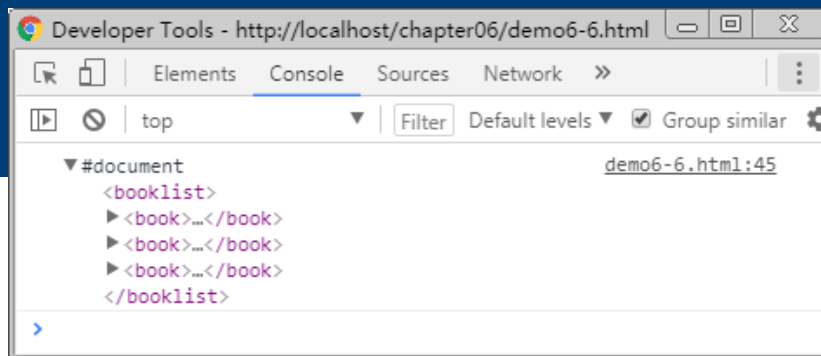


## 6.2 jQuery的Ajax操作

### 3. 数据格式处理

使用\$.get()方法获取XML数据格式: 控制台中输出了从服务器获取的数据，  
获取到的数据是文档对象。

```
$.get('target.xml', function(data) {  
    console.log(data);  
}, 'xml');
```





## 6.2 jQuery的Ajax操作

### 3. 数据格式处理

#### 解析XML数据

```
$.get('target.xml', function(data, status, xhr) {  
    var html = '';  
    $(data).find('book').each(function(index, ele) {  
        html += '<tr>';  
        $(ele).children().each(function(index, ele) {  
            html += '<td>' + $(ele).text() + '</td>';  
        });  
        html += '</tr>';  
    });  
    $( '#dataTable' ).append(html); //将拼合的数据插入到指定元素中  
}, 'xml');
```



The screenshot shows a web browser window with the address bar displaying 'localhost/chapter06/demo6-6.html'. Below the address bar, there is a button labeled '加载数据'. Below the button is a table with three columns: '书名' (Book Name), '作者' (Author), and '价格' (Price). The table contains three rows of data.

书名	作者	价格
JavaScript高级程序设计	扎卡斯	¥78.20
HTML5移动Web开发	黑马程序员	¥39.50
MongoDB设计模式	科普兰	¥28.40



## 6.2 jQuery的Ajax操作

### 3. 数据格式处理

在实际开发中，XML数据格式的优点在于：

- ❑ 可以轻松的跨平台应用，便于信息的检索，
- ❑ 同时具有很强的可扩展性。
- ❑ 但相比JSON格式，在提取信息时较为不便。



## 6.2 jQuery的Ajax操作

### 3. 数据格式处理

**JSON数据格式:** 一种key/value（键值对）数据格式，类似于JavaScript的对象格式。它的优势在于，能被处理成对象，方便获取信息字段。

target.json

```
[  
  {"name": "JavaScript高级程序设计", "author": "扎卡斯", "price": "¥78.20"},  
  {"name": "HTML5移动Web开发", "author": "黑马程序员", "price": "¥39.50"},  
  {"name": "MongoDB设计模式", "author": "科普兰", "price": "¥28.40"}  
]
```





## 6.2 jQuery的Ajax操作

### 3. 数据格式处理

注意

需要注意的是，JSON数据格式要求键名必须使用双引号包裹。

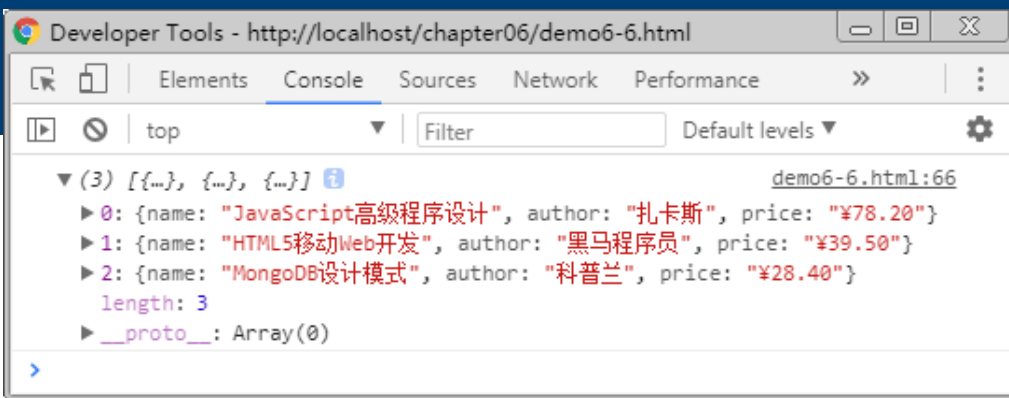


## 6.2 jQuery的Ajax操作

### 3. 数据格式处理

获取JSON数据格式: 控制台中输出了从服务器获取的数据，获取到的数据是元素为对象的数组。

```
$.get('target.json', function(data) {  
    console.log(data);  
}, 'json');
```





## 6.2 jQuery的Ajax操作

### 3. 数据格式处理

JSON数据以对象的形式获取相应信息

```
$.get('target.json', function(data) {  
    var html = '';  
    for (var i = 0; i < data.length; ++i) {  
        html += '<tr>';  
        for (var key in data[i]) {  
            html += '<td>' + data[i][key] + '</td>';  
        }  
        html += '</tr>';  
    }  
    $('#dataTable').append(html);  
}, 'json');
```



The screenshot shows a web browser window with the address bar displaying 'localhost/chapter06/demo6-6.html'. The page contains a button labeled '加载数据' (Load Data) and a table with three columns: '书名' (Book Name), '作者' (Author), and '价格' (Price). The table contains three rows of data.

书名	作者	价格
JavaScript高级程序设计	扎卡斯	¥78.20
HTML5移动Web开发	黑马程序员	¥39.50
MongoDB设计模式	科普兰	¥28.40



## 6.2 jQuery的Ajax操作

### 3. 数据格式处理

TEXT数据格式: TEXT数据格式就是最基本的字符串。

在jQuery中得到这部分字符串之后，可以通过一定的处理方式，将信息处理成需要的形式。

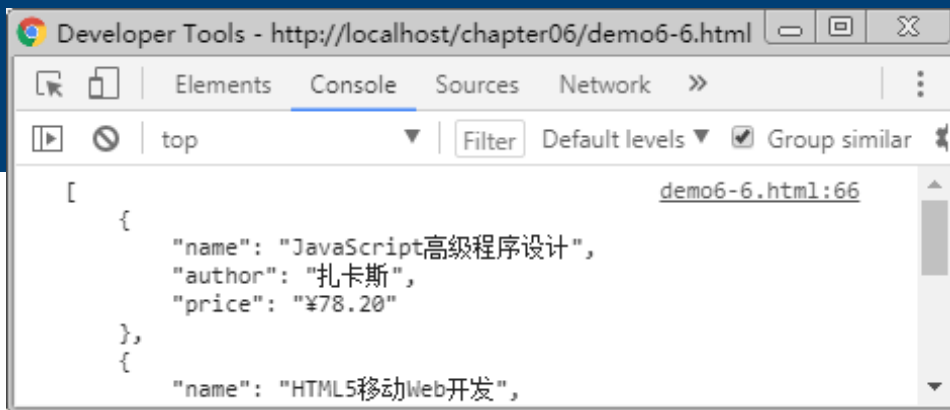


## 6.2 jQuery的Ajax操作

### 3. 数据格式处理

获取TEXT数据格式: 控制台中输出了从服务器获取的数据，获取到的数据是字符串。

```
$.get('target.txt', function(data) {  
    console.log(data);  
}, 'text');
```



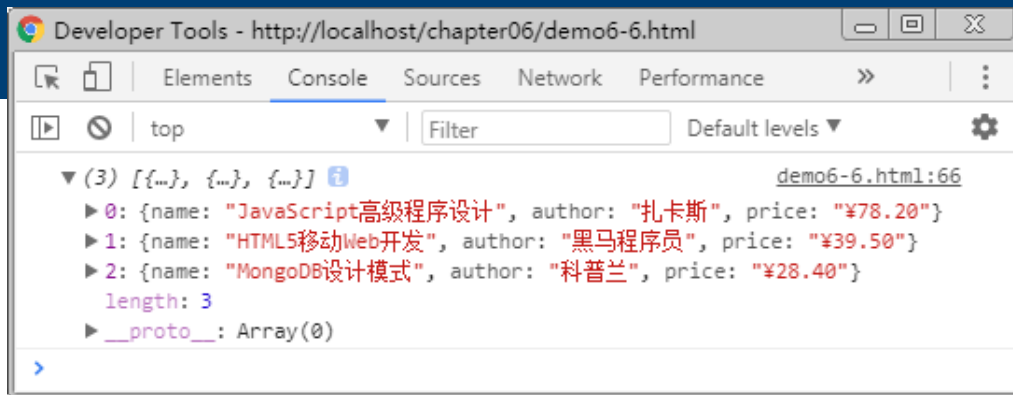


## 6.2 jQuery的Ajax操作

### 3. 数据格式处理

处理TEXT数据格式: 使用JSON.parse()方法可以把具有对象格式的字符串转化为成为对象。

```
$.get('target.txt', function(data) {  
    console.log(JSON.parse(data));  
}, 'text');
```





## 6.2 jQuery的Ajax操作

### 4. 获取JSON数据

#### \$.getJSON()方法的使用:

用途: 获取JSON数据

语法: `$.getJSON(url, [data], [callback]);`

参数:

- ❑ url - 必需，规定加载资源的路径；
- ❑ data - 可选，发送至服务器的数据；
- ❑ callback - 可选，请求完成时执行的函数；



## 6.2 jQuery的Ajax操作

### 4. 获取JSON数据

使用\$.getJSON()方法时不用传入数据类型参数

```
$.getJSON('target.json', function(data, statu, xhr) {  
    var html = '';  
    for (var book in data) {  
        html += '<tr>';  
        for (var key in data[book]) {  
            html += '<td>' + data[book][key] + '</td>';  
        }  
        html += '</tr>';  
    }  
    $('#dataTable').append(html);  
});
```



The screenshot shows a web browser window with the address bar displaying 'localhost/chapter06/demo6-6.html'. Below the address bar, there is a button labeled '加载数据'. Below the button is a table with three columns: '书名' (Book Name), '作者' (Author), and '价格' (Price). The table contains three rows of data.

书名	作者	价格
JavaScript高级程序设计	扎卡斯	¥78.20
HTML5移动Web开发	黑马程序员	¥39.50
MongoDB设计模式	科普兰	¥28.40





## 6.2 jQuery的Ajax操作

### 4. 获取JSON数据

#### 跨域请求相关概念:

跨域: 在网络中, 协议、域名、端口号有任何一个不同都属于都属于不同的域。而跨域就是指一个域的页面请求另外一个域的资源。

同源策略: 出于安全考虑, 浏览器限制了跨域行为, 只允许页面访问本域的资源



## 6.2 jQuery的Ajax操作

### 4. 获取JSON数据

#### □ 模拟实现JSONP跨域请求：

- 通过变量\$callback接收来自URL参数中的callback回调函数名；
- 对变量进行了字符串拼接，拼接结果为“回调函数名(123);”
- “123”是返回给浏览器的数据，实际开发时根据需求换成其他数据；
- “回调函数名(123);”是函数调用的形式，如果页面中已经存在“回调函数名”的函数，则会通过参数传递而得到数据“123”；

```
$callback = $_GET['callback'];  
echo "$callback(123);";
```

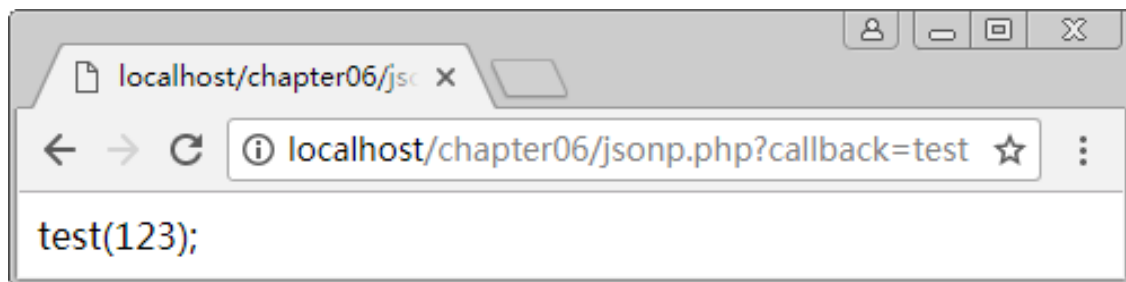


## 6.2 jQuery的Ajax操作

### 4. 获取JSON数据

通过浏览器访问<http://localhost/chapter06/jsonp.php?callback=test>

**test(123):** 是函数执行的写法规则，如果页面上已经存在**test**函数，并将此字符串作为代码来执行，则页面上的**test**函数会接收到服务器传来的数据**123**





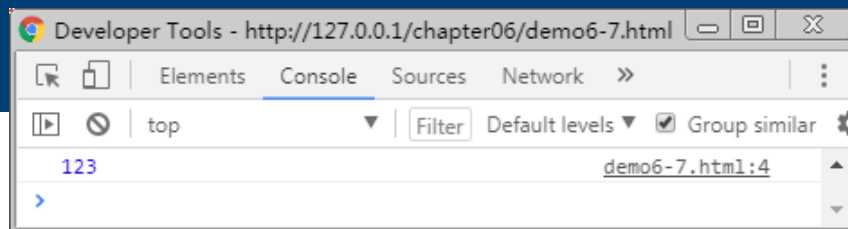
## 6.2 jQuery的Ajax操作

### 4. 获取JSON数据

使用\$.getJSON进行跨域请求: 在请求地址后添加“callback=?”即可实现跨域

使用\$.getJSON跨域请求jsonp.php

```
var url = 'http://localhost/chapter06/jsonp.php?callback=?';  
$.getJSON(url, function(data) {  
    console.log(data);  
});
```



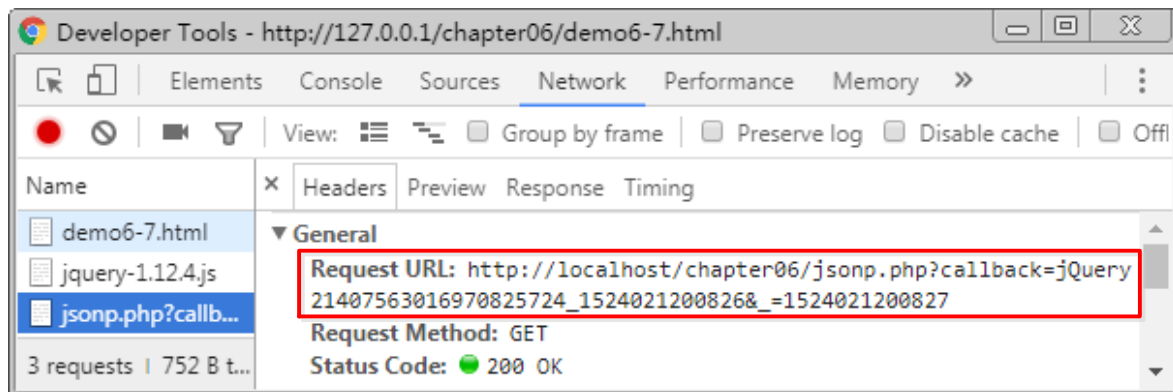


## 6.2 jQuery的Ajax操作

### 4. 获取JSON数据

jQuery对跨域的处理：Network面板，可以看到\$.getJSON()发送的URL参数。

\$.getJSON()发送了两个URL参数，分别是“callback”和“\_”，第1个参数是自动生成的回调函数名，第2个参数用于解决Ajax缓存问题。

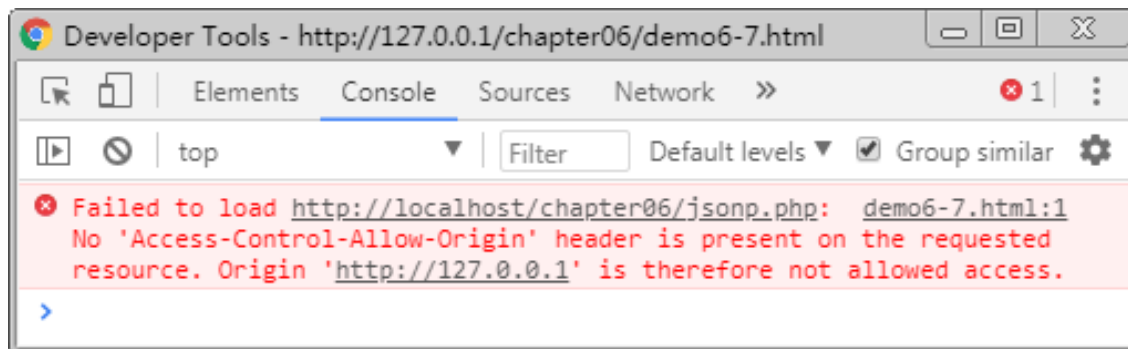




## 6.2 jQuery的Ajax操作

### 4. 获取JSON数据

不作跨域设置时跨域请求出错误：将URL参数callback删除后，通过浏览器访问测试，会看到跨域请求失败的错误提示。





## 6.2 jQuery的Ajax操作

### 5. 获取JavaScript代码并执行

#### \$.getScript()方法的使用:

用途: 获取JavaScript代码并执行

语法: `$.getScript(url, [data], [callback]);`

参数:

- ❑ url - 必需, 规定加载资源的路径;
- ❑ data - 可选, 发送至服务器的数据;
- ❑ callback - 可选, 请求完成时执行的函数;



## 6.2 jQuery的Ajax操作

### 5. 获取JavaScript代码并执行

案例演示: 使用\$.getScript()方法获取createEle.js文件，利用其创建元素。

createEle.js

```
function createEle(tagName, styleOptions) {  
    var newEle = document.createElement(tagName);  
    for (var key in styleOptions) {  
        newEle.style[key] = styleOptions[key];  
    }  
    return newEle;  
}
```



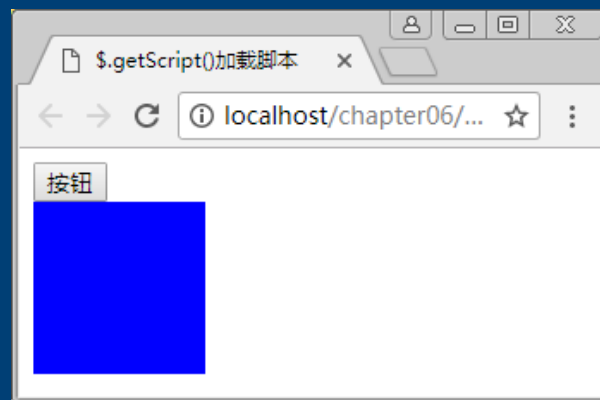


## 6.2 jQuery的Ajax操作

### 5. 获取JavaScript代码并执行

获取createEle.js文件，创建蓝色div

```
$.getScript('lib/createEle.js',function() {  
    var styleObj = {  
        width: '100px',  
        height: '100px',  
        background: 'blue'  
    };  
    var divBox = createEle('div', styleObj);  
    $('body').append(divBox);  
});
```





## 6.3 Ajax的底层操作

### 1. \$.ajax()的基本使用

#### \$.ajax()相关概念:

用途: jQuery中所有Ajax方法都是基于该方法实现, 是jQuery中最底层Ajax方法, 可用于实现任何形式的Ajax请求。

语法: \$.ajax(options) 或\$.ajax(url, [options]);

#### 参数:

- url - 必需, 规定加载资源的路径;
- options - 对象, 该对象将Ajax请求需要的设置包含在属性中;



## 6.3 Ajax的底层操作

### 1. \$.ajax()的基本使用

使用\$.ajax()方法封装\$.get()、\$.post()方法

```
jQuery.each([ "get", "post" ], function( i, method ) {  
    jQuery[ method ] = function( url, data, callback, type ) {  
        if ( jQuery.isFunction( data ) ) {  
            type = type || callback; callback = data; data = undefined;  
        }  
        return jQuery.ajax({  
            url: url, type: method, dataType: type, data: data, success: callback  
        });  
    };  
});
```

封装\$.get()、\$.post()方法

使用\$.ajax()方法实现功能



## 6.3 Ajax的底层操作

### 1. \$.ajax()的基本使用

\$.ajax()方法部分参数:

参数	描述
url	请求地址，默认是当前页面
data	发送至服务器的数据
xhr	用于创建XMLHttpRequest对象的函数
callback	请求完成时执行的函数
dataType	预期的服务器响应的数据类型



## 6.3 Ajax的底层操作

### 1. \$.ajax()的基本使用

实现Ajax的基本操作: 分别通过\$.get()方法、 \$.ajax()方法请求HTML文件

// 利用\$.get()方法实现

```
$.get('target.html',function(data) {  
    $('#box').html(data);  
}, 'html');
```

// 利用\$.ajax()方法实现

```
$.ajax({  
    url: 'target.html',  
    success: function(data) {  
        $('#box').html(data);  
    },  
    dataType: 'html'  
});
```



## 6.3 Ajax的底层操作

### 1. \$.ajax()的基本使用

实现跨域请求：分别通过\$.getJSON()方法、\$.ajax()方法实现跨域请求

```
//使用$.getJSON()方法实现跨域请求
$.getJSON('http://localhost/chapter06/jsonp.php?callback=?',
    function(data) {
        console.log(data);
    }
);
```



## 6.3 Ajax的底层操作

### 1. \$.ajax()的基本使用

实现跨域请求：分别通过\$.getJSON()方法、\$.ajax()方法实现跨域请求

//使用\$.ajax()方法实现跨域请求

```
$.ajax({  
    url: 'http://localhost/chapter06/jsonp.php',  
    dataType: 'jsonp',  
    jsonp: 'callback',  
    success: function(data) {  
        console.log(data);  
    }  
});
```

表示发出JSONP请求时指定  
参数名称，形如“callback=?”



## 6.3 Ajax的底层操作

### 2. Ajax相关事件

#### 全局事件：

概述：全局事件会被页面上任意一个Ajax请求触发，当Ajax请求达到了某个请求过程时，就会触发相应的全局事件。

语法：\$(document).事件名(fn);

参数：fn是事件触发时执行的回调函数，在使用时，不同事件的回调函数会接收到不同的参数。





## 6.3 Ajax的底层操作

### 2. Ajax相关事件

全局事件参数：event表示当前事件对象，xhr表示与请求对应的XMLHttpRequest对象，options表示触发该事件的请求的各项配置，ex表示请求发生错误时的错误描述信息。

事件方法	描述
ajaxStart(fn())	请求开始时
ajaxStop(fn())	请求结束时
ajaxSuccess(fn(event,xhr,options))	请求成功时
ajaxError(fn(event,xhr,options,ex))	请求发生错误时
ajaxSend(fn(event,xhr,options))	请求发送前



## 6.3 Ajax的底层操作

### 2. Ajax相关事件

注意

Ajax的全局事件在jQuery1.8版本之前可绑定在普通DOM对象上，而在jQuery1.8版本之后只能绑定在文档对象document上。



## 6.3 Ajax的底层操作

### 2. Ajax相关事件

form获取用户数据，table显示请求状态、返回数据

```
<form>
  <fieldset>
    用户名 : <input type="text" id="username"> <br>
    密码 : <input type="text" id="password"> <br>
  </fieldset>
</form>
<button id="btn">提交数据</button>
<table border="1" cellpadding="0" cellspacing="0">
  <tr><td>状态 : </td><td id="status"> </td></tr>
  <tr><td>返回数据 : </td><td id="data"> </td></tr>
</table>
```



## 6.3 Ajax的底层操作

### 2. Ajax相关事件

全局事件：通过一个案例演示全局事件的使用

单击按钮时获取用户输入数据，发出请求

```
$('#btn').click(function() {  
    var userData = {  
        username: $('#username').val(),  
        password: $('#password').val()  
    };  
    $.post('register.php', userData, function(data) {  
        $('#data').html(data);  
    });  
});
```



## 6.3 Ajax的底层操作

### 2. Ajax相关事件

全局事件：jQuery分别绑定了ajaxStart和ajaxStop事件，请求开始和结束时分别给id为status的元素设置不同的内容，表示事件被触发。

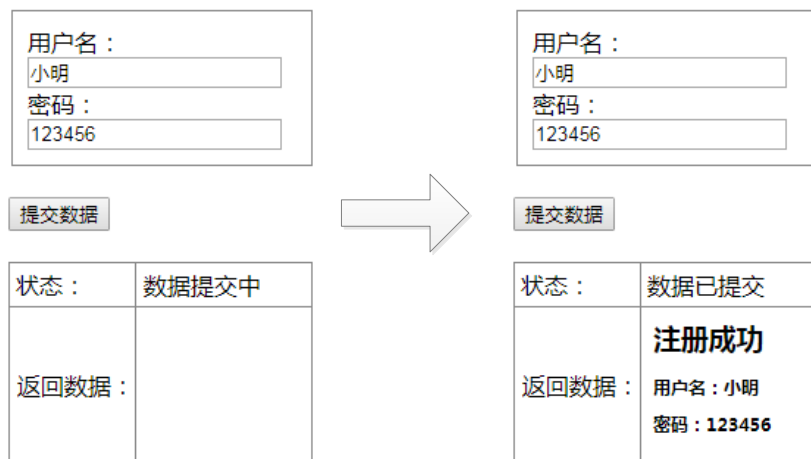
```
$(document).ajaxStart(function() {  
    $('#status').html('数据提交中');  
});  
$(document).ajaxStop(function() {  
    // 弹出警告框时会暂停脚本，此时可观察状态文本  
    alert('测试');  
    $('#status').html('数据已提交');  
});
```



## 6.3 Ajax的底层操作

### 2. Ajax相关事件

全局事件：在Ajax请求开始时，触发ajaxStart事件，将id为status的元素的内容设置为“数据提交中”。在请求结束后，又将该元素的内容设置为“数据已提交”，并在id为data的元素中填入获得的数据。

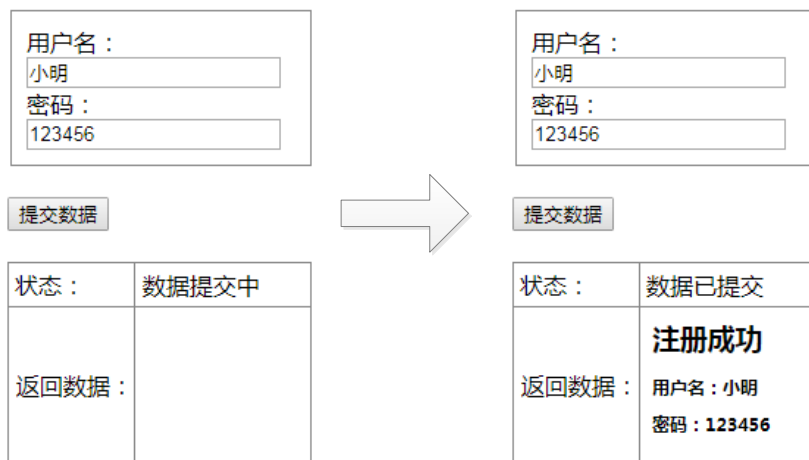




## 6.3 Ajax的底层操作

### 2. Ajax相关事件

全局事件：加载过快无法观察到状态文本变化，设置弹出框来中断程序运行，则可观察到状态文本为“数据提交中”，关闭警告框，状态文本变为“数据已提交”。





## 6.3 Ajax的底层操作

### 2. Ajax相关事件

#### 局部事件：

概述：Ajax的局部事件只与某一个具体请求相关，在指定请求的对应时间节点才能被触发。

语法：通过\$.ajax()方法中参数对象来设置





## 6.3 Ajax的底层操作

### 2. Ajax相关事件

局部事件：参数：xhr表示该请求对应的Ajax对象；result表示从服务器返回的数据；status表示错误信息；error表示请求错误描述。

事件方法	描述
beforeSend(xhr)	发送请求前执行的函数
success(result,status,xhr)	请求成功时执行的函数
error(xhr,status,error)	请求失败时执行的函数
complete(xhr,status)	请求完成时运行的函数（请求成功或失败时都会调用，顺序在success和error 函数之后）
beforeSend(xhr)	发送请求前执行的函数



## 6.3 Ajax的底层操作

### 2. Ajax相关事件

局部事件：演示局部事件的设置方式，同时对比其和全局事件的区别

demo6-10.html 文档结构：id名为“btn1”和“btn2”的按钮

```
<button id="btn1">发送请求1</button>
```

```
<button id="btn2">发送请求2</button>
```



## 6.3 Ajax的底层操作

### 2. Ajax相关事件

局部事件：演示局部事件的设置方式，同时对比其和全局事件的区别

分别绑定Ajax局部事件

```
$('#btn1').click(function() {  
    $.ajax({type: 'get', url: 'register.php', complete: function() {  
        console.log('请求1-局部事件-complete');  
    });  
});  
  
$('#btn2').click(function() {  
    $.ajax({type: 'get', url: 'register.php', complete: function() {  
        console.log('请求2-局部事件-complete');  
    });  
});
```



## 6.3 Ajax的底层操作

### 2. Ajax相关事件

局部事件：演示局部事件的设置方式，同时对比其和全局事件的区别

demo6-10.html 页面行为：绑定Ajax全局事件

```
$(document).ajaxComplete(function() {  
    console.log('全局事件-complete');  
});
```



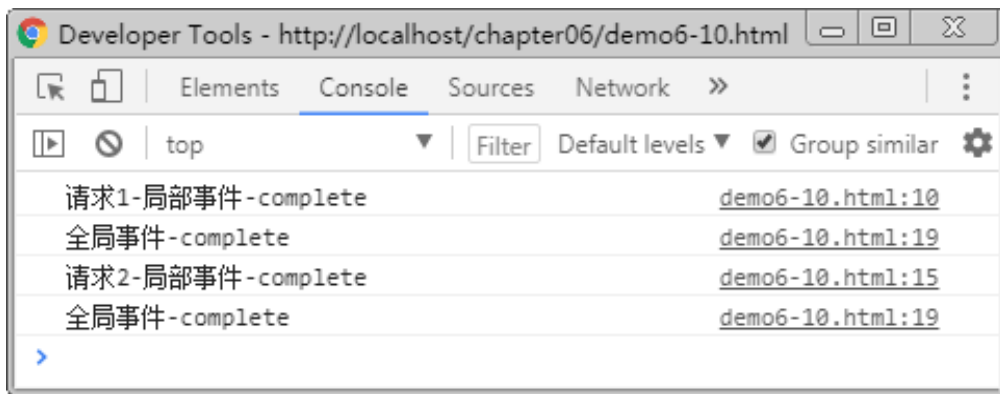
## 6.3 Ajax的底层操作

### 2. Ajax相关事件

局部事件：演示局部事件的设置方式，同时对比其和全局事件的区别

单击两个按钮后观察发现：

- ❑ 请求执行时，只会触发自身的局部事件
- ❑ 任意请求执行时，都会触发全局事件





## 6.3 Ajax的底层操作

多

学

一

招

### Ajax错误处理

jQuery提供的Ajax方法，允许在发送请求后再对结果进行处理。

例如，在调用\$.post()方法后，链式调用done()、fail()和always()方法。

```
$.post('register.php', function() {  
    console.log('success');  
}).done(function() {console.log('done');  
}).fail(function() {console.log('fail');  
}).always(function() {console.log('always');  
});
```



## 6.3 Ajax的底层操作

多

学

一

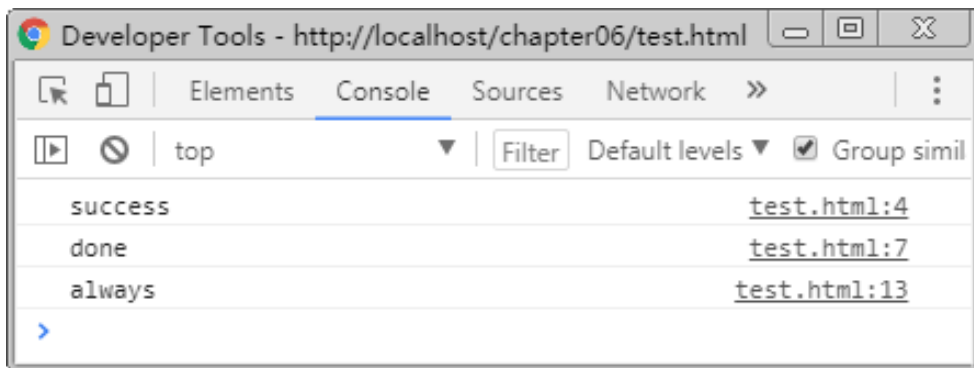
招

### Ajax错误处理

在请求成功时：

`$.post()`方法的回调函数的执行时机，早于`done()`方法的回调函数。

由于请求过程中没有发生错误，因此`fail()`方法的回调函数没有执行。





## 6.3 Ajax的底层操作

多

学

一

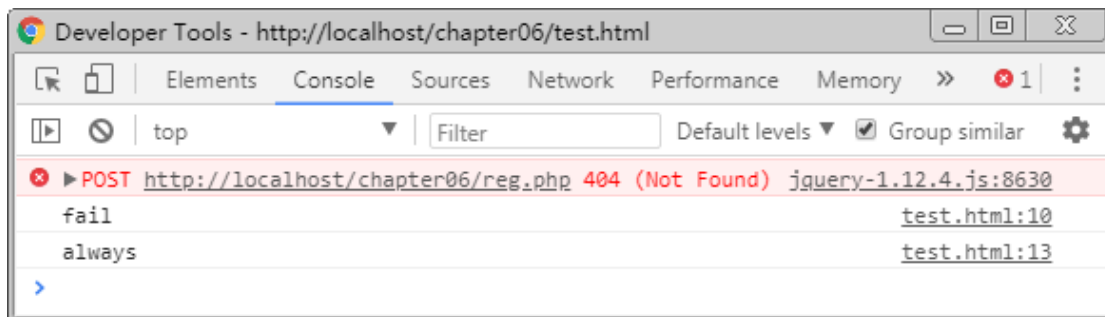
招

### Ajax错误处理

在请求失败时：

`$.post()`方法、`done()`方法的回调函数没有执行，`fail()`方法的回调函数执行了。

因此，在开发中可以将错误处理的操作放在`fail()`方法的回调函数中。







## 6.3 Ajax的底层操作

### 3. Ajax全局配置

#### 全局配置：

概述：对所有的Ajax请求的相关参数进行统一的设置

语法：\$.ajaxSetup()、\$.ajaxPrefilter()

目的：减少代码冗余



## 6.3 Ajax的底层操作

### 3. Ajax全局配置

#### \$.ajaxSetup() :

概述: 为Ajax请求设置默认参数值, 该方法设置的参数值适用于所有的Ajax请求。

语法: \$.ajaxSetup(options)

参数: options参数的使用方法与\$.ajax()完全相同



## 6.3 Ajax的底层操作

### 3. Ajax全局配置

通过一个案例演示\$.ajaxSetup()的使用:

demo6-11.html 文档内容: id名为“btn1”和“btn2”的按钮

```
<button id="btn1">发送请求1</button>  
<button id="btn2">发送请求2</button>
```



## 6.3 Ajax的底层操作

### 3. Ajax全局配置

通过一个案例演示\$.ajaxSetup()的使用：

demo6-11.html 页面行为：使用进行Ajax全局设置

```
$.ajaxSetup({  
    type: 'post',  
    url: 'register.php',  
    data: {username: 'btn1', password: 1}  
});
```



## 6.3 Ajax的底层操作

### 3. Ajax全局配置

通过一个案例演示\$.ajaxSetup()的使用：

demo6-11.html 页面行为： 分别为id为btn1和btn2的两个按钮绑定单击事件，在事件触发时发出Ajax请求。

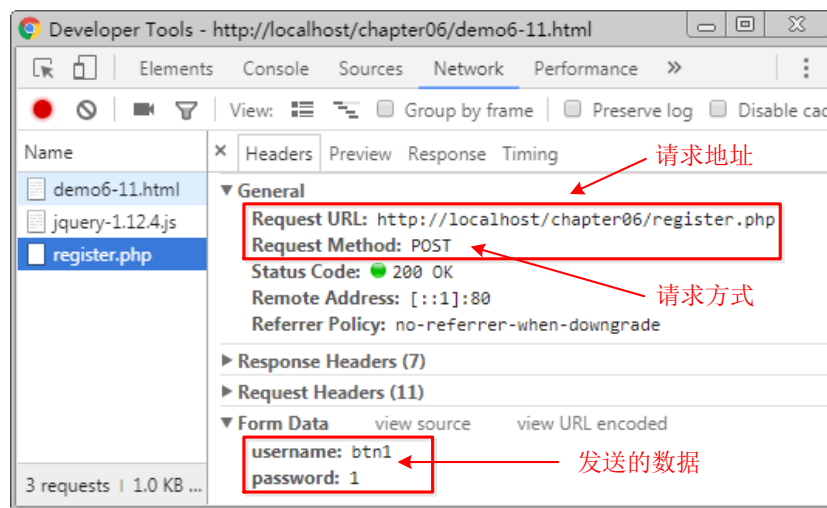
```
$('#btn1').click(function() {  
    $.ajax();  
});  
$('#btn2').click(function() {  
    $.ajax({data: {username: 'btn2', password: 2}});  
});
```



## 6.3 Ajax的底层操作

### 3. Ajax全局配置

单击id名为“btn1”的元素得到结果为：请求地址为“register.php”，请求类型为“post”，发送的数据为“{username: 'btn1', password: 1}”，数据和\$.ajaxSetup()方法设置的参数默认值相同。

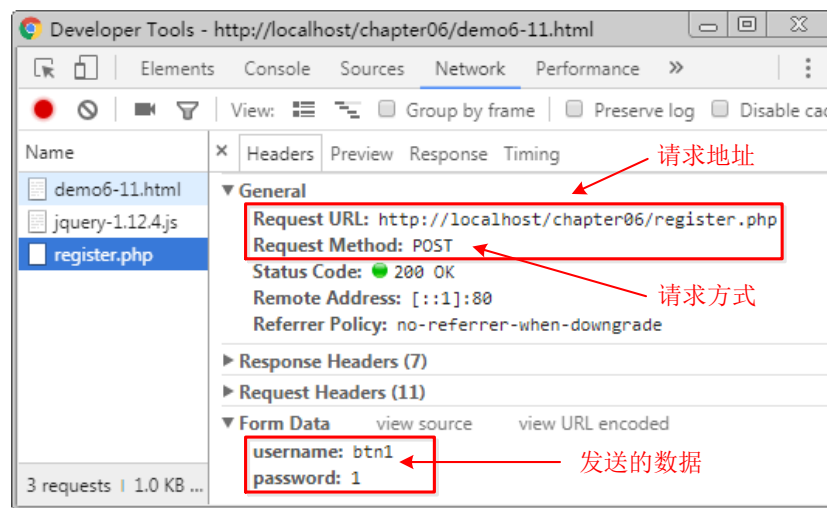




## 6.3 Ajax的底层操作

### 3. Ajax全局配置

单击id名为“btn2”的元素得到结果为：除了发送的数据不同，请求地址和请求方式都与\$.ajaxSetup()设置的默认参数值相同。由此可见，在发出请求时方法设置的参数会覆盖\$.ajaxSetup()方法设置的参数默认值。





## 6.3 Ajax的底层操作

### 3. Ajax全局配置

#### \$.ajaxPrefilter():

**概述：** 预先处理Ajax参数，回调函数会在Ajax请求发出的之前，重新处理Ajax参数。

**语法：** \$.ajaxPrefilter([dataType], handler(options, originalOptions, xhr))

**参数：** options参数的使用方法与\$.ajax()完全相同





## 6.3 Ajax的底层操作

### 3. Ajax全局配置

#### \$.ajaxPrefilter()参数:

参数	描述
dataType	需要处理何种请求数据类型的Ajax
handler	对Ajax参数选项预处理的函数

#### 回调函数handler参数:

参数	描述
options	当前Ajax请求的所有参数选项
originalOptions	给\$.ajax()方法的未经修改的参数选项
xhr	当前请求的XMLHttpRequest对象



## 6.3 Ajax的底层操作

### 3. Ajax全局配置

通过一个案例演示\$.ajaxPrefilter()的使用:

demo6-12.html 文档内容: id名为“btn1”和“btn2”的按钮

```
<button id="btn1">发送请求1</button>  
<button id="btn2">发送请求2</button>
```



## 6.3 Ajax的底层操作

### 3. Ajax全局配置

通过一个案例演示\$.ajaxPrefilter()的使用：

demo6-12.html 文档内容：使用\$.ajaxPrefilter()方法处理请求数据类型为

“html”的Ajax请求，在请求地址的后面追加了查询字符串

“?username=xiaoming&password=123456”。

```
$.ajaxPrefilter('html', function(option) {  
    option.url += '?username=xiaoming&password=123456';  
});
```



## 6.3 Ajax的底层操作

### 3. Ajax全局配置

通过一个案例演示\$.ajaxPrefilter()的使用：

demo6-12.html 文档内容：分别为id为“btn1”和“btn2”的按钮绑定单击事件，并在事件方法中发送Ajax请求，分别请求不同类型的文件

```
$('#btn1').click(function() {  
    $.ajax({url: 'register.php', dataType: 'html'});  
});  
$('#btn2').click(function() {  
    $.ajax({url: 'target.json', dataType: 'json'});  
});
```

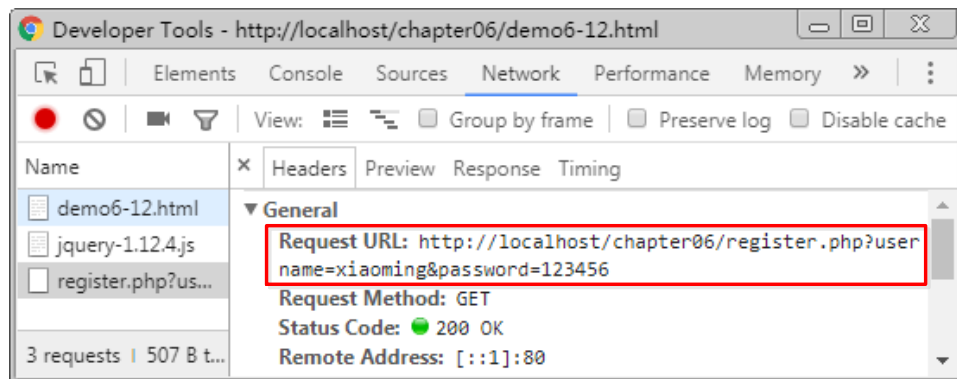


## 6.3 Ajax的底层操作

### 3. Ajax全局配置

单击“发送请求1”按钮，发送的请求如下图：

从图中可以看出，经过\$.ajaxPrefilter()处理后，在请求地址后面追加了相应的字符串。



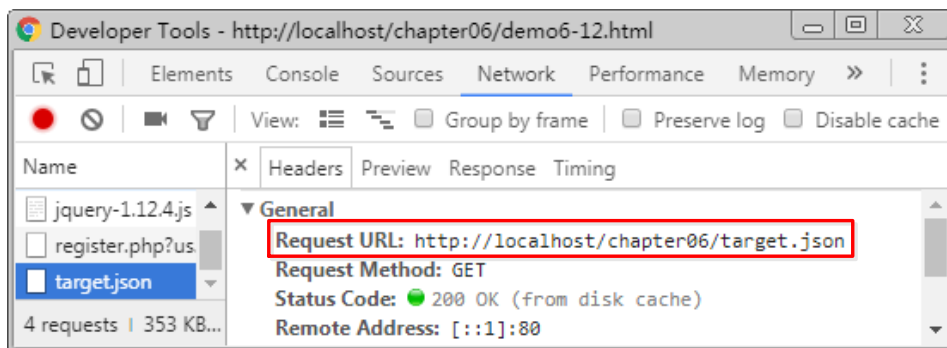


## 6.3 Ajax的底层操作

### 3. Ajax全局配置

单击“发送请求2”按钮，发送的请求如下图：

请求地址并没有发生改变。原因在于当前请求的数据类型是“json”，而第4行代码中\$.ajaxPrefilter()处理的请求数据类型是“html”。





## 6.4 序列化表单

### 1. 表单序列化为字符串

#### serialize():

概述：可将表单元素序列化为字符串

语法：formEle.serialize()

结果：formEle对象调用serialize()方法后会返回一段字符串，该字符串就是表单元素的数据序列化后的结果。格式形如“key1=value1& key2=value2& key3=value3”。



## 6.4 序列化表单

### 1. 表单序列化为字符串

接下来通过案例演示serialize()方法的使用：

demo6-13.html 文档内容：

```
<form id="formData">
  姓名 : <input type="text" name="username"> <br>
  年龄 : <input type="text" name="age"> <br>
  性别 : <input type="radio" name="gender" value="male"> 男
        <input type="radio" name="gender" value="female"> 女
</form>
<button id="btn">序列化</button>
```





## 6.4 序列化表单

### 1. 表单序列化为字符串

接下来通过案例演示serialize()方法的使用：

demo6-13.html 页面行为：为“序列化”按钮绑定单击事件，单击按钮后获取id为formData的form标签并调用serialize()方法序列化表单中的数据。

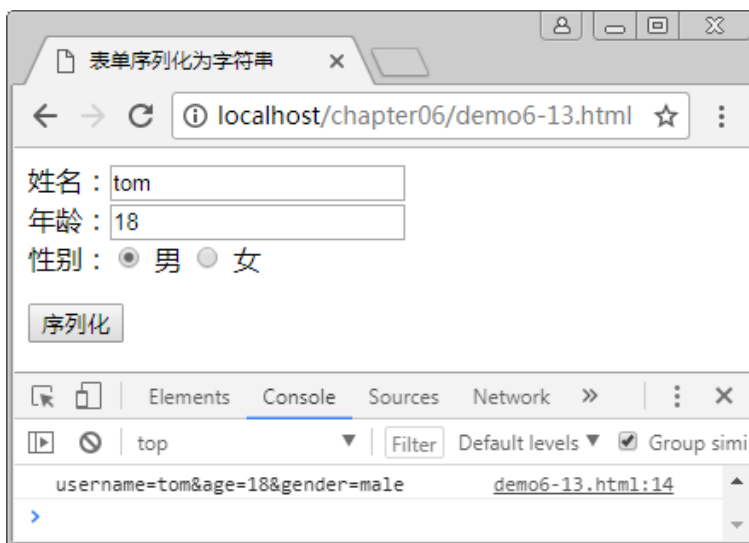
```
$('#btn').click(function() {  
    var result = $('#formData').serialize()  
    console.log(result);  
});
```



## 6.4 序列化表单

### 1. 表单序列化为字符串

填写表单，单击“序列化”按钮，结果如下图：可以观察到，`serialize()`方法根据表单元素的`name`属性获取用户输入的数据，多个数据间使用“&”连接，形成类似请求地址中查询字符串的形式。





## 6.4 序列化表单

### 1. 表单序列化为字符串

注意

如果表单元素未设置name属性，`serialize()`将忽略该元素，不作序列化；同时如果用户输入特殊字符如汉字、标签符号“< >”等，`serialize()`会自动进行字符编码。



## 6.4 序列化表单

### 1. 表单序列化为字符串

[serialize\(\)](#)将表单数据格式化成为类似查询字符串的形式。

在使用jQuery的Ajax方法发送数据时：

- ❑ 在前面加上“?”作为请求地址中的查询字符串使用
- ❑ 作为POST方式发送的数据使用



## 6.4 序列化表单

### 2. 表单序列化为对象

#### serializeArray():

**概述：** 可将表单数据序列化为对象

**语法：** formEle. serializeArray()

**结果：** 返回一个数组，数组中每一个元素都是一个对象，表示表单项。在对象中，“name”属性表示表单项的name属性值、“value”属性表示用户输入的值。



## 6.4 序列化表单

### 2. 表单序列化为对象

接下来通过案例演示serializeArray ()方法的使用：

demo6-13.html 页面行为：为“序列化”按钮绑定单击事件，单击按钮后获取id为formData的form标签并调用serializeArray ()方法序列化表单中的数据。

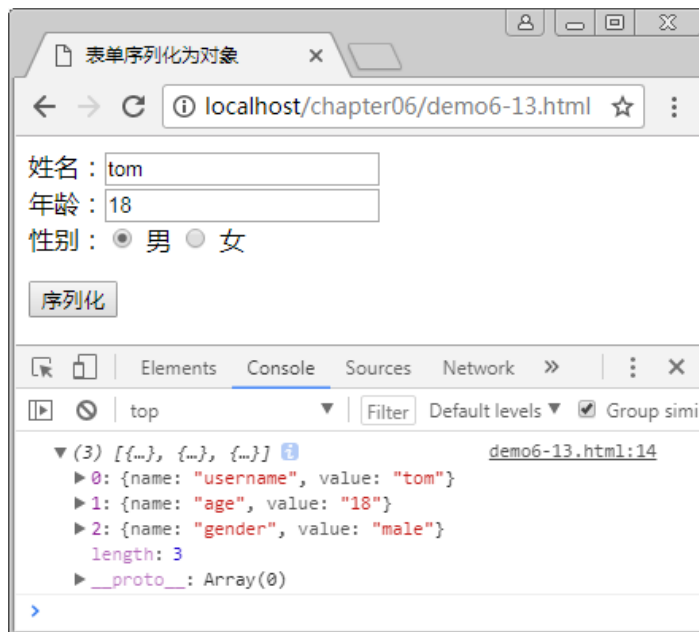
```
$('#btn').click(function() {  
    var result = $('#formData').serialize()  
    console.log(result);  
});
```



## 6.4 序列化表单

### 2. 表单序列化为对象

填写表单，单击“序列化”按钮，结果如下图：可以观察到，返回的数据中每一个元素都是一个对象。在对象中，“name”属性表示表单项的name属性值、“value”属性表示用户输入的值。





## 6.5 图书管理系统

### 1. 功能介绍

#### 图书查询

在本系统中，页面打开后会显示一个完整的图书列表，如下图所示：



图书编号	图书名称	图书作者	出版社	总页数	出版年份	价格	操作
138450072656	JavaScript DOM编程艺术 (第2版)	[英] Jeremy Keith	人民邮电出版社	300	2011	49	<a href="#">修改</a>   <a href="#">删除</a>
217477399306	Web前端黑客技术揭秘	钟晨峰	电子工业出版社	361	2013	59	<a href="#">修改</a>   <a href="#">删除</a>
422397735167	JavaScript高级程序设计 (第3版)	[美] Nicholas C. Zakas	人民邮电出版社	748	2012	99	<a href="#">修改</a>   <a href="#">删除</a>
511681708756	精通CSS (第2版)	[英] Andy Budd	人民邮电出版社	266	2010	49	<a href="#">修改</a>   <a href="#">删除</a>
616866816887	编写可维护的JavaScript	扎卡斯	人民邮电出版社	226	2013	55	<a href="#">修改</a>   <a href="#">删除</a>
771494383439	高性能网站建设指南	Steve Souders	电子工业出版社	146	2008	35	<a href="#">修改</a>   <a href="#">删除</a>
805089803458	高性能JavaScript	Nicholas C.Zakas	电子工业出版社	210	2010	49	<a href="#">修改</a>   <a href="#">删除</a>
908786897160	Head First HTML与CSS (第2版)	Elisabeth Robson	中国电力出版社	762	2013	98	<a href="#">修改</a>   <a href="#">删除</a>
958834914968	高性能网站建设进阶指南	Steve Souders	电子工业出版社	260	2010	49	<a href="#">修改</a>   <a href="#">删除</a>
968149724998	JavaScript权威指南 (第6版)	David Flanagan	机械工业出版社	1004	2012	139	<a href="#">修改</a>   <a href="#">删除</a>





## 6.5 图书管理系统

### 1. 功能介绍

在图书列表的上方提供了查询图书的功能。在文本框中输入关键字后，单击“查询”按钮，即可筛选出相关图书，如下图所示：



图书编号	图书名称	图书作者	出版社	总页数	出版年份	价格	操作
138450072656	JavaScript DOM编程艺术（第2版）	[英] Jeremy Keith	人民邮电出版社	300	2011	49	<a href="#">修改</a>   <a href="#">删除</a>
422397735167	JavaScript高级程序设计（第3版）	[美] Nicholas C. Zakas	人民邮电出版社	748	2012	99	<a href="#">修改</a>   <a href="#">删除</a>
616866816887	编写可维护的JavaScript	扎卡斯	人民邮电出版社	226	2013	55	<a href="#">修改</a>   <a href="#">删除</a>
805089803458	高性能JavaScript	Nicholas C.Zakas	电子工业出版社	210	2010	49	<a href="#">修改</a>   <a href="#">删除</a>
968149724998	JavaScript权威指南（第6版）	David Flanagan	机械工业出版社	1004	2012	139	<a href="#">修改</a>   <a href="#">删除</a>



## 6.5 图书管理系统

### 1. 功能介绍

#### 值得一提

与关键字“JavaScript”相关的图书被筛选出来了。值得一提的是，如果没有填入查询字符串，则展示所有图书数据。



## 6.5 图书管理系统

### 1. 功能介绍

#### 新增和修改图书：

单击“新增”按钮，可以弹出用于填写图书信息的表单，页面效果如下图所示。填写完成后，单击“提交”按钮提交即可保存。

图书信息编辑

图书编号：

图书名称：

图书作者：

出版社：

总页数：

出版年份：

价格：



## 6.5 图书管理系统

### 1. 功能介绍

#### 新增和修改图书：

单击“修改”链接，会弹出图书信息编辑表单，用于修改图书信息，页面效果如下图所示。填写完成后，单击“提交”按钮提交即可保存。

图书管理系统

localhost/chapter06/book/

图书信息编辑

图书编号: 138450072656

图书名称: JavaScript DOM编程艺术 (!

图书作者: [英] Jeremy Keith

出版社: 人民邮电出版社

总页数: 300

出版年份: 2011

价格: 49

提交 取消



## 6.5 图书管理系统

### 1. 功能介绍

删除图书：单击每一行最后一个单元格中的“删除”链接，可以删除当前行对应的图书信息。



## 6.5 图书管理系统

### 2. 系统设计

服务器:

- ❑ 服务器语言: PHP
- ❑ 服务器功能: WampServer



## 6.5 图书管理系统

### 2. 系统设计

项目结构：主目录为“chapter06/book”，目录结构及功能的介绍如下

类型	文件名	描述
文件	index.html	主页面
	process.php *	处理图书信息数据
目录	js	存放js文件
	css	存放css文件
	data	存放数据文件
文件	js/jquery-1.12.4.js *	jQuery文件
	js/operate.js	index.html中引入的js文件
	css/style.css *	页面样式文件
	data/book.json *	图书信息数据



## 6.5 图书管理系统

### 3. 用户界面

顶部“关键词输入域”，“查询”按钮、“新增”按钮：

```
<div class="top">  
  <input type="text" class="search-input">  
  <button class="search">查询</button>  
  <button class="add">新增</button>  
</div>
```





## 6.5 图书管理系统

### 3. 用户界面

图书展示区域:

```
<table>
  <tr>
    <th data-name="id">图书编号</th>
    <th data-name="title">图书名称</th>
    <th data-name="author">图书作者</th>
    <th data-name="publisher">出版社</th>
    <th data-name="pages">总页数</th>
    <th data-name="pubdate">出版年份</th>
    <th data-name="price">价格</th> <th>操作</th>
  </tr>
</table>
```



## 6.5 图书管理系统

### 3. 用户界面

“图书信息编辑”区域：

```
<div class="edit" style="display:none">
  <div class="edit-bg"></div>          <!-- 编辑图书信息时遮盖层 -->
  <div class="edit-center">            <!-- 编辑图书信息展示、填写区域-->
    <div class="edit-title">图书信息编辑</div>
    <p><span>图书编号：</span><input type="text" name="id"></p>
    <p><span>图书名称：</span><input type="text" name="title"></p>
    ...
  </div>
</div>
```



## 6.5 图书管理系统

### 4. 查询图书

获取所有图书信息：编写Booklist构造函数用于展示数据

```
function Booklist(obj) {  
    this.obj = obj;  
    var fields = []; // 保存图书列表中每一列的字段名  
    obj.find('tr:first th [data-name]').each(function() {  
        fields.push($(this).attr('data-name'));  
    });  
    this.fields = fields;  
}
```

通过data-name属性获取目标元素，并获取、保存data-name属性值

图书数据保存在实例的fields属性中



## 6.5 图书管理系统

### 4. 查询图书

编写Booklist构造函数原型方法append用于展示从服务器获取数据

```
Booklist.prototype = {  
  append: function(data, opt) { // 在图书列表中添加一行图书  
    var arr = []; // 保存每列字段名对应的数据  
    $.each(this.fields, function() {  
      arr.push(data[this]); // 通过遍历this.fields，确保每列顺序完全对应  
    });  
    var html = '<td>' + arr.join('</td><td>') + '</td><td>' +  
    opt.join(' | ') + '</td>';  
    this.obj.append('<tr data-id="" + data.id + "">' + html + '</tr>');  
  },  
};
```

保存每一条图书记录的唯一标识“id”，在开发“图书修改”和“图书删除”功能时，将会用到这个标识。



## 6.5 图书管理系统

### 4. 查询图书

通过\$.getJSON()方法请求process.php，以获取图书数据

```
var serverUrl = 'process.php'; // 请求地址
var list = $('.booklist'); // 页面中的图书列表
var opt = [ // 操作链接
    '<a href="#" class="booklist-edit">修改</a>',
    '<a href="#" class="booklist-del">删除</a>'
];
var booklist = new Booklist(list.find('table')); // 请求图书列表数据
$.getJSON(serverUrl, function(data) {
    for (var i in data) {
        booklist.append(data[i], opt);
    }
});
```

调用了booklist对象的append()方法，用于在图书列表中添加一行图书



## 6.5 图书管理系统

### 4. 查询图书

为Booklist构造函数添加原型方法search用于关键字查询功能

```
search: function( keyword ) {  
    var len = this.fields.length;  
    this.obj.find('tr:gt(0)').each(function() {  
        var show = true;  
        $(this).find('td:lt(' + len + ')').each(function() {  
            show = $(this).text().indexOf(keyword) !== -1;  
            if (show) {return false;}  
        });  
        $(this).toggle(show);  
    });  
}
```

查询关键字

检索条件



## 6.5 图书管理系统

### 4. 查询图书

通过关键字筛选图书：为“查询”按钮添加单击事件，调用booklist对象的search()方法实现筛选

```
$('.search').click(function() {  
    booklist.search($.trim($('.search-input').val()));  
});
```

去除两端空格



## 6.5 图书管理系统

### 5. 新增和编辑图书

显示图书编辑表单：新增和修改图书时，都会在页面中显示一个填写图书信息的表单。为了对表单进行控制，下面在js/operate.js中编写一个Editor构造函数。

```
function Editor(obj) {  
    this.obj = obj;  
}
```







## 6.5 图书管理系统

### 5. 新增和编辑图书

显示图书编辑表单

```
Editor.prototype = {  
  fill: function(data) {  
    this.empty(); // 先清空原有内容，防止data数据不完整出现未填充项  
    for (var k in data) {this.obj.find('[name=' + k + ']').val(data[k]);}  
  },  
  empty: function() {  
    this.obj.find('input[name]').val('');  
  }  
};
```



## 6.5 图书管理系统

### 5. 新增和编辑图书

显示图书编辑表单：为Booklist构造函数添加原型方法getData，获取被修改图书原有的信息

获取表格中指定索引值i的一行图书的原有信息

```
getData: function(i) {  
    var data = {};  
    var obj = this.obj.find('tr:eq(' + i + ')').find('td');  
    $.each(this.fields, function(i) {  
        data[this] = obj.eq(i).text();  
    });  
    return data;  
},
```



## 6.5 图书管理系统

### 5. 新增和编辑图书

显示图书编辑表单：修改图书,单击“修改”链接，显示被填入图书信息的图书编辑表单，修改图书信息。

```
var edit = $('.edit');  
var editor = new Editor(edit);  
list.on('click', '.booklist-edit', function() {  
    var i = $(this).parents('tr').index();  
    var data = booklist.getData(i);  
    var attr = {'data-action': 'update', 'data-i': i, 'data-id': data.id};  
    editor.fill(data); edit.attr(attr).fadeIn(200); return false;  
});
```

获取当前修改图书的索引值

// 修改链接

把图书信息填入到图书编辑表单



## 6.5 图书管理系统

### 5. 新增和编辑图书

显示图书编辑表单: 取消编辑,在单击“取消”按钮、遮罩层时,隐藏图书编辑表单,取消编辑图书信息

```
$('.edit-cancel').click(function() { // 取消按钮
    edit.hide();
});
$('.edit-bg').click(function() { // 遮罩层
    edit.hide();
});
```



## 6.5 图书管理系统

### 5. 新增和编辑图书

显示图书编辑表单: 新增图书, 单击“新增”按钮, 清空图书编辑表单数据, 以供用户填入数据, 增加图书数据

```
$('.add').click(function() { // 新增按钮  
    editor.empty();  
    edit.attr('data-action', 'add').fadeIn(200);  
});
```



## 6.5 图书管理系统

### 5. 新增和编辑图书

提交图书编辑表单: 在Editor.prototype对象中增加getData()方法, 用于获取表单数据。

```
getData: function() {  
    var data = {};  
    this.obj.find('input[name]').each(function() {  
        data[$(this).attr('name')] = $(this).val();  
    });  
    return data;  
}
```



## 6.5 图书管理系统

### 5. 新增和编辑图书

提交图书编辑表单: 在Booklist.prototype对象中增加update()方法, 用于替换图书列表中索引值为“i”的图书的信息

```
update: function(i, data) {  
    var obj = this.obj.find('tr:eq(' + i + ')').find('td');  
    $.each( this.fields , function( i ) {  
        obj.eq(i).text(data[this]);  
    });  
}
```

替换指定图书的信息



## 6.5 图书管理系统

### 5. 新增和编辑图书

提交图书编辑表单：提交表单到服务器。

当操作为修改时，提交需要更新的数据

```
$('.edit-save').click(function() {  
    var action = edit.attr('data-action'); var data = editor.getData();  
    if (action === 'update') {  
        var id = edit.attr('data-id');  
        $.post(serverUrl + '?action=update&id=' + id, data, function() {  
            booklist.update(edit.attr('data-i'), data); edit.hide();  
        }).fail(function(xhr) {alert('保存失败 , 错误信息 : ' + xhr.responseText);});  
    } else if(action === 'add') {.....}  
});
```





## 6.5 图书管理系统

### 6. 删除图书

删除图书信息: 在Booklist.prototype对象中新增remove()方法，删除索引值为i的图书

```
remove: function(i) {  
    this.obj.find('tr:eq(' + i + ')').remove();  
}
```



## 6.5 图书管理系统

### 6. 删除图书

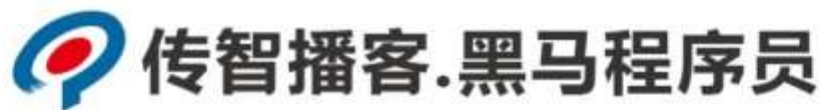
删除图书信息: 单击“删除”链接，删除对应的图书信息

```
list.on('click', '.booklist-del', function() {  
    var tr = $(this).parents('tr'), i = tr.index(), id = tr.attr('data-id');  
    $.post(serverUrl + '?action=del&id=' + id, function() {  
        booklist.remove(i);  
    });  
    return false;  
});
```



## 本章小结

本章介绍了jQuery中的Ajax操作，包括如何使用jQuery直接请求页面、JSON数据、JavaScript文件等，以及如何发出GET和POST请求。然后介绍了jQuery中最底层的Ajax方法\$.ajax()，讲解了其参数的使用方式。最后，本章对jQuery中Ajax的事件、配置以及表单的序列化都进行了详细讲解。



# Thank You!

[yx.boxuegu.com](http://yx.boxuegu.com)

