高级程序设计徐编著课后习题答案

1:原始值和引用值

原始值：存储在栈中的简单数据。即：它们的值是直接存储在变量访问的值。

引用值：存储在堆中的对象。即，存储在变量出的值是一个指针，指向存储对象的内存处。

存放在栈中可以快速查询，　由于原始类型占据的空间是固定的，　所以可以将它们存储在较小的内存区域 —— 栈中。

为变量赋值时，　ECMAScript　的解释程序必须判断该值是原始类型还是引用类型。

2：原始类型

ECMAScript　有 5 种原始类型　(primitive type)　即：Undefined　, Null , String , Boolean　, Number　。

可以用　typeof　运算符判断

返回值讲解

undefined　　如果变量是　Undefined　类型

boolean　　如果变量是　Boolean　类型

number　　如果变量是　Number　类型

string　　String　类型

object　　如果变量是引用类型或　　Null　类型

现在，　null　被认为是对象的占位符，但因　　js 的设计错误，　Null　仍是原始类型。

3:Undefined　类型

该类型只有一个值，即　　undefined　。

如果声明了变量为初始化，该变量的默认值为　　undefined

当函数无明确返回值时，返回的也是　　undefined

未声明的变量仅可以用　　typeof　运算符，其他运算符时报错　　,虽不会包　js 错误，但不会显示想要的效果。

```
var test;
alert(typeof test);　　//undefined
alert(test==undefined); //true
alert(test=="undefined") //false
```

4：Null　类型

该类型只有一个专用值　　null，值　defined　实际是从值　null　派生来的，因此　ECMAScript　把它们定义是为相等的。

```
alert(null=defined);　　//true
```

尽快两值相等，但它们的含义不同。

undefined 是声明了变量但为对其初始化。

null 用于尚未存在的对象，如果函数或方法返回的是对象，那么找不到该对象时，通常返回 null

5：Bloolean 类型

有两个值 true 和 false，即使 false 不等域 0，0 也可以在必要时被转换成 false

6：Number 类型

其值包括整数和浮点数。

var test=086; //86

var t=070; //八进制 56

其还包括几个特殊的值： Number.MAX_VALUE 和 Number.MIN_VALUE, 它们定义了 Number 值集合的外边界

超出边界的值就会被赋值为边界，也也意味着不再有数字值。

如果计算返回的结果是无穷大值，那该结果不能再用于计算。

事实上有专门的值表示无穷大。

infinity:Number.POSITIVE_INFINITY,-infinity:Number.NAGATIVE_INFINITY

有判断是否有穷的方法 isFinity

var iResult=iNum*some_large_number;

alert(isFinite(iResult)) ; finite:adj ，有限的。 finity ：n，有限

还有一个特殊值 NaN(not a number), 发生在类型 (String,Boolean) 转换失败时。

但由于其特殊性自身不等于自身。

alert(NaN==NaN) ;//false

不推荐使用 NaN，而是用 isNaN() 进行数字的判断。

7：String 类型

是唯一没有固定大小的原始类型

8:类型转换

3 种主要的原始值 Boolean 值，数字和字符串都有 toString() 方法

Number 类型的 toString 比较特殊。默认模式和基模式

默认模式下： toSring 方法只是用相应的字符串输出数字值

var a=070;//56

var b=10.0;//10

var c=0.3e2;//30

基模式：

var iNum=10;

alert(iNum.toString(2));//1010

alert(iNum.toString(8));//12

alert(iNum.toString(16));//A

---

parseInt("123and");//123

parseInt("red");//NaN

parseInt("AF",16);//175

如果十进制数包含前导　0，那么最好用基为　10. 否则会得到八进制

parseInt("010");//8

parseInt("010",8);//8

parseInt("010",10);//10

---

parseFloat() 与 paseInt 不同之处是，字符串必须是以十进制形式表示浮点数。因该方法会忽略前导 0。

parseFloat("123abc");//123

parseFloat("0xA");//NaN

parseFloat("120.3.2");//120.3

parseFloat 与 paseInt 另一个不同之处是，没有基模式。

----parseInt("12.3");//12

----parseFloat("123");//123

---------- 强制类型转换 ----------

---------- 运算符 ----------

1：delete

删除对以前定义的【对象】属性或方法的应用。

var a=new Object();

a.name="shiy";

alert(a.name);//shiy

delete a.name;

alert(a.name);//undefined

这里删除 name 属性，意味着强制解除对它的引用，将其值设置为 undefined

2：void

void 运算符对任何值都返回 undefined 。该运算符通常用于避免输出不应该输出的值。

例如：从 html 的<a>元素调用 javascript 函数，要正确做到这一点，函数不能返回有效值，否则浏览器将清空页面，只显示函数的结果。

click me

此时， window.open 方法返回对新窗口的引用。然后该对象被转换成要显示的字符串。

为避免这种结果，可以用 void 运算符调用 window.open() 函数：

click me

这使 window.open() 调用返回 undefined ，它不是有效值，不会显示在浏览器中。

【记住】没有返回值的函数真正返回的是 undefined 。

3：for-in 语句

for-in 语句是严格的迭代，用于枚举对象的属性

for(propert in expression) statement

例如 :

for(sProp in window){

alert(sProp);

}

4:with 语句

with 语句用于设置代码在特定对象中的作用域。

语法： with(expression) statement

例如：

var ss="hello,word";

with(smessage){

alert(toUpperCase());

}

【注意】 with 语句是运行缓慢的代码段， 尤其是在已经设置了属性值时， 通常情况下尽量避免使用它。

5：js 函数重载

例如

function a(){alert("no");}

function a(ok){alert(ok);}

alert();

alert("good");

结果会显示： undefined ， good

原因是： 可用相同的名字在同一个作用域中定义两个函数, 而不会引发错误, 但真正使用的是最后一个函数。

6:【arguments 对象】

在函数对象代码中, 使用特殊对象 arguments , 开发者无需明确指出参数名, 就能访问他们。

如：在函数 sayHi() 中，第一个参数是 message ，用 arguments[0] 也可以访问这个值。

```
function sayHi(){
if (arguments[0]=="bye")
return;
alert(arguments[0]);
}
```

此外，还可以用 arguments 对象检测传递给函数的参数个数，引用属性 arguments.length 即可。

例如： function howManyArgs(){

```
alert(arguments.length);
}
howManyArgs("stirng",20);//2
```

与其他程序设计语言不同, ECMAScript 不会检验传递给函数的参数个数是否等于函数定义的个数。

可以传给函数任意个数的参数，多的忽略，遗漏的都会以 undefined 传递给函数。

8:Function 类

用 Function 类可以直接创建函数

语法

```
var function_name= new Function(argument1,arguments2,...argumentN,function_body);
```

在这种形式中，每个 argument 都是一个参数，最后一个参数是函数体，这些参数必须是字符串。

```
function doAdd(iNum){alert(iNum+100);}
doAdd=new Function("iNum","alert(iNum+10)");
```

doAdd 的值指向对象的指针, 函数名只是指向函数对象的引用值, 行为就像其他指针一样。

甚至可以使两个变量指向同一个函数。

```
var alsoDoAdd=doAdd;
```

如果函数名只是指向函数的变量，那么可以把函数作为参数传递给另一个函数？

例如：

```
function callAnotherFunc(fnFunciton, vArgument)
{
 fnFunction(vArgument);
}
var doAdd=new Function("iNum","alert(iNum+10)");
callAnotherFunc(doAdd,10);//20
```

【注意】尽管可用 Functiong 构造函数创建函数， 但因用其相对传统方式慢得多，所以最后别用。】

不过，所有函数都应该看作是 Function 类的实例。