**函数**

-函数也是一个对象

除了5个基础数据类型 其他都是对象

网页里 所能看见的东西 都是对象

-把特定的代码片段，提取成为一个独立运行的实体，就是函数

-使用typeof检查一个函数对象时会返回function

**函数的好处**

代码复用

方便维护

方便理解、阅读

**创建函数**

创建一个函数对象

**构造函数**

可以将要封装的代码以字符串的形式传递给构造函数

var fun = new Function("a","b","return a+b");

封装到函数中的代码不会立即执行

函数中的代码会在函数调用的时候执行

我们在实际开发中很少使用构造函数来创建一个函数对象

**函数声明**

更多使用函数声明来创建一个函数

**语法：**

function 函数名([形参1,形参2,...形参N]){

语句...

}

**函数表达式**

使用函数表达式来创建一个匿名函数

**语法：**

var 函数名 = function([形参1,形参2...形参N]){

语句....

}

**调用函数**

**语法：函数对象();**

eg. fun();

当调用函数时，函数中封装的代码会按照顺序执行

**函数的参数**

形参：默认值的undefined，占位不赋值 ，是一个局部变量

**1.声明函数时，在（）中来指定一个或者多个形参（形式参数）**

对个形参之间使用逗号","隔开，声明形参就相当于在函数内部声明了相应的变量 但是并不赋值

**2.调用函数时，在()中指定实参（实际参数)**

实参将会赋值给函数中对应的形参

**3.调用函数时解析器不会检查实参的类型**

实参可以是任何值

所以要注意，是否有可能会接收到非法的参数，

如果有可能则需要对参数进行数据类型的检查

**4.调用函数时，解析器也不会检查实参的数量**

多余实参不会被赋值

如果实参的数量少于形参的数量，则没有对应实参的形参不会被赋值 值将是undefined

eg. 定义了形参但是调用的时候不传实参 则函数作用域内形参的值为undefined

**5.实参可以是一个对象**

当参数过多时，可以将参数封装到一个对象中，然后通过对象传递

**6.实参也可以是一个函数**

开发中经常用到将一个匿名函数当成实参传递给函数

fun(a)

-调用函数

-相当于使用的函数的返回值

fun

-函数对象

-相当于直接使用函数对象

**arguments:封装实参的对象**

——arguments是一个类数组的对象，也可以通过索引来操作数据，也可以获取长度

——在调用函数时，我们所传递的实参都会在arguments中保存

——使用arguments.length来获取参数的个数

——我们即使不定义形参，也可以通过arguments来使用实参，

只不过比较麻烦

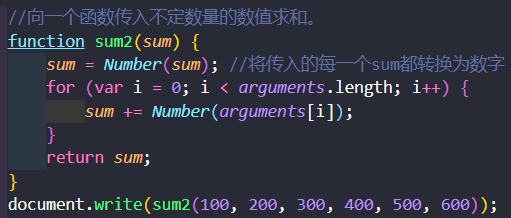
arguments[0]表示第一个实参

arguments[1]表示第二个实参。。。

——它里边有—个属性叫做callee,

这个属性对应一个函数对象,就是当前正在指向的函数的对象

JS中函数不介意传递进来多少个参数，也不在乎传进来参数是什么数据类型，在调用函数时也未必一定要传递指定数量的参数，原因是 ECMAScript 中的参数在内部是用一个数组(arguments)来表示的。函数接收到的始终都是这个数组，而不关心数组中包含哪些参数（如果有参数的话）。



**函数的返回值**

**1.可以使用return来设置函数的返回值**

**语法：**

return 值;

**2.return后的值将作为函数的执行结果返回**

可以定义一个变量接收函数的执行结果

**3.在函数中return后的语句都不会执行，交回函数的执行权**

break可以退出当前的循环

continue用于跳过当次循环

return可以结束整个函数

**返回值的类型**

返回值可以是任意的数据类型，也可以是一个对象，也可以是一个函数

fun(a)

-返回函数的返回值

-相当于返回使用的函数的返回值

fun

-返回函数对象

-相当于直接返回函数对象

**匿名函数：立即执行函数**

函数定义完，立即被调用，这种函数叫做立即执行函数

立即执行的函数往往只会执行一次

语法：(function（a,b）{

alert(""我是一个匿名函数~~~" );

})（a,b） 最后的括号加上以后整个就等于调用；

**作用域（Scope）**

作用域指一个变量的作用的范围

**局部变量:**

定义在函数内部的变量，这个变量只能在函数内部使用，即作用域范围只是函数内部，另外，形参也是局部变量.

作用范围是某个函数体内部

在函数体内部通过var关键字定义的变量或者形参，都是局部变量

当局部变量与全局变量重名时，在函数体内部局部变量优先于全局变量

**全局变量:**

全局变量就是定义在函数外部的变量，这个变量在任何函数中都有效，即作用域范围是当前文件的任何地方.

作用范围为整个程序的执行范围

在函数体外部定义的变量就是全局变量

在函数体内部不使用var定义的就是全局变量

**js中两种作用域**

**1.全局作用域**

直接编写在script便签中的代码都在全局作用域

全局作用域在页面打开时创建，在页面关闭时销毁

在全局作用域中，有一个全局对象window，

它代表着浏览器的窗口，由浏览器创建，可以直接使用

在全局作用域中，

创建的变量都会作为window对象的属性保存

创建的函数都会作为window对象的方法保存

全局作用域中的变量都是全局变量，在页面的任意部分都能访问的到

**2.函数作用域**

调用函数时创建函数作用域，函数执行完毕以后，函数 作用域销毁

每调用一次函数就会创建一个新的函数作用域，他们之间是互相独立的

在函数作用域中可以访问到全局作用域的变量

在全局作用域中无法访问到函数作用域的变量

当在函数作用域操作一个变量时，它会先在自身作用域中寻找，如果有就直接使用
如果没有则向上一级作用域中寻找（就近原则）

**什么是作用域链：**

查找变量的过程中，局部变量优先，再找全局变量的过程

在函数作用域也有声明提前的特性：

使用var关健字声明的变量，会在函数中所有的代码执行之前被声明

在定义变量时, 如果不写关键字var也是合法的, 且是全局变量，但是这样写不安全，容易在其他地方被更改, 所以我们在函数中写变量要加上var

函数声明也会在函数中所有的代码执行之前执行

在函数中不使用var声明的变量都会设置为全局变量

定义形参就相当于在函数作用域中声明了变量

**变量的声明提前**

——使用var关键字声明的变量，会在所有的代码执行之前被声明（但是不会赋值）

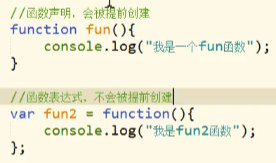
但是如果声明变量时不使用var关键字，则变量不会被声明提前

**函数的声明提前**

——使用函数声明形式创建的函数function函数(){}

它会在所有的代码执行之前就被创建，所以我们可以在函数声明前来调用函数

使用函数表达式声明的函数不会被提前



**函数的递归调用**

函数自己调用自己，必须要有临界条件，称为函数的递归调用

**递归调用的方式:**

1. 首先去找临界值，即无需计算，获得的值(一般是返回该值)。

2. 找这一次和上一次的关系(一般从后往前找)

3. 假设当前函数已经可以使用，调用自身计算上一次的运行结果，再写出这次的运行结果。

**递归的使用场景：**

多层二级菜单

文件的磁盘遍历

对象的深度拷贝

循环能做的事情，递归都能实现，但是递归有性能问题

**this**

解析器在调用函数每次都会向函数内部传递进一个隐含的参数

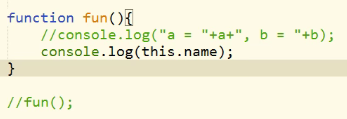
这个隐含的参数就是this. this指向的是一个对象

这个对象我们成为函数执行的 上下文对象，

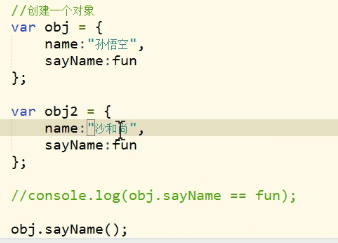
**根据函数调用方式的不同，this会指向不同的对象**

1.以函数的形式调用，this就是window

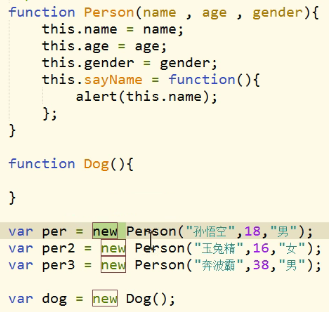
因为window也是一个对象 调用函数可以看作调用window的方法



2.以方法的形式调用，this就是调用方法的对象



3.当以构造函数的形式调用时， this就是新创建的那个对象



**函数的方法**

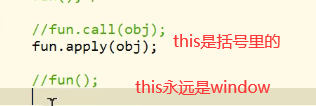
**call（）和apply（）**

——这两个方法都是函数对象的方法，需要通过函数对象来调用

——当对函数调用call（）和apply（）都会调用函数形式执行

——在调用call（）和apply（）可以将一个对象指定为第一个参数

此时这个对象将会成为函数执行时的this



——call()方法可以将实参在对象之后依次传递

——apply()方法需要将实参封装到一个数组中统—传递

截图.png

**this的情况:**

1.以函数形式调用时this永远都是window

2.以方法的形式调用时, this是调用方法的对象

3.以构造函数的形式调用时，this是新创建的那个对象

4.使用call和apply调用时， this是指定的那个对象