JS 简介及作用

1、JavaScript 是属于网络的脚本语言！
2、JavaScript 被数百万计的网页用来改进设计、验证表单、检测浏览器、创建 cookies，以及更多的应用。
3、JavaScript 是一种轻量级的编程语言。
4、JavaScript 是可插入 HTML 页面的编程代码。
5、JavaScript 插入 HTML 页面后，可由所有的现代浏览器执行。

JS 实现

JS 脚本语言必须包含在标签<script></script>之间，其中可以写入脚本语句，可以进行定义方法，例如：

```
<script>
    document.write("页面内容");
    function test(){
        document.getElementById("id").value="aaa"; //代码块
    }
    test();//相当于方法被调用
</script>
```

注意：在页面加载时，在方法外面的 JS 脚本会被直接执行，方法则不会，方法会在被调用的时候执行。

JS 输出

调用的方法：
1、 document.getElmentById("div1").innerHTML="改变层的内容";
此方法经常被使用，不仅仅可以改变标签元素的文本内容，同样可以改变标签元素的结构，例如：
document.getElmentById("div1").innerHTML="<table style='width:100px;height=100px'><tr><td>第一个单元内容</td><td>第二个单元个内容</td></tr></table>";
此方法向层中添加了新的元素标签，增加了一个表格。写的时候注意双引号中的单引号。
2、 document.write("页面内容");
此方法在初始页面直接执行的时候，会在页面中加入新的内容，但是在页面加载后，再执行此代码会将当前的页面进行覆盖，慎用！

JS 注释

单行注释：// 块注释 /* */

JS 变量

定义变量使用关键字:var 只能使用 var 来进行定义，不管定义的变量是什么数据类型，例如：
var x=1; var a = “abc”; x 为数字类型，a 为字符类型
如果在定义变量时没有使用关键字 var 那么 JS 则会给变量自动识别，并且将变量声明为全局变量不管是在方法内部还是在方法外部。
在定义变量时可以指定数据类型，
例如：var x = new Number();

JS 数据类型

JS 的数据类型有：字符串、数字、布尔、数组、对象、Null、Undefined。JS 拥有动态类型。这意味着相同的变量可用作不同的类型。这是由定义变量的关键字 var 导致。
例如：var x = 1; x=”abc”;
定义变量直接声明类型：

```
var a = new String();
var b = new Number();
var c = new Boolean();
var d = new Array();
var e = new Object();
```

JS 对象

JavaScript 中的所有事物都是对象：字符串、数字、数组、日期，等等。
在 JavaScript 中，对象是拥有属性和方法的数据。
举例 1：

```
var obj = new Object();
obj.name = “张三”;
obj.action = function(){
    JS 代码块
}
```

举例 2：

```
var obj = {
    name : “张三”,
    action : function(){
        JS 代码块
    }
}
```

JS 函数

函数是由事件驱动的或者当它被调用时执行的可重复使用的代码块。
举例：

```
function test(){
    JS 代码块
}
```

JS 运算符

1、 算数运算符

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（保留整数）	x=y%2	x=1
++	累加	x=++y	x=6
--	递减	x=--y	x=4

注意：整型跟字符型相加时相当于拼接字符串，
举例：var x=2;var y="a";var z=x+y; z="2a"

2、 赋值运算符

运算符	例子	等价于	结果
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

3、 比较运算符

运算符	描述	例子
==	等于	x==8 为 false
===	全等（值和类型）	x===5 为 true；x==="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

4、 逻辑运算符

运算符	描述	例子
&&	and	(x < 10 && y > 1) 为 true
||	or	(x==5 || y==5) 为 false
!	not	!(x==y) 为 true

注意：或运算符||的另外一中用法，var x = a || b;如果 a 为真则 x=a，否则 x=b;如果都为 true，还是 x=a
举例：var x = “abc” || 123; x=”abc”

5、 条件运算符 (三目运算符)

var x= 条件?value1:value2；如果条件为 true，则 x 等于 value1 否则 x 等于 value2

举例： var a = 5>3? “aaa”:”bbb”;

JS 流控制

1、 If….else

```
举例：if(true){
        JS 代码块
      }else{
        JS 代码块
      }
```

注意：如果代码块只有一句话组成，可以不适用{}，例如:if(true) alert(“aaa”); else alert(“bbb”);

2、 switch()…case

```
举例：switch(n){
        case 1:
          JS 代码块
          break;
        case 2:
          JS 代码块
          break;
        .....
        default:
          JS 代码块
      }
```

3、 for

```
举例：for(var i=0;i<12;i++){
        JS 代码块
      }
```

4、 for in

```
举例：var persion = {name:”zhangshan”,gender:”male”,age:22}
    for( x in persion){
        persion[x] //或者使用   persion.x
    }
```

注意：x 代表的不是索引值，而是属性值。

5、 while(条件){JS 代码块}

举例： var x = 5;

```
while(i<20){
    x += 10;
    i++;
}
```

6、 do{JS 代码块}while(条件)

举例：var x = 5;var i=0;

```
do{
   x+=10;
   i++;
}while(i<20)
```

JavaScript 表单验证

1、 判断是否为空，举例：

```
var username = document.getElementById(“username”).value;
if(username==null || username==””){
    alert(“请输入用户名”);
    return;
}
```

2、 格式校验，举例 Email 校验，可以使用正则表达式

```
var   regex = /^\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*$/;
var   emailValue = document.getElementById(“e_mail”).value;
var   flag = regex.text(emailValue);//返回 Boolean 类型的值，符合就为 true，否则就为 false
if(flag){
    alert(“符合邮箱格式”);
}else{
    alert(“邮箱格式不正确”);
}
```

3、 长度判断

```
var password = document.getElementById(“password”).value;
if(password.length>8){
    alert(“密码长度超过 8 位，请重新输入！”);
    return;
}
```

4、 逻辑校验

举例：身份证号码校验

```
//正则表达式   /^\d{15}|\d{18}$/;
var idCard = document.getElementById("idCard").value;
if(idCard==""){
    alert("身份证号为必填项！");
    return;
```

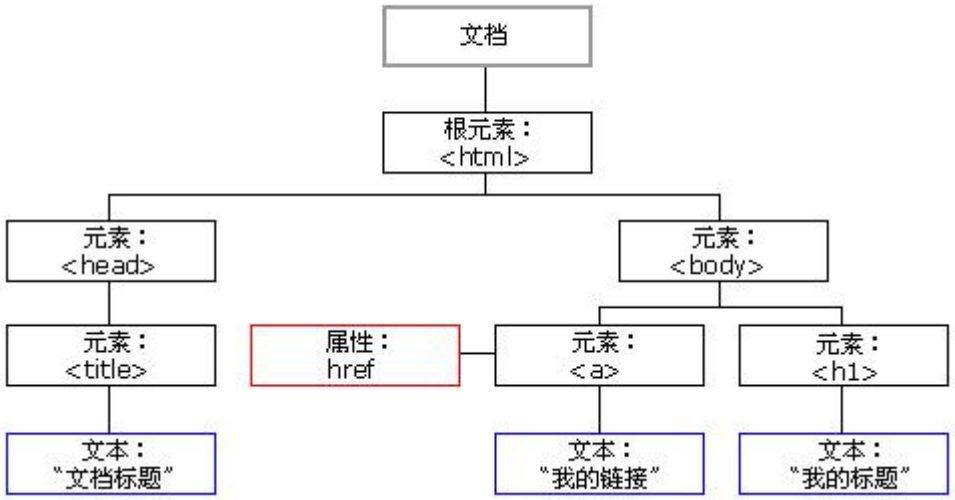
```
}else if(idCard.length==15){
//测试数据   421022881006452
        var year =    idCard.substring(6,8);
        var month = idCard.substring(8,10);
        var day = idCard.substring(10,12);
        var date = new Date(year,parseFloat(month)-1,parseFloat(day));
        if(date.getYear()!=parseInt(year) || date.getMonth()!=parseInt(month)-1
                || date.getDate()!=parseFloat(day)){
                alert("身份证号码出生日期不正确");
                return false;
       }else{
                alert("身份证号码出生日期正确");
                return true;
       }
}else if(idCard.length==18){
        //测试数据
        var year =    idCard18.substring(6,10);
        var month = idCard18.substring(10,12);
        var day = idCard18.substring(12,14);
        var date = new Date(year,parseInt(month)-1,parseInt(day));
        if(date.getFullYear()!=parseFloat(year)    || date.getMonth()!=parseFloat(month)-1
       || date.getDate()!=parseFloat(day)){
                alert("身份证号码出生日期不正确！ ");
                return false;
        }
        var wi = [ 7, 9, 10, 5, 8, 4, 2, 1, 6, 3, 7, 9, 10, 5, 8, 4, 2, 1 ];//加权因子
        var valideCode = [ 1, 0, 10, 9, 8, 7, 6, 5, 4, 3, 2 ];//验证码
        var idArray = idCard.replace(/\s/g,"").split("");
        var sum = 0;
        if (idArray[17].toLowerCase() == 'x') {
                idArray[17] = 10;//如果是 10 替换为数字
        }
        for ( var i = 0; i < 17; i++) {
                sum += wi[i] * idArray[i];//加权求和
        }
        var index = sum % 11;//得到验证码所位置
        if (idArray[17] == valideCode[index]) {
                alert("身份证号码为真");
        } else {
                alert("身份证号码为假");
        }
        }else{
                alert("身份证号码位数不正确，请重新输入！ ");
                return;
```

}

注意：目前此方法缺少地区、格式校验

JSDOM 讲解

当网页被加载时，浏览器会创建页面的文档对象模型（Document Object Model）。
HTML DOM 模型被构造为对象的树。
了解 DOM 树



一、查找 html 元素对象两种常用方式：

1、 通过 id 查找 HTML 元素
举例：<input type=”text” id=”username” name=”username” value=”” />
Var username = document.getElementById(“username”).value;

2、 通过 name 查找 HTML 元素
举例：<input type=”text” id=”username” name=”username” value=”” />
document.getElementsByName("username")；
注意：返回的是数组对象，可根据下标索引去找到具体的元素
var username = document.getElementsByName("username")[0].value;
常用在对复选框对象的获取上，举例：

```
<input type="checkbox" name="avocation" value="01">足球
<input type="checkbox" name="avocation" value="02">篮球
<input type="checkbox" name="avocation" value="03">排球
var valueArray = var username = document.getElementsByName("avocation ");
var valString = “”;
for(var i=0;i<valueArray.length;i++){
    valString = valString+valueArray[i].value+”;”;
}
```

结果：valString = “01;02;03;”

二、改变 HTML 元素样式

1、通过 style 属性改变，举例：

```
<p id="p2">文本信息</p>
document.getElementById("p2").style.color="blue";
```

2、通过样式表类进行改变，举例：

```
<style type=”text/css”>
    .class1{
        color:#fff;
    }
document.getElementById("p2").className=”class1”;
```

注意：样式表颜色的表示方式，1 常见的代表颜色的英文单词，如 red　2、#66CCFF：六位十六进制的数字表示，根据三种基础颜色 rgb{red,green,blue}来显示不同色彩值，前两位表示数 66 表示红色基调，中间两位 CC 表示绿色基调，FF 表示蓝色基调，如果两位数相同还可以进行缩写如上例颜色还可以写为：#6CF；

3、 样式 display 与 visibility 的区别（额外补充）

display 这个属性用于定义建立布局时元素生成的显示框类型, visibility 属性规定元素是否可见。

举例： 层的内容

document.getElementById("div1").style.display=”none”;表示该元素不在页面中显示，同时不占据空间。类似于页面中没有这个元素

document.getElementById("div1").style. visibility=”hidden”；表示该元素在页面中不显示，但是会占据空间。类似于页面中有个元素，只是看不到。

三、Event 对象

在事件发生的时候，会产生一个临时对象 event。这个对象存储着事件行为的相关信息。如果没有此事件的处理函数，则 event 对象立刻消失；否则在事件处理程序完成后，event 对象才消失。

1、onblur 失去焦点事件，支持该事件的 HTML 标签：<a>,<button>, ,<form>,<iframe>, <img>, <input> <p><select> <td>, <textarea>

常用的标签为 input，举例：

```
<input type=”text” onblur=”check()” id=”username” name=”username”/>
function check(){
    var username = document.getElementById(“username”).value;
    if(username==””){
        alert(“请输入用户名！”);
        document.getElementById(“username”).blur();//让文本框获取焦点
    }
}
```

2、onchange 值域发生改变触发事件，支持该事件的 HTML 标签：<input type="text">, <select>, <textarea>

举例 1：<input type="text" id="fname" onchange="upperCase(this.id)" />

```
function upperCase(x){
    var fname = document.getElementById(x).value;
    document.getElementById(x).value=fname.toUpperCase();
}
```

注意：this 表示当前标签对象，this.id 就表示当前对象的 ID 属性值，toUpperCase()String 对象的方法，将字符串对象变为大写的格式。

举例 2：<form action="somepage.asp" name="theForm">

```
        <select name="province" onchange="getCity()">
            <option value="0">请选择所在省份</option>
            <option value="直辖市">直辖市</option>
            <option value="江苏省">江苏省</option>
            <option value="福建省">福建省</option>
            <option value="广东省">广东省</option>
            <option value="甘肃省">甘肃省</option>
        </select>
        <select name="city">
            <option value="0">请选择所在城市</option>
        </select>
</form>
```

额外补充 Form 获取控件对象的方式

注意：获取 form 表单中的 text、select 等的信息，可以通过 form 表单获取，有两种方式：

1、 通过表单名称:document.forms[“表单名”].控件名;

document.forms[“formName”].username.value;

2、 如果页面中含有多个 form 表单可以通过索引值来获取，通常情况下一个页面中只会有一个表单 form

document.forms[0].username.value;

3、 浏览器保留一张表单中所有控件元素的对象，如果对象唯一，则返回对象，如果对象不唯一，则返回对象的列表 Array

Var proArray = document.forms["formName"].elements["province"];

说明：获得属性 name 为 province 值的控件数组对象，举例：

```
username1:<input type="text" value="" name="username" value=”aaa” size=20/><br>
username2:<input type="text" value="" name="username" value=”bbb” size=20/><br>
var us1 = document.forms[0].elements["username"][0].value;
var us2 = document.forms[0].elements["username"][1].value;
    var city=[
        ["北京","天津","上海","重庆"],          //直辖市
        ["南京","苏州","南通","常州"],          //江苏省
        ["福州","福安","龙岩","南平"],          //福建省
        ["广州","潮阳","潮州","澄海"],          //广东省
        ["兰州","白银","定西","敦煌"]           //甘肃省
    ];//定义二维数组对象 city

    function getCity(){
        //获得省份和城市下拉列表框的引用
```

```
        var sltProvince =document.forms["theForm"].elements["province"];
        var sltCity = document.forms["theForm"].elements["city"];
        //得到对应于省份的城市列表数组
        var provinceCity=city[sltProvince.selectedIndex-1];
        //将城市下拉列表框清空，仅留第一个提示选项
        sltCity.length=1;
        //将相应省市的城市填充到城市选择框中
        for(var i=0;i<provinceCity.length;i++){
            //创建新的 Option 对象并将其添加到城市下拉列表框中
            sltCity[i+1]=new Option(provinceCity[i],provinceCity[i]);
        }
    }
```

3、onclick 鼠标单击事件，支持该事件的 HTML 标签：<a>,<button>,<input>

举例：<input type="button" value="提交" onclick="check()"/>

```
function check(){
    JS 代码块
}
```

4、 onfocus 获取焦点事件，支持该事件的 HTML 标签：<input><textarea>

举例：<input type="text" id="email" onfocus="setDefalut()"/>

```
function setDefalut (){
    document.getElementById("email").value="example@163.com";
}
```

5、 onkeydown 键盘按下事件，支持该事件的 HTML 标签：<input><body>

举例：body 整个页面监听事件

```
<span id="sp1"></span>
document.onkeydown=showValue;
function  showValue(e){
    document.getElementById("sp1").innerHTML=e.keyCode;
}
```

说明：参数 e 是当前监听的 Event 对象，e.keyCode 代表的键盘的值

6、 onkeyup 键盘按下后松开键盘触发的事件，使用方式同上 5 项。

7、 onload 事件，页面加载事件，支持的事件的 HTML 标签：<body>，经常用来对页面初始化时需要操作的事情

```
<body onload="init()"></body>
//常用来进行初始化参数，绑定事件等操作
var a,b,c;
function init(){
    a,b,c="hello";
    document.onkeydown=changeValue;
}
function changeValue(){
    JS 代码块
}
```

8、 还有常用事件：onmouseover 鼠标移到元素上触发的事件，对应的 onmouseout 鼠标从元素上移开触

发的事件，用法参考以上几项，不在进行举例。

四、DOM 节点

在 HTML DOM 中，所有事物都是节点。DOM 是被视为节点树的 HTML。整个文档是一个文档节点，每个 HTML 元素是元素节点，HTML 元素内的文本是文本节点，每个 HTML 属性是属性节点

节点父、子和同胞：节点树中的节点彼此拥有层级关系。

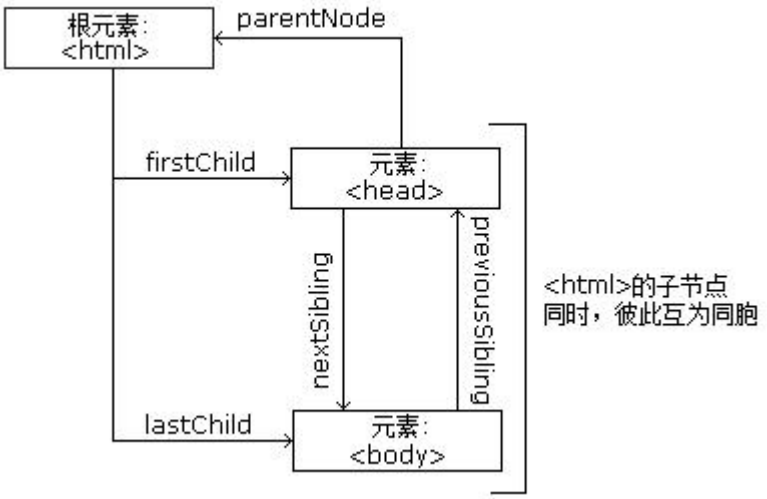
父（parent）、子（child）和同胞（sibling）等术语用于描述这些关系。父节点拥有子节点。同级的子节点被称为同胞（兄弟或姐妹）。

在节点树中，顶端节点被称为根（root）

每个节点都有父节点、除了根（它没有父节点）

一个节点可拥有任意数量的子

同胞是拥有相同父节点的节点




```
<html>
  <head>
    <title>DOM 教程</title>
  </head>
  <body>
    <h1>DOM 第一课</h1>
    <p>Hello world!</p>
  </body>
</html>
```

从上面的 HTML 中：

<html> 节点没有父节点；它是根节点;<head> 和 <body> 的父节点是 <html> 节点;文本节点 "Hello world!" 的父节点是 <p> 节点

并且：

<html> 节点拥有两个子节点：<head> 和 <body>; <head> 节点拥有一个子节点：<title> 节点; <title> 节点也拥有一个子节点：文本节点 "DOM 教程";<h1> 和 <p> 节点是同胞节点，同时也是 <body> 的子节点

并且：

<head> 元素是 <html> 元素的首个子节点;<body> 元素是 <html> 元素的最后一个子节点;<h1> 元素是 <body> 元素的首个子节点;<p> 元素是 <body> 元素的最后一个子节点

1、 var htTag = document.documentElement;//获取根元素

var htmlContent = htTag.innerHTML;

五、HTML DOM 对象

1、方法和属性

方法	描述
getElementById()	返回带有指定 ID 的元素。
getElementsByTagName()	返回包含带有指定标签名称的所有元素的节点列表（集合/节点数组）。
getElementsByClassName()	返回包含带有指定类名的所有元素的节点列表。
appendChild()	把新的子节点添加到指定节点。
removeChild()	删除子节点。
replaceChild()	替换子节点。
insertBefore()	在指定的子节点前面插入新的子节点。
createAttribute()	创建属性节点。
createElement()	创建元素节点。
createTextNode()	创建文本节点。
getAttribute()	返回指定的属性值。
setAttribute()	把指定属性设置或修改为指定的值。

举例：

```
<span id="sp1" name="sp1">苹果</span>
<span id="sp2" name="sp1">橘子</span>
<span id="sp3" name="sp1">香蕉</span>
</div>
```

var sp1 = document.getElementById(“sp1”);

var sp1Value = sp1.innerHTML;或者 sp1.textContent;

注意：innerHTML 获取的内容如果有标签，则显示出标签内容，textContent 则不会但会显示出空格换行：例如下面

父节点: var divNode = sp1.parentNode;

var divNodeName = divNode.nodeName; //divNodeName=” div”;

var dValue1 = divNode.innerHTML;//值如下图

```
“
<span id=“sp1” name=“sp1”>span标签的内容111</span><br>
<span id=“sp2” name=“sp1”>span标签的内容222</span><br>
<span id=“sp3” name=“sp1”>span标签的内容333</span><br>
<br>
<br>
”
```

Var dvalue2 = divNode. textContent;//值如下图所示

所以：获取文本值建议使用 textContent 属性，同时还要 trim()一下，String 对象的方法，去掉两边的空格。

建议：不要使用 Node 节点的方式去对 dom 进行读取，因为使用 node 方式，其中代码中的换行、空格都会当做文本节点去处理，所以建议使用元素方式 Element

举例：divNode. childNodes[0].textContent; //为空

divNode.children[0].textContent;//苹果，childern 返回的是元素数组对象

上一个元素：var sp2 = document.getElementById(“sp2”);

同级：var sp1 = sp2.previousElementSibling;

Var sp3 = sp2.nextElementSibling;

上级：var div = sp2.parentElement;

下级：var spArray = div.children;

2、Node 节点获取属性

举例：var sp1 = document.getElementById(“sp1”);

var sp1Array = sp1.attributes;

//访问方式两种，类似访问对象的属性 1、[“属性名”]2、对象.属性。注意：获取的是属性对象而不是值

var name = sp1Array.name;// sp1Array[“name”]

var nameValue = name.value;

等同于：var nameValue = sp1Array.name.value; // nameValue 的值是” sp1”

3、创建新的元素

```
举例：<div id="d1">
        <p id="p1">This is a paragraph.</p>
        <p id="p2">This is another paragraph.</p>
      </div>
var para=document.createElement("p");
var node=document.createTextNode("This is new.");//创建文本元素
para.appendChild(node);
var element=document.getElementById("d1");
element.appendChild(para);
```

4、删除已有的 HTML 元素

举例：
```
<div id="div1">
    <p id="p1">This is a paragraph.</p>
    <p id="p2">This is another paragraph.</p>
</div>
var parent=document.getElementById("div1");
var child=document.getElementById("p1");
parent.removeChild(child);
```

JS 对象

一、String 对象

1、charAt()

返回指定索引位置处的字符。
```
var str = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var s;
s = str.charAt(3);    //s=”D”
```

2、concat(String)

返回字符串值，该值包含了两个或更多个提供的字符串的连接。
```
var str1 = "hello "
var str2 = " world";
var s = str1.concat(str2);//s=”hello    world”
```

3、indexOf(String)

返回 **String** 对象内第一次出现子字符串的字符位置
```
var str1 = "hi,nice to meet you."
var s = str1.indexOf(“nice”); //s 的值是 3
```

4、lastIndexOf(string)

返回 **String** 对象中子字符串最后出现的位置
```
var str = “ab11ab22ab33ab”;
```

var s = str. lastIndexOf(“ab”); //s 的值是 11

5、replace(reg,String)

返回根据正则表达式进行文字替换后的字符串的复制.

Var str = “ab test ab test ab test ab”;

Var reg = /ab/;

Var s1 = str.replace(reg,”A”);//s1 的值是 “A test ab test ab test ab”

Reg = /ab/g;

Var s2 = str.replace(reg,”A”);//s1 的值是 “A test A test A test A”

正则表达式后面追加 g 表示全部替换。

6、split(string)

将一个字符串分割为子字符串，然后将结果作为字符串数组返回

Var str = “abcdef”;

Var strArray = str.split(“”); //strArray 的值是[“a”,“b”,”c”,”d”,”e”,”f”];

7、substr(index,length)

返回一个从指定位置开始的指定长度的子字符串.

Var str = “abcdefjhiklmn”;

Var s = str.substr(3,5);//s 的值是”defjh”

8、substring(startIndex,endIndex)

返回位于 **String** 对象中指定位置的子字符串，开始索引位置，结束索引位置结束位置必须大于开始位置，否则只返回当前开始位置的一个字符

Var str = “0123456789”;

Var numStr = str.substring(3,6);//numStr 的值是“345”

二、Array 对象

1、concat(Array)

返回一个新数组，这个新数组是由两个或更多数组组合而成的

var a, b, c, d;

a = new Array(1,2,3);

b = "JScript";

c = new Array(42, "VBScript);
d = a.concat(b, c);//d 的值是数组 [1, 2, 3, "JScript", 42, "VBScript"]

2、pop()

移除数组中的最后一个元素并返回该元素，如果该数组为空，那么将返回 **undefined**。
var a = new Array(1,2,3,4);
var b = a.pop();//b 的值是 4

3、push(object)

将新元素添加到一个数组中，并返回数组的新长度值。
var a = new Array(1,2,3,4);
var b = a.push(5);//b 的值是：5

4、shift()

移除数组中的第一个元素并返回该元素
var a = new Array(1,2,3,4);
var b = a.shift();//b 的值是：1

5、unshift(object)

将指定的元素插入数组开始位置并返回该数组
var a = new Array(1,2,3,4);
a = a.unshift(5);//a 的值是[5,1,2,3,4]

三、Date 对象

Var date = new Date();

1、getDate()

返回 Date 对象中用本地时间表示的一个月中的日期值。
date.getDate() //例如当前日期为 2013-12-14 返回 值为 14

2、getDay()

返回 Date 对象中用本地时间表示的一周中的日期值。
date.getDay()//返回值为 0(周日),1(周一),2(周二),3(周三),4(周四),5(周五),6(周六)

3、getFullYear()

返回 Date 对象中用本地时间表示的年份值
date.getFullYear();//例如当前日期为 2013-12-14 返回 值为 2013

4、getHours()

返回 Date 对象中用本地时间表示的小时值
date.getHours();//例如当前时间为 18:23 返回值为 18

5、getMilliseconds()

返回 Date 对象中用本地时间表示的毫秒值
date. getMilliseconds();

6、getMinutes()

返回 Date 对象中用本地时间表示的分钟值
date. getMinutes();//例如当前时间为 18:23 返回值为 23

7、getMonth()

返回 Date 对象中用本地时间表示的月份值
date.getMonth();//例如当前日期为 2013-12-14 返回 值为 11
1 月份返回 0；2 月份返回 1；3 月份返回 2.。。。。。12 月份返回 11

8、getSeconds()

返回 Date 对象中用本地时间表示的秒钟值
date. getSeconds();//例如当前时间为 18:23:33 秒，返回 33

对应的还有相应的 set 方法

四、Math 对象

1、abs()

返回数字的绝对值
举例：Math.abs(-1)//返回 1

2、floor()

返回小于等于其数值参数的最大整数

举例：Math.floor(12.34);//返回 12

3、random()

返回介于 0 和 1 之间的伪随机数
举例：Math. random()

4、ceil(Float)

返回大于等于其数字参数的最小整数
举例：Math.ceil(12.44);//13

JS Window 对象

1、window.open(URL)

举例：window.open(“http://www.baidu.com”);
//打开一个新的窗口

2、window. showModalDialog()

文件 1、parent.htm

```
<script>
        var obj = new Object();
        obj.name="51js";
        window.showModalDialog("modal.htm",obj,"dialogWidth=200px;dialogHeight=100px");
</script>
```

文件 2、modal.htm

```
<script>
        var obj = window.dialogArguments
        alert("您传递的参数为：" + obj.name)
</script>
```

有返回值的 showModalDialog()
文件 1、parent.htm

```
<script>
```

```
        str =window.showModalDialog("modal.htm",,"dialogWidth=200px;dialogHeight=100px");
        alert(str);
</script>
```

文件 2、modal.htm

```
<script>
    Function closeWindow(){
        window.returnValue="http://homepage.yesky.com";
        window.close();
    }

</script>
<html><body><button onclick=”closeWindow()”>关闭</button></body></html>
```

3、Location

location.hostname 返回 web 主机的域名
location.pathname 返回当前页面的路径和文件名
location.port 返回 web 主机的端口 （80 或 443）
location.protocol 返回所使用的 web 协议（http:// 或 https://）

```
<html>

<head>

<script>

function newDoc()

  {
  window.location.assign("http://www.w3school.com.cn")
  }
</script>
</head>
<body>


<input type="button" value="加载新文档" onclick="newDoc()">


</body>
</html>
```

4、History

history.back() 方法加载历史列表中的前一个 URL

history forward() 方法加载历史列表中的下一个 URL

5、Navigator

属性	描述	IE	F	O
appCodeName	返回浏览器的代码名。	4	1	9
appMinorVersion	返回浏览器的次级版本。	4	No	No
appName	返回浏览器的名称。	4	1	9
appVersion	返回浏览器的平台和版本信息。	4	1	9
browserLanguage	返回当前浏览器的语言。	4	No	9
cookieEnabled	返回指明浏览器中是否启用 cookie 的布尔值。	4	1	9
cpuClass	返回浏览器系统的 CPU 等级。	4	No	No
onLine	返回指明系统是否处于脱机模式的布尔值。	4	No	No
platform	返回运行浏览器的操作系统平台。	4	1	9
systemLanguage	返回 OS 使用的默认语言。	4	No	No
userAgent	返回由客户机发送服务器的 user-agent 头部的值。	4	1	9
userLanguage	返回 OS 的自然语言设置。	4	No	9

6、cookie

方法：setCookie(name,value,option)

Name 是要设置的 cookie 的名称，value 是设置 cookie 的值，option 包括了其他选项，是一个对象作为参数。

```
Cookie.setCookie=function(name,value,option){
    Var str = name +”=”+escape(value);
    If(option){
        //如果设置了过期时间
        If(option.expireDays){
            Var date=new Date();
            Var ms = option.expireDays*24*3600*1000;
            date.setTime(date.getTime()+ms);
            str +=”;expires=”+date.toGMTString();
        }
    }
    document.cookie=str;
}
```

删除 cookie，方法 deleteCookie(name),name 指定 cookie 的名称，根据名称来删除

删除 cookie 的方法是通过 setCookie 来实现的，将 option 的 expireDays 属性指定为负数即可。

```
Cookie.deleteCookie=function(name){
    This.setCookie(name,””,{expireDays:-1})
}
```

JS 消息框

1、 alert(“警告框”);
2、 confirm(“确认框”)
举例:var flag = confirm(“此操作不可逆，是否删除记录？”);
flag 的值为 true 或者 false
3、 popmt(“标题”)；输入框

JS 数据类型转换

数据类型转换
到 Boolean 的转换 new Boolean(value)

Undefined	总是 false
Null	总是 false
Number	0 或 NaN 时为 false，其他为 true
String	字符串为空时（长度为 0）为 false，其他为 true
Object	总是 true

到数字类型的转换 new Number(value)

Undefined	NaN
Null	0
Boolean	true 1 ， false 0
String	如果可以转换为数字，则返回字符串表示的数字，否则返回 NaN

到字符串的转换 new String(value)

Undefined	“Undefined”
Null	“Null”
Boolean	“true” “false”
Number	具体的值

JS 中的面向对象

一、基础讲解

JS 的面向对象，那么什么是面向对象？
var obj = new Object(); var array = new Array();对象是属性和方法的集合。
等价于 var obj = {} var array = [];
如果我们想定义一个对象或者类那么可以使用 function 函数的方法来定义一个对象来冒充类。例如：

```
funtion Car(){
    this.price = 12.5;
```

```
    this.run = function(){
        alert("the car is start.");
    }
}
```

Var car = new Car();获取对象 Car 的实例。用 this 表示本身的属性和方法，类似于 java 中的 public 修饰符，可以让实例对象使用，在内部使用 var 定义相当于用 private 修饰，实例对象不能使用。但是可以提供公用方法访问私有变量，类似 JavaBean 的概念，JavaBean 在 JSP-Servlet 中会讲道。

举例：

```
function Car(){
    var price = 12.5;
    this.getPrice=function(){
        alert(price);//或者 return price;
    }
}
```

指定类型的对象 var array = new Array(); var num = new Number(); var date = new Date();
var obj = new Object();
无类型对象 var car = { name:"bmw",start:function(){alert("start")}}

对象间引用：举例

```
var array = new Array();
array.push("数学");array.push("英语");array.push("语文");
var student = {
    name:"张三",
    classNo:"040112",
    studyContent:array,
    step:function(){
        alert("学习进度");
    }
}
alert(student.studyContent[0]);
```

注意：被引用的对象必须先与对象之前定义，否则会无效，找不到该对象。

对象的属性跟方法的删除操作：

```
student.name=undefined;   student.step=undefined;
```

二、函数对象

可以使用 function 关键字定义一个函数，并为每个函数指定一个函数名，通过函数名来进行调用。在 javascript 解释执行的时候，函数都是被维护为一个对象，就是函数对象。

函数对象与其他用户所定义的对象有着本质的区别，这一类对象被称为内部对象，例如 Date Array String 都属于内部对象同样是函数对象。这些内置对象的构造器是由 javascript 本身所定义的。通过 new 返回一个对象，javascript 内部有一套机制来初始化返回的对象，而不是由用户来指定对象的构造方式。

在 javascript 中，函数对象对应的类型是 Function，内部定义 Date、Array、Object 都是函数对象

可以通过 typeof()方法进行验证，返回值为 function,举例：typeof(Array)
typof()的返回值 "number," "string," "boolean," "object," "function," 和 "undefined."

自己定义的对象返回 object,例如：var array = new Array(); typeof(array)返回 object
function test (){} ; var tes1 = new test(); typeof(tes1)返回的就是 object
类似于： function Date(); var date = new Date();
function Array(); var array=new Array();
function test(); var te = new test();
typeof()返回 function typof()返回 object

可以通过 new Function()来创建一个函数对象，也可以通过 function 关键字来创建一个对象。
函数 function 定义的对象返回的都是 function，例如 function test(){} typeof(test)返回 function

```
function myfunction(a,b){
    return a+b;
}
```
等价于
```
var myfunction = new Function("a","b","return a+b");
```

函数可以作为参数进行传递
举例：
```
function f1(){
    alert("ssss");
}
function f2(test){
    test();
}
f2(f1);
```

三、JS 中的 prototype

prototype 对象是实现面向对象的一个重要机制。每个函数其实也是一个对象，他们对应的类是 Function。类似于数组对应的类是 Array，字符串对应的类是 String，日期对应的类是 Date 一样。每个函数对象都具有一个子对象 prototype。prototype 表示了函数的原型，而函数也是类，prototype 就表示了一个类的成员的集合。当通过 new 来获取一个类的对象的时候，prototype 对象的成员都会成为实例化对象的成员。

1、原型法设计模式

原型法的主要思想是，现在有 A 类, 我想要创建一个类 B, 这个类是以 A 为原型的, 并且能进行扩展。我们称 B 的原型为 A。

2、javascript 的方法 function 分类

A 对象方法 B 类方法 C 原型方法 举例：

```
function Car(){
    this.shoes = 4;
    this.start = function(){       //对象方法
        alert(“the car is start.”);
    }
}
var car = new Car();
car.stop=function(){
    alert(“the car is stop”); //类的方法，相当于私有方法，只能自己调用
}

Car.proptotype.load(){//对 Car 原数据类型操作，类似于在 Car 定义时又多加了一个方法
    alert(“可乘坐四人”);
}
```

重点：car 是 Car 的实例，Car 中通过 prototyp 原型的调用获取了对原数据的操作，又多加入了一个方法，那么作为它的实现类，car 可以调用的方法变成了三个，一个是 Car 开始定义的 start()方法，一个是对象本身定义的私有的 stop()方法。第三是 Car 对象后来追加的 load 方法。

在生成另外一个实例 car2:例如 var car2 = new Car(); car2 可调用的只能是 Car 定义的 start 与 load 方法。

3、prototype 的使用

Function 类型的作用就是给函数对象本身定义一些方法和属性，借助于函数的 prototype 对象，可以修改和扩展 Function 类型的定义。例如

```
<script type="text/javascript">
        Function.prototype.method1=function(){
            alert("function");
        }
        function func1(a,b,c){
            return a+b+c;
        }

        1、func1.method1();
        2、func1.method1.method1();
    </script>
```

说明：Function 通过原型定义了一个方法 method1，所有在定义任何函数的时候，所有的函数都带有了一个默认的方法 method1，类似 1，因为 method1 是函数，所以函数又可以相当于带有默认的 method 方法。类似递归调用，可以无限调用下去。

函数的 apply call 方法和 length 属性

javascript 为函数对象定义了两个方法　apply 和 call，他们的作用都是将函数绑定到另外一个对象上去运行，两者仅在定义参数的方式上有所区别

Function.prototype.apply(thisArg,argArray);

Function.prototype.call(thisArg[,arg1[,arg2…]]);

从函数原型可以看到，第一个参数都被取名为 thisArg，也就是说所有函数内部的 this 指针都会被复制为 thisArg，这就实现了将函数作为另外一个对象的方法运行的目的。

```
<script type="text/javascript">
    function func1(){
        this.p="func1-";
        this.A=function(arg){
            alert(this.p+arg);
        }
    }
    function func2(){
        this.p="func2-";
        this.B=function(arg){
            alert(this.p+arg);
        }
    }
    var obj1 = new func1();
    var obj2 = new func2();
    obj1.A("byA");
    obj2.B("byB");
    obj1.A.apply(obj2,["byA"]);
    obj2.B.apply(obj1,["byB"]);
    obj1.A.call(obj2,"byA");
    obj2.B.call(obj1,"byB");

</script>
```

运行后可以看到 obj1 的方法 A 被绑定到 obj2 运行后，整个函数 A 的运行环境就转移到了 obj2，this 指针指向了 obj2.同样 obj2 的函数 B 也被绑定到了 obj1 对象去运行。

函数对象的 length 属性表示函数定义时所指定的参数的个数，而不是调用的时候的个数

举例：function test(arg1,arg2,arg3) test.length=3

调用时：test(1,2,3,4,5)

传递给函数的隐式参数

当进行函数调用的时候，除了指定参数外，还创建一个隐含的参数对象：arguments。

arguments 是一个类似数组但不是数组的对象。

类似数组是因为：可以使用 arguments[index]取值，拥有数组的 length 属性。

arguments 对象存储的是实际传递给函数的参数，而不局限于函数声明所定义的参数列表。

比如：

```
<script type="text/javascript">
    function func(a,b){
        for(var i = 0;i<arguments.length;i++){
            alert(arguments[i]);
        }
    }
    func(1,2,3);
</script>
```

这样做带来了很大的灵活性，即使不指定参数列表，仍然可以通过 arguments 获取参数。
arguments 对象的另外一个属性是 callee，它表示对函数对象本身的引用。
递归计算 1 到 n 的自然数之和

```
<script type="text/javascript">
    var sum = function(n){
        if(1==n)return 1;
        else return n+sum(n-1);
    }
    alert(sum(100));
</script>
```

其中函数内部包含了对 sum 自身的调用，对于 javascript 来说，函数名仅仅是一个变量名，在函数内部调用 sum 就是相当于调用一个全局变量，不能很好的体现出是调用自身，使用 callee 属性是不错的选择：

```
<script type="text/javascript">
    var sum = function(n){
        if(1==n)return 1;
        else return n+arguments.callee(n-1);
    }
    alert(sum(100));
</script>
```

JavaScript 中的 this 指针

表示当前运行的对象。也代表对自身的引用。

和其他语言不通，javascript 中的 this 指针是一个动态变量，一个方法内的 this 指针并不是始终指向定义该方法的对象，前面说的 apply 和 call 方法就是这样。

```
var obj1 = new Object();
var obj2 = new Object();
obj1.p = 1;
obj2.p = 2;
obj1.getP = function() {
    alert(this.p);
}
obj1.getP();
```

```
obj2.getP = obj1.getP;
obj2.getP();
```

分别弹出 1 和 2 ，getP 函数仅定义了一次，但内部的 this 却发生了改变。

4、类的实现

在 javascript 中，function 关键字来定义一个类。
使用 this 指针引用的变量或方法会成为类的成员。
比如：

```
<script language="JavaScript" type="text/javascript">
        function class1(){
            var s="abc";
            this.p1 =s;
            this.method1= function(){
                alert("test methos");
            }
        }
    </script>
```

这里面 p1 和 method1 都是，而 s 不是。
使用 new 创建对象的过程

1. 当解释器遇到 new 操作符时就创建一个空对象；
2. 开始运行 class1 这个函数，并将其中的 this 指针都指向这个新建立的对象；
3. 当给对象里面不存在的属性赋值的时候，解释器就会为对象创建该属性，这样函数执行的就是初始化这个过程，实现了构造函数的引用。
4. 当函数执行完后，new 操作符就返回初始化后的对象

function 的定义实际就是实现了一个构造器，这样的方式的缺点是：
将所有的初始化语句、成员定义都放到一起，代码逻辑不够清晰，不容易实现复杂的功能。
每创建一个类的示例，都要执行一次构造函数。构造函数中定义的属性和方法总被重复创建。

使用 prototype 对象定义类成员

```
<script language="JavaScript" type="text/javascript">
    function class1(){
        this.prop=1;
    }
    class1.prototype.showProp=function(){
        alert(this.prop);
    }
    var obj1=new class1();
    obj1.showProp();
</script>
```

prototype 是一个 javascript 对象，可以为 prototype 对象添加、修改、删除方法和属性，从而为一个类添加

成员定义。

在来看 new 的执行过程：

1. 创建一个新的对象，让 this 指针指向它
2. 将函数的 prototype 对象的所有成员都赋给这个新的对象
3. 执行函数体，对这个对象进行初始化操作
4. 返回 1 中创建的对象

需要注意的是，原型对象的定义必须放在创建类的实例的语句之前，否则将不会起作用。

5、一种类的设计模式

```
function class1(){}
class1.prototype.someProperty=”aa”;
class1.prototype.someMethod=function(){};
```

这样的代码对类的定义清晰了很多，但每定义一个属性或方法都要使用一次 class1.prototype，代码体积变大，不易读。可以进一步改进，使用无类型对象的构造方法指定 prototype 对象

```
function class1(){}
class1.prototype={
    someProperty:”sample”,
    someMethod:function(){}
}
```

公有成员、私有成员和静态成员

前面所说的都是公有成员，该类的任何实例都对外公开这些属性和方法。

私有。javascript 并没有特殊的机制来定义私有成员，可以使用一些技巧来实现。

主要是通过变量的作用域性质来实现。一个函数内部定义的变量成为局部变量，该变量不能够被此函数外的程序所访问，但可以被函数内部定义的嵌套函数所访问。

```
<script language="JavaScript" type="text/javascript">
function class1(){
    var pp=" this is a private property";     //私有属性成员 pp
    function pm(){      //私有方法成员 pm，显示 pp 的值
        alert(pp);
    }
    this.method1=function(){
        pp="pp has been changed";
    }
    this.method2=function(){
        pm();           //在公有成员中调用私有方法
    }
}
var obj1=new class1();
obj1.method1();    //调用公有方法 method1
obj1.method2();    //调用公有方法 method2
```

```
</script>
```

实现静态成员

静态成员属于类，可以通过类名调用。在 javascript 中，可以给一个函数对象直接添加成员来实现静态成员，因为函数也是一个对象，所以对象的相关操作，对函数同样适用。

```
function class1(){}   //构造函数
class1.staticproperty="static sample";
class1.staticMethod=function(){
    alert(class1.staticproperty);
}
class1.staticMethod();
```

注意：这种实现方式类似于对对象的操作给对象添加属性跟方法。

如果要给每个函数对象都添加通用的静态方法，可以通过函数对象所对应的类 Function 来实现

```
Function.prototype.showArgsCount=function(){
    alert(this.length);
}
fucntion class1(a){}
class1.showArgsCount();
```

通过 Function 的 prototype 原型对象可以给任何函数都加上通用的静态成员。

6、使用 for(in)来实现反射

```
for(var p in obj){}
```

将返回所有对象的属性或方法

```
for(var p in obj){
    if(typeof(obj[p])=="function"){
        obj[p]();
    }else{
        alert(obj[p]);
    }
}
```

遇到方法执行之，遇到属性打印之。

7、类的继承

使用 prototype 实现继承

javascript 没有专门的机制来实现继承，可以通过拷贝一个类的 prototype 到另外一个类来实现继承。

```
function class1(){}
function class2(){}
```

```
class2.prototype=class1.prototype; //两个类的原型引用指向了同一个实例
class2.prototype.moreProperty1=”xxx”;
class2.prototype.moreMethod1=function(){}

javascript 提供了 instanceof 操作符来判断一个对象是否是某个类的实例
var obj1 = new class1(); var obj2=new class();
obj instanceof class1 //返回 true
obj instanceof class2//返回 true

表面看来上面的继承可以，但有问题：
看下面代码
<script language="JavaScript" type="text/javascript">
function class1(){
}
class1.prototype={
      m1:function(){
            alert(1);
      }
}
function class2(){
}
//让 class2 继承于 class1
class2.prototype=class1.prototype;
//给 class2 重复定义方法 method
class2.prototype.method=function(){
      alert(2);
}
//创建两个类的实例
var obj1=new class1();
var obj2=new class2();
//分别调用两个对象的 method 方法
obj1.method();
obj2.method();

</script>
```

当对 class2 进行 prototype 的修改，class1 的 prototype 也改变。class1 和 class2 仅仅是构造函数不同的两个类，保持着相同的成员定义。class1 和 class2 的 prototype 是完全相同的，是对同一个对象的引用。

8、利用反射和 prototype 实现继承

思路：使用反射列出所有父类 prototype 的成员，并复制给子类的 prototype 对象。

```
function class1(){
}
class1.prototype={
    method:function(){
        alert(1);
    },
    method2:function(){
        alert("method2");
    }
}
function class2(){
}
//让 class2 继承于 class1
for(var p in class1.prototype){
    class2.prototype[p]=class1.prototype[p];
}
//覆盖定义 class1 中的 method 方法
class2.prototype.method=function(){
    alert(2);
}
var obj1=new class1();
var obj2=new class2();
//调用 method 方法
obj1.method();   //执行结果 1
obj2.method();   //执行结果 2
//调用 method2 方法
obj1.method2(); //执行结果 method2
obj2.method2();//执行结果 method2
```

obj2 中重复定义的 method 已经覆盖了继承的 method 方法，同时 method2 方法没有受到影响。而且 obj1 中欧哪个的 method 方法仍然保持原有状态。

为了方便开发，可以为每个类添加一个共有的方法，实现类的继承：

```
Function.prototype.extends=function(baseClass){
    for(var p in baseClass.prototype){
        this.prototype[p]=baseClass.prototype[p];
    }
}
```

这样可以直接

```
class2.extends(class1)
```

不过，在 class2 中添加类成员定义的时候，不能给 prototype 直接赋值，而只能对其属性进行赋值，也就是：

不能写成：

```
class2.prototype={
```

```
    …
}
```

因为：相当于对 class2 的 prototype 进行了重新定义

而只能写成

```
class2.prototype.propertyName=sth;
class2.prototype.methodName=function(){}
```

举例：

```
//定义 extend 方法,Object 自身的方法
Object.extend = function(destination, source) {
   for (p in source) {
      destination[p] = source[p];
   }
   return destination;
}
//Object 原型 extentd 上的方法，所有 obj 对象都有此方法
Object.prototype.extend = function(object) {
   return Object.extend.apply(this, [this, object]);
}
function class1(){
      //构造函数
}
class1.prototype={
      method:function(){
            alert("class1");
      },
      method2:function(){
            alert("method2");
      }
}
function class2(){
}
//让 class2 继承于 class1 并定义新成员
class2.prototype=(new class1()).extend({
      method:function(){
            alert("class2");
      }
});

//创建两个实例
var obj1=new class1();
var obj2=new class2();
//试验 obj1 和 obj2 的方法
obj1.method();
obj2.method();
```

```
obj1.method2();
obj2.method2();
```

Object.extend 方法很容易理解，它是 Object 类的一个静态方法，用于将参数中 source 的所有属性都拷贝到 destination 对象中，并返回 destination 的引用。
因为 Object 是所有对象的父类，所以这里是为所有对象都加一个 extend 方法
这里使用了 apply，是将 Object 类的静态方法作为对象的方发运行，this 是指向对象实例自身，第二个参数是一个数组，包括两个元素：对象本身和传进来的对象参数。
函数功能是将参数对象 object 的所有属性和方法复制给调用该方法的对象自身，并返回自身的引用。

在调用的时候，使用 class2.prototype=(new class1()).extend({

```
    method:function(){
        alert("class2");
    }
});
```

而不是使用

```
class2.prototype=class1.prototype.extend({
    method:function(){
        alert("class2");
    }
});
```

因为 new class1 是创建了对象，并将实例对象的成员复制给 class2 的 prototype。本质相当于创建了 class1 的 prototype 的一个拷贝，在这个拷贝上进行操作不会影响原有类的 prototype 的定义。

9、抽象类

javascript 中，没有具体的 abstract 关键字。变通实现。
java 中的抽象类。
1. 不能实例化
2. 可以有抽象方法
3. 可以有方法的实现

在抽象类中，定义 N 个方法。比如两个。f1 和 f2
f1 调用 f2 但是 f2 是空实现，这个 f2 由子类完成。

抽象类也就是实现了一种模板策略，抽象类中定义的方法可以调用一个不存在的方法，这个方法在子类中实现。
举例：

```
//定义 extend 方法
Object.extend = function(destination, source) {
  for (property in source) {
    destination[property] = source[property];
  }
  return destination;
```

```
}
Object.prototype.extend = function(object) {
   return Object.extend.apply(this, [this, object]);
}
//定义一个抽象基类 base
function base(){}
base.prototype={
      initialize:function(){
            this.oninit();    //调用了一个虚方法
      }
}
function class1(){
}
//让 class1 继承于 base 并实现其中的 oninit 方法
class1.prototype=(new base()).extend({
      oninit:function(){    //实现抽象基类中的 oninit 虚方法
            //oninit 函数的实现
      }
});
```

当 class1 的示例中调用继承得到的 initialize 方法的时候，就会自动执行子类中的 oninit()方法。这里也能看出解释性语言的特点，在运行的时候才检查。当然如果希望在父类中定义一个虚方法，也可以

```
function base(){}
base.prototype={
      initialize:function(){
            this.oninit();
      }
      oninit:function(){}        //虚方法，由子类实现。
}
```