

XDL兄弟连IT教育

www.itxd.cn



LAMP高级阶段：JAVASCRIPT

讲师：刘田 <1538731090@qq.com>

版权归兄弟连所有•2015年版

□ 章节大纲

- 第1天：JS引入和基础语法
- 第2天：JS内建消息框、循环语句和函数
- 第3天：函数的参数、作用域和部分内建函数
- 第4天：数组对象和数学对象
- 第5天：BOM对象访问
- 第6天：DOM对象访问
- 第7天：DOM节点操作
- 第8天：事件处理
- 第9天：AJAX请求

第1天：JS引入和基础语法

- JS概述和引入
- JS语法：变量、数据类型、运算符
- 流程控制语句
- 制作简单计算器

```
<script>
```

```
function f() {  
    var name = prompt("点谁出台?", "");  
    if (name) {  
        var flag = confirm("确认点" + name + "吗?");  
        if(flag)    alert("我来也");  
        else       alert("来嘛, 大爷!");  
    }  
}
```

```
</script>
```

```
<input type="button" value="点我" onclick="f();" />
```

□ JAVASCRIPT简介

人们通常所说的JavaScript，其正式名称为ECMAScript。这个标准由ECMA组织发展和维护。**ECMA-262**是正式的JavaScript标准。这个标准基于网景（Netscape）公司提出的JavaScript语言和微软提出的JScript语言。

JavaScript是一种基于对象（Object）和事件驱动（Event Driven）并具有安全性能的脚本语言。使用这种语言的目的是：**与Web客户交互作用，美化页面等。**

- JavaScript常被直接嵌入HTML页面，向页面添加交互行为
- JavaScript是一种轻量级、解释性语言，由JS解释器解析和执行。
- JavaScript是一种基于对象和事件驱动的语言
- JavaScript和Java 是完全不同的两种语言

JS的定义

- 使用<**script**> 标签，分隔页面中的HTML和JS代码

A、完整定义

```
<script type="text/javascript">  
    //JavaScript代码  
</script>
```

B、简写定义(推荐)

```
<script>  
    // JavaScript代码  
</script>
```

JS的引入

□ JS可以在页面中的不同位置被引入，这决定它的执行顺序

- head头部定义

JS脚本代码位于 head 头部分

文档内容加载前执行，执行完成后再加载body

- body区域定义

JS脚本代码位于 body 部分

按照body中的代码位置，顺序执行

- 外部引入JS文件

JS脚本代码存放于单独的外部文件中（*推荐采用）

目的：减少http请求数，和css代码独立成单独的外部css文件原理一样

JS调试工具

□ JS的错误提示不直接输出在页面 你需要专业的查看和调试工具

- 使用**Firefox**火狐浏览器执行JS代码

- 使用火狐**FireBug**插件查看JS错误 

- 1、打开火狐 在地址栏执行 about:addons 进入插件搜索页

- 2、在右上角输入框中查找"firebug"，显示后进行下载

- 3、重启浏览器 按F12查看工作台



JS语法：输出语句

□ 一行语句有内容和分号组成

- document.write() — 输出内容至HTML
- document.writeln() — 带换行的输出

```
<script>
//使用document.write()的方法向页面输入内容
document.write("hello world!");
document.write(" <i>hello world!</i>");
//同时输出多个内容
document.write("Li", "&nbsp;De", " &nbsp;lin");
//带有换行的输出
document.writeln("this is a line");
document.writeln("this is another line");
</script>
```

- 注释：单行// ... 多行 /* ... */

JS语法：输出语句

- 使用document.write()方法输出如下效果：

姓名：

JS语法：变量

□ 变量—存储信息的容器

- 变量的组成：变量名和值
- 声明变量：**var** 关键字

```
var num= 123;           //变量赋值  
var str= "string";  
var boo = true;  
var nul= null;  
document.writeln(num);
```

- 变量名称规则：

变量名遵循标识符统一规范，以字母、下划线或"\$"开头，且不能是**保留关键字**
JS中名称对**大小写敏感**（y 和 Y 是两个不同的变量）
变量命名需要见名知意

JS语法：变量

□ 示例：

```
<script>  
undeclare = "hello";  
var unknow;  
</script>
```

- 变量定义可以用 **var** 声明 不经 **var** 声明的变量默认是【全局作用域】
- 声明但未赋值的变量默认值是 **undefined**
- 引用一个未定义的变量 会报错
document.writeln(tmp);
✗ ReferenceError: Can't find variable: tmp
- 删除变量 使用 **delete** 运算

JS语法：数据类型

□JS中的基本数据类型

- 数值型（整数或实数）
- 字符串型（用双引号或单引号括起来）
- 布尔型（true或false）
- 空值型（null）

```
<script>
var num= 123;
var str= "string";
var boo = true;
var nul= null;
document.writeln(typeof num); //使用typeof查看变量的数据类型
</script>
```

JS语法：数据类型

□ JS中的复合数据类型

- **数组**：经过编号的数据的集合

```
var arr = ["lucy", "female", 23];  
arr["addr"] = "Anhi,China";  
document.writeln(typeof arr);    //object  
document.writeln(arr[0]+"的年龄是：" +arr[2]);
```

- **对象**：经过命名的数据的集合

```
var obj = {name:"Lucy", sex:"female", age:23}  
document.writeln(typeof obj);    //object  
document.writeln(obj.name+"的年龄是：" +obj.age);
```

- **数组和对象为引用类型**

```
var arr2 = arr;           arr2[2] = 32;           document.writeln(arr[2]);    //32
```

JS语法：运算符

□ 运算符是用来完成赋值、计算等操作的一系列符号

```
<script>  
var x = 5;  
var y = 2;  
var z = x + y;  
</script>
```

1. 算术和赋值运算符
2. 特殊的加号+运算符
3. 比较和逻辑运算符
4. 条件运算符（三目运算符）

JS语法：算术运算符

□ 算术运算符用法详见下面的表格，给定 $y=5$

运算符	描述	例子	结果
+	加	$x=y+2$	$x=7$
-	减	$x=y-2$	$x=3$
*	乘	$x=y*2$	$x=10$
/	除	$x=y/2$	$x=2.5$
%	求余数 (整数保留)	$x=y\%2$	$x=1$
++	累加	$x=++y$	$x=6$
--	递减	$x=--y$	$x=4$

JS语法：算术运算符

□ 特殊的“加号” + 运算符

- 用于字符串连接的 + 运算符

```
var str1= "hello";  
var str2= "world";  
document.write(str1 + str2);
```

- 对数字进行加法运算的 + 运算符

```
var num1 = 123;  
var num2 = 456;  
document.write(num1 + num2);
```

- 字符串和数字相加？！

```
document.write(str1 + num1);
```

□ 说出下面JS代码的结果：

```
<script>
var res1 = 123 + "321";
document.writeln(res1);      //res1输出结果_____

var res2 = 123;
res2 += 321;
document.writeln(res2);      //res2输出结果_____

var res3 = "123";
res3 += "123";
document.writeln(res3);      //res3输出结果_____

var res4 = 1+2+3;
document.writeln(res4++);     //res4输出结果_____
document.writeln(++res4);     //res4输出结果_____
</script>
```

□ 几种基本数据类型之间的转化

	数值	字符串	布尔值
数值		使用单双引号； 进行 “+” 运算； String()函数和 toString()方法	使用Boolean()函数 ； 非0 转换为true， 0 转化为false；
字符串	进行 “-” 运算； Number()和 parseInt()函数；		非空为true； 空为false；
布尔值	true转化为1； false转化为0	字符串的 “true” 和 “false”	

让学习成为一种习惯!

JS语法：比较运算

□ 给定 $x=5$ ，下面的表格解释了比较运算符：

运算符	描述	例子
==	(值) 等于	$x==8$ 为 false
===	(值和类型) 全等	$x===5$ 为 true ; $x===\text{"5"}$ 为 false
!=	不等于	$x!=8$ 为 true
>	大于	$x>8$ 为 false
<	小于	$x<8$ 为 true
>=	大于或等于	$x>=8$ 为 false
<=	小于或等于	$x<=8$ 为 true

JS语法：逻辑运算

□ 给定 $x=6$ 以及 $y=3$ ，下表解释了逻辑运算符：

运算符	描述	例子
&&	与 (and)	$(x < 10 \ \&\& \ y > 1)$ 为 true
 	或 (or)	$(x==5 \ \ y==5)$ 为 false
!	非 (not)	$!(x==y)$ 为 true

□ 三目运算符 (条件运算)

```
var str= null;  
var txt = (str != null) ? str : "str为空值";  
document.write(txt);  
document.write(str || "str为空值");    //替换写法
```

JS语法：逻辑运算

- 请说出下面JS代码的执行结果，并验证

```
var a = 10;  
var b = 20;  
!(a == b) ? c = true : c = false;  
document.write( c == true && a == b ? "c为true" : "c为false" );
```

- 运算符优先级：

赋值运算 < 条件运算(? :) < 逻辑运算 < 算数运算符 < 引用运算(. [] ())

JS语法：流程控制

□ 用于完成不同条件下的程序流程控制

- if 语句
在一个指定的条件成立时执行代码
- if...else 语句
在指定的条件成立时执行代码，当条件不成立时执行另外的代码
- if..else if...else 语句
使用这个语句可以选择执行若干块代码中的一个
- switch 语句
使用这个语句可以选择执行若干块代码中的一个

```
语法：  if (条件) {  
                                           //条件成立时执行此代码  
        } else {  
                                           //条件不成立时执行代码  
        }
```

JS语法：switch语句

□ 如果希望选择执行若干代码块中的一个，你可以使用 switch 语句：

● 语法：

```
switch( 条件 或 变量 ) {  
    case 结果1:  
        执行代码块 1  
        break;  
    case 结果2:  
        执行代码块 2  
        break;  
    ... ..  
    case 结果N:  
        执行代码块 N;  
        break;  
    default:  
        以上条件均不满足时 默认执行的代码  
}
```

```
var c = "5"; //将此行代码再改为 var c = 5;  
  
switch (c) {  
    case 5 :  
        document.write("数字5");  
    case '5' :  
        document.write("字符串5");  
        break;  
    default:  
        document.write("c=5");  
}
```


□ 使用JS制作简单的计算器

- 效果如下：



第一个数:

第二个数:

结果为:

```
/**  
 * 给按钮绑定事件 匿名函数的写法  
 */  
document.forms.cal.plus.onclick = function () {  
    //获取用户输入值  
    var m = parseFloat(document.forms.cal.m.value);  
    var n = parseFloat(document.forms.cal.n.value);  
    //计算结果  
    var result = m + n;  
    //将结果放入文本框  
    document.forms.cal.result.value = m + " + " + n +  
    " = " + result;  
}
```

第2天：JS内建消息框、循环语句和函数

- JS内建消息框
 - 警告框
 - 确认框
 - 提示框
- JS循环语句
 - while循环
 - for循环
- 函数
 - 函数的定义

JS内建消息框

□ JS内置了三种消息框：

- 警告框：输出提示信息
- 确认框：用户可选择的确认信息
- 提示框：支持用户输入内容框

JS内置警告框

- 警告框经常用于确保用户可以得到某些信息。
 - 当警告框出现后，**用户需要点击确定按钮才能继续进行操作。**



- 语法：**alert('提示语句');**

```
alert("hello world");
```

//单行内容提示框

```
alert("hello" + "\n" + "world");
```

//多行内容提示框

JS内置确认框

- 确认框用于使用户可以验证或者接受某些信息。
 - 当确认框出现后，用户需要点击后才能继续进行操作。
 - 如果用户点击确认，返回值为 true；
 - 如果用户点击取消，返回值为 false；
 - 语法：**confirm('提示语句');**

```
var result = confirm("点我吗?");  
if (result == true) {  
    alert("感谢你点我，我会服务好的");  
} else {  
    alert("100块都不给我");  
}
```



JS内置提示框

□ 提示输入框经常用于提示用户在进入页面输入某个值。

- 语法：`prompt('提示语句', ['默认输入值']);`

```
var result = prompt("请输入你的微信号：", "");  
if(result != null) {  
    alert(result);  
} else {  
    alert("你未输入值或者点击了取消");  
}
```



- 如果用户点击确定则返回值，点击取消则返回NULL

几种消息框用法案例

```
<script>
function f() {
    var name = prompt("点谁出台?", "");
    if (name) {
        var flag = confirm("确认点" + name + "吗?");
        if(flag)    alert("我来也");
        else       alert("来嘛，大爷!");
    }
}
</script>
```

JS语法：循环语句

□ 常见循环语句：

- while 循环
- do while 循环
- for 循环
- 循环中的continue、break语句

JS语法：循环语句

□ 满足某个条件时执行一段功能代码

● 语法：

```
while ( 循环条件 ) {  
    //循环条件为真时运行的代码  
}
```

```
var i=1;  
while(i<=10) {  
    document.write("第" + i + "行<BR>");  
    i++; //循环步进值累加  
}
```

□ 执行一段功能代码直到不满足条件时 语法：

```
do{  
    //循环条件为真时运行的代  
    码  
}while ( 循环条件 )
```

```
var i=1;  
do{  
    document.write("第" + i + "行<br>");  
    ++i; //循环步进值累加  
}while(i<=10)
```

JS语法：循环语句

□ 将循环的执行要素在一个语句中执行

- 语法：

for (循环开始值; 循环满足条件; 循环步进) {

 //在循环开始后反复执行的代码

 //直到不再满足循环条件，则不再反复执行

 //结束for循环

}

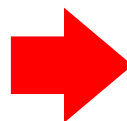
```
for(var i=1;i<=10;i++) {  
    document.write("第" + i + "行<br>");  
}
```

- 循环中使用**continue**表示跳过本次循环继续下一循环
- 循环中使用**break**表示终止循环

JS语法：循环语句

- 用do while循环写如下程序：
 - 当用户输入exit字符串时循环终止退出
 - 当用户输入数字时，循环终止退出
- 使用for循环输出如下月份选框

请选择月份 ▼



请选择月份 ▼

请选择月份

1月

2月

3月

4月

5月

6月

7月

8月

9月

10月

11月

12月

JS语法：循环的嵌套

- 循环的嵌套是在一个循环体里嵌套另外一个循环

```
for(var i=1;i<=10;i++) {  
    for(var j = 1;j<=5;j++) {  
        document.write(i + "-" + j);  
    }  
    document.write("<BR>");  
}
```

JS语法：循环的嵌套

□ 用循环嵌套的方法，打印出如下显示效果页面

1*1=1									
1*2=2	2*2=4								
1*3=3	2*3=6	3*3=9							
1*4=4	2*4=8	3*4=12	4*4=16						
1*5=5	2*5=10	3*5=15	4*5=20	5*5=25					
1*6=6	2*6=12	3*6=18	4*6=24	5*6=30	6*6=36				
1*7=7	2*7=14	3*7=21	4*7=28	5*7=35	6*7=42	7*7=49			
1*8=8	2*8=16	3*8=24	4*8=32	5*8=40	6*8=48	7*8=56	8*8=64		
1*9=9	2*9=18	3*9=27	4*9=36	5*9=45	6*9=54	7*9=63	8*9=72	9*9=81	

让学习成为一种习惯!

JS语法：for...in语句

□ 对象属性和数组元素的循环

- 定义：**for** (var property **in** object) {}

```
var o = {  
    name: "Lucy",  
    age: 23,  
    sex: "female"  
};  
  
for (var property in o) {  
    //此处不可使用o.property硬编码 使用o[property]替代  
    document.write(property + "::" + o[property] + "<br />");  
}
```

JS语法：函数

- 定义函数
- 作为**数据类型**的函数
- 函数参数
- 函数作用域
- 函数的**属性**

JS语法：函数

□ 函数简介

- 计算机中的函数是一个**可以反复使用**的程序段
- 从其他的程序段中可以通过该程序段的名称来调用这段程序
- 语法：

```
返回值类型 function 函数名 ( 参数1[, 参数2, ...] ) {  
    //函数内部代码  
}
```

```
function dw(text) {  
    document.write(text);  
}  
dw();
```

函数名遵从标识符统一规范，一般推荐使用**驼峰命名法**

JS语法：定义函数

□ 使用**function**关键字定义函数：

自定义函数sum：

- 1、求两个数x,y之和
- 2、计算1~n之间的整数和

自定义函数max：

- 3、求两个数m,n最大值的函数
- 4、求一组数的最大值

自定义函数sort：

- 5、对一组数按从小到大排序

JS语法：定义函数

□ 函数是一种特殊的对象数据类型

```
function f() {  
    var a = 1;  
}
```

```
alert(typeof f);           //function
```

```
alert(f.toString());      //查看函数体
```

此处调用了f函数原型（Function）的toString()方法 说明函数也是一个对象，是一种数据类型。

JS语法：作为数据类型的函数

□ 作为数据类型的函数 - 匿名函数

- 定义

```
var f = function () {  
    alert("hello,我出身匿名函数！");  
}  
f(); //调用函数
```

- 当把函数赋值一个对象的属性时即为方法。

```
var graph = {background:red};  
graph.square = function (width,height) {  
    return width*height;  
}  
dw(graph.square(100,200));
```

```
<input type="button" id="btn" value="点我" />  
<script>  
document.getElementById("btn").onclick = function ()  
{  
    alert("点你等于点ji！");  
}  
</script>
```

JS语法：创建函数对象

□ 使用构造函数定义函数（**不推荐**）

- 定义：`new Function`(参数列表, 函数体);

```
var sum = new Function ("x,y", "return x + y");  
dw(sum(1,-1));           //调用并输出结果
```

JS语法：函数参数

□ JS函数的参数：

- 调用时可以不严格匹配 如果缺省则函数体内使用形参默认值为 undefined, 如果多出参数则忽略

```
function f(x, y) {  
    document.write(x, ":", y);  
}  
//调用并传参  
f(1,"a");
```

JS语法：函数参数

□ 类似数组的对象：Arguments对象

- 一个封装了函数实参作为属性的对象：

属性：length 实参的个数

```
function _max() {  
    var m = arguments[0];  
    for (var i = 1; i < arguments.length; i++) {  
        if (arguments[i] > m) {  
            m = arguments[i];  
        }  
    }  
    return m;  
}
```

dw(_max(1,2,3,4,5)); //输出 5 ;

- 函数体外无法使用arguments对象

dw(arguments); //arguments is not defined

JS语法：函数参数

□ 使用对象作为函数参数

- 减少参数列表的长度，避免记参数顺序

```
function greet(name,age,from,email) {  
    return "i am" + name + ", 今年" + age + "岁，来自" + from + "请多关照，大  
    家可以记一下我的邮箱 方便联系" + email;  
}
```

//参数使用对象替代

```
function geeting (introduce) {  
    return "i am" + introduce.name+ ", 今年" + introduce.age+ "岁，来自" +  
    introduce.from+ "请多关照，大家可以记一下我的邮箱 方便联系" + introduce.email;  
}
```

JS语法：函数参数

□ 功能函数小助手

- 定义一个复制对象属性到数组的函数

```
function copyPropertyNameToArr(o, a) {  
    var arr = a || [];  
    for (var p in o) {  
        arr.push(p);  
    }  
    return arr;  
}
```


JS语法：函数参数

□ wbox弹出层的使用，加载插件的一般过程：

- 将插件下载包引入项目
- 在页面中引入相应的JS和CSS文件
- 绑定页面元素
- 设置配置

登陆



用户登陆

账号:

密码:

登陆

□ 变量的作用域：

- 全局变量
- 局部变量

```
var txt = "外部的全局变量值";  
function myDate(y, m, d) {  
    alert(txt); //同名的局部变量，产生了歧义  
    var txt = y + "年" + m + "月" + d + "日";  
    return txt;  
}
```

```
function myDate(y, m, d) {  
    var txt = y + "年" + m + "月" + d + "日";    //txt, y, m, d局部变量  
    return txt;  
}
```

```
var v1 = prompt("year: ", 2009);    // v1全局变量  
var v2 = prompt("month: ", 11);    // v2全局变量  
var v3 = prompt("day:", 05);    // v3全局变量  
var str = myDate(v1, v2, v3);  
document.write("您输入的日期是"+str);
```

JS语法：变量作用域

- ❑ 函数体内的变量为局部变量，只能作用于函数体内；函数体内未经var声明的变量为全局变量 在整个脚本中都有效。
- ❑ 全局变量也可作用于函数体内 但默认会先从函数体内寻找变量 没有则使用全局
- ❑ 函数体内经过var声明的变量在整个函数体内有效

```
var scope = "global";  
function f() {  
    //var scope = "local":  
    scope = "local";  
}  
f();  
alert(scope);
```

```
var scope = "global";  
function f() {  
    alert(scope);  
    var scope = "local";  
}  
f();
```

JS语法：函数属性

□ function函数的属性

- length属性：获取形参的个数
- 自定义函数属性

```
f.count = 0;  
function f(x, y) {  
    alert(++f.count);  
}  
f();  
f();  
f();
```

JS语法：对象

□ JavaScript 是面向对象的编程语言 (OOP)。

- 对象是一种特殊的数据类型。对象拥有属性和方法。

属性——描述对象特性的变量

```
var txt = "Hello World ! ";  
document.write(txt.length);           //字符串对象的长度属性
```

方法——对象用来完成某种操作的函数

```
var txt="Hello World!"  
document.write(txt.toUpperCase( ));    //字符串对象的大写转换方法
```

JS语法：对象

□ JS对象的构成：

- 对象由属性和方法构成
- 属性的调用：

对象.属性



属性可以看做对象的**变量**

- 方法的调用：

对象.方法([参数])



方法可以看做 对象的**函数**



中括号在语法描述中表示可选

- 实例：

老王.年龄;

老王.交谈('吃了吗?');

JS语法：常见内建对象

□ JavaScript内建的常见对象原型（类）

- 字符串对象
- 日期对象
- 数组对象
- 数学对象
- BOM对象
- DOM对象

JS内建字符串对象

□ String 对象用于处理字符型数据

- 语法：

```
var txt = new String("Hello World");
```

- 属性：

属性名称	说明	语法示例
length	length 属性包含一个整数，用来指出 String 对象中的字符数。	"Hello".length

□ 字符串对象常见方法

方法名称	说明	语法示例
indexOf	返回字符串中 <u>第一次</u> 出现子字符串的字符位置。 返回-1代表不存在	"helloWorld".indexOf("World")
lastIndexOf	返回字符串中 <u>最后一次</u> 出现子字符串的字符位置。	"helloWorld".lastIndexOf("o")
replace	返回字符串进行文字替换后的内容（只替换第一个匹配结果）	"helloWorld".replace("o", "O")
split	将字符串分割为子字符串，然后将结果（数组）作为字符串数组返回	"helloWorld".split("o")
substr	返回一个从指定位置开始的指定长度的子字符串。	"helloWorld".substr(1, 5)
toLowerCase	返回转换为小写字母的字符串	"helloWorld".toLowerCase()
toUpperCase	返回转换为大写字母的字符串	"helloWorld".toUpperCase()

JS内建字符串对象

1. 用indexOf方法显示字符串"Welcome !"中字母m出现的字符位置数。
2. 用lastIndexOf方法显示字符串"Welcome !"中字母e最后出现的字符未知数。
3. 用replace方法将字符串"Welcome !"中的感叹号"!"改为星号(*)。
4. 用split方法将字符串"Welcome !"用"|"分割为数组。
5. 用substr方法将字符串"Welcome !"截取出结果"come"。
6. 用toLowerCase和toUpperCase方法将字符串"Welcome !"转化为全部字符的小写和大写。

JS内建字符串对象

□ 字符串的多次或不区分大小写替换：

- 正则——高级替换（**仅作了解**）


```
var str = "lamp兄弟连是专业的php培训机构，学习PHP的最佳选择";  
alert(str.replace(/php/, "P-H-P"));    //只替换第一个小写的php  
alert(str.replace(/php/g, "P-H-P"));    //替换所有的小写php字符串  
alert(str.replace(/php/gi, "P-H-P"));    //替换所有的php字符串,不区分大小写
```

JS内建日期时间对象

□ 取得系统当前的（或指定年月日的）日期和时间。

□ 语法实例：

```
var d1 = new Date();           //当前时间  
var d2 = new Date(2015, 11, 6); //指定日期
```



注意：JS的月份从0开始
这里的11代表12月份

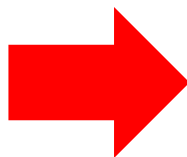
方法名称	说明	<code>var d = new Date();</code> 语法示例
getFullYear	返回 Date 对象中的年份值	d.getFullYear()
getMonth	返回 Date 对象中的月份值(真实月份-1)	d.getMonth()
getDate	返回 Date 对象中的日期值	d.getDate()
getHours	返回 Date 对象中的小时值	d.getHours ()
getMinutes	返回 Date 对象中的分钟值	d.getMinutes()
getSeconds	返回 Date 对象中的秒数值	d.getSeconds ()
getMilliseconds	返回 Date 对象中的毫秒数值 所返回的毫秒值处于 0-999 之间	d.getMilliseconds ()
getTime	返回一个整数值，这个整数代表了从 1970 年 1 月 1 日开始计算到 Date 对象中的时间之间的 毫秒数 （unix时间戳）	d.getTime()
getDay	返回Date对象中的星期值 周一到周六为1-6的数字；周日以0表示	d.getDay()

JS内建日期时间对象

- 写一个自定义函数，用来计算当前日期与输入日期的天数差距，要求：
- 输入日期使用插件完成
 - 日期差值可以进位取整

日期:

计算距今天数



日期:

计算距

« « 三月 2016 » »

日	一	二	三	四	五	六
28	29	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

🔄 清空 今天 确定

JS内建对象：数组

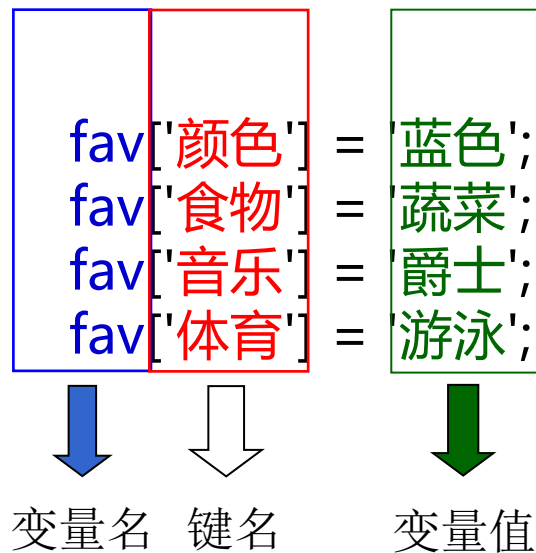
□ 什么是数组

- 同一类型的所有数据的一个集合，并用索引来区分或指定每一个数据元素

爱好	我的选择
颜色	蓝色
食物	蔬菜
音乐	爵士
体育	游泳

左右两列的对应关系
——键值对

什么是数组：
——多个键值对的集合



让学习成为一种习惯!

JS内建对象：定义数组

□ 创建数组

- 数组直接量

```
var arr = [];  
var arr = [1, "a", true];
```

- 显式的创建

```
var arr = new Array();  
var arr = new Array(1, "a", true);  
//当传入一个数字时 表示数组长度  
var arr = new Array(10);  
document.writeln(arr);
```

- 数组对象的属性：

```
document.write(arr.length); //查看数组元素的个数
```


JS内建对象：数组访问

□ 访问数组元素：

- 通过下标或者索引给数组元素赋值

```
arr[0] = 1;  
arr[1] = "a";  
arr[2] = true;
```

```
arr["name"] = "jac";  
arr["sex"] = "male";  
arr["age"] = 23;
```

- 读取数组元素的值：

```
document.writeln(arr[0], arr[1], arr[2]);
```

```
document.writeln(arr["name"], arr["sex"], arr["age"]);
```

- **FAQ?**

使用length属性查看数组元素个数

使用document.write()输出整个数组

JS内建对象：数组

□ 循环获取数组元素

- 索引是连续的数字下标

```
for (var i = 0; i < arr.length; i++) {  
    document.write(i, ":", arr[i], "<br />");  
}
```

- 索引是字符串

```
for (var key in arr) {  
    document.write(key, ":", arr[key], "<br />");  
}
```

□ 数组的常见方法：

```
var arr = [1, "hello", 2, "b"];  
var arr1 = new Array("world", 3);
```

方法名	说明	返回值类型	语法示例
concat	连接两个或多个数组	数组	arr3 = arr1.concat(arr2)
join	将数组元素用字符串连接起来	字符串	str = arr1.join("-")
pop	弹出数组最后一个元素并返回弹出的元素	元素类型	num1 = arr2.pop()
push	将一个新元素压入数组尾部，并返回新数组的元素个数	数字型	len = arr2.push(7) alert(len)
unshift	将一个新元素压入数组头部，并返回新数组的元素个数	IE6 (undefined) FF (数字型，新数组元素个数)	num = arr1.unshift(0);
shift	移除数组的第一个元素并返回	元素类型	num2 = arr1.shift()
reverse	返回一个顺序相反的数组	数组	arr5 = arr1.reverse()
sort	对数组进行排序（自定义函数排序）	数组	arr6 = arr2.sort()

JS内建对象：数组的应用

1. 定义一个数组，元素值从1-20，用左尖括号"<"连接为一个字符串，并输出
2. 将0数字压入arr数组的开头，将数字21存入arr数组的末尾；
3. 声明arr2数组，内容是arr的反序内容；输出arr2的所有元素值

要求：

- 用for循环输出，元素间用连字符"-"分隔；
- 最后一个元素用句号"."结束；

JS内建对象：数组的应用

□ 计算机找零钱：

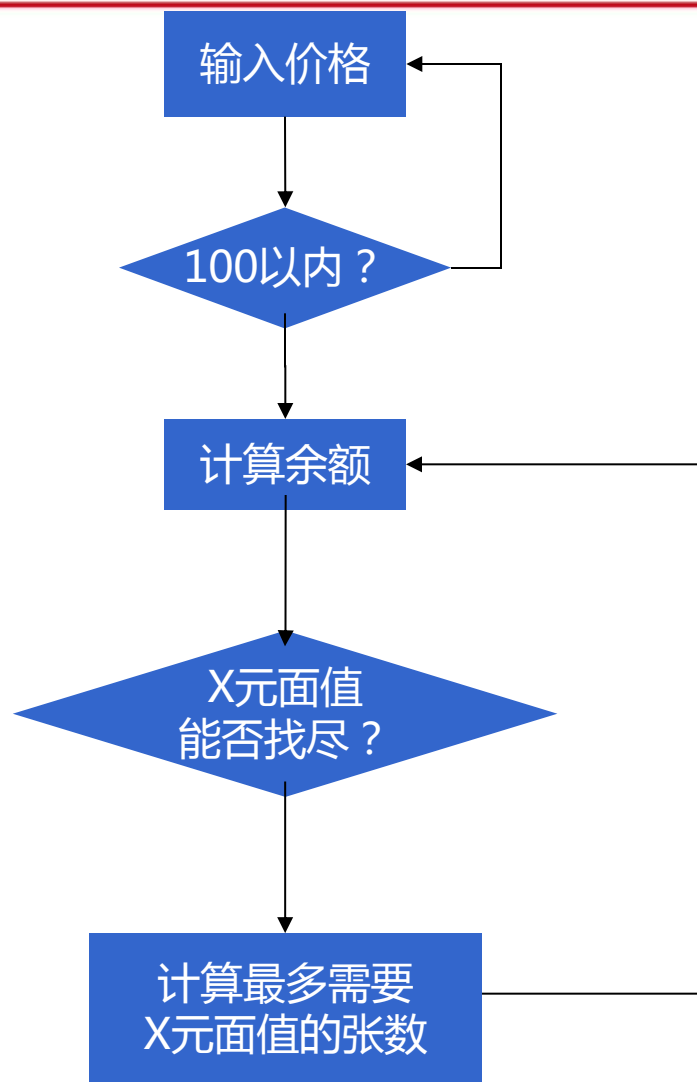
- 假设您目前一共有100元可够支付；
- 通过输入语句，告诉计算机购买的商品价格（只考虑整数），100元以内，若超过100元，则判断价格超过现有支付能力，提示客户无法购买，重新输入；（价格若为0或负数则 提示：价格错误）；点击“取消”则程序结束
- 计算机自动计算找零的金额，要求如下：
 - ① 找零的纸币金额有：1元、5元、10元、20元、50元；
 - ② 找零后的钱根据纸币数量统计，要求钱币的纸张总数最少；



支付金额: 100元
商品价格: 57
共需找零: 43元

20元——2张
2元——1张
1元——1张

找零纸币共: 4张



JS内建对象：数学对象

□ 数学对象提供基本的数学函数和常数Math

——数学对象是JS中的固有对象（不需实例化）

□ 数学对象常见的属性：

属性名称	说明	语法示例
E	自然对数的底。 E 属性约等于 2.718	alert(Math.E);
PI	圆周率。 圆周长与直径的比值，约等于 3.1415926	alert(Math.PI);

JS内建对象：数学对象

```
var num1 = 20;  
var num2 = -10.5;  
var num3 = 3.1415926
```

□ 数学对象常见的方法：

方法名	说明	语法示例
abs	返回数字的绝对值	Math.abs(num2)
floor	返回小于等于其数值参数的最大整数	Math.floor(num3)
ceil	最大整数取整	Math.ceil(num3)
round	将小数点后部分按照四舍五入计算结果，返回与给出的数值表达式最接近的整数	Math.round(num2)
max	返回给出的多个数值表达式中较大者	Math.max(num1, num2)
min	返回给出的多个数值表达式中较小的值	Math.min(num1, num2)
random	返回介于 0 和 1 之间的随机数	Math.random()
pow	返回底表达式的指定次幂	Math.pow(2, 3)
sqrt	返回数字的平方根	Math.sqrt(25)

JS内建对象：数学对象

□ 使用数学对象方法制作随机点名系统

```
<input type="button" value="开始" onclick="call.start();"/>  
<input type="button" value="停止" onclick="call.stop();"/>  
<p><span id="name"></span></p>
```

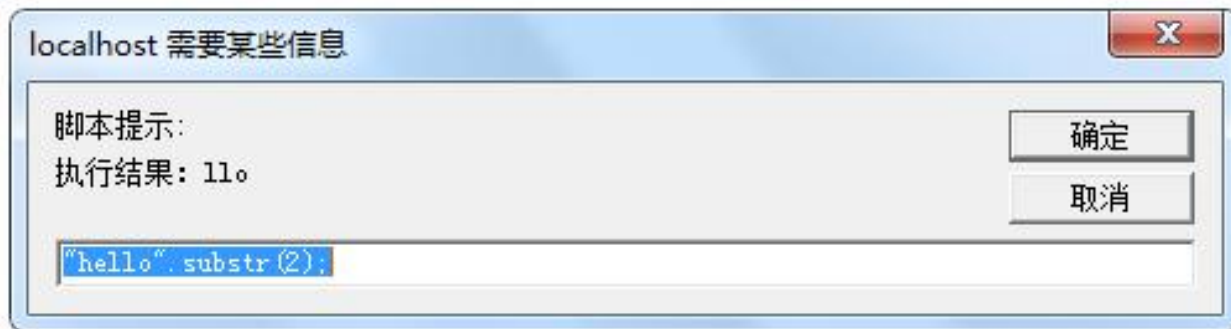
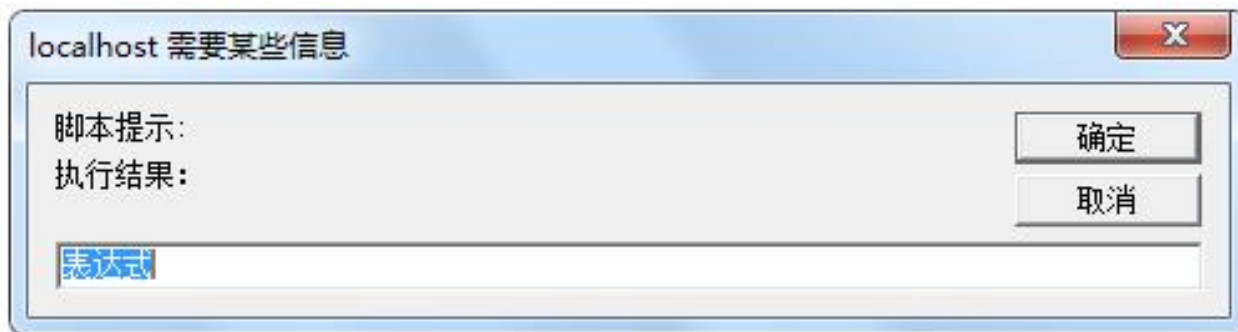
```
var call = {  
    names: ["尚飞虎", "梁昌", "刘彪", "秋雅", "王龙", "王宵琼"],  
    timer:null,  
    start:function (speed) {  
        var speed = speed || 500;  
        call.timer = setInterval(call.setName, speed);  
    },  
    setName: function () {  
        var key = Math.floor(Math.random()*call.names.length);  
        document.getElementById("name").innerHTML = call.names[key];  
    },  
    stop:function () {  
        clearInterval(call.timer);  
    }  
}
```

开始

结束

秋雅

□ JS书签的应用 - JS代码测试小助手



让学习成为一种习惯!

□ JavaScript内建的常见对象

1. 字符串对象
2. 日期对象
3. 数组对象
4. 数学对象
- 5. BOM对象**
- 6. DOM对象**
- 7. 事件处理**



BOM对象

□ BOM对象：

- BOM是browser object model的简称，是js将【浏览器】抽象出的一个模型，用以客户端实现一些渲染效果或者访问客户端浏览器的相关信息，这个模型主要包括

Window 对象：包含浏览器窗口的信息

Navigator 对象：获取浏览器特性（版本、厂商、内核、产品名等）

Screen对象：获取屏幕信息

History对象：获取浏览器历史记录

Location对象：获得当前浏览器登陆信息

他们都属于顶层window对象里的属性

BOM之location对象

□ 使用for...in语句查看**location**对象信息

```
for (var property in location) {  
    document.write(property + "::" + location[property] + "<br />");  
}
```

□ 常见属性和方法如下：

- **href**属性可读可写：当设置值时可以实现页面的跳转
location.href = "http://localhost";
- **replace()**方法：使当前页面加载指定页面，会替换掉浏览器历史记录
location.replace("http://localhost");
- **reload()**方法：重新载入当前页面
location.reload();

BOM对象之location对象

- 定义一个方法 获取地址栏所有参数信息 组装成数组或者对象形式 类似于PHP中的\$_GET。

```
function getArgs(param) {  
    $_GET = new Array();           //不使用var声明 则为全局变量  
    var param = param || location.search;  
    //提取有效的参数值 去除?  
    var query = param.substr(1);  
    //打散成数组  
    var pairs = query.split("&");  
    //循环获取相应的值  
    for (var i = 0; i < pairs.length; i++) {  
        var pos = pairs[i].indexOf("=");  
        var name = pairs[i].substr(0, pos);  
        var value = pairs[i].substr(pos + 1);  
        $_GET[name] = $value;  
    }  
}
```


BOM对象之history对象

- history对象用来访问用户历史记录，出于安全和隐私的角度 现代浏览器不允许js和其他脚本过多的访问用户浏览浏览记录，只支持有限范围内的页面跳转

```
for (var property in history) {  
    document.write(property + "::" + history[property] + "<br />");  
}
```

- go()方法：跳转到指定页面
- back()方法：后退
- forward()方法：前进

BOM对象之screen对象

□ screen对象抽象出屏幕的特性：

```
for (var property in screen) {  
    document.write(property + "::" + screen[property] + "<br />");  
}
```

属性	值（单位px）	含义
availWidth	1366	屏幕的有效宽度
availHeight	728	屏幕的有效高度
width	1366	屏幕的实际宽度
height	768	屏幕的实际高度

屏幕的有效宽度或高度是指去除任务栏后的宽度或高度，这部分通常是浏览器内容所不能达到的。

BOM对象之navigator对象

□ navigator对象提供浏览器相关信息

```
for (var property in navigator) {  
    document.write(property + "::" + navigator[property] + "<br />");  
}
```

□ 基本属性：

- appCodeName 代码提供方
- appName 软件名
- appVersion 版本号
- platform 运行平台

```
if (navigator.appName == "Microsoft Internet Explorer") {  
    alert("IE浏览器");  
} else if (navigator.appName == "Netscape") {  
    alert("基于网景的火狐或其他浏览器");  
}
```

□ 可依据appName简单判断浏览器类型

BOM对象之window对象

□ window对象包含当前窗口信息，常用方法：

方法名称	说明	代码示例
close	关闭窗口	w.close()
open	打开新窗口	var w = window.open(http://localhost[,,,]);
moveTo	移动窗口至绝对位置	w.moveTo(100,100);
moveBy	相对自身移动位置	w.moveBy(100, 100);
resizeTo	重置窗口大小	w.resizeTo(50, 50);
resizeBy	以原大小进行缩放	w.resizeBy(50, 50);
onresize	改变大小时触发的事件	w.onresize() = function () {}
onError	页面发生错误时触发的事件	w.onError = function (msg, file, line, col) {}
print	打印窗口内容	window.print()

制作窗口弹跳效果

JS DOM对象

- Document对象介绍
- DOM模型和节点访问
- 脚本化DOM样式
- DOM动画

JS document对象

- JS将客户端浏览器窗口抽象出的模型中包括一个document文档对象，可以用来**脚本化**HTML文档和文档内容。

```
for (var property in document) {  
    document.write(property,"::",document[property],"<br />");  
}
```

- HTML文件属性和操作

- HTML文档内容访问

遗留DOM (0级DOM对象)

W3C 标准DOM

- 事件属性

DOM document对象

□ HTML文件属性

属性	示例	说明
parentWindow	[object Window]	所属窗口对象
fileCreatedDate	03/06/2016	文件创建时间
fileModifiedDate	03/06/2016	文件修改时间
fileSize	3291	文件大小
fileUpdatedDate	03/05/2016	文件更新时间
mimeType	HTML 文档	文件类型
nameProp	这是标题	名称属性
protocol	HyperText Transfer Protocol	传输协议

JS document对象

□ HTML文件操作：

- 使用**document.write()** 方法向文档中写入内容时，要求文档必须是打开的状态下；如果文档已经关闭 则默认会重新打开一个新的文档，新文档会覆盖之前文档内容。显式的操作文档可以使用如下方法：

document.open(); //打开一个新文档

document.close(); //关闭当前打开的文档

```
function dw() {  
    document.write("section One");  
    //document.close();  
    document.write("section Two");  
}
```

JS DOM对象

- JS将HTML文档**内容**，即HTML标签和文本（**节点**）抽象出的模型，称为文档对象模型（DOM），该模型分为遗留的DOM和W3C DOM。

注册用户: [返回上一页](#)

昵称:

密码:

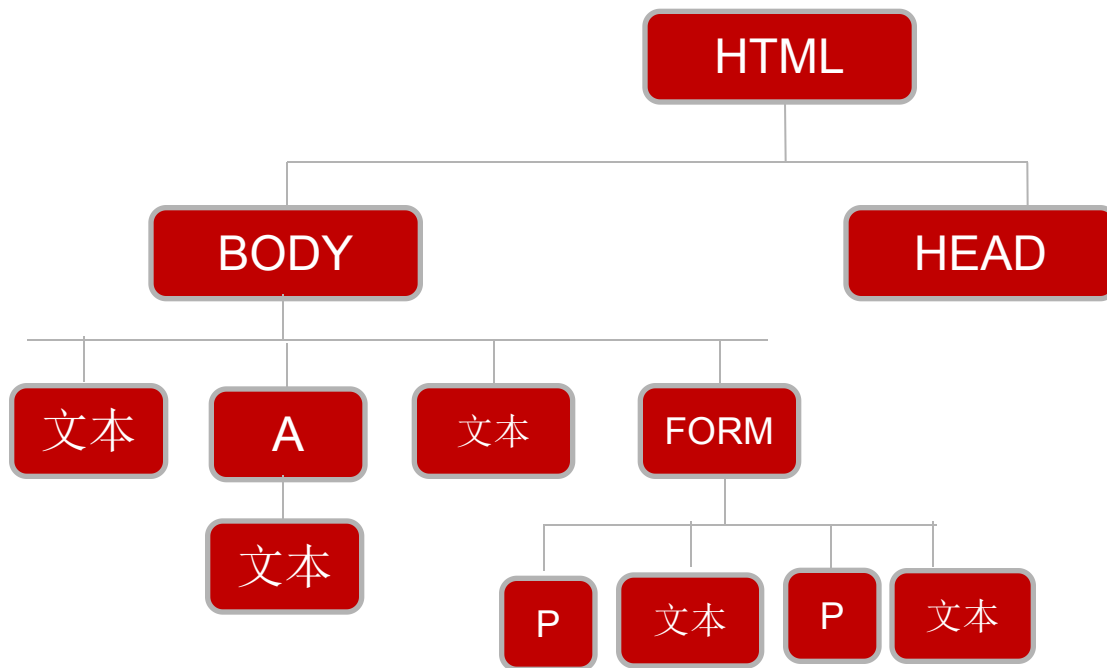
性别: ☐ 男、☐ 女

爱好: ☐ 读书、☐ 音乐、☐ 体育

城市:

个人介绍:

[已有账号? 登录](#)



标签也称为元素，在DOM中称为节点；每个节点，都是一个对象，有属性和方法

让学习成为一种习惯!

DOM节点访问：遗留DOM

- 遗留DOM是早期浏览器为访问HTML抽象出的模型，并被W3C所接纳作为0级别的DOM，它提供几种方式可用来访问HTML节点。

属性名	示例	说明
all	document.all	访问所有节点，返回节点对象数组
body	document.body	访问body节点
forms	document.forms	访问所有表单元素节点
links	document.links	访问所有链接元素节点
anchors	document.anchors	访问所有锚点
images	document.images	访问所有图片

在此处除了**body**，其余都是返回节点对象集合数组

JS 0级DOM

- ❑ document.forms包含所有页面的表单元素，可通过下标按照在文档中出现的先后顺序访问具体的元素，如果该表单是经过name命名的 则也可通过名称访问；
 - document.forms[0] 获取第一个表单
 - document.forms["registerForm"] 获取name是registerForm的表单
- ❑ 经过命名的表单可看成是document对象的属性，即 document.registerForm，同理经过命名的表单元素控件也可看成是表单的属性，则
 - document.regiterForm.nickname可访问name="nickname"的输入控件
 - 如果该控件为一组 则返回数组document.registerForm.sex包含两个元素

```
var dom = document.registerForm.nickname;
document.write(dom,"<br/>");           //[object HTMLInputElement]
for (var i = 0; i < document.registerForm.sex.length; i++) {
    document.write(document.registerForm.sex[i], "<br />");
}
```

DOM对象属性

- DOM集合了每个**元素节点的标准属性**作为自身的属性，可直接访问；并支持使用**getAttribute()**和**setAttribute()**单独的接口访问非标准属性

- 查看标准属性

```
var dom = document.registerForm.nickname;  
document.write("type属性：", dom.type, "<br />");  
document.write("name属性：", dom.name, "<br />");  
document.write("value属性：", dom.value, "<br />");
```

- 访问非标准属性

```
var dom = document.registerForm.nickname);  
dom.setAttribute("uid", 10086);  
dom.getAttribute("uid");
```

DOM对象属性

□ 使用innerHTML获取节点内容

```
var dom = document.links[0];  
document.write(dom.innerHTML);//或  
document.write(dom.firstChild.data);
```

□ 使用innerText获取节点的文本（限IE）。

```
alert(document.registerForm.innerHTML);  
alert(document.registerForm.innerText);
```

□ 注册表单的验证：

```
document.registerForm.onSubmit= function () {  
    var nickname = document.registerForm.nickname.value;  
    var password = document.registerForm.password.value;  
    var sex = "";  
    for (var i = 0; i < this.sex.length; i++) {  
        if (this.sex[i].checked) sex = this.sex[i].value;  
    }  
    var fav = new Array();  
    for (var i = 0; this["fav[]"].length; i++) {  
        if (this["fav[]"][i].checked) fav.push(this["fav[]"][i].value);  
    }  
    //.....  
    var flag = true;  
    var message = "";  
    if (nickname.mactch(/^\s*$/)) {  
        message += "昵称未填写或为空\n";  
        flag = false;  
    }  
}
```

JS 遍历DOM节点

- 除了访问特殊的节点对象外，JS还可以通过节点的关联关系查找节点。

属性	实例	说明
previousSibling	dom.previousSibling	前一个兄弟节点
nextSibling	dom.nextSibling	后一个兄弟节点
parentNode	dom.parentNode	父节点
firstChild	dom.firstChild	长子
lastChild	dom.lastChild	老幺

- 节点类型，通过dom.nodeType属性查看，几种类型如下：

类型	数值	说明
Node.ELEMENT_NODE	1	元素节点
Node.TEXT_NODE	3	文本节点
Node.COMMET_NODE	8	注释

JS 遍历DOM节点

- 遍历一个元素节点的所有子节点，并输出文本内容：

```
function listNode(n) {  
    for (var x = n.firstChild; x != null; x = x.nextSibling) {  
        if (x.nodeType == 1) {  
            listNode(x);  
        } else if (x.nodeType == 3) {  
            document.write(x.data || null, "<br />");  
        }  
    }  
}
```

DOM 节点修改

- 除了访问节点以外，DOM还提供了方法对文档节点的创建和修改，这无疑对操作文档提供了最有利的支撑。

方法名	示例	说明
createElement	document.createElement("hr")	创建一个hr节点
insertBefore	parentNode.insertBefore(n1, n2)	往parentNode的子节点n1之前插入n2节点
appendChild	parentNode.appendChild(n);	往parentNode的子节点最后追加n节点
removeChild	paretNode.removeChild(n);	删除parentNode中的子节点n;

```
var hr = document.createElement("hr");
document.body.insertBefore(hr, document.links[0]);
```

DOM 节点修改

- JS对象为引用类型，同样的节点对象也是如此

```
var n = hr;  
document.body.insertBefore(n, document.body.firstChild);
```

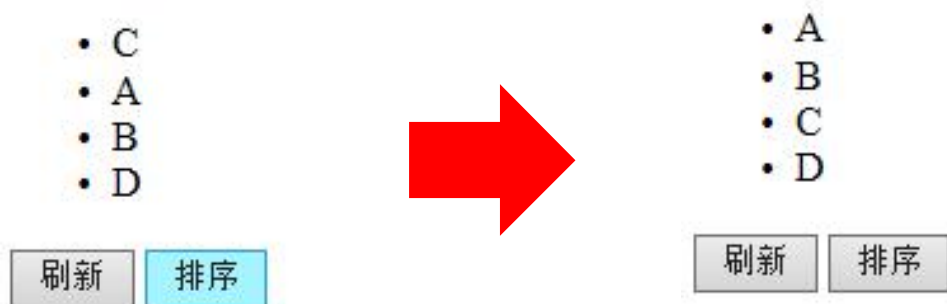
- 使用`cloneNode()`克隆节点对象

```
var n = hr.cloneNode();  
document.body.insertBefore(n, document.body.firstChild);
```

DOM节点修改

□ JS DOM节点操作 - 页面中元素排序：

- 点击“排序”，列表中内容按“从小到大”**肾虚**排列
- 点击“刷新”，列表恢复初始化状态



W3C DOM

❑ 除了遗留DOM外，W3C还提供了专门接口方便访问页面中的所有元素

- `document.getElementById ()` 通过ID标记访问元素

```
var nickname = document.getElementById("nickname");
```

获取ID为nickname的节点

- `document.getElementsByName ()` 通过name属性访问元素

```
var sex = document.getElementsByName("sex");
```

获取页面中所有name值为sex的节点 返回对象集合的数组

- `document.getElementsByTagName ()` 通过标签名访问元素

```
var dom= document.getElementsByTagName("a");
```

获取页面中所有a标签节点 返回对象集合的数组

W3C DOM

- 使用W3C标准DOM接口 验证注册表单，要求
 - 省市支持二级菜单联动
 - 文本域使用编辑器

JS DOM

□ 脚本化DOM节点样式

- 通过访问节点对象的style属性设置：

```
dom.style.border = "1px solid #ccc"; //相当于  
dom.style.borderWidth = 1 + "px";  
dom.style.borderColor = "#ccc";  
dom.style.borderStyle = "solid";
```

驼峰法

- 给节点绑定类名，通过css类选择器设置样式

```
.A {border:1px solid #ccc;}  
dom.setAttribute("class", "A");
```

DOM

□ JS表单登陆验证，要求：

- 在相应的输入框后面输出错误提示信息
- 当输入正确信息后 消除之前的错误提示

昵称: 账号不能为空

密码: 密码长度应为6-15位

登陆

JS 动画

- JS的动画效果都是基于定时调节元素的位置或者尺寸。
 - 将一个DIV容器3S内水平移动300PX，则平移速度为10PX/100MS，可设计如下运动方式：
 - 执行时间间隔为 100MS
 - 每帧元素移动10PX
 - 共30帧画面

```
var animate = {  
    perFrame : 100,  
    numFrames : 30,  
    timer : null,  
    frame : 0,  
    start : function () {  
        animate.timer = setInterval(animate.nextFrame,animate.perFrame);  
    },  
    nextFrame : function () {  
        if (animate.frame > animate.numFrames) {  
            clearInterval(animate.timer);  
        }  
        var e = document.getElementById("container");  
        e.style.left = 10*animate.frame + "px";  
        animate.frame++;  
    },  
};
```

封装摇号插件

□ 制作一个能随机多个数字并显示加载进度的插件，要求：

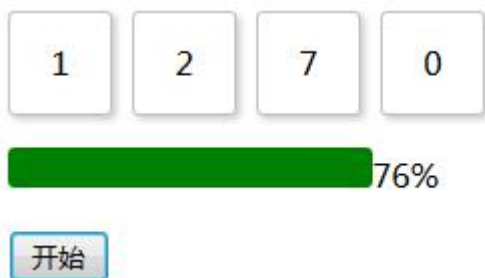
- 使用面向对象封装，支持用户配置

count：数字个数

width：容器的大小

skin：容器的皮肤

result：预设结果



```
<div id="result"></div>
<p id="progress"></p>
<input type="button" value="开始"
onclick="Lottery.start({
    count:4,
    width:50,
    skin:'lightGray',
    result : {0 : [1,2,3,4]}
});" />
```

JS 事件处理

- ❑ 事件是事先给元素绑定的处理程序，当满足语义时执行该段处理程序，以达到交互的效果。事件的基本要素为：**元素**和**场景**以及**处理**。
- ❑ 常见注册事件的三种方式：
 - 标签内设置事件属性
 - 通过匿名函数给标签属性赋值
 - <script>标签声明（IE支持）
- ❑ 事件的简单划分：
 - 设备相关事件：键盘事件、鼠标事件
 - 语义相关事件：onchange、onsubmit

JS 注册事件

- 通过属性赋值注册事件：设置事件属性值为一段可执行的JS代码。

```
<input type="button" value=" 点 我 " onclick="alert('好的，马上出台');" />
```

- 使用document.write(typeof dom.onclick)查看该属性值的数据类型为function，则可以将要执行代码片段使用函数替代。

```
<input type="button" value=" 点 我 " onclick="f();" />
```

- 使用匿名函数给属性赋值

```
<input type="button" id="btn" value=" 点 我 " />  
document.getElementById("btn").onclick = function () {}
```

JS 事件处理

□ 使用toString()方法查看事件的函数体：

```
<input type="button" id='btn' value=" 点 我 " onclick="alert('好的，马上出台');" />
<script>
document.write(document.getElementById("btn").onclick.toString());
</script>
```

- 定义事件 就是定义函数和方法体。事件名为函数名，执行代码为函数体，事件的触发就是调用该函数。可以显式的调用函数来模拟事件的执行。

```
document.getElementById("btn").onclick();
```

- 既然事件是执行函数，那么就有返回值

JS 事件处理

- 事件执行时传入的event对象存储了事件的相关信息

```
<input type="button" id='btn' value=" 点 我 " onclick="f(event);" />
<script>
function f(event) {
    for (var p in event) {
        document.write(p, "::" , event[p], "<br />");
    }
}
</script>
```

JS 注册事件

- IE提供了另外的方法，可以通过script标签声明一个事件

```
<input type="button" id='btn' value=" 点 我 " onclick="alert('好的，马上出台');" />  
<script for="btn" event="onclick">  
alert("好的马上出台");  
</script>
```


□ JS常见事件列表

事件	支持标签	语义
onload	body	加载文档时执行的动作
onclick	多数标签	点击鼠标时 触发的动作
onmouseover	多数标签	鼠标经过时触发的动作
onmouseout	多数标签	鼠标移走时触发的动作
onfocus	表单控件	元素获取焦点时触发的动作
onblur	表单控件	元素失去焦点时触发的动作
onchange	select	表单值发生改变时的动作
onkeydown	输入控件等	按下键盘瞬间执行的动作
onkeyup	输入控件等	按键释放瞬间执行的动作
onsubmit	form	表单提交时执行的动作

让学习成为一种习惯!

JS onclick事件

- 使用鼠标点击事件，生成一定数量的输入框，并计算重复值

请输入文本框的个数：

生成

第1个文本框：

第2个文本框：

第3个文本框：

第4个文本框：

第5个文本框：

比较结果

第2个元素和第4个元素值重复：2

第3个元素和第5个元素值重复：3

计算重复值

JS onchange事件

□ 使用onchange事件制作省市二级联动

城市：-请选择- ▼

- 请选择-
- 安徽省-
- 浙江省-
- 江苏省-
- 江西省-



城市：-请选择- ▼ -请选择- ▼



城市：-安徽省- ▼

- 合肥
- 芜湖
- 马鞍山
- 伦敦
- 巴黎

JS onkeyup事件

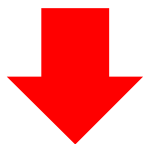
- 使用键盘事件验证用户输入，如填写手机号时只允许输入数字 否则不显示本次输入值

```
<script>
function f(obj) {
    //获取用户输入值
    var phone = obj.value;
    //验证用户输入值 判断其是否能匹配一段数字 如果不能则表示输入有错
    if (!phone.match(/^d*$/)) { //输入不合法
        obj.value = obj.defaultValue;
    } else { //输入合法 将该值存储起来
        obj.defaultValue = obj.value;
    }
}
</script>
```

手机号 :

JS onmouseover事件

- 使用鼠标经过事件制作评分工具：
 - 初始化为10个未点亮的小星星，显示0分。
 - 当鼠标移至小星星上时 高亮选中部分 并显示相应的分数



AJAX 概述

- AJAX是指异步（ Asynchronous ）的 JavaScript 和（ And ）XML。
 - 它提供了JS和服务端进行通信并交换数据的方式，使得很多原本需要PHP脚本才能执行的功能使用JS即可实现。这样页面无需跳转，达到无刷新、不阻塞用户的效果，增强了操作的体验度。
- AJAX是一种称谓，并不具体指代某个具体技术，它的核心是JS中的XMLHttpRequest对象，依赖于Javascript和XML。该对象提供了JS与PHP通信的方法。
- AJAX技术自1998左右开始出现和应用（同期应用的另一项互联网革命性技术是CSS2）。允许客户端脚本发送HTTP请求（XMLHTTP）的第一个组件由Outlook Web Access小组写成。该组件原属于微软Exchange Server，尔后成为了IE4.0的一部分。
- 2005年伴随很多事件的影响使得AJAX迅速传播，尤其以google地图最为知名。如今ajax已被广泛运用以提高系统性能，优化用户界面。

认识 XMLHttpRequest

- ❑ 所有现代浏览器（IE7+、Firefox、Chrome、Safari 以及 Opera）均内建 XMLHttpRequest 对象。
- ❑ 创建并查看XMLHttpRequest对象信息

```
var request= new XMLHttpRequest();  
for (var p in request) {  
    document.write(p + "::" + request[p], "<br />");  
}
```

XMLHttpRequest请求过程

□ 使用数字标记整个请求过程的不同阶段

客户端JS
未初始化
状态值为0

与服务器建立连接，状态值为1



服务器接受请求，状态值为2

PHP脚本
处理中：3
返回值：4

XMLHttpRequest请求状态值

- request对象的readyState属性用来表示请求过程的状态，有以下几种情况：

状态	值	说明
request.UNSEND	0	请求未发送
request.OPEND	1	已和服务器建立连接
request.HEADERS_RECEIVED	2	服务器已接受
request.LOADING	3	服务器处理中
request.DONE	4	完成并返回结果

- 查看当前请求状态：alert(request.readyState);

建立AJAX请求

□ 向服务器发送请求：request.open(方式, 地址, [是否异步=true]);

- get方式：将要发送到服务器的数据以GET方式随URL地址传输

```
request.open("get", "user.php?uid=111&rand="+Math.random());  
request.send(null);
```

- post方式：模拟表单的post数据提交方式 你需要发送头信息

```
request.open("post","user.php",true);  
request.setRequestHeader("Content-type","application/x-www-form-urlencoded");  
request.send("uid=111");
```

- GET和POST相比，GET传参轻便但数据容量有限，另POST更安全。

建立AJAX请求

- 如果注册了**onloadstart()**事件，则会在建立请求时会调用该事件

```
<div id="show"></div>
```

注册事件必须在建立请求前

```
<script>
request.onloadstart = function () {
    var span = document.createElement("span");
    span.innerHTML = "请求已建立...";
    document.getElementById("show").appendChild(span);
}
</script>
```

请求过程中处理

□ 注册请求处理中的事件处理程序：**onprogress**

```
<script>
request.onprogress= function () {
    var span = document.createElement("span");
    span.innerHTML = "请求处理中...";
    document.getElementById("show").appendChild(span);
}
</script>
```

获取返回结果

□ 使用onloadend()事件执行返回结果的处理

```
request.onloadend = function () {  
    var span = document.createElement("span");  
    span.innerHTML = "请求已完成！";  
    document.getElementById("show").appendChild(span);  
    //获取返回结果  
    var result = this.responseText;  
    document.write(result);  
}
```

request对象的**responseText**属性用来获取返回的字符串结果

PHP 服务器端处理

□ 依据传输方式获取JS数据，并返回一个执行结果

- 如js以get方式发送的请求 则 `$v = $_GET[param]`,
- 如js以post方式发送的请求 则 `$v = $_POST[param]`,
- 当返回一个结果是一个数组或复合类型数据时，不能直接通过HTTP协议传输，必须转化成字符串形式 并且保证JS能正常解析和还原结构，这需要一个能跨脚本支持的数据格式即JSON或XML，如果使用json数据格式 则

php编码数组或对象为json字符串的函数为：`$str = json_encode($arr);`

php反编码json字符串的函数为：`$result = json_decode($str);`

JS解析json字符串

- 在js中使用responseText属性可获取从PHP返回的经过json编码的字符串，经过PHP json编码后的字符串样式如下：

原值	编码后	说明
"hello"	"hello"	原样输出字符串
array("name" => "jack", "sex" => "female")	'{"name":"jack","sex":"female"}'	类似对象的字符串
array(0 =>array("name" => "jack", "sex" => "female"))	'[{"name":"jack","sex":"female"}]'	类似对象集合的字符串

在js中使用eval()函数执行该字符串转换成相应的数据格式
var result = eval(request.responseText);

捕获请求过程中状态

- 通过注册 **onreadystatechange** 事件捕获请求过程中状态改变时的处理程序。

```
request.onreadystatechange = function () {  
    if (this.readyState == this.DONE && this.status == 200) { //结束  
        var result = eval("(" + this.responseText + ")");  
    } else { //执行等待期间的处理  
    }  
}
```

- request的status属性存储了http请求的状态码，常见的状态如下：

状态码	说明
200	请求成功
403	禁止访问
404	未找到页面

JS AJAX

- 使用AJAX无刷新的分页显示

JS库：JQUERY简介

□ jQuery - javascript优秀的框架

- 什么是框架？框架是一种在软件开发中可以重复使用的设计构建（程序的骨架）。
- jquery官网：<http://jquery.com>
- 口号：write less, do more
- 特点：简洁、快速
- 参考资料：《锋利的jQuery》第二版



引入jquery

□ 使用jquery之前 需引入jquery文件

➤ 第一步：下载jQuery.js

☐ PRODUCTION (19KB, Minified and Gzipped)

压缩后版本 (适用于实际项目)

☐ DEVELOPMENT (120KB, Uncompressed Code)

未压缩版本 (适用于开发测试)

➤ 第二步：在HTML文件中引入该文件

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

开始jquery

□ 开始jquery - 使用jquery实现表格奇偶行换色

```
$(document).ready(function () {  
    //当文档加载完后 在此处加载代码  
    alert("hello jquery");  
})
```

1	2	3
1	2	3
1	2	3

```
var tbl = document.getElementById("tbl");  
var tbody = tbl.getElementsByTagName("tbody")[0];  
var trs = tbody.getElementsByTagName("tr");  
for (var i = 0; i < trs.length; i++) {  
    if (i % 2 == 0) {  
        trs[i].style.backgroundColor = "yellow";  
    }  
}  
  
$("table tbody tr:odd").css("backgroundColor", "red");
```

JQUERY

□ 学习提纲

- jquery DOM获取
- jquery 链式操作
- jquery DOM操作
- jquery 事件驱动
- jquery 常见动画
- jquery AJAX封装
- jquery 常用插件

jquery DOM获取

□ JQ获取dom节点 - **\$()**方法:

- JQ获取的是经过包装后的jquery对象 不能使用js DOM的方法去操作,同样js DOM 也不能调用jquery中的方法。

```
<p>昵称 : <input type="text" name="nickname" id="nickname" value="小强" /> </p>
<p>爱好 : <input type="checkbox" name="fav[]" value="读书" />读书、
          <input type="checkbox" name="fav[]" value="音乐" />音乐、
          <input type="checkbox" name="fav[]" value="体育" />体育
</p>
```

```
$(document).ready(function () {
    alert($("#nickname").value);
    alert($("#nickname").val());
    var nickname = document.getElementById("nickname");
    alert(nickname.val());
});
```

JQ DOM

□ JQ对象和JS对象的转换

- 使用[]可以将一个jquery对象转化成一个JS的对象 或者写成 favs.get(0)

```
var favs = $("input[name='fav[]']");  
var f1 = favs[0]; //var f1 = favs.get(0);  
alert(f1.value);
```

- 使用\$()将js dom转化成jquery dom

```
var jsdom= document.getElementById("nickname");  
var jqdom = $(jsdom);  
alert(jqdom.val());
```

JQ DOM

- ❑ **\$()**方法冲突：如果页面中已经存在自定义的**\$()**方法 则会引起与jq的冲突。

```
function $(id) {  
    return document.getElementById(id);  
}  
alert($("#nickname").val());           //jq $()失效
```

- ❑ 使用noConflict()交回\$控制权，使用**jQuery**替代\$

```
jQuery.noConflict();  
alert(jQuery("#nickname").val());  
var $j = jQuery.noConflict();  
alert($j("#nickname").val());
```


jquery选择器

- jquery 获取DOM的方式和CSS选择器的使用一脉相承，一般也称为Jquery选择器，几种常见的**基本选择器**：

选择器	示例	说明
ID选择器	<code>\$("#id")</code>	获取id="id"的节点
类选择器	<code>\$(".cname")</code>	获取class="cname"的节点集合
元素选择器	<code>\$("tag")</code>	获取标签名为tag的节点集合
后代选择器	<code>\$("div span")</code>	获取div里面的所有span
子元素选择器	<code>\$("div > span")</code>	获取div的子节点中的span
相邻兄弟选择器	<code>\$("p+b")</code>	获取和p相邻的b标签
并列选择器	<code>\$("b, u, i")</code>	获取b和u和i标签节点

□ jquery DOM示例

```
<div class="parent">
  <div class="son" id="d1">firstChild</div>
  <div class="son" id="d2">lastChild</div>
  <span>son</span>
  <p><span>sunson</span></p>
</div>
```

document.writeln(\$("#d1").attr("id"));	//attr()方法用来查看jq dom的属性
document.writeln(typeof \$(".son"));	//类选择器返回的是对象集合的数组
document.writeln(\$(".son").length);	
document.writeln(\$(".son").attr("id"));	//自动获取第一个元素的结果返回
document.writeln(\$("#div").length);	
document.writeln(\$("#div span").html());	//html()方法用来查看jqdom的内容
document.writeln(\$("#div > span").html());	
document.writeln(\$("#div+div").html());	
document.writeln(\$("#p,span").length);	

□ 过滤选择器

选择器	示例	说明
:first	<code>\$("ul li:first")</code>	选中ul中第一个li
:last	<code>\$("ul li:last")</code>	选中ul中最后一个li
:eq	<code>\$("ul li:eq(1)")</code>	定位ul中第二个li
:not	<code>\$("ul li:not('.no')")</code>	过滤出类名不为no的li
:contains	<code>\$("ul li:contains('txt')")</code>	过滤出内容中包含txt的li
:empty	<code>\$("ul li:empty")</code>	找出空的li
:checked	<code>\$("input:checked")</code>	找到选中的input标签
[]	<code>\$("input[type='text']")</code>	所有多选框

`$("ul li").first().next().html();` //链式操作

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
  <li>list item 4</li>
  <li></li>
  <li class="no">list item 5</li>
</ul>
<form>
<p>昵称：<input type="text" name="nickname" id="nickname" value="小强" /></p>
<p>爱好：<input type="checkbox" name="fav[]" value="读书" />读书、
  <input type="checkbox" name="fav[]" value="音乐" />音乐、
  <input type="checkbox" name="fav[]" value="体育" />体育
</p>
</form>
```

jquery dom属性

- 由于jquery选择器获取的是jquery dom对象，因此需要专门的**方法** 获取相应的dom属性值

方法名	示例	说明
attr()	<code>\$("#nickname").attr("type");</code>	获取id为nickname的控件type属性
removeAttr()	<code>\$("img").removeAttr("width")</code>	删除img对象的width属性
val()	<code>\$("#nickname").val()</code>	获取id为nickname的控件值
addClass()	<code>\$("img").addClass("c")</code>	给img对象绑定类c
removeClass()	<code>\$("img").remove("c")</code>	移除img对象的类c
html()	<code>\$("span").html();</code>	获取span标签的内容

```


<p>
    <input type="text" value="小强" name="uname" id="uname"/>
    <input type="checkbox" value="check1"/> check1
    <input type="checkbox" value="check2"/> check2
</p>

<ul>
    <li>list item</li>
</ul>

<input type="button" value="使用attr的方法访问属性" />
<input type="button" value="使用removeAttr的方法删除属性" />
<input type="button" value="使用val的方法获取value属性值" />
<input type="button" value="使用addClass的方法绑定类名" />
<input type="button" value="使用removeClass的方法删除类名" />
<input type="button" value="使用html的访问内容" />
```

jquery dom属性

□ 使用attr()方法 获取jquery dom的属性值

- `$dom.attr("属性名")` 获取属性值
- `$dom.attr("属性名", "属性值")` 设置属性值
- `$dom.attr({"属性1": "值1", "属性2": "值2"});` 同时设置多个属性
- `$dom.attr("属性名", function (索引, 原值) {})`

```
$("#img").attr("width", function (idx, wid) {  
    return wid*2;  
})
```

□ 使用removeAttr()删除属性：

```
$("#img").removeAttr("width");
```

jquery dom属性

- 对于表单控件常用的value属性值 可以通过`val()`方法 单独访问
 - `$dom.val("属性值")` 设置属性值
 - `$dom.val(["值1", "值2"]);` 同时给多个对象设置值

```
alert($("#uname").val());  
$("#uname").val("Jack");  
$("input:checkbox").val(["check1", "check2"]);
```


jquery dom 属性

□ 使用addClass()给元素绑定类名

- \$dom.addClass("类名");
- \$dom.addClass('类名',function (idx, cname) {});

```
$("#uname").addClass(function(idx, cname) {  
    $("#uname").removeClass(cname);  
    return "newName";  
});
```

□ 使用removeClass()删除元素属性

```
$("#uname").removeClass("cname");
```

jquery dom 内容

- html()方法可以用来获取节点的内容，text()方法获取文本
 - \$dom.html() 获取内容
 - \$dom.text() 获取文本
 - \$dom.html("值") 重置内容
 - \$dom.html(function (idx, html) {})

```
//获取对象的内容
```

```
alert($("#ul").html());
```

```
//获取对象的文本
```

```
alert($("#ul").text().replace(/\s+/g, " "));
```

```
//设置对象内容
```

```
$("#ul").html(function (index, html) {  
    return html + html;  
});
```

jquery dom 链式操作

- 使用链式操作过滤选择器：对于过滤选择器除遵从统一的CSS选择器规则外，也支持链式操作，常见链式操作中的方法

方法	示例	说明
first()	<code>\$("ul li").first()</code>	获取ul 后代li中的第一个
last()	<code>\$("ul li").last()</code>	获取ul 后代li中的最后一个
eq()	<code>\$("ul li").eq(1)</code>	定位ul 后代li中第二个
not()	<code>\$("ul li").not(\$("#first"))</code>	获取所有id不为first的li
contents()	<code>\$("div").contents();</code>	获取div的所有子节点
find()	<code>\$("div").find("span");</code>	查找div后代中的span
has()	<code>\$("ul").has(\$(".menu"))</code>	选中元素是否有某个后代元素
hasClass()	<code>\$("ul li").hasClass("menu")</code>	判断选中元素中是否有某个类名

```
<ul>
```

```
    <li>Hello</li>
```

```
    <li class="menu">Hello</li>
```

```
    <li>Hello</li>
```

```
</ul>
```

```
<div>
```

```
    <b>b</b> <u>u</u> <span>span</span>
```

```
</div>
```

```
<input type="button" value="使用eq定位文档元素" />
```

```
<input type="button" value="使用first定位选中元素的第一个" />
```

```
<input type="button" value="使用last定位选中元素的最后一个" />
```

```
<input type="button" value="使用hasClass判断选中元素是否有某个类名" />
```

```
<input type="button" value="使用HAS判断选中元素是否某个后代元素" />
```

```
<input type="button" value="使用not过滤掉指定的选择器" />
```

```
<input type="button" value="使用find查找已选中元素里面的指定的选择器" />
```

```
<input type="button" value="使用contents获取所有子节点" />
```

```
<style>
```

```
.menu {
```

```
    list-style: circle inside none;
```

```
    font: italic bold 15px Monaco;
```

```
    color: blue;
```

```
}
```

```
</style>
```

jquery dom 操作

□ jquery dom节点操作：

方法	示例	说明
append()	<code>\$("ul").append(\$("hr"));</code>	在ul最后面追加hr
appendTo()	<code>\$("ul").appendTo(\$("hr"))</code>	将ul追加到hr最后面
prepend()	<code>\$("ul").prepend(\$("hr"))</code>	在ul最前面追加hr
prependTo()	<code>\$("ul").prependTo(\$("hr"))</code>	将ul追加到hr最前面
after()	<code>\$("ul li").last().after(" ")</code>	在最后一个li后面加入内容
before()	<code>\$("ul li").first().before(" ");</code>	在第一个li前面加入内容
remove()	<code>\$("ul li").first().remove();</code>	删除第一个li
clone()	<code>\$("hr").clone()</code>	克隆一个hr节点

```
<ul>
    <li>list item1</li>
    <li class="menu">list item2</li>
    <li>list item3</li>
</ul>
<hr />
```

```
<style>
.submenu { color:red; }
.menu {
    list-style:circle inside none;
    font:italic bold 15px Monaco;
    color:blue;
}
</style>
```

```
<input type="button" value="使用APPEND追加文档元素" />
<input type="button" value="使用appendTo追加文档元素" />
<input type="button" value="使用PREPEND前置文档元素" />
<input type="button" value="使用prependTo前置文档元素" />
<input type="button" value="使用after在当前元素后面加入内容" />
<input type="button" value="使用before在当前元素前面加入内容" />
<input type="button" value="使用remove删除当前选中的文档元素" />
```

jquery 样式

□ jquery提供了css方法用来实现jquery dom样式化

- `$dom.css("样式")` 读取一个的样式
- `$dom.css("样式", "值")` 设置一个样式
- `$dom.css({样式1:"值1", 样式2:"值2"})` 同时设置多个样式
- `$dom.css("样式", function () {})` 函数参数设置值

```
$("ul li").css("width", function (idx, wid) {  
    return parseInt(wid) * ( idx + 1 );  
})
```

√ 当jquery选中的为对象集合数组时，默认会为每个对象添加样式

对于宽和高的样式属性 可以使用width()和height()方法操作

jquery 事件类型

□ jquery中的事件通过不同的**方法**实现，如：

方法	示例	说明
click()	<code>\$dom.click(function() {})</code>	给\$dom绑定点击事件
mouseover()	<code>\$dom.mouseover(function () {})</code>	鼠标经过事件
mouseout()	<code>\$dom.mouseout(function () {})</code>	鼠标移出事件
foucs()	<code>\$dom.focus(function () {})</code>	元素获取焦点事件
blur()	<code>\$dom.blur(function () {})</code>	元素失去焦点事件
submit()	<code>\$dom.submit(function () {});</code>	表单提交事件

jquery 事件处理

□ 给列表中的li绑定事件

```
$("#ul li").click(function () {  
    alert($(this).html());  
});
```

- 当jquery选中的为对象集合数组时，默认会为每个对象绑定事件
- 在匿名函数体内 可以使用`$(this)`表示当前对象

□ 循环给每个对象绑定事件 - `each()`方法

```
$("#ul li").each(function (i) { //i表示循环的下标  
    alert($(this).html());  
});
```

jquery 动画效果

□ jquery内置了一些简单实用的动画 方便随时使用，常见动画如下：

方法	示例	说明
show()	<code>\$("div").show()</code>	显示div
hide()	<code>\$("div").hide()</code>	隐藏div
toggle()	<code>\$("div").toggle()</code>	自动切换
slideDown()	<code>\$("div").slideDown()</code>	向下打开
slideUp()	<code>\$("div").slideUp()</code>	向上闭合
slideToggle()	<code>\$("div").slideToggle()</code>	自动开合
fadeIn()	<code>\$("div").fadeIn();</code>	渐进
fadeOut()	<code>\$("div").fadeOut()</code>	淡出
animate()	<code>\$("div").animate()</code>	自定义动画

```
<div id="d1" style="width:200px;height:200px;background:red;display: none;"></div>
<div id="d2" style="width:100px;height:100px;background:blue;">
    some text
</div>

<p>
    <input type="button" value="show" />
    <input type="button" value="hide" />
    <input type="button" value="toggle" />
    <input type="button" value="slideDown" />
    <input type="button" value="slideUp" />
    <input type="button" value="slideToggle" />
    <input type="button" value="fadeIn" />
    <input type="button" value="fadeOut" />
    <input type="button" value="animate" />
</p>
```

jquery 动画 显示控制

□ \$dom.show()/\$dom.hide()/\$dom.toggle()

- \$dom.show([速度][, 回调函数])

速度既支持数值（毫秒）也可设置固定类型值 "normal", "fast", "slow" 等
回调函数在动画执行完调用

```
$("#d1").show();
```

```
$("#d1").show(5000);
```

```
$("#d1").show(5000, function () {})
```

jquery 动画 自动开合

- `$dom.slideDown()/$dom.slideUp()/$dom.slideToggle()`
 - `$dom.slideDown([速度][, 回调函数])`

速度既支持数值（毫秒）也可设置固定类型值 "normal", "fast", "slow" 等
回调函数在动画执行完调用

```
$("#d1").slideDown();  
$("#d1").slideDown(5000);  
$("#d1").slideDown(5000, function () {})
```

jquery 动画淡入淡出

□ \$dom.fadeIn()/\$dom.fadeOut()

- \$dom.fadeIn([速度][, 回调函数])

速度既支持数值（毫秒）也可设置固定类型值 "normal", "fast", "slow" 等
回调函数在动画执行完调用

```
$("#d1").fadeIn(300);  
$("#d1").fadeOut(5000);  
$("#d1").fadeOut(5000, function () {})
```

jquery css动画

□ 使用animate()方法自定义CSS过渡动画效果

```
$("#d2").animate({  
    left:500,  
    top:300,  
    fontSize:30px;  
}, 3000);           //加载前需设置位置属性 #d2{position:absolute;}
```

□ 封装js的AJAX

```
function ajax(param) {  
    //创建ajax请求  
    var request = new XMLHttpRequest();  
    //发送请求  
    if (param.type == "get") {  
        request.open("get", param.url + "?rand=" + Math.random() + "&" + param.data);  
        request.send(null);  
    } else if (param.type == "post") {  
        request.open("post", param.url );  
        request.setRequestHeader("Content-type","application/x-www-form-urlencoded");  
        request.send(param.data);  
    }  
    //执行成功  
    request.onloadend = function() {  
        if (param.dataType == "json") {  
            param.success(eval("(" + this.responseText + ")");  
        } else if (param.dataType == "text") {  
            param.success(this.responseText);  
        }  
    }  
}
```