# 基本语法

HTML 中的JavaScript脚本必须位于 <script> 与 </script> 标签之间。

<script>标签可被放置在 HTML 页面的 <body> 和 <head> 部分中。

<script> //标识标签

alert("我的第一个 JavaScript");//脚本语句

</script>

也可以使用外部脚本<script src="myScript.js"></script>

几种输出语句：

* 使用 **window.alert()** 弹出警告框。



* 使用 **document.write()** 方法将内容写到 HTML 文档中。





* 使用 **console.log()** 写入到浏览器的控制台 （类似日志功能）
* 使用**window.confirm()**;输出带有判断的弹出框

如果⽤户单击“确定”，该框返回 true。如果⽤户单击“取消”，该框返回 false。

* 使用 **window.prompt("请输入姓名");**输出提示框

如果⽤户单击“确定”，该框返回输⼊值。如果⽤户单击“取消”，该框返回 NULL。

测试片段

测试输入带有默认值的

**window.prompt("请输入姓名"，”张三”);**

**测试确定/取消**

<input type="button" value="测试" onclick="fun()" />

<script>

function fun(){

var content;

if(confirm("确定 & 取消")){

content\_true ="您选择了确定";

document.write(content\_true);

}else{

content\_false ="您选择了取消";

document.write(content\_false);

}

}

</script>

测试输入框的返回值

<input type="button" value="测试" onclick="fun()" />

function fun() {

var name = prompt("我最喜欢的球员是：", "姚明");

if(name == null || name == "") {

var content\_null = "该⽤户取消了输⼊"

document.write(content\_null);

} else {

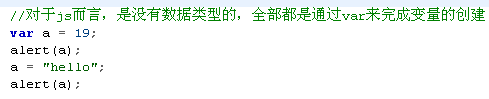
document.write(name + "⽜逼");

}

}

# javascript定义变量

定义变量：首先强调javascript声明任何变量全部用var(甚至没有var都可以，俩者有一点区别)，这说明javascript是一门弱类型的语言，变量没有具体数据类型。



|  |
| --- |
| JavaScript中的数据很简洁的。**简单数据**只有 undefined, null, boolean, number和string这五种，而**复杂数据**只有一种，即object。     JavaScript中的代码只体现为一种形式，就是function。  注意：以上单词都是小写的，不要和Number, String, Object, Function等JavaScript内置函数混淆了。要知道，JavaScript语言是区分大小写的呀!     任何一个JavaScript的标识、常量、变量和参数都只是unfined, null, bool, number, string, object 和 function类型中的一种，也就typeof返回值表明的类型。除此之外没有其他类型了。      先说说简单数据类型吧。      undefined:   代表一切未知的事物，啥都没有，无法想象，代码也就更无法去处理了。                       注意：typeof(undefined) 返回也是 undefined。     null:            有那么一个概念，但没有东西。无中似有，有中还无。虽难以想象，但已经可以用代码来处理了。                       注意：typeof(null)返回object，但null并非object，具有null值的变量也并非object。      boolean:     布尔类型：值跟java一样 只有false trus     number:      数字类型，整形，浮点                                  string:         字符串 |

Javacript的类型

测试以下代码执行结果?



输出number

假设要进行类型转换怎么办呢？



输出111

如果要转换需要：

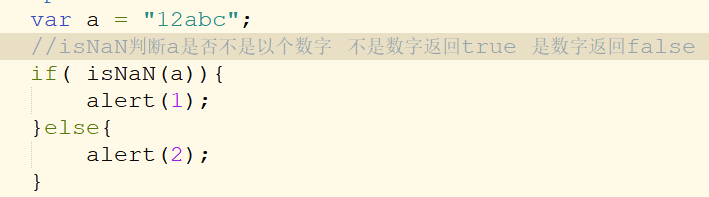


输出12

假设转换一个不是数字的值：



利用NaN可以做一些判断然后分支。。。



NaN表示不是纯数字

经常可能需要设置字体大小，那么就可能用到16px等。font-size:12px :



输出NaN

看到Number(b)没办法转换了，接着往下：



输出12

利用这一点就可以实现通过javascript修改字体大小

# 选择器

* document.getElementById() 通过元素id来查找元素
* document.getElementsByTagName() 通过标签名来查找元素
* document.getElementsByClassName() 通过类名来查找元素
* document.getElementsByName() 通过指定的name值来查找元素

通过id查找，得到的是一个元素，同过标签，类，name属性查找到的是一个数组，不是单个的元素，使用其中的内容需要遍历。

# 修改元素

## 3.1修改元素内容

|  |
| --- |
| <html>  <head>  <script>  function upd(){  // 获取元素对象  var a = document.getElementById("a");  var ipt = document.getElementById("ipt");  a.innerHTML = "京东";  ipt.value = "hello world";  }  </script>  </head>  <body>  <a href="" id="a">百度一下</a>  <input type="text" id="ipt"><br>  <button onclick="upd()">修改内容</button>  </body>  </html> |

## 3.2修改元素样式

元素对象.style.样式 = “样式值”

|  |
| --- |
| <html>  <head>  <script>  function upd(){  // 获取元素对象  var a = document.getElementById("a");  a.style.color = "green";  a.style.backgroundColor = "red";  }  </script>  </head>  <body>  <a href="" id="a">百度一下</a>  <button onclick="upd()">修改内容</button>  </body>  </html> |

## 3.3修改元素属性

|  |  |
| --- | --- |
| 元素对象. setAttribute(“属性名”,”属性值”) | 添加元素属性 |
| 元素对象. removeAttribute (“属性名”) | 移除元素属性 |
| 元素对象. getAttribute(“属性名”) | 获取元素属性 |

|  |
| --- |
| <html>  <head>  <script>  function upd(){  // 获取元素对象  var a = document.getElementById("a");  //添加属性  a.setAttribute("href","http://www.baidu.com");  //移除属性  a.removeAttribute("id");  }  </script>  </head>  <body>  <a href="" id="a">百度一下</a>  <button onclick="upd()">修改内容</button>  </body>  </html> |

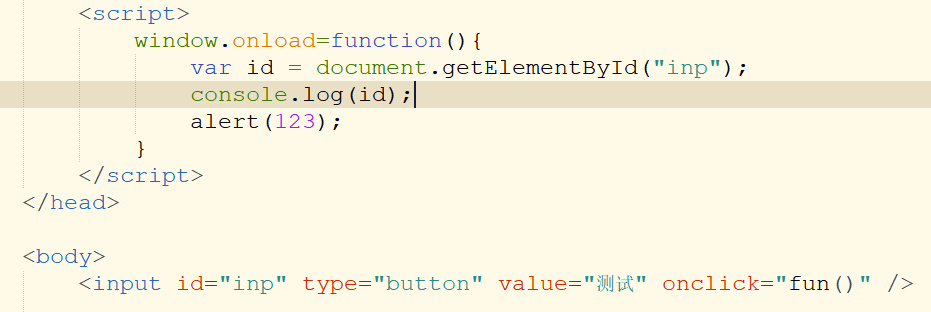
## 3.4添加/删除元素

|  |  |
| --- | --- |
| parentNode | 获取父节点 |
| childNodes | 获取子节点 |
| document. createElement | 创建标签节点 |
| document. createTextNode | 创建文本节点 |
| appendChild | 添加节点 |

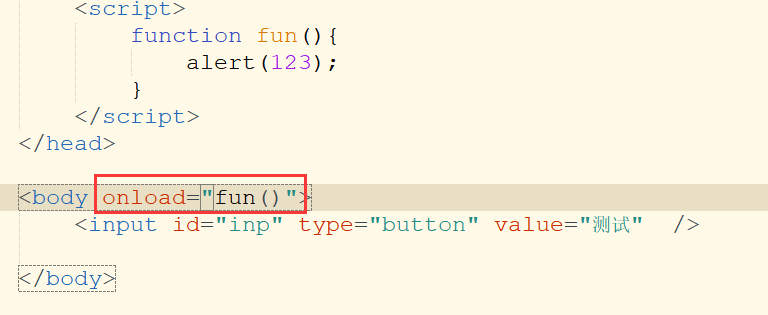
# 事件



事件用法1：



用法2.



Onload是一个页面事件

# 常用工具类

## 4.1date类



|  |
| --- |
| var date = new Date();//当前日期  var year = date.getFullYear();//年份  var month = date.getMonth()+1;//月份（从0开始的所以加1）  var day = date.getDate();//哪一天 （1-31）  var house = date.getHours();//时 24小时制  var min = date.getMinutes();//分  var sec = date.getSeconds();//秒  var d = year+"-"+month+"-"+day+" "+house+":"+min+":"+sec;//组合时间  console.log(d); |

## 4.2定时器

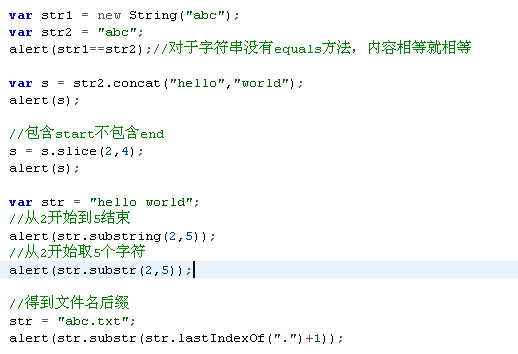
|  |  |
| --- | --- |
| setInterval(function, milliseconds) | 设置定时器 |
| clearTimeout() | 清空定时器 |

|  |
| --- |
| <html>  <head>  <script>  var time;  (function setSpan(){  console.log(222);  time = setInterval(getDate,1000);  })();  function getDate(){  var s = document.getElementById("s");  var date = new Date();  var year = date.getFullYear();  var month = date.getMonth()+1;  var day = date.getDate();  var house = date.getHours();  var min = date.getMinutes();  var sec = date.getSeconds();  month = month<10?"0"+month:month;  day = day<10?"0"+day:day;  house = house<10?"0"+house:house;  min = min<10?"0"+min:min;  sec = sec<10?"0"+sec:sec;  var d = year+"-"+month+"-"+day+" "+house+":"+min+":"+sec;  s.innerHTML = d;  }  //清空定时器  function clears(){  clearTimeout(time);  }  </script>  </head>  <body>  <span id="s"></span>  <button onclick="clears()">清除定时器</button>  </body>  </html> |

## 4.3常用对象String



案例

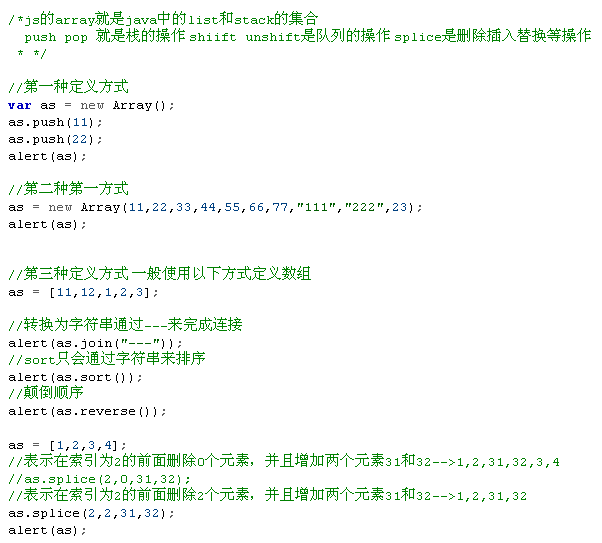


以下为选修，

## 4.4 Array对象



案例



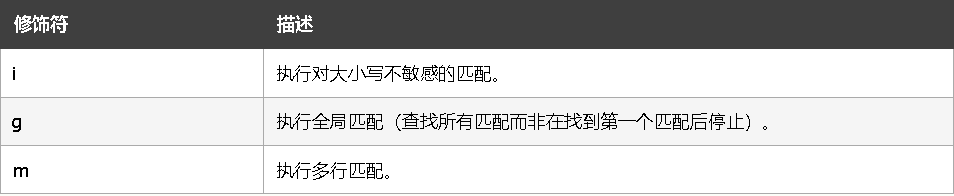
## 4.5 Math对象



## 4.6 Number对象



# 正则表达式









## 1创建正则表达式

**Js代码  [收藏代码](javascript:void())**

1. **var** re = **new** RegExp();//RegExp是一个对象,和Aarray一样
2. //但这样没有任何效果,需要将正则表达式的内容作为字符串传递进去
3. re =**new** RegExp("a");//最简单的正则表达式,将匹配字母a
4. re=**new** RegExp("a","i");//第二个参数,表示匹配时不分大小写

RegExp构造函数第一个参数为正则表达式的文本内容,而第一个参数则为可选项标志.标志可以组合使用   
  
•g （全文查找）   
•i （忽略大小写）   
•m （多行查找） 

**Js代码  [收藏代码](javascript:void())**

1. **var** re = **new** RegExp("a","gi");//匹配所有的a或A

正则表达式还有另一种正则表达式字面量的声明方式

**Js代码  [收藏代码](javascript:void())**

1. **var** re = /a/gi;

测试正则表达式是如何工作的!

**Js代码  [收藏代码](javascript:void())**

1. //test方法,测试字符串,符合模式时返回true,否则返回false
2. **var** re = /he/;//最简单的正则表达式,将匹配he这个单词
3. **var** str = "he";
4. alert(re.test(str));//true
5. str = "we";
6. alert(re.test(str));//false
7. str = "HE";
8. alert(re.test(str));//false,大写,如果要大小写都匹配可以指定i标志(i是ignoreCase或case-insensitive的表示)
9. re = /he/i;
10. alert(re.test(str));//true
11. str = "Certainly!He loves her!";
12. alert(re.test(str));//true,只要包含he(HE)就符合,如果要只是he或HE,不能有其它字符,则可使用^和$
13. re = /^he/i;//脱字符(^)代表字符开始位置
14. alert(re.test(str));//false,因为he不在str最开始
15. str = "He is a good boy!";
16. alert(re.test(str));//true,He是字符开始位置,还需要使用$
17. re = /^he$/i;//$表示字符结束位置
18. alert(re.test(str));//false
19. str = "He";
20. alert(re.test(str));//true
21. //当然,这样不能发现正则表达式有多强大,因为我们完全可以在上面的例子中使用==或indexOf
22. re = /\s/;// \s匹配任何空白字符，包括空格、制表符、换页符等等
23. str= "user Name";//用户名包含空格
24. alert(re.test(str));//true
25. str = "user     Name";//用户名包含制表符
26. alert(re.test(str));//true
27. re=/^[a-z]/i;//[]匹配指定范围内的任意字符,这里将匹配英文字母,不区分大小写
28. str="variableName";//变量名必须以字母开头
29. alert(re.test(str));//true
30. str="123abc";
31. alert(re.test(str));//false

当然,仅仅知道了字符串是否匹配模式还不够,我们还需要知道哪些字符匹配了模式

1. **var** osVersion = "Ubuntu 8";//其中的8表示系统主版本号
2. **var** re = /^[a-z]+\s+\d+$/i; //+号表示字符至少要出现1次,\s表示空白字符,\d表示一个数字

alert(re.test(osVersion));//true,但我们想知道主版本号

1. //另一个方法exec,返回一个数组,数组的第一个元素为完整的匹配内容
2. re=/^[a-z]+\s+\d+$/i;
3. arr = re.exec(osVersion);
4. alert(arr[0]);//将osVersion完整输出,因为整个字符串刚好匹配re
5. //我只需要取出数字
6. re=/\d+/;
7. **var** arr = re.exec(osVersion);
8. alert(arr[0]);//8

## 2更复杂的用法,使用子匹配

1. //exec返回的数组第1到n元素中包含的是匹配中出现的任意一个子匹配
2. re=/^[a-z]+\s+(\d+)$/i;//用()来创建子匹配
3. arr =re.exec(osVersion);
4. alert(arr[0]);//整个osVersion,也就是正则表达式的完整匹配
5. alert(arr[1]);//8,第一个子匹配,事实也可以这样取出主版本号
6. alert(arr.length);//2
7. osVersion = "Ubuntu 8.10";//取出主版本号和次版本号
8. re = /^[a-z]+\s+(\d+)\.(\d+)$/i;//.是正则表达式元字符之一,若要用它的字面意义须转义
9. arr = re.exec(osVersion);
10. alert(arr[0]);//完整的osVersion
11. alert(arr[1]);//8
12. alert(arr[2]);//10

**注意,当字符串不匹配re时,exec方法将返回null**  
  
3String对象的一些和正则表达式有关的方法

**Js代码  [收藏代码](javascript:void())**

1. //replace方法,用于替换字符串
2. **var** str ="some money";
3. alert(str.replace("some","much"));//much money
4. //replace的第一个参数可以为正则表达式
5. **var** re = /\s/;//空白字符
6. alert(str.replace(re,"%"));//some%money
7. //在不知道字符串中有多少空白字符时,正则表达式极为方便
8. str ="some some             \tsome\t\f";
9. re = /\s+/;
10. alert(str.replace(re,"#"));//但这样只会将第一次出现的一堆空白字符替换掉
11. //因为一个正则表达式只能进行一次匹配,\s+匹配了第一个空格后就退出了
12. re = /\s+/g;//g,全局标志,将使正则表达式匹配整个字符串
13. alert(str.replace(re,"@"));//some@some@some@
14. //另一个与之相似的是split
15. **var** str = "a-bd-c";
16. **var** arr = str.split("-");//返回["a","bd","c"]
17. //如果str是用户输入的,他可能输入a-bd-c也可能输入a bd c或a\_bd\_c,但不会是abdc(这样就说他输错了)
18. str = "a\_db-c";//用户以他喜欢的方式加分隔符s
19. re=/[^a-z]/i;//前面我们说^表示字符开始,但在[]里它表示一个负字符集
20. //匹配任何不在指定范围内的任意字符,这里将匹配除字母处的所有字符
21. arr = str.split(re);//仍返回["a","bd","c"];
22. //在字符串中查找时我们常用indexOf,与之对应用于正则查找的方法是search
23. str = "My age is 18.Golden age!";//年龄不是一定的,我们用indexOf不能查找它的位置
24. re = /\d+/;
25. alert(str.search(re));//返回查找到的字符串开始下标10
26. //注意,因为查找本身就是出现第一次就立即返回,所以无需在search时使用g标志
27. //下面的代码虽然不出错,但g标志是多余的
28. re=/\d+/g;
29. alert(str.search(re));//仍然是10

**注意,当search方法没有找到匹配时,将返回-1**   
  
类似于exec方法,String对象的match方法也用于将字符串与正则表达式进行匹配并返回结果数组 

**Js代码  [收藏代码](javascript:void())**

1. **var** str = "My name is CJ.Hello everyone!";
2. **var** re = /[A-Z]/;//匹配所有大写字母
3. **var** arr = str.match(re);//返回数组
4. alert(arr);//数组中只会包含一个M,因为我们没有使用全局匹配
5. re = /[A-Z]/g;
6. arr = str.match(re);
7. alert(arr);//M,C,J,H
8. //从字符串中抽取单词
9. re = /\b[a-z]\*\b/gi;//\b表示单词边界
10. str = "one two three four";
11. alert(str.match(re));//one,two,three,four

RegExp对象实例的一些属性

**Js代码  [收藏代码](javascript:void())**

1. **var** re = /[a-z]/i;
2. alert(re.source);//将[a-z]字符串输出
3. //请注意,直接alert(re)会将正则表达式连同前向斜线与标志输出,这是re.toString方法定义的

每个RegExp对象的实例具有lastIndex属性,它是被查找字符串中下一次成功匹配的开始位置,默认值是-1。 lastIndex 属性被 RegExp 对象的 exec 和 test 方法修改.并且它是可写的.

**Js代码  [收藏代码](javascript:void())**

1. **var** re = /[A-Z]/;
2. //exec方法执行后,修改了re的lastIndex属性,
3. **var** str = "Hello,World!!!";
4. **var** arr = re.exec(str);
5. alert(re.lastIndex);//0,因为没有设置全局标志
6. re = /[A-Z]/g;
7. arr = re.exec(str);
8. alert(re.lastIndex);//1
9. arr = re.exec(str);
10. alert(re.lastIndex);//7

当匹配失败（后面没有匹配），或lastIndex值大于字符串长度时，再执行exec等方法会将lastIndex设为0(开始位置)

**Js代码  [收藏代码](javascript:void())**

1. **var** re = /[A-Z]/;
2. **var** str = "Hello,World!!!";
3. re.lastIndex = 120;
4. **var** arr = re.exec(str);
5. alert(re.lastIndex);//0

RegExp对象的静态属性 （选看）

**Js代码  [收藏代码](javascript:void())**

1. //input 最后用于匹配的字符串（传递给test,exec方法的字符串）
2. **var** re = /[A-Z]/;
3. **var** str = "Hello,World!!!";
4. **var** arr = re.exec(str);
5. alert(RegExp.input);//Hello,World!!!
6. re.exec("tempstr");
7. alert(RegExp.input);//仍然是Hello,World!!!,因为tempstr不匹配
8. //lastMatch 最后匹配的字符
9. re = /[a-z]/g;
10. str = "hi";
11. re.test(str);
12. alert(RegExp.lastMatch);//h
13. re.test(str);
14. alert(RegExp["$&"]);//i  ,$&是lastMatch的短名字，但由于它不是合法变量名，所以要。。
15. //lastParen 最后匹配的分组
16. re = /[a-z](\d+)/gi;
17. str = "Class1 Class2 Class3";
18. re.test(str);
19. alert(RegExp.lastParen);//1
20. re.test(str);
21. alert(RegExp["$+"]);//2
22. //leftContext  返回被查找的字符串中从字符串开始位置到最后匹配之前的位置之间的字符
23. //rigthContext 返回被搜索的字符串中从最后一个匹配位置开始到字符串结尾之间的字符
24. re = /[A-Z]/g;
25. str = "123ABC456";
26. re.test(str);
27. alert(RegExp.leftContext);//123
28. alert(RegExp.rightContext);//BC456
29. re.test(str);
30. alert(RegExp["$`"]);//123A
31. alert(RegExp["$'"]);//C456

multiline属性返回正则表达式是否使用多行模式,这个属性不针对某个正则表达式实例，而是针对所有正则表达式，并且这个属性可写.(IE与Opera不支持这个属性) （选看）

**Js代码  [收藏代码](javascript:void())**

1. alert(RegExp.multiline);
2. //因为IE，Opera不支持这个属性，所以最好还是单独指定
3. **var** re = /\w+/m;
4. alert(re.multiline);
5. alert(RegExp["$\*"]);//RegExp对象的静态属性不会因为给RegExp某个对象实例指定了m标志而改变
6. RegExp.multiline = **true**;//这将打开所有正则表达式实例的多行匹配模式
7. alert(RegExp.multiline);

使用元字符注意事项:元字符是正则表达式的一部分，当我们要匹配正则表达式本身时，必须对这些元字符转义.下面是正则表达式用到的所有元字符   
**( [ { \ ^ $ | ) ? \* + .**

**Js代码  [收藏代码](javascript:void())**

1. **var** str = "?";
2. **var** re = /?/;
3. alert(re.test(str));//出错，因为？是元字符，必须转义
4. re = /\?/;
5. alert(re.test(str));//true

使用RegExp构造函数与使用正则表达式字面量创建正则表达式注意点 

**Js代码  [收藏代码](javascript:void())**

1. **var** str = "\?";
2. alert(str);//只会输出?
3. **var** re = /\?/;//将匹配?
4. alert(re.test(str));//true
5. re = **new** RegExp("\?");//出错,因为这相当于re = /\?/
6. re = **new** RegExp("\\?");//正确，将匹配？
7. alert(re.test(str));//true

既然双重转义这么不友好，所以还是用正则表达式字面量的声明方式   
  
如何在正则表达式中使用特殊字符？ 

**Js代码  [收藏代码](javascript:void())**

1. //ASCII方式用十六进制数来表示特殊字符
2. **var** re = /^\x43\x4A$/;//将匹配CJ
3. alert(re.test("CJ"));//true
4. //也可使用八进制方式
5. re = /^\103\112$/;//将匹配CJ
6. alert(re.test("CJ"));//true
7. //还可以使用Unicode编码
8. re =/^\u0043\u004A$/;//使用 Unicode，必须使用u开头，接着是字符编码的四位16进制表现形式
9. alert(re.test("CJ"));

另处，还有一些其它的预定义特殊字符，如下表所示:   
  
字符    描述   
\n      换行符   
\r      回车符   
\t      制表符   
\f      换页符（Tab）   
\cX     与X对应的控制字符   
\b      退格符(BackSpace)   
\v      垂直制表符   
\0      空字符("")   
  
字符类 ---〉简单类，反向类，范围类，组合类，预定义类 

**Js代码  [收藏代码](javascript:void())**

1. //简单类
2. **var** re = /[abc123]/;//将匹配abc123这6个字符中一个
3. //负向类
4. re = /[^abc]/;//将匹配除abc之外的一个字符
5. //范围类
6. re = /[a-b]/;//将匹配小写a-b 26个字母
7. re = /[^0-9]/;//将匹配除0-9 10个字符之处的一个字符
8. //组合类
9. re = /[a-b0-9A-Z\_]/;//将匹配字母，数字和下划线

下面是正则表达式中的预定义类   
  
  
代码  等同于                  匹配   
.     IE下[^\n]，其它[^\n\r]  匹配除换行符之外的任何一个字符   
\d    [0-9]                   匹配数字   
\D    [^0-9]                  匹配非数字字符   
\s    [ \n\r\t\f\x0B]         匹配一个空白字符   
\S    [^ \n\r\t\f\x0B]        匹配一个非空白字符   
\w    [a-zA-Z0-9\_]            匹配字母数字和下划线   
\W    [^a-zA-Z0-9\_]           匹配除字母数字下划线之外的字符   
  
  
  
量词(下表量词单个出现时皆是贪婪量词)   
  
代码  描述   
\*     匹配前面的子表达式零次或多次。例如，zo\* 能匹配 "z" 以及 "zoo"。 \* 等价于{0,}。   
+     匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于 {1,}。   
?     匹配前面的子表达式零次或一次。例如，"do(es)?" 可以匹配 "do" 或 "does" 中的"do" 。? 等价于 {0,1}。   
{n}   n 是一个非负整数。匹配确定的 n 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。   
{n,}  n 是一个非负整数。至少匹配n 次。例如，'o{2,}' 不能匹配 "Bob" 中的 'o'，但能匹配 "foooood" 中的所有 o。'o{1,}' 等价于 'o+'。

'o{0,}' 则等价于 'o\*'。   
{n,m} m 和 n 均为非负整数，其中n <= m。最少匹配 n 次且最多匹配 m 次。刘， "o{1,3}" 将匹配 "fooooood" 中的前三个 o。'o{0,1}' 等价于 'o?'。请注意在逗号和两个数之间不能有空格。   
  
  
贪婪量词与惰性量词   
  
•用贪婪量词进行匹配时，它首先会将整会字符串当成一个匹配，如果匹配的话就退出，如果不匹配，就截去最后一个字符进行匹配，如果不匹配，继续将最后一个字符截去进行匹配，直到有匹配为止。直到现在我们遇到的量词都是贪婪量词   
•用惰性量词进行匹配时，它首先将第一个字符当成一个匹配，如果成功则退出，如果失败，则测试前两个字符，依些增加，直到遇到合适的匹配为止   
  
惰性量词仅仅在贪婪量词后面加个"?"而已,如"a+"是贪婪匹配的,"a+?"则是惰性的

**Js代码  [收藏代码](javascript:void())**

1. **var** str = "abc";
2. **var** re = /\w+/;//将匹配abc
3. re = /\w+?/;//将匹配a

多行模式

**Js代码  [收藏代码](javascript:void())**

1. **var** re = /[a-z]$/;
2. **var** str = "ab\ncdef";
3. alert(str.replace(re,"#"));//ab\ncde#
4. re =/[a-z]$/m;
5. alert(str.replace(re,"#"));//a#\ncde#

分组与非捕获性分组

**Js代码  [收藏代码](javascript:void())**

1. re = /abc{2}/;//将匹配abcc
2. re = /(abc){2}/;//将匹配abcabc
3. //上面的分组都是捕获性分组
4. str = "abcabc ###";
5. arr = re.exec(str);
6. alert(arr[1]);//abc
7. //非捕获性分组 (?:)
8. re = /(?:abc){2}/;
9. arr = re.exec(str);
10. alert(arr[1]);//undefined

候选（也就是所说的“或”）

**Js代码  [收藏代码](javascript:void())**

1. re = /^a|bc$/;//将匹配开始位置的a或结束位置的bc
2. str ="add";
3. alert(re.test(str));//true
4. re = /^(a|bc)$/;//将匹配a或bc
5. str ="bc";
6. alert(re.test(str));//true

当包含分组的正则表达式进行过test,match,search这些方法之后，每个分组都被放在一个特殊的地方以备将来使用，这些存储是分组中的特殊值，我们称之为反向引用

**Js代码  [收藏代码](javascript:void())**

1. **var** re = /(A?(B?(C?)))/;
2. /\*上面的正则表达式将依次产生三个分组
3. (A?(B?(C?))) 最外面的
4. (B?(C?))
5. (C?)\*/
6. str = "ABC";
7. re.test(str);//反向引用被存储在RegExp对象的静态属性$1—$9中
8. alert(RegExp.$1+"\n"+RegExp.$2+"\n"+RegExp.$3);
9. //反向引用也可以在正则表达式中使用\1 ,\2...这类的形式使用
10. re = /\d+(\D)\d+\1\d+/;
11. str = "2008-1-1";
12. alert(re.test(str));//true
13. str = "2008-4\_3";
14. alert(re.test(str));//false

使用反向引用可以要求字符串中某几个位置上的字符必须相同.另外，在replace这类方法中可用特殊字符序列来表示反向引用

**Js代码  [收藏代码](javascript:void())**

1. re = /(\d)\s(\d)/;
2. str = "1234 5678";
3. alert(str.replace(re,"$2 $1"));//在这个里面$1表示第一个分组1234,$2则表示5678

其它——〉正向前瞻,用来捕获出现在特定字符之前的字符,只有当字符后面跟着某个特定字符才去捕获它。与正向前瞻对应的有负向前瞻，它用匹配只有当字符后面不跟着某个特定字符时才去匹配它。在执行前瞻和负向前瞻之类的运算时，正则表达式引擎会留意字符串后面的部分，然而却不移动index 

**Js代码  [收藏代码](javascript:void())**

1. //正向前瞻
2. re = /([a-z]+(?=\d))/i;
3. //我们要匹配后面跟一个数字的单词，然后将单词返回，而不要返回数字
4. str = "abc every1 abc";
5. alert(re.test(str));//true
6. alert(RegExp.$1);//every
7. alert(re.lastIndex);//使用前瞻的好处是,前瞻的内容(?=\d)并不会当成一次匹配，下次匹配仍从它开始
8. //负向前瞻(?!)
9. re = /([a-z](?!\d))/;i
10. //将匹配后面不包含数字的字母,并且不会返回(?!\d)中的内容
11. str = "abc1 one";
12. alert(re.test(str));
13. alert(RegExp.$1);//one

构建一个验证电子邮箱地址有效性的正则表达式。电子邮箱地址有效性要求(我们姑且这样定义)：用户名只能包含字母数字以及下划线，最少一位，最多25位，用户名后面紧跟@，后面是域名，域名名称要求只能包含字母数字和减号（-），并且不能以减号开头或结尾，然后后面是域名后缀（可以有多个），域名后缀必须是点号连上2-4位英文字母

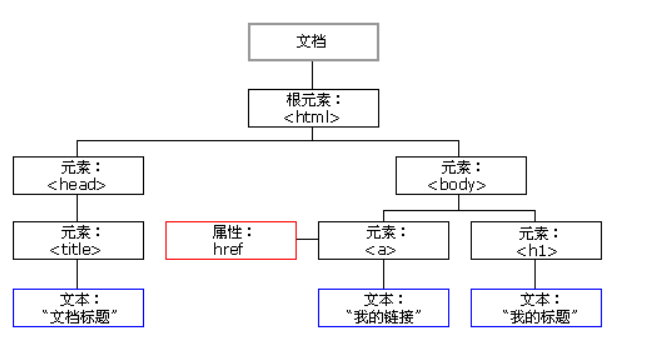
**Js代码  [收藏代码](javascript:void())**

1. **var** re = /^\w{1,15}(?:@(?!-))(?:(?:[a-z0-9-]\*)(?:[a-z0-9](?!-))(?:\.(?!-)))+[a-z]{2,4}$/;

# 八.DOM概述

当网页被加载时，浏览器会创建页面的文档对象模型（****D****ocument ****O****bject ****M****odel）。

HTML ****DOM**** 模型被结构化为 ****对象树****



通过这个对象模型，JavaScript 获得创建动态 HTML 的所有力量：

JavaScript 能改变页面中的所有 HTML 元素

JavaScript 能改变页面中的所有 HTML 属性

JavaScript 能改变页面中的所有 CSS 样式

JavaScript 能删除已有的 HTML 元素和属性

JavaScript 能添加新的 HTML 元素和属性

JavaScript 能对页面中所有已有的 HTML 事件作出反应

JavaScript 能在页面中创建新的 HTML 事件

换言之：HTML DOM 是关于如何获取、更改、添加或删除 HTML 元素的标准

节点Node，是构成我们网页的最基本的组成部分，网页中的每一个部分都可以称为是一个节点。

比如：html标签、属性、文本、注释、整个文档等都是一个节点。

虽然都是节点，但是实际上它们的具体类型是不同的。

比如：标签我们称为元素节点、属性称为属性节点、文本称为 文本节点、文档称为文档节点。

节点的类型不同，属性和方法也都不尽相同。

节点：Node——构成HTML文档最基本的单元。

常用节点分为四类：

文档节点：整个HTML文档

元素节点：HTML文档中的HTML标签

属性节点：元素的属性

文本节点：HTML标签中的文本内容

