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/>"); //false  
  
  
/\*==|!=|===|!==\*/ //=== 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