第一章 JavaScript基本语法

1.1 JavaScript 概述

1、为什么需要JavaScript?

JavaScript 诞生于 1995 年，主要是进行用户输入的合法性验证。
在 1995 年之前，**Web 页面的一些验证工作都是由服务器端的语言来实现，这就要求用户输入的数据必须先通过网络传输到服务器端，服务器端进行相应的处理后**，再将结果反馈给客户端。

为了避免给服务端增加压力，出现了JavaScript，可以提前在前端校验数据是否合法，再允许请求访问服务端。

2、JavaScript执行原理

JavaScript 是**一种轻型的、解释性的脚本语言**，是一种由浏览器内的解释器执行的程序语言
当客户端向服务器端请求页面时，服务器端将整个页面包含 JavaScript 的脚本代码发送到客户端，**浏览器从上往下逐行读取并解析其中的 HTML 或脚本代码。**

3、JavaScript代码的执行地方

（1）内嵌式的使用

标签中直接编写脚本程序代码

演示示例：

```
<body>
   <script  type="text/javascript">
      document.write("<h2> 欢迎来到 JavaScript 课堂 </h2>");
   </script>
</body>
```

（2）导入式

脚本程序代码放置在一个单独的文件中，然后在网页文件中引用这个脚本程序文件

```
<script src="./js/index.js"></script>
```

注意：

带有 src 属性的 <script> 标签**不应该**在 <script> 标签之间包含任何额外的 JavaScript 代码，**否则嵌入的代码会被忽略**

（3）嵌入式

将脚本程序代码作为某个 HTML 页面元素的事件属性值或超链接的 href 属性值

```
<a href="javascript:document.write('<h2>欢迎来到这里! </h2>');">22</a>
```

1.2 JavaScript 核心语法

1、变量

（1）变量的三种使用方式：
（1）先声明再赋值　如：var message;　message="hi";
（2）同时声明和赋值变量　如：var message="hi";
（3）不声明直接赋值　如：message="hi";

示例：

```
<script>
        // 分两步：
        var hi;
        hi = "hello";
        alert(hi);
        // 不声明直接赋值使用。【不推荐这种声明方式，容易出错】
        message = "11";
        alert(message);
        // 直接声明并赋值
        var his = "leo";
        alert(his);
</script>
```

需要注意的是：**变量可以不经声明而直接使用**，但是这种方法很容易出错，因此**不推荐使用。**对变量的输出测试可以采用 alert() 弹出对话框的方式。

（2）变量名的命名规则：
1）第一个字符必须是一个字母、下划线（_）或一个美元符号（$）
2）其他字符可以是字母、下划线、美元符号或数字
3）区分大小写
4）不能与关键字同名，如 while、for 和 if 等

运算符的简单使用：

示例：

```
<body>
   <script  type="text/javascript">
      var  x=2;
      var  y=3;
      var  z=x+y;
      alert("x="+x+",y="+y+",z="+z);
   </script>
</body>
```

2、数据类型

JavaScript 中有 5 种简单数据类型，也称为基本数据类型：

1. undefined
2. null
3. boolean

4. number
5. string

   另外还有一种复杂数据类型——object对象类型

   由于**JavaScript中的变量是弱类型**，可通过**typeof操作符获取变量的数据类型**

数据类型	数据值	typeof
number类型	浮点数、整数	number
boolean类型	true、false	boolean
string类型	单引号或双引号引起来的若干字符	string
null类型	只存在一个值null，表示对象不存在	object
undefined类型	只存在一个值undefined，表示未赋值或未声明的变量	undefined

   示例：

```
// number类型
    var num1 = 10;
    var num2 = 20.2;
    console.log(typeof(num1));
    console.log(typeof(num2));

    // 字符串类型
    var a = "abc";
    console.log(typeof(a));

    // boolean类型
    var c = true;
    console.log(typeof(c));

    // null类型
    var d = null;
    console.log(typeof(d)); //得到object

    // undefined 未赋值或未定义的意思
    var e;
    console.log(typeof(e)); //未赋值
    console.log(typeof(g)); //未定义
```

3、JavaScript 注释

JavaScript 与很多语言一样，如 Java、C# 都支持同样的注释形式：
单行注释：//
多行注释：　/* 注释内容　*/

示例：

```
// 单行注释
// var a = 10;
/* 多行注释 */
/*
    var b = 20;
    var c = 30;
*/
```

1.3 顺序结构和选择结构

顺序结构，顾名思义就是程序按照语句出现的先后顺序依次执行。

```
<body>
   <script  type="text/javascript">
      document.write(" 程序开始执行……<br/>");
      document.write(" 程序正在执行中……<br/>");
      document.write(" 程序执行完毕……<br/>");
   </script>
</body>
```

选择结构：需要根据特定的条件执行不同的语句。
JavaScript中选择结构使用if语句和switch语句。

if 语句有 3 种形式：**单分支、双分支和多分支。**

1. 单分支结构

   语法：

   ```
   if( 条件表达式 ) {
      语句或语句块
   }
   ```

   示例：

   ```
   <body>
      <script  type="text/javascript">
         var  undf;
         if(typeof  undf=="undefined")  {
            undf="Hello  World !  ";
         }
         document.write(" 名称是："+undf);
      </script>
   </body>
   ```

2. if双分支结构

   语法：

```
if( 条件表达式 ) {
   语句或语句块 1
}  else{
   语句或语句块 2
}
```

示例：

```
   <script  type="text/javascript">
      var  x=-4,y;
      if(x>0){
         y=x;
      }else{
         y=-x;
      }
      document.write(x+' 的绝对值是：'+y);
   </script>
```

另一个结构： **条件1?结果1:结果2;** 的方式可以得到：

```
    var a = 10;
    var b = 20;
    var max = a > b ? a : b;
    alert(max);
```

3. if 多分支语句

语法：

```
if( 条件表达式 1){
   语句或语句块 1
}
else  if( 条件表达式 2){
   语句或语句块 2
}......
else  if( 条件表达式 n){
   语句或语句块 n
}
else{
   语句或语句块 n+1
}
```

示例：

```
    var date = new Date();
    var hour = date.getHours();
    if(hour<=11){
        document.write(" 早上好 ");
     }else  if(hour<=18){
        document.write(" 下午好 ");
     }else{
        document.write(" 晚上好 ");
     }
```

4. switch 语句

   语法：

```
switch（表达式）{
   case  取值 1：语句或语句块 1;    break;
   case  取值 2：语句或语句块 2;    break;
   ......
   case  取值 n：语句或语句块 n;    break;
   default:  语句或语句块 n+1;    break;
}
```

   示例：

```
      var  time=new  Date();
      var  week=time.getDay();
      var  weekstr;
      switch(week){
         case  1: weekstr=" 一 ";
         case  2: weekstr=" 二 ";
         case  3: weekstr=" 三 ";
         case  4: weekstr=" 四 ";
         case  5:
            weekstr=" 五 ";
            document.write(" 今天是星期 "+weekstr+",  努力工作吧！ ");
            break;
         default:
            document.write(" 今天是周末，放松一下吧！ ");
            break;
      }
```

1.4 循环结构

JavaScript中，循环结构有 while 循环、do-while 循环和 for 循环

1. while 循环语句：

```
while（条件表达式）{
   语句或语句块
}
```

   示例：

```
   <table border="1" width="200">
       <tr align="center">
           <td>摄氏温度</td>
           <td>华氏温度</td>
       </tr>
       <script>
           var s = 0; //摄氏温度
           var h = 0; //华氏温度
           while(s<=180)
```

```
            {
                 //摄氏温度转换为华氏温度
                 h = s*9/5.0 + 32;
                document.write("<tr align='center'><td>"+s+"</td><td>"+h+"
</td></tr>");
                 //退出循环
                 s += 20;
            }
        </script>
    </table>
```

2. do-while循环语句：

语法：

```
do{
   语句或语句块
}while( 条件表达式 );
```

示例：

```
   <script  type="text/javascript">
      var  i=1;
      var  num=1;
      do{
         i++;
         num=num*i;
      }while(i<5);
      document.write("i="+i+",num="+num);
   </script>
```

3. for循环语句

语法：

```
for( 初始化表达式 ; 循环条件表达式 ; 循环后的操作表达式 ){
   语句或语句块
}
```

示例：

```
        <script>
            var s = 0; //摄氏温度
            var h = 0; //华氏温度

            for(;s<=180;s+=20)
            {
                 //摄氏温度转换为华氏温度
                 h = s*9/5.0 + 32;
                document.write("<tr align='center'><td>"+s+"</td><td>"+h+"
</td></tr>");
            }
        </script>
```

4. break语句和continue语句

   JavaScript跳转语句：**break语句和continue语句**
   break语句用于中断循环
   continue语句用于跳过本次循环要执行的剩余语句，然后开始下一次循环

```
var  i=1;
   var  sum=0;
   while(i<=20){
      if(i%2!=0){
         i++;
         continue;   // 是奇数就结束本次循环，开始下一次循环
      }
      sum=sum+i;
      if(sum>30){
         document.write(" 当加到："+i+" 时，累加和已经超过 30");
         break;      // 累加和超过 30，跳出循环
      }
      i++;
   }
```

九九乘法口诀：

```
        for(var i=1;i<=9;i++)
        {
            for(var j=1;j<=i;j++)
            {
                document.write(i+"*"+j+"="+i*j + "    ");
            }
            document.write("<br>");
        }
```

第二章 JavaScript 函数与事件

2.1 JavaScript自定义函数

函数就是为了完成程序中的某些特定功能而进行专门定义的一段程序代码。
函数包括**自定义函数**和**系统函数**
使用函数可以实现特定的功能，在使用函数前必须对函数进行定义，代码放置在 标签之间

1、函数的定义

函数定义语法：

```
function  函数名 ( 形式参数 1, 形式参数 2,…, 形式参数 n){
   语句
}
```

说明：

1. 使用 function 关键字。function 后是函数名，JavaScript 中的函数不必说明返回值的类型。
2. 函数的命名规则与变量名的命名规则相同。
3. 函数名后的 () 可以包含若干参数，也可以选择不带任何参数。
4. 最后是一对 {}，包含具体实现程序中的某些特定功能的若干语句或脚本代码

函数是不会自动执行的，**调用一个函数的方法是使用函数名称**，并且在函数名后用括号包含所需要传入的参数值。

调用函数的语句也需要放置在里

示例：

```
function getArea(width,height)
{
    var area = width * height;
    document.write(area);
}
getArea(10,20.5);
```

2、函数的参数

函数参数作用：调用函数时将数据传递给被调函数的方式
JavaScript中函数参数的**特殊性**：
（1）函数声明了参数，调用时也可以不传递参数，这在其他编程语言中是会出现编译错误的
（2）不管函数声明时有多少个参数，调用函数时可以传递若干个参数值给函数，并且实际传递的参数值还可以在函数内部获得
（3）在函数被调用时，一个 arguments 对象就会被创建，它只能使用在函数体中，以数组的形式来管理函数的实际参数

使用示例：

```
<script>
 function getSalary()
 {
     var count = arguments.length;
     var sum = 0;
     for(var i=0;i<count;i++)
     {
         sum += arguments[i];
     }
     document.write("公司总人数："+count+"，总工资："+sum);
 }

</script>

<h1>A公司的工资：</h1>
<script>
    getSalary(2000,3666,8888,6666,7777);
</script>
 <h1>B公司的工资：</h1>
```

```
    <script>
        getSalary(2000,3666,8888,6666,7777,9999);
    </script>
```

3、返回值

函数的返回值能够将一个函数内部产生的结果返回给外部语句使用。
实现函数返回值的语句是 return。

```
return  返回值 ;
```

示例：

```
    function getArea(width,height)
    {
        var area = width * height;
        // document.write(area);
        return area;
    }
    var area = getArea(10,20.5,30,50);
    document.write(area);
```

思考：

（1）函数体内，return 语句后一定要有返回值吗?

答：不一定，可以返回空值。

（2）请说出以下函数中两条 return 语句的作用?

```
function  getArea(width,height){
     if(width<=0||height<=0){
      return;
     }
   return  width*height;
}
```

第一个返回为空，第二个返回结果。

4、匿名函数

匿名函数就是没有名字的函数，也被称为拉姆达函数，是一种使用 JavaScript 函数的强大方式。

语法：

```
语法一：
（function （形式参数列表）{
   语句
 }）（实际参数列表）；

语法二：
var  变量 =（function （形式参数列表）{
   语句
} ）；
变量（实际参数列表）；
```

示例：

```
    // 匿名函数的第一种结构：
        (function(width,height){
            var area = width * height;
            document.write(area);
        })(10,20);

    // 匿名函数的第二种结构：
    var getArea = (function(width,height){
        var area = width*height;
        return area;
    });
    var sum = getArea(10,20);
    document.write(sum);
```

5、变量的作用域

（1）变量的作用域主要分为全局变量和局部变量两种
（2）全局变量是在函数体外部声明的，可以在任何地方，包括函数的内部使用
（3）局部变量是在函数体内声明的，只能在函数体内使用。局部变量随着函数的结束而消失

注意：

如果全局变量和局部变量出现重名的情况，**局部变量优先，即无论局部变量的值怎么改变，全局变量的值不会受到影响**

使用示例：

```
    var width = 20;
    function getAreas()
    {
        //局部变量 =》 局部变量修改值，对全局变量没有影响
        var width = 30;
        document.write(width);
    }
    getAreas();
    document.write(width);
```

2.2 JavaScript系统函数

1、parseInt() 函数

parseInt函数将字符串转换为整数。它从字符串的开头开始解析，在第一个非整数位置停止解析并返回前面读到的所有整数。**如果字符串不是以整数开头，将返回NaN(Not a Number：非数字值)**

参考示例：

parseInt(string)	
字符串	**结果**
“150cats”	150
“cats”	NaN
“6”	6
“-6”	-6
“6.56”	6

2、parseFloat() 函数

parseFloat()函数和parseInt()函数类似，只不过它是返回一个浮点数

参考示例：

parseFloat(string)****	
字符串	**结果**
“route66.5”	NaN
“8.5dogs”	8.5
“6”	6
“6.56”	6.56
“.7”	0.7

3、isNaN() 函数

isNaN() 函数用于判断参数是否是NaN（不是数字）。如果是 NaN， 那么 isNaN 函数返回 true ，否则返回 false

参考示例：

isNaN**(参数)**	
参数	**结果**
‘134’	false
'2a34'	true
'2.34'	false
' ' (空格)	false
'wh'	true

4、eval() 函数

eval() 函数运行是以字符串形式表示的 JavaScript 代码串，并返回执行代码串后的结果

示例：

```
<script type="text/javascript">
    function calc() {
        var express=document.getElementById("info").value;
        var result=eval(express);
        alert(" 输入在文本框中的表达式的结果是："+result);
    }
</script>
<input type="text" id="info"/>
<input type="button" value=" 计算 " id="btn" onClick="calc()"/>
```

2.3 JavaScript 事件

1、事件

JavaScript 是基于对象、采用事件驱动的脚本语言
事件：用户使用鼠标或键盘在浏览器窗口或页面元素上执行的操作
事件源：产生事件的元素
事件处理程序：对事件进行处理的程序或函数
事件驱动：将一段程序代码与某个事件源上发生的事件进行绑定，当触发此事件，浏览器就会自动执行与之绑定的程序代码

2、事件与处理程序的绑定

在JavaScript 中，有两种方式将对象事件与处理程序代码进行绑定：
（1）在事件源对象所对应的 HTML 页面标签上增加一个要处理的事件属性，让事件属性值等于处理该事件的函数名或程序代码

示例：

```
<script type="text/javascript">
     function changeSize() {
        var obj=document.getElementById("text");
        obj.style.fontSize="30px";
     }
  </script>
<p id="text" onClick="changeSize()"> 事件与处理程序的绑定 </p>
```

（2）可以直接在 JavaScript 代码中设置元素对象的事件属性，让事件属性值等于处理该事件的函数名或程序代码

示例：

```
   <p id="text"> 事件与处理程序的绑定 </p>
  <script type="text/javascript">
     function changeSize() {
        var obj = document.getElementById("text");
        obj.style.fontSize="30px";
     }
     document.getElementById("text").onclick=changeSize;
  </script></p>
```

或者为：也可以使用匿名函数来简化，即事件名=function(){...}

```
   <p id="text">破晓春月</p>
    <script>
     document.getElementById("text").onclick=function() {
        var obj = document.getElementById("text");
        obj.style.fontSize="30px";
     };
   </script>
```

2.4 JavaScript 的常用事件

1、鼠标事件

（1）onclick事件

onclick 事件：鼠标单击页面元素时触发的事件

示例：

```
   <script type="text/javascript">
     function showGender(obj) {
        alert(" 你选择的性别是："+obj.value);
     }
  </script>
  <h2> 性别：
<input type="radio" value=" 男 " name="gender" onClick="showGender(this)"/> 男
<input type="radio" value=" 女 " name="gender" onClick="showGender(this)"/> 女
  </h2>
```

（2）onmouseover 事件和 onmouseout 事件

onmouseover 事件和 onmouseout 事件：鼠标移入、移出页面元素时触发的事件

示例：

```
<body>
   <marquee direction="right" onmouseover="stop()" onmouseout="start()">
      <img src="../img/img_1.jpg" />
   </marquee>
</body>
```

（3）onmousemove 事件

onmousemove 事件：鼠标指针移动时发生的事件

示例：

```
<body>
      鼠标移动了 <span id="sp">0</span>px
   <div style="width:200px; height:200px; background-color:#ccc"
onmousemove="move()"></div>
   <script type="text/javascript">
      // 计数器
      var count = 0;
      function move() {
         document.getElementById("sp").innerHTML = ++count;
      }
   </script>
</body>
```

2、其他事件

（1）onload 事件：页面加载完成后立即发生

示例：

```
<body>
   <script type="text/javascript">
         window.onload=function() {
        alert(" 页面加载完成 ");
      }
   </script>
</body>
```

（2）onblur 事件：光标或者焦点离开元素后触发的事件

示例：

```
<body>
   <p> 请输入密码：<input  type="password"  id="txtPwd"  onblur="checkPwd(this)"/>
</p>
   <script  type="text/javascript">
      function  checkPwd(obj)  {
         var  pwd=obj.value;
         if(pwd.length>=6){
            alert(" 密码输入正确 ");
         }  else  {
            alert(" 密码的长度必须是 6 位或以上 ");
         }
      }
   </script>
</body>
```

(3)onchange事件：输入框的值发生了变化或者改变下拉列表框的选项时会触发的事件

示例：

```
   <script  type="text/javascript">
      function  changeLink(obj)  {
         var  site  =  obj.value;
         if(site  !=" 请选择 ")  {
            window.open(site);
         }
      }
   </script>
      友情链接：<select  onChange="changeLink(this)">
      <option  value=" 请选择 "> 请选择 </option>
      <option  value="http://www.baidu.com"> 百度 </option>
          //其他option标签代码略
```

3、表单事件

单击表单元素的“提交按钮”会触发form标签的 onsubmit 事件，浏览器对这个事件的默认处理方式是提交数据给 action 属性指定的页面进行处理
如果要阻止提交数据到指定的页面，就需要编写一个事件处理程序来改变浏览器对form标签的 onsubmit 事件的默认处理方式

示例：

```
<body>
   <script  type="text/javascript">
      function  check()  {
         event.returnValue=false;
      }
   </script>
   <form  action="info.html"  onSubmit="check()">
      <input  type="submit"  value=" 提交 "/>
   </form>
</body>
```

验证用户是否在表单中输入姓名：

```
<script type="text/javascript">
    function check() {
        // 获取输入在 id="name" 文本框中的内容
        var userName=document.getElementById("name").value;
        if(userName.trim().length>0) {
            return true;
        } else {
            alert(" 请输入用户名 ");
            return false;
        }
    }
</script>
<form action="info.html" onSubmit="return check()">
    <p> 用户名：<input type="text" id="name"/></p>
    <p><input type="submit" value=" 提交 "/></p>
</form>
```

按回车键触发登陆事件：

点击键盘的 Enter 键或者单击“登录”按钮，验证用户输入的邮箱和密码是否正确。

```
<body onkeypress="submitForm()">
    <form action="index.html">
        密码：<input type="password" name="pass" id="pass">
        <br>
        <input type="submit" onclick="login()" value="提交">
    </form>

    <script>
            function submitForm()
            {
                //判断是不是按了回车键
                var key = event.keyCode;
                //按的回车键为13
                if(key == 13)
                {
                    login();
                }
            }
            function login()
            {
                alert("登陆成功！");
            }
    </script>
</body>
```

第三章 JavaScript 浏览器对象模型

3.1 浏览器对象模型

BOM 是浏览器对象模型的简称。JavaScript 将整个浏览器窗口按照实现的功能不同拆分成若干个对象一个完整的 BOM 主要包括 window 对象、history 对象、location 对象和 document 对象等

BOM中，整个对象的层次关系：

window对象是最大的，包括了history对象、location对象以及document对象。document对象包括了body中的所有标签，例如a、div、form、p.....。

3.2 window 对象

window 对象处于对象模型的第一层，对于每个打开的窗口系统都会自动将其定义为 window 对象。
window 对象常用属性：

属性	含义
document	窗口中当前显示的文档对象
history	history 对象保存窗口最近加载的 URL
location	当前窗口的 URL
status	状态栏文本

window 对象常用方法：

方法	说明
prompt	显示可提示用户输入的对话框
alert	显示带有一个提示消息和一个确定按钮的警示框
confirm	显示一个带有提示信息、确定和取消按钮的确认框
close	关闭浏览器窗口
open	打开一个新的浏览器窗口，加载给定URL所指定的文档
setTimeout	在设定的毫秒数后调用函数或计算表达式
setInterval	按照设定的周期（以毫秒计）来重复调用函数或表达式
clearInterval	取消重复设置，与setInterval对应

1、alert() 方法弹出警告对话框

示例：

```
    <script>
        var name = "张三";
        var age = 10;
        alert("名字是：“+name+"\n年龄是：“+age);
    </script>
```

2、prompt() 方法创建提示对话框

示例：

```
    <script>
        var  name = window.prompt("你的名字是？");
        if(name)
        {
            alert("名字是：“+name);
        }
        else
        {
            alert("没有输入！");
        }
    </script>
```

3、confirm() 方法创建确认对话框

示例：

```
    var flag = window.confirm("您是否需要删除该选项？"); //点击确定时返回true，点击取消返回的是false
    if(flag)
    {
        //点击确定执行这里
        alert("您点击了确定！");
    }
    else
    {
        //点击取消执行这里
        alert("您点击了取消！");
    }
```

4、打开一个新的窗口 window.open()

window对象的 open() 方法和 close() 方法用于打开和关闭窗口
open方法的第一个参数是新窗口的URL，第二个参数是给新窗口的命名，第三个参数是设置新窗口的特征

窗口的常见特征	
名称	**说明**
height、width	窗口文档显示区的高度、宽度，单位为像素
left、top	窗口与屏幕左边、顶端的距离，单位为像素

示例：

```
window.open("window.html","网页标题名称","width=300,height=400");
```

5、setTimeout()方法

setTimeout() 方法会在指定的时间执行指定的代码并退出。

示例：

```
setTimeout("show()",2000);
function show()
{
    // window.open("onchange.html","onchange","width=300,height=400");
    alert("间隔2S执行我！");
}
```

6、setInterval() 定时器

setInterval() 方法会根据设置的时间间隔反复执行指定的代码，直至程序结束或利用clearInterval() 方法取消。setInterval() 定时器 设置多少秒重复执行这部分代码。

示例：

```
var num = 0;
function count()
{
    document.write(num+"<br>");
    num++;
}
setInterval("count()",1000);
```

7、requestAnimationFrame() 方法

requestAnimationFrame() 方法是浏览器用于定时循环操作的一个接口，类似于 setTimeout，主要用途是按帧对网页进行重绘
优势在于充分利用显示器的刷新机制，比较节省系统资源

语法：

```
requestID  =  window.requestAnimationFrame(callback);
```

使用示例：

```
<style>
    .box{
        width:100px;
        height:100px;
        background:blue;
        /* 相对于原来的位置动 */
        position:relative;
    }
</style>
<div class="box" id="box">
    一个滑块
</div>
<input type="button" value="开始" onclick="start()">
<input type="button" value="停止" onclick="stop()">
<input type="button" value="反着走" onclick="reverse()">
<script>
    //初始化位置默认为0
    var position = 0;
    function move()
    {
        var box = document.getElementById("box");
        box.style.left = position+"px";
        position+=1;
    }
    function move2(){
        var box = document.getElementById("box");
        box.style.left = position+"px";
        //反着走，就是往左边 【减】
        position-=1;
    }
    var setIn;
    function reverse()
    {
        //同时开启多个定时器，点一次就执行一次，会多个定时器同时生效！！
        setIn = setInterval("move2()",100);
    }
    var setIn2;
    var animation;
    function start()
    {
        // setIn2 = setInterval("move()",100);
        // 相当于刷新页面（不是整个页面的刷新）
        // 重复执行自己本身（这有一个递归效果）
        animation = requestAnimationFrame(start);
        // 执行移动方法
        move();
    }
    function stop()
    {
        //清除定时器 =》一个页面是可以存在多个定时器，这时候呢需要告诉它清除的是【哪一个】
        clearInterval(setIn);
        // clearInterval(setIn2);
        cancelAnimationFrame(animation);
    }

</script>
```

3.3 history 对象和 location 对象

1、history对象

history对象保存了当前浏览器窗口中打开页面的一个历史记录列表，使用 history对象可以将当前浏览器页面跳转到某个曾经打开过的页面

History**对象的方法**	
方法	**描述**
back()	后退一个页面，相当于浏览器后退按钮
forward()	前进一个页面，相对于浏览器前进按钮
go()	打开一个指定位置的页面

需要注意的是：**可以使用 history.go(-1) 和 history.go(1) 代替 histroy.back() 和 history.forward()**

示例：

```
    <!-- history对象 指的是历史记录列表，可以返回上一个页面和下一个页面。或者指定的某个页面！！ -->
    <a href="onchange.html">onchange页面</a>
    <a href="javascript:window.history.back();">返回上一个页面</a>
    <a href="javascript:history.forward();">前进到下一个页面</a>
    <!-- go里面的数字指的是列表中的【第几个】历史记录的页面 -->
    <a href="javascript:history.go(-1);">返回上一个页面</a>
    <a href="javascript:history.go(1);">前进到下一个页面</a>
```

2、location对象

location对象用于管理当前打开窗口的URL信息，相当于浏览器的地址栏

location**对象的常用属性和方法**	
名称	**描述**
href 属性	返回或设置当前页面的 URL
hostname 属性	返回 Web 主机的域名
pathname 属性	返回当前页面的路径和文件名
port 属性	返回 Web 主机的端口（80 或 443）
protocol 属性	返回所使用的 Web 协议（http:// 或 https://）
reload() 方法	重新加载当前页面，相对于浏览器的刷新按钮
assign() 方法	加载新的文档

示例：

```
    <script>
        document.write("当前的访问路径："+location.href+"<br>");
        document.write("当前的访问主机域名："+location.hostname+"<br>");
//www.baidu.com
        document.write("当前的访问协议："+location.protocol+"<br>");// http://
        document.write("当前的访问端口号："+location.port+"<br>");
        // 刷新当前页面(用得最多！！)
        location.reload();
    </script>
```

3.4 screen 对象和 navigator 对象

window.screen 对象包含了用户屏幕的相关信息，在编写时可以不使用 window前缀

screen**对象的常用属性**	
属性	**描述**
availWidth	返回显示屏幕的可用宽度（除 Windows 任务栏之外）
availHeight	返回显示屏幕的可用高度（除 Windows 任务栏之外）
colorDepth	返回目标设备或缓冲器上的调色板的比特深度
pixelDepth	返回显示屏幕的颜色分辨率（比特每像素）
width	返回显示器屏幕的宽度
height	返回显示器屏幕的高度

示例：使用screen对象中的属性获取访问者的屏幕信息

```
    <h3> 你的屏幕：</h3>
  <script  type="text/javascript">
     document.write(" 总宽度 / 高度 :");
     document.write(screen.width+"*"+screen.height);
     document.write("<br/>");
     document.write(" 可用宽度 / 高度 :");
     document.write(screen.availWidth+"*"+screen.availHeight);
     document.write("<br/>");
     document.write(" 色彩深度 :");
     document.write(screen.colorDepth);
     document.write("<br/>");
     document.write(" 色彩分辨率 :");
     document.write(screen.pixelDepth);
  </script>
```

navigator对象包含了浏览器的有关信息
navigator 对象的实例是唯一的，可以用 window 对象的 navigator 属性来引用它

navigator对象的常见方法****	
方法	**描述**
javaEnabled()	规定浏览器是否启用 Java
taintEnabled()	规定浏览器是否启用数据污点，仅适用于 IE 浏览器（Data Tainting）

navigator对象的常用属性如下：

navigator对象的常用属性****	
属性	**描述**
appCodeName	返回浏览器的代号
appMinorVersion	返回浏览器的次级版本
appName	返回浏览器的名称
appVersion	返回浏览器的平台和版本信息
browserLanguage	返回当前浏览器的语言
cookieEnabled	返回浏览器中是否启用 cookie 的布尔值

navigator对象的常用属性****	
属性	**描述**
cpuClass	返回浏览器系统的 CPU 等级
onLine	返回系统是否处于联机模式的布尔值
platform	返回运行浏览器的操作系统平台
systemLanguage	返回操作系统使用的默认语言
userAgent	返回由客户机发送服务器的 user-agent 头部的值
userLanguage	返回操作系统的自然语言设置

使用navigator对象中的属性和方法获取当前浏览器的相关信息:

```
<body>
   <div  id="info"></div>
   <script  type="text/javascript">
      var  info  =  "<p> 浏览器代号 :"  +  navigator.appCodeName     +  "</p>";
         info+="<p> 浏览器名称 :"+navigator.appName+"</p>";
         info+="<p> 浏览器版本 :"+navigator.appVersion+"</p>";
```

```
        info+="<p> 是否处于联机模式 :"+navigator.onLine+"</p>";
        info+="<p> 启用 Cookies:"+navigator.cookieEnabled+"</p>";
        info+="<p> 硬件平台 :"+navigator.platform+"</p>";
        info+="<p> 用户代理 :"+navigator.userAgent+"</p>";
        info+="<p> 是否启用 Java:"+navigator.javaEnabled()+"</p>";
      document.getElementById("info").innerHTML=info;
   </script>
</body>
```

由于 navigator 会误导浏览器检测，所以可以使用对象检测来嗅探不同的浏览器。但不同的浏览器支持不同的对象，因此对于不同的浏览器，要使用不同的对象来检测

navigator 对象集合**	
集合	**描述**
	返回对文档中所有嵌入式对象的引用
plugins[]	该集合是一个 plugin 对象的数组，其中的元素代表浏览器已经安装的插件。plugin对象提供的是有关插件的信息，其中包括它所支持的 MIME 类型的列表
	虽然 plugins[ ] 数组是由 IE4 定义的，但是在 IE4 中它却总是空的，因为 IE4 不支持插件和 plugin 对象

使用navigator对象显示出浏览器安装了哪些插件:

```
<body>
   <h2> 你的浏览器安装了以下插件：</h2>
   <script type="text/javascript">
      var plug = navigator.plugins;
      for (var i=0; i<plug.length;i++){
         document.write(plug[i].name+"<br/>")
      }
   </script>
</body>
```

第四章 JavaScript 文档对象模型

4.1 文档对象模型简介及属性

DOM（Document Object Model）是文档对象模型的简称
DOM 把HTML 文档看成由元素、属性和文本组成的一棵倒立的树

可以把HTML 文档中的每个成分看成一个节点，所以DOM 的核心操作是查看节点、创建节点、增加节点、删除节点以及替换节点。节点的特点如下:

1. 整个文档是一个文档节点
2. 每个HTML 标签是一个元素节点

3. 包含在HTML 元素中的文本是文本节点
4. 每个HTML 属性是一个属性节点
5. 注释属于注释节点
6. HTML 文档中的节点彼此间都存在关系，类似一张家族图谱

分析：

1. 除文档根节点之外的每个节点都有父节点。
2. 大部分元素节点都有子节点。
3. 当节点共享同一个父节点时，它们就是同辈。
4. 节点也可以拥有后代，后代指某个节点的所有子节点，或者这些子节点的子节点，以此类推。
5. 节点也可以拥有先辈。先辈是某个节点的父节点，或者父节点的父节点，以此类推。

整个HTML 文档在DOM 中是一个document 对象，常用属性如下表：

bgColor : 页面的背景颜色

fgColor：文本的前景色

title：页面标题

使用示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>第四章</title>
</head>
<body>
    哈哈哈哈
    <script>
        document.title = "aaa";
        document.bgColor = "red";
        document.fgColor = "white";
    </script>
</body>
</html>
```

4.2 document 对象查找 HTML 元素

1、使用 document.getElementById(id) 方法可以通过 id 获取 HTML 页面的元素。

示例：

```
    <a href="" id="link">哈哈哈标签</a>
    <script>
        var link = document.getElementById("link");
        link.innerHTML = "<b>我很好! </b>";
        document.write(link.innerHTML);
    </script>
```

注意：innerHTML属性是一个字符串，用来设置或获取位于对象起始和结束标签之间的HTML内容

2、使用 document.getElementsByName(name) 方法可以通过name获取页面元素。
参数 name 为必选项，为字符串类型。
返回值为数组对象，如果无符合条件的对象，则返回空数组。

示例：

```
<table border="1">
        <tr>
            <td>
                <input type="checkbox" name="all" onclick="check(this)">
                全选
            </td>
            <td>项目</td>
            <td>状态</td>
            <td>类型/数量</td>
            <td>单价</td>
            <td>小计</td>
        </tr>
        <tr>
            <td><input type="checkbox" name="choices" id=""></td>
            <td>iphone X</td>
            <td>可购买</td>
            <td>1</td>
            <td>8888</td>
            <td>8888</td>
        </tr>
        <tr>
            <td><input type="checkbox" name="choices" id=""></td>
            <td>iphone 12</td>
            <td>可购买</td>
            <td>1</td>
            <td>8888</td>
            <td>8888</td>
        </tr>
    </table>
    <script>
        function check(obj)
        {
            /*获取name值相同的表单*/
            var boxs = document.getElementsByName("choices");
            var bool = obj.checked;
            for(var i=0;i<boxs.length;i++)
            {
                boxs[i].checked = bool;
            }
        }
```

```
    </script>
```

3、使用 document.getElementsByTagName(tagname) 方法可以通过标签名访问页面元素。
参数 tagname 为必选项，为字符串类型。
返回值是指定标签名的对象的集合，如果无符合条件的对象，则返回空数组。

示例：

```
    <div class="nav" id="nav">
        <ul>
            <li><a href="">首页</a></li>
            <li><a href="">产品列表</a></li>
            <li><a href="">新闻</a></li>
        </ul>
    </div>

    <script>
        var nav = document.getElementById("nav");
        var lis = nav.getElementsByTagName("li");
        for(var i=0;i<lis.length;i++)
        {
            lis[i].onmouseover = function(){
                this.style.background = "red";
            }
            lis[i].onmouseout = function(){
                this.style.background = "#ccc";
            }
        }
    </script>
```

4、使用 document.getElementsByClassName(classname) 方法可以通过类名访问页面元素。
参数 classname为必选项，是字符串类型，指需要获取的元素类名。
返回值为NodeList 对象，表示指定类名的元素集合。可通过节点列表中的节点索引号来访问列表中的节点。

示例：

```
    <ul>
        <li class="cl"><a href="">我是第一行，你说什么？</a></li>
        <li><a href="">我是第二行，你说什么？</a></li>
        <li class="cl"><a href="">我是第三行，你说什么？</a></li>
        <li><a href="">我是第四行，你说什么？</a></li>
        <li class="cl"><a href="">我是第五行，你说什么？</a></li>
        <li><a href="">我是第六行，你说什么？</a></li>
    </ul>
    <script>
        //拥有同个类名的标签背景都变红色
        var cl = document.getElementsByClassName("cl");
        for(var i=0;i<cl.length;i++)
        {
            cl[i].style.background = "red";
        }
    </script>
```

可以使用NodeList对象的length属性来确定指定类名的元素个数，并循环各个元素来获取需要的元素

4.3 document 对象改变 HTML

1、文档中输出内容 document.write

JavaScript 中的 document 对象能够动态地创建 HTML 内容。document.write() 方法可用于直接向 HTML 的输出流写内容

示例：

```
    <script>
        document.write("当前时间："+new Date());
    </script>
```

不要在文档加载完成之后使用document.write()方法，这样做会覆盖该文档中所有的内容。

2、在页面中插入HTML标签代码

修改 HTML 内容最简单的方法是使用 innerHTML 属性

语法：

```
document.getElementById(id).innerHTML= 新的 HTML 内容
```

示例：

```
    <script type="text/javascript">
     function insert() {
           var obj = document.getElementById("content");
        obj.innerHTML="<h2> 我是动态添加的内容 </h2>"
     }
   </script>
   <div id="content">
   </div>
   <input type="button" value=" 向页面中添加内容 " onClick="insert()"/>
```

3、修改标签样式 对象.style.CSS样式 = 新样式;

如果需要改变 HTML 元素的样式，可使用以下语法：

```
document.getElementById(id).style.property= 新样式
```

示例：

```
    <div id="content"></div>
    <input type="button" value="添加内容" onclick="addHtml()">
    <input type="button" value="改变样式" onclick="changeCss()">
    <script>
        var c = document.getElementById("content");
        function addHtml()
```

```
        {
            c.innerHTML = "<h2>我新增的内容哦</h2>";
        }
        //改变样式
        function changeCss()
        {
            c.style.background = "red";
            c.style.color = "white";
            c.style.fontSize = "18px";
        }
    </script>
```

小结：

1. 每个 HTML 对象都有用于访问 CSS 样式的 style 属性，style 对象中包含一系列与 CSS 属性相对应的属性
2. style 对象的属性同CSS的属性命名不同，它删除了“-”，第一个单词后面的每个单词首字母大写

如果 HTML 文档中包含超链接，要实现当鼠标移入链接时，超链接文本大小变为 30px，下述代码是否正确，如果不正确该如何修改？

```
<a href="#" onmouseover="this.style.fontSize='18px'">登陆</a>
```

5、如果需要改变 HTML 元素的属性，可使用以下语法：

语法：

```
document.getElementById(id).attribute= 新属性值
```

示例：

```
    <p>
    <img  id="image"  src="../img/smiley.gif"  width="160"      height="120"/>
    </p>
  <p>
    <input  type="button"  value=" 更换图片 "      onClick="changePic()"/>
    </p>
  <script  type="application/javascript">
      function  changePic()  {
        var  img  =  document.getElementById("image");
        img.src="../img/landscape.jpg";
      }
  </script>
```

4.4 DOM节点操作

1、DOM节点的使用

可以根据层级关系来查找节点，在 DOM 中每个节点都具有访问其他节点的属性

节点属性	
属性	**描述**
parentNode	当前节点的父节点引用
childNodes	当前节点的所有子节点
firstChild	当前节点的第一个子节点
lastChild	当前节点的最后一个子节点
previousSibling	当前节点的前一个兄弟节点
nextSibling	当前节点的后一个兄弟节点

使用示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>节点的使用</title>
</head>
<body>
    <style>
        .con{
            width:100px;
            height:100px;
            background: tomato;
        }
    </style>
    <div class="con">这是一个盒子哦</div><div id="box"><a href="">哈哈哈</a><a
href="">2222</a><a href="">3333</a></div>
    <input type="button" value="父节点添加样式" onclick="addParent()">
    <input type="button" value="子节点添加样式" onclick="addChild()">
    <input type="button" value="第一个节点" onclick="first()">
    <input type="button" value="最后一个节点" onclick="last()">
    <input type="button" value="当前节点的前一个兄弟节点" onclick="prev()">
    <input type="button" value="当前节点的后一个兄弟节点" onclick="back()">
    <script>
        var box = document.getElementById("box");
        function addParent()
        {
            box.parentNode.style.background = "tomato";
        }
        function addChild()
        {
            var length = box.childNodes.length;
            var lis = box.childNodes;
            for(var i=0;i<length;i++)
            {
                if(lis[i].nodeType == 3)
                {
                    console.log(lis[i]);
                }
            }
```

```
        }
        //获取第一个节点
        function first()
        {
            // 如果使用firstChild获取，它的下一个节点存在空字符，那么获取到的是空文本。 如果想获取第一个标签，可以通过firstElementChild获取。
            // var first = document.getElemenById("box").firstChild;
            var first = document.getElementById("box").firstElementChild;
            console.log(first);
            first.style.color = "red";
        }
        function last()
        {
            // 如果使用firstChild获取，它的下一个节点存在空字符，那么获取到的是空文本。 如果想获取第一个标签，可以通过lastElementChild获取。
            var last = document.getElementById("box").lastChild;
            // var last = document.getElementById("box").lastElementChild;
            console.log(last);
            last.style.color = "red";
        }
        function prev()
        {
             //注意节点间不能有空格！！ 建议用previousElementSibling
            var prev = document.getElementById("box").previousSibling;
            prev.style.background = "blue";
        }
        function back()
        {
            //注意节点间不能有空格！！ 建议用nextElementSibling
            var back = document.getElementsByClassName("con")[0].nextSibling;
            back.style.background = "red";
        }
    </script>
</body>
</html>
```

1. childNodes 属性，它返回当前节点的所有子节点，其中子节点包括元素节点、属性节点和文本节点。
2. 通过节点对象的 nodeType 属性可以判断属于哪种类型的节点。
3. 当 nodeType 是 1 时就是元素节点；nodeType 为 2 时是属性节点；nodeType 为 3 时则是文本节点。

以上获取节点下的子标签时，建议使用firstElementChild、lastElementChild、previousElementSibling、nextElementSibling

2、添加删除节点

1、添加节点：
（1）使用 createElement 创建节点
（2）使用 appendChild(node) 方法将指定的节点追加到现有节点的末尾
2、删除节点：
（1）使用 removeChild 删除节点

使用 DOM 删除元素时，需要清楚地知道要删除元素的父元素。

使用示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>创建删除元素</title>
    <style>
        .box{
            width:200px;
            height: 100px;
            background: tomato;
        }
        .box h1{
            color:#fff;
        }
    </style>
</head>
<body id="body">

    <textarea name="say" id="say" cols="30" rows="4"></textarea>
    <input type="button" value="发表说说" onclick="sendComment()">

    <h3>说说列表：</h3>
    <ul id="content">
        <!-- 评论列表 -->
    </ul>

    <script>
        var say = document.getElementById("say");
        var content = document.getElementById("content");

        //发表评论
        var num = 1;
        function sendComment()
        {
            //获取文本框内容
            var con = say.value;
            //创建li
            var li = document.createElement("li");
            //拼凑发表内容格式
            var str = num+"楼："+con+"<a href='#' onclick='del(this)'>删除评论
</a>";
            li.innerHTML = str;
            //在 父节点后的节点前插入一个新标签：  父节点.insertBefore(新标签,父元素下的新
节点对应的位置);
            // content.insertBefore(li,content.childNodes[0]);
            content.appendChild(li);
            num++;
        }

        //删除评论
        function del(obj)
        {
            if(confirm("您是否要删除该评论？"))
            {
```

```
                //把当前这个标签的父节点删除
                content.removeChild(obj.parentNode);
            }
        }
    </script>
</body>
</html>
```

第五章 JavaScript 对象

5.1 Object 对象和 Date 对象

1、Object对象的使用

JavaScript 提供了一些非常有用的内部对象，按使用方式可分为两种：

1. 实例对象。在引用该对象的属性和方法时，必须先使用 new 关键字创建一个对象实例，然后再使用“对象实例名 . 成员”的格式来进行访问。
2. 静态对象。在引用该对象的属性和方法时不需要使用 new 关键字来创建对象实例，可以直接使用“类名 . 成员”的格式来进行访问

Object 对象提供了一种创建自定义对象的简单方式，因为它不需要开发人员定义构造函数。
在程序运行时可以为JavaScript 对象添加属性，因此使用 Object 对象创建出自定义对象非常简便

注意：对象的属性可以使用索引运算符“[ ]”进行访问

使用示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Object对象</title>
</head>
<body>

    <script>
        //下面这种声明格式为静态方法声明:
        var person = new Object();
        person.name = "张三";
        person.age = 20;
        //创建方法
        person.getArea = function(){
            alert("哈哈哈！");
        }
        person.getArea();
    </script>

</body>
</html>
```

2、Date对象

通过创建 Date 对象，可以获取本地计算机中的日期与时间。
初始化 Date 对象有 4 种方式，语法如下：

```
（1）new  Date()  // 当前日期和时间
（2）new  Date(milliseconds)  // 返回从 1970 年 1 月 1 日至今的毫秒数
（3）new  Date(dateString)
（4）new  Date(year, month, day, hours, minutes, seconds, milliseconds)
```

说明：每个Date对象都只是计算机的一个毫秒级快照，Date对象只是保存了它被创建时的时间信息

使用示例：

```
        //1、new  Date()  // 当前日期和时间
        var now = new Date();
        document.write(now+"<br>");
        //2、new  Date(dateString)
        var date2 = new Date("2020-12-12 10:50:30");
        document.write(date2+"<br>");
        //3、new  Date(year, month, day, hours, minutes, seconds, milliseconds)
        var date3 = new Date(2020,12,11,8,56,32,0);
        document.write(date3+"<br>");
        //4、new  Date(milliseconds)  // 返回从 1970 年 1 月 1 日至今的毫秒数
        var date4 = new  Date(1563256879000);
        document.write(date4+"<br>");
```

常用方法：

方法	功能
getDate()	返回一个月中的某一天（1 ~ 31）
getDay()	返回一周中的某一天（0 ~ 6），0 为周日，1 为周一，以此类推
getFullYear()	以四位数返回年份
getHours()	返回小时（0 ~ 23）
getMilliseconds()	返回毫秒
getMinutes()	返回分钟（0 ~ 59）
getMonth()	返回月份（0 ~ 11），0 为一月，1 为二月，以此类推
getSeconds()	返回秒数（0 ~ 59）
getTime()	返回 1970 年 1 月 1 日至今的毫秒数

使用示例一：当前时间显示

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

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Object对象</title>
</head>
<body>

    <div id="box"></div>

    <script>
        var person = new Object();
        person.name = "张三";
        person.age = 20;
        //创建方法
        person.getArea = function(){
            // alert("哈哈哈！");
        }
        // person.getArea();
        //1、new  Date()  // 当前日期和时间
        var now = new Date();
        document.write(now+"<br>");
        //2、new  Date(dateString)
        var date2 = new Date("2020-12-12 10:50:30");
        document.write(date2+"<br>");
        //3、new  Date(year, month, day, hours, minutes, seconds, milliseconds)
        var date3 = new Date(2020,12,11,8,56,32,0);
        document.write(date3+"<br>");
        //4、new  Date(milliseconds)  // 返回从 1970 年 1 月 1 日至今的毫秒数
        var date4 = new  Date(1563256879000);
        document.write(date4+"<br>");

        //document.write(now.getTime());
        function nowTime()
        {
            var box = document.getElementById("box");
            //获取当前日期的展示
            var date = new Date();
            //获取年:  四位数
            var year = date.getFullYear();
            //获取月：0~11
            var month = date.getMonth()+1;
            //获取日：一个月的某一天 1~31
            var day = date.getDate();
            //获取当前小时 0~23
            var hour = date.getHours();
            //获取当前分
            var minutes = check(date.getMinutes());
            //获取秒
            var seconds = check(date.getSeconds());

            var str = hour+":"+minutes+":"+seconds+"
<br>"+year+"年"+month+"月"+day+"日";
            box.innerHTML = str;
        }

        //拼格式返回
        function check(time)
        {
            if(time < 10)
            {
```

```
                time = "0"+time;
            }
            return time;
        }

        setInterval("nowTime()",1000);

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

示例二：倒计时功能

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>倒计时</title>
</head>
<body>
    <!-- 时间盒子 -->
    <div id="box"></div>

    <script>
        //计算时间差
        function countTime(year,month,day,hour,minutes,seconds)
        {
            //获取当前的时间
            var now = new Date();
            var nowYear = now.getFullYear();
            var nowMonth = now.getMonth()+1;
            var nowDay = now.getDate();
            var nowHours = now.getHours();
            var nowMinutes = now.getMinutes();
            var nowSecond = now.getSeconds();

            //传进来的时间
            var next = new Date(year,month,day,hour,minutes,seconds);
            if(next <= now)
            {
                alert("输入倒计时的时间需要大于当前时间！");
                return;
            }
            //计算秒数差值 => 毫秒转换成秒数
            var diff = (next.getTime() - now.getTime())/1000;

            //计算相差的差值
            var diffDay = parseInt(diff/(24*60*60));//换成具体的日期差值
            var diffHour = parseInt(diff/(60*60)%24);//计算小时的差值
            var diffMiutes = parseInt(diff/60%60);//计算分钟的差值
            var diffSeconds = parseInt(diff%60);//计算分钟的差值

            //输出结果
            document.getElementById("box").innerHTML = "当前时
间："+nowYear+"年"+nowMonth+"月"+nowDay+"日"+nowHours
```

```
            +":"+nowMinutes+":"+nowSecond+"
<br>"+"距"+year+"年"+month+"月"+day+"日"+hour
            +":"+minutes+":"+seconds+"<br>"+diffDay+"天"+diffHour+"小
时"+diffMiutes+"分"+diffSeconds+"秒";
        }
        setInterval("countTime(2021,10,1,0,0,0)",1000);
    </script>
</body>
</html>
```

5.2 Image 对象和 Math 对象

1、Image对象

网页中使用图片，只需要调用 标签，然后在 src 属性中设置图片的绝对路径或相对路径即可。
如果实现动画或者图像效果，则需要使用图像缓存技术，让用户对图像效果获得较好的体验，使用这种技术需要借助Image对象。

使用示例：替换图片显示效果

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>图像对象</title>
</head>
<body>

    <img src="./img/pc.jpg" alt="" id="img">
    <input type="button" value="切换图像" onclick="changeImg()">
    <script>
        //创建图片标签
        var imgObj = new Image();
        imgObj.src = "./img/mobile.jpg";
        function changeImg()
        {
            //使用1：
            document.getElementById("img").src = imgObj.src;
        }
        //使用2：
        document.body.appendChild(imgObj);
    </script>
</body>
</html>
```

2、Math对象

JavaScript 中的基本数值运算符可以用来进行一些简单的数学计算，而使用Math 对象可以进行更多的高级运算，如平方根、三角函数、对数和随机数等
与其他对象不同，Math 不需要使用new 关键字创建对象的实例

Math对象中常用的方法：

方法	功能
Math.abs(number)	返回number 的绝对值
Math.ceil(number)	对number 向上取整，如Math.ceil(67.6) 返回值是68
Math.floor(number)	对number 向下取整，如Math.floor (67.6) 返回值是67
Math.max(number1,number2)	返回number1 与number2 中的较大值
Math.min(number1,number2)	返回number1 与number2 中的较小值
Math.pow(x,y)	返回x 的y 次幂
Math.random()	返回0 和1 之间的伪随机数，可能为0，但总是小于1
Math.round(number)	返回最接近number 的整数
Math.sqrt(number)	number 的平方根

使用示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Math对象</title>
</head>
<body>

    <script>
        document.write(Math.abs(-1)+"<br>");//1
        document.write(Math.ceil(10.33)+"<br>");//10
        document.write(Math.floor(1.35)+"<br>");//1
        document.write(Math.max(10,30)+"<br>");//30
        document.write(Math.min(-1,100)+"<br>");//-1
        document.write(Math.pow(3,2)+"<br>");//9
        document.write(Math.random()+"<br>");//随机数
        document.write(Math.round(10.222)+"<br>");//10
        document.write(Math.sqrt(5)+"<br>");//25
    </script>

</body>
</html>
```

猜大小：

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

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Math对象</title>
</head>
<body>
    <input type="text" name="" id="number"><input type="button" value="检查"
onclick="check()">
    <script>
        //随机生成一个数
        var num = Math.random()*10;//0-1 之间
        num = Math.ceil(num);
        console.log(num);
        function check()
        {
            //容易忽略这个值
            var number = document.getElementById("number").value;
            if(num == number)
            {
                alert("你猜到了！");
            }else if(num < number)
            {
                alert("您猜大了！");
            }else{
                alert("您猜小了！");
            }
        }
    </script>

</body>
</html>
```

5.3 数组与 String 对象

1、数组列表
数组列表用于表示一组数据的集合，它由一对方括号（[]）包围，列表中的每个元素用逗号分隔，数组元素可以是任意类型的数据（包括其他数组），数组的定义如下：

```
var arr=["happy",12,45.6];
```

需要注意的：

1. 每个数组变量都有一个length属性，表示该数组中元素的个数
2. 定义一个数组变量后，就可以使用“数组变量名[索引号]”的格式来访问每个数组元素
3. 数组列表中的第一个元素的索引号为0，其后的每个元素的索引号依次递增，最后的元素索引为数组的长度-1
4. 如果数组元素本身是一个数组，那么这个元素称为子数组，可以使用“数组变量名[子数组索引号][子数组中的元素索引号]”的格式来访问子数组中的元素

数组的使用：

```
<body>
     <script type="text/javascript">
    var arr=[' 白色',' 紫色',' 橙色',' 红色'];
    for(var i=0; i<arr.length;i++) {
         document.write(arr[i]+"<br/>");
    }
     </script>
</body>
```

需要知道的是：数组元素的下标不仅可以是数字，还可以是文本

```
<body>
     <script type="text/javascript">
    var arr=[]; // 声明数组变量
    arr['暖色调']=[' 红色',' 橙色',' 黄色'];
    arr['冷色调']=[' 绿色',' 青色',' 蓝色'];
    document.write("<h2>");
    // 输出冷色调的第3 种颜色
    document.write(arr[' 冷色调'][2]);
    document.write("</h2>");
     </script>
</body>
```

Array的使用：

1. JavaScript 中提供了一个名为Array 的内部对象，可用它来创建数组。通过调用Array 对象的各种方法，可以方便地对数组进行排序、删除和合并等操作。
2. Array 对象创建数组常用的3种方式：

使用语法：

```
（1）var arr=new Array()       //数组初始元素个数为0
（2）var arr=new Array(4);  //创建具有指定大小的Array 对象
（3）var arr=new Array(1,2,3); //用指定的元素列表去初始化Array 对象，数组的长度是设置的元素的数目
```

2、string对象

当某字符串使用单引号或双引号标注时，可以被当作字符串对象实例进行处理，从而直接调用String 对象的属性和方法
常用属性：length
常用方法：

方法	描述
charAt()	返回字符串对象中指定索引处的字符，索引从0 开始，如"Hello World".charAt(3)，返回字符"l"
indexOf()	返回某个子字符串在目标字符串中首次出现的位置，如"Hello World".indexOf("a")，返回-1，在目标字符串中没有子字符串"a"，故返回-1
substr()	从指定索引位置开始截取指定长度的字符串， 如"Hello World".substr(2,3)，返回"llo"。第一个参数表示从索引为2 的字符开始截取，即"l"，第二个参数表示截取的长度

使用示例：

```
//字符串对象
var str = "hello world!!";
console.log("字符串长度为："+str.length+"<br>");
//charAt() 找到指定索引位置的字符返回  => 索引从0开始
var num = str.charAt(3);
console.log(num);
//indexOf 找到字符首次出现的位置  => 索引从0开始
var find = str.indexOf("l");
console.log(find);
// substr() 截取指定长度字符串(包含3的位置开始往后截取)
var sub = str.substr(3,5);
console.log(sub);
```

常用方法：

方法	功能
substring()	返回指定索引范围内的字符串， 如"Hello World".substring(2,3)，返回"l"，返回索引2和3 间的字符串，并且包括索引2 对应的字符，不包括索引3 对应的字符
toLowerCase()	将字符串转化为小写
toUpperCase()	将字符串转化为大写，如"Hello World".toUpperCase()，返回" HELLO WORLD"
split()	返回按照指定分隔符拆分的若干子字符串数组，如var arr="hello,world".split(",");arr 是数组变量，其中第一个元素是"hello"，第二个元素是"world"

使用示例：

```
//字符串对象
var str = "hello world!!";
//substring() 返回指定范围内的字符串(不包括对应索引3的字符)
var subs = str.substring(0,3);//hel
console.log(subs);
//toLowerCase() 转换成小写
var lower = str.toLowerCase();
```

```
        console.log(lower);
        //toUpperCase() 转换成大写
        var upper = str.toUpperCase();
        console.log(upper);
        // split() 字符分割
        var arrs = str.split(" ");
        console.log(arrs[0]);
        console.log(arrs[1]);
```

示例：验证用户输入的电子邮箱和密码是否合法：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>检查邮箱密码</title>
</head>
<body>

    <form action="http://www.shuaiqi100.com" onsubmit="checkForm()">
        邮件：<input type="text" name="email" id="email"><br/>
        密码：<input type="password" name="Pass" id="pass"><br/>
        <input type="submit" value="密码">
    </form>

    <script>
        function checkForm()
        {
            var email = document.getElementById("email").value;
            var pass = document.getElementById("pass").value;

            //判断邮件格式是否找到@字符
            if(email.indexOf("@") == -1)
            {
                alert("请输入正确的邮箱格式！");
                event.returnValue = false;
                document.getElementById("email").focus();
                return;
            }
            if(email.indexOf(".") == -1)
            {
                alert("请输入正确的邮箱格式！");
                event.returnValue = false;
                return;
            }

            //检查密码是否符合条件
            if(pass.length < 6)
            {
                alert("密码长度不能小于6！");
                event.returnValue = false;
                return;
            }
        }
```

```
    </script>

</body>
</html>
```

5.4 对象的创建及常用语句

1、function的方式声明一个类

类（class）是一个模板，模板描述了本类中所有对象共同的属性和行为
定义JavaScript 类，需要编写一个函数，函数名为对象的类名，在函数体中定义属性和方法，即 JavaScript 中的函数也是对象

定义方法：

> 用function 创建类时，要求属性和方法必须使用this 关键字来引用，表示当前类的实例。

示例：使用编写函数的方式创建类

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>创建一个类</title>
</head>
<body>

    <script>
        // 下面这种声明方式为实例化方法声明：
        function Book(name,author,price){
            this.name = name;
            this.author = author;
            this.price = price;

            this.getInfo = function(){
                console.log("书名为"+this.name+",作者为："+this.author+",价格
为："+this.price)
            }
        }
        //实例化时需要传值进去
        var book = new Book("《阿甘正传》","王菲",88);
        book.getInfo();
    </script>

</body>
</html>
```

2、with关键字的使用

在一段连续的程序代码中，在with 关键字后的小括号中写出这个对象的名称，**就可以在其后大括号里的执行语句中直接引用该对象的属性名或方法名，省略对象实例名和点（.）**

语法：

```
with( 对象名称){
      执行语句块
}
```

实现示例：获取当前的时间展示：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>创建一个类</title>
</head>
<body>

    <div id="box"></div>

    <script>

        // with关键字的使用：
        function getTime()
        {
            var box = document.getElementById("box");

            //获取这个对象中的对应方法
            var date = new Date();
            with(date)
            {
                var year = getFullYear();
                var month = getMonth();
                var day = getDate();
                var Hour = getHours();
                var minutes = getMinutes();
                var seconds = getSeconds();
            }
            //展示对应格式到盒子中
            box.innerHTML = year+"年"+month+"月"+day+"日
"+Hour+":"+minutes+":"+seconds;
        }
        setInterval("getTime()",1000);

    </script>

</body>
</html>
```

3、for...in 语句：
（1）对某个对象的所有属性进行循环操作，它将某个对象的所有属性名称逐一赋值给一个变量，不需要事先知道对象属性的个数
（2）根据索引取出每个数组元素

语法：

```
for( 变量 in 对象){
      执行语句
}
```

使用示例：对类的遍历

```
        function person()
        {
            this.name = "张三";
            this.age = 20;
            this.sex = "男";
        }
        var person = new person();
        for(var p in person)
        {
            //输出类中的全部属性
            console.log(p);
        }
```

使用示例2：打印数组

```
<script type="text/javascript">
      var arr=[" 欧冠"," 英超"," 意甲"," 西甲"]
      for(var i in arr){
    document.write(arr[i]);
      }
</script>
```

综合案例：创建一个人工类，实例化3个人对象，存储到数组中，再循环遍历打印把数据展示到表格：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>综合案例</title>
</head>
<body>

    <table id="table" border="1">
        <tr>
            <th>姓名</th>
            <th>年龄</th>
            <th>性别</th>
            <th>部门</th>
        </tr>
```

```
    </table>

    <script>
        var table = document.getElementById("table");
        //创建一个人类
        function person(name,age,sex,department){
            this.name = name;
            this.age = age;
            this.sex = sex;
            this.department = department;
        }

        //实例化三个类
        var p1 = new person("张三",25,"男","人事部");
        var p2 = new person("李四",24,"男","财务部");
        var p3 = new person("王五",23,"女","研发部");

        //存储到数组中
        var persons = [p1,p2,p3];
        //循环数组取再展示到表格中
        for(var p in persons)
        {
            var tr = document.createElement("tr");
            var per = new
person(persons[p].name,persons[p].age,persons[p].sex,persons[p].department);
            with(per)
            {
                var names = name;
                var ages = age;
                var sexs = sex;
                var departments = department;
            }
            tr.innerHTML = "<td>"+names+"</td><td>"+ages+"</td><td>"+sexs+"</td>
<td>"+departments+"</td>";
            table.appendChild(tr);
        }

    </script>

</body>
</html>
```

第六章 初识 jQuery

6.1 jQuery概述

1. jQuery 基本功能

   （1）为了解决开发过程中的兼容性问题，产生了许多JavaScript库，目前被频繁使用的JavaScript库包括 jQuery、Prototype、Spry 和 Ext　JS。

   （2）其中使用最广泛的JavaScrip库是jQuery, 是于2006年创建的一个JavaScript库
   集 JavaScript、CSS、DOM 和 Ajax 于一体的强大框架体系
   它的主旨是以更少的代码实现更多的功能（Write　less,do　more）

jQuery 基本功能：
（1）访问和操作 DOM 元素
（2）对页面事件的处理
（3）大量插件在页面中的运用
（4）与 Ajax 技术的完美结合
（5）大幅提高开发效率

2. 搭建 jQuery 开发环境

下载 jQuery 文件库（http://jquery.com）
引入 jQuery 文件库：

```
在页面<head></head> 标签对中加入如下代码：
<script  type="text/javascript"  src="../js/jquery-3.3.1.min.js"></script>
```

简单使用：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>图片展示隐藏</title>
    <script src="./js/jquery-3.3.1.min.js"></script>
    <script>
       $(document).ready(function(){
           $("#btn").click(function(){
              //    show里面的2000表示展示2秒展示完整出来
              $("#doc").show(2000);
           });
       })

        //等同于下面这种格式
        $(function(){
            $("#btn").click(function(){
              //    show里面的2000表示展示2秒展示完整出来
              $("#doc").show(2000);
           });
        });

    </script>
</head>
<body>

    <img src="./img/adv.jpg" id="doc" alt="" style="display: none;">
    <input type="button" value="显示图片" id="btn">
</body>
</html>
```

分析：

（1）$(document).ready(function(){ }); 类似JavaScript代码:window.onload=function(){ }
（2）$(document).ready 在页面框架下载完毕后就执行；而 window.onload 必须在页面全部加载完毕（包括下载图片）后才能执行
（3）$(document).ready(function(){ }) 可以简写成 $(function(){ })
（4）$("#btn").click(function(){ }); 使用 click() 方法，将函数绑定到指定元素的 click 事件中，单击按钮时，绑定的函数就会执行

window.onload和$(document).ready(function(){})区别：

一般的加载页面时调用js方法如下：

```
window.onload = function() {
$("table tr:nth-child(even)").addClass("even"); //这个是jquery代码
};
```

这段代码会在整个页面的document全部加载完成以后执行。不幸的这种方式不仅要求页面的DOM tree全部加载完成，而且要求所有的外部图片和资源全部加载完成。更不幸的是，如果外部资源，例如图片需要很长时间来加载，那么这个js效果就会让用户感觉失效了。

jquery:

```
$(document).ready(function() {

// 任何需要执行的js特效
$("table tr:nth-child(even)").addClass("even");
});
```

就仅仅只需要加载所有的DOM结构，在浏览器把所有的HTML放入DOM tree之前就执行js效果。包括在加载外部图片和资源之前。

3.jQuery 代码的特点

特点示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>jquery</title>
    <style>
        .color{
            color:red;
        }
        .weight{
            font-weight: normal;
        }
    </style>
    <script src="./js/jquery-3.3.1.min.js"></script>
    <script>
        $(function(){
```

```
            $("#title").click(function(){
                //toggleClass 对设置或移除被选元素的一个或多个类进行切换。
                //next() 获得匹配元素集合中每个元素紧邻的同胞元素。如果提供选择器，则取回匹配该选择器的下一个同胞元素。
                //toggle() 方法切换元素的可见状态。如果被选元素可见，则隐藏这些元素，如果被选元素隐藏，则显示这些元素。
                $(this).toggleClass("color weight").next("ul").toggle(1000);
            });
        });
    </script>
</head>
<body>

    <h3 id="title">文化娱乐</h3>
    <ul>
        <li>鲜花绿植</li>
        <li>111</li>
        <li>古董收藏</li>
        <li>音乐美好</li>
    </ul>

</body>
</html>
```

分析：

（1）$符号标志：是JQuery对象的简写形式，$()等效于jQuery()，是jQuery程序的标志
（2）隐式循环：当使用jQuery查找符合要求的元素后，无须一一遍历每一个元素，直接通过jQuery命令操作符合要求的元素
（3）链式书写：可以将针对于同一个对象多个操作命令写在一个语句中，还可以插入换行和空格

jquery实现改变背景色：

```
    <select name="" id="bg">
        <option value="white">白色</option>
        <option value="yellow">黄色</option>
        <option value="orange">橘色</option>
    </select>
    <script>
        $(function(){
            $("#bg").change(function(){
                var val = this.value;
                $("body").css("background",val);
            });
        });
    </script>
```

6.2 jQuery基本选择器

1. jQuery 选择器概述

   使用jQuery选择器可以选择页面元素，并生成jQuery对象，能够使用jQuery中的方法。jQuery 对页面元素的选择完全继承了 CSS 选择器的语法规则

   语法：

```
//selector 选择器  $(selector)为jQuery对象  .action()是对元素执行的操作
$(selector).action()
```

2. 基本选择器

基本选择器是由标签 id、class、标签名和多个选择符组成：

选择器	功能	返回值
#id	根据 id 属性值选取元素	单个元素
.class	根据 class 属性值选取元素	元素集合
element	根据给定的标签名选取元素	元素集合
*	选取所有元素，包括 html 和 body	元素集合
selector1,selector2,...,selectorN	将每一个选择器匹配到的元素合并后一起返回	元素集合

使用示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>选择器的使用</title>
    <script src="./js/jquery-3.3.1.min.js"></script>
    <script>
        $(function(){
            //以下方式叫做：基本选择器
            //选择所有标签
            $("*").css({"margin":"0","padding":"0"});
            //标签选择器 多个样式要用花括号括起来用

 $("div,span").css({"color":"red","width":"200px","height":"300px","border":"1px solid #ccc"});
            //id选择器 单个样式直接加：
            $("#yellow").css("display","block");
            // 类名选择器
            $(".bg").css("background","yellow");
            // 多个标签共用同一个样式，需要使用【逗号间隔】
            $("li , .bg2").css("font-size","20px");
        });
    </script>
</head>
```

```
<body>

    <div class="bg">盒子1</div>
    <div>盒子2</div>
    <ul>
        <li class="bg2">哈哈</li>
        <li>22</li>
        <li>33</li>
    </ul>
    <span id="yellow">111</span>
    <span>222</span>
    <span>333</span>
</body>
</html>
```

6.3 过滤选择器

1. 基本过滤选择器

   过滤选择器
   过滤选择器附加在所有选择器的后面，**通过特定的过滤规则来筛选出一部分元素，书写时以冒号（：）开头**
   过滤选择器可以分为
   （1）基本过滤选择器
   （2）内容过滤选择器
   （3）可见性过滤选择器
   （4）属性过滤选择器

   基本过滤器有：

选择器	功能	返回值
first() 或 :first	获取第一个元素	单个元素
last() 或 :last	获取最后一个元素	单个元素
:not(selector)	获取除给定选择器之外的所有元素	元 素 集 合， 如$("li:not(.title)") 获 取 class 不是 title 的 2. 元素
:even	获取索引值为偶数的元素，索引号从 0 开始	元素集合
:odd	将每一个选择器匹配到的元素合并后一起返回	元素集合
:eq(index)	获取索引值等于 index 的元素，索引号从 0 开始	单个元素，如 $(“li:eq(1)”) 获取索引等于 1 的 3. 元素
:gt(index)	获取索引值大于 index 的元素，索引号从 0 开始	元素集合，如 $("li:gt(1)") 获取索引大于但不包括 1 的 4. 元素
:lt(index)	获取索引值小于 index 的元素，索引号从 0 开始	元素集合，如 $("li:lt(1)") 获取索引小于但不包括 1 的 5. 元素
:header	获取所有标题元素，如 h1~h6	元素集合
:animated	获取正在执行动画效果的元素	元素集合

使用示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>基本过滤器</title>
    <script src="./js/jquery-3.3.1.min.js"></script>
</head>
<body>

    <ul>
        <li>哈哈</li>
        <li>22</li>
        <li class="title">33</li>
        <li>44</li>
        <li class="title">55</li>
        <li>66</li>
        <li>77</li>
        <li>88</li>
        <li>99</li>
    </ul>

    <h1>我是h1</h1>
```

```
    <h2>h2啊</h2>
    <script>
        // 获取第一个元素
        // $("ul li:first").css("color","red");
        //或者：
        $("ul li").first().css("color","blue");
        //获取最后一个元素
        $("ul li:last").css("font-weight","bold");
        //或者：
        $("ul li").last().css("color","red");

        // 除某个选择器外的其他选择器
        $("ul li:not(.title)").css("font-size","20px");

        //获取索引为偶数的元素，索引从0开始
        $("ul li:even").css("background","#ccc");
        //获取索引为奇数的元素，索引从0开始
        $("ul li:odd").css("background","yellow");

        //获取索引值等于 index 的元素，索引号从 0 开始
        $("ul li:nth(1)").css("text-align","center");

        // 获取索引值大于 index 的元素，索引号从 0 开始，不包括指定索引值
        $("ul li:gt(3)").css("color","orange");

        // 获取索引值小于 index 的元素，索引号从 0 开始，不包括指定索引值
        $("ul li:lt(3)").css("font-size","12px");

        //获取所有标题元素(如果没有指定范围内的标题，则等同于*)
        $(":header").css("font-weight","normal");
    </script>

</body>
</html>
```

6. 内容过滤选择器

内容过滤选择器根据元素中的文字内容或所包含的子元素特征来获取元素

选择器	功能	返回值
:contains(text)	获取含有文本内容为 text 的元素	元素集合
:empty	获取不包含后代元素或者文本的空元素	元素集合
:has(selector)	获取含有后代元素为 selector 的元素	元素集合
:parent	获取含有后代元素或者文本的非空元素	元素集合

注意：

> （1）在 `:contains(text)` 内容过滤选择器中，如果参数 `text` 内容出现在匹配元素的任何后代元素中，也认为该元素含有文本内容 `text`。如果参数 `text` 使用英文字母，则有大小写的区别
> （2）`<input>`、`<img>`、`<br>` 和 `<hr>` 等标签属于不包含后代元素和文本的空元素

使用示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>选择器的使用</title>
    <script src="./js/jquery-3.3.1.min.js"></script>
</head>
<body>
    <!-- 测试:contains() -->
    <div><p>盒子</p>1</div>
    <div>盒子2</div>
    <div>盒子3</div>
    <div>4</div>
    <div>5</div>
    <!-- 测试:empty() -->
    <div></div>
    <div></div>

    <!-- 测试包含:has() -->
    <div>
        <h1>哈哈</h1>
    </div>
    <div>
        <h1>哈哈2</h1>
    </div>

    <!-- 测试:parent() -->
    <div style="width:100px;height:100px;">
        <h3>11</h3>
    </div>
    <div style="width:100px;height:100px;">
        <h3>22</h3>
    </div>
    <div style="width:100px;height:100px;">
        <h3>222
    </div>
    <script>
        $(function(){
            //判断存在某个关键字的盒子被匹配到
            $("div:contains(盒
子)").css({"width":"100px","height":"100px","background":"red","float":"left"
,"margin-right":"10px"});
            // 判断获取为空内容的盒子

 $("div:empty").css({"width":"100px","height":"100px","background":"blue"});
            //获取盒子下含有某个标签元素的盒子,修饰
            $("div:has(h1)").css({"font-size":"20px","color":"red"});
            // 选取所有带有子元素或包含文本的 h3 的元素：
            $("h3:parent").css({"background":"red","color":"blue"})
        });
    </script>

</body>
</html>
```

7. 可见性过滤选择器

可见性过滤选择器根据元素是否可见的特征来获取元素

选择器	功能	返回值
:hidden	选取不可见元素	元素集合，如 $(":hidden") 选取所有隐藏的元素
:visible	选取可见元素	元素集合，如 $(":visible") 选取所有可见的元素

不可见元素包括：css 样式中 display 属性值为 none 的元素、type 属性值为 hidden 的 元素及宽高设置为 0 的元素。

使用示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>jquery</title>
    <style>
        img{
            display: none;
        }
    </style>
    <script src="./js/jquery-3.3.1.min.js"></script>
    <script>
        $(function(){
            $("#btnShow").click(function(){
                $("#img:hidden").show(2000);
            })
            $("#btnHide").click(function(){
                $("#img:visible").hide(1000);
            });
        });

    </script>
</head>
<body>
    <img src="./img/adv.jpg" alt="" id="img">
    <input type="button" value="显示图片" id="btnShow">
    <input type="button" value="隐藏图片" id="btnHide">
</body>
</html>
```

8. 属性过滤选择器

属性过滤选择器根据元素的某个属性获取元素，以“[”号开始、以“]”号结束

选择器	说明	返回值
[attribute]	获取拥有该属性的所有元素，如 $('li[title]') 表示获取所有包含title属性的 • 元素	元素集合
[attribute=value]	获取某属性值为 value 的所有元素，如 $('li[title=test2]') 表示获取所有包含 title 属性且属性值等于 test2 的 • 元素	元素集合
[attribute!=value]	获取某属性值不等于 value 的所有元素，如 $('li[title!=test2]') 表示获取所有包含 title 属性且属性值不等于 test2 的 • 元素	元素集合
[attribute^=value]	选取属性值以 value 开头的所有元素，如 $('a[href^="mailto:"]') 表示获取所有包含 href 属性，且属性值以 mailto: 开头的 元素	元素集合
[attribute$=value]	选取属性值以 value 结束的所有元素，如 $('a[href$=".zip"]') 表示获取所有包含 href 属性，且属性值以 .zip 结尾的 元素	元素集合
[attribute*=value]	选取属性值中包含value的所有元素，如 $('a[href*="jquery.com"]') 表示获取所有包含 href 属性且属性值中包含 jquery.com的 元素	元素集合
[selector1] [selector2]... [selectorN]	合并多个选择器，满足多个条件，每选择一次将缩小一次范围，如 $('li[id] [title^=test]') 选取所有拥有属性 id 且属性 title 以 test开头的 • 元素	元素集合

使用示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>属性选择器</title>
    <script src="./js/jquery-3.3.1.min.js"></script>
    <script>
        $(function(){
            $("#all").click(function(){
                //属性方式获取：[attribute]
                // $("input[id]").attr("checked","true");
                //属性值方式获取：[attribute=value] 属性设置或返回属性值的方法
                // $("input[type='checkbox']").attr("checked","true");
```

```
                //获取某属性值不等于 value 的所有元素： [attribute!=value]
                // $("input[value!='足球']").attr("checked","true");
                //选取属性值以 value 开头的所有元素: [attribute^=value]
                // $("a[href^='http://']").css("background","red");
                // 选取属性值以 value 结束的所有元素:[attribute$=value]
                // $("a[href$='.com']").css("background","red");
                // 选取属性值中包含value的所有元素:[attribute*=value]
                // $("a[href*='s']").css("background","red");//包含s字符的都匹配到
                // 合并多个选择器，满足多个条件 (匹配复选框，并且类名不是abc的盒子回来)
                $("input[type='checkbox']
[class!='abc']").attr("checked","true");

            });

        });
    </script>
</head>
<body>
    <p>您的爱好是：</p>
    <input type="checkbox" name="choice" class="abc" value="篮球">篮球
    <input type="checkbox" name="choice" class="abc" value="足球">足球
    <input type="checkbox" name="choice" id="" value="乒乓球">乒乓球
    <input type="checkbox" name="choice" id="" value="羽毛球">羽毛球

    <input type="button" value="全选" id="all">

    <!-- 匹配某个值开头的字符串 -->
    <a href="http://www.baidu.com">百度</a>
    <a href="https://www.shuaiqi100.com">村长</a>
    <a href="http://www.sina.com">新浪</a>
</body>
</html>
```

实现伸缩隐藏功能：【is()方法用于查看选择的元素是否匹配选择器】

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>显示隐藏列表</title>
    <script src="./js/jquery-3.3.1.min.js"></script>
    <script>
        $(function(){
            $("#open").click(function(){
                //判断当前的状态是否显示
                //is()方法用于查看选择的元素是否匹配选择器 :visible可见选择器
                var flag = $("li:gt(3)").is(":visible");
                //为可见状态返回为true,
                if(flag)
                {
                    $(this).text("展示");
                    $("li:gt(3)").hide(1000);
                }
                else
```

```
                {
                    $("li:gt(3)").show(1000);
                    $(this).text("隐藏");
                }
            });
        });

    </script>
</head>
<body>

    <h3>影视排行榜（<a href="#" id="open">展开</a>）</h3>
    <ul>
        <li>捉妖记</li>
        <li>暴徒</li>
        <li>金港城</li>
        <li>功夫</li>
        <li>小说</li>
        <li>游戏</li>
        <li>其他</li>
    </ul>
</body>
</html>
```

6.4 层次选择器和表单选择器

1. 层次选择器

   层次选择器通过 DOM 元素之间的层次关系获取元素，其**主要的层次关系包括后代、父子、相邻和兄弟关系**。层次选择器由两个选择器组合而成

名称	语法	功能	返回值
后代选择器	selector1 selector2	从 selector1 的后代元素里选取 selector2	元素集合，如 $(“#nav span”) 表示选取 #nav下所有的元素
子选择器	selector1>selector2	从 selector1 的子元素里选取 selector2	元素集合，如 ("#nav>span")表示选取 #nav 的子元素
相邻元素选择器	selector1+selector2	从 selector1 后面的第一个兄弟元素里选取 selector2	元素集合，如 $("h2+dl") 表示选取紧邻 **元素之后的同辈元素**
同辈元素选择器	selector1~selector2	从 selector1 后面的所有兄弟元素里选取 selector2	元素集合，如 $("h2~dl") 表示选取 **元素之后所有的同辈元素**

使用示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>6-4的层级选择器</title>
    <script src="./js/jquery-3.3.1.min.js"></script>
    <script>
        $(function(){
            // 后代选择器【selector1  selector2】：从 selector1 的后代元素里选取 selector2 （祖先与后代）
            $("#box li").css("background","yellow");
            // 子选择器 【selector1>selector2】 父子关系
            $("#box ul>li").css("font-size","20px");
            // 相邻元素选择器 【selector1+selector2】 从 selector1 后面的第一个兄弟元素里选取 selector2
            $("#box ul>li+li").css("font-weight","bold");
            // 同辈元素选择器 【selector1~selector2】 从 selector1 后面的所有兄弟元素里选取 selector2
            $("#box ul>li~li").css("color","red");
        });
    </script>
</head>
<body>

    <div id="box">
        <ul>
```

```
            <li>首尔</li>
            <li>北京</li>
            <li>南宁</li>
            <li>美国</li>
            <li>英国</li>
        </ul>
    </div>


</body>
</html>
```

需要注意的是：

（1）selector1　selector2 与 selector1>selector2 所选择的元素集合是不同的，前者的层次关系是祖先与后代，而后者是父子关系
（2）selector1+selector2 可以使用 jQuery 对象的 next() 方法代替
（3）selector1~selector2 从 selector1 后面的所有兄弟元素里选取 selector2，不能获取前面部分，可以使用nextAll() 方法代替。而 siblings() 方法获取全部（4）的相邻元素，不分前后
（4）selector1　selector2与 selector1:has(selector2) 虽然这两个选择器都要求 selector2 是 selector1 的后代元素，但是前者最终选取的是后代元素，后者最终选取的是拥有这些后代元素的元素

2. 表单选择器

选择器	功能	返回值
:input	获取 <select><button> 元素	

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">

    <title>表单选择器</title>

    <script src="./js/jquery-3.3.1.min.js"></script>

</head>

<body>

    文本表单：<input type="text" name="hello" id="">
<br>

    文本内容：<textarea name="txt" id="" cols="30"
rows="10"></textarea><br>

    密码：<input type="password" name="pass" id=""><br>

    性别：<input type="radio" name="sex" id="">男<br>

    兴趣：  <input type="checkbox" name="hobbit" id="">
篮球<br>

    图片：<input type="image" value="./img/11.png"><br>

    文件：<input type="file" name="file" id="">文件<br>

    可见性：<input type="hidden" name="hide"><br/>

    按钮:<button>普通按钮</button><br>

    提交按钮：<input type="submit" value="提交"><br/>

    重置按钮：<input type="reset" value="重置">

    <script>

        $(function(){

            // 获取 <input><textarea><select><button>
元素

            $(":input").css("background","#ccc");
```

选择器	功能	返回值

	// 获取符合 [type=text] 的 <input> 元素
	$(":text").css("color","#ccc");
	// 获取符合 [type=password] 的 <input> 元素
	$(":password").css("font-size","18px");
	// 获取符合 [type=radio] 的 <input> 元素
	$(":radio").css("width","100px");
	// 获取符合 [type=checkbox] 的 <input> 元素
	$(":checkbox").css("width","100px");
	// 获取符合 [type=image] 的 <input> 元素
	$(":image").css("width","100px");
	// 获取符合 [type=file] 的 <input> 元素
	$(":file").css("background","red");
	// 参考“可见性过滤选择器”
	$(":hidden").css("color","red");
	// 获取 <button> 元素和符合 [type=button] 的 <input> 元素
	$(":button").css("background","orange");
	// 获取符合 [type=submit] 的 <input> 元素
	$(":submit").css("background","yellow");
	// 获取符合 [type=reset] 的 <input> 元素
	$(":reset").css("background","purple");
	});
	
	</script>
	</body>
	</html>
	
	表单对象属性过滤选择器
	表单对象属性过滤选择器也是专门针对表单元素的选择器，**它属于过滤选择器的范畴**，可以附加在其他选择器的后面，主要功能是对所选择的表单元素进行过滤	

选择器	功能	返回值

选择器	功能
:enabled	选取可用的表单元素
:disabled	选取不可用的表单元素
:checked	选取被选中的 元素
:selected	选取被选中的

使用示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>选择器</title>
    <script src="./js/jquery-3.3.1.min.js"></script>
</head>
<body>

    <div id="txt">
        <input type="text" name="" value="不可用表单" disabled
id="">
        <input type="text" name="" value="可用的表单" id="">
    </div>

    <div id="love">
        兴趣爱好：<input type="checkbox" name="love" value="篮
球">篮球
        <input type="checkbox" name="love" value="足球">足球
        <input type="checkbox" name="love" value="乒乓球">乒乓球
        <input type="checkbox" name="love" value="羽毛球">羽毛球
        <input type="button" value="获取标签"
onclick="getCheckbox()">
    </div>

    <div class="choices">
        <select name="" id="choice">
            <option value="white">白色</option>
            <option value="blue">蓝色</option>
            <option value="red">红色</option>
        </select>
        <input type="button" value="改变背景色"
onclick="changeColor()">
    </div>


    <script>
        // :disabled 不可用表单
```

选择器		返回值

```
        $("#txt :disabled").css("background","red");

        // :enabled 表单可用获取
        $("#txt :enabled").css("background","blue");

        function getCheckbox()
        {
            //让选中的复选框隐藏
           $("#love :checkbox:checked").hide();
        }
        function changeColor()
        {
            //下拉框选中的实现
            $("body").css("background",$("#choice
:selected").val());
        }

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

第七章 jQuery 操作 DOM

7.1 DOM 对象和 jQuery 对象

7.2 jQuery 中使用 DOM 操作元素

7.3 jQuery 中使用 DOM 操作节点

7.4 遍历元素

第八章 jQuery 动画与特效