javascript学习--2017-11-7：第八章：函数

笔记本：	javascript		
创建时间：	2017/11/7 11:05	**更新时间：**	2017/11/9 9:51
作者：	xiethan		

vscode所需要的插件://www.cnblogs.com/rocketRobin/p/6638849.html

1.函数的定义

函数的两种定义方式：函数语句+表达式

例子：

```
<script type="text/javascript">
console.log(factorial(3));
//1.函数声明语句，
function factorial(x){
if(x<=1) return 1;
return x*factorial(x-1);
}


//2.函数表达式:函数名是可选的
var square = function(x) {return x*x}
//这种相对于上面这种，可以包含名称，这在递归是很有用
var f=function fact(x){
if(x<=1) return 1;
else return x*fact(x-1);
}
//函数表达式也可以作为参数传递给其它函数
//data.sort(function(a,b){return a-b;});

//函数表达式有时定义后立即调用
var tensquared = (function(x) {return x*x}(10));
console.log(tensquared);
</script>
```

注意：

1.表达式的方式，函数名称是可选的

2.函数声明语句实际上是，声明了一个变量，，并把这个函数对象赋值给它。相对于，定义函数表达式是，并没有声明一个变量。

3.函数可以命名

4.如果一个函数定义表达式包含名称，函数的局部作用域将会包含一个，绑定到函数对象的名称，实际上，函数的名称将成为函数内部的一个局部变量。

5.通常而言，表达式定义函数时都不需要名称，这会让定义他们的代码更为紧凑，特别适合定义只用用到一次的函数

6.按照函数规则，函数声明这种方式“被提前”外部脚本、外部函数作用域的顶部，所以，这种定义方式，可以在它定义之前调用。

7.而表达式方式的，不一样，为了调用一个函数，必须把它赋值给一个变量，变量的声明提前了，但给变量赋值是不会提前的。所以在这个之前是无法使用的。

8.函数声明语句并非真正的语句，它可以出现在全局代码里，或者内嵌在其它函数，但是它不能出现在循环、条件判断，或者try/cache/finally以及with语句中。函数定义表达式可以出现在任何位置

```
console.log(factorial(3));
//1.函数声明语句
function factorial(x){
// if(x<=1) return 1;
// return x*factorial(x-1);
console.log("hello.javascript");
}

//函数表达式有时定义后立即调用
var tensquared = (function(x) {console.log("hello.javascript2");}(10));
console.log(tensquared);

结果为
                hello.javascript
demo1.html:6 undefined
demo1.html:27 hello.javascript2
```

```
demo1.html:28 undefined
```

2.函数的调用
有四种方式：

- 作为函数
- 作为方法
- 作为构造函数
- 通过它们的call()/apply()方法间接调用

方法调用：
1.，调用表达式，普通的函数调用：由一个或多个函数表达式组成，每个函数表达式，都是由一个函数对象和左圆括号，参数列表，右圆括号组成。
2.如果一个函数f和一个对象o，则可以使用下面的方式定义个m()的方法
o.m=f;
给对象o定义了个方法m()，调用它时就像这样：
o.m();//如果有两个参数，则使用o.m(x,y);
方法调用与函数调用，有个重要的区别：调用上下文。

```
//=========方法的调用
var calculator={
operand1:1,
operand2:1,
add:function()
{
this.result=this.operand1+this.operand2;
}
}
console.log(calculator.result);//undefined ,测试
console.log(calculator.add());//无返回值，undefined ,这个方法调用计算1+1的结果
console.log(calculator.result);//=》2
//方法调用可能包括更复杂的属性表达式
customer.surname.toUppCase();//调用cusomer.surname的方法
f().m();//在f()调用结束后，继续调用返回值的方法m()
```

方法与this关键字是面向对象编程的核心

//是否为严格模式

```
//定义并调用一个函数，来确定当前脚本运行时是否为严格模式
var strict=(function(){return !this;}());
console.log(strict);//=>false
```

如果嵌套函数作为方法调用，其this的值指向调用它的对象
如果嵌套函数作为函数调用，其this的值不是全局对象（非严格模式下），就是undefined（严格模式下）

```
//self保存this
var o={
m:function(){
var self=this;
console.log(this===o);//=>true
f();

function f(){
console.log(this===o);//false this的值不是全局对象（非严格模式下），就是undefined（严格模式下）
console.log(self===o);//true
}
}
}
o.m();
```

构造函数的调用
这凡是没有形参的构造函数，都可以省略圆括号

```
如：
var o= new object();
var o= new object;
```

间接调用：call()、apply()

函数的实参与形参：实际上，javascript函数调用，不检查形参的个数

1.当调用函数时的，传入的实参比函数声明时，指定的形参个数要少，剩下的形参都将设置为undefined
在函数内，可以使用：if（a=== undefined）a = []; a= a || [];
同时也可以使用，argument[0] 与argument[1] ……来对实参进行判断，也包含length
2.少传的的实参都是undefined，多出的参数都将自动省略
3.callee属性，比如在名函数中，通过callee类递归的调用自身
return x*arguments.callee(x-1);
//使用arge.from ---------不用记住参数的顺序P176

//作为值的函数
1.在javascript中，函数不仅是一种语法，也是值，可以将函数赋值给变量，还可以；对象的属性与数组的元素

```
//作为值的函数
var a=[function(x) {return x*x},20];
console.log(a[0](a[1]));
```

1.自定义函数属性
2.命名空间的函数

代码：

```
<!doctype html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script type="text/javascript">
console.log(factorial(3));
//=============函数的声明
//1.函数声明语句
function factorial(x){
if(x<=1) return 1;
return x*factorial(x-1);
}

//2.函数表达式:函数名是可选的
var square = function(x) {return x*x}
//这种相对于上面这种，可以包含名称，这在递归是很有用
var f=function fact(x){
if(x<=1) return 1;
else return x*fact(x-1);
}
//函数表达式也可以作为参数传递给其它函数
//data.sort(function(a,b){return a-b;});

//函数表达式有时定义后立即调用
var tensquared = (function(x) {return x*x}(10));
console.log(tensquared);

//==========2.javascript语言中，函数支持嵌套在其它函数里,如：
function hypotenuse(a,b)
{
function square(x){
return x*x;
}
return Math.sqrt(square(x)+square(b));
}

//========函数的调用
function printprops(o){
for(var p in o){
console.log(p+":"+o[p]+"\n");
```

```
}
}
printprops({x:1});//=>x:1
var total=factorial(3);

//==========方法的调用
var calculator={
operand1:1,
operand2:1,
add:function()
{
this.result=this.operand1+this.operand2;
}
}
console.log(calculator.result);//undefine ,测试
console.log(calculator.add());//无返回值，undefine ,这个方法调用计算1+1的结果
console.log(calculator.result);//=》2
//方法调用可能包括更复杂的属性表达式
// customer.surname.toUppCase();//调用cusomer.surname的方法
//f().m();//在f()调用结束后，继续调用返回值的方法m()

//定义并调用一个函数，来确定当前脚本运行时是否为严格模式
var strict=(function(){return !this;}());
console.log(strict);//=>false

//self保存this
var o={
m:function(){
var temp=1;
var self=this;
console.log(this===o);//=>true
f();

function f(){
console.log(this===o);//false this的值不是全局对象（非严格模式下），就是undefined（严格模式下）
console.log(self===o);//true
console.log(temp++);
}
}
}
o.m();

//作为值的函数
var a=[function(x) {return x*x},20];
console.log(a[0](a[1]));
console.log(Math.pow);
</script>
<title>无标题文档</title>
</head>
<body>
</body>
</html>
```