

# 1- 入门函数

jQuery 是一个 JavaScript 的开发框架

## 1.1 jQuery优势

- 提高开发效率，降低开发难度，降低开发成本。jQuery 框架是一个免费开源框架。
- jQuery 框架也是使用 JavaScript 开发出来，本质上 jQuery 框架就是JS代码。

### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>初识jQuery</title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    div{
      width: 100px;
      height: 100px;
      border: 1px solid #000;
    }
  </style>
  <script src="js/jquery-1.12.4.js"></script>
</head>
<body>
<div></div>
<div class="box1"></div>
<div id="box2"></div>
<script type="text/javascript">
  // 原生js
  window.onload = function (event){
    console.log("---原生JS--");
    // 1.利用原生JS的查找DOM元素
    let div1 = document.getElementsByTagName("div")[0];
    let div2 = document.getElementsByClassName("box1")[0];
    let div3 = document.getElementById("box2");
    // 输出结果
    console.log(div1);
    console.log(div2);
    console.log(div3);
    // 2.利用原生的js修改背景颜色
    /*div1.style.backgroundColor = "red";
    div2.style.backgroundColor = "blue";
    div3.style.backgroundColor = "yellow";*/
  }

  // jq方式
  $(function (){
    console.log("---jQuery方式---");
```

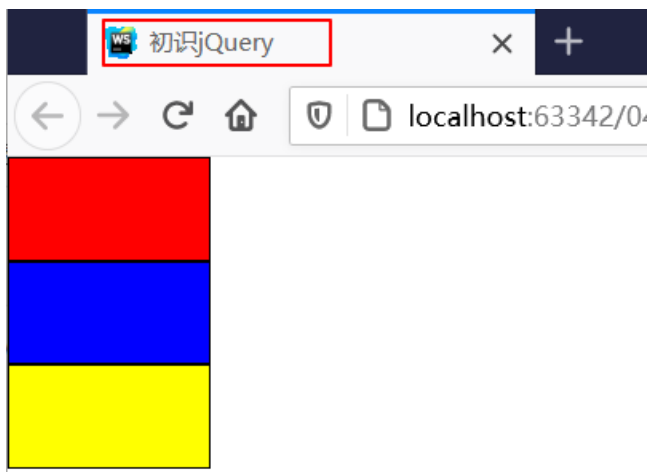
```

let $div1 = $("div");
let $div2 = $(".box1");
let $div3 = $("#box2");
// 输出结果
/*console.log($div1);
console.log($div2);
console.log($div3);*/

$div1.css({
  background: "red",
  width: "200px",
  height: "200px"
});
$div2.css({
  background: "blue"
});
$div3.css({
  background: "yellow"
});
});
</script>
</body>
</html>

```

## 2、执行代码



## 1.2 jQuery特点

### 1、基本介绍

- 轻量级：框架本身很小，占用资源少。
- 兼容性：可以运行在所有主流的浏览器上。
- 插件：本身还支持大量的插件。
- 宗旨：write less do more。

### 2、代码示例

- 准备 jquery 框架，复制到项目中 js 目录下
- 在 HTML 中使用 script 标签导入 jquery.js 文件就可以使用了

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">

```

```

<title>jQuery的导入</title>
<script src="js/jquery-3.3.1.js"></script>
</head>
<body>
<script type="text/javascript">
    /*
    参数是一个匿名函数，这个函数功能相当于：
    window.onload = function() 页面加载完毕以后自动执行的代码
    */
    $(function () {
        //我们的代码写在这里
        alert("Hello jQuery");
    });
</script>
</body>
</html>

```

## 1.3 转换

在jQuery框架中使用过程中，有JS对象和JQ对象之分。

### 1、JS对象与jq对象的区别

- JS对象：通过以前的JS代码得到的对象，如：`document.getElementById()`，只能使用JS中方法和属性。
- jQuery对象：通过JQ选择器得到的对象，可以使用JQ中提供的方法，JQ对象本质上是一个JS数组。

### 2、转换原因

JS对象只能使用以前JS的方法和属性，不能使用JQ对象的方法。JQ对象中有很多功能强大方法，如果JS对象要调用JQ对象，就必须将JS对象转换成JQ对象。反之亦然。

### 3、转换语法

| 操作               | 方法  |
|------------------|---|
| 将JS对象-->jQuery对象 | <code>\$(JS对象)</code>                           |
| 将jQuery对象-->JS对象 | <code>JQ对象[0]</code> 或 <code>JQ对象.get(0)</code> |

### 4、代码示例

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>JS对象与JQ对象转换</title>
    <script src="js/jquery-3.1.1.js"></script>
</head>
<body>
<h2>JS对象与JQ对象的转换</h2>
<input type="text" id="name" value="我太难了">
<input type="button" id="b1" value="JS得到值">
<input type="button" id="b2" value="JQ得到值">
<script type="text/javascript">
    //JS对象的方法

```

```

document.getElementById("b1").onclick = function () {
    //文本框对象，是JS对象
    let name = document.getElementById("name");
    //JS->JQ对象
    let jq = $(name);
    //得到文本框的值
    //var value = name.value;
    //调用jq对象的方法
    let value = jq.val();
    //显示出来
    console.log("js-jquery:" + value);
}

//JQ对象的方法，方法参数就是事件处理函数
$("#b2").click(function () {
    //1.文本框对象，JQ对象
    let name = $("#name");

    //jq-> JS对象
    //var js = name[0];

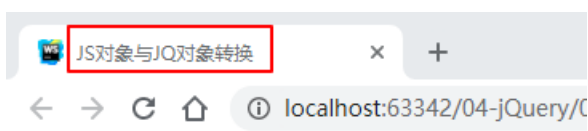
    let js = name.get(0);
    let val = js.value;

    //2.得到文本框中值，调用jq对象的方法，得到值
    //var val = company.val();

    //3.显示出来
    console.log("jquery-js:" + val);
});
</script>
</body>
</html>

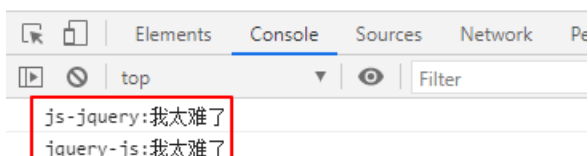
```

## 5、执行结果



## JS对象与JQ对象的转换

我太难了 JS得到值 JQ得到值



## 6、总结

| 操作                | 方法  |
|-------------------|---|
| 将JS对象-->jQuery对象  | <code>\$(JS对象)</code>                               |
| 将jQuery对象--> JS对象 | <code>JQ 对象 [0]</code> 或 <code>JQ 对象 .get(0)</code> |

## 1.4 jQuery和JS入口函数

### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jQuery和js的区别</title>
  <script src="js/jquery-3.1.1.js"></script>
  <script type="text/javascript">
    window.onload = function (event){
      console.log("--js原生实现--")
      // 1.通过原生的JS入口函数可以拿到DOM元素
      let image = document.getElementsByTagName("img")[0];
      console.log(image);

      // 2.通过原生JS入口函数可以拿到DOM元素
      let width = window.getComputedStyle(image).width;
      console.log(width);
    }

    $(document).ready(function (){
      console.log("---jquery实现---")
      // 1.通过jQuery入口函数可以拿到DOM元素
      let $img = $("img");
      console.log($img);
      // 2.通过jQuery入口函数不可以拿到DOM元素的宽高
      let $width = $img.width();
      console.log($width);
    });

    /*
    1. 原生的JS如果编写了多个入口函数,后面编写的会覆盖前面编写的
    2. jQuery中编写多个入口函数,后面的不会覆盖前面的
    */
    window.onload = function (event){
      console.log("JS-hello guardwhy1");
    }
    window.onload = function (event){
      console.log("JS-hello guardwhy2");
    }

    $(document).ready(function (){
      console.log("jq-hello guardwhy01");
    });
    $(document).ready(function (){
      console.log("JQ-hello guardwhy02");
    });
  </script>
```

```

</head>
<body>


</body>
</html>

```

## 2、执行结果



## 2-核心函数

### 2.1 核心函数 \$()

\$() 代表调用 jQuery 的核心函数。

#### 1、代码示例

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jQuery核心函数</title>
  <script src="js/jquery-3.1.1.js"></script>
</head>
<body>
<div class="box1"></div>
<div id="box2"></div>
<span>桃李春风一杯酒，江湖夜雨十年灯</span>
<script type="text/javascript">
  // $();jQuery原理();就代表调用jQuery的核心函数
  // 1.接收一个函数
  $(function () {
    // 输出结果
    console.log("hello jQuery");
  });

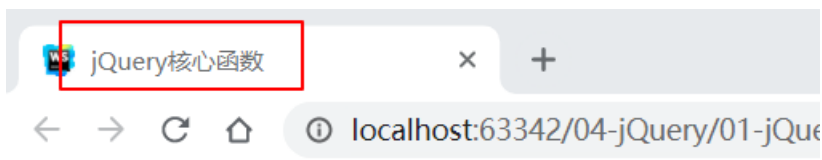
```

```

// 2.接收一个字符串
// 返回一个jQuery对象,对象中保存了找到的DOM元素
let $box1 = $(".box1");
let $box2 = $("#box2");
// 输出结果
console.log($box1);
console.log($box2);
// 接收一个字符串代码片段
let $p = $("<p>窗外日光弹指过, 席间花影坐前移。</p>");
console.log($p);
$box1.append($p);
// 3.接收一个DOM元素,会被包装成一个jQuery对象返回
let span = document.getElementsByTagName("span")[0];
console.log(span);
let $span = $(span);
console.log($span);
});
</script>
</body>
</html>

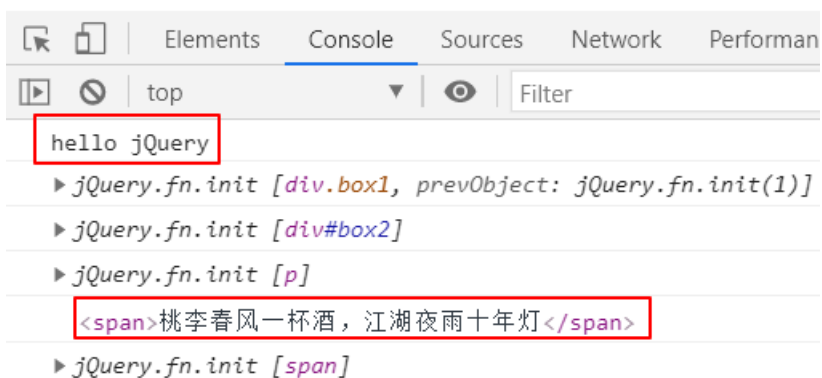
```

## 2、执行结果



窗外日光弹指过, 席间花影坐前移。

桃李春风一杯酒, 江湖夜雨十年灯



## 2.2 jquery对象

### 1、代码示例

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jQuery对象</title>
  <script src="js/jquery-3.1.1.js"></script>

```

```

</head>
<body>
<div>愿我如星君如月，夜夜流光相皎洁。</div>
<div>洞庭西望楚江分，水尽南天不见云。</div>
<div>一叶舟轻，双桨鸿惊</div>
<script type="text/javascript">
  $(function (){
    /*
    jQuery对象是一个伪数组。
    伪数组:有0到length-1属性,并且有length属性
    */
    let $div = $("div");
    console.log($div);

    // 创建arr数组
    let array = [1,5,9];
    // 输出结果
    console.log(array);
  });
</script>
</body>
</html>

```

## 2、执行结果



## 3- 静态方法

### 3.1 静态方法和实例方法

#### 1、代码示例

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>静态方法和实例方法</title>
  <script src="js/jquery-3.1.1.js"></script>
</head>
<body>
<script type="text/javascript">

```



```

// 1. 定义一个类
function AClass(){

}

// 2. 直接添加给类的就是静态方法
AClass.staticMethod = function (){
    console.log("staticMethod调用....");
}

// 3. 静态方法通过类名调用
AClass.staticMethod();

// 4. 给这个类添加一个实例方法
AClass.prototype.instanceMethod = function (){
    console.log("instanceMethod调用....")
}

// 5. 创建一个对象
let obj1 = new AClass();
// 6. 对象调用实例方法
obj1.instanceMethod();
</script>
</body>
</html>

```

## 3.2 静态方法each

### 1、代码示例

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>静态方法each方法</title>
    <script src="js/jquery-3.1.1.js"></script>
</head>
<body>
<script type="text/javascript">
    // 定义数组
    let Array = [1, 3, 5, 7, 9];
    // 定义伪数组
    var obj = {0:1, 1:3, 2:5, 3:7, 4:9, length:5};
    /*
        第一个参数：遍历到的元素
        第二个参数：当前遍历到的索引
        注意点：原生的forEach方法只能遍历数组，不能遍历伪数组
    */
    console.log("---原生JS数组遍历---");
    Array.forEach(function (value, index){
        console.log(index, value);
    });

    // 2. jQuery静态方法遍历数组
    /*
        第一个参数：当前遍历到的索引
        第二个参数：遍历到的元素
        注意点：
        jQuery的each方法是可以遍历伪数组的
    */

```

```

console.log("---原生jQuery数组遍历---");
$.each(Array, function (index, value){
    console.log(index, value);
});
console.log("---原生jQuery伪数组遍历---");
$.each(obj, function (index, value){
    console.log(index, value);
});
</script>
</body>
</html>

```

## 2、执行结果

```

---原生JS数组遍历---
0 1
1 3
2 5
3 7
4 9
---原生jQuery数组遍历---
0 1
1 3
2 5
3 7
4 9
---原生jQuery伪数组遍历---
0 1
1 3
2 5
3 7
4 9

```

## 3.3 静态方法map

### 1、代码示例

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>静态方法map方法</title>
  <script src="js/jquery-3.1.1.js"></script>
</head>
<body>
<script type="text/javascript">
  // 定义数组
  let Array = [1, 3, 5, 7, 9];
  // 定义伪数组
  let obj = {0:1, 1:3, 2:5, 3:7, 4:9, length:5};
  // 1.利用原生JS的map方法遍历
  /*
  第一个参数：当前遍历到的元素
  第二个参数：当前遍历到的索引
  第三个参数：当前被遍历的数组
  */

```

注意点：

和原生的forEach一样，不能遍历的伪数组

\*/

```
console.log("---原生JS数组遍历---");
Array.map(function (value, index, array){
    console.log(index, value, array);
});
```

// 2.jQuery静态方法遍历数组

/\*

第一个参数：要遍历的数组

第二个参数：每遍历一个元素之后执行的回调函数

回调函数的参数：

第一个参数：遍历到的元素

第二个参数：遍历到的索引

注意点：

和jQuery中的each静态方法一样，map静态方法也可以遍历伪数组

\*/

```
console.log("---原生jQuery数组遍历---");
$.map(Array, function (value, index){
    console.log(index, value);
});
```

/\*

jQuery中的each静态方法和map静态方法的区别：

each静态方法默认的返回值就是，遍历谁就返回谁

map静态方法默认的返回值是一个空数组

each静态方法不支持在回调函数中对遍历的数组进行处理

map静态方法可以在回调函数中通过return对遍历的数组进行处理，然后生成一个新的数组返回

\*/

```
console.log("---原生jQuery伪数组遍历---");
```

```
let res1 = $.map(obj, function (value, index){
    console.log(index, value);
    return value + index;
});
console.log(++map方法++);
```

```
let res2 = $.each(obj, function (value, index){
    console.log(index, value);
    return value + index;
});
console.log(++each方法++);
// 输出结果
console.log("res1:" + res1);
console.log("res2:" + res2);
```

</script>

</body>

</html>

## 2、执行结果

|                                      |                       |
|--------------------------------------|-----------------------|
| 静态方法map方法                            |                       |
| localhost:63342/04-jQuery/01-jQuery基 |                       |
| 查看器 控制台 调试器 网络 >> 过滤输出               |                       |
| 错误 警告 日志 信息 调试 CSS XHR 请求            |                       |
| ---原生JS数组遍历---                       | 10-静态方法map.html:22:11 |
| 0 1 ▶ Array(5) [ 1, 3, 5, 7, 9 ]     | 10-静态方法map.html:24:13 |
| 1 3 ▶ Array(5) [ 1, 3, 5, 7, 9 ]     | 10-静态方法map.html:24:13 |
| 2 5 ▶ Array(5) [ 1, 3, 5, 7, 9 ]     | 10-静态方法map.html:24:13 |
| 3 7 ▶ Array(5) [ 1, 3, 5, 7, 9 ]     | 10-静态方法map.html:24:13 |
| 4 9 ▶ Array(5) [ 1, 3, 5, 7, 9 ]     | 10-静态方法map.html:24:13 |
| ---原生jQuery数组遍历---                   | 10-静态方法map.html:37:11 |
| 0 1                                  | 10-静态方法map.html:39:13 |
| 1 3                                  | 10-静态方法map.html:39:13 |
| 2 5                                  | 10-静态方法map.html:39:13 |
| 3 7                                  | 10-静态方法map.html:39:13 |
| 4 9                                  | 10-静态方法map.html:39:13 |
| ---原生jQuery伪数组遍历---                  | 10-静态方法map.html:50:11 |
| 0 1                                  | 10-静态方法map.html:53:13 |
| 1 3                                  | 10-静态方法map.html:53:13 |
| 2 5                                  | 10-静态方法map.html:53:13 |
| 3 7                                  | 10-静态方法map.html:53:13 |
| 4 9                                  | 10-静态方法map.html:53:13 |
| ++map方法++                            | 10-静态方法map.html:56:11 |
| 1 0                                  | 10-静态方法map.html:59:13 |
| 3 1                                  | 10-静态方法map.html:59:13 |
| 5 2                                  | 10-静态方法map.html:59:13 |
| 7 3                                  | 10-静态方法map.html:59:13 |
| 9 4                                  | 10-静态方法map.html:59:13 |
| ++each方法++                           | 10-静态方法map.html:62:11 |
| res1:1,4,7,10,13                     | 10-静态方法map.html:64:11 |
| res2:[object Object]                 | 10-静态方法map.html:65:11 |

## 3.4 其他方法

### 1、代码示例

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jquery其他静态方法</title>
  <script src="js/jquery-3.1.1.js"></script>
</head>
<body>
<script type="text/javascript">
  /*
  $.trim();
  作用：去除字符串两端的空格
  参数：需要去除空格的字符串
  返回值：去除空格之后的字符串
  */
  let str1 = " guardwhy ";
  let res1 = $.trim(str1);
  // 输出结果

```

```

console.log("----" + str1 + "----");
console.log("----" + res1 + "----");

// 定义数组
let Array = [1, 3, 5, 7, 9];
// 定义伪数组
let ArrayLike = {0:1, 1:3, 2:5, 3:7, 4:9, length:5};
// 对象
let obj = {"name":"guardwhy", age:"26"};
// 函数
let func = function (){};
// 创建window对象
let w = window;

/*
$.iswindow();
作用：判断传入的对象是否是window对象
返回值：true/false
*/
let res2 = $.iswindow(w);
// 输出结果
console.log("res2:" + res2);

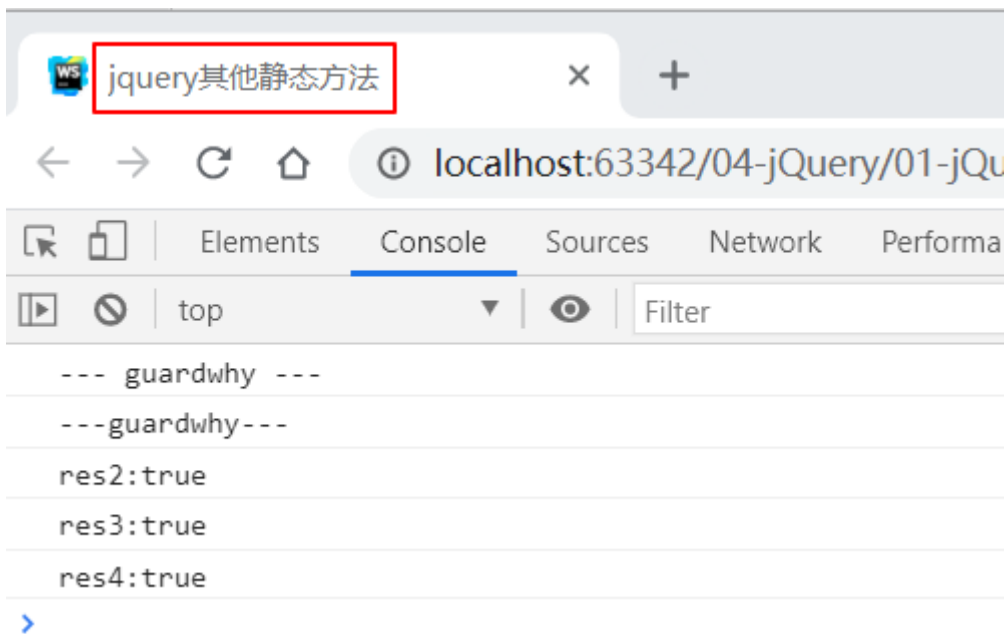
/*
$.isArray();
作用：判断传入的对象是否是真数组
返回值：true/false
*/
let res3 = $.isArray(Array);
console.log("res3:" + res3);

/*
$.isFunction();
作用：判断传入的对象是否是一个函数
返回值：true/false
注意点：jQuery框架本质上是一个函数
*/
let res4 = $.isFunction(jQuery);
console.log("res4:" + res4);

</script>
</body>
</html>

```

## 2、执行结果



## 4- jQuery选择器

### 4.1 内容选择器

#### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jQuery内容选择器</title>
  <style type="text/css">
    div{
      width: 150px;
      height: 100px;
      background: red;
      margin-top: 5px;
    }
  </style>
  <script src="js/jquery-1.12.4.js"></script>
</head>
<body>
  <div></div>
  <div>我有一瓢酒,可以慰风尘。</div>
  <div>君埋泉下泥销骨,我寄人间雪满头。</div>
  <div><span></span></div>
  <div><p></p></div>
  <script type="text/javascript">
    $(function (){
      // 1.empty 作用:找到既没有文本内容也没有子元素的指定元素
      let $div1 = $("div:empty");
      console.log($div1);

      // 2.:parent 作用: 找到有文本内容或有子元素的指定元素
      let $div2 = $("div:parent");
      console.log($div2);

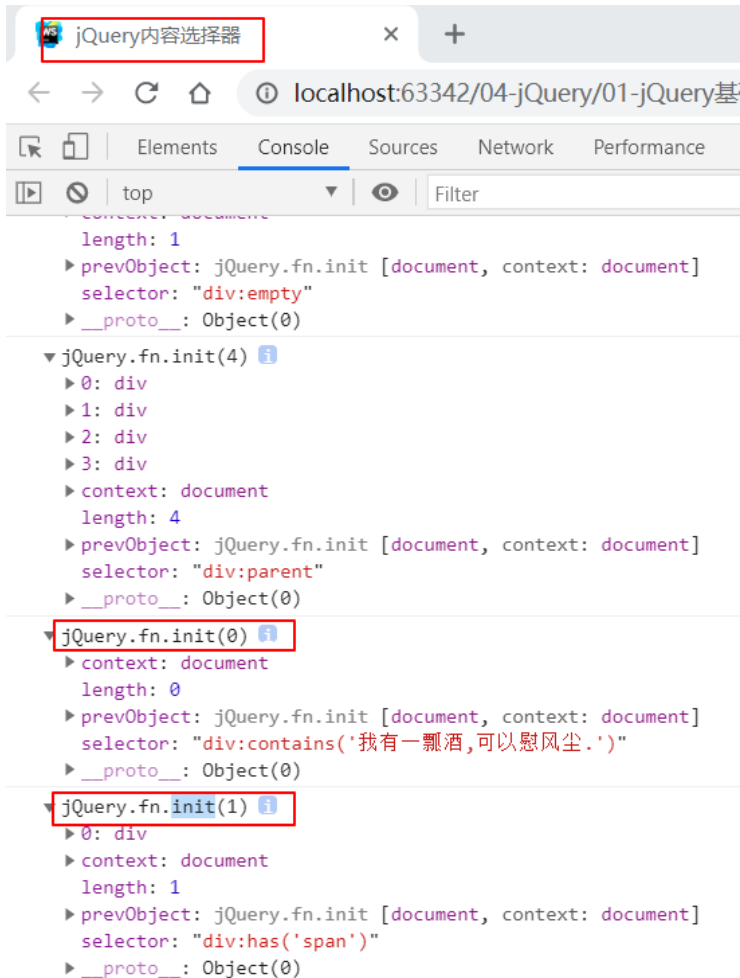
      // 3.:contains(text) 作用:找到包含指定文本内容的指定元素
      let $div3 = $("div:contains('我有一瓢酒,可以慰风尘.')");
```

```

console.log($div3);
// 4.:has(selector) 作用：找到包含指定子元素的指定元素
let $div4 = $("div:has('span')");
console.log($div4);
});
</script>
</body>
</html>

```

## 2、执行结果



# 5- jQuery属性操作

## 5.1 属性节点

### 1、代码示例

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>属性节点</title>
  <script src="js/jquery-1.12.4.js"></script>
</head>
<body>
  <span name="imooc"></span>
  <script type="text/javascript">
    $(function (){
      /*

```

### 1. 属性节点:

在编写HTML代码时,在HTML标签中添加的属性就是属性节点。

在`attributes`属性中保存的所有内容都是属性节点。

### 2. 操作属性节点

DOM元素.`setAttribute("属性名称", "值");`

DOM元素.`getAttribute("属性名称");`

### 3. 属性和属性节点区别

任何对象都有属性,但是只有DOM对象才有属性节点。

```
*/  
let span = document.getElementsByTagName("span")[0];  
// 设置属性节点  
span.setAttribute("name", "guardwhy");  
// 输出结果  
console.log(span.getAttribute("name"));  
});  
</script>  
</body>  
</html>
```

## 2、执行结果

```
▶ attributeStyleMap: StylePropertyMap {size: 0}  
▼ attributes: NamedNodeMap  
  ▼ 0: name  
    baseUrl: "http://localhost:63342/04-jQuery/01-jQuery/  
    ▶ childNodes: NodeList []  
    firstChild: null  
    isConnected: false  
    lastChild: null  
    localName: "name"  
    name: "name"  
    namespaceURI: null  
    nextSibling: null  
    nodeName: "name"  
    nodeType: 2  
    nodeValue: "guardwhy"  
    ▶ ownerDocument: document  
    ▶ ownerElement: span  
    parentElement: null  
    parentNode: null  
    prefix: null  
    previousSibling: null  
    specified: true  
    textContent: "guardwhy"  
    value: "guardwhy"  
    ▶ __proto__: Attr  
    length: 1  
    ▶ name: name
```



## 5.2 属性方法

### 1、代码示例

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <title>属性</title>  
  <script src="js/jquery-1.12.4.js"></script>  
</head>  
<body>  
  <span name="imooc"></span>  
  <script type="text/javascript">  
    $(function () {
```



```

/*
1.属性：对象身上保存的变量就是属性
2.操作属性
    对象.属性名称 = 值；
    对象.属性名称；
    对象["属性名称"] = 值；
    对象["属性名称"]；
*/
// 定义函数
function Person(){}
// 创建对象
let p = new Person();
// 属性操作
p.name = "guardwhy";
console.log(p.name);

});
</script>
</body>
</html>

```

## 2、attr 和 removeAttr 方法

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>属性方法</title>
    <script src="js/jquery-1.12.4.js"></script>
</head>
<body>
<span class="obj1" name = "Curry"></span>
<span class="obj2" name = "James"></span>

<script type="text/javascript">
    $(function (){
        /*
        1.attr(name|prop|key,val|fn)
            作用：获取或者设置属性节点的值,可以传递一个参数，也可以传递两个参数。
            传递一个参数，获取属性节点的值。传递两个参数，设置属性节点的值。
            注意点：
            如果是获取:无论找到多少个元素，都只会返回第一个元素指定的属性节点的值
            如果是设置:找到多少个元素就会设置多少个元素，如果设置的属性节点不存在，那么系统会自动新增。
        */
        // 打印节点
        /*console.log($(".span").attr("class"));
        // 设置
        $(".span").attr("class","box");
        // 设置属性节点不存在
        $(".span").attr("guardwhy", "26");*/

        /*
        2.removeAttr(name)
            删除属性节点
            注意点:会删除所有找到元素指定的属性节点
        */
    });

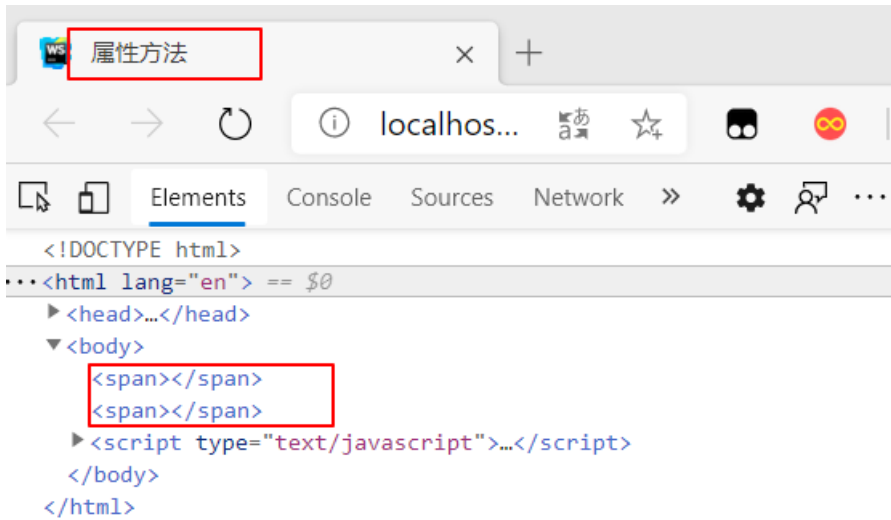
```

```

    $("span").removeAttr("class name");
  })
</script>
</body>
</html>

```

### 3、执行结果



### 4、prop 与 attr 方法

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>属性方法2</title>
  <script src="js/jquery-1.12.4.js"></script>
</head>
<body>
  <span class="obj1" name = "Curry"></span>
  <span class="obj2" name = "James"></span>
  <input type="checkbox" checked>

  <script type="text/javascript">
    $(function (){
      /*
      1.prop方法
      特点和attr方法一致
      2.removeProp方法
      特点和removeAttr方法一致
      */
      // 1.设置操作
      /*$("span").eq(0).prop("NBA", "kobe");
      $("span").eq(1).prop("NBA", "Rondo");*/
      // 打印属性
      // console.log($("span").prop("NBA")); // kobe

      // 2.移除操作
      /*$("span").removeProp("NBA");*/

      /*
      注意点：
      prop方法不仅能够操作属性，还能操作属性节点。

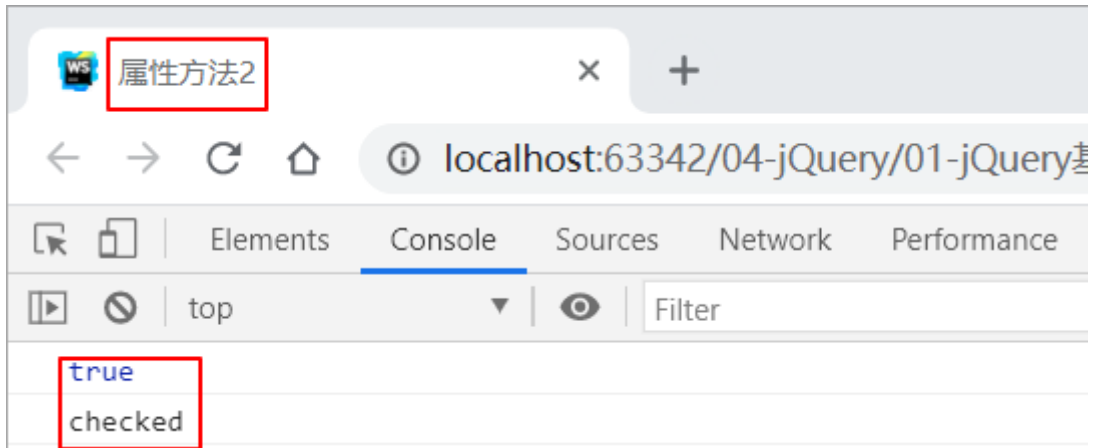
```

```

checked,selected或者disabled使用prop(),其他的使用 attr()。
*/
console.log($("#input").prop("checked"));    // true
console.log($("#input").attr("checked"));    //checked
});
</script>
</body>
</html>

```

## 5、执行结果



## 5.4 hasClass属性

### 1、代码示例

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>hasClass属性</title>
  <script src="js/jquery-1.12.4.js"></script>
  <script type="text/javascript">
    /*
    hasClass(): 判断元素中是否包含指定类
    */
    $(function () {
      // 1.传递参数, 只要调用者其中一个包含指定类就返回true,否则返回false
      console.log($("#div").hasClass("kobe"));    // true
      console.log($("#div").hasClass("james"));  // false
      // 2.没有传递参数,返回false
      console.log($("#div").hasClass()); // false
    });
  </script>
</head>
<body>
  <div class="guardwhy kobe Curry"></div>
  <div class="abc cc"></div>
</body>
</html>

```

## 6- jQuery CSS相关

## 6.1 jQuery操作类

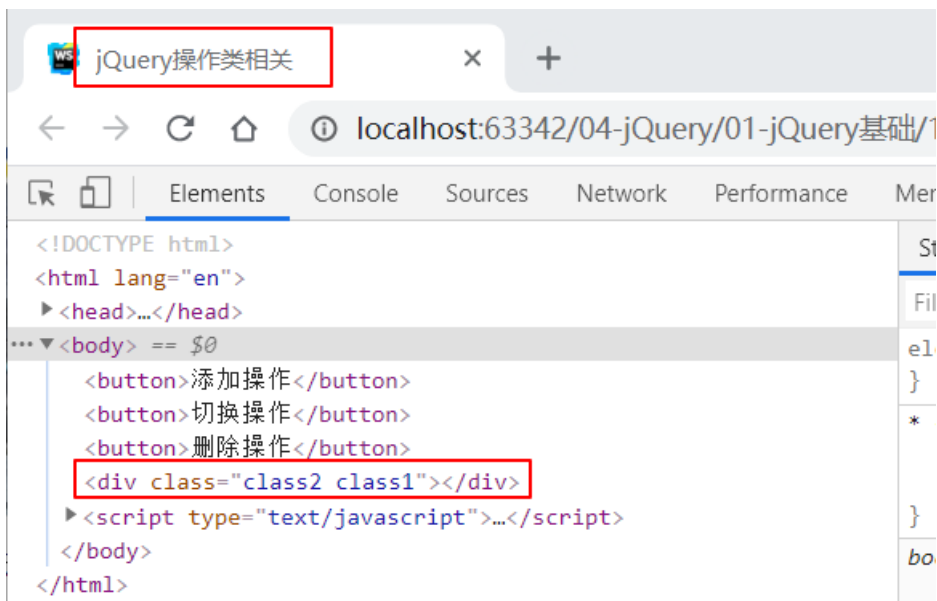
### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jQuery操作类相关</title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    .class1{
      width: 100px;
      height: 100px;
      background: red;
    }
    .class2{
      border: 10px solid #000;
    }
  </style>
  <script src="js/jquery-1.12.4.js"></script>
</head>
<body>
  <button>添加操作</button>
  <button>删除操作</button>
  <button>切换操作</button>

  <div></div>
  <script type="text/javascript">
    $(function (){
      /*
      1.addClass(class|fn)
      作用：添加一个类要添加多个，多个类名之间用空格隔开即可。
      2.removeClass([class|fn])
      作用：删除一个类,想删除多个，多个类名之间用空格隔开即可。
      3.toggleClass(class|fn[,sw])
      作用：切换类有就删除，没有就添加。
      */
      let btns = document.getElementsByTagName("button");
      // 添加类
      btns[0].onclick = function (){
        $("div").addClass("class1 class2");
      }
      // 删除类
      btns[1].onclick = function (){
        $("div").removeClass("class2 class1");
      }
      // 切换操作
      btns[2].onclick = function (){
        $("div").toggleClass("class2 class1")
      }
    });
  </script>
</body>
</html>
```

```
</script>
</body>
</html>
```

## 2、执行结果



## 6.2 jQuery文本值

### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jQuery文本值相关</title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    div{
      width: 100px;
      height: 100px;
      border: 1px solid #000;
    }
  </style>
  <script src="js/jquery-1.12.4.js"></script>
</head>
<body>
  <button>设置html</button>
  <button>获取html</button>
  <button>设置text</button>
  <button>获取text</button>
  <button>设置value</button>
  <button>获取value</button>
  <div></div>
  <input type="text">
  <script type="text/javascript">
    $(function (){
      /*
```

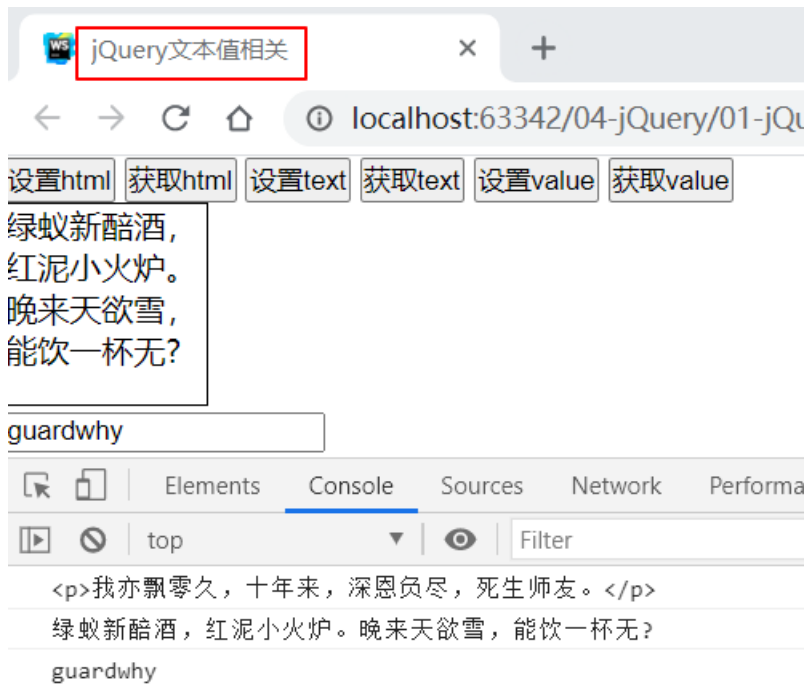
```

1.html([val|fn])
和原生JS中的innerHTML一模一样
2.text([val|fn])
和原生JS中的innerText一模一样
3.val([val|fn|arr])
*/
let btns = document.getElementsByTagName("button");
btns[0].onclick = function (){
    $("div").html("<p>我亦飘零久，十年来，深恩负尽，死生师友。</p>");
}
btns[1].onclick = function (){
    console.log($("div").html());
}
btns[2].onclick = function (){
    $("div").text("绿蚁新醅酒，红泥小火炉。晚来天欲雪，能饮一杯无?");
}
btns[3].onclick = function (){
    console.log($("div").text());
}

btns[4].onclick = function (){
    $("input").val("请输入内容");
}
btns[5].onclick = function (){
    console.log($("input").val());
}
});
</script>
</body>
</html>

```

## 2、执行结果



## 6.3 jQuery操作CSS样式

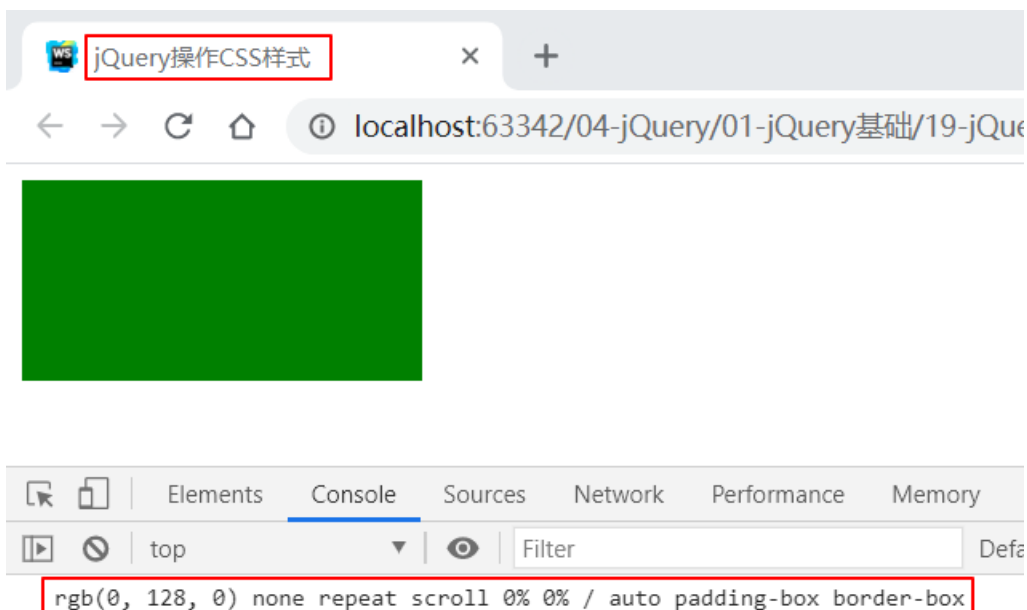
### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jQuery操作CSS样式</title>
  <script src="js/jquery-1.12.4.js"></script>
</head>
<body>
<div></div>
<script type="text/javascript">
  $(function () {
    // 1. 每个设置
    $("div").css("width", "100px");
    $("div").css("height", "100px");
    $("div").css("background", "red");

    // 2. 链式设置
    $("div").css("width", "100px").css("height", "200px").css("background",
"green");
    // 3. 批量设置
    $("div").css({
      width: "200px",
      height: "100px",
      background: "green"
    });

    // 4. 获取CSS样式值
    console.log($("div").css("background"));
  });
</script>
</body>
</html>
```

### 2、执行结果



## 6.4 jQuery位置和尺寸操作

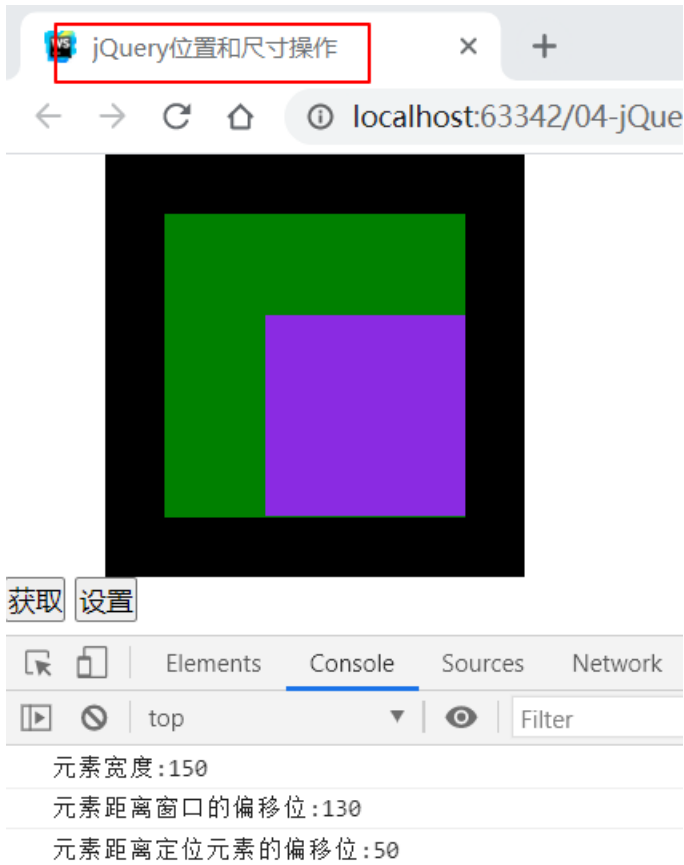
### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jQuery位置和尺寸操作</title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    .father{
      width: 150px;
      height: 150px;
      background: green;
      border: 30px solid #000;
      margin-left: 50px;
      position: relative;
    }
    .son{
      width: 100px;
      height: 100px;
      background: blueviolet;
      position: absolute;
      left: 50px;
      top: 50px;
    }
  </style>
  <script src="js/jquery-1.12.4.js"></script>
</head>
<body>
  <div class="father">
    <div class="son"></div>
  </div>
  <button>获取</button>
  <button>设置</button>
  <script type="text/javascript">
    $(function (){
      // 1.拿到button
      let btns = document.getElementsByTagName("button");
      // 2.监听获取
      btns[0].onclick = function (){
        // 2.1 获取元素的宽度
        console.log("元素宽度:" + $(".father").width());
        // 2.2 offset: 获取元素距离窗口的偏移位
        console.log("元素距离窗口的偏移位:" + $(".son").offset().left);
        // 2.3 position: 获取元素距离定位元素的偏移位
        console.log("元素距离定位元素的偏移位:" + $(".son").position().left);
      }
      // 3.监听设置
      btns[1].onclick = function (){
        // 3.1 设置元素的宽度
        $(".father").width("500px");
        // 3.2 元素距离窗口的偏移位
```



```
$.son").offset({
    left:10
});
// 3.3 position方法只能获取不能设置,只能通过CSS设置
$(".son").css({
    left: "10px"
});
}
});
</script>
</body>
</html>
```

## 2、执行结果



## 6.5 jQuery的scrollTop

## 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jQuery的scrollTop</title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    .scrollTop{
      width: 200px;
      height: 200px;
```

```

border: 1px solid #000;
overflow: auto;
}
</style>
<script src="js/jquery-1.12.4.js"></script>
</head>
<body>
<div class="scroll">
    1.我亦飘零久，十年来，深恩负尽，死生师友。——顾贞观《金缕曲二首》2.桃李春风一杯酒，江湖夜雨十年灯。——黄庭坚《寄黄几复》
    3.雨中黄叶树，灯下白头人。——司空曙《喜外弟卢纶见宿》4.人间有味是清欢。——苏轼《浣溪沙 从泗州刘倩叔游南山》
    5.绿蚁新醅酒，红泥小火炉。晚来天欲雪，能饮一杯无？——白居易《问刘十九》6.我有一瓢酒，可以慰风尘。——韦应物《简卢陟》
    7.少年听雨歌楼上，红烛昏罗帐。壮年听雨客舟中，江阔云低、断雁叫西风。而今听雨僧庐下，鬓已星星也。悲欢离合总无情。一任阶前、点滴到天明。——蒋捷《虞美人 听雨》
    8.君埋泉下泥销骨，我寄人间雪满头。——白居易《梦微之》9.直道相思了无益，未妨惆怅是清狂。——李商隐《无题六首其三》
    10.欲买桂花同载酒，终不似，少年游。——刘过《唐多令·芦叶满汀洲》11.人言落日是天涯，望极天涯不见家。已恨碧山相阻隔，碧山还被暮云遮。——李觏《乡思》
    12.以色列事他人，能得几时好。——李白《妾薄命》13.山有木兮木有枝，心悦君兮君不知。——《越人歌》
    14.从此无心爱良夜，任他明月下西楼。——李益《写情》
    15.故园便是无兵马，犹有归时一段愁。——陈与义《送人归京师》16.我是人间惆怅客，知君何事泪纵横，断肠声里忆平生。——纳兰性德《浣溪沙 残雪凝辉画冷屏》
</div>
<button>获取</button>
<button>设置</button>
<script type="text/javascript">
    $(function (){
        // 1.拿到button
        let btns = document.getElementsByTagName("button");
        // 2.监听获取
        btns[0].onclick = function (){
            // 2.1 获取滚动的偏移位
            // console.log($(".scroll").scrollTop());
            // 2.2 获取网页滚动的偏移位
            console.log($(".body").scrollTop()+$(".html").scrollTop());
        }
        // 3.设置网页偏移
        btns[1].onclick = function (){
            // 3.1 设置滚动的偏移位
            // $(".scroll").scrollTop(200);
            // 3.2 设置网页滚动偏移位
            $(".html,body").scrollTop(300);
        }
    })
</script>
</body>
</html>

```

## 7- jQuery事件

## 7.1 点击事件

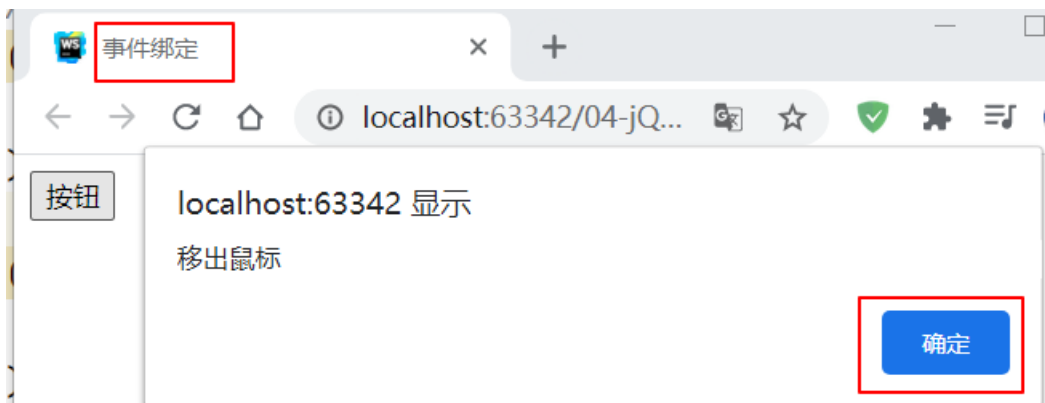
### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>事件绑定</title>
  <script src="js/jquery-1.12.4.js"></script>
</head>
<body>
<button>按钮</button>
<script type="text/javascript">
  $(function () {
    /*
    可以添加多个相同或者不同类型的事件, 不会覆盖
    */
    $("button").click(function () {
      alert("hello jQuery!!!");
    });

    $("button").click(function () {
      alert("guardwhy");
    });

    $("button").mouseleave(function () {
      alert("移出鼠标");
    });
    // 移入鼠标
    $("button").mouseenter(function () {
      alert("鼠标移入!!!");
    });
  });
</script>
</body>
</html>
```

### 2、执行结果



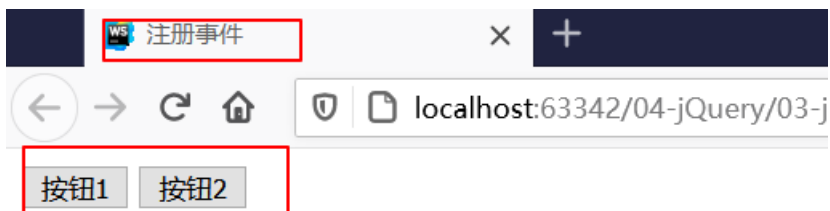
## 7.2 注册(绑定)事件

### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>注册事件</title>
  <script src="js/jquery-1.12.4.js"></script>
  <script type="text/javascript">
    /*
      1.1 on(type, callback): 注册事件。
      1.2 注册多个相同类型事件，后注册的不会覆盖先注册的。
      1.3 注册多个不同类型事件，后注册的不会覆盖先注册的。
    */
    $(function () {
      $("button").on("click", function () {
        alert("hello click1!!!");
      });
      $("button").on("click", function () {
        alert("hello click2!!!");
      });

      $("button").on("mouseenter", function () {
        alert("hello mouseenter!!!");
      });
      $("button").on("mouseleave", function () {
        alert("hello mouseleave!!!");
      });
    });
  </script>
</head>
<body>
  <button>按钮1</button>
  <button>按钮2</button>
</body>
</html>
```

### 2、执行结果



## 7.3 事件解绑

### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
```



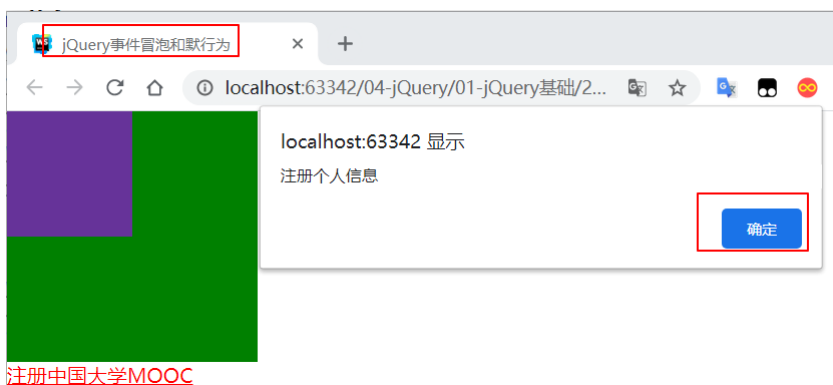
```

    }
    .div2{
        width: 100px;
        height: 100px;
        background: rebeccapurple;
    }
</style>
<script src="js/jquery-1.12.4.js"></script>
</head>
<body>
<div class="div1">
    <div class="div2"></div>
</div>
<a href="https://www.icourse163.org/">注册中国大学MOOC</a>
<script type="text/javascript">
    $(function (){
        // 1.阻止冒泡事件
        /*$(".div2").click(function (event){
            alert("div2");
            // 1.1 方式
            // return false;
            event.stopPropagation();
        });
        $(".div1").click(function (){
            alert("div1");
        })*/

        // 2.阻止默认行为
        $("a").click(function (event){
            alert("注册个人信息");
            // 2.1 方法
            // return false;
            event.preventDefault();
        });
    });
</script>
</body>
</html>

```

## 2、执行结果



## 7.5 事件自动触发

### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jquery事件自动触发</title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    .div1{
      width: 200px;
      height: 200px;
      background: green;
    }
    .div2{
      width: 100px;
      height: 100px;
      background: rebeccapurple;
    }
  </style>
  <script src="js/jquery-1.12.4.js"></script>
</head>
<body>
  <div class="div1">
    <div class="div2"></div>
  </div>
  <a href="https://www.icourse163.org/"><span>注册中国大学MOOC</span></a>
  <script type="text/javascript">
    $(function (){
      /*
      trigger: 如果利用trigger自动触发事件,会触发事件冒泡.
      triggerHandler: 如果利用triggerHandler自动触发事件, 不会触发事件冒泡.
      */
      $(".div2").click(function (event){
        alert("div2子类元素");
      });
      $(".div1").click(function (){
        alert("div1父类元素");
      });

      // $(".div2").trigger("click");
      // $(".div2").triggerHandler("click");

      // 2.超链接自动触发事件
      $("span").click(function (){
        alert("超链接...")
      });
      // 2.1 自动触发
      $("span").trigger("click");
    });
  </script>
</body>
```

```
</html>
```

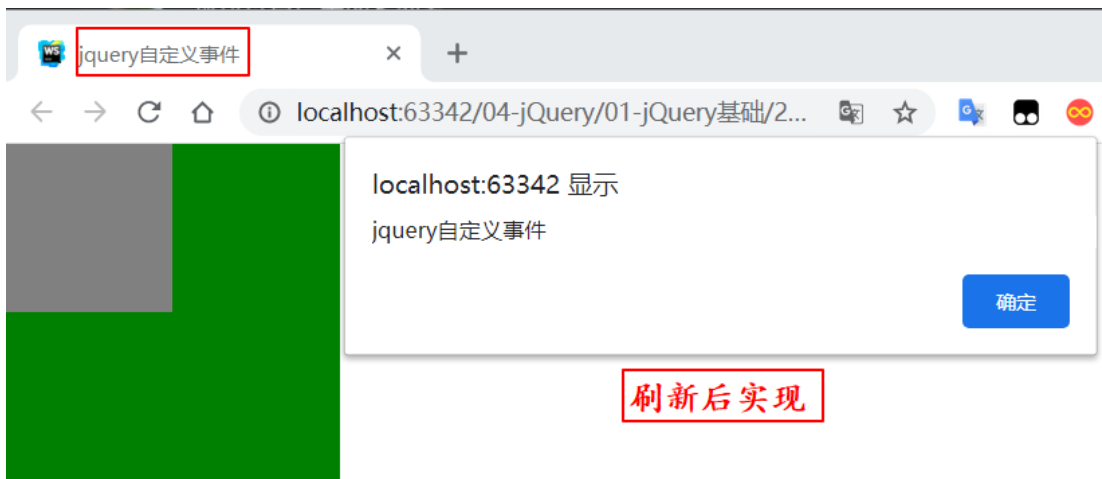
## 7.6 自定义事件

### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jquery自定义事件</title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    .div1{
      width: 200px;
      height: 200px;
      background: green;
    }
    .div2{
      width: 100px;
      height: 100px;
      background: gray;
    }
  </style>
  <script src="js/jquery-1.12.4.js"></script>
  <script type="text/javascript">
    /*
      想要自定义事件，必须满足两个条件
      1. 事件必须是通过on绑定的
      2. 事件必须通过trigger来触发
    */
    $(function (){
      $(".div2").on("myClick", function (){
        alert("jquery自定义事件");
      });
      $(".div2").trigger("myClick");
    });
  </script>
</head>
<body>
  <div class="div1">
    <div class="div2"></div>
  </div>
</body>
</html>
```

### 2、执行结果





## 7.7 事件命名空间

### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jquery命名空间</title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    .div1{
      width: 200px;
      height: 200px;
      background: green;
    }
    .div2{
      width: 100px;
      height: 100px;
      background: gray;
    }
  </style>
  <script src="js/jquery-1.12.4.js"></script>
  <script type="text/javascript">
    /*
    事件的命名空间有效，必须满足两个条件
    1. 事件必须是通过on绑定的
    2. 事件必须通过trigger来触发
    */
    $(function () {
      $(".div2").on("click.curry", function () {
        alert("jquery事件的命名空间1");
      });
      $(".div2").on("click.james", function () {
        alert("jquery事件的命名空间2");
      });

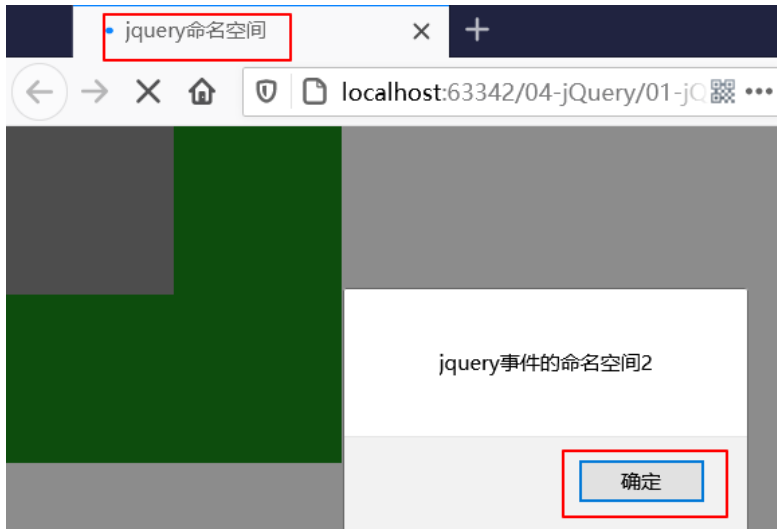
      $(".div2").trigger("click.james");
    });
  </script>
```

```

</head>
<body>
<div class="div1">
  <div class="div2"></div>
</div>
</body>
</html>

```

## 2、执行结果



## 7.8 事件委托

### 1、代码示例

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jquery事件委托</title>
  <script src="js/jquery-1.12.4.js"></script>
  <script type="text/javascript">

```

```

    /*
    事件委托：请别人帮忙做事情，然后将做完的结果返回
    */

```

```

    $(function (){
      $("button").click(function (){
        $("ul").append("<li>新增的li</li>");
      });
    })

```

以下代码的含义，让ul帮li监听click事件

之所以能够监听，是因为入口函数执行的时候ul就已经存在了，所以能够添加事件之所以this是

li。

是因为点击的是li，而li没有click事件，所以事件冒泡传递给了ul。

ul响应了事件，既然事件是从li传递过来的,所以ul必然指定this是谁。

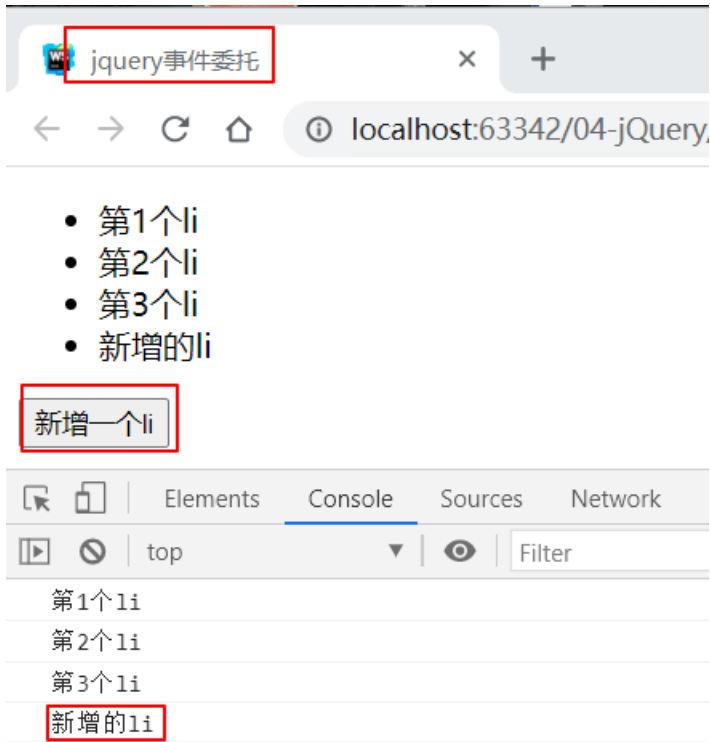
```

    /*
    $("ul").delegate("li", "click", function (){
      // 输出结果
      console.log($(this).html());
    })
  });
</script>
</head>

```

```
<body>
<ul>
  <li>第1个li</li>
  <li>第2个li</li>
  <li>第3个li</li>
</ul>
<button>新增一个li</button>
</body>
</html>
```

## 2、执行结果



## 7.9 移入移出事件

### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jquery移入移出事件</title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    .div1{
      width: 200px;
      height: 200px;
      background: green;
    }
    .div2{
      width: 100px;
      height: 100px;
      background: gray;
    }
  </style>
</head>
<body>
  <div class="div1">
    <div class="div2">
      移入事件
    </div>
    移出事件
  </div>
</body>
</html>
```

```

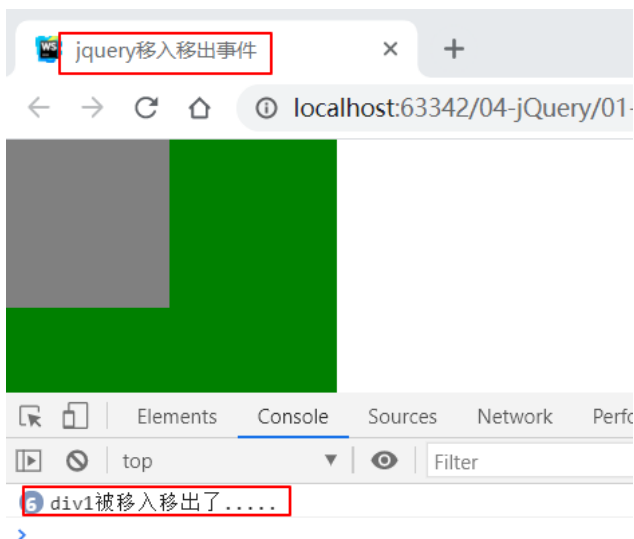
</style>
<script src="js/jquery-1.12.4.js"></script>
<script type="text/javascript">

    $(function () {
        /*
            子元素被移入移出不会触发父元素的事件
        */

        /*
            $(".div1").hover(function () {
                console.log("div1被移入了..");
            }, function () {
                console.log("div1被移出了");
            })
        */
        $(".div1").hover(function () {
            // 输出结果
            console.log("div1被移入移出了.....");
        });
    });
</script>
</head>
<body>
<div class="div1">
    <div class="div2"></div>
</div>
</body>
</html>

```

## 2、执行结果



## 7.10 事件加载

### 1、代码示例

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>加载事件</title>
    <script src="js/jquery-1.12.4.js"></script>

```

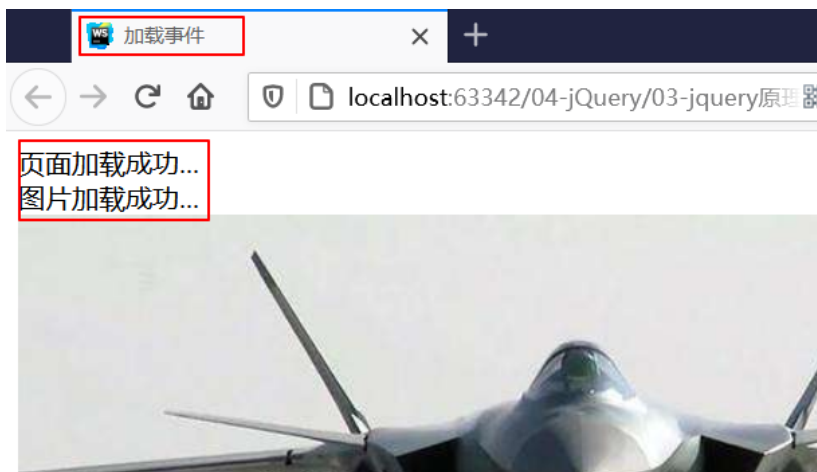
```

<script type="text/javascript">
    $(function () {
        // 1. 加载页面
        $(document).ready(function () {
            $("#message1").html("页面加载成功...");
        });
        // 2. 图片加载
        $("#img").load(function () {
            $("#message2").html("图片加载成功...");
        });
    });
</script>
</head>
<body>
<div id="message1"></div>
<div id="message2"></div>


</body>
</html>

```

## 2、执行结果



## 7.11 焦点事件

### 1、代码示例

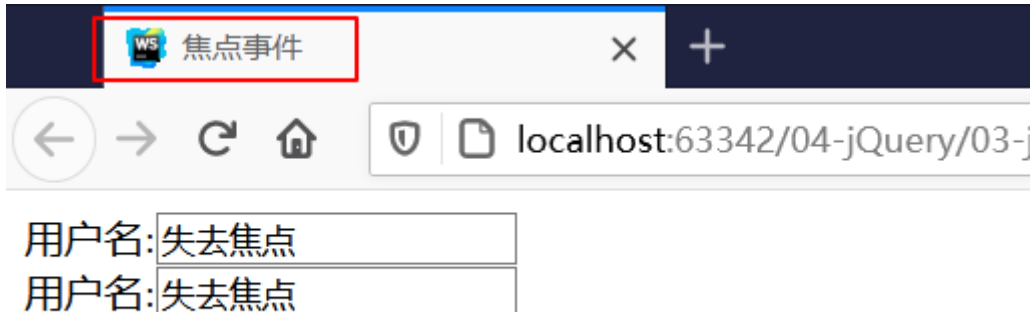
```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>焦点事件</title>
    <script src="js/jquery-1.12.4.js"></script>
    <script type="text/javascript">
        $(function () {
            $("input").focus(function () {
                $(this).val("获得焦点");
            });
            $("input").blur(function () {
                $(this).val("失去焦点");
            });
        });
    </script>

```

```
</head>
<body>
用户名:<input type="text"><br/>
用户名:<input type="text"><br/>
</body>
</html>
```

## 2、执行结果



# 8-jQuery 动画效果

## 8.1显示和隐藏动画

### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jquery显示和隐藏动画</title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    div{
      width: 200px;
      height: 200px;
      background: red;
      display: none;
    }
  </style>
  <script src="js/jquery-1.12.4.js"></script>
  <script type="text/javascript">
    $(function (){
      $("button").eq(0).click(function (){
        $("div").show(1000, function (){
          alert("显示动画执行完毕");
        })
      });

      $("button").eq(1).click(function (){
        $("div").hide(1000, function (){
          alert("隐藏动画执行完毕");
        })
      });
    });
  </script>
</html>
```

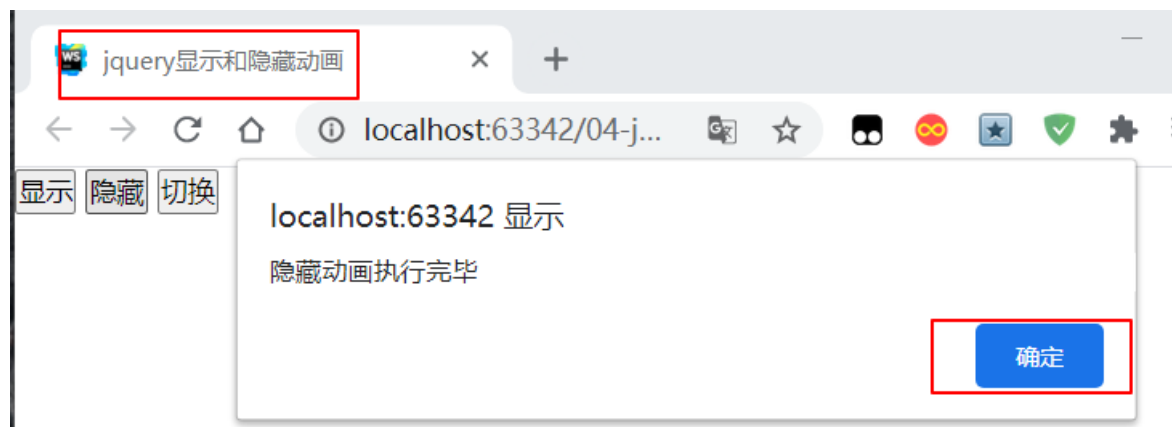
```

});

$("button").eq(2).click(function (){
    $("div").toggle(1000, function (){
        alert("切换动画执行完毕");
    });
});
});
});
</script>
</head>
<body>
<button>显示</button>
<button>隐藏</button>
<button>切换</button>
<div></div>
</body>
</html>

```

## 2、执行结果



## 8.2 展开和收起动画

### 1、代码示例

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>jQuery展开和收起动画</title>
<style type="text/css">
    *{
        margin: 0;
        padding: 0;
    }
    div{
        width: 100px;
        height: 200px;
        background: red;
        display: none;
    }
</style>
<script src="js/jquery-1.12.4.js"></script>
<script type="text/javascript">
    $(function (){

```

```

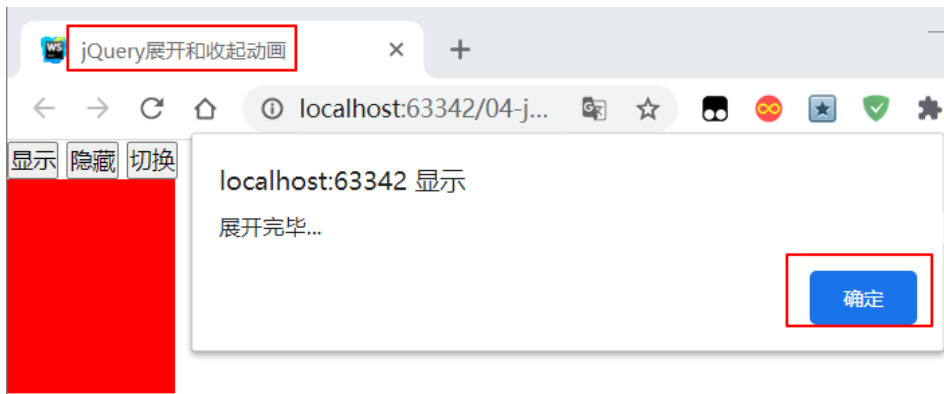
    $("button").eq(0).click(function (){
        $("div").slideDown(1000, function (){
            alert("展开完毕...");
        })
    });

    $("button").eq(1).click(function (){
        $("div").slideUp(1000, function (){
            alert("收起完毕..");
        })
    });

    $("button").eq(2).click(function (){
        $("div").slideToggle(1000, function (){
            alert("切换操作...");
        });
    });
});
</script>
</head>
<body>
<button>显示</button>
<button>隐藏</button>
<button>切换</button>
<div></div>
</body>
</html>

```

## 2、执行结果



## 8.3 折叠菜单

### 1、代码示例

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>折叠菜单</title>
    <style type="text/css">
        *{
            margin: 0;
            padding: 0;
        }
        .nav{

```



```

        list-style: none;
        width: 300px;
        margin: 100px auto;
    }
    .nav>li{
        border: 1px solid #000;
        line-height: 35px;
        border-bottom: none;
        text-indent: 2em;
        position: relative;
    }
    .nav>li:last-child{
        border-bottom: 1px solid #000;
        border-bottom-right-radius: 10px;
        border-bottom-left-radius: 10px;
    }
    .nav>li:first-child{
        border-top-right-radius: 10px;
        border-top-left-radius: 10px;
    }
    .nav>li>span{
        background: url("images/arrow_right.png") no-repeat center center;
        display: inline-block;
        width: 32px;
        height: 32px;
        position: absolute;
        right: 10px;
        top: 5px;
    }
    .sub{
        display: none;
    }
    .sub>li{
        list-style: none;
        background: mediumpurple;
        border-bottom: 1px solid white;
    }
    .sub>li:hover{
        background: red;
    }
    .nav>.current>span{
        transform: rotate(90deg);
    }
</style>
<script src="js/jquery-1.12.4.js"></script>
<script type="text/javascript">
    $(function (){
        // 1.1 监听一级菜单的点击事件
        $(".nav>li").click(function (){
            // 1.2. 拿到二级菜单
            let $sub = $(this).children(".sub");
            // 1.3 展开二级菜单展开
            $sub.slideDown(1000);
            // 1.4 拿到所有非当前的二级菜单
            let otherSub = $(this).siblings().children(".sub");
            // 1.5 让所有非当前的二级菜单收起
            otherSub.slideUp(1000);
            // 1.6 让被点击的一级菜单箭头旋转

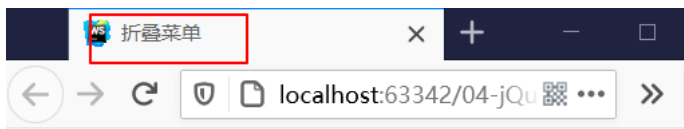
```

```

        $(this).addClass("current");
        // 1.7 让所有非点击的一级菜单箭头还原
        $(this).siblings().removeClass("current");
    })
});
</script>
</head>
<body>
<ul class="nav">
    <li>计算机语言<span></span>
        <ul class="sub">
            <li>java</li>
            <li>javascript</li>
            <li>HTML</li>
            <li>Python</li>
            <li>C++</li>
        </ul>
    </li>
    <li>计算机语言<span></span>
        <ul class="sub">
            <li>java</li>
            <li>javascript</li>
            <li>HTML</li>
            <li>Python</li>
            <li>C++</li>
        </ul>
    </li>
    <li>计算机语言<span></span>
        <ul class="sub">
            <li>java</li>
            <li>javascript</li>
            <li>HTML</li>
            <li>Python</li>
            <li>C++</li>
        </ul>
    </li>
    <li>计算机语言<span></span>
        <ul class="sub">
            <li>java</li>
            <li>javascript</li>
            <li>HTML</li>
            <li>Python</li>
            <li>C++</li>
        </ul>
    </li>
</ul>
</body>
</html>

```

## 2、执行结果



## 8.4 下拉菜单

### 1、代码示例

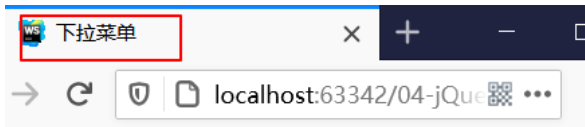
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>下拉菜单</title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    .nav{
      list-style: none;
      width: 300px;
      margin: 100px auto;
    }
    .nav>li{
      border: 1px solid #000;
      line-height: 35px;
      border-bottom: none;
      text-indent: 2em;
      position: relative;
    }
    .nav>li:last-child{
      border-bottom: 1px solid #000;
      border-bottom-right-radius: 10px;
      border-bottom-left-radius: 10px;
    }
    .nav>li:first-child{
      border-top-right-radius: 10px;
      border-top-left-radius: 10px;
    }
    .nav>li>span{
      background: url("images/arrow_right.png") no-repeat center center;
```

```

        display: inline-block;
        width: 32px;
        height: 32px;
        position: absolute;
        right: 10px;
        top: 5px;
    }
    .sub{
        display: none;
    }
    .sub>li{
        list-style: none;
        background: mediumpurple;
        border-bottom: 1px solid white;
    }
    .sub>li:hover{
        background: red;
    }
    .nav>.current>span{
        transform: rotate(90deg);
    }
</style>
<script src="js/jquery-1.12.4.js"></script>
<script type="text/javascript">
$(function (){
    /*
    在jQuery中如果需要执行动画，建议在执行动画之前先调用stop方法,然后再执行动画
    */
    // 1.1监听一级菜单的移入事件
    $(".nav>li").mouseenter(function (){
        // 1.2拿到二级菜单
        let $sub = $(this).children(".sub");
        // 1.3.停止当前的正在运行的动画
        $sub.stop();
        // 1.4 让二级菜单展开
        $sub.slideDown(1000);
    });
    // 2.监听一级菜单的移出事件'
    $(".nav>li").mouseleave(function (){
        // 1.2拿到二级菜单
        let $sub = $(this).children(".sub");
        // 1.3.停止当前的正在运行的动画
        $sub.stop();
        // 1.4 让二级菜单展开
        $sub.slideup(1000);
    });
});
</script>
</head>
<body>
<ul class="nav">
<li>计算机语言<span></span>
    <ul class="sub">
        <li>java</li>
        <li>javascript</li>
        <li>HTML</li>
        <li>Python</li>
        <li>C++</li>
    
```

```
</ul>
</li>
</ul>
</body>
</html>
```

## 2、执行结果



## 8.5 淡入淡出动画

### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jQuery淡入淡出动画</title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    div{
      width: 300px;
      height: 300px;
      background: red;
      display: none;
    }
  </style>
  <script src="js/jquery-1.12.4.js"></script>
  <script type="text/javascript">
    $(function (){
      // 1.1 淡入操作
      $("button").eq(0).click(function (){
        $("div").fadeIn(1000, function (){
          alert("淡入完毕...");
        });
      });
    });
  </script>
```

```

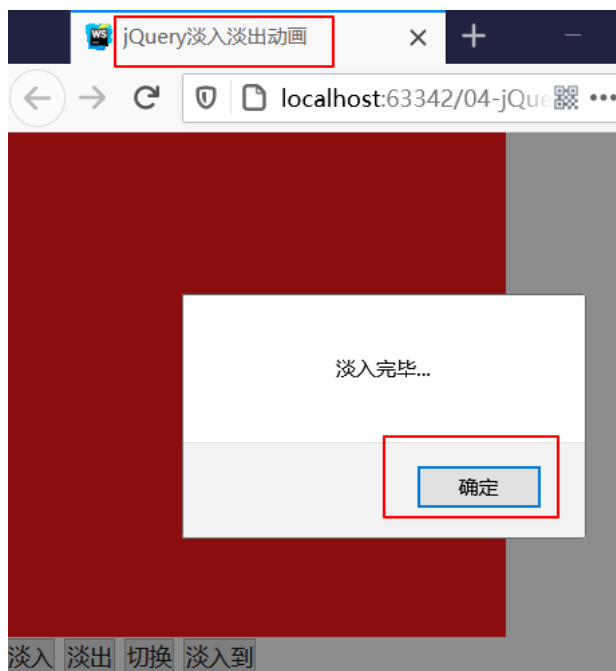
// 淡出完毕
$("#button").eq(1).click(function (){
    $("#div").fadeOut(1000, function (){
        alert("淡出完毕...");
    });
});

// 切换完毕
$("#button").eq(2).click(function (){
    $("#div").fadeToggle(1000, function (){
        alert("切换完毕...");
    });
});

// 淡入到..
$("#button").eq(0).click(function (){
    $("#div").fadeTo(1000, function (){
        alert("淡入完毕...");
    });
});
});
</script>
</head>
<body>
<div></div>
<button>淡入</button>
<button>淡出</button>
<button>切换</button>
<button>淡入到</button>
</body>
</html>

```

## 2、执行结果



## 8.6 弹框广告

### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>弹框广告</title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    .ad{
      position: fixed;
      right: 0;
      bottom: 0;
      display: none;
    }
    .ad>span{
      display: inline-block;
      width: 30px;
      height: 30px;
      position: absolute;
      top: 0;
      right: 0;
    }
  </style>
  <script src="js/jquery-1.12.4.js"></script>
  <script type="text/javascript">
    $(function (){
      // 1.监听span的点击事件
      $("span").click(function (){
        $(".ad").remove();
      });
      // 2.执行广告动画
      $(".ad").stop().slideDown(1000).fadeOut(1000).fadeIn(1000);
    });
  </script>
</head>
<body>
<div class="ad">
  
  <span></span>
</div>
</body>
</html>
```

### 2、执行结果



## 8.7 自定义动画属性

### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>自定义动画</title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    div{
      width: 100px;
      height: 100px;
      margin-top: 10px;
      background: red;
    }
    .two{
      background: green;
    }
  </style>
  <script src="js/jquery-1.12.4.js"></script>
  <script type="text/javascript">
    $(function (){
      $("button").eq(0).click(function (){
        /*
          第一个参数：接收一个对象，可以在对象中修改属性
          第二个参数：指定动画时长
          第三个参数：指定动画节奏，默认就是swing
          第四个参数：动画执行完毕之后的回调函数
        */
        $(".one").animate({
          marginLeft:500
        }, 5000, function (){
          alert("自定义动画执行完毕1...");
        });

        $(".two").animate({
          marginLeft:500
        }, 5000,"linear", function (){
```



```

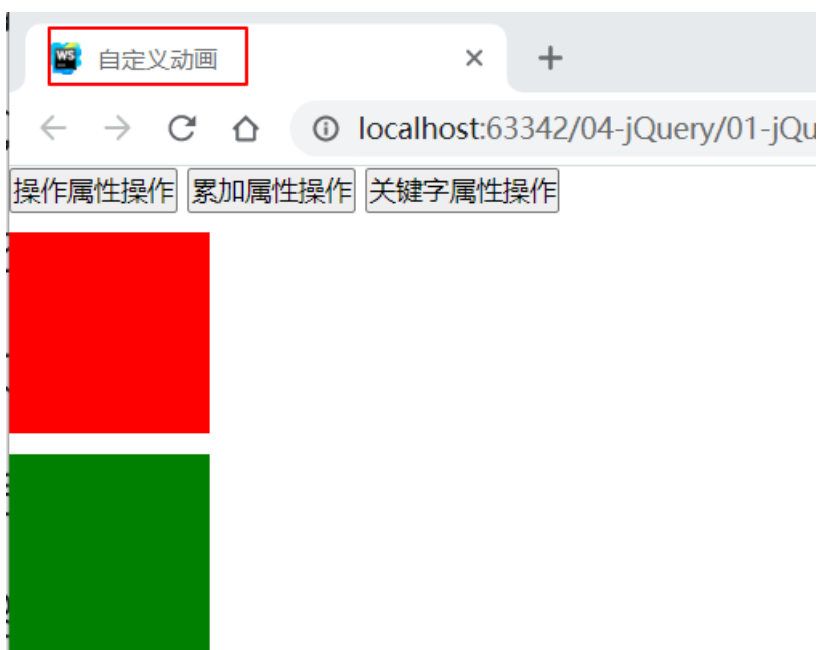
        alert("自定义动画执行完毕2...");
    });
});

$("button").eq(1).click(function (){
    // 1.累加属性操作
    $(".one").animate({
        width: "+=100"
    }, 1000, function (){
        alert("自定义动画(累加属性完毕)");
    });
});

$("button").eq(2).click(function (){
    // 1.累加属性操作
    $(".one").animate({
        // width: "hide"
        width: "toggle"
    }, 1000, function (){
        alert("自定义动画(关键字属性完毕)");
    });
});
});
</script>
</head>
<body>
<button>操作属性操作</button>
<button>累加属性操作</button>
<button>关键字属性操作</button>
<div class="one"></div>
<div class="two"></div>
</body>
</html>

```

## 2、执行结果



## 8.8 动画stop和delay方法

### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>stop和delay方法</title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    .one{
      width: 100px;
      height: 100px;
      background: blueviolet;
    }
  </style>
  <script src="js/jquery-1.12.4.js"></script>
  <script type="text/javascript">
    $(function (){
      $("button").eq(0).click(function (){
        /*
          在jQuery的{}中可以同时修改多个属性，多个属性的动画也会同时执行。
          delay方法的作用就是用于告诉系统延迟时长
        */

        /*
          $(".one").animate({
            width: 500
          }, 1000).delay(2000).animate({
            height: 500
          }, 1000);
        */

        $(".one").animate({
          width: 500
        }, 2000);
        $(".one").animate({
          height: 500
        }, 2000);

        $(".one").animate({
          width: 100
        }, 2000);
        $(".one").animate({
          height: 100
        }, 2000);
      });

      $("button").eq(1).click(function (){
        // 2.1立即停止当前动画,继续执行后续的动画
        /*
          $(".div").stop();
          $(".div").stop(false);
        */
      });
    });
  </script>
</html>
```

```

        $("div").stop(false, false);
        */

        //2.2立即停止当前和后续所有的动画
        /*
        $("div").stop(true);
        $("div").stop(true, false);
        */

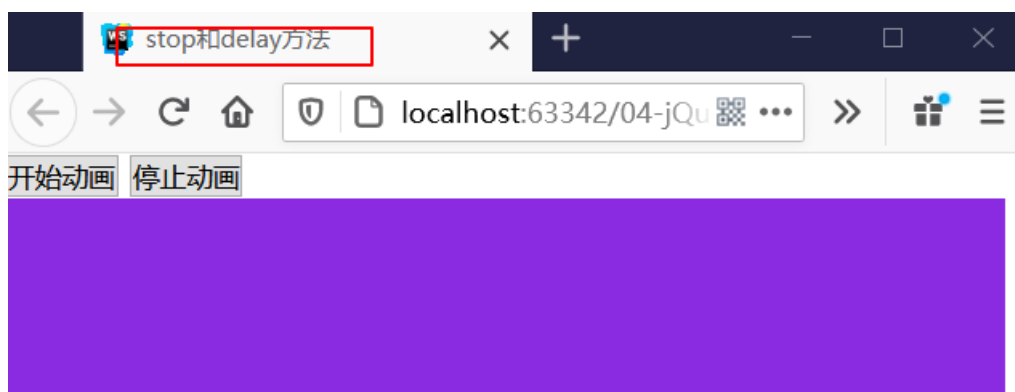
        // 2.3立即完成当前的，继续执行后续动画
        // $("div").stop(false, true);

        // 2.4 立即完成当前的，并且停止后续所有的
        $("div").stop(true, true);
    });

});
</script>
</head>
<body>
<button>开始动画</button>
<button>停止动画</button>
<div class="one"></div>
</body>
</html>

```

## 2、执行结果



# 9-jQuery 文档处理

## 9.1 添加节点

### 1、代码示例

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>jQuery添加节点</title>
    <script src="js/jquery-1.12.4.js"></script>
    <script type="text/javascript">
        $(function () {
            $("button").click(function () {
                // 1.创建一个节点
                let $li = $("<li>新增的li</li>");
            });
        });
    </script>

```

```

// 2.添加节点

/*
  内部插入
*/
// $("ul").append($li); // 最后
// $("ul").prepend($li); // 最前面

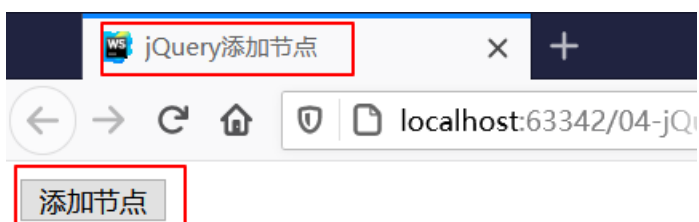
// $li.appendTo("ul"); // 后面
// $li.prependTo("ul"); // 最前面

/*
  外部插入
*/
// $("ul").after($li); // 会将元素添加到指定元素外部的后面
// $("ul").before($li); // 会将元素添加到指定元素外部的前面

$li.insertAfter("ul"); // 会将元素添加到指定元素外部的后面
});
});
</script>
</head>
<body>
<button>添加节点</button>
<ul>
  <li>第1个li</li>
  <li>第2个li</li>
  <li>第3个li</li>
</ul>
</body>
</html>

```

## 2、执行结果



- 第1个li
- 第2个li
- 第3个li

新增的li

## 9.2 删除节点

### 1、代码示例

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jQuery删除节点</title>
  <script src="js/jquery-1.12.4.js"></script>

```

```

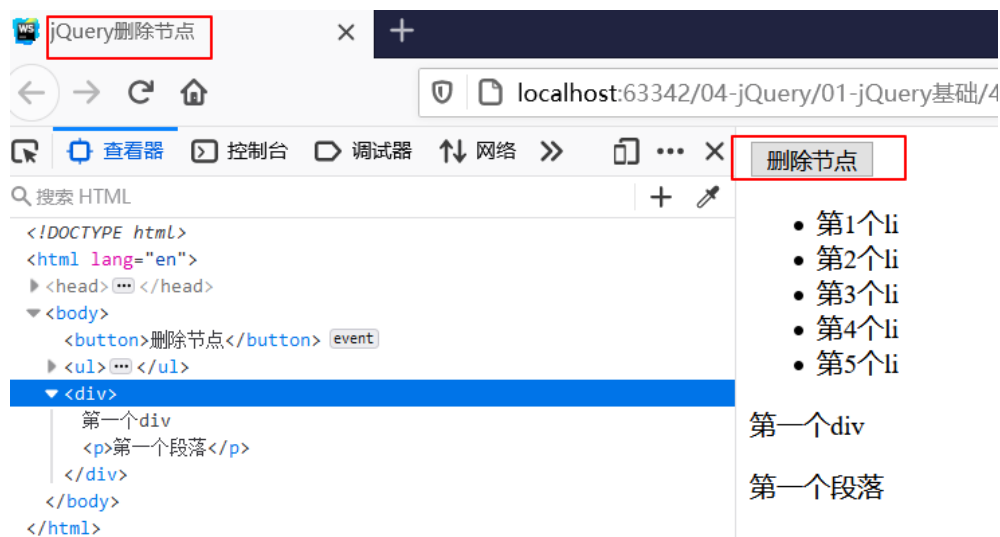
<script type="text/javascript">
  $(function (){
    /*
    删除：remove([expr])
    删除指定元素：empty()
    删除指定元素的内容和子元素，指定元素自身不会被删除：detach([expr])
    */
    $("button").click(function (){
      // $("div").remove();
      // $("div").empty();
      // $("li").remove(".item");

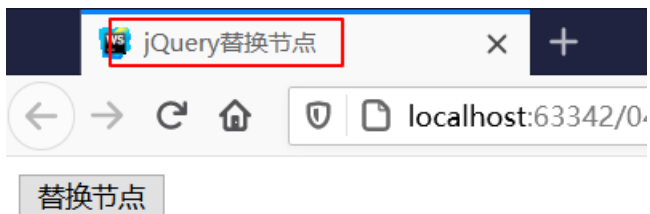
      // 1.利用remove删除之后再重新添加,原有的事件无法响应
      /*
      let $div = $("div").remove();
      console.log($div);
      */

      // 2.利用detach删除之后再重新添加,原有事件可以响应
      let $div = $("div").detach();
      console.log($div);
      $("body").append($div);
    });
  });
</script>
</head>
<body>
<button>删除节点</button>
<ul>
  <li class="item">第1个li</li>
  <li>第2个li</li>
  <li class="item">第3个li</li>
  <li>第4个li</li>
  <li class="item">第5个li</li>
</ul>
<div>
  第一个div
  <p>第一个段落</p>
</div>
</body>
</html>

```

## 2、执行结果





## Vue.js

## Vue.js

CSS

### 9.4 复制节点

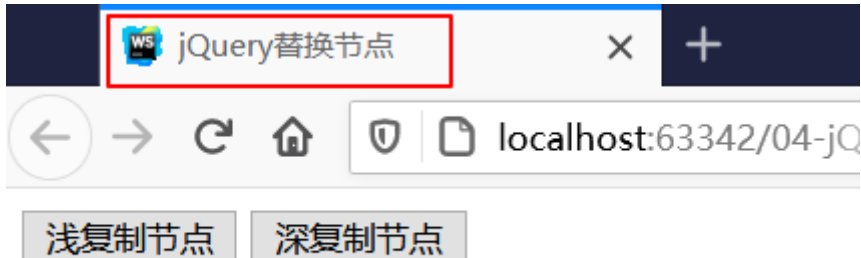
#### 1、代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jQuery替换节点</title>
  <script src="js/jquery-1.12.4.js"></script>
  <script type="text/javascript">
    $(function (){
      /*
      clone([Even[,deepEven]])
      如果传入false就是浅复制，如果传入true就是深复制
      浅复制只会复制元素，不会复制元素的事件
      深复制会复制元素，而且还会复制元素的事件
      */
      $("#button").eq(0).click(function (){
        // 1.1浅复制一个元素
        let $li = $("li:first").clone(false);
        // 1.2 将复制的元素添加到ul中
        $("ul").append($li);
      });

      $("#button").eq(1).click(function (){
        // 1.1深复制一个元素
        let $li = $("li:first").clone(true);
        // 1.2 将复制的元素添加到ul中
        $("ul").append($li);
      });
      $("li").click(function (){
        alert($(this).html());
      });
    });
  </script>
</head>
<body>
  <button>浅复制节点</button>
  <button>深复制节点</button>
  <ul>
    <li>java</li>
    <li>vue.js</li>
```

```
</li>html</li>
</li>C++</li>
</li>CSS</li>
</ul>
</body>
</html>
```

## 2、执行结果



- java
- vue.js
- html
- C++
- CSS
- java
- java

5

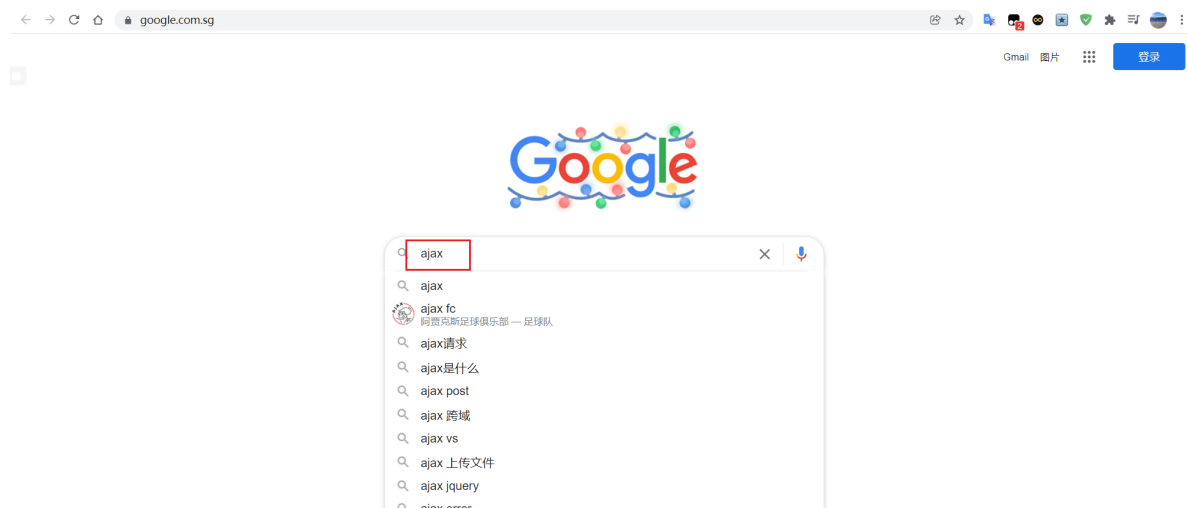
# 10- jQuery AJAX

## 10.1 AJAX基本介绍

AJAX(Asynchronous JavaScript And XML): 异步的 JavaScript 和 XML。本身不是一种新技术,而是多个技术综合。用于快速创建动态网页的技术。

一般的网页如果需要更新内容,必需重新加载个页面。而 AJAX 通过浏览器与服务器进行少量数据交换,就可以使网页实现异步更新,也就是在不重新加载整个页面的情况下,对网页的部分内容进行局部更新。

比如通过Google搜索关键字 AJAX, 搜索框下面弹出来的搜索内容【联想查询】, 就是局部更新。

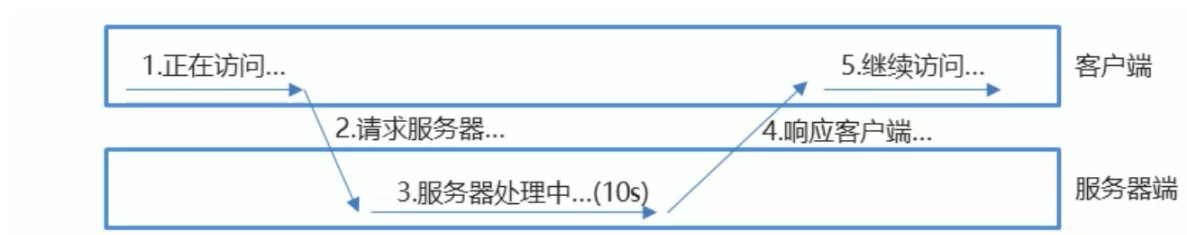




## 10.2 同步和异步操作

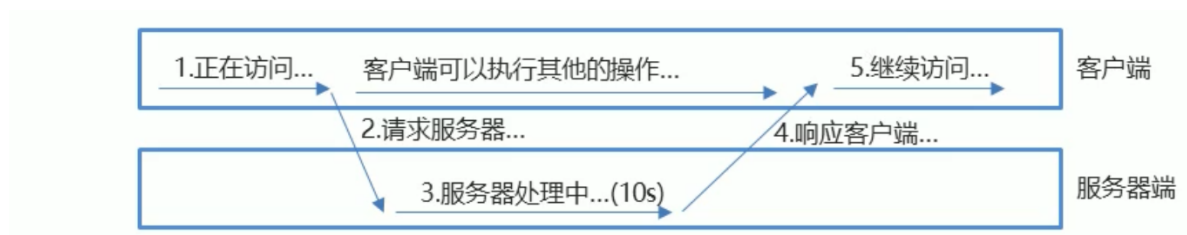
### 1、同步操作

客户端向服务器发生请求后，需要等待服务器的响应以后，中途客户端不能执行其他操作。



### 2、异步操作

客户端向服务器发生请求后，在等待服务器的响应过程中，中途客户端可以执行其他操作。



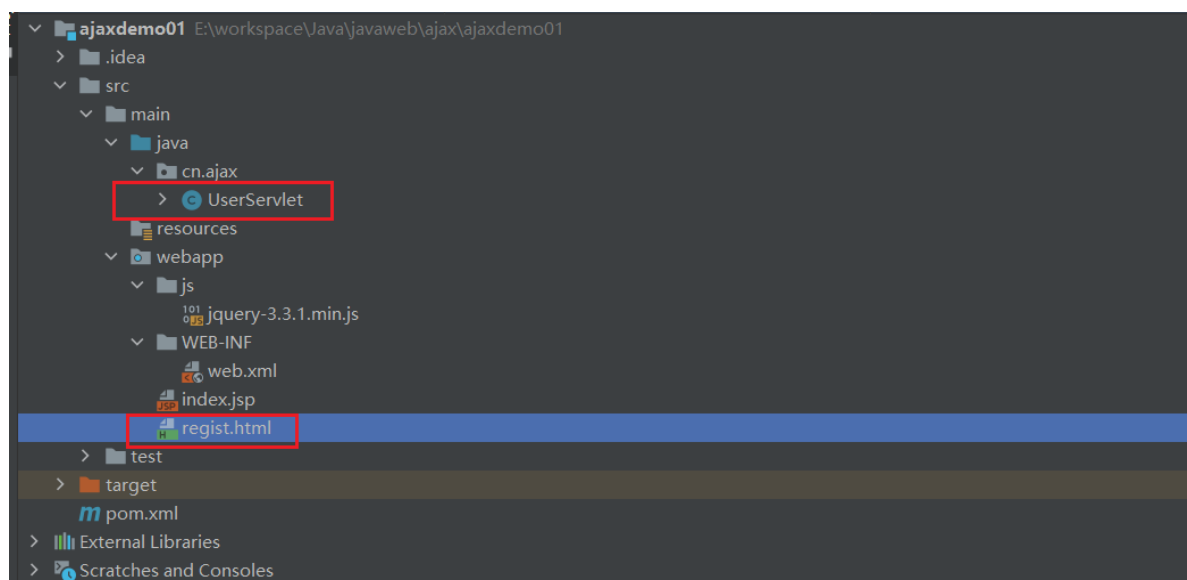
## 10.3 JQuery实现AJAX

### 10.3.1 Get方式实现

核心语法: `$.get(url,[data],[callback],[type]);`

| 基本参数                  | 作用   |
|-----------------------|--|
| <code>url</code>      | 请求的资源路径。   |
| <code>data</code>     | 发送给服务器端的请求参数，格式可以是 <code>key=value</code> ，也可以是 <code>js</code> 对象。  |
| <code>callback</code> | 当请求成功后的回调函数，可以在函数中编写逻辑代码。  |
| <code>type</code>     | 预期的返回数据的类型，取值可以是 <code>xml</code> , <code>html</code> , <code>js</code> , <code>json</code> , <code>text</code> 等。 |

### 1、创建Maven工程，导入项目所需依赖



相关依赖: pom.xml

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

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>cn.guardwhy</groupId>
  <artifactId>ajaxdemo01</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <!--导入相关依赖-->
  <dependencies>
    <!--Servlet-->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>4.0.1</version>
    </dependency>
    <!--jsp-->
    <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>javax.servlet.jsp-api</artifactId>
      <version>2.3.3</version>
    </dependency>
  </dependencies>
  <!--在build中配置resources，来防止我们资源导出失败的问题-->
  <build>
    <resources>
      <resource>
        <directory>src/main/resources</directory>
        <includes>
          <include>**/*.properties</include>
          <include>**/*.xml</include>
        </includes>
        <filtering>true</filtering>
      </resource>
      <resource>
        <directory>src/main/java</directory>
        <includes>
          <include>**/*.properties</include>
          <include>**/*.xml</include>
        </includes>
        <filtering>true</filtering>
      </resource>
    </resources>
  </build>
</project>
```

## 2、服务端实现: UserServlet

```
package cn.ajax;
```

```

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet("/userService")
public class UserService extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        // 设置请求和响应乱码
        req.setCharacterEncoding("UTF-8");
        resp.setContentType("text/html;charset=UTF-8");

        // 获取请求参数
        String username = req.getParameter("username");

        // 判断姓名是否已经注册
        if("curry".equals(username)){
            resp.getWriter().write("<font color = 'red'>用户名已注册</font>");
        }else {
            resp.getWriter().write("<font color = 'green'>用户名可用</font>");
        }
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        super.doPost(req, resp);
    }
}

```

### 3、页面端实现: regist.html

```

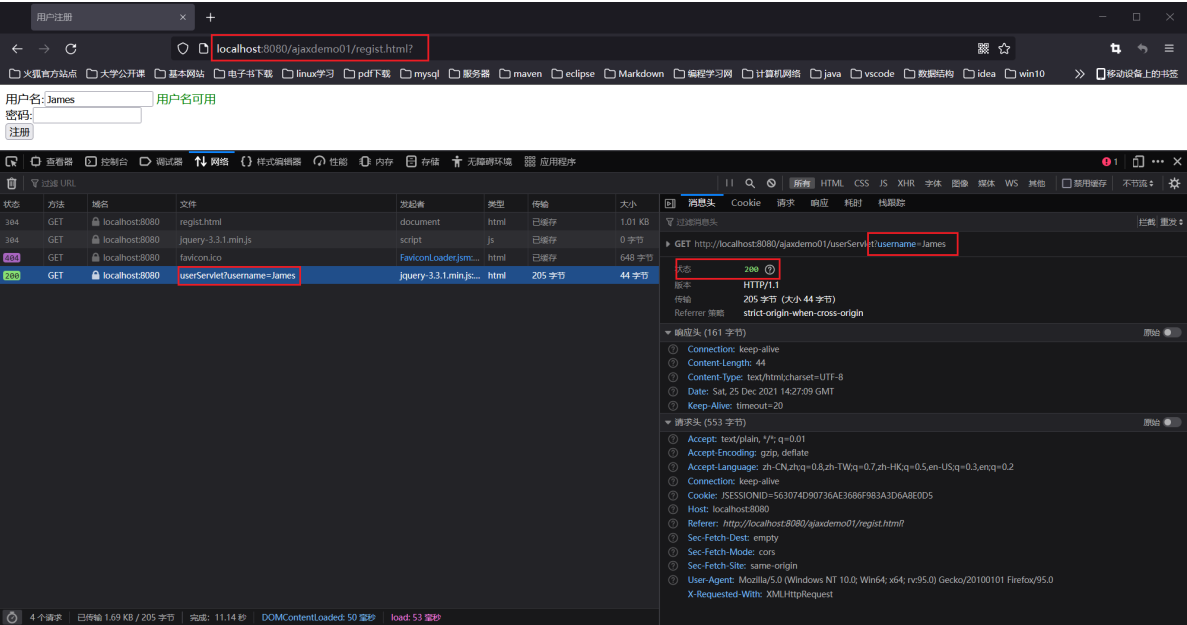
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>用户注册</title>
</head>
<body>
    <form autocomplete="off">
        用户名:<input type="text" id="username">
        <span id="IdSpan"></span>
        <br/>
        密码:<input type="password" id="password">
        <br/>
        <input type="submit" value="注册">
    </form>
</body>

<!--引入jquery-->
<script src = "js/jquery-3.3.1.min.js"></script>
<script>
    // 失去焦点

```

```
$( "#username" ).blur(function () {
    // 拿到用户名
    let username = $( "#username" ).val();
    // jQuery的Get方式实现AJAX
    $.get(
        // 请求资源路径
        "userServlet",
        // 请求参数
        "username=" + username,
        // 回调函数
        function (data) {
            // 响应数据显示到span标签上
            $( "#IdSpan" ).html( data );
        },
        // 响应数据形式
        "text"
    );
});
</script>
</html>
```

4、启动服务器，执行成功



10.3.2 POST方式实现

核心语法: \$.post(url,[data],[callback],[type]);

| 基本参数     | 作用  |
|----------|---|
| url      | 请求的资源路径。  |
| data     | 发送给服务器端的请求参数，格式可以是 key=value ，也可以是 js 对象。                   |
| callback | 当请求成功后的回调函数，可以在函数中编写逻辑代码。                                   |
| type     | 预期的返回数据的类型，取值可以是 `xml` , `html` , `js` , `json` , `text` 等。 |

1、服务端实现: UserServlet

```

import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet("/userService")
public class UserService extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        super.doGet(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        // 设置请求和响应乱码
        req.setCharacterEncoding("UTF-8");
        resp.setContentType("text/html; charset=UTF-8");

        // 获取请求参数
        String username = req.getParameter("username");

        // 判断姓名是否已经注册
        if("curry".equals(username)){
            resp.getWriter().write("<font color = 'red'>用户名已注册</font>");
        }else {
            resp.getWriter().write("<font color = 'green'>用户名可用</font>");
        }
    }
}

```

## 2、页面端实现: regist.html

```

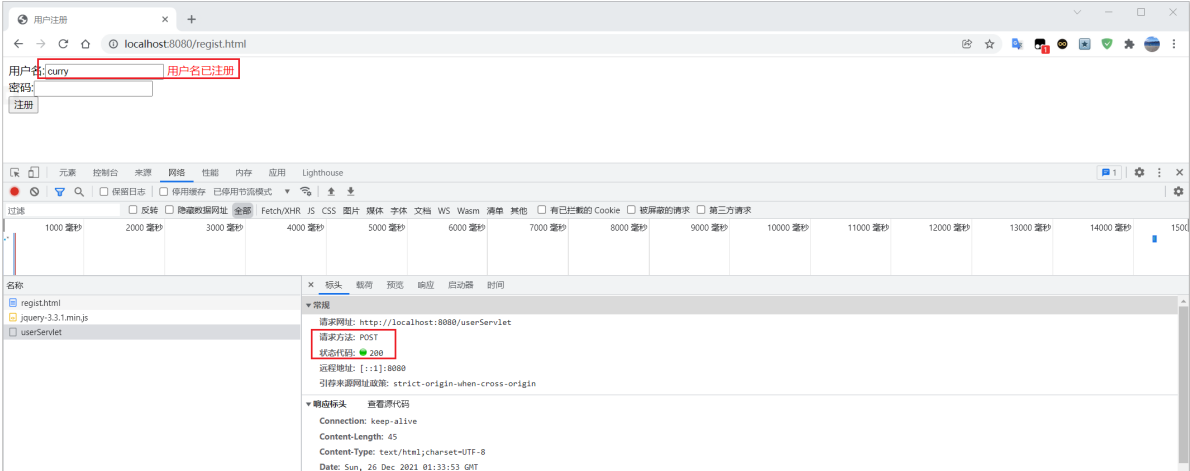
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>用户注册</title>
</head>
<body>
    <form autocomplete="off">
        用户名:<input type="text" id="username">
        <span id="IdSpan"></span>
        <br/>
        密码:<input type="password" id="password">
        <br/>
        <input type="submit" value="注册">
    </form>
</body>

<!--引入jquery-->
<script src = "js/jquery-3.3.1.min.js"></script>
<script>
    // 失去焦点
    $("#username").blur(function () {
        // 拿到用户名
        let username = $("#username").val();
        // jQuery的Post方式实现AJAX
    });

```

```
$.post(  
    // 请求资源路径  
    "userServlet",  
    // 请求参数  
    "username=" + username,  
    // 回调函数  
    function (data) {  
        // 响应数据显示到span标签上  
        $("#IdSpan").html(data);  
    },  
    // 响应数据形式  
    "text"  
);  
});  
</script>  
</html>
```

3、启动服务器，执行成功



10.3.3 通用方式实现AJAX

核心语法: \$.ajax({name:value,name:value,...});

| 基本参数     | 作用                                      |
|----------|---|
| url      | 请求的资源路径。                                |
| async    | 是否异步请求，true-是，false-否【默认是true】          |
| data     | 发送给服务器端的请求参数，格式可以是key=value，也可以是js对象。   |
| type     | 请求方式，POST或GET【默认是GET】                   |
| dataType | 预期的返回数据的类型，取值可以是xml,html,js,json,text等。 |
| success  | 请求成功时调用的回调函数。                           |
| error    | 请求失败时调用的回调函数。                           |

1、服务端实现: UserServlet

```
package cn.ajax;  
  
import javax.servlet.ServletException;
```

```

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet("/userServlet")
public class UserServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        // 设置请求和响应乱码
        req.setCharacterEncoding("UTF-8");
        resp.setContentType("text/html;charset=UTF-8");

        // 获取请求参数
        String username = req.getParameter("username");

        // 判断姓名是否已经注册
        if("curry".equals(username)){
            resp.getWriter().write("<font color = 'red'>用户名已注册</font>");
        }else {
            resp.getWriter().write("<font color = 'green'>用户名可用</font>");
        }
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        // 设置请求和响应乱码
        req.setCharacterEncoding("UTF-8");
        resp.setContentType("text/html;charset=UTF-8");

        // 获取请求参数
        String username = req.getParameter("username");

        // 判断姓名是否已经注册
        if("curry".equals(username)){
            resp.getWriter().write("<font color = 'red'>用户名已注册</font>");
        }else {
            resp.getWriter().write("<font color = 'green'>用户名可用</font>");
        }
    }
}

```

## 2、页面端实现: `regist.html`

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>用户注册</title>
</head>
<body>
    <form autocomplete="off">
        用户名:<input type="text" id="username">
    
```

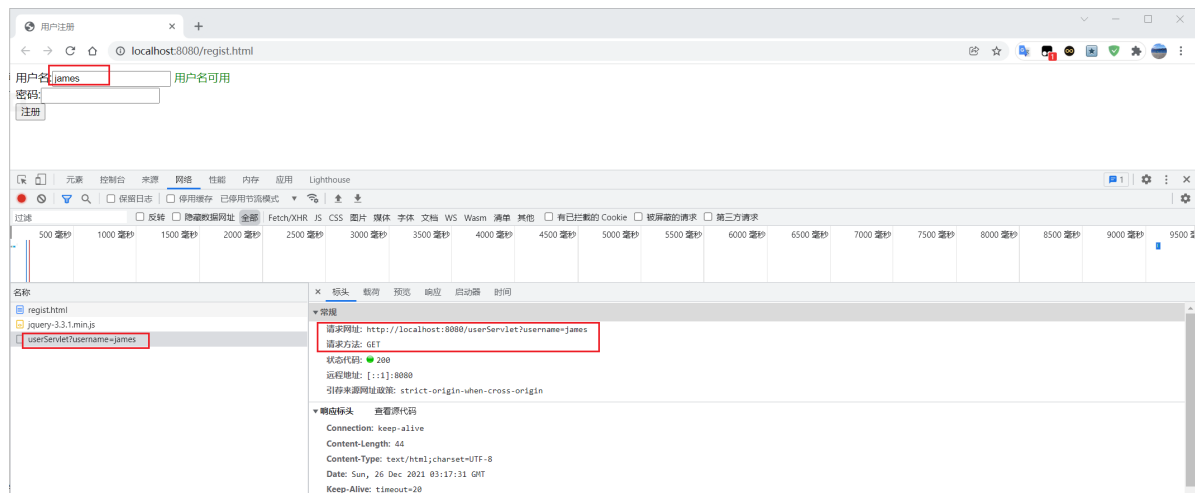
```

<span id="IdSpan"></span>
<br/>
密码:<input type="password" id="password">
<br/>
<input type="submit" value="注册">
</form>
</body>

<!--引入jquery-->
<script src = "js/jquery-3.3.1.min.js"></script>
<script>
    // 失去焦点
    $("#username").blur(function () {
        // 拿到用户
        let username = $("#username").val();
        // jQuery的通用方式实现AJAX
        $.ajax({
            // 请求资源路径
            url: "/userServlet",
            // 是否异步
            async:true,
            // 请求参数
            data:"username="+username,
            // 请求参数
            type:'GET',
            // 数据形式
            dataType:"text",
            // 请求成功后回调回调函数
            success:function (data){
                // 将响应的数据显示到span标签
                $("#IdSpan").html(data);
            },
            // 请求失败后调用的回调函数
            error:function (){
                alert("操作失败....")
            }
        })
    });
</script>
</html>

```

### 3、启动服务器，执行成功





# 10.4 JSON处理

## 10.4.1 基本介绍

### 1、JSON基本介绍

- JSON【JavaScript Object Notation, JS 对象标记】是一种轻量级的数据交换格式。
- 采用完全独立于编程语言的文本格式来存储和表示数据。简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言。

### 2、注意事项

在 JavaScript 语言中，一切都是对象。因此，任何 JavaScript 支持的类型都可以通过 JSON 来表示。JSON 键值对是用来保存 JavaScript 对象的一种方式 and JavaScript 对象的写法也大同小异。键/值对组合中的键名写在前面并用双引号 "" 包裹，使用冒号 : 分隔，然后紧接着值：

- 对象表示为键值对，数据由逗号分隔。
- 花括号保存对象
- 方括号保存数组

| 对象类型    | 语法                                      |  |
|---------|---|--|
| 对象类型    | {name:value,name:value...}              |  |
| 数组/集合类型 | [{name:value,...},{name:value,...}]     |  |
| 混合类型    | {name:[{namev:val...},{name:value...}]} |  |

name是字符串类型，value可以是任意类型！！！！

### 3、常用方法

| 成员方法          | 说明                  |
|---------------|---------------------|
| stringify【对象】 | 将指定对象转换为 json 格式字符串 |
| parse【字符串】    | 将指定 json 格式字符串解析成对象 |

### 案例实现

```
<script type="text/javascript">
// 1.编写一个JS对象
var user = {
    name: "guardwhy",
    age: 26,
    sex: "男"
};

// 2.将js对象转换成JSON对象
var json = JSON.stringify(user);
console.log(json);

// 输出结果
console.log("=====");

// 3.将JSON对象转换成JavaScript对象
var obj = JSON.parse(json);
```

```
console.log(obj);
</script>
```

## 4、JSON 和 JS 区别

JSON 是 JavaScript 对象的字符串表示法，它使用文本表示一个 JS 对象的信息，本质是一个字符串。

```
var obj = {a: 'guard', b: 'why'};    //这是一个对象，注意键名也是可以使用引号包裹的
var json = '{"a": "guard", "b": "why"}';    //这是一个 JSON 字符串，本质是一个字符串
```

## 5、JSON 和 JS 互转

要实现从 JSON 字符串转换为 JavaScript 对象，使用 JSON.parse() 方法：

```
var obj = JSON.parse('{"a": "guard", "b": "why"}');    //结果是 {a: 'guard', b: 'why'}
```

要实现从 JavaScript 对象转换为 JSON 字符串，使用 JSON.stringify() 方法：

```
var json = JSON.stringify({a: 'Hello', b: 'world'});    //结果是 '{"a": "Hello", "b": "world"}'
```

## 10.4.2 JSON 转换工具

### 1、基本概念

除了可以在 JavaScript 中来使用 JSON 以外，在 JAVA 中同样也可以使用 JSON。JSON 的转换工具是通过 JAVA 封装好的一些 JAR 工具包。可以将 JAVA 对象或集合转换成 JSON 格式的字符串，也可以将 JSON 格式的字符串转成 JAVA 对象。

### 2、Jackson

Jackson：开源免费的 JSON 转换工具，SpringMVC 转换默认使用 Jackson。

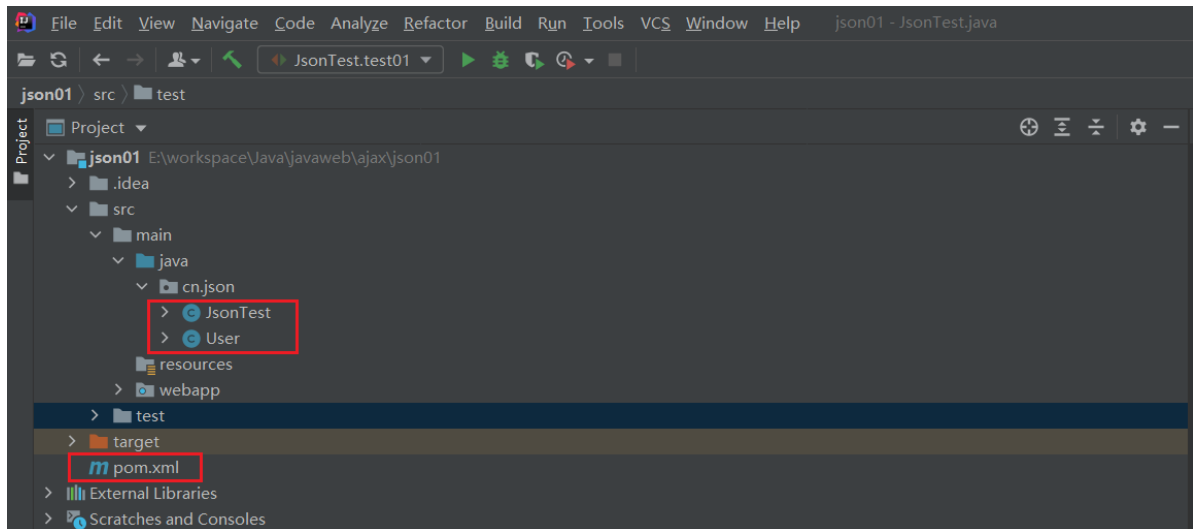
| 类名            | 具体使用   |
|---------------|--|
| ObjectMapper  | 是 Jackson 工具包的核心类，它提供一些方法来实现 JSON 字符串和对象之间的转换。 |
| TypeReference | 对集合泛型的反序列化，使用 TypeReference 可以明确的指定反序列化的对象类型。  |

ObjectMapper 类常用方法

| 方法名  | 具体使用                  |
|--|-----------------------|
| String writeValueAsString(Object obj)                    | 将 Java 对象转换成 JSON 字符串 |
| <T> T readValue(String json, Class<T> valueType)         | 将 JSON 字符串转换成 Java 对象 |
| <T> T readValue(String json, TypeReference valueTypeRef) | 将 JSON 字符串转换成 Java 对象 |

### 10.4.3 JSON转换

创建Maven工程，导入项目所需依赖



依赖：pom.xml

```
<!--导入相关依赖-->
<dependencies>
    <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-
    annotations -->
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-annotations</artifactId>
        <version>2.13.0</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-
    core -->
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-core</artifactId>
        <version>2.13.0</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-
    databind -->
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.13.0</version>
    </dependency>
    <!--测试类-->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
    </dependency>
</dependencies>
```

#### 1、对象转JSON，JSON转对象

实体类：User

```
package cn.json;
```

```

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@AllArgsConstructor
@Data
@NoArgsConstructor
// 实体类
public class User {
    private String name;
    private Integer age;
}

```

测试类: `JsonTest`

```

package cn.json;

import com.fasterxml.jackson.databind.ObjectMapper;
import org.junit.Test;

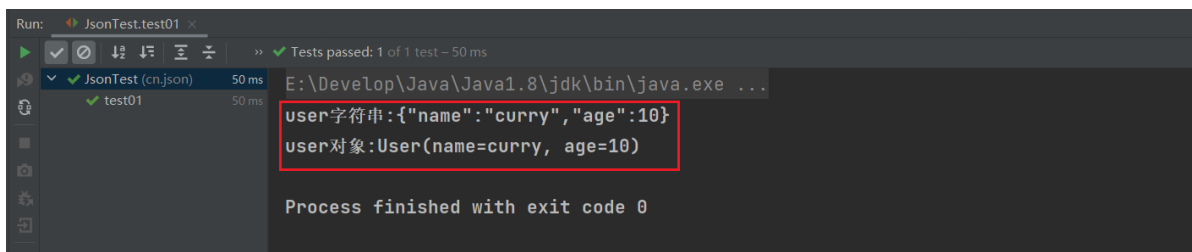
public class JsonTest {
    // 创建Mapper对象
    private ObjectMapper mapper = new ObjectMapper();

    @Test
    public void test01() throws Exception {
        // 对象转json
        User user1 = new User("curry", 10);
        String json = mapper.writeValueAsString(user1);
        System.out.println("user字符串:" + json);

        // json转对象
        User user2 = mapper.readValue(json, User.class);
        System.out.println("user对象:" + user2);
    }
}

```

执行结果



## 2、Map 转 JSON, JSON 转 Map

测试类: `JsonTest`

```

package cn.json;

import com.fasterxml.jackson.databind.ObjectMapper;
import org.junit.Test;

```

```

import java.util.HashMap;

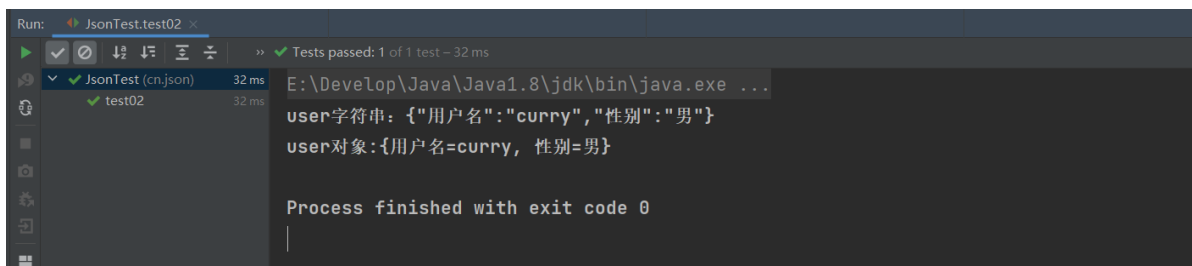
public class JsonTest {
    // 创建Mapper对象
    private ObjectMapper mapper = new ObjectMapper();

    @Test
    public void test02() throws Exception{
        // Map转json
        HashMap<String, String> map1 = new HashMap<>();
        map1.put("用户名", "curry");
        map1.put("性别", "男");
        String json = mapper.writeValueAsString(map1);
        System.out.println("user字符串: " + json);

        // json转map
        HashMap<String, String> map2 = mapper.readValue(json, HashMap.class);
        System.out.println("user对象:" + map2);
    }
}

```

## 执行结果



## 3、Map 转 JSON，JSON 转 Map

测试类: `JsonTest`

```

package cn.json;

import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.junit.Test;

import java.util.HashMap;

public class JsonTest {
    // 创建Mapper对象
    private ObjectMapper mapper = new ObjectMapper();

    @Test
    public void test03() throws Exception{
        // Map转json
        HashMap<String, User> map1 = new HashMap<>();
        map1.put("东部联盟", new User("curry", 10));
        map1.put("西部联盟", new User("字母哥", 29));
        String json = mapper.writeValueAsString(map1);
        System.out.println("user字符串: " + json);

        // json转map
    }
}

```

```

        HashMap<String, User> map2 = mapper.readValue(json, new
        TypeReference<HashMap<String, User>>(){});
        System.out.println("user对象:" + map2);
    }
}

```

执行结果

```

Run: JsonTest.test03
Tests passed: 1 of 1 test - 60 ms
E:\Develop\Java\Java1.8\jdk\bin\java.exe ...
user字符串: {"西部联盟":{"name":"字母哥","age":29},"东部联盟":{"name":"curry","age":10}}
user对象:{西部联盟=User(name=字母哥, age=29), 东部联盟=User(name=curry, age=10)}

Process finished with exit code 0

```

#### 4、List转JSON,JSON 转 List

测试类: `JsonTest`

```

package cn.json;

import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.junit.Test;

import java.util.ArrayList;
import java.util.HashMap;

public class JsonTest {
    // 创建Mapper对象
    private ObjectMapper mapper = new ObjectMapper();

    @Test
    public void test04() throws Exception{
        // List转json
        ArrayList<String> list1 = new ArrayList<>();
        list1.add("curry");
        list1.add("james");
        String json = mapper.writeValueAsString(list1);
        System.out.println("user字符串:" + json);

        // json转List<String>
        ArrayList<String> list2 = mapper.readValue(json, ArrayList.class);
        System.out.println("user对象:" + list2);
    }
}

```

执行结果

```

Run: JsonTest.test04
Tests passed: 1 of 1 test - 29 ms
E:\Develop\Java\Java1.8\jdk\bin\java.exe ...
user字符串:["curry","james"]
user对象:[curry, james]

Process finished with exit code 0

```

## 5、List转JSON,JSON 转 List

测试类: `JsonTest`

```
package cn.json;

import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.junit.Test;


import java.util.ArrayList;
import java.util.HashMap;

public class JsonTest {
    // 创建Mapper对象
    private ObjectMapper mapper = new ObjectMapper();

    @Test
    public void test05() throws Exception{
        // List转json
        ArrayList<User> list1 = new ArrayList<>();
        list1.add(new User("curry", 10));
        list1.add(new User("字母哥", 10));
        String json = mapper.writeValueAsString(list1);
        System.out.println("user字符串:" + json);

        // json转List<String>
        ArrayList<User> list2 = mapper.readValue(json, new
TypeReference<ArrayList<User>>() {});
        System.out.println("user对象:" +list2);
    }
}
```

执行结果



```
Run: JsonTest.test05
Tests passed: 1 of 1 test - 56 ms
E:\Develop\Java\Java1.8\jdk\bin\java.exe ...
user字符串:[{"name":"curry","age":10},{"name":"字母哥","age":10}]
user对象:[User(name=curry, age=10), User(name=字母哥, age=10)]
Process finished with exit code 0
```

## 10.5 联想查询实现

### 10.5.1 案例需求

#### 1、页面端需求

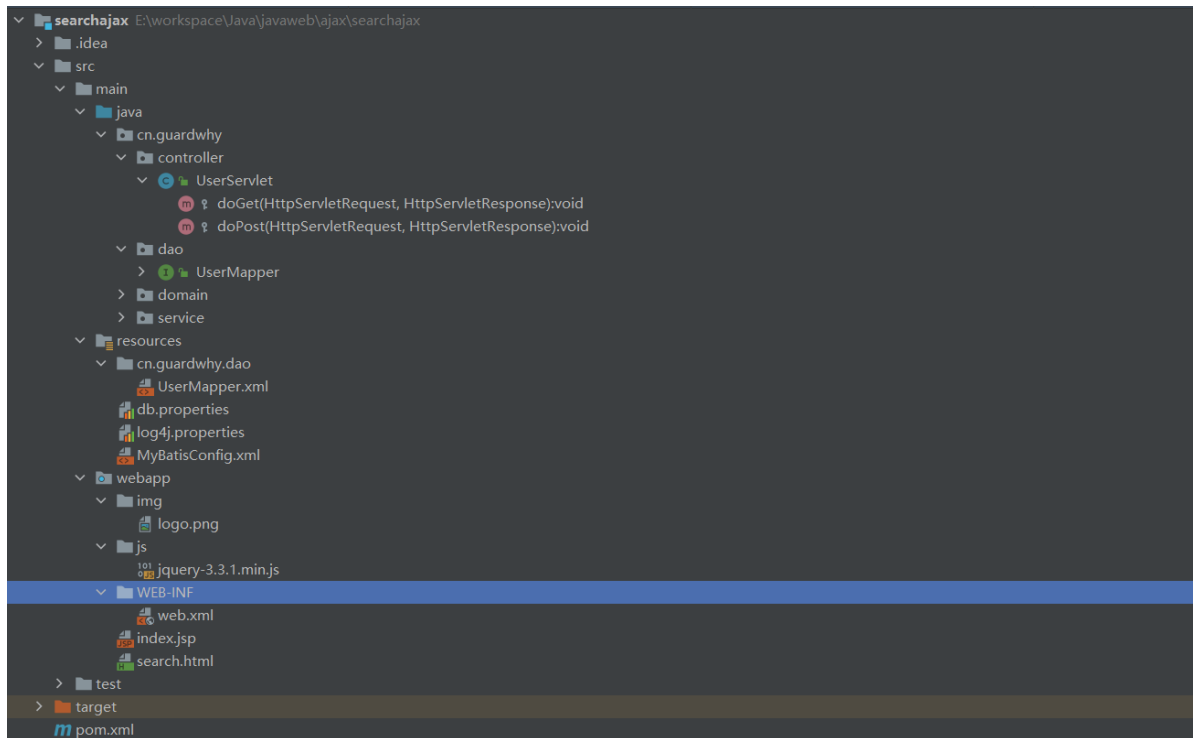
- 为用户名输入框绑定鼠标点击事件。获取输入的用户名数据。
- 判断用户名是否为空。如果为空，则将联想提示框隐藏。
- 如果不为空，则发送 `AJAX` 请求，并将响应的数据显示到联想查询搜索框。

#### 2、服务端需求

- 获取请求参数，调用业务层的模糊查询方法。
- 将返回的数据转成 `JSON`，并响应给客户端。

## 10.5.2 环境搭建

### 1、创建Maven项目，导入项目所需的依赖和重要的配置文件



相关依赖： pom.xml

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

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>cn.guardwhy</groupId>
  <artifactId>searchajax</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <!--导入相关依赖-->
  <dependencies>
    <!--Servlet-->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>4.0.1</version>
    </dependency>
    <!--jsp-->
    <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>javax.servlet.jsp-api</artifactId>
      <version>2.3.3</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-
annotations -->
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-annotations</artifactId>
```



```
<version>2.13.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-
core -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-core</artifactId>
  <version>2.13.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-
databind -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.13.0</version>
</dependency>
<!-- mybatis相关依赖-->
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>3.5.2</version>
</dependency>
<!-- mysql数据库相关依赖-->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.47</version>
</dependency>
<!-- 日志相关依赖-->
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.17</version>
</dependency>
<!-- 测试相关依赖-->
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
</dependency>
<!-- Lombok插件-->
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.16</version>
</dependency>
</dependencies>
<!--在build中配置resources，来防止我们资源导出失败的问题-->
<build>
  <resources>
    <resource>
      <directory>src/main/resources</directory>
      <includes>
        <include>**/*.properties</include>
        <include>**/*.xml</include>
      </includes>
      <filtering>true</filtering>
    </resource>
  </resources>
</build>
```

```

    <resource>
      <directory>src/main/java</directory>
      <includes>
        <include>**/*.properties</include>
        <include>**/*.xml</include>
      </includes>
      <filtering>true</filtering>
    </resource>
  </resources>
</build>
</project>

```

## 2、编写对应的数据查询语句

```

CREATE DATABASE db_ajax;

-- 使用db10数据库
USE db_ajax;

-- 创建user表
CREATE TABLE USER(
  id INT PRIMARY KEY AUTO_INCREMENT,  -- 主键id
  NAME VARCHAR(20),                  -- 姓名
  age INT,                           -- 年龄
  search_count INT                    -- 搜索数量
);

-- 插入数据
INSERT INTO USER VALUES (NULL, '张三', 23, 25), (NULL, '李四', 24, 5),
  (NULL, '王五', 25, 3), (NULL, '赵六', 26, 7), (NULL, '张三丰', 93, 20),
  (NULL, '张衡', 18, 23), (NULL, '张飞', 33, 21), (NULL, '张小斐', 65, 6);

-- 查询数据
select * from user;

-- 查询姓名中包括张的，并且按照搜索数量降低排序，搜索前四条数据
SELECT * FROM user WHERE name LIKE '%张%' ORDER BY search_count DESC LIMIT 0,4;

```

## 10.5.3 代码实现

### 1、前端页面实现：search.html

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>用户搜索</title>
  <style type="text/css">
    .content {
      width: 643px;
      margin: 100px auto;
      text-align: center;
    }

    input[type='text'] {
      width: 530px;
    }
  </style>
</head>
<body>
  <div class="content">
    <input type="text" value="请输入搜索内容" />
    <input type="button" value="搜索" />
  </div>
</body>
</html>

```

```

        height: 40px;
        font-size: 14px;
    }

    input[type='button'] {
        width: 100px;
        height: 46px;
        background: #38f;
        border: 0;
        color: #fff;
        font-size: 15px
    }

    .show {
        position: absolute;
        width: 535px;
        height: 100px;
        border: 1px solid #999;
        border-top: 0;
        display: none;
    }
</style>
</head>
<body>
<form autocomplete="off">
    <div class="content">
        
        <br/><br/>
        <input type="text" id="username">

        <input type="button" value="搜索一下">
        <!--用于搜索显示联想查询的数据-->
        <div id="show" class="show"></div>
    </div>
</form>
</body>
<script src="js/jquery-3.3.1.min.js"></script>
<script>
    // 用户输入框绑定鼠标点击事件
    $("#username").mousedown(function () {
        // 获取输入的用户名
        let username = $("#username").val();
        // 3.判断用户是否为空
        if(username == null || username == ""){
            // 4.如果为空，将联想查询搜索框隐藏
            $("#show").hide();
            return;
        }

        // 5.如果不为空发送ajax请求,并将数据显示到联想框
        $.ajax({
            // 请求路径
            url: 'userServlet',
            // 请求参数
            data: {"username":username},
            // 请求参数
            type:"POST",
            // 响应数据形式

```

```

        dataType:"json",
        // 请求成功后回调函数
        success:function (data){
            // 将返回的数据显示到div
            let names = "";
            for (let i= 0; i< data.length; i++){
                names += "<div>" + data[i].name + "</div>";
            }

            $("#show").html(names)
            $("#show").show();
        }
    });
});
</script>
</html>

```

## 2、服务端实现: `UserServlet`

```

package cn.guardwhy.controller;

import cn.guardwhy.domain.User;
import cn.guardwhy.service.UserService;
import cn.guardwhy.service.impl.UserServiceImpl;
import com.fasterxml.jackson.databind.ObjectMapper;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.List;

@WebServlet("/userServlet")
public class UserServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        //设置请求和响应的编码
        req.setCharacterEncoding("UTF-8");
        resp.setContentType("text/html;charset=UTF-8");

        // 拿到请求的参数
        String username = req.getParameter("username");
        // 调用业务层的模拟查询方法得到数据
        UserService userService = new UserServiceImpl();
        List<User> users = userService.selectLike(username);

        // 将数据转换成JSON,响应到客户端
        ObjectMapper mapper = new ObjectMapper();
        String json = mapper.writeValueAsString(users);
        resp.getWriter().write(json);
    }

    @Override

```

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
    doPost(req, resp);
}
}
```

## 10.5.4 执行结果

启动服务器，查看前端页面

