JS高级程序设计3rd阅读笔记

第一章 在HTML中使用JavaScript

1.1 <script></script>

<body>  
 <script type="text/javascript">  
 **function** sayScript()  
 {  
 alert("</script>");

<!-- </script>解析到这里就结束了，所以后面的都会在浏览器输出 -->

}  
 sayScript();  
 </script>  
</body>

1.2 script标签的位置

<body>  
  
  
 *<!--这里放内容，把JS文件放在最底下，让JS文件最后加载，保证页面优先加载-->* <script type="text/javascript" src="function.js"></script>  
</body>

1.3 延迟脚本

--------------------------------------------------------------------------------

.html

--------------------------------------------------------------------------------

<body>  
  
 *<!--这里放内容，让JS文件最后加载，保证页面优先加载-->* <script type="text/javascript" defer="defer" src="function.js"></script>  
 <script>  
 alert("111");  
 </script>  
</body>

--------------------------------------------------------------------------------

function.js

--------------------------------------------------------------------------------

alert("222");

--------------------------------------------------------------------------------

注：defer="defer"：延迟脚本文件的执行，使其遇到</html>标签后再执行，延迟脚本按照它们指定的顺序执行。

使用外部JS文件时不能使用document.write()，Chrome浏览器提示：

Failed to execute 'write' on 'Document': It isn't possible to write into a document from an asynchronously-loaded external script unless it is explicitly opened.

function.js:5 Failed to execute 'write' on 'Document': It isn't possible to write into a document from an asynchronously-loaded external script unless it is explicitly opened.

所以这里先执行alert("111");后执行alert("222");去掉defer属性，执行顺序相反。

1.4 异步脚本

<body>  
  
  
 *<!--这里放内容，让JS文件最后加载，保证页面优先加载-->* <script type="text/javascript" async="async" src="function.js"></script>  
</body>

注：async="async"：不让页面等待脚本的下载和执行，从而异步加载页面其他内容，虽然不必等待其他脚本，也不会阻塞文档呈现，但不能保证它们按照页面中出现的顺序执行。

使用外部JS文件时不能使用document.write()，Chrome浏览器提示：

Failed to execute 'write' on 'Document': It isn't possible to write into a document from an asynchronously-loaded external script unless it is explicitly opened.

function.js:5 Failed to execute 'write' on 'Document': It isn't possible to write into a document from an asynchronously-loaded external script unless it is explicitly opened.

1.5 兼容XHTML（嵌入代码，非外部JS文件）

<body>  
  
 *<!--(<)号在HTML中会被解析为开始一个新的标签，如<body>中的(<)-->* <script type="text/javascript">  
 //<![CDATA[  
 **function** compare(a,b)  
 {  
 **if**(a<b) //如果不使用CDATA片段，这个<号在XHTML中不会被解析为小于号  
 {  
 a = b;  
 }  
 }  
 //]]  
 </script>  
 *<!--这里放内容，让JS文件最后加载，保证页面优先加载-->* <script type="text/javascript" async="async" src="function.js"></script>  
</body>

1.6 不推荐的使用语法（嵌入代码，非外部JS文件）

<body>  
  
 *<!--(<)号在HTML中会被解析为开始一个新的标签，如<body>中的< -->* <script type="text/javascript">  
 *<!--* **function** compare(a,b)  
 {  
 **if**(a<b) //如果不使用CDATA片段，这个<号在XHTML中不会被解析为小于号  
 {  
 alert(a);  
 }  
 }  
  
 compare(1,2);  
 //-->  
 </script>  
 *<!--这里放内容，让JS文件最后加载，保证页面优先加载-->* <script type="text/javascript" async="async" src="function.js"></script>  
</body>

--------------------------------------------------------------------------------

注：所有浏览器都支持了JavaScript脚本，没必要这么做了，相反，这种注释会使脚本在XHTML中被忽略。

1.7 <noscript>标签

<body>  
 <noscript>  
 <p>浏览器不支持JS脚本</p>  
 </noscript>  
 *<!--这里放内容，让JS文件最后加载，保证页面优先加载-->* <script type="text/javascript" async="async" src="function.js"></script>  
</body>

--------------------------------------------------------------------------------

注： 1.浏览器不支持脚本

2.虽然支持脚本，但浏览器禁用了脚本

<noscript>只要满足以上任何一个要求，都会执行标签内的内容。

第二章 基本概念

2.1 标识符

1.第一个字母必须是字母、下划线(\_)、或一个美元符号($)  
2.其他字符可以是字母、下划线、美元符号或数字  
3.驼峰大小写格式： myFirst，yourCar

4.不能把关键字、保留值、布尔值和null用作标识符

2.2 typeof操作符

**var** str = "string";  
**var** fun = **function**(){};  
  
alert(**typeof null**); //object，null被认为是空的对象的引用  
alert(**typeof** 95); //number  
alert(**typeof** str); //string  
alert(**typeof** fun); //function

2.3 undefined、null类型

//声明但却未初始化的变量的默认值是undefined  
//引入undefined是为了区分空对象指针null和未初始化的变量  
//如果养成了声明并初始化的习惯，那么就可以通过typedef来检测变量有没有初始化  
**var** age;  
alert(**typeof** age); //undefined，表明没有初始化  
//如果是没有声明的变量  
alert(**typeof** ame); //undefined，表明没有声明  
alert(**typeof** name);//string ?? 为什么是string，根本没声明过!  
  
  
**var** car = **null**; //一个空对象,建立对象时最好初始化null  
alert(**typeof** car); //object  
**if**(car != **null**)  
{  
 //确保对象非空，然后执行操作  
}  
  
//其实undefined派生自null  
//在ECMAScript第三版本中引入undefined来区分空对象指针和未经初始化的变量  
alert(**null** == undefined); //true  
alert(**null** === undefined);//false

2.4 Boolean类型

------------------------------------------------------------------  
 数据类型 **true false**------------------------------------------------------------------  
 Boolean **true false** String 非空字符 ""  
 Number 任何非零数值(包括无穷大) 0和NaN  
 Object 任何对象 **null** Undefined undefined  
------------------------------------------------------------------  
//布尔值自动类型转换  
**var** a = NaN;  
  
**if**(!a)  
{  
 alert(**false**); //false  
}  
  
**var** b = {};  
  
**if**(b)  
{  
 alert(**true**); //true  
}

2.5 Number类型

//浮点数值的最高精度是17位  
**var** a = 0.1; //存储的二进制数并不能精确的表示0.1  
**var** b = 0.2; //  
  
alert(a+b); //0.30000000000000004  
  
**if**((a + b) == 0.3) {   
 //  
}  
  
**else**{  
 alert("here"); //执行了这条代码  
}

2.6 Number、parseInt and parseFloat

*/\*\*  
 \* function Number(value) {}  
 \* 功能：转换为数字  
 \* 参数：value ——> 可以传入任何形式  
 \*/*document.write(Number(**true**)+"<br/>"); //1  
document.write(Number(1)+"<br/>"); //1  
document.write(Number(undefined)+"<br/>"); //NaN  
document.write(Number("000011")+"<br/>"); //11  
document.write(Number("0x0B")+"<br/>"); //11  
document.write(Number("")+"<br/>"); //0  
  
//其余任何形式都是NaN  
document.write(Number("abc")+"<br/>"); //NaN  
**var** a = {};  
document.write(Number(a)+"<br/>"); //NaN  
document.write("<br/>");  
  
*/\*\*  
 \* parseInt = function(s,radix) {};  
 \* 功能：转换为整型数，规则更合理  
 \* 参数：1. s ——> 可以传入任何形式  
 \* 2. radix ——> 表明需要转换的进制数  
 \* 说明：第2参数可选，默认转换为十进制数  
 \*/*document.write(parseInt(**true**)+"<br/>"); //NaN  
document.write(parseInt(1)+"<br/>"); //1  
document.write(parseInt(undefined)+"<br/>"); //NaN  
document.write(parseInt("000011")+"<br/>"); //11  
document.write(parseInt("0x0B")+"<br/>"); //11  
document.write(parseInt("")+"<br/>"); //NaN  
  
document.write("<br/>");  
document.write(parseInt("1234acd")+"<br/>"); //1234  
document.write(parseInt("")+"<br/>"); //NaN  
document.write(parseInt("0xA")+"<br/>"); //10  
document.write(parseInt(22.5)+"<br/>"); //22  
document.write(parseInt("070")+"<br/>"); //70 ECMAScript5  
//document.write(parseInt("070")+"<br/>"); //56(八进制) ECMAScript3  
  
document.write(parseInt("070",8)+"<br/>"); //56(八进制)  
document.write(parseInt("070",10)+"<br/>"); //70(十进制)  
document.write(parseInt("AF",16)+"<br/>"); //175(十六进制)  
document.write(parseInt("AF")+"<br/>"); //NaN  
  
*/\*\*  
 \* parseFloat = function(x) {};  
 \* 功能：转换为浮点数  
 \* 参数：x ——> 可以传入任何形式  
 \* 说明：该函数只转换十进制数，始终忽略前导的0  
 \*/*document.write("<br/>");  
document.write("<br/>");  
document.write(parseFloat(**true**)+"<br/>"); //NaN  
document.write(parseFloat(1)+"<br/>"); //1  
document.write(parseFloat(undefined)+"<br/>"); //NaN  
document.write(parseFloat("000011")+"<br/>"); //11  
document.write(parseFloat("")+"<br/>"); //NaN  
  
document.write(parseFloat("0x0B")+"<br/>"); //0(不解析十六进制数，始终为0)  
document.write(parseFloat("1234blue")+"<br/>"); //1234(整数)  
document.write(parseFloat("22.5")+"<br/>"); //22.5  
document.write(parseFloat("22.5.6.8")+"<br/>"); //22.5  
document.write(parseFloat("0908.5")+"<br/>"); //908.5  
document.write(parseFloat("3.125e7")+"<br/>"); //31250000

2.7 String类型

*/\*\*  
 \* Number.prototype.toString = function(radix) {};  
 \* 功能：转换为字符串  
 \* 参数：radix ——> 需要转换为radix进制的字符串，默认为10进制  
 \*/***var** num = 10;  
document.write(num.toString()+"<br/>"); //10  
document.write(num.toString(2)+"<br/>"); //1010  
document.write(num.toString(8)+"<br/>"); //12  
document.write(num.toString(10)+"<br/>"); //10  
document.write(num.toString(16)+"<br/>"); //a  
document.write("<br/>");  
  
//null和undefined没有toString的方法，因为不是Number对象  
  
*/\*\*  
 \* function String(value) {}  
 \* 功能：转换为字符串  
 \* 说明：1.如果有toString的方法，先调用没有参数的该方法  
 \* 2.如果是null，返回"null"  
 \* 3.如果是undefined，返回"undefined"  
 \*/*document.write(String(10)+"<br/>"); //10  
document.write(String(**true**)+"<br/>"); //true  
document.write(String(**null**)+"<br/>"); //null  
document.write(String(undefined)+"<br/>"); //undefined

2.8 布尔操作符

/\*逻辑非\*/  
**var** oPerson = {};  
document.write(!oPerson+"<br/>"); //false  
document.write(!""+"<br/>"); //true  
document.write(!"非空"+"<br/>"); //false  
document.write(!0+"<br/>"); //true  
document.write(!3+"<br/>"); //false  
document.write(!Infinity+"<br/>"); //false  
document.write(!**null**+"<br/>"); //true  
document.write(!NaN+"<br/>"); //true  
document.write(!undefined+"<br/>"); //true  
  
/\*逻辑与\*/  
//var found = true;  
//var result = (found && someUndefinedVariable);  
//Uncaught ReferenceError: someUndefinedVariable is not defined  
//alert(result);  
**var** found = **false**;  
**var** result = (found && someUndefinedVariable);  
alert(result); //false  
  
/\*逻辑或\*/  
//var found = false;  
//var result = (found || someUndefinedVariable);  
//Uncaught ReferenceError: someUndefinedVariable is not defined  
//alert(result);  
**var** bFound = **true**;  
**var** result = (bFound || someUndefinedVariable);  
alert(result); //true

2.9 乘性、加减操作符

/\*乘法\*/  
**var** num = NaN \* 3;  
document.write(num+"<br/>"); //NaN  
num = Infinity \* 0;  
document.write(num+"<br/>"); //NaN  
num = Infinity \* 4;  
document.write(num+"<br/>"); //Infinity  
num = Infinity \* -4;  
document.write(num+"<br/>"); //Infinity  
num = Infinity \* Infinity;  
document.write(num+"<br/>"); //Infinity  
num = "3" \* 4;  
document.write(num+"<br/>"); //12  
/\*除法\*/  
num = 3 / NaN;  
document.write(num+"<br/>"); //NaN  
num = NaN / 3;  
document.write(num+"<br/>"); //NaN  
num = Infinity / Infinity;  
document.write(num+"<br/>"); //NaN  
num = 4 / 0;  
document.write(num+"<br/>"); //Infinity  
num = 4 / -0;  
document.write(num+"<br/>"); //-Infinity  
num = Infinity / 4;  
document.write(num+"<br/>"); //Infinity  
num = "12" / 4;  
document.write(num+"<br/>"); //3  
/\*模除\*/  
num = Infinity % 4;  
document.write(num+"<br/>"); //NaN  
num = 4 % 0;  
document.write(num+"<br/>"); //NaN  
num = Infinity % Infinity;  
document.write(num+"<br/>"); //NaN  
num = 55 % Infinity;  
document.write(num+"<br/>"); //55  
num = 0 % 0;  
document.write(num+"<br/>"); //NaN  
num = "12" % 25;  
document.write(num+"<br/>"); //12  
/\*加法\*/  
**var** num = 5;  
**var** message = "Two num add is " + num + num;  
document.write(message+"<br/>"); //Two num add is 55  
**var** message = "Two num add is " + (num + num);  
document.write(message+"<br/>"); //Two num add is 10  
/\*减法\*/  
**var** result = 5 - **true**;  
document.write(result+"<br/>"); //4  
result = NaN - 1;  
document.write(result+"<br/>"); //NaN  
result = 5 - **null**;  
document.write(result+"<br/>"); //5  
result = 5 - "";  
document.write(result+"<br/>"); //5  
result = 5 - "2";  
document.write(result+"<br/>"); //3

2.10 关系操作符

/\*<|>\*/  
**var** result = "23" < 3;  
document.write(result + "<br/>"); //false  
result = "23" < "3"  
document.write(result + "<br/>"); //true  
result = "a" < 3;  
document.write(result + "<br/>"); //false,因为a转换为NaN  
result = NaN < 3;  
document.write(result + "<br/>"); //false  
result = NaN >= 3;  
document.write(result + "<br/>"); //false  
document.write("<br/>");   
  
  
/\*==|!=|===|!==\*/ //=== 严格等于或全等于不进行类型转换  
result = (**null** == undefined);  
document.write(result + "<br/>"); //true  
result = (**null** === undefined);  
document.write(result + "<br/>"); //false  
result = (NaN == NaN);  
document.write(result + "<br/>"); //false  
result = (undefined == 0);  
document.write(result + "<br/>"); //false  
result = (**null** == 0);  
document.write(result + "<br/>"); //false  
result = ("55" == 55);  
document.write(result + "<br/>"); //true  
result = ("55" === 55);  
document.write(result + "<br/>"); //false  
  
/\*,\*/  
result = (1,2,3,4,5,6);  
document.write(result + "<br/>"); //6

第三章 变量、作用域和内存问题

3.1 基本类型和引用类型的复制区别

/\*基本类型(null undefined Number String Boolean)复制\*/  
**var** num1 = 5;  
**var** num2 = num1;  
num1 = 10;  
alert(num2); //5  
  
/\*  
  
复制前  
  
-------------------------  
 |  
-------------------------  
 |  
-------------------------  
 num1 | 5(Number类型)  
-------------------------  
  
复制后  
  
-------------------------  
 |  
-------------------------  
 num2 | 5(Number类型)  
-------------------------  
 num1 | 5(Number类型)  
-------------------------  
  
注：两个不同的内存空间，相互之间的变量互不影响  
  
 \*/  
  
/\*引用类型(Object类型、Array类型、Date类型、Function类型、RegExp类型)复制\*/  
**var** oPerson = {};  
**var** oPerson1 = oPerson;  
oPerson.Name = "ZHUXIANKANG";  
alert(oPerson1.Name); //ZHUXIANKNG  
  
/\*  
  
 复制前  
  
-------------------------  
 |  
-------------------------  
 |  
------------------------- 指向了堆中的一个对象  
 oPerson | (Object类型) ---------------------  
------------------------- |  
 |  
 复制后 Object  
 |  
------------------------- |  
 | |  
-------------------------指向了堆中的同一个对象 |  
 oPerson1 | (Object类型) ---------------------  
------------------------- |  
 oPerson | (Object类型) ---------------------  
-------------------------  
  
 注：复制时，其实内存中存放的是指向对象的指针，两个变量实际引用的是同一个对象  
  
 所以在进行函数传递参数时要注意，进行类型引用的参数传递时，传递的是指向对象的  
  
 指针地址，如果函数内的局部变量发生了变化，会反映到函数的外部。  
  
 \*/  
  
**function** setName(obj)  
{  
 obj.Name = "JavaScript";  
}  
  
**var** oPerson = **new** Object();  
setName(oPerson);  
alert(oPerson.Name); //JavaScript

3.2 类型检测

/\*基本类型检测\*/  
/\*null undefined Number String Boolean\*/  
**var** s = "1235";  
**var** b = **true**;  
**var** c = 22;  
**var** u;  
**var** n = **null**;  
//var o = new Object();  
  
document.write(**typeof** s + "<br/>"); //string  
document.write(**typeof** b + "<br/>"); //boolean  
document.write(**typeof** c + "<br/>"); //number  
document.write(**typeof** u + "<br/>"); //undefined  
document.write(**typeof** n + "<br/>"); //object 空指针对象  
//document.write(typeof o + "<br/>"); //object  
  
  
/\*引用类型检测\*/  
/\*Object类型、Array类型、Date类型、Function类型、RegExp类型\*/  
/\*result = variable instanceof caonstructor\*/  
**var** oDate = **new** Date();  
**var** oObject = **new** Object();  
**var** oArray = **new** Array();  
**var** oFunction = **new** Function();  
**var** oRegExp = **new** RegExp();  
  
document.write((oDate **instanceof** Date) + "<br/>"); //true  
document.write((oObject **instanceof** Object) + "<br/>"); //true  
document.write((oArray **instanceof** Array) + "<br/>"); //true  
document.write((oFunction **instanceof** Function) + "<br/>"); //true  
document.write((oFunction **instanceof** Array) + "<br/>"); //false  
document.write((oRegExp **instanceof** RegExp) + "<br/>"); //true  
  
/\*引用类型的值是保存在内存中的对象，Object是所有对象的基础\*/  
/\*所有对象都有Object的属性和方法\*/  
document.write((oDate **instanceof** Object) + "<br/>"); //true  
document.write((oObject **instanceof** Object) + "<br/>"); //true  
document.write((oArray **instanceof** Object) + "<br/>"); //true  
document.write((oFunction **instanceof** Object) + "<br/>"); //true  
document.write((oFunction **instanceof** Object) + "<br/>"); //true  
document.write((oRegExp **instanceof** Object) + "<br/>"); //true

3.3 作用域链

/\*全局执行环境是Window对象，所以所有的全局变量和函数都是作为Window对象的属性和方法创建的\*/  
**var** color = "red"; //window对象的属性 color  
alert(window.color); // red  
  
**function** alertColor() //window对象的方法 alertColor  
{  
 alert(window.color);  
}  
  
window.alertColor(); //red  
  
  
/\*每个执行环境都有一个与之关联的变量对象  
\* 这个环境中所定义的所有变量和函数都保存在这个对象中  
\* 执行环境中的所有代码执行完毕后，环境被销毁  
\* 全局执行环境直到关闭网页或浏览器才会被销毁\*/  
  
/\*当代码在环境中执行时，会创建变量对象的作用域链  
\* 作用域链的前端始终是当前执行代码所在环境的变量对象  
\* 如果这个环境是对象，则将其活动对象作为变量对象  
\* 活动对象在最开始时只包含一个变量，即arguments对象  
\* 这个对象在全局环境中是不存在的  
\* 作用域链的下一个变量对象来自包含该当前执行代码所在环境变量对象的环境  
\* 一直延续到全局执行环境，所以window始终是作用域链中的最后一个对象\*/  
  
**var** color = "blue"; //window对象  
   
**function** changeColor() { //changeColor对象，也是window对象的方法  
 **var** anotherColor = "red";  
  
 **function** anotherChangeColor(){ //anotherChangeColor对象，也是changeColor对象的方法  
 **var** tempColor;  
 tempColor = color;  
 color = anotherColor;  
 anotherColor = tempColor;  
 }  
  
 anotherChangeColor();  
}  
  
changeColor();

/\*anotherChangeColor执行环境中，作用域链包含三个对象

它自己的变量对象(包括arguments对象)（局部环境）、changeColor对象（局部环境）、

window对象（全局执行环境）\*/

作用域链

window

|

|--------color

|--------changeColor()

|

|--------anotherColor

|--------anotherChangeColor()

|

|---------tempColor

注意：函数参数也被当作变量来对待，访问规则与执行环境中的其他变量相同

/\*延长作用域链\*/

1.try-catch语句的catch块

2.with语句

With语句会将指定对象添加到作用域链中

**function** buildUrl(){  
 **var** qs = "?debug=true";  
 **with**(location){  
 **var** url = href + qs;  
 }  
  
 **return** url;  
}  
  
document.write(buildUrl());  
//http://localhost:63342/%E7%A4%BA%E4%BE%8B%E4%BB%A3%E7%A0%81/index.html?debug=true

//其实href实际是引用了location.href

//使用with将location对象的所有属性和方法添加到了当前执行环境的作用域链的前端

3.4 管理内存

/\*垃圾回收机制\*/  
/\*标记清除和引用计数\*/  
/\*引用计数中如果有循环引用，容易造成内存泄漏\*/

/\*优化内存占用的最佳方式是一旦数据不再有用，通过将其设置为null来释放引用

这个做法叫做解除引用，适用于大多数的全局变量和全局对象的属性

局部变量会在离开执行环境时被自动解除引用\*/

/\*解除引用的真正作用是让值脱离执行环境，以便垃圾收集器下次运行时将其回收\*/

1.基本类型值在内存中占据固定大小的空间，保存在栈内存中

2.引用类型的值是对象，保存在堆内存中

3.包含引用类型值得变量实际上包含的并不是对象本身，而是一个指向该对象的指针

4.从一个变量向另一个变量复制引用类型的值，复制的其实是指针，两个变量最终都指向同一个对象

5.离开作用域的值将被自动标记为可以回收，因此将在垃圾收集期间被删除

6.标记清除是目前主流的垃圾收集算法，这种算法的思想是给当前不使用的值加上标记，然后回收其内存

7.引用计数是跟踪记录所有值被引用的次数

第四章 引用类型

4.1 Object类型

**function** displayName(oArg){  
 **var** output = "";  
  
 **if**(**typeof** oArg.age == "number"){  
 output = oArg.age + " ";  
 }  
  
 **if**(**typeof** oArg.name == "string"){  
 output = output + oArg.name;  
 }  
  
 alert(output);  
}  
  
//这种模式适用于向函数传入大量的可选的参数的情形，使用对象字面量来封装多个可选的参数  
displayName({ name:"zxk",  
 age:29  
});  
  
displayName({  
 name:"vitco"  
});

注：创建object实例的方法 1.var oPerson = new Object();

2.var oPerson = {name: , age: };

3.var oPerson = {};

属性访问的方式： 1.点表示法

2.方括号表示法（可以用于遍历属性、可以通过变量来访问属性、点表示法中不能表示的容易导致错误的字符或属性名中使用的是关键字或保留字）

4.2 Array类型

创建Array实例的方法 1. **var** colors = **new** Array();  
**var** colors = **new** Array(20);  
**var** colors = **new** Array("red","blue","yellow");

2. **var** name = [];

**var** value = [1,2,3];

/\*数组索引\*/  
**var** colors = ["red","yellow","green"];  
colors[3] = "brown"; //新增第四项  
alert(colors[3]); //brown  
  
/\*数组长度属性：length\*/  
**var** colors = ["red","yellow", "green"];  
colors.length = 2;  
alert(colors[2]); //undefined，移除了第三项green  
  
colors[colors.length] = "red"; //在最后添加一项,数组最后一项的索引始终是length-1，所以新添加的最后一项是length  
colors[colors.length] = "black"; //在最后添加另一项  
document.write(colors[2] + "<br/>"); //red  
document.write(colors[3] + "<br/>"); //black  
  
/\*检测数组\*/  
**var** arr = [];  
  
**if**(arr **instanceof** Array) {  
 alert("is array");  
}  
  
//有缺陷，instanceof假定只有一个全局执行环境  
//如果包含多个框架，实际上就有两个以上不同的全局执行环境  
//因为Array是window的属性  
//从一个框架中传入另一个框架中的数组，在第二个框架中使用instanceof，会判断为不是数组  
  
**if**(Array.isArray(arr)) {  
 alert("is array");  
}  
  
//有缺陷，浏览器支持问题  
//重写isArray可以在尚未支持Array.isArray的浏览器中使用  
**function** isArray(value) {  
 **return** Object.prototype.toString.call(value) == "[object Array]";  
}  
  
**if**(isArray(arr)) {  
 alert("is array");  
}  
  
  
//更完美的写法  
**function** isArray(value) {  
  
 **if**(Array.isArray) {  
 **return** Array.isArray(value);  
 } **else** {  
 **return** Object.prototype.toString.call(value) == "[object Array]";  
 }  
}  
  
**if**(isArray(arr))  
{  
 alert("isArray");  
}

/\*数组转换成字符串\*/  
**var** colors = ["red","blue","yellow"];  
  
document.write(colors.toString() + "<br/>");//red,blue,yellow 下面都是这个结果  
document.write(colors.valueOf() + "<br/>");//返回数组，调用数组每一项的toString()方法  
alert(colors); //在后台调用数组每一项的toString()方法  
  
**var** person1 = {  
 toString: **function**() {  
 **return** "person1 toString";  
 },  
  
 toLocaleString: **function**() {  
 **return** "person1 toLocaleString";  
 }  
};  
  
**var** person2 = {  
 toString: **function**() {  
 **return** "person2 toString";  
 },  
  
 toLocaleString: **function**() {  
 **return** "person2 toLocaleString";  
 }  
}  
  
**var** person = [person1,person2];  
alert(person); //person1 toString，person2 toString  
alert(person.toString()); //person1 toString，person2 toString  
alert(person.toLocaleString()); //person1 toLocaleString，person2 toLocaleString  
  
alert(person.join());  
alert(person.join(undefined));  
alert(person.join(",")); //以上三个结果一样，都是以逗号分隔，  
alert(person.join("|")); //person1 toLocaleString|person2 toLocaleString

/\*数组的栈方法\*/  
  
//栈是一种后进先出的数据结构

//数组的后端添加项，数组的后端删除项  
**var** num = **new** Array();  
**var** count = num.push(1,2,3); //返回数组长度，推入3个数  
alert(count);  
**var** item = num.pop(); //推出最后一个数3,返回的是最后一个推出的数  
alert(item); //3  
alert(num); //1,2  
  
num.push(3);  
num.push(4);  
alert(num); //1,2,3,4  
num.pop();  
alert(num); //1,2,3

/\*数组的队列方法\*/  
  
//队列数据结构的访问规则是先进先出  
//数组的后端添加项，数组的前端删除项  
**var** num = **new** Array();  
**var** count = num.push(1,2,3); //3  
**var** item = num.shift(); //取得第一项，返回第一项的值  
alert(num); //2,3 推出了1  
  
num.push(4,5,6);  
num.shift();  
alert(num); //3,4,5,6 推出了2

//数组的前段添加项，数组的末端移除项  
**var** num = **new** Array();  
**var** count = num.unshift(1,2,3); //在前端屠推入3项  
alert(num); //1,2,3 注意不是3，2，1  
  
num.pop();  
alert(num); //1,2  
  
num.unshift(3);  
alert(num); //3,1,2