返回目录

第二章　Web应用

本章导读

Python很适合开发Web应用，而Web是目前最为重要应用形式。Web开发必须了解HTML、CSS、JavaScript，这是Web前端也就是浏览器端开发所需要的最基础技术。为了更好地操纵页面和交互，一般需要掌握一种JavaScript库，此处以著名的jQuery为例。

Python有多种Web开发框架，其中Flask已得到广泛应用。掌握Flask，也很有助于学习其他框架。

学习目标：

1. 了解HTML、CSS、JavaScript和jQuery；
2. 掌握Python Web框架Flask；
3. 掌握网页模板；
4. 掌握浏览器与服务器之间的数据交互技术；

本章目录

第一节　快速了解

浏览器是Web应用的重要角色。当在浏览器输入网址并执行，其本质是向网址对应的在互联网上的计算机发出请求，这台计算机被称之为服务器(Sever)。当服务器收到请求后，会根据请求内容作出反应，并反馈相应内容，浏览器收到相应内容后进行解读并显示其内容。图2-1是浏览器服务器工作模式简图。

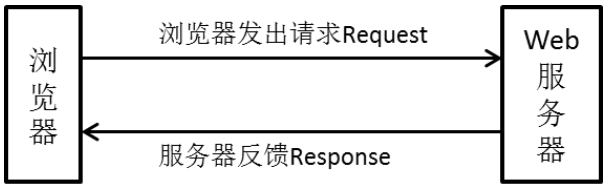


图2-1　浏览器服务器模式

1、HTML与CSS初步

例程2-1是一个简单网页的代码，假如以index.html存储，图2-2是其在浏览器中以文件打开方式浏览时的效果图，即浏览器解读代码后呈现的效果。第1行代码告知本文件的类型为html，可以认为是网页文件的默认开始代码。HTML是HyperText Markup Language(超文本标记语言)的简写，用于撰写网页。第2行代码与第12行配对，网页相关代码一般都撰写在html标记(tag)对之间，标记又称元素(element)。

网页代码大致有head和body两大部分组成。head部分用于描述网页总体属性和信息，包括网页标题等等。第4行代码用于设置网页内容的编码，建议设置为utf-8。除了utf-8外，汉语为主的网页还可以设置为gb2312等。第5行代码用于设置网页的标题。网页标题出现在浏览器标题栏，如图2-2顶部所示。head之中还可以出现其他信息，这些信息常常不会呈现，但对网页效果等由较大影响，将后续内容中予以描述。

例程2-1

```
第1行  <!doctype html>
第2行  <html lang="zh">
第3行      <head>
第4行          <meta charset="UTF-8">
第5行          <title>我的第一个网页</title>
第6行      </head>
第7行      <body>
第8行          <p style="font-size:24px;">李白</p>
第9行          <img src="img/LiBai.jpg" alt="李白画像"><br>
第10行         <a href="libai.html">更多了解</a>
第11行     </body>
第12行 </html>
```

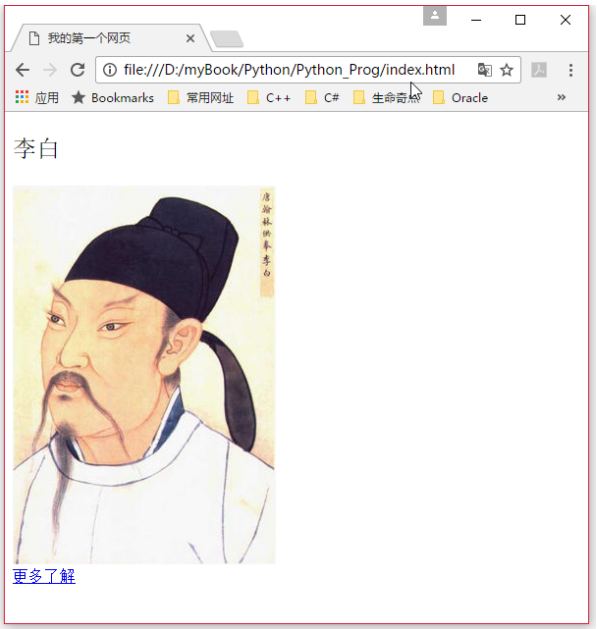


图2-2　例程2-1执行效果图

第7-11行为网页的body部分，body是网页内容显示的主体。在上例代码中，body内有三个标记，分别是p(paragraph的简写，段落)、img(image的简写，图片)、a(anchor简写，链接)和br(break简写，换行)。html语言大部分标记都是配对使用，用以

标明标记作用范围，称之为双端标记，也有少部分为单端标记，如：img、br等。双端标记结尾部分与开始部分相同，但多了正斜杠。

在第10行代码中，href是a元素的属性，其功能是定义链接被点击后访问的地址，其等号后的内容为href的属性值，属性值一般用配对的英文双引号或单引号界定。不同的元素有不同的属性，如img标记的属性src用于定义图片来源，这些属性可以可称为专用属性。在html中还有一种通用属性，几乎所有标记都可以使用，且其功能相同，如style属性用于定义标记的显示。在例程第8行中，style的功能是定义p元素中内容的字号为24px(px是pixel像素英文简写)。style是html中定义显示效果的最重要属性，将在本章后续内容中简单讲解。

编写好的网页文件，要发布到Web服务器才能被更多访问。Python提供Web服务器模块，可以用如下命令启动，启动命令执行效果如图2-3。在网页所在文件夹启动命令，则该文件夹下的网页就可以直接在浏览器中访问。

```
python -m http.server
```

```
D:\myBook\Python\Python_Prog>python -m http.server
Serving HTTP on 0.0.0.0 port 8000 ...
192.168.1.101 - - [05/Mar/2017 11:07:39] "GET / HTTP/1.1" 200 -
192.168.1.101 - - [05/Mar/2017 11:07:39] "GET /img/LiBai.jpg HTTP/1.1" 200 -
192.168.1.101 - - [05/Mar/2017 11:07:39] code 404, message File not found
192.168.1.101 - - [05/Mar/2017 11:07:39] "GET /favicon.ico HTTP/1.1" 404 -
192.168.1.101 - - [05/Mar/2017 11:08:04] "GET /1.html HTTP/1.1" 200 -
```

图2-3　python -m http.server执行效果图

图2-3第2行“Serving HTTP on 0.0.0.0 port 8000”的含义是Web服务器允许0.0.0.0IP地址访问，其中4个0表示所有能访问该计算机的计算机都可以访问。port 8000的含义是访问是要添加端口号8000。在笔者计算机的浏览器地址栏输入“http://192.168.1.101:8000”则可以访问网页。port又称端口，8000是端口号。端口是计算机与外部信息交流的通道。在计算机内部，每个端口背后都有相应程序，接受外部设备通过端口进入的请求并反馈信息。不同的服务，都有其相应默认端口，如Web服务默认端口号80。Python的http.server可以设置端口号，格式如下。在笔者计算机，输入“http://192.168.1.101”即可访问index.html文件中的内容。如果要访问其他网页文件，如index.html所在文件夹下有一个名为1.html的文件，则输入：“192.168.1.101/1.html”即可。

```
python -m http.server 80
```

在开发过程中，如果仅是访问本地计算机，127.0.0.1可以通用代表本地当前计算机，也可以用localhost代表，比如：http://127.0.0.1或http://localhost即可访问本地当前计算机，如果有端口号，同样需要加上端口号。

在Web服务中，一般默认index.html为网站入口文件，即直接输入网址或者IP地址，不输入文件名，直接默认访问index.html。一般说来，可以认为index.html为首页文件，通过该文件可以导航访问网站其他内容。

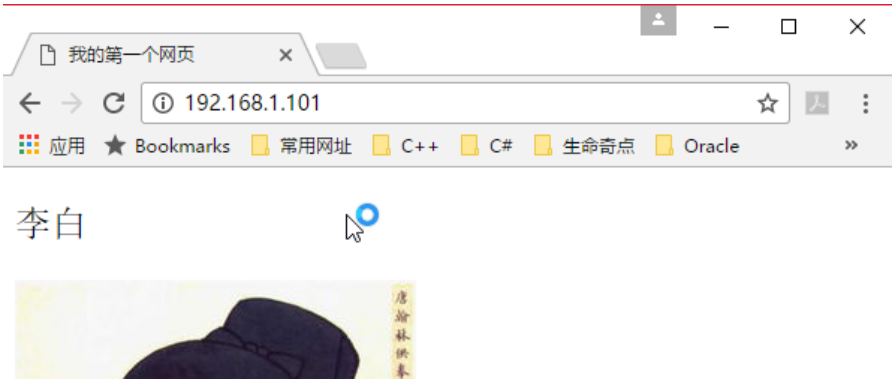


图2-4　没有端口号的网页访问

例程2-2执行效果如图2-5图左所示，唐诗宋词作者列表都是红色，且每个条目前有一个红色圆点。原点的产生与UL-LI有关，UL的含义是unorder list的首字母简写。字号大小以及颜色是例程第7行作用结果。从代码可以看出，花括号内的内容与例程2-1第8行相似，其功能是设置字体为18px颜色为red，花括号前li的含义是设置li元素为花括号内的效果。所有body及其body中的元素都可以通过类似方式设置显示效果。一般说来，将元素的显示效果与内容分离，有助于网页简洁及其更好地维护。对于一个内容很多

的网页，找到元素并修改其显示效果，是一件并不美妙过程，而将显示效果放置在head区域是更好的解决方案。可以将放置在head区域的style称之为page-style，而将在元素内的style称之为inline-style。

例程2-2

```
第1行  <!doctype html>
第2行  <html lang="zh">
第3行    <head>
第4行      <meta charset="UTF-8">
第5行      <title>唐诗宋词</title>
第6行      <style type="text/css">
第7行        li{font-size:18px;color:red}/*定义li标记的显示效果*/
第8行      </style>
第9行    </head>
第10行   <body>
第11行     <p>唐诗</p>
第12行     <ul>
第13行       <li>李白</li>
第14行       <li>杜甫</li>
第15行       <li>王维</li>
第16行       <li>白居易</li>
第17行     </ul>
第18行     <p>唐诗</p>
第19行     <ul>
第20行       <li>苏东坡</li>
第21行       <li>陆游</li>
第22行       <li>文天祥</li>
第23行     </ul>
第24行   </body>
第25行 </html>
```

图2-5　例程2-2执行效果

例程2-2是将所有li元素设置为统一显示效果，很明显图2-5图右的效果对唐诗和宋词并不一样，其实现方法如例程2-3所示，即对唐诗和宋词的li设置不同的class，然后在head之style中分别设置.Tang和.Song的显示效果。注意Tang和Song前的英文句点不能省略，用以区分名称是class还是标记。

例程2-3

```
第1行  <!doctype html>
第2行  <html lang="zh">
```

```
第3行     <head>
第4行       <meta charset="UTF-8">
第5行       <title>唐诗宋词</title>
第6行       <style type="text/css">
第7行         .Tang{font-size:18px;color:Red}/*设置class名称为Tang的元素显示效果*/
第8行         .Song{font-size:24px;color:green}/*设置class名称为Song的元素显示效果*/
第9行       </style>
第10行    </head>
第11行    <body>
第12行      <p>唐诗</p>
第13行      <ul>
第14行        <li class="Tang">李白</li>
第15行        <li class="Tang">杜甫</li>
第16行        <li class="Tang">王维</li>
第17行        <li class="Tang">白居易</li>
第18行      </ul>
第19行      <p>宋词</p>
第20行      <ul>
第21行        <li class="Song">苏东坡</li>
第22行        <li class="Song">陆游</li>
第23行        <li class="Song">文天祥</li>
第24行      </ul>
第25行    </body>
第26行  </html>
```

观察例程2-4第7-8行以及第12、20行。id是HTML标记的通用属性，几乎每个标记都支持该属性，相当于给元素另外一个唯一名称。在head的style中，#Tang li{font-size:18px;color:Red}的含义是设置id为Tang所代表元素的下级标记为li的显示效果。如果省略li则是设置Tang所在标记此例成为ul标记的显示效果。

例程2-4

```
第1行   <!doctype html>
第2行   <html lang="zh">
第3行     <head>
第4行       <meta charset="UTF-8">
第5行       <title>唐诗宋词</title>
第6行       <style type="text/css">
第7行         #Tang li{font-size:18px;color:Red}/*设置id 为Tang之下的li元素显示效果*/
第8行         #Song li{font-size:24px;color:green}/*设置id 为Song之下的li元素显示效果*/
第9行       </style>
第10行    </head>
第11行    <body>
第12行      <p>唐诗</p>
第13行      <ul id="Tang">
第14行        <li>李白</li>
第15行        <li>杜甫</li>
第16行        <li>王维</li>
第17行
```

```
            <li>白居易</li>
        </ul>
        <p>宋词</p>
        <ul id="Song">
            <li>苏东坡</li>
            <li>陆游</li>
            <li>文天祥</li>
        </ul>
    </body>
</html>
```

2、JavaScript初步

JavaScript是运行在浏览器的最主要程序设计语言，浏览器之于Web体系，其重要性不言而喻，因此JavaScript之于网页其重要性也不言而喻。可以说没有JavaScript就没有精彩的网页。和Python一样，JavaScript同为最流行的十大程序设计语言，其余8种语言分别是Java、C、C++、C#、Visual Basic、PHP、Dephi/Object Pascal以及Swift(据2017年3月TIOBE数据)。

在网页源码中，JavaScript代码撰写在script元素之间，如例程2-5第8-27行所示。第9行为注释，以双反斜杠引起，其使用用法和功能如同Python的井字符号(#)。第10行prompt()如同Python的input()函数，用于输入。在JavaScript中，prompt()函数输入时将弹出输入对话框(如图2-6所示)。在本例中，输入值将存入变量n。在JavaScript中，申明新变量时，变量前可以有var关键字，也可以省略。如省略，就如同Python。第11行代码parseInt()函数的功能是将变量n转换为整数，并将其结果存入变量N之中。

例程第12-16行是JavaScript的if选择语句，其含义是如果isPrime(N)的结果为true则执行13行否则执行第15行。和Python语言相比，if语句的控制范围由一对花括号界定，而Python则是由缩进层次界定，这是很大的不同。相比而言，JavaScript方式更加通用，在C/C++、JavaScript等语言中都是如此。另外，true(真)和false(假)在JavaScript中全部是小写字母，而在Python则是首字母大写，即True和False。另外，alert()函数是弹出对话框显示其函数参数值，与Python的print()函数功能相似。

第19-26行是JavaScript的函数定义，使用关键字function表明函数定义而Python则为def。如同流程控制语句用花括号界定范围，在函数定义中同样用花括号界定范围。第22-24行为for循环语句，这和Python有很大不同，其含义是i为变量，刚进入循环是其值为2，然后判断其值是否小于N，如果满足小于N的条件则执行循环体此处为第23行。循环体执行完毕后执行i++。i++的含义为i自增1，相当于Python的i+=1。JavaScript除了自增外还有自减(运算符为--)，不过Python没有自增和自减而是以i+=1或i-=1实现。

JavaScript的自定义函数isPrime()函数与Python的自定义函数isPrime()函数功能相同语法相似，或者说算法相同语法相似。从某种意义上讲是算法相同语法因语言不同而或多或少不同。

例程2-5

```
<!doctype html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>JavaScript快速了解</title>
    </head>
    <body>
        <script type="text/javascript">
            //判断输入数是否为质数
            var n=prompt("请输入一个数",97);
            var N=parseInt(n); //转换为正整数
            if(isPrime(N)==true){
                alert(N+"是一个质数!");
            }else{
                alert(N+"不是一个质数!");
            }
```

```
第17行
第18行          //定义名为isPrime()的函数以判断是否为质数
第19行          function isPrime(N){
第20行             if (N<=1)return false;
第21行             if(N==2)return true;
第22行             for (var i=2;i<N;i++){
第23行                if (N%i==0)return false;
第24行             }
第25行             return true;
第26行          }
第27行       </script>
第28行    </body>
第29行 </html>
```

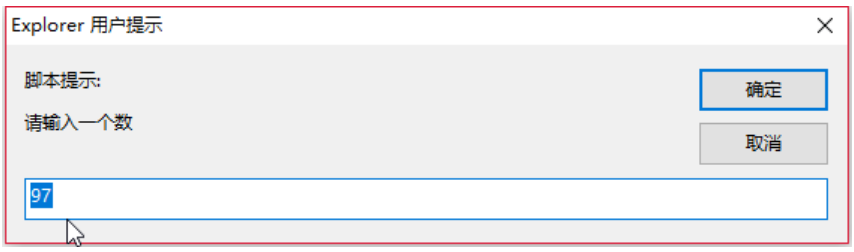


图2-6　JavaScript之prompt()函数输入对话框

例程2-6第9行申明studScore为数组(此处为空)，用于存储学生成绩，与Python类似。confirm()函数与alert()、prompt()函数一样为JavaScript内置函数，用于显示带有“确定”和“取消”两个命令按钮的对话框。当单击“确定”时返回值为true，否则为false。第14行的while是循环的另外一种形式，当满足条件时即循环，此处为state==true时循环。如果删除第19行，则会死循环或一次都不循环(如第13行的值为false)，因为while循环体中没有state的值其值永远保持state保持不变。将while(state==true)修改为while(state=true)在Python中是错误，在JavaScript中则将死循环，其原因是state=true为赋值语句，state的值即为表达式的值。第16行studScore.push(score)代码中的push()函数的功能是向数组尾部追加新数据。

例程第24行是JavaScript的for-in循环，和Python的for-in循环不同，JavaScript的for-in循环中的i不代表studScore中的一个值，而是序号或者下标值，因此要通过studScore[i]才能访问其值。

在JavaScript循环中，同样有break和continue，其语义与Python语言相通，也与C/C++、Java等相通。和Python相比，JavaScript循环没有else子句。

例程2-6

```
第1行  <!doctype html>
第2行  <html lang="en">
第3行     <head>
第4行        <meta charset="UTF-8">
第5行        <title>JavaScript-列出高于平均成绩的分数</title>
第6行     </head>
第7行     <body>
第8行        <script type="text/javascript">
第9行           var studScore=[];//JavaScript数组，类似Python之list
第10行          //总成绩初始值为0，学生人数初始值为0
第11行          var totalScore=0,studCount=0;
第12行
第13行          var state=confirm("开始录入成绩？");
第14行          while(state==true){
```

```
第15行              var score=parseInt(prompt("请录入成绩：",""));
第16行              studScore.push(score);
第17行              totalScore+=score;
第18行              studCount++;
第19行              state=confirm("继续录入成绩？");
第20行          }
第21行
第22行          //计算出平均成绩
第23行          var avgScore=totalScore/studCount;
第24行          for(var i in studScore){
第25行              if(studScore[i]>avgScore)
第26行                  alert(studScore[i]);
第27行          }
第28行          //上述代码存在漏洞，此处仅为示例
第29行      </script>
第30行  </body>
第31行 </html>
```

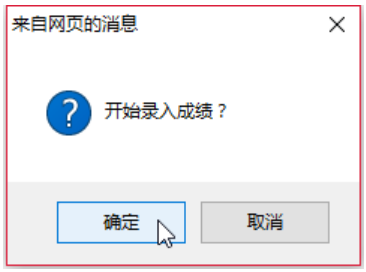


图2-7　JavaScript之confirm()函数对话框

例程2-7是数组的更多示例，参考注释将能了解数组对象各种方法的使用。JavaScript的类似字典功能可以用数组实现，如例程第26-32行所示。该字典的顺序稳定，即数据存储顺序与输入顺序相同。

第34行类似Python语言的字典，实际是JavaScript语言的对象，其中X和Y是point对象的两个属性。每个属性和属性值构成一对称之为键值(key/value)，属性名称与属性值之间用冒号间隔，属性与属性之间用逗号分隔。这在其后jQuery中常用。

例程2-7

```
第1行  <!doctype html>
第2行  <html lang="en">
第3行      <head>
第4行          <meta charset="UTF-8">
第5行          <title>JavaScript-数组示例</title>
第6行      </head>
第7行      <body>
第8行          <script type="text/javascript">
第9行              var myArr=[12,23,45,65,11,9,19,98,35];
第10行             myArr[0]=122;//修改编号为0也就是第一个元素的元素值
第11行
第12行             var arrCount=myArr.length;//Array.length获得数组的长度
第13行             myArr.length=5;//缩短数组长度，如果原数组长则剪短否则正常；
第14行             //新成员的值为undefined(未定义)
第15行
```

```
第16行          myArr.pop();//删除最后一个元素；
第17行          myArr.shift();//删除第一个元素；
第18行
第19行          myArr.unshift(12,11,54);//在数组头部增加一个或多个元素，此处为3个
第20行
第21行          myArr.slice(2,5);//从数组中截取片段，此处从编号2开始截取到编号5，但不包含5.
第22行
第23行          myArr.sort();
第24行          alert(myArr);//显示为：11,12,23,45,54,65
第25行
第26行          var fruitArr=new Array();//申明fruitArr为数组，与var fruitArr=[]效果相同
第27行          fruitArr["Apple"]="苹果";
第28行          fruitArr["Peach"]="桃";
第29行          fruitArr["Lemon"]="柠檬";
第30行          for(var fruit in fruitArr){
第31行              alert(fruit+"->"+fruitArr[fruit])
第32行          }//显示：Apple->苹果,Peach->桃,Lemon->柠檬
第33行
第34行          var point={X:100,Y:210};//生成一个名为Point的对象
第35行          alert("坐标：X="+point.X+" Y="+point.Y);
第36行          //显示：坐标：X=100 Y=210
第37行      </script>
第38行  </body>
第39行 </html>
```

在很多程序语言中，字符串处理都是很重要的组成部分，JavaScript也不例外，例程2-8是字符串应用示例，请通过注释了解相关其相关功能。

例程2-8

```
第1行  <!doctype html>
第2行  <html lang="en">
第3行      <head>
第4行          <meta charset="UTF-8">
第5行          <title>JavaScript-字符串应用示例</title>
第6行      </head>
第7行      <body>
第8行          <script type="text/javascript">
第9行              var numA=parseInt(prompt("请输入一个整数",100));
第10行             alert(convert2CN(numA));//将整数转换成汉字，如123转换壹贰叁
第11行             function convert2CN(N){
第12行                 var strN=String(N);//将数值转换为字符串
第13行                 var strLen=strN.length;//求字符串长度，字符串字符组成个数
第14行                 result="";
第15行                 strHZ="零壹贰叁肆伍陆柒捌玖";
第16行                 for(var i=0;i<strLen;i++){
第17行                     result+=strHZ.charAt(parseInt(strN.charAt(i)));
第18行
```

```
            }//String.charAt(n)取得编号为n处字符
第19行          return result;
第20行      }
第21行      //更多字符串应用示例
第22行      var myFileName="c:/abc/xyz/123.jpg";
第23行      alert(myFileName.indexOf("."));//取得句点所在位置，输出14
第24行      alert(myFileName.lastIndexOf("/"))//取得/最后一次出现位置，输出10
第25行      alert(myFileName.split("/"));//转换数组，["c:","abc","xyz","123.jpg"]
第26行      alert(myFileName.substring(myFileName.lastIndexOf("/")+1));
第27行      //取得myFileName文件名，输出123.jpg
第28行      var strA="美丽的祖国美丽的人民";
第29行      alert(strA.replace("美丽","可爱"));//可爱的祖国美丽的人民，只进行一次替换
第30行      alert(strA.substring(3,7));//输出：祖国美丽
第31行    </script>
第32行  </body>
第33行 </html>
```

例程2-9主要是JavaScript按时间周期重复执行函数的使用。第18行代码的含义是每隔2000毫秒执行alert(checkRandNumPrime())，其setInterval()函数的返回值存入变量timeClock，该变量可以用于clearInterval()，其功能是清除setInterval()函数设置的周期执行。与setInterval()相似的是setTimeout()，该函数功能是等到设定时间执行一次。但在第29行代码中，当等到5000毫秒将再次调用execOne()，因此将再次被执行，效果类似setInterval()，即可以用setTimeout()模拟setInterval()函数的执行。在例程中，execCount和timeClock变量都是全局变量，可以函数内部读取或者星期其值。Math.random()的功能是产生0~1之间的随机小数，其中Math是JavaScript的内置对象。除Math外，常用内置对象还有Date(用于日期操作)、RegExp(用于正则表达式)等。已讲解Array和String其本质也是JavaScript内置对象。

例程2-9

```
第1行  <!doctype html>
第2行  <html lang="en">
第3行    <head>
第4行      <meta charset="UTF-8">
第5行      <title>JavaScript-按时间重复执行</title>
第6行    </head>
第7行    <body>
第8行      <script type="text/javascript">
第9行        function checkRandNumPrime(){
第10行         var randNum=parseInt(Math.random()*1000)%50+250;
第11行         //产生250-300间的随机整数
第12行
第13行         for(var i=2;i<randNum;i++){
第14行           if (randNum%i==0)    return randNum+"不是质数";
第15行         }
第16行         return randNum+"是质数";
第17行       }
第18行       var timeClock=setInterval("alert(checkRandNumPrime())",2000);
第19行       //每2000毫秒重复执行第1个参数
第20行
第21行       var execCount=0;
```

```
第22行
第23行            function execOne(){
第24行                execCount++;
第25行                if(execCount==5){
第26行                    clearInterval(timeClock);//清除已设置的setInterval()
第27行                    alert("不再重复执行！");
第28行                }else{
第29行                    setTimeout("execOne()",5000);//到5000毫秒时执行一次execOne()
第30行                }
第31行            }
第32行
第33行            execOne();//调用函数执行
第34行        </script>
第35行    </body>
第36行</html>
```

JavaScript语言相对简单，但一旦和网页元素结合，将产生万千变化。

3、jQuery初步

网页内容繁多，jQuery能方便地操控网页方方面面，包括：元素内容、显示效果、事件处理、动画效果以及数据交互等等。jQuery与JavaScript、HTML+CSS搭档，可谓如虎添翼。

例程2-10

```
第1行 <!doctype html>
第2行 <html lang="en">
第3行     <head>
第4行         <meta charset="UTF-8">
第5行         <title>jQuery-入门</title>
第6行         <script type="text/javascript" src="jquery-3.1.1.min.js"></script>
第7行     </head>
第8行     <body>
第9行         <div id="Test">Welcome to jQuery World!</div>
第10行        <script type="text/javascript">
第11行            alert("下一条语句执行后，页面背景将变成红色。");
第12行            $("body").css("background-color","red");//网页背景变为红色
第13行            $("#Test").html("欢迎来到jQuery世界！");//id为Test的元素内容被更改
第14行        </script>
第15行    </body>
第16行</html>
```

网页能使用jQuery功能的前提是引入jQuery框架，如例程2-10第6行代码所示。jQuery框架是开源系统，其特征是文件容量小运行速度快浏览器兼容好，该框架可从http://www.jquery.com下载。下载后无需安装，将之放到网页所在或其下的文件夹即可。第6行代码表明jQuery和使用jQuery的网页在同一个文件夹，如写成<script type="text/javascript" src="js/jquery-3.1.1.min.js"></script>则表明jQuery框架位于网页所在文件夹下的js文件夹内。另外，jQuery也可以无需下载直接引用，如新浪网等CDN(Content Delivery Network简写，内容分发网络)即提供jQuery链接，此时可将src改为src="http://lib.sinaapp.com/js/jquery/3.1.0/jquery-3.1.0.min.js"，其效果完全相同。jquery-3.1.1.min.js中的3.1.1是其版本号，min表明是压缩后的文件。

当script元素使用src属性时，写在标记对之间的代码不起作用。第12-13行是jQuery代码的使用。jQuery代码也是JavaScript代码，因此也遵守JavaScript规范，如大小写敏感等等。

第12-13行代码中的$("body")和$("#Test")表示选中元素，其中$("body")表示选中HTML元素，所有的HTML元素都可以如此选中。$("#Test")表示选中id名称为Test的元素，在本例中是一个div元素。jQuery还可以通过class选择元素，如$(".ABC")表示选中class名为ABC的元素。当元素选中后，将返回一个jQuery对象，这个对象有一系列函数或者说方法，如例程中的css()、html()就是jQuery对象所拥有的函数。css()用于设置选中元素的css也就是style属性，html()用于修改选中元素的内容。

如下列，当鼠标指向唐诗宋词作者时发生显示效果变化，让用户感知焦点所在，是一种常见设计，如图2-8所示，其实现代码如例程2-11所示。在Web应用中，很多动态显示效果常用JavaScript实现，一些基于JavaScript的框架更是有助于实现各种效果，如jQuery等，图2-8的效果就是用jQuery实现。

图2-8　例程2-11鼠标移动效果

例程2-11

```
第1行  <!doctype html>
第2行  <html lang="zh">
第3行     <head>
第4行        <meta charset="UTF-8">
第5行        <title>唐诗宋词</title>
第6行        <script type="text/javascript" src="jquery-3.1.1.min.js"></script>
第7行        <style type="text/css">
第8行           #Tang li{font-size:18px;color:Red}/*设置id 为Tang之下的li元素显示效果*/
第9行           #Song li{font-size:24px;color:green}/*设置id 为Song之下的li元素显示效果*/
第10行          .standOut{font-weight:bold;}
第11行       </style>
第12行    </head>
第13行    <body>
第14行       <p>唐诗</p>
第15行       <ul id="Tang">
第16行          <li>李白</li>
第17行          <li>杜甫</li>
第18行          <li>王维</li>
第19行          <li>白居易</li>
第20行       </ul>
第21行       <p>宋词</p>
第22行       <ul id="Song">
第23行          <li>苏东坡</li>
第24行          <li>陆游</li>
第25行          <li>文天祥</li>
第26行       </ul>
第27行       <script type="text/javascript">
```

```
第28行            $("li").mouseover(function(){
第29行                $(this).addClass("standOut");//对鼠标移动的标记增加名为standOut的class
第30行            });
第31行            $("li").mouseout(function(){
第32行                $(this).removeClass("standOut");//鼠标移出是删除名为standOut的class
第33行            });
第34行        </script>
第35行    </body>
第36行 </html>
```

第28-30、31-33行代码结构如同$("li").mouseover(function(){})，其中$("li")的含义是选中当前网页中的所有li元素，mouseover是事件，类似事件还有很多，如mouseout(鼠标移出)、click(点击事件)、keypress(键盘按键)等等，function(){}表示将要做的事情。上述几个部分联络在一起的含义是当在li元素上发生mouseover事件时执行function(){}内的代码。由于括号较多，撰写代码要特别注意，建议先写$("li").mouseover()部分，然后在mouseover后的圆括号内写function(){}，然后在花括号之间位置回车分行，将需要执行代码写在下一行即可，这样不至于忘了括号配对。

第29、32行的$(this)的功能是选中事件发生时元素，第29行是选中触发mouseover事件的元素，第32行则是触发mouseout事件的元素。注意，this两侧不能家引号，否则将选中名为this的元素，而该元素并不存在，将不能达到预期目标。

第29行的addClass()的功能是向选中元素增加class。第32行的removeClass()则是移走class。由于class可以定义显示效果，因此显示效果将事件或增加/移走class而呈现不同效果。注意：一个元素可以有多个class。

图2-9是网页常见效果，在有限版面内显示多幅图片，通过点击底部圆点进行图片切换(此处假定6幅图片的编号依次为1.jpg、2.jpg、……、6.jpg)，当然实际应用中还会有定时切换等等。在HTML中，图片显示用img元素，确定具体显示图片用src属性，只要改变src属性的值，即可变化图片。在jQuery中，可以用$("img").attr("src","图片位置")实现，其中，attr(属性名称,属性值)用于设置选中元素指定属性的值。当attr()只有一个参数，即如attr(属性名称)时，则不能设置属性之值，用于读取属性值。例程2-12是图2-9实现代码。

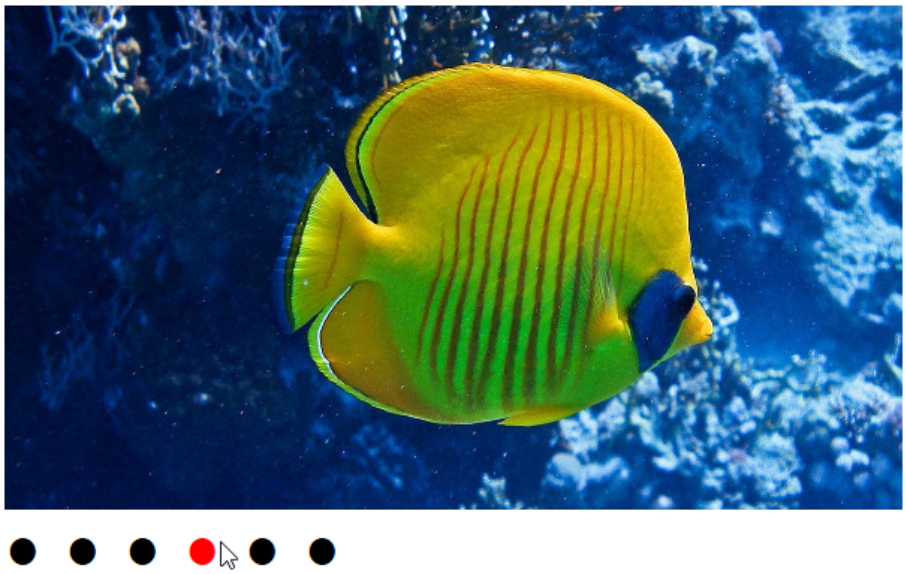


图2-9　图片切换效果图

例程第14行使用了元素span，其默认功能是不换行显示元素。如果要控制网页中的某个组成，必须受到一个或多个元素控制，然后设定该元素相关属性。第8行用于设定span元素的显示参数，其中的display用于设定显示模式，取值有block|inline|inline-block，inline是默认取值。当display取值为block时，该元素前后将自动换行，可以设定元素的宽度和高度；当取值inline时，前后不换行，不能设置元素宽度和高度；当取值inline-block时，前后不换行，但能设置元素的宽度和高度。

例程2-12

```
第1行 <!doctype html>
第2行 <html lang="zh">
第3行     <head>
第4行         <meta charset="UTF-8">
```

```
第5行         <title>图片切换</title>
第6行         <script type="text/javascript" src="jquery-3.1.1.min.js"></script>
第7行         <style type="text/css">
第8行            span{display:inline-block;width:40px;font-size:40px}/*定义span显示效果*/
第9行            .selectedDot{color:red}/*定义被选中圆点的显示效果*/
第10行        </style>
第11行     </head>
第12行     <body>
第13行        <img src="4.jpg" width="600px"><br>
第14行        <span No="1">●</span><span No="2">●</span><span No="3">●</span><span No="4" class="sel
    ectedDot">●</span><span No="5">●</span><span No="6">●</span>
第15行        <script type="text/javascript">
第16行           $("span").click(function(){
第17行              imgNo=$(this).attr("No");//imgNo是变量
第18行              $("img").attr("src",imgNo+".jpg");
第19行              $(".selectedDot").removeClass("selectedDot");//移走选中圆点的selectedDot
第20行              $(this).addClass("selectedDot");//当前点击圆点增加selectedDot
第21行           });
第22行        </script>
第23行     </body>
第24行  </html>
```

第17行代码的含义是将$(this).attr("No")的结果放到imgNo之中，imgNo称之为变量，相当于名称imgNo中存放了$(this).attr("No")的值。变量名称中只能有字母、数字、美元符号($)和下划线，但首字母不能是数字。另外，变量命名要“名副其实”，即名称反映其功能。将例程中的imgNo全部更换为a或者其他，其执行效果不会发生任何变化，但是imgNo更能代表其含义，看其名知其意，有利于代码撰写者事后了解其作用，更也有利于其他程序相关人如答疑者、测试人员等。另外，变量命名最好有良好的可读性，如imgno、IMGNO显然没有imgNo更一目了然。每个意义单元首字母都大写称之为帕斯卡命名法，如ImgNo；imgNo是骆驼命名法，和帕斯卡命名法相比，第一个意义单元全部小写。具体帕斯卡命名法还是骆驼命名法，可根据个人习惯，如是团体开发，则遵守团体约定。

第19行代码中的$(".selectedDot")含义是选中class名为selectedDot的元素，即按class选择元素时，需要在class名称前增加一个英文句点。在jQuery中，除了能按元素名称，class名称选择外，还可以按id选择，此时需要在id名称前增加#。对于上例，假如img元素的id值为bigImg，则选中该img元素的jQuery语句则是$("#bigImg")，即将第18行的$("img")修改为$("#bigImg")。通常一个网页中会有多个img元素，用id或者class能更有效区分。

例程2-13是对例程2-12的改进，增加了图片定时切换。在本例中，每隔3000毫秒(3秒)换一幅图，同时图底部的红色圆点也相应而动。第22行功能是每隔3000毫秒执行changeImg()函数。在该函数中，第24行取得当前显示图片的编号，即第几个圆点被选中，在此基础上计算出下一个图片编号，如果图片编号大于6，则从头开始，即从第1幅图开始。下面两行代码与上例相同。第31行代码圆点前的部分稍难理解。假定nowImgNo为2，则对应代码相当于$("span[No='2']")，其含义是选中No属性值为2的span元素。jQuery为了能更方便操控网页，有多种元素选择方式，请参考本章jQuery进阶。由于2需要变化，因此用变量代替，形成字符串拼接，即2之前为一个字符串，2之后为一个字符串，字符串必须用引号界定并用字符串运算符连接，因此变为$("span[No='"+2+"']")，将2换为变量即可。注意：2之前的双引号是与span前的双引号配对，2之后的双引号是最后一个双引号配对。

例程2-13

```
第1行  <!doctype html>
第2行  <html lang="zh">
第3行     <head>
第4行        <meta charset="UTF-8">
第5行        <title>图片切换</title>
第6行
```

```
        <script type="text/javascript" src="jquery-3.1.1.min.js"></script>
第7行   <style type="text/css">
第8行     span{display:inline-block;width:40px;font-size:40px}/*定义span显示效果*/
第9行     .selectedDot{color:red}/*定义被选中圆点的显示效果*/
第10行  </style>
第11行 </head>
第12行 <body>
第13行   <img src="1.jpg" width="600px"><br>
第14行   <span No="1">●</span><span No="2">●</span><span No="3">●</span><span No="4" class="sel
ectedDot">●</span><span No="5">●</span><span No="6">●</span>
第15行   <script type="text/javascript">
第16行     $("span").click(function(){
第17行       imgNo=$(this).attr("No");//imgNo是变量
第18行       $("img").attr("src",imgNo+".jpg");
第19行       $(".selectedDot").removeClass("selectedDot");//移走选中圆点的selectedDot
第20行       $(this).addClass("selectedDot");//当前点击圆点增加selectedDot
第21行     });
第22行     var cycleImg=setInterval("changeImg()",3000);//每隔3000毫秒换图
第23行     function changeImg(){
第24行       var nowImgNo=$(".selectedDot").attr("No");//取得当前的图片编号
第25行       nowImgNo++;//下一个编号
第26行       if(nowImgNo>6){//图片编号大于6时，回到第1幅图
第27行         nowImgNo=1;
第28行       }
第29行       $("img").attr("src",nowImgNo+".jpg");
第30行       $(".selectedDot").removeClass("selectedDot");
第31行       $("span[No='"+nowImgNo+"']").addClass("selectedDot");//选择当前编号对应的span元素
第32行     }
第33行     $("img").mouseover(function(){
第34行       clearInterval(cycleImg);//清除定时换图变量
第35行     });
第36行     $("img").mouseout(function(){
第37行       cycleImg=setInterval("changeImg()",3000);//注意cycleImg前不能有var
第38行     })
第39行   </script>
第40行   </body>
第41行 </html>
```

第33-35行代码的功能是当鼠标放在图片上时，不切换图片，以便于用户进一步操作，其功能通过第34行代码实现。如果第22行不将setInterval()函数执行结果存入变量，第34行将无法清除setInterval()的设置。第36-38行则是当鼠标移出图片时，继续开始定时切换图片。注意：cycleImg前不能有var，如果有则cycleImg是函数内的局部变量，当mouseover时，两个函数之间没法直接访问。如果没有则是全局变量，都是对第22行变量cycleImg的读写。

4、Flask初步

上述例程都只是静态页面，如何才能将服务器程序执行结果输出到浏览器呢？如果能，则将架起浏览器与Web服务器之间的桥梁，同时其输出不再是黑乎乎窗口。例程2-14是熟悉的Python代码，其函数isPrime(N)用于判断一个数N是否质数，如果是返回

True，否则返回False，第8-10行用于输出200-300之间的质数。如何将第8-10行的执行输出到浏览器，例程2-15是其实现，图2-10是其执行效果。

例程2-14

```
第1行 def isPrime(N):
第2行     if N<=1:return False
第3行     if N==2:return True
第4行     for i in range(2,N):
第5行         if N%i==0:return False
第6行     return True
第7行
第8行 for i in range(200,300):
第9行     if isPrime(i)==True:
第10行        print(i)
第11行
第12行 #eof
```

相比例程2-14，例程2-15多了一些代码。第8行的含义是引入Flask功能扩展库。Flask是第三方库，不是Python标准库，因此需要安装，其指令是**pip install Flask**，这是Flask能被使用的前提。Flask是基于Python的轻量级Web应用框架，当前较为流行，很适合Web开发。

第9行代码的功能是生成一个名为app的Flask对象，第19行app.run()则是调用app对象的run()函数或方法，一旦代码正确，其在命令行执行对应指令(如图2-10上所示)，则该程序即成为一个服务，静待用户从浏览器访问。图2-10下即是浏览器输入地址执行的结果，此时Python程序执行结果能通过浏览器反馈。

例程2-15

```
第1行 def isPrime(N):
第2行     if N<=1:return False
第3行     if N==2:return True
第4行     for i in range(2,N):
第5行         if N%i==0:return False
第6行     return True
第7行
第8行 from flask import Flask #新增代码。装入Flask
第9行 app=Flask(__name__) #新增代码
第10行
第11行 @app.route("/") #新增代码，对应执行root()函数
第12行 def root():
第13行     content=""
第14行     for i in range(200,300):
第15行         if isPrime(i)==True:content+=str(i)+" "
第16行     return content
第17行
第18行 if __name__=="__main__": #新增代码
第19行     app.run()
第20行
第21行 #eof
```

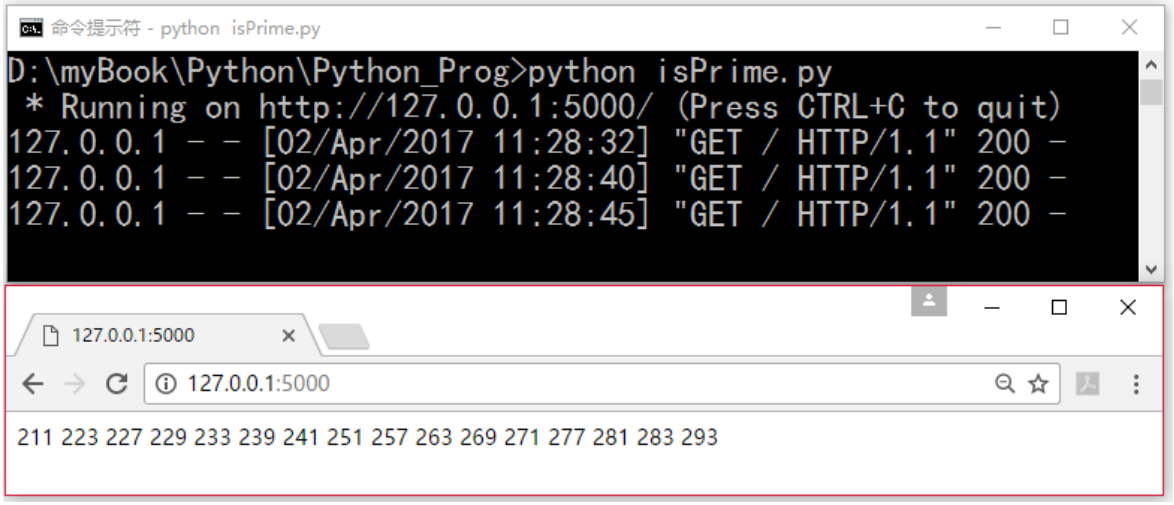


图2-10　例程2-15执行效果

观察图2-10上会发现有“http://127.0.0.1:5000/”，其含义是访问该服务在浏览器地址栏输入http://127.0.0.1:5000/即可，其中127.0.0.1是IP地址，代表本机，也可用localhost替代，5000表示端口号，即该服务在5000端口。另外，如果知道本机真实IP地址，也可以用于替代127.0.0.1，效果相同。如果要终端该程序的执行，可以按CTRL+C或CTRL+Break。

第19行的app.run()可以含有多个参数，常见的用法是app.run(host='0.0.0.0', port=80,debug=True)，其中host用于指定IP地址，如果指定为0.0.0.0则表示所有地址均可访问。port用于指定端口号，如果指定80则为Web默认端口号，输入地址时可以省略端口，其他则必须同时输入端口号。如果debug等于True，则如果运行时修改代码，Flask将及时刷新，极有利于调试。

例程第11行代码的@app.route("/")用于规划路由表及其对应执行函数，此处/代表网站起点，即仅输入网址时，执行对应的函数，此处为root()函数，该函数由用户编写并命名。例程2-16是多路由示例，当在地址栏输入http://localhost/checkPrime/200时，将判断200是不是质数，并在浏览器窗口输出。例程2-16的其他变化，请对照上一段内容理解。

例程2-16

```
第1行  def isPrime(N):
第2行      if N<=1:return False
第3行      if N==2:return True
第4行      for i in range(2,N):
第5行          if N%i==0:return False
第6行      return True
第7行
第8行  from flask import Flask #新增代码。装入Flask
第9行  app=Flask(__name__) #新增代码
第10行
第11行 @app.route("/") #新增代码，对应执行root()函数
第12行 def root(): #另外一种实现方式
第13行     primeList=[N for N in range(200,300) if isPrime(N)]
第14行     return str(primeList) #需要返回字符串
第15行
第16行 @app.route("/checkPrime/200") #新增代码，对应执行root()函数
第17行 def checkPrime():
第18行     if isPrime(200)==True:
第19行         return "200是质数"
第20行     else:
第21行         return "200是合数"
第22行
第23行
```

```
if __name__=="__main__":
    app.run(host="0.0.0.0",port=80,debug=True)

#eof
```

如何要判断更多整数是否质数？很明显，疯狂重复第16-21行不是程序设计解决方案，将其改为如例程2-17所示，此时Num可代表任意正整数。注意：第2行checkPrime()的参数名称要与@app.route()中带尖括号界定的名称保持一致。另外，第1行代码@app.route("/checkPrime/<Num>")可以修改为@app.route("/checkPrime/<int:Num>")，int称之为转换器(converter)，此时第3行没必要存在，其功能是将Num设置为整数类型，常用的还有float(带小数点数据)。如果省略converter，则默认为字符串类型。

例程2-17

```
@app.route("/checkPrime/<Num>")
def checkPrime(Num):
    Num=int(Num) #转换为整数
    if isPrime(Num)==True:
        return str(Num)+"是质数"
    else:
        return str(Num)+"是合数"
```

例程2-18是前述几个例程的整合，注意第11-12行，其含义是当输入地址如127.0.0.1或127.0.0.1/200-300都执行root()函数。

例程2-18

```
def isPrime(N):
    if N<=1:return False
    if N==2:return True
    for i in range(2,N):
        if N%i==0:return False
    return True

from flask import Flask
app=Flask(__name__)

@app.route("/")
@app.route("/<first_second>")
def root(first_second="200-300"): #另外一种实现方式
    firstNum=int(first_second.split("-")[0]) #转换为整数
    secondNum=int(first_second.split("-")[1]) #转换为整数
    primeList=[N for N in range(firstNum,secondNum) if isPrime(N)]
    return str(primeList) #需要返回字符串

@app.route("/checkPrime/<int:Num>")
def checkPrime(Num):
    if isPrime(Num)==True:
        return str(Num)+"是质数"
    else:
        return str(Num)+"是合数"

```

```
第26行 if __name__=="__main__":
第27行     app.run(host="0.0.0.0",port=80,debug=True)
第28行
第29行 #eof
```

例程2-19执行效果如图2-11所示，所列质数为1-100之间的质数。从代码可以看出，content字符串内容固定，不能反映变量primeList，例程2-20是例程2-19的简单修改，此时能反映primeList的变化。

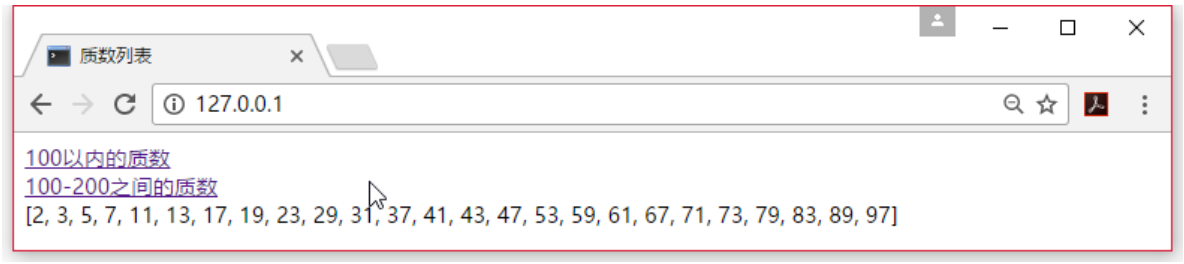


图2-11　例程2-19执行效果图

例程2-19

```
第1行  def isPrime(N):
第2行      if N<=1:return False
第3行      if N==2:return True
第4行      for i in range(2,N):
第5行          if N%i==0:return False
第6行      return True
第7行
第8行  from flask import Flask #新增代码。装入Flask
第9行  app=Flask(__name__) #新增代码
第10行
第11行 @app.route("/") #新增代码，对应执行root()函数
第12行 @app.route("/<first_second>")
第13行 def root(first_second="200-300"): #另外一种实现方式
第14行     firstNum=int(first_second.split("-")[0]) #转换为整数
第15行     secondNum=int(first_second.split("-")[1]) #转换为整数
第16行     primeList=[N for N in range(firstNum,secondNum) if isPrime(N)]
第17行
第18行     #产生HTML源代码
第19行     content="""<!doctype html>
第20行     <html lang="zh">
第21行         <head>
第22行             <meta charset="utf-8">
第23行             <title>质数列表</title>
第24行         </head>
第25行         <body>
第26行             <a href="/1-100">100以内的质数</a><br>
第27行             <a href="/100-200">100-200之间的质数</a><br>
第28行             [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
第29行         </body>
第30行     </html>"""
```

```
第31行     return content #需要返回字符串
第32行
第33行 if __name__=="__main__":
第34行     app.run(host="0.0.0.0",port=80,debug=True)
第35行
第36行 #eof
```

观察例程2-20第29行代码，其功能相当于字符串拼接。相比而言，例程2-19和例程2-20更像网页，功能直接在网页上反映，简单点击即可，而不是在地址栏输入，虽然现在的功能也还非常简单。

例程2-20

```
第1行  def isPrime(N):
第2行      if N<=1:return False
第3行      if N==2:return True
第4行      for i in range(2,N):
第5行          if N%i==0:return False
第6行      return True
第7行
第8行  from flask import Flask #新增代码。装入Flask
第9行  app=Flask(__name__) #新增代码
第10行
第11行 @app.route("/") #新增代码，对应执行root()函数
第12行 @app.route("/<first_second>")
第13行 def root(first_second="200-300"): #另外一种实现方式
第14行     firstNum=int(first_second.split("-")[0]) #转换为整数
第15行     secondNum=int(first_second.split("-")[1]) #转换为整数
第16行     primeList=[N for N in range(firstNum,secondNum) if isPrime(N)]
第17行
第18行     #产生HTML源代码
第19行     content="""<!doctype html>
第20行     <html lang="zh">
第21行         <head>
第22行             <meta charset="utf-8">
第23行             <title>质数列表</title>
第24行         </head>
第25行         <body>
第26行             <a href="/1-100">100以内的质数</a><br>
第27行             <a href="/100-200">100-200之间的质数</a><br>
第28行     """
第29行     content+=str(primeList)
第30行     content+="""
第31行         </body>
第32行     </html>"""
第33行     return content #需要返回字符串
第34行
第35行 if __name__=="__main__":
第36行
```

```
    app.run(host="0.0.0.0",port=80,debug=True)
第37行
第38行 #eof
```

如果网页代码量较大，这很常见，而变化部分较少，采用例程2-20的方式并不友好，而是采用例程2-21与例程2-22的联合实现方式。注意第8行增加了render_template，其功能是处理网页模板，此处为第17行render_template()函数中的prime.html。注意：网页模板默认必须放在名为templates的文件夹下，该文件夹位于Python源程序所在文件夹之下。

例程2-21

```
第1行 def isPrime(N):
第2行     if N<=1:return False
第3行     if N==2:return True
第4行     for i in range(2,N):
第5行         if N%i==0:return False
第6行     return True
第7行
第8行 from flask import Flask,render_template #增加render_template
第9行 app=Flask(__name__) #新增代码
第10行
第11行 @app.route("/") #新增代码，对应执行root()函数
第12行 @app.route("/<first_second>")
第13行 def root(first_second="200-300"): #另外一种实现方式
第14行     firstNum=int(first_second.split("-")[0]) #转换为整数
第15行     secondNum=int(first_second.split("-")[1]) #转换为整数
第16行     primeList=[N for N in range(firstNum,secondNum) if isPrime(N)]
第17行     return render_template("prime.html",primeNums=primeList)
第18行
第19行 if __name__=="__main__":
第20行     app.run(host="0.0.0.0",port=80,debug=True)
第21行
第22行 #eof
```

例程2-22

```
第1行 <!-- 注意:本网页文件名为prime.html-->
第2行 <!doctype html>
第3行 <html lang="zh">
第4行     <head>
第5行         <meta charset="utf-8">
第6行         <title>质数列表</title>
第7行     </head>
第8行     <body>
第9行         <a href="/1-100">100以内的质数</a><br>
第10行         <a href="/100-200">100-200之间的质数</a><br>
第11行         {{primeNums}}
第12行     </body>
第13行 </html>
```

例程2-22第11行代码{{primeNums}}表示该位置将用primeList的值进行替换，被替换内容用双花括号界定。primeNums的名称与例程2-21第17行第二个参数保持一致。

图2-12有输入文本框，当输入数比如23时，点击提交按钮，将该数提交到后台服务器处理程序以判断是否质数，然后返回判断结果。例程2-23和例程2-24是其相关代码。注意例程2-23第8行代码增加了request，用于处理从浏览器端发送到服务器端的数据。具体处理代码如第21行所示。

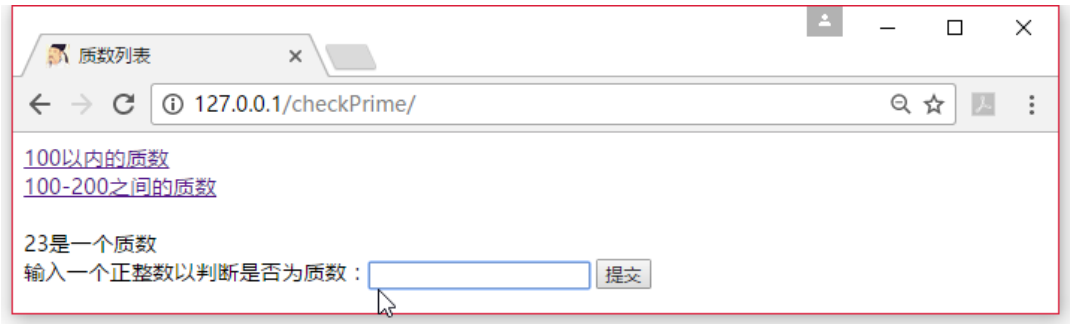


图2-12　例程2-23及例程2-24执行结果

例程2-23

```
第1行  def isPrime(N):
第2行      if N<=1:return False
第3行      if N==2:return True
第4行      for i in range(2,N):
第5行          if N%i==0:return False
第6行      return True
第7行
第8行  from flask import Flask,render_template,request #增加了request
第9行  app=Flask(__name__) #新增代码
第10行
第11行 @app.route("/") #新增代码，对应执行root()函数
第12行 @app.route("/<first_second>")
第13行 def root(first_second="200-300"): #另外一种实现方式
第14行     firstNum=int(first_second.split("-")[0]) #转换为整数
第15行     secondNum=int(first_second.split("-")[1]) #转换为整数
第16行     primeList=[N for N in range(firstNum,secondNum) if isPrime(N)]
第17行     return render_template("prime.html",primeNums=primeList,numRange=first_second)
第18行
第19行 @app.route("/checkPrime/",methods=["get","post"])
第20行 def checkPrime():
第21行     checkNum=int(request.form["intNum"])
第22行     if isPrime(checkNum)==True:
第23行         checkResult=str(checkNum)+"是一个质数"
第24行     else:
第25行         checkResult=str(checkNum)+"不是一个质数"
第26行
第27行     return render_template("checkPrime.html",checkResult=checkResult)
第28行
第29行 if __name__=="__main__":
第30行     app.run(host="0.0.0.0",port=80,debug=True)
第31行
```

```
第32行  #eof
```

例程2-24第12-15行代码产生输入文本框以及提交按钮。第12行中的action="/checkPrime/"表示当单击“提交”按钮时将到服务器端执行的路由。method="post"表示执行方式，常用get或post，在form中常用post。如果选择post，则服务器端的Python代码也必须明确说明，否则将按默认的get执行，如例程2-23第19行所示。methods=["get","post"]表示"/checkPrime/"路由支持get和post两种方式执行。如果二者不一致，将导致错误。

例程2-24

```
第1行  <!-- 注意:本网页文件名为checkPrime.html-->
第2行  <!doctype html>
第3行  <html lang="zh">
第4行     <head>
第5行        <meta charset="utf-8">
第6行        <title>质数列表</title>
第7行     </head>
第8行     <body>
第9行        <a href="/1-100">100以内的质数</a><br>
第10行       <a href="/100-200">100-200之间的质数</a><br><br>
第11行       {{checkResult}}
第12行       <form action="/checkPrime/" method="post">
第13行          输入一个正整数以判断是否为质数：<input type=text name="intNum">
第14行           <input type=submit>
第15行       </form>
第16行    </body>
第17行 </html>
```

例程2-25

```
第1行  <!--文件名为：prime.html-->
第2行  <!doctype html>
第3行  <html lang="zh">
第4行     <head>
第5行        <meta charset="utf-8">
第6行        <title>质数列表</title>
第7行        <link rel="shortcut icon" href="/static/favicon.ico"/>
第8行     </head>
第9行     <body>
第10行       <a href="/1-100">100以内的质数</a><br>
第11行       <a href="/100-200">100-200之间的质数</a><br><br>
第12行       {{numRange}}范围内的质数<br>
第13行       {{primeNums}}<br><br>
第14行       <form action="/checkPrime/" method="post">
第15行          输入一个正整数以判断是否为质数：<input type=text name="intNum">
第16行           <input type=submit>
第17行       </form>
第18行    </body>
第19行 </html>
```

模板文件可以处理从Python程序通过render_template()函数传递的数据，具体处理方法见本章后续章节。另外，如果仅仅显示静态，可用例程2-27第13行方式处理。静态文件执行效果如图2-13所示，当输入两个整数求其范围的所有质数时，其服务器端对应处理代码如例程2-27第32-37行所示。

图2-13 例程2-23及例程2-24执行结果

例程2-26

```
第1行 <!--文件名为：staticPrime.html-->
第2行 <!doctype html>
第3行 <html lang="zh">
第4行     <head>
第5行         <meta charset="utf-8">
第6行         <title>质数导航</title>
第7行     </head>
第8行     <body>
第9行         <a href="/1-100">100以内的质数</a><br>
第10行        <a href="/100-200">100-200之间的质数</a><br><br>
第11行        <form action="/rangePrime/" method="post">
第12行            计算从<input type=text name="firstNum">到<input type=text name="secondNum">范围内的质数
第13行            <input type=submit>
第14行        </form>
第15行    </body>
第16行 </html>
```

例程2-27

```
第1行 def isPrime(N):
第2行     if N<=1:return False
第3行     if N==2:return True
第4行     for i in range(2,N):
第5行         if N%i==0:return False
第6行     return True
第7行
第8行 from flask import Flask,render_template,request #增加render_template
第9行 app=Flask(__name__) #新增代码
第10行
第11行 @app.route("/")
第12行 def Index():#发送一个静态页面
第13行     return app.send_static_file("staticPrime.html")
第14行
第15行 @app.route("/<first_second>")
```

```
第16行 def root(first_second="200-300"): #另外一种实现方式
第17行     firstNum=int(first_second.split("-")[0]) #转换为整数
第18行     secondNum=int(first_second.split("-")[1]) #转换为整数
第19行     primeList=[N for N in range(firstNum,secondNum) if isPrime(N)]
第20行     return render_template("prime.html",primeNums=primeList,numRange=first_second)
第21行
第22行 @app.route("/checkPrime/",methods=["get","post"])
第23行 def checkPrime():
第24行     checkNum=int(request.form["intNum"])
第25行     if isPrime(checkNum)==True:
第26行         checkResult=str(checkNum)+"是一个质数"
第27行     else:
第28行         checkResult=str(checkNum)+"不是一个质数"
第29行
第30行     return render_template("checkPrime.html",checkResult=checkResult)
第31行
第32行 @app.route("/rangePrime/",methods=["get","post"])
第33行 def rangePrime():
第34行     firstNum=int(request.form["firstNum"])
第35行     secondNum=int(request.form["secondNum"])
第36行     primeList=[N for N in range(firstNum,secondNum) if isPrime(N)]
第37行     return render_template("prime.html",primeNums=primeList,numRange="{}-{}".format(firstNum,secondNum))
第38行
第39行 if __name__=="__main__":
第40行     app.run(host="0.0.0.0",port=80,debug=True)
第41行
第42行 #eof
```

注意：在Flask中，静态文件默认放在Python程序所在文件夹之下的static文件夹内，即staticPrime.html文件默认放在static文件夹内。假如静态网页文件中含有图片、JavaScript等文件，也需放在static文件夹下，并说明其位置。如例程2-28所示更多详细说明，请阅读本章后续章节。

例程2-28

第二节　HTML进阶

表2-1：HTML元素列表

标签	描述
<!--...-->	定义注释。
<!DOCTYPE>	定义文档类型。
<html>	定义 HTML 文档。
<head>	定义文档相关信息。
<meta>	定义文档元信息，如<meta charset="utf-8">等等。
<title>	定义文档标题。
<link>	定义文档与外部资源的关系，如链接外部CSS文件、favicon文件等等。
<style>	定义文档的样式信息，与网页美观紧密相关。更多了解参考本章“CSS进阶”

<body>	定义文档的主体，可以认为是网页显示元素的起点元素。
<a>	定义链接，常用属性有href(定义被链接目标)、name(定义链接名称，常用于网页内跳转)和target(链接打开位置)。更多了解参考本章相关章节。
<article>	定义文章，如同报纸由多个文章组成，每个具有一定独立性，每个文章宜用article界定，可独立使用或分发。
<section>	定义section。一个article可以有多个section。和div相比，section有语义，div纯为显示元素。
<header>	定义section、article或网页的头部信息，header和footer同为语义元素。
<footer>	定义section、article或网页的尾部信息，header和footer同为语义元素。
<aside>	定义页面内容之外的内容。
<address>	定义文档作者或拥有者的联系信息。
<abbr>	定义缩写。
<acronym>	定义只取首字母的缩写。
<b>	定义粗体字。
<base>	定义页面中所有链接的默认地址或默认目标。
<bdi>	定义文本的文本方向，使其脱离其周围文本的方向设置。
<bdo>	定义文字方向。
<big>	定义大号文本。
<blockquote>	定义长的引用。
 	定义简单的换行。
<button>	定义按钮 (push button)。注意与<input type=button>的区别。
<canvas>	定义图形。
<cite>	定义引用(citation)。
<code>	定义计算机代码文本。
<command>	定义命令按钮。
<datalist>	定义下拉列表。
<del>	定义被删除文本。
<details>	定义元素的细节。
	定义文档中的节。
<dfn>	定义定义项目。
<dialog>	定义对话框或窗口。
<dl>	定义定义列表。
<dt>	定义定义列表中的项目。
<dd>	定义定义列表中项目的描述。
<embed>	定义外部交互内容或插件。
<fieldset>	定义围绕表单中元素的边框。
<legend>	定义 fieldset 元素的标题。
<figure>	定义媒介内容的分组，以及它们的标题。
<figcaption>	定义 figure 元素的标题。
<form>	定义供用户输入的 HTML 表单。
<textarea>	定义多行的文本输入控件。
<input>	定义输入控件。

<label>	定义 input 元素的标注。
<select>	定义选择列表（下拉列表）。
<option>	定义选择列表中的选项。
<frame>	定义框架集的窗口或框架。
<frameset>	定义框架集。
<h1>-<h6>	定义 HTML 标题。
<hr>	定义水平线。
<i>	定义斜体字。
<iframe>	定义内联框架。
<ins>	定义被插入文本。
<kbd>	定义键盘文本。
<keygen>	定义生成密钥。
<map>	定义图像映射。
<area>	定义图像映射内部的区域。
<mark>	定义有记号的文本。
<menu>	定义命令的列表或菜单。
<menuitem>	定义用户可以从弹出菜单调用的命令/菜单项目。
<meter>	定义预定义范围内的度量。
<nav>	定义导航链接。
<noframes>	定义针对不支持框架的用户的替代内容。
<noscript>	定义针对不支持客户端脚本的用户的替代内容。
<object>	定义内嵌对象。
<optgroup>	定义选择列表中相关选项的组合。
<output>	定义输出的一些类型。
<p>	定义段落。
<param>	定义对象的参数。
<pre>	定义预格式文本。
<progress>	定义任何类型的任务的进度。
<q>	定义短的引用。
<rp>	定义若浏览器不支持 ruby 元素显示的内容。
<rt>	定义 ruby 注释的解释。
<ruby>	定义 ruby 注释。
<samp>	定义计算机代码样本。
<script>	定义客户端脚本。常用属性有src，用于定义脚本文件来源。如果元素有src属性，则不能在元素之间撰写代码。
<small>	定义小号文本。
<source>	定义媒介源。
	定义文档中的节。
<summary>	为 <details> 元素定义可见的标题。
	定义强调文本。

<sup>	定义上标文本。
<sub>	定义下标文本。
	定义强调文本。
<table>	定义表格。
<caption>	定义表格标题。
<tbody>	定义表格中的主体内容。
<tr>	定义表格中的行。
<td>	定义表格中的单元。
<tfoot>	定义表格中的表注内容（脚注）。
<th>	定义表格中的表头单元格。
<thead>	定义表格中的表头内容。
<col>	定义表格中一个或多个列的属性值。
<colgroup>	定义表格中供格式化的列组。
<time>	定义日期/时间。
<track>	定义用在媒体播放器中的文本轨道。
<tt>	定义打字机文本。
	定义无序列表，常与li元素配合使用。
	定义有序列表，常与li元素配合使用。
	定义列表的项目，是ul或ol的子元素。
<var>	定义文本的变量部分。
<audio>	定义声音内容。
<video>	定义视频。
<img>	定义图像。
<wbr>	定义可能的换行符。

1、语义元素

2、表单元素

例程2-29是上传的网页代码，例程2-30是Python Flask代码。虽然上述例程能实现文件上传，但是存在一定风险。如果上传一个可执行文件，而这个可执行文件与系统某个文件系统，可能导致系统出现问题，因此需要约束文件类型。另外，如果不同用户上传相同文件名，有可能导致后上传文件覆盖已上传文件。因此需要进行调整，如例程2-31和例程2-32所示。

例程2-29

```
第1行  <!--文件名：upLoadFile.html-->
第2行  <!doctype html>
第3行  <html lang="zh">
第4行      <head>
第5行          <meta charset="UTF-8">
第6行          <title>上载文件</title>
第7行      </head>
第8行      <body>
第9行          <form action="/File" method="post" enctype="multipart/form-data">
第10行             <input type=file name="yourPic"><br>
第11行             <input type=submit>
```

```
第12行            </form>
第13行        </body>
第14行    </html>
```

例程2-30

```
第1行    from flask import Flask,request
第2行
第3行    app=Flask(__name__)
第4行
第5行    @app.route("/")
第6行    def root():
第7行        return app.send_static_file("upLoadFile.html")
第8行
第9行    @app.route("/File",methods=["post"])
第10行   def upLoadFile():
第11行       file = request.files['yourPic'] #获得文件对象
第12行       savingFileName=file.filename #获得上传文件的名称
第13行       file.save(savingFileName) #保存上传文件
第14行       return "文件上传成功!"
第15行
第16行   if __name__=="__main__":
第17行       app.run(host="0.0.0.0",port=80,debug=True)
```

在浏览器端，例程2-31第17行取得上传文件的名称，第18行取得最后一个圆点的位置后1个字符【fileMsg.lastIndexOf(".")+1】并据此获得文件的扩展名。第19行首先将扩展名转换为全部小写字母【fileType.toLowerCase()】，然后利用jQuery的inArray()函数判断是否在文件类型数组之中，如果在该函数返回所在位置，否则返回-1。如果文件不在数组中，则return false，即取消上传，否则上传。

例程2-31

```
第1行    <!--文件名：upLoadFile.html-->
第2行    <!doctype html>
第3行    <html lang="zh">
第4行        <head>
第5行            <meta charset="UTF-8">
第6行            <title>上载文件</title>
第7行            <script type="text/javascript" src="http://lib.sinaapp.com/js/jquery/3.1.0/jquery-3.1.0.min.js"></script>
第8行        </head>
第9行        <body>
第10行           <form action="/File" method="post" enctype="multipart/form-data">
第11行               <input type=file name="yourPic"><br>
第12行               <input type=submit ID="btnOk">
第13行           </form>
第14行           <script type="text/javascript">
第15行               $("body").css("background-color","gray");
第16行               $("form").submit(function(){
第17行                   fileMsg=$(":input[type='file']").val();
第18行                   fileType=fileMsg.substring(fileMsg.lastIndexOf(".")+1);
```

```
第19行                    if($.inArray(fileType.toLowerCase(),["png","jpg","gif"])==-1){
第20行                        alert("仅允许上传扩展名为png、jpg、gif格式文件");
第21行                        return false;
第22行                    }
第23行                });
第24行            </script>
第25行        </body>
第26行    </html>
```

例程2-32是服务器端处理代码。首先是增加了第2行时间扩展模块，第14行获得时间偏移量的值，该值精确到毫秒，将该值附加到文件名之前，重复的可能性大大降低。当然，还可以增加其他信息，以降低重复概率以及增加可管理性等等。

例程2-32

```
第1行   from flask import Flask,request
第2行   from time import time #增加时间处理
第3行
第4行   app=Flask(__name__)
第5行
第6行   @app.route("/")
第7行   def root():
第8行       return app.send_static_file("upLoadFile.html")
第9行
第10行  @app.route("/File",methods=["post"])
第11行  def upLoadFile():
第12行      file = request.files['yourPic'] #获得文件对象
第13行      savingFileName=file.filename #获得上传文件的名称
第14行      timeMsec=int(time()*1000) #获得1970年01月01日00时00分00秒到现在的毫秒数
第15行      file.save(str(timeMsec)+"_"+savingFileName) #保存上传文件，加上时间标记
第16行      return "文件上传成功!"
第17行
第18行  if __name__=="__main__":
第19行      app.run(host="0.0.0.0",port=80,debug=True)
```

3、多媒体元素

第三节　CSS进阶

第四节　jQuery进阶

jQuery是兼容多种浏览器的JavaScript包，具有体积小、速度快、功能丰富、用途广泛、扩展性良好且易于使用特征，常用于HTML元素遍历及其操纵、事件处理、动画以及AJAX(Asynchronous JavaScript and XML的简写，异步JavaScript和XML)。自2006年推出以来，已在全世界范围内得到广泛应用。

1、元素选择

在很多网页操作中，首先是选中元素。由于网页元素众多，应用场景多变，如何简捷选中元素在过去面临颇多挑战。随着jQuery推出，元素选择变得方便。$("p")是选中网页中的所有p元素，其中p在jQuery中被称为选择器(selector)。jQuery的选择器较多，能完成各种条件下的元素查找定位。如表2-2所示。

表2-2：jQuery选择器

选择器	示例	描述
*	$("*")	选中页面所有元素

#id	$("#yourName")	选中id名为yourName的元素，id具有唯一性
.class	$(".intro")	选中所有class名为intro的元素
element	$("p")	所有页面中所有p元素
.class .class	$(".intro .demo")	选中class名称有intro和demo的元素
:first	$("p:first")	选中所有p元素中的第一个
:last	$("p:last")	选中所有p元素中的最后一个
:even	$("tr:even")	选中所有tr中编号为偶数的元素
:odd	$("tr:odd")	所有所有tr中编号为奇数的元素
:eq(index)	$("ul li:eq(3)")	选中所有ul元素下li中编号为3的元素
:gt(no)	$("ul li:gt(3)")	选中所有ul元素下li中编号大于3的元素
:lt(no)	$("ul li:lt(3)")	选中所有ul元素下li中编号小于3的元素
:not(selector)	$("input:not(:empty)")	选中所有不为空的input元素
:header	$(":header")	选中所有标题元素，即h1-h6元素
:animated		所有动画元素
:contains(text)	$(":contains('ABC')")	选中所有包含ABC的元素。 $("p:contains('ABC')")则是选中所有p元素中含有ABC的元素
:empty	$(":empty")	选中所有无子元素的所有元素
:hidden	$("p:hidden")	选中所有隐藏的p元素
:visible	$("table:visible")	选中所有可见的表格元素
s1,s2,s3	$("p,#yourName,.Demo")	选中所有p元素、id名yourName的元素，class名为Demo的元素，是3个独立选中元素的并集
[attribute]	$("[href]")	选中所有带有href属性的元素
[attribute=value]	$("[href='#']")	选中所有href属性的值为"#"的元素
[attribute!=value]	$("[href!='#']")	选中所有href属性的值不等于"#"的元素
[attribute$=value]	$("[href$='gif']")	选中所有href属性的值以"gif"结尾的元素
:input	$(":input")	选中所有input元素，仅适用于input元素。注：$(":p")不是选中所有p元素是错误。
:text	$(":text")	选中所有type="text"的input元素
:password	$(":password")	选中所有type="password"的input元素
:radio	$(":radio")	选中所有type="radio"的input元素
:checkbox	$(":checkbox")	选中所有type="checkbox"的input元素
:submit	$(":submit")	选中所有type="submit"的input元素
:reset	$(":reset")	选中所有type="reset"的input元素
:button	$(":button")	选中所有type="button"的input元素
:image	$(":image")	选中所有type="image"的input元素
:file	$(":file")	选中所有type="file"的input元素
:enabled	$(":enabled")	选中所有被激活的input元素
:disabled	$(":disabled")	选中所有被禁用的input元素
:selected	$(":selected")	选中所有被选择的input元素
:checked	$(":checked")	选中所有被选中的input元素

$("p")可以称之为初选，在此基础上，还可以过滤(Filtering)和遍历查找(Traversing)。过滤是在当前已有集合中筛选部分元素，而遍历是以查找当元素的出发点，通过网页元素的树形结构找到其他元素。$("div").filter(".Demo")是在所有div元素中，筛选出含有class名为Demo的元素，也就是div中class为Demo的div元素，含Demo的div元素是所有div元素的子集。$("div").find(".Demo")则是在div元素的后代元素中查找class为Demo的元素，以找到的元素为结果集，不是div自身，此为查找。jQuery提供了多种形式的过滤和查找函数，如表2-3所示。

表2-3：jQuery过滤及遍历函数

函数	示例	描述
.eq()	$("p").eq(5)	匹配所有p元素中编号为5(第6个)的元素，此处与$("p:eq(5)")功能类似。
.first()	$("p").first()	匹配所有p元素中的第1个元素，相当于$("p").eq(0)或$("p:first")
.last()	$("p").last()	匹配所有p元素中的最后元素，此处相当于$("p:last")
.has()	$("div").has("p")	从所有div元素中过滤出含有后代元素为p的div元素
.filter()	$("div").has(":Demo")	从所有div元素中过滤出含有class名为Demo的元素。
.is()	$("div").is(".Demo")	如果所有div元素中有一个含有class为Demo，则返回为true，否则返回false。根据选择器检查当前匹配元素集合，如果存在至少一个匹配元素，则返回 true。
.slice()	$("div").slice(2,4)	从所有匹配元素(此处为所有div)选择编号为2到4的子集。
.not()	$("div").not(".Demo")	从所有匹配元素(此处为div)集合中删除元素(此处删除div中其class名为Demo的元素)。
.find()	$("div").find("p")	后代查找。从当前选中的每个元素中(此处为div)查找后代(不仅是子元素)为p的元素。以查找到元素为匹配集合。
.children()	$("div").children("p")	子元素查找。从当前选中的每个元素中(此处为div)查找子元素为p的元素。以查找到元素为匹配集合。
.contents()		获得匹配元素集合中每个元素的子元素，包括文本和注释节点。
.parent()	$("td").parent("tr")	父元素查找。从当前选中的每个元素中(此处为td)查找符合选择器的父元素。parent()可以省略参数，此时查找所有父元素。
.parents()	$("p").parents("div")	长辈元素查找。从当前选中的每个元素中(此处为p)查找符合选择器的长辈元素。
.offsetParent()		获得用于定位的第一个父元素。
.parentsUntil()		获得当前匹配元素集合中每个元素的祖先元素，直到遇到匹配选择器的元素为止。
.closest()		从元素本身开始，逐级向上级元素匹配，并返回最先匹配的祖先元素。
.next()		获得匹配元素集合中每个元素紧邻的同辈元素。
.nextAll()		获得匹配元素集合中每个元素之后的所有同辈元素，由选择器进行筛选（可选）。
.nextUntil()		获得每个元素之后所有的同辈元素，直到遇到匹配选择器的元素为止。
.prev()		获得匹配元素集合中每个元素紧邻的前一个同辈元素，由选择器筛选（可选）。
.prevAll()		获得匹配元素集合中每个元素之前的所有同辈元素，由选择器进行筛选（可选）。
.prevUntil()		获得每个元素之前所有的同辈元素，直到遇到匹配选择器的元素为止。
.siblings()		获得匹配元素集合中所有元素的同辈元素，由选择器筛选（可选）。
.add()	$("div").add("p")	将add()函数选择器中匹配元素添加到已匹配元素的集合中。此处为在所有div元素中新增p元素，构成一个所有div和p构成的元素集合。
.andSelf()		把堆栈中之前的元素集添加到当前集合中。
.each()		对 jQuery 对象进行迭代，为每个匹配元素执行函数。
.end()		结束当前链中最近的一次筛选操作，并将匹配元素集合返回到前一次的状态。
.map()		把当前匹配集合中的每个元素传递给函数，产生包含返回值的新 jQuery 对象。

例程2-33是$.each()、$.nextUntil()以及$.filter()的综合应用。假如唐诗、宋词和元曲作为h1元素其下有很多文本，然后h2分别是某个文人。需要自动生成目录，h1标题前是“一、”、“二、”和“三、”，h2前面的编号类似1.1表示一级标题编号为1二级

编号1，以此类推。第16行代码$("h1").each(function(idxH1){})中的idxH1为变量名称，代表h1的从0开始的序号。第17行的this代表某个h1，第20-21行的this代表某个h2。第19行的$(this).nextUntil("h1").filter("h2")中this代表某个h1，从其含义是从h1开始一直到下一个h1之间或其后的所有元素，然后过滤出全部h2。

例程2-33

```
第1行  <!doctype html>
第2行  <html lang="zh">
第3行     <head>
第4行        <meta charset="UTF-8">
第5行        <title>自动目录</title>
第6行        <script type="text/javascript" src="jquery-3.1.1.min.js"></script>
第7行     </head>
第8行     <body>
第9行        <h1>唐诗</h1>
第10行       <h2>李白</h2><h2>杜甫</h2><h2>岑参</h2><h2>杜牧</h2>
第11行       <h1>宋词</h1>
第12行       <h2>苏轼</h2><h2>姜夔</h2><h2>李清照</h2><h2>王安石</h2>
第13行       <h1>元曲</h1>
第14行       <h2>马致远</h2><h2>关汉卿</h2><h2>白朴</h2><h2>郑光祖</h2>
第15行       <script type="text/javascript">
第16行          $("h1").each(function(idxH1){//idxH1相当于每个h1的编号，从0开始
第17行             thisContent=$(this).html();//this代表循环当前的h1
第18行             $(this).html("零一二三四五六七八九".charAt(idxH1+1)+"、"+thisContent);
第19行             $(this).nextUntil("h1").filter("h2").each(function(idxH2){
第20行                thisContent=$(this).html();//this代表某h2
第21行                $(this).html((idxH1+1)+"."+(idxH2+1)+thisContent);
第22行             })
第23行          })
第24行       </script>
第25行    </body>
第26行 </html>
```

2、事件处理

表2-4：jQuery事件相关表

事件分类	事件名称	描述	备注
鼠标事件	click()	当单击鼠标时触发该事件；或在指定元素触发该事件；	mouseover与mousemove的区别是：mouseover仅在鼠标进入元素时触发事件，在元素内移动不触发事件，而mousemove则当鼠标在元素内移动时连续触发事件。mouseout与mouseleave的区别是当鼠标离开被选元素及其子元素时，都将触发mouseout事件，而mouseleave仅在离开被选元素触发该事件，与子元素没有关系。mouseover与mouseenter与之类似。当触发dblclick与mousedown
	dblclick()	当双击鼠标时触发该事件；或在指定元素触发该事件；	
	mousedown()	当鼠标按下时触发该事件；或在指定元素触发该事件；	
	mouseenter()	当鼠标进入某元素时触发该事件；或在指定元素触发该事件；	
	mouseleave()	当鼠标离开某元素时触发该事件；或在指定元素触发该事件；	
	hover(over,out)	鼠标悬停事件，相当于mouseenter()和mouseleave()的综合。	
	toggle(fn,fn)	鼠标单击切换事件，依次调用toggle()函数的多个函数。	
	mousemove()	当鼠标在某元素移动触发该事件；或在指定元素触发该事件；	
	mouseout()	当鼠标离开某元素时触发该事件；或在指定元素触发该事件；	
	mouseover()	当鼠标进入某元素时触发该事件；或在指定元素触发该事件；	

	mouseup()	当鼠标松开时触发该事件；或在指定元素触发该事件；	触发dblclick与mousedown事件时，click事件同样被激活。
表单事件	blur()	当元素失去焦点触发该事件；或在指定元素触发该事件；	主要用于带有输入性质的元素，如input等；
	select()	文本被选择时触发该事件；或在指定元素触发该事件；	
	submit()	当提交表单时触发该事件；或在指定表单触发该事件；	仅适用于Form元素
	change()	当元素内容发生改变时触发该事件；或在指定表单触发该事件；	该事件仅适用于文本域、textarea和select 元素。
	focus()	当元素获得焦点时触发该事件；或在指定元素触发该事件；	鼠标点击元素或tab键定位到元素时，该元素获得焦点。
键盘事件	keydown()	当键按下时触发该事件；或在指定元素触发该事件；	keypress只能捕获单个可显示字符，区分大小写；keydown和keyup能捕获出PrtSc之外的所有字符，包括特殊键如：方向键、ctrl状态、alt状态等。
	keypress()	当键按下时触发该事件；或在指定元素触发该事件；	
	keyup()	当键松开时触发该事件；或在指定元素触发该事件；	
窗口/文档事件	load()	当元素及其子元素加载时触发该事件；	load与unload在不同浏览器存在兼容性问题。ready()仅适用于文档，有三种写法：$(document).ready(fn)、$().ready(fn)、$(fn)，其中fn代表函数(function)。
	ready()	当网页文档就绪时触发该事件，多数事件都应在该事件出发时开始工作，以防止网页内容不完整导致问题。	
	unload()	仅适用于window对象，即进入新网页或离开，以及关闭窗口时触发unload 事件	
浏览器事件	resize()	浏览器窗口尺寸改变时触发该事件；	
	scroll()	当浏览器滚动条时触发事件及其代码；	

3、文档操作

文档处理用于处理匹配元素中的内容，前面已经多次使用过的$.html()或$.html(content)即是常用的文档处理函数。文档处理包括：内部插入、外部插入以及包裹等。也还包括清空、删除以及复制等。如图2-14所示，相关代码如例程2-34所示。

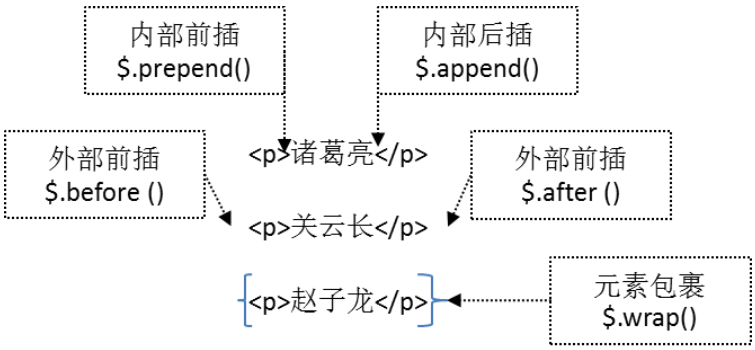


图2-14 文档处理示意图

例程2-34

```
第1行  <Html>
第2行    <Head>
第3行      <Title>文档处理</Title>
第4行      <Script Language="JavaScript" src="./jQuery-1.9.1.min.js"></Script>
第5行    </Head>
第6行    <Body>
第7行      <P>诸葛亮</P>
第8行      <P>关羽</P>
第9行      <P>张飞</P>
```

```
第10行          <P>赵云</P>
第11行          <P>黄忠</P>
第12行          <Script Language="JavaScript">
第13行             $("P:first").prepend("蜀国军师:");//内部前插
第14行             $("P:first").append("，<BR>字孔明，号卧龙。");//内部后插
第15行             var mySelection=$("P:eq(2)");
第16行             mySelection.before("<p>姜维</p>");//外部前插
第17行             mySelection.after("<p>廖化</p>");//外部后插
第18行             $("P:contains('黄忠')").wrap("<div style='font-weight:bold'></div>");
第19行          </Script>
第20行       </Body>
第21行    </Html>
```

内部插入分为：内部前插(prepend，第13行)和内部后插(append，第14行)；外部插入分为：外部前插(before，第16行)和外部后插(after，第17行)；包裹元素常用wrap()函数(第17行)。

值得注意的是：如果将第16、17行代码修改为$("P:eq(2)").before(" 姜维 ");和$("P:eq(2)").after(" 廖化 ");将与现有效果不同。当执行before()之前，“张飞”的编号是2；当执行before()后，“张飞”的编号变成3，而“姜维”编程2。而var mySelection=$("P:eq(2)");则是将“张飞”所在元素保存在变量mySelection中，只要不修改变量内容，其所代表的一直是“张飞”所在元素。

(1) 内部插入

内部插入用于在匹配元素内插入新内容，插入的内容可以在该元素原有内容之后(append)、也可以在原有内容之前(prepend)，有多种方法支持内部插入，如表1所示。

表2-5：内部插入操作

操作	描述
$.append(content)	在每个匹配元素原内容之后插入content所指定的新内容。
$.appendTo(selector)	将匹配元素移动并追加到selector参数所匹配的元素内。
$.append(function(index,html))	将function(index,html)返回值插入到匹配元素原有内容之后，其中index代表操作元素编号，html表示内容。
$.prepend(content)	在每个匹配元素原内容之前插入content所指定的新内容。
$.prependTo(selector)	将匹配元素移动并前插到selector参数所匹配的元素内。
$.prepend(function(index,html))	将function(index,html)返回值插入到匹配元素原有内容之前，其中index代表操作元素编号，html表示内容。
$.after(content)	在匹配元素后插入content所代表的内容。
$.after(function(idx){})	在匹配元素后插入function(idx)所返回的值，idx代表匹配元素的编号。
$.before(content)	在匹配元素前插入content所代表的内容。
$.before(function(idx){})	在匹配元素前插入function(idx)所返回的值，idx代表匹配元素的编号。
$.insertAfter(selector)	将匹配元素移动并插入到selector所指定的元素之后。
$.insertBefore(selector)	将匹配元素移动并插入到selector所指定的元素之前。
$.wrap(html)	将所有匹配元素用html代码包裹。
$.wrap(element)	把所有匹配的元素用DOM元素包裹。
$.wrap(function(Idx){})	把所有匹配的元素用function()函数的返回值包裹。
$.unwrap()	移走匹配元素的父元素。
$.wrapAll(html)	将所有匹配元素用Html元素包裹，形成所有匹配元素的父元素。
$.wrapAll(element)	将所有匹配元素DOM元素包裹，形成所有匹配元素的父元素。

操作	描述
$.wrapInner(html)	将匹配元素的子元素及其内容用Html代码包裹，在匹配元素与原有内容之间插入包裹元素。
$.wrapInner(element)	将匹配元素的子元素及其内容用DOM元素包裹，在匹配元素与原有内容之间插入包裹元素。
$.wrapInner(function(Idx){})	将匹配元素的子元素及其内容用function()返回的Html代码包裹，在匹配元素与原有内容之间插入包裹元素。
element是DOM对象，有关DOM对象将后续章节中讲解。	
$.replaceWith(content)	用content所代表的内容替换匹配元素。
$.replaceAll(selector)	用$(参数)的内容替换selector匹配所匹配的元素，如例程2-41第5行所示。
$.empty()	清空匹配元素包括子元素在内的内容，但该元素还存在。
$.remove(content)	删除所有匹配元素，被删除元素被保存在jQuery中，但事件等被删除。
$.detach(content)	删除所有匹配元素，元素被保留在jQuery中，包括事件等。
$.clone()	克隆并选中匹配元素。
$.clone(true)	克隆元素及其事件并且选中匹配元素

例程2-35是部分内部插入功能的应用，图2-15是其执行效果。

例程2-35

```
第1行  <Body>
第2行    <UL>
第3行      <LI>政
第4行        <UL>
第5行          <LI class="L4">元春</LI>
第6行          <LI class="L4">珠</LI>
第7行          <LI class="L4">宝玉</LI>
第8行          <LI class="L4">环</LI>
第9行          <LI class="L4">探春</LI>
第10行       </UL>
第11行     </LI>
第12行     <LI>赦</LI>
第13行   </UL>
第14行   <Script Language="JavaScript">
第15行     $("LI").prepend("<b>贾</b>");//在LI内部内容之前插入一个加粗“贾”字
第16行     $("LI").append("[荣国府]");//每个LI内部内容之后插入“[荣国府]”
第17行     $(".L4").prepend(function(index,html){
第18行       return index+". ";
第19行     });
第20行   </Script>
第21行 </Body>
```

- 贾政
  - 0. 贾元春 ［荣国府］
  - 1. 贾珠 ［荣国府］
  - 2. 贾宝玉 ［荣国府］
  - 3. 贾环 ［荣国府］
  - 4. 贾探春 ［荣国府］

［荣国府］
- 贾赦 ［荣国府］

图2-15 例程5-24执行后效果图

注意：“贾赦”后面有“[荣国府]”,而“贾政”后面没有，但“贾探春”后面有两个“[荣国府]”，其原因是例程2-35第16行在每个LI后面都增加了“[荣国府]”，而“贾政”对应的LI还包括第4行到第10行的代码，因此增加在该位置。

第17行到第19行是$.prepend(function(index,html))的一个应用，function(index,html)表示该函数可以输入两个参数，index代表选中的".L4"的序号(索引值)，html代表其对应的内容，其函数返回值插入到其原有内容的前面。$.prepend(function(index,html))很适合用于章节编号，假定某网页中有大量H2标记，其内容前没有编号，使用该功能，能方便地加上编号，其代码如例程2-36所示。

例程2-36

```
第1行  <Script Language="JavaScript">
第2行    $("H2").prepend(function(index,html){
第3行      return (index+1)+".";//index从开始编号，index+1则实现从1开始编号
第4行    });
第5行  </Script>
```

例程2-37是$.appendTo(content)的应用。appendTo用于将$(参数)匹配元素移动并追加到content选定的元素，prependTo与之类似，不过一个插在原内容之后，一个插在原内容之前。图2-16是例程2-37执行效果，从代码可以看出，P元素原有内容字号不大且没有下划线，移动到Div元素后，font-size和text-decoration都发生了变化，其原因是$("P")所匹配的元素已经被插入到"#Box"所匹配的元素中。

例程2-37

```
第1行  <Body>
第2行    <P>世事洞明皆学问，人情练达即文章！</P>
第3行    <Div style="font-size:40px;text-decoration:underline" ID="Box">摘自《红楼梦》 作者：曹雪芹</Div>
第4行    <Script Language="JavaScript">
第5行      $("P").appendTo("#Box");
第6行    </Script>
第7行  </Body>
```

摘自《红楼梦》 作者：曹雪芹

世事洞明皆学问，人情练达即文章！

图2-16 例程2-37执行效果图

(2) 外部插入

内部插入内容在选中元素之内，而外部插入在选中元素之外，新插入的内容与匹配元素成为兄弟姐妹关系，是同辈关系。如选中"李白"元素后，执行$("b").after("杜甫")，则形成内容形式为"李白杜甫"，即在“李白”之后插入“杜甫”。下表是外部插入操作，例程2-38是外部插入的一个应用。

例程2-38

```
第1行  <HTML>
```

```
<HEAD>
  <TITLE>外部插入</TITLE>
  <Script language="JavaScript" src="./jquery-1.9.1.min.js"></Script>
  <META http-equiv="Content-type" content="text/html; charset=gb2312">
</HEAD>

<BODY>
  <Img src="1.gif" title="春意盎然" style="width:220px;">
  <Script Language="JavaScript">
    $("Img").after(function(){
      var imgTitle=$(this).attr("title");
      $(this).after("<Div>"+imgTitle+"</Div>");
    });
  </Script>
</Body>
</HTML>
```

(3) 元素包裹

在“<Div>李白</Div>”代码中称Div元素包裹b元素，在jQuery中，可以给匹配元素包裹一个新元素，如上例的实现代码为$("b").wrap("<Div></Div>")。下表是元素包裹的多种方式，例程2-39是元素包裹的一个应用，图2-17是其执行效果。

例程2-39

```
<Html>
  <Head>
    <Title>文档处理</Title>
    <Script Language="JavaScript" src="./jQuery-1.9.1.min.js"></Script>
    <Style type=text/css>
      #Box{width:200px;text-align:Center}
      P{margin:0px;}
    </Style>
  </Head>
  <Body style="width:200px">
    <Div id="Box">
      <P>诸葛亮</P>
      <P>关羽</P>
      <P>张飞</P>
      <P>赵云</P>
      <P>黄忠</P>
      <P>周瑜</P>
      <P>鲁肃</P>
      <P>张昭</P>
      <P>甘宁</P>
      <P>陆逊</P>
      <P>吕蒙</P>
    </Div>
    <Script Language="JavaScript">
```

```
第25行          $("P").slice(0,5).wrap("<div style='border:1px solid gray'></Div>");
第26行          $("P").slice(6).wrapAll("<div style='border:1px solid gray'></Div>");
第27行          $("P:gt(8)").wrapInner("<b></b>");//相当于<P><b>陆逊</b></P><P><b>吕蒙</b></P>
第28行       </Script>
第29行    </Body>
第30行 </Html>
```

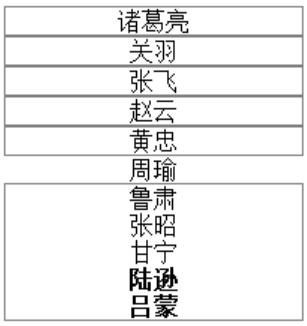


图2-17　不同包裹模式效果图

例程2-39第25行wrap()函数是给每个选中的元素外包裹一个带边框的Div元素，所以从“诸葛亮……黄忠”每个元素都有边框(共计5个)；而第26行wrapAll()则是在所有匹配元素外增加一个带边框的Div元素(总计1个)；第27行的wrapInner()则是将每个匹配元素内的内容包括在B元素内。

例程2-40是wrap()函数的一个具体应用，图2-17是其执行效果。

例程2-40

```
第1行  <HTML>
第2行     <HEAD>
第3行        <TITLE>元素包裹</TITLE>
第4行        <Script language="JavaScript" src="./jquery-1.9.1.min.js"></Script>
第5行        <META http-equiv="Content-type" content="text/html; charset=gb2312">
第6行        <Style type="text/css">
第7行           .imgBox{width:250px;text-align:Center;background-color:#E6E6E6;padding-top:20px}
第8行           .imgBox Div{font-weight:bold;margin-top:10px;margin-bottom:10px}
第9行        </Style>
第10行    </HEAD>
第11行
第12行    <BODY>
第13行       <Img src="1.gif" title="春意盎然" style="width:220px;">
第14行       <Script Language="JavaScript">
第15行          $("Img").wrap("<Div class='imgBox'></Div>");//元素包裹
第16行          $("Img").after(function(){
第17行             var imgTitle=$(this).attr("title");
第18行             $(this).after("<Div>"+imgTitle+"</Div>");
第19行          }); //元素外部插入
第20行       </Script>
第21行 </Body>
第22行 </HTML>
```

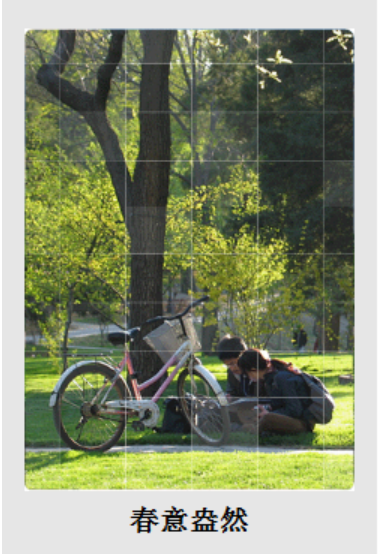


图2-18　元素包裹

(4) 替换、删除和克隆

一般说来，内部插入、外部插入以及包裹都不会改变被选中的元素，而替换将使匹配元素被新的元素或内容所替代，匹配元素将不存在，如例程2-41所示。删除将影响到匹配元素，$.empty()将删除匹配元素内的子元素及其内容；$.detach()和$.remove()则将删除匹配元素及其内的内容，二者之间稍有区别，请看表4。$.clone()则可以克隆元素，如例程2-42。

例程2-41

```
<SCRIPT LANGUAGE="JavaScript">
   //<P>张飞</P>被替换为<B>张飞</B>，注意不是<P><b>张飞</b></P>
   $("P:contains('张飞')").replaceWith("<b>张翼德</b>");
   //<b>周公瑾</b>替换<P>周瑜</P>
   $("<b>周公瑾</b>").replaceAll("P:contains('周瑜')");
   alert($(".State:eq(2)").html());
</SCRIPT>
```

例程2-42

```
<HTML>
   <HEAD>
      <TITLE>元素clone</TITLE>
      <Script language="JavaScript" src="./jquery-1.9.1.min.js"></Script>
      <META http-equiv="Content-type" content="text/html; charset=gb2312">
   </HEAD>

   <BODY>
      <Div><img src="1.gif" style="width:120px"></Div>
      <Script Language="JavaScript">
         $("div").append($("img").clone()).append($("img").clone());
      </Script>
   </Body>
</HTML>
```

图2-19 元素clone

4、属性操作

不少元素含有属性，如Img含有src等属性，table含有cellpadding等属性。jQuery可以方便地操作属性，$.attr(attrName)获取匹配元素的指定属性(attrName)的值(内容)，而$.attr(attrName,attrValue)则可以设置匹配元素指定属性(attrName)的值(attrValue)，$.removeAttr(attrName)则可以从选中元素移走(删除)指定属性(attrName)。更多操作如下表所示。

表2-6：属性操作

操作	描述
$.attr(attrName)	获取第一个匹配元素的属性值，如该元素没有指定属性(attrName)，则返回undefined。
$.attr(attrName/attrValue值对)	以“名/值”形式设置所有匹配元素的属性。
$.attr(attrName,attrValue)	为匹配元素指定属性(attrName)并设置属性值(attrValue)。
$.attr(attrName,function(){})	为匹配元素指定属性(attrName)设置由函数function()返回的属性值。
$.removeAttr(attrName)	删除所有匹配元素中指定的属性(attrName)。

例程2-43、例程2-44(效果如图2-20所示)是jQuery操作attr的应用。

例程2-43

```
<script Language="JavaScript">
    //读取img的src属性作为每个img元素的title属性值
    $("img").attr("title",function(){
        return $(this).attr("src");
    })
</script>
```

例程2-44

```
<html>
    <head>
        <title>jQuery操作attr</title>
        <script language="JavaScript" src="./jquery-1.9.1.min.js"></script>
        <meta http-equiv="Content-type" content="text/html; charset=gb2312">
        <style type="text/css">
            .imgBox{display:inline-block;width:220px;}
            .imgBox img{width:200px}
            #imgTitle{text-align:Center;font-weight:bold}
            #controlArea span{display:inline-block;width:30px;height:30px;border:1px solid red}
            .selectedSpan{background-color:red}
        </style>
    </head>

<body>
```

```
第16行     <div class="imgBox">
第17行       <img src="1.gif" ID="imgShow">
第18行       <div ID="imgTitle">春意</div>
第19行       <div ID="controlArea">
第20行         <span imgName="1.gif" imgTitle="春意" class="selectedSpan">□</span>
第21行         <span imgName="2.gif" imgTitle="钟亭">□</span>
第22行         <span imgName="3.gif" imgTitle="入画">□</span>
第23行         <span imgName="4.gif" imgTitle="淡定">□</span>
第24行         <span imgName="5.gif" imgTitle="绿塘">□</span>
第25行       </div>
第26行     </div>
第27行
第28行     <script Language="JavaScript">
第29行       $("#controlArea>span").click(function(){
第30行
第31行         $(this).addClass("selectedSpan");//为选定元素addClass
第32行         //基于当前选定查找同辈class名为selectedSpan元素，并removeClass
第33行         $(this).siblings(".selectedSpan").removeClass("selectedSpan");
第34行
第35行         var thisImgTitle=$(this).attr("imgTitle");//获取当前元素的imgTitle属性
第36行         var thisImgName=$(this).attr("imgName");//获取当前元素的imgName属性
第37行
第38行         $("#imgTitle").html(thisImgTitle);//改变ID为imgTitle之html内容
第39行         $("#imgShow").attr("src",thisImgName);//修改ID为imgShow的src属性值
第40行       });
第41行     </script>
第42行   </body>
第43行   </html>
```

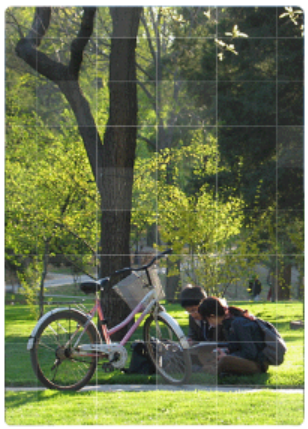


图2-20　例程2-44执行效果图

5、CSS操作

CSS是网页的重要组成部分，是网页美观的基础。jQuery提供了多种读取CSS属性的函数(方法)。$.addClass()和$.removeClass()能改变匹配元素的Class名称，而Class可以设定多种CSS效果。但是这种方法是将Class作为一个整体进行控制，而

不能控制CSS每个项目。即$.addClass()和$.removeClass()的操作单位是class，而读取CSS属性则是具体到每个CSS项目。读取粒度有很大不同。

表2-7：读写CSS属性值

操作	描述
$.css(cssItemName)	读取第一个匹配元素的CSS属性的值。
$.css(cssItemName,value)	设置所有匹配元素指定样式(cssItemName)的属性值。注意：数字将被转换为像素值。
$.css(cssItemName,function(index,value){})	设置所有匹配元素指定样式(cssItemName)的属性值(由function()返回值确定)。数值将自动转换为像素值。index为元素的索引序号，value为原cssItemName的值，返回值为设置值。如例程2-45所示，执行效果如图2-21所示。
$.css(cssItemName/Value值对)	把一个“cssItemName/Value值对”设置为所有匹配元素的样式属性，适用于在所有匹配的元素上设置多属性。例程2-45第15行是其应用示例。
$.offset()	读取匹配元素相对于文档左上角的相对偏移，返回值为一个对象，属性有top和left，分别对应于垂直偏移和水平偏移。例程2-46第15行、16行是其应用示例。
$.offset(坐标值对)	设置匹配元素相对于文档偏移坐标，坐标值对形如{top:10,left:30}。例程2-46第35行是其应用示例。
$.position()	读取匹配元素相对于父元素的偏移，返回值对象与$.offset()相同。
$.scrollTop()	获得匹配元素相对于垂直滚动条顶部的偏移。
$.scrollTop(value)	设置匹配元素相对于垂直滚动条顶部的偏移。
$.scrollLeft()	获得匹配元素相对于水平滚动条左侧的偏移；
$.scrollLeft(value)	设置匹配元素相对于水平滚动条左侧的偏移；
$.height()	获得第一个匹配元素的高度值(单位为px)；
$.height(val)	设置匹配元素的高度值，如未指明单位，则默认为px；
$.width()	获得第一个匹配元素的宽度值(单位为px)；
$.width(val)	设置匹配元素的宽度值，如未指明单位，则默认为px；
$.innerHeight()	获取第一个匹配元素内部高度(包括补白、不包括边框)，对可见和隐藏元素均有效
$.innerWidth()	获取第一个匹配元素内部宽度(包括补白、不包括边框)，对可见和隐藏元素均有效
$.outerHeight(switch)	获取第一个匹配元素外部高度(默认包括补白、边框)，对可见和隐藏元素均有效。如switch值为false，则不包括补白和边框。
$.outerWidth(switch)	获取第一个匹配元素外部宽度(默认包括补白、边框)，对可见和隐藏元素均有效。如switch值为false，则不包括补白和边框。

例程2-45

```
第1行  <html>
第2行    <head>
第3行      <title>CSS属性设置</title>
第4行      <script Language="JavaScript" src="./jQuery-1.9.1.min.js"></script>
第5行    </head>
第6行    <body>
第7行      <div class="BigMan">
第8行        <P>诸葛亮</P>
第9行        <P>关羽</P>
第10行       <P>张飞</P>
第11行       <P>赵云</P>
第12行
```

```
            <P>黄忠</P>
第13行      </Div>
第14行      <Script LANGUAGE="JavaScript">
第15行        $("P").css({"font-size":"12px","margin":"0px"});//用key/Value设置CSS属性
第16行        //用户function()函数的返回值设置属性
第17行        $("P").css("font-size",function(index,cssValue){
第18行          //注意:font-size的属性值含有px字符，形如:19px。
第19行          //cssValue为当前元素font-size的CSS属性值
第20行          //相当于$(this).css("font-size");
第21行          var currentFontSize=cssValue;
第22行          currentFontSize=currentFontSize.replace("px","");//替换px为空白，相当于删除px
第23行          return currentFontSize*(index+1)+"px";//返回值用于设置属性
第24行        });
第25行      </script>
第26行    </body>
第27行  </html>
```

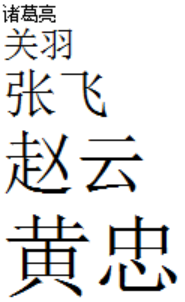


图2-21　例程2-45执行效果

例程2-46

```
第1行   <html>
第2行     <head>
第3行       <title>元素横向弹动</title>
第4行       <script Language="JavaScript" src="./jQuery-1.9.1.min.js"></script>
第5行     </head>
第6行     <body>
第7行       <div id="Box" style="width:300px;height:100px;border:1px solid gray;padding-top:30px">
第8行         <span style="color:red">●</span>
第9行       </div>
第10行      <script LANGUAGE="JavaScript">
第11行
第12行        setTimeout("walkingDiv(true)",10);
第13行
第14行        function walkingDiv(Direction){
第15行          var BoxPosition=$("#Box").offset();//取得id名为Box元素的相对于文档左上角的偏移
第16行          var BoxLeft=BoxPosition.left;//左边的值
第17行          var BoxRight=BoxLeft+$("#Box").width();//到右边框的值
第18行          var myDiv=$("#Box>span");//取得span元素；
第19行          var nowPosition=myDiv.offset();//span元素相对于文档左上角的偏移量
```

```
第20行                if(Direction==true){//根据Direction的值确定左移动还是右移动
第21行                    var newLeft=nowPosition.left+2;//右移
第22行                }
第23行                else{
第24行                    var newLeft=nowPosition.left-2;//左移
第25行                }
第26行                if(newLeft>=BoxRight-BoxLeft){//到达右边界处理
第27行                    newLeft=newLeft-2;//左移
第28行                    Direction=false;//设定移动方向
第29行                }
第30行                if(newLeft<=BoxLeft){//到达左边界处理
第31行                    newLeft=newLeft+2;//右移
第32行                    Direction=true;//设定移动方向
第33行                }
第34行                var newTop=nowPosition.top;//顶坐标不改变
第35行                myDiv.offset({left:newLeft,top:newTop});
第36行
第37行                //启动下一次移动
第38行                setTimeout("walkingDiv("+Direction+")",10);
第39行            }
第40行        </script>
第41行    </body>
第42行 </html>
```

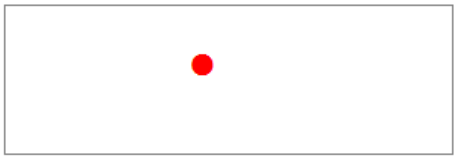


图2-22　小球横向弹动效果

6、数据交互

网页能与远程后台服务器数据交互，网页将不再数据静态页面，同时也有了与其他用户交互数据的可能。如图2-23所示，浏览器可以从Web服务器获取数据也可以向Web服务器发送数据。通过服务器中介，还可以实现浏览器与浏览器之间的数据交互。即浏览器A可以将数据交互到Web服务器保存，浏览器B可以从Web服务器获得浏览器A提供的数据。例程2-47是从服务器获取数据的示例。

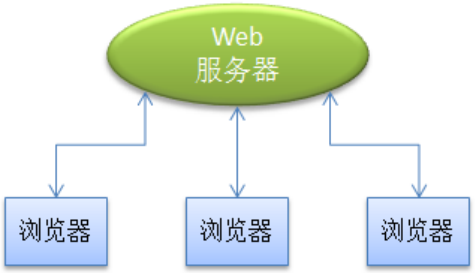


图2-23　B/S架构模型

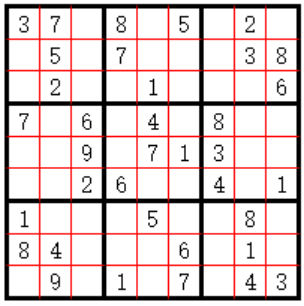
例程2-47第39行$.get("sudoku.data",function(data){})用于从服务器端获取数据，其中sudoku.data是服务器端存放数据的文件，与网页在同一文件夹。function(data)中的data可以任意命名，用于代表从服务器端取得的数据。

例程2-47

```
第1行 <!doctype html>
```

```
第2行  <html lang="zh">
第3行    <head>
第4行      <meta charset="UTF-8">
第5行      <title>从服务器获得数独数据</title>
第6行      <script type="text/javascript" src="jquery-3.1.1.min.js"></script>
第7行      <style type="text/css">
第8行        td{width:20px;height:20px;text-align:center}/*定义表格宽度和高度，内容居中*/
第9行        .tdBottom{border-bottom:1px solid red;}
第10行       .tdRight{border-right:1px solid red;}
第11行       .tdTop{border-top:3px solid black}
第12行       .tdLeft{border-left:3px solid black}
第13行     </style>
第14行   </head>
第15行   <body>
第16行     <div id="contentBox"></div>
第17行     <script type="text/javascript">
第18行       var sudoku="<table id='sudokuTab' cellpadding='0' cellspacing='0'>";
第19行       for(var i=0;i<9;i++){
第20行         sudoku+="<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>";
第21行       }
第22行       sudoku+="</table>";
第23行       $("#contentBox").html(sudoku);
第24行       //上述代码生成容纳数独数据的表格
第25行
第26行       $("#sudokuTab td").addClass("tdBottom").addClass("tdRight");//给每个td元素添加下右边线
第27行
第28行       $("#sudokuTab td:nth-child(3n)").css("border-right","3px solid black").removeClass("tdRight");
第29行       $("#sudokuTab tr:nth-child(3n) td").css("border-bottom","3px solid black");
第30行
第31行       $("#sudokuTab tr:last td").removeClass("tdBottom");
第32行       $("#sudokuTab tr").find("td:last").removeClass("tdRight");
第33行       $("#sudokuTab tr").find("td:first").addClass("tdLeft");
第34行       $("#sudokuTab tr:first td").addClass("tdTop");
第35行       //上述代码生成各种表格线
第36行
第37行       var sudokuRows=$("#sudokuTab tr");
第38行       //sudoku.data内容：370805020050700038020010006706040800009……
第39行       $.get("sudoku.data",function(data){
第40行         for(var row=0;row<9;row++){
第41行           for(var col=0;col<9;col++){
第42行             var num=data.charAt(row*9+col);
第43行             if (num=="0")num="";//将0处理为空字符
第44行             sudokuRows.eq(row).children("td").eq(col).html(num);
第45行           }
```

```
第46行                }
第47行            });//$.get()代码块从服务器获取数据并添加到表格中
第48行        </script>
第49行    </body>
第50行 </html>
```



3	7		8		5		2	
	5		7				3	8
	2			1				6
7		6		4		8		
		9		7	1	3		
		2	6			4		1
1				5			8	
8	4				6		1	
	9		1		7		4	3

图2-24　例程2-47执行效果

7、JSON与XML

数据存储还常用XML格式。XML(eXtensible Markup Language，可扩展标记语言)与HTML(HyperText Markup Language)都有共同的ML两个字符，也都是Markup Language的简写，预示二者存在联系。观察例程2-48就会发现，其代码真有某些相似，元素名称开始和元素结束放在一对尖括号中，结束时名称有斜杠(/)，元素可以有属性，也可以没有属性。和很多HTML元素一样，元素内还可以包含子元素，在本例中，student元素包含有name、sex、native元素。但是，和HTML有很大的不同，XML的元素名称似乎都没有在HTML出现过，看起来是自由命名。确实如此，XML的标记(元素)可以根据需要扩展，只要这种标记集合能为使用者所认可。如果程序由一个程序员完成，只要在其开发范围内获得认可即可；如果由多个程序员完成就需要在程序员之间约定标记集合形成规范，使不同程序员开发的程序能互相交换数据；当需要在更大范围内交换数据，则需要形成公司规范、行业标准、国家标准乃至国际标准。当前已有多种国际标准，比如：描述数学公式的标准、矢量图形的标准、电子商务标准等。

例程2-48

```
第1行 <?xml version="1.0" encoding="utf-8"?>
第2行 <root>
第3行     <student id="1100012010">
第4行         <name>灭绝师太</name>
第5行         <sex>女</sex>
第6行         <native>峨眉山</native>
第7行     </student>
第8行     <student id="1100012011">
第9行         <name>张三丰</name>
第10行        <sex>男</sex>
第11行        <native>武当山</native>
第12行    </student>
第13行    <student id="1100012110">
第14行        <name>黄药师</name>
第15行        <sex>男</sex>
第16行        <native>桃花岛</native>
第17行    </student>
第18行 </root>
```

和HTML不同，XML对大小写是敏感，每个元素都必须有开始和结束标记，其开始和结束标记大小写必须匹配。对于没有内容的XML元素，可以写成形如<native></native>，亦或<native/>，选择后一种更好。XML元素的属性值必须用半角引号包围，而HTML的属性值是否用引号属于可选。XML全部数据必须包含在有且仅有一个根元素内，如文档root元素。

XML文档(数据)第一行通常需要申明该文档(或数据)所采用XML版本号及其编码。在例程2-48中，设定该XML文档版本号为1.0，其编码为utf-8。常用编码标准还有gb2312等。

XML数据可以读取融进网页中，如例程2-49所示。

例程2-49

```
第1行  <HTML>
第2行    <HEAD>
第3行      <TITLE>读取XML数据</TITLE>
第4行      <Script type="Text/JavaScript" language="JavaScript" src="./jquery-1.9.1.min.js"></Script>
第5行      <style type=text/css>
第6行        #showData table{width:500px;}
第7行        #showData td{height:30px;line-height:28px;border-bottom:1px dotted gray;text-align:Center}
第8行        #showData th{height:40px;background-color:#DFDFDF;border-bottom:2px solid black}
第9行      </style>
第10行   </HEAD>
第11行   <BODY>
第12行     <Div id="showData"></Div>
第13行     <SCRIPT LANGUAGE="JavaScript">
第14行       $.get("xmlData.xml",function(xml){
第15行         var tabHeader="<table cellpadding='0' cellspacing='0'>";
第16行         var tabFooter="</table>";
第17行         var tabRow="<tr><th>学号</th><th>姓名</th><th>籍贯</th>";
第18行         $(xml).find("student").each(function(){
第19行           tabRow+="<tr>";
第20行           tabRow+="<td>"+$(this).attr("id")+"</td>";
第21行           tabRow+="<td>"+$(this).children("name").text()+"</td>";
第22行           tabRow+="<td>"+$(this).children("native").text()+"</td>";
第23行           tabRow+="</tr>";
第24行         });
第25行         $("#showData").html(tabHeader+tabRow+tabFooter);
第26行       });
第27行     </SCRIPT>
第28行   </BODY>
第29行 </HTML>
```

学号	姓名	籍贯
1100012010	灭绝师太	峨眉山
1100012011	张三丰	武当山
1100012110	黄药师	桃花岛

图2-25 例程2-49执行效果图

例程2-49第14行代码$.get("xmlData.xml",function(xml){})的功能是获取xmlData.xml中的数据，并交由函数function(xml)去处理，其中xmlData.xml是URL，可以是远程Web地址中的数据，并常常是如此体现，此处为获得当前网页所在文件夹下的xmlData.xml数据。function(xml)中的xml并不是关键词，相当于一个参数，代表获得的数据，可以更改为其他名称如data，xmlData等，最好用含xml的名称，以表明获得的数据是XML格式。

8、动画效果

观察图2-26，有些像电影胶片。假定这些图片都出现在第一个图片框中，可以想象随着时间推移，图片在从右到左移动进场，其实现代码如例程2-50所示。在例程中，“setInterval("slidePic()",40);”的功能是每个40毫秒执行一次slidePic()这个函数；而slidePic()的核心功能是每次减小图片left值20个像素(Left是一个CSS属性)。综合其描述，其功能就是每隔40毫秒，图片的left值减小20个像素，视觉效果就是图片从右到左运动进场。

图2-26　动画效果示意图

jQuery提供了动画函数，如例程2-50所示，一行代码即可实现。“$("#Box>img").animate({"left": '0px'}, "slow");”的功能是选中id为Box下的img元素，并执行动画函数，修改其left属性，直到其值变化为0px为止，其动画速度为slow(慢)。凡元素CSS数值属性都可以用animate函数予以控制，实现动画效果，如:Left、Top、Height、Width、Padding、Margin、Font-Size等等。jQuery动画的本质就是以指定的速度(单位时间内改变值，如40毫秒减少10个像素)改变CSS属性，其速度可通过计算按照某种规律变化，不一定匀速改变。

例程2-50

```
第1行  <Html>
第2行    <Head>
第3行      <Title>动画</Title>
第4行      <Script Language="JavaScript" src="./jQuery-1.9.1.min.js"></Script>
第5行      <Style type=text/css>
第6行        #Box{width:152px;height:214px;overflow:hidden;background-color:#EBEBEB;display:inline-block}
第7行        #Box Img{position:relative;width:152px;left:152px}
第8行      </Style>
第9行    </Head>
第10行   <Body>
第11行     <Div id="Box"><Img src="3.gif"></Div>
第12行     <Script LANGUAGE="JavaScript">
第13行       setInterval("slidePic()",40);
第14行       function slidePic(){
第15行         var myImg=$("#Box>img");
第16行         var position=myImg.position();
第17行         var positionLeft=position.left;
第18行         positionLeft-=20;
第19行         myImg.css("left",positionLeft+"px");
第20行       }
第21行     </Script>
第22行   </Body>
第23行 </Html>
```

例程2-51

```
第1行  <Script LANGUAGE="JavaScript">
第2行    $("#Box>img").animate({"left": '0px'}, "slow");
第3行  </Script>
```

$.animate()功能强大，jQuery还提供其他简捷的动画函数(方法)，如表8所示。

表2-8：动画方法

操作	描述
$.show()	显示隐藏的匹配元素。
$.show(speed[,function(){}])	以动画方式显示所有匹配元素，并可在动画完成后触发并执行函数function()(可选)。显示时根据指定的速度动态地改变每个匹配元素的高度、宽度以及不透明度等。
$.hide()	隐藏匹配元素。
$.hide(speed[,function(){}])	以动画方式隐藏所有匹配元素，并可在动画完成后触发并执行函数function()(可选)。隐藏时根据指定的速度动态地改变每个匹配元素的高度、宽度以及不透明度等。
$.toggle()	切换元素的可见状态，如可见，则切换为隐藏，如隐藏，则切换为可见。相当于$.show()和$.hide()的组合。
$.toggle(switch)	switch为true或false。如为true，则相当于$.show()，如为false，则相当于$.hide()。
$.toggle(speed[,function(){}])	切换元素可见状态(隐藏或显示)。相当于$.show()和$.hide()的组合。
$.slideDown(speed[,function(){}])	通过高度变化(向下增大)动态显示所有匹配元素，在动画完成后触发并执行可选函数function()。
$.slideUp(speed[,function(){}])	通过高度变化(向上减小)动态显示所有匹配元素，在动画完成后触发并执行可选函数function()。
$.slideToggle(speed[,function(){}])	通过高度变化来切换所有匹配元素，相当于$.slideDown()和$.slideUp()的组合。
$.fadeIn(speed[,function(){}])	通过透明度变化实现所有匹配元素的淡入效果(最终显示)，并在动画完成后触发并执行可选函数function()。
$.fadeOut(speed[,function(){}])	通过透明度变化实现所有匹配元素的淡出效果(最终隐藏)，并在动画完成后触发并执行可选函数function()。
$.fadeTo(speed,opacity[,function(){}])	通过透明度变化实现所有匹配元素的渐变到指定透明度opacity，并在动画完成后触发并执行可选函数function()。
$.animate(properties[,duration[,easing[,function(){}]]])	$.animate()函数的简单形式，如*例程progAnimation04*所示。properties用于设置CSS属性及其属性值(动画完成时的终值)，以对象形式申明，如*例程progAnimation04*第10行所示。duration用于设置动画速度，单位为毫秒，也可以使用slow、normal和fast。easing用于设置easing插件，easing的使用前提是正确应用easing插件，如如*例程progAnimation04*所示第4行所示，easing使用如第10行之“easeOutBounce”。function()为动画执行完毕后触发并执行的函数，与前述动画函数相同。
$.animate(properties,options)	$.animate()函数的完整形式，如例程2-54所示。有关options的相关参数，请参见表9所示。
$.stop(stop([clearQueue],[gotoEnd]))	停止在匹配元素上正在运行的动画。
$.delay(duration[,queueName])	适用于动画的时间延迟，duration为时间，单位为毫秒；queueName为动画对象名称，省略为系统默认动画，也可设置为自定义动画名称。
方括号表示可选参数。speed可设置为slow、normal和fast，也可设置为具体数值，单位毫秒。	

表2-9：自定义动画options参数选项

操作	描述
duration	取值String或Number，默认值为400毫秒。duration用于设置动画速度，单位为毫秒(数值)，也可以使用slow、normal和fast。

操作	描述
easing	取值String，默认值为swing。easing用于设置easing插件，easing的使用前提是正确应用easing插件。
queue	取值String或Boolean，默认值为true。queue用于设置动画是否进入动画序列。如果取值为false，则后续动画立即启动，否则动画先后启动。如例程2-54单击“按钮1”则是3个动画同时启动，而“按钮2”则是3个动画先后启动。
specialEasing	取值自定义对象。用于设置多个CSS属性以及对应变化模式(easing选项)，如例程2-55第14行。
step	每个动画每个CSS属性每一步执行时触发并执行的函数。
progress	每个动画被执行后所触发并执行的函数。
complete	成功执行完毕后触发并执行的函数。
done	动画结束时触发并执行的函数。
fail	动画执行失败所触发并执行的函数。
always	当动画完成或停止不成功所触发并执行的函数。

不少动画方法在动画完成都可以触发并执行一个函数，例程2-52是用动画函数slideDown()将一个隐藏的图片显示，并在显示完毕后执行一个函数显示“动画结束”4个汉字。例程2-53是对例程2-52改进，让动画完成后执行的函数能将图片的标题以动画的形式显示在图片底部(执行效果如图2-27所示)。

例程2-52

```
第1行  <Html>
第2行     <Head>
第3行        <Title>动画</Title>
第4行        <Script Language="JavaScript" src="./jQuery-1.9.1.min.js"></Script>
第5行     </Head>
第6行     <Body>
第7行        <Img src="3.gif" style="display:none" alt="湖入画 画如湖">
第8行        <Script LANGUAGE="JavaScript">
第9行           $("img").slideDown("slow",function(){
第10行              alert("动画结束");
第11行           });
第12行       </Script>
第13行    </Body>
第14行 </Html>
```

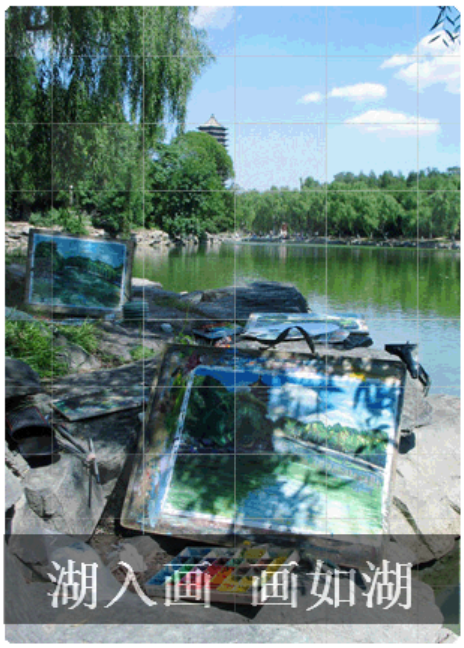


图2-27　例程2-53执行效果图

例程2-53

```
<Html>
  <Head>
    <Title>动画</Title>
    <Script Language="JavaScript" src="./jQuery-1.9.1.min.js"></Script>
    <Style type=text/css>
      #imgTitle{display:none;position:absolute;color:white;padding-top:15px;
                font-weight:bold;text-align:center;font-size:2em}
    </Style>
  </Head>
  <Body>
    <Img src="3.gif" style="display:none" alt="湖入画 画如湖">
    <Div id="imgTitle"></Div>
    <Script LANGUAGE="JavaScript">
      $("img").slideDown("slow",function(){//图片显示完毕后执行的函数
        var imgPosition=$(this).offset();//取得图片位置
        var imgHeight=$(this).height();//取得图片高度
        var imgWidth=$(this).width();//取得图片宽度
        var imgTitle=$(this).attr("alt");//取得图片标题
        $("#imgTitle")
          //设置DivCSS属性
          .css({"width":imgWidth,"height":"60px","background-color":"black",opacity:0.5})
          .offset({left:imgPosition.left-3,top:imgPosition.top+imgHeight-75})//设置图片位置
          .html(imgTitle)//设置元素内容
          .slideDown("slow");//动画显示该元素
      });
    </Script>
  </Body>
</Html>
```

$.animate()都支持一个叫easing的插件，能让动画更加自然，*例程progAnimation04*是一种弹簧弹动的效果(运行代码能更好地感知)。使用easing功能需要下载插件，其引用方式与jQuery类似，如第5行所示。

例程2-54是$.animate()完整形式的应用。当单击id名为btnGo1的按钮时，3个animate()函数同时执行，而单击id名为btnGo2的按钮时，3个animate()则是先后执行，即执行完毕一个函数后，再执行另外的animate()函数。区别就是queue参数的使用，如果不使用queue参数，默认值为true。

例程2-54

```
第1行  <Html>
第2行    <Head>
第3行      <Title>自定义动画$.animate()的应用</Title>
第4行      <Script Language="JavaScript" src="./jQuery-1.9.1.min.js"></Script>
第5行      <Style type="text/css">
第6行        #txtDiv1,#txtDiv2{border:1px solid red;width:80px}
第7行      </Style>
第8行    </Head>
第9行    <Body>
第10行     <button id="btnGo1">按钮1</button>
第11行     <button id="btnGo2">按钮2</button>
第12行     <div id="txtDiv1">程序设计</div><BR>
第13行     <div id="txtDiv2">思维体操</div>
第14行     <Script LANGUAGE="JavaScript">
第15行       $("#btnGo1").click(function(){
第16行        $("#txtDiv1").animate( { width: "400px"},{queue: false, duration: 1000 })
第17行          .animate( { fontSize: '5em' } ,{queue: false, duration: 1000 })
第18行          .animate( { borderLeftWidth:15 }, 1000);
第19行       });
第20行
第21行       $("#btnGo2").click(function(){
第22行        $("#txtDiv2").animate( { width: "400px"}, 1000 )
第23行          .animate( { fontSize: '5em' } , 1000 )
第24行          .animate( { borderRightWidth:15 }, 1000);
第25行       });
第26行     </Script>
第27行   </Body>
第28行 </Html>
```

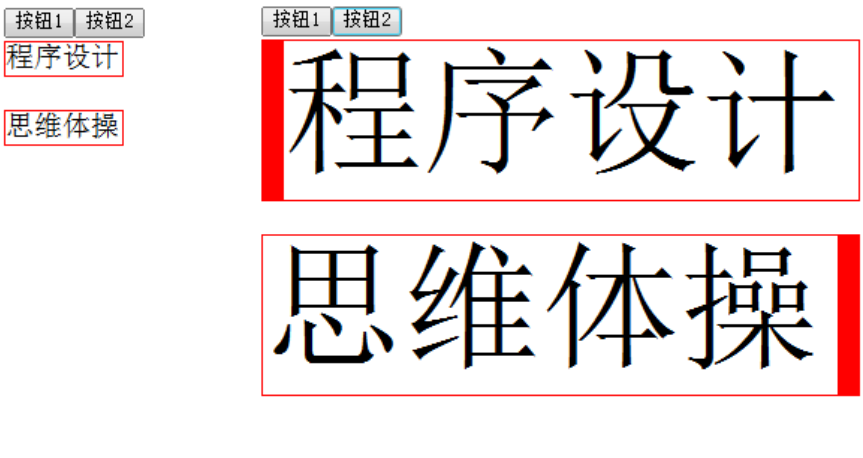


图2-28 例程2-54执行效果图

例程2-55是specialEasing的应用实例，其width是线性变化，而height是弹簧模式。使用specialEasing的前提是正确引用Easing插件，如第4行所示。

例程2-55

```
第1行  <Html>
第2行    <Head>
第3行      <Title>自定义动画之specialEasing应用</Title>
第4行      <Script Language="JavaScript" src="./jQuery-1.9.1.min.js"></Script>
第5行      <Script Language="JavaScript" src="./jquery.easing.1.3.min.js"></Script>
第6行    </Head>
第7行    <Body>
第8行      <Img src="3.gif" width="20">
第9行      <Script LANGUAGE="JavaScript">
第10行       $("img").animate(
第11行         {width:"304px",height:"426px"},
第12行         {
第13行           duration:5000,
第14行           specialEasing:{width: 'linear',height: 'easeOutBounce'}
第15行         }
第16行       );
第17行     </Script>
第18行   </Body>
第19行 </Html>
```

第五节 JavaScript进阶

1、正则表达式

第六节 网页模板

第七节 Flask进阶

1、请求对象

对于Web应用，与客户端发送给服务器的数据交互至关重要。在Flask中由全局的request(请求)对象提供相关信息。例程2-56是request的简单应用。

当在浏览器地址栏输入“http://127.0.0.1/?xyz=world&ab=hellohello&ABC=XYZCYXZZZZ”时，http://表示向服务器端请求http服务也就是Web服务；127.0.0.1表示访问目标的IP地址，也可以是域名地址。问号前的/表示路由或者路径此处为根文件夹或者路由。问号后的内容为URL参数。接收从浏览器传来的URL参数或Form数据，需要引入Flask的request模块，如例程第1行所示。如果要访问URL参数，需要使用request.args，这是一个Python字典对象，可以用字典方式访问，如例程第8-9行所示。访问Form数据，需要使用request.form，这也是一个Python字典对象，同样可以用字典方式访问，如例程18-19行所示。

在浏览器访问服务器时，还有其他一些信息随之而来，可以通过request.headers字典对象访问，如例程第15-16行所示。如其中User-Agent包含了浏览器运行的操作系统及其版本、浏览器类型及其版本；通过Referer可以看到向浏览器发起请求的URL等信息。

例程2-56

```
第1行  from flask import Flask,request,render_template
第2行
第3行  app=Flask(__name__)
```

```
第4行
第5行  @app.route("/")
第6行  def root():
第7行      content=""
第8行      for (i,j) in request.args.items():
第9行          content+=i+"->"+j+"<br>"
第10行     return render_template("test.html",content=content)
第11行
第12行 @app.route("/check",methods=["get","post"])
第13行 def checkData():
第14行     content=""
第15行     for (i,j) in request.headers.items():
第16行         content+=i+"->"+j+"<br>"
第17行     content+="#==============#<br>"
第18行     for (i,j) in request.form.items():
第19行         content+=i+"->"+j+"<br>"
第20行     return content
第21行
第22行 if __name__=="__main__":
第23行     app.run(host="0.0.0.0",port=80,debug=True)
```

例程2-57

```
第1行  <!--文件名：test.html-->
第2行  <!doctype html>
第3行  <html lang="zh">
第4行      <head>
第5行          <meta charset="UTF-8">
第6行          <title>Flask_Request</title>
第7行      </head>
第8行      <body>
第9行              {{content|safe}}
第10行             <form action="/check" method="post">
第11行                 姓名:<input type=text name="yourName"><br>
第12行                 性别:<input type=radio name="yourSex" value="男">男<input type=radio name="yourSex" value
="女">女<br>
第13行                 <select name="yourBirth">
第14行                     <option>1969</option>
第15行                     <option>1970</option>
第16行                     <option>1971</option>
第17行                     <option>1972</option>
第18行                 </select>
第19行                 <input type=submit>
第20行             </form>
第21行     </body>
第22行 </html>
```

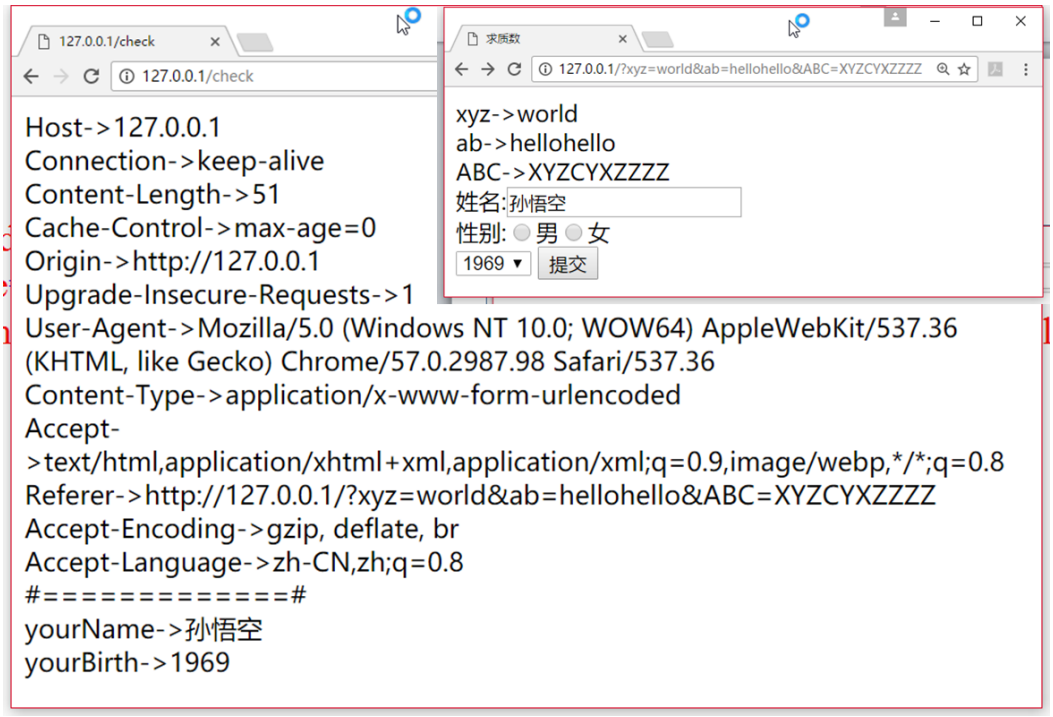


图2-29　例程2-56与例程2-57执行效果

2、会话对象

在一些网站的多个页面，经常需要登录后才能使用。如果这些页面每次访问都需要登录，将非常不友好，最好能“一次登录多页面多次使用”。如果要达到类似效果，就需要记录登录状态，让页面通过访问该状态判定是否已经登录，如果已经登录则正常使用，如果没有登录则转移到登录页面登录后使用。如图2-30左所示，当输入网址时，如果尚未登录则提示登录，登录后成功后，页面如图右所示。

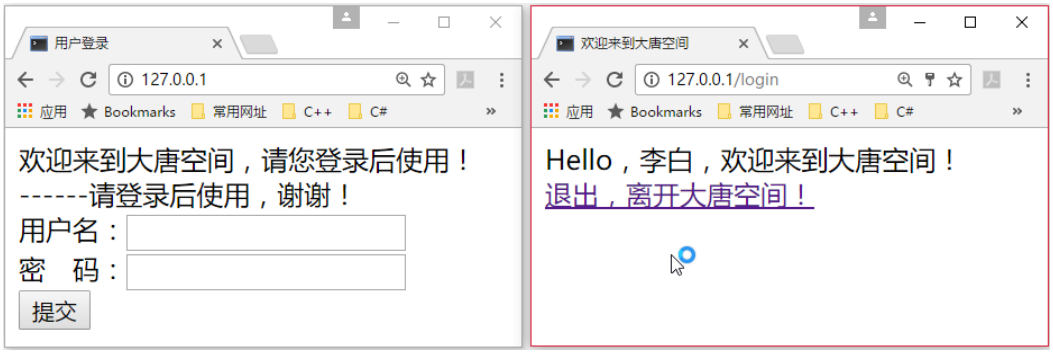


图2-30　例程2-58、例程2-59与例程2-60执行效果

例程2-58

```
第1行  <!--文件名：login.html-->
第2行  <!doctype html>
第3行  <html lang="zh">
第4行    <head><meta charset="UTF-8"><title>用户登录</title></head>
第5行    <body>
第6行      欢迎来到大唐空间，请您登录后使用！<br>
第7行      {{MSG}}
第8行      <form action="/login" method="post">
第9行        用户名：<input type=text name="userName"><br>
第10行       密　码：<input type=password name="userPassword"><br>
第11行       <input type=submit>
第12行     </form>
第13行   </body>
```

```
第14行  </html>
```

例程2-59

```
第1行  <!--文件名：welcome.html-->
第2行  <!doctype html>
第3行  <html lang="zh">
第4行     <head><meta charset="UTF-8"><title>欢迎来到大唐空间</title></head>
第5行     <body>
第6行        Hello，{{WHO}}，欢迎来到大唐空间！<br>
第7行        <a href="/logout">退出，离开大唐空间！</a>
第8行     </body>
第9行  </html>
```

session会话对象是字典。例程2-60第9行判断userName条目是否在session之中，如果不在，表明尚未没有成功登录，指定第11行代码，即进入login.html登录页面。当在登录页面(图2-30图左)输入用户名和密码后，执行服务器端/login对应代码，即第13-23行。如果用户名和密码正确，则执行第19行，即在session字典中设置条目(key)userName的值，否则执行第22行。

例程2-60

```
第1行  from flask import Flask,session,request,render_template
第2行
第3行  app = Flask(__name__)
第4行  nameDict={"李白":"libai123","杜甫":"dufu345","王维":"wangwei567"}
第5行  #假设的用户名和密码表，密码不是好密码，仅是示意
第6行
第7行  @app.route('/')
第8行  def index():
第9行     if 'userName' in session:
第10行       return render_template("welcome.html",WHO=session['userName'])
第11行    return render_template("login.html",MSG="------请登录后使用，谢谢！")
第12行
第13行 @app.route('/login', methods=['GET', 'POST'])
第14行 def login():
第15行    if request.method == 'POST':
第16行       whoLogin=request.form['userName']
第17行       whoPWD=request.form["userPassword"]
第18行       if whoLogin in nameDict and nameDict[whoLogin]==whoPWD:
第19行          session['userName'] = whoLogin
第20行          return render_template("welcome.html",WHO=whoLogin)
第21行       else:
第22行          return render_template("login.html",MSG="------密码或用户名错误。")
第23行    return render_template("login.html",MSG="")
第24行
第25行 @app.route('/logout')
第26行 def logout():
第27行    session.pop('userName', None) #删除userName条目
第28行    return render_template("login.html",MSG="------请您登录后再次使用！")
第29行
```

```
第30行 if __name__=="__main__":
第31行     #设置secret_key，可以用import os;os.urandom(24)随机生成24字节字符串
第32行     app.secret_key = b'\xe4D\x1f\x88\xc2\xa1b\x00$\xc2PA\xd8\x8b\x0bd\\\x89IG\x8e@\xec\xff'
第33行     app.run(host="0.0.0.0",port=80,debug=True)
```

3、重定向与URL构造

观察图2-30左，会发现显示内容是welcome.httml，但地址栏显示为http://127.0.0.1/login，似乎还在登录状态。如需修正，可按例程2-61第20行修改，其redirect()函数用于重定向，即从当前链接转移到指定链接。redirect()参数为链接，可以按route()设置，也可以设定为其他链接或者地址，甚至是非本网站地址，只要是合规地址即可。

例程2-61

```
第1行  from flask import Flask,session,request,render_template,redirect,url_for
第2行
第3行  app = Flask(__name__)
第4行  nameDict={"李白":"libai123","杜甫":"dufu345","王维":"wangwei567"}
第5行  #假设的用户名和密码表，密码不是好密码，仅是示意
第6行
第7行  @app.route('/')
第8行  def index():
第9行      if 'userName' in session:
第10行         return render_template("welcome.html",WHO=session['userName'])
第11行     return render_template("login.html",MSG="------请登录后使用，谢谢！")
第12行
第13行 @app.route('/login', methods=['GET', 'POST'])
第14行 def login():
第15行     if request.method == 'POST':
第16行         whoLogin=request.form['userName']
第17行         whoPWD=request.form["userPassword"]
第18行         if whoLogin in nameDict and nameDict[whoLogin]==whoPWD:
第19行             session['userName'] = whoLogin
第20行             return redirect("/") #此处修改
第21行         else:
第22行             return render_template("login.html",MSG="------密码或用户名错误。")
第23行     return render_template("login.html",MSG="")
第24行
第25行 @app.route('/logout')
第26行 def logout():
第27行     #remove the username from the session if it's there
第28行     session.pop('userName', None)
第29行     return redirect(url_for("login"))#此处修改
第30行
第31行 if __name__=="__main__":
第32行     #设置secret_key，可以用import os;os.urandom(24)随机生成24字节字符串
第33行     app.secret_key = b'\xe4D\x1f\x88\xc2\xa1b\x00$\xc2PA\xd8\x8b\x0bd\\\x89IG\x8e@\xec\xff'
第34行     app.run(host="0.0.0.0",port=80,debug=True)
```

观察例程第29行，会发现redirect函数中使用url_for()函数，该函数的功能是构造URL，其第一个参数为调用函数名称。例程2-62是url_for()的更多示例。

例程2-62

```
第1行 from flask import Flask,redirect,url_for,render_template
第2行
第3行 app = Flask(__name__)
第4行
第5行 @app.route("/")
第6行 def index():
第7行     return "欢迎来到Flask乐园!"
第8行
第9行 @app.route('/userInfo/<userName>', methods=['GET','POST'])
第10行 def userInfo(userName=""):
第11行     return userName
第12行
第13行 @app.route("/login")
第14行 def sign_in():
第15行     print(url_for("userInfo", userName="libai", test="TEST",_method="POST"))
第16行     return redirect(url_for("userInfo", userName="libai", test="TEST",_method="POST"))
第17行
第18行 @app.route("/file")
第19行 def gotoFile():
第20行     return render_template("libai.html",WHO="李白")
第21行
第22行 if __name__=="__main__":
第23行     app.run(host="0.0.0.0",port=80,debug=True)
```

上例中的libai.html代码如例程progUrlFor01所示。例程第5行代码{{url_for('static',filename='jQuery-3.1.1.min.js')}}的含义是相对于设定static文件夹之下的jQuery-3.1.1.min.js，而{{url_for('static',filename='img/libai.jpg',_external=True)}}则是同样是相对于设定static文件夹之下的img文件夹下的libai.jpg，同时_external=True则表示该URL转换为绝对地址。可通过网页源码查看二者之间的区别。另外，关于静态文件夹设置、模板文件夹路径设置、URL文件夹设置等，请参看后续“文件夹设置”。

例程2-63

```
第1行 <!--文件名：libai.html-->
第2行 <!doctype html>
第3行 <html lang="zh">
第4行     <head><meta charset="UTF-8"><title>欢迎来到大唐空间</title></head>
第5行     <script type="text/javascript" src="{{url_for('static',filename='jQuery-3.1.1.min.js')}}"></script>
第6行     <body>
第7行         Hello，{{WHO}}！欢迎来到大唐空间！<br>
第8行         <img src="{{url_for('static',filename='img/libai.jpg',_external=True)}}">
第9行     </body>
第10行     <script type="text/javascript">
第11行         $("body").css("background-color","gray");
第12行     </script>
第13行 </html>
```

4、反馈及其cookie对象

在前述例程中，反馈(response)对象已多次默认使用，如例程2-64第5-12行所示。当浏览器向服务器发起请求时(request)，可以分析请求的相关内容，如例程2-56所示。同理，服务器返回内容，也会包含大量信息供浏览器分析处理。不过前述例程都是默

认返回，服务器按默认处理。对于第5-12行代码，服务器默认返回类型是text/html即mimetype的值为text/html，其status code值为200，即服务器成功处理请求。如果status code为404，则表示服务器没有找到相关资源。当浏览器接受到相关信息后，根据mimetype的类型进行相关处理。如第28行的ding()和第34行的dingding()，虽然代码几乎相同，但由于mimetype不同，因此一个被处理成网页一个被处理成文本而没有网页效果。response可以通过Flask的make_response对象深入定制以满足不同需求，如例程第16行、第22行分别用于文本文件或图片下载。当浏览器看到这些相关信息，将自动启动下载而不是在浏览器窗口呈现。

例程2-64

```
第1行  from flask import Flask,make_response
第2行
第3行  app = Flask(__name__)
第4行
第5行  @app.route("/")
第6行  def Index():
第7行      content="""
第8行      <a href='/image'>点击直接下载图片</a><br>
第9行      <a href='/text'>点击直接下载文本文件</a><br>
第10行     <a href='/ding'>网页内容</a><br>
第11行     <a href='/dingding'>文本内容</a><br>
第12行     """
第13行     return content #默认生成mimetype为text/html
第14行
第15行 @app.route("/text")
第16行 def TextDownLoad(): #文本文件下载
第17行     res = make_response("桃花潭水深千尺不及汪伦送我情")
第18行     res.headers["Content-Disposition"] = "attachment; filename=libai.txt;"
第19行     return res
第20行
第21行 @app.route("/image")
第22行 def Image(): #图片文件下载
第23行     res = make_response(app.send_static_file("libai.jpg"))
第24行     res.headers["Content-Disposition"] = "attachment; filename=libai.jpg;"
第25行     return res
第26行
第27行 @app.route('/ding')
第28行 def ding():
第29行     rsp = make_response("<hr><span style='font-size:40px'>dong</span>") #输出内容
第30行     rsp.mimetype = 'text/html' #设置属性
第31行     return rsp
第32行
第33行 @app.route('/dingding')
第34行 def dingding():
第35行     rsp = make_response("<hr><span style='font-size:40px'>dong</span>") #输出内容
第36行     rsp.mimetype = 'text/plain' #设置属性
第37行     return rsp
第38行
第39行 if __name__=="__main__":
```

```
第40行    app.run(host="0.0.0.0",port=80,debug=True)
```

当浏览某些网站时，浏览器会根据服务器反馈代码的要求在本地硬盘存储一定信息，这些信息称之为cookie，下图是笔者Chrome浏览器cookie信息查看结果。这些信息一般说来具有安全性。浏览器端的cookie和服务器端的session相似，常用于保存网站运行状态。比如：网站登录是网站常见操作，如果每次都输入用户名会比较啰嗦，因此可以考虑将用户名保存在cookie之中。另外，电子商务的购物车操作尝尝需要在多个页面选购，如果都存储在服务器，会对服务器造成较大负担，可以考虑将其存储在cookie之中。进入购物车或者结算时，从cookie统一取出全部商品。

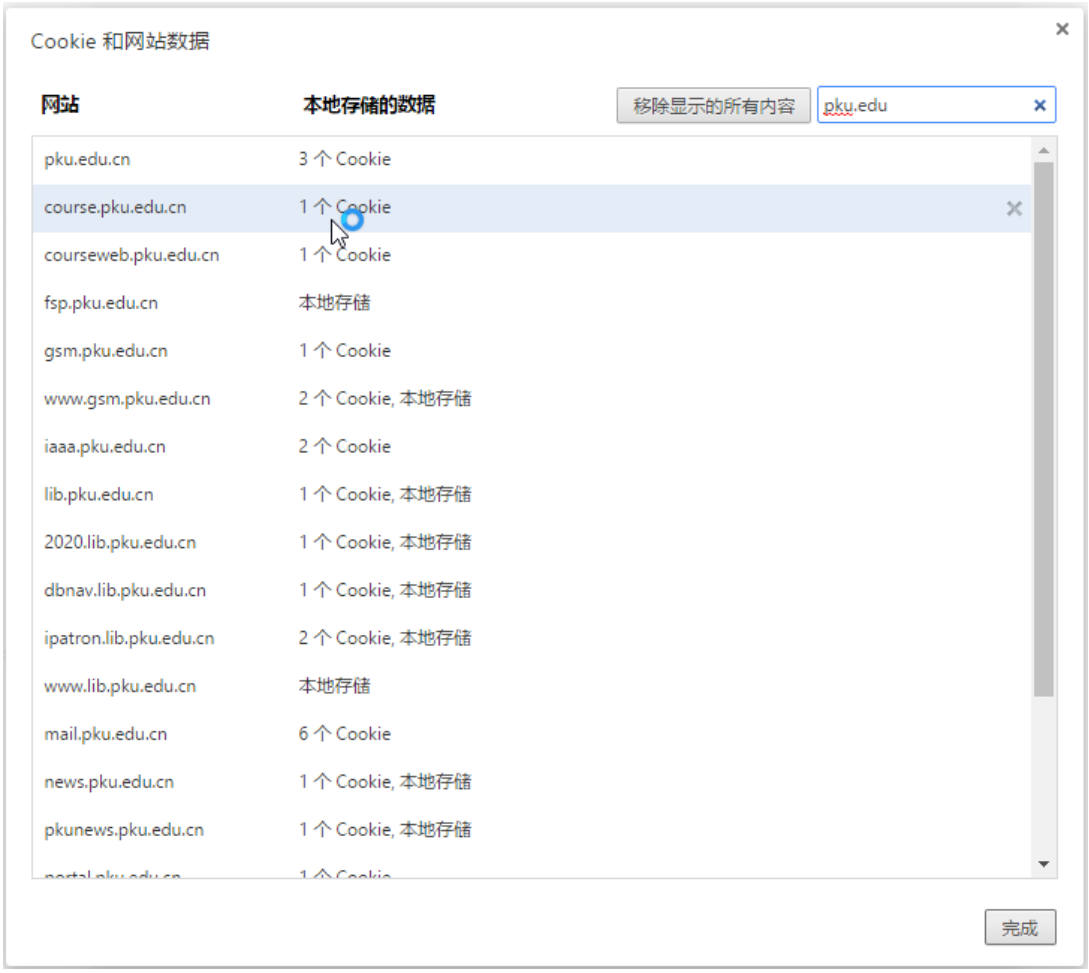


图2-31　Chrome浏览器cookie信息查看

在例程2-65中第11-12行分别是设置cookie。当浏览器执行此处时，将自动将该信息写入到浏览器端的cookie存储区域，除非浏览器设置不允许写入cookie信息等。当执行@app.route("/first")也即执行first.html时，可以通过例程2-66第10行读取到所有cookie信息，同时也可以通过第12行的方式写入cookie信息。当执行@app.route("/second")时，可以获得浏览器端所有cookie信息，并通过第26-27行的方式进行读取。

例程2-65

```
第1行   from flask import Flask,make_response,request,render_template
第2行
第3行   app = Flask(__name__)
第4行
第5行   @app.route("/")
第6行   def Index():
第7行       content="""<a href='/first'>访问first</a><br>
第8行       <a href='/second'>访问second</a>
第9行       """
第10行      res=make_response(content)
第11行      res.set_cookie("poet","Libai")
第12行      res.set_cookie("poetDynasty","Tang")
```

```
    res.mimetype="text/html"
    return res

@app.route("/first")
def First():
    return render_template("first.html")

@app.route("/second")
def Second():
    content =""
    for (i,j) in request.headers.items():
        content+=i+"->"+j+"<br>"
    content+="<br>request.cookies是字典，可通过字典方式访问<br>"
    for (i,j) in request.cookies.items():
        content+=i+"->"+j+"<br>"
    return content

if __name__=="__main__":
    app.run(host="0.0.0.0",port=80,debug=True)
```

例程2-66

```
<!--文件名：first.html-->
<!doctype html>
<html lang="zh">
    <head><meta charset="UTF-8"><title>欢迎来到大唐空间</title></head>
    <script type="text/javascript" src="{{url_for('static',filename='jQuery-3.1.1.min.js')}}"></script>
    <body>
        <div id="cookieBox"></div>
    </body>
    <script type="text/javascript">
        var cookie=document.cookie; //JS获得该网页的cookie信息
        $("#cookieBox").html(cookie);
        document.cookie="visitTime="+(new Date()).getTime();//JS设置cookie
        //获得从1970.1.1开始的毫秒数
    </script>
</html>
```

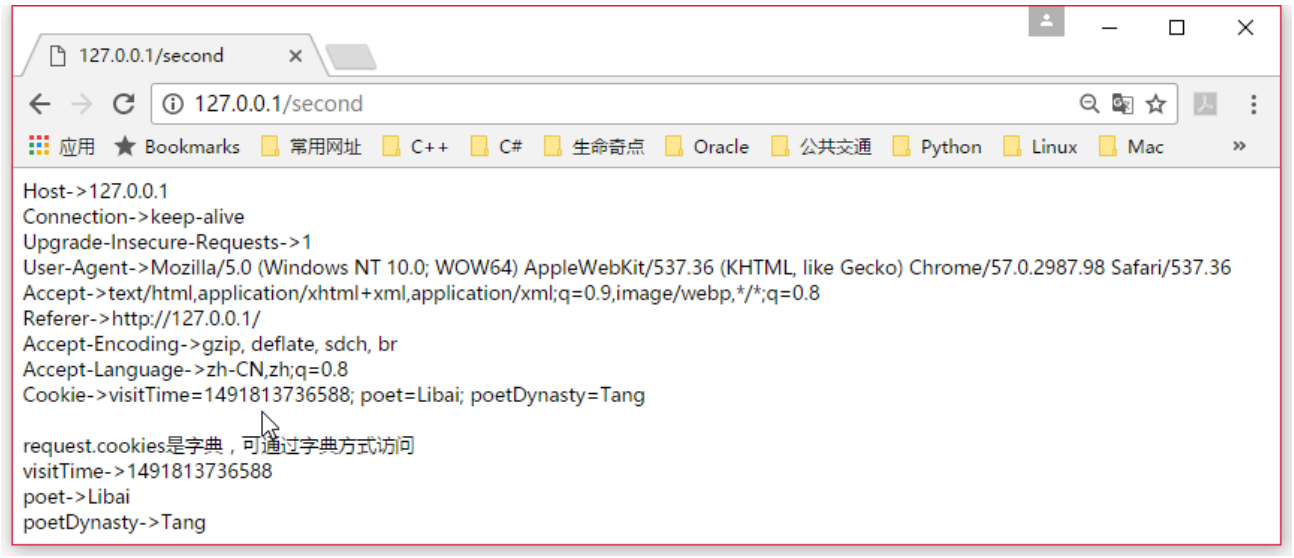


图2-32　Chrome浏览器cookie信息查看

5、错误处理

错误总是难免，比如：错误链接、服务器故障等等，最常见的错误是404和500错误，对应分别是“资源找不到”和“服务器内部错误”。网址输入错误，资源已经删除都可能导致404错误，这是最常见错误。例程2-67第9行代码@app.errorhandler(404)定义404处理路由，第11行通过render_template()执行模板网页，并将404状态码随之输出。注意：使用render_template()时一般没有状态码，其默认值为200。在设计错误页面时，除了提示信息外，一般需要有返回首页或者其他页的链接，防止用户在该网页不知如何行为。另外，一些公益组织将404页面设置为寻亲页面，这非常好。虽然对于一个网站来讲发生404概率不高，但难免会发生，此时展示寻亲信息，有助于企业社会形象。同时很多网站参与，有助于寻亲。

例程2-67

```
第1行  from flask import Flask,render_template
第2行
第3行  app = Flask(__name__)
第4行
第5行  @app.route("/")
第6行  def Index():
第7行      return "欢迎光临！<br><a href='/index'>继续访问</a>"
第8行
第9行  @app.errorhandler(404)
第10行 def page_not_found(error): #注意：error不能省略
第11行     return render_template('page_not_found.html'), 404
第12行
第13行 if __name__=="__main__":
第14行     app.run(host="0.0.0.0",port=80,debug=True)
```

例程2-68

```
第1行  <!--文件名：page_not_found.html-->
第2行  <!doctype html>
第3行  <html lang="zh">
第4行      <head><meta charset="UTF-8"><title>欢迎光临</title></head>
第5行      <body>
第6行          非常抱歉，您需要的信息不存在！<br>
第7行          <a href="/">返回首页</a>
第8行      </body>
```

```
第9行  </html>
```

6、文件夹设置

在Flask的Web开发中，一般需要在Python代码所在文件夹之下建立static和templates两个文件夹。static用于存放静态文件，包括：静态网页、CSS文件、图片文件或JavaScript文件等等。templates用于存放网页模板文件。这两个功能的文件夹，都可以在程序中用参数设定。如例程2-69所示。在本例中，静态图片放在JingTai文件夹下，JavaScript文件此处为jQuery-3.1.1.min.js放在JingTai之下的JS文件夹下。

在例程第3行代码中，设定static_folder为当前Python代码所在文件夹下的JingTai文件夹下，这即意味着所有静态文件都将在该文件夹下查找。设置该文件夹时，支持路径相关符号，如：/表示根文件夹，.(英文句点)表示当前文件夹，..(双英文句点)表示父文件夹。同理，template_folder用于设定模板文件所在文件夹，所有模板文件都将在该文件夹寻找，其设置方法与static_folder相同。

例程2-69

```
第1行  from flask import Flask,render_template,url_for
第2行
第3行  app = Flask(__name__,static_folder="JingTai",static_url_path="/ABC",template_folder="MuBan")
第4行
第5行  @app.route("/")
第6行  def Index():
第7行      content="""欢迎光临！<br>
第8行      <a href="/img">静态文件测试</a><br>
第9行      <a href="/templet">模板文件测试</a><br>
第10行     """
第11行     return content
第12行
第13行 @app.route("/img")
第14行 def Image():
第15行     return app.send_static_file("staticFilePath.html")
第16行
第17行 @app.route("/templet")
第18行 def TestFile():
第19行     return render_template("testFilePath.html")
第20行
第21行 if __name__=="__main__":
第22行     app.run(host="0.0.0.0",port=80,debug=True)
```

在模板文件中，可以用url_for()函数生成静态文件，如例程2-70第6行所示，生成的文件相对于静态文件夹的位置，支持子文件夹等等。对于模板文件，静态文件可以用例程2-71第5、7行形式。

例程2-70

```
第1行  <!--文件名：testFilePath.html-->
第2行  <!doctype html>
第3行  <html lang="zh">
第4行      <head><meta charset="UTF-8"><title>文件位置测试</title></head>
第5行      <body>
第6行          <img src="{{url_for('static',filename='DuFu.jpg',_external=True)}}">
第7行      </body>
第8行  </html>
```

静态网页不支持url_for()函数，相关资源文件只能固定，如下例的第5、7行所示。注意：Python代码所在文件夹下并没有ABC文件夹，该文件夹名字来自static_url_path设置(参见例程2-69第3行)。可以认为，static_url_path是static_folder的别名。如果省略该参数，则须写上static_folder文件夹的设置，如<img src="/ABC/LiBai.jpg">则须写成<img src="/JingTai/LiBai.jpg">。另外，static_url_path要求以/开始。

例程2-71

```
第1行  <!--文件名：staticFilePath.html-->
第2行  <!doctype html>
第3行  <html lang="zh">
第4行     <head><meta charset="UTF-8"><title>文件位置测试</title></head>
第5行     <script type="text/javascript" src="/ABC/JS/jquery-3.1.1.min.js"></script>
第6行     <body>
第7行           <img src="/ABC/LiBai.jpg">
第8行           <script type="text/javascript">
第9行              $("body").css("background-color","gray");
第10行          </script>
第11行    </body>
第12行 </html>
```

第八节　小结