# 对象

JS中一共有6种数据类型

\* 基本数据类型

\* String Number Boolean Null Undefined

\* 引用数据类型

\* Object（对象）

基本数据类型保存的都是一个一个单一的值，值与值之间没有任何的联系，不能表示一些复杂的数据，而对象是一种复合数据类型，对象就像是一个容器，在它里面可以装任意数据类型的属性

## 对象的分类

1.内建对象

\* - 内建对象指由ES标准定义的对象，任何对ES标准的实现都可以使用这些对象

\* - 比如：Math String Number Boolean RegExp

2.宿主对象

\* - 宿主对象指由JS运行环境为我们提供的对象，目前对于我们来说就是指浏览器为我们提供的对象

\* - 比如BOM对象 和 DOM对象

\* - window document console 。。。。。。

3.自定义对象

\* - 由我们自己定义的对象

## 属性的基本操作

创建对象

调用构造函数（constructor）创建一个对象

使用new关键字调用的函数称为构造函数

var obj = new Object();

使用typeof检查一个对象时，会返回object

在对象中可以保存不同类型值，这些值我们称为对象的属性

向对象中添加属性

语法：对象.属性名 = 属性值

obj.name = "孙悟空";

obj.age = 18;

obj.gender = "男";

从对象中读取属性

语法：对象.属性名

console.log(obj.gender);

如果读取对象中没有的属性，不会报错，而是返回undefined

修改对象的属性

语法：对象.属性名 = 新值

obj.name = "tom";

删除对象中的属性

语法：delete 对象.属性名

delete obj.name;

检查对象中是否含有某个属性，可以使用in运算符

语法 "属性名" in 对象

检查对象中是否含有指定的属性，如果有则返回true，没有则返回false

console.log("name" in obj); //false

对象的属性名，没有任何要求，也不用遵循的标识符的规范，可以起任何的稀奇古怪的名字，但是如果是一些特别的名字不能使用 对象.属性名 的形式添加，而需要采取其他的方式。

虽然可以起任意的名字，但是在开发中属性名尽量还是遵循标识符的规范

例如obj.123=”你好” 就会报错，不能使用对象.属性名的形式添加

添加属性的第二种方式

对象["属性名"] = 属性值

[]这种方式比我们使用 . 更加灵活，而且[]中可以传变量

var str = "name";

console.log(obj[str]);

读取属性：

对象["属性名"]

属性值

\* - 属性值可以是任意的数据类型（String Number Boolean Null Undefined）

\* - 也可以一个对象

var obj2 = new Object();

obj2.name = "沙和尚";

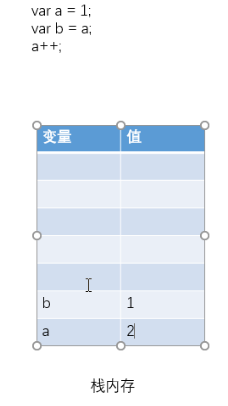
obj.test = obj2;

console.log(obj.test.name);

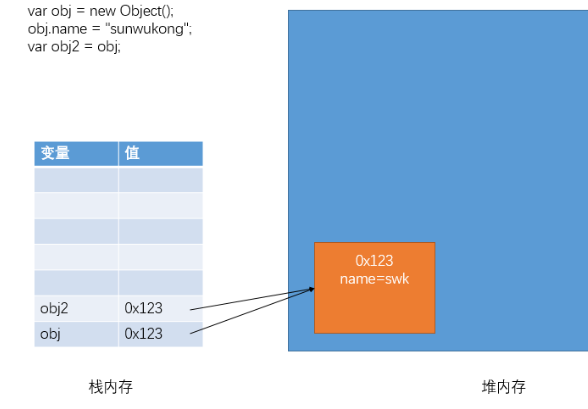
## 基本和引用类型的区别

程序在运行时，内存分为两种结构栈内存、堆内存

基本数据类型的值是直接保存到栈内存中的，基本数据类型的值都是互相独立的，修改一个不会影响到其他的值



引用数据类型对象是保存在堆内存中的，而在变量中保存的是对象的引用（内存地址），我们是通过引用来操作对象的，当修改一个对象时有可能影响到其他的变量



对基本数据类型进行相等比较时，比较的是数据的值

而两个引用数据类型进行相等比较时，比较的是对象的地址，如果两个对象一模一样，但是地址不同也会返回false

var obj5 = new Object();

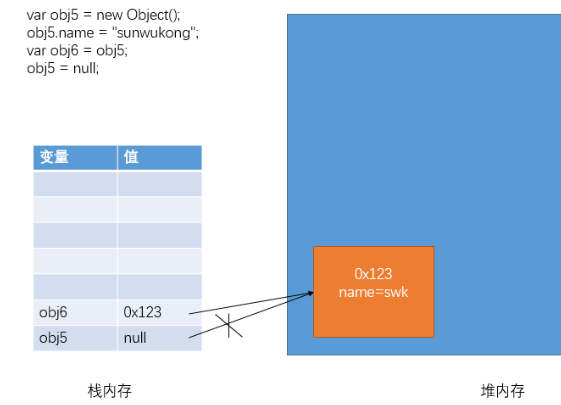
obj5.name = "孙悟空";

var obj6 = obj5;

//设置obj5为null

obj5 = null;

console.log(obj6.name); //孙悟空



## 对象字面量

使用对象字面量来创建对象

var obj = {};

obj.name = "孙悟空";

创建对象的同时为其添加属性

对象字面量的语法：

{属性名:属性值,属性名:属性值,属性名:属性值...}

对于特殊的属性名，可以通过双引号引起来

var obj2 = {

name:"孙悟空",

age:18,

address:"花果山",

"123":"hello",

test:{

name:"猪八戒",

age:28,

address:"高老庄"

}

};

## 属性的枚举

使用 for...in 语句对一个对象中的属性进行枚举

\* 语法：

\* for(变量 in 对象){

\* 语句

\* }

\* 对象中有几个属性，循环体就会执行几次，每次执行时都会将一个对象中的属性名赋值给变量

//创建一个对象

var obj = {

name:"猪八戒",

age:28,

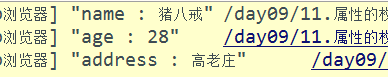
address:"高老庄"

};

for(var n in obj){

console.log(n + " : " +obj[n]);

}



# 垃圾回收

垃圾回收（GC）

当一个对象没有任何的变量对其引用时，此时我们将无法在获取到该对象，也无法在对该对象做任何的操作，但是这种对象依然会占用内存空间，这种垃圾过多以后会浪费大量的内存，导致程序运行变慢。像这种对象我们必须进行回收，使其不占用内存空间。

在JS中有自动的垃圾回收机制，浏览器会自动将没有被引用的对象清除出内存。回收的工作由浏览器自动完成，而我们只需要将一些不再使用的对象设置为null即可

# 函数（function）

\* - 函数也是一个对象，具有Object的所有的功能

\* - 不同是的函数对象可以保存一些可执行的代码，并且在需要的时候可以调用这些代码

创建一个函数对象

创建函数对象时，可以在构造函数中传递一个字符串作为参数，这个字符中可以保存一些js代码

var fun = new Function("alert('我是函数中的代码');");

使用typeof检查一个函数对象时，会返回function，除了函数，其余的对象返回都是object

如果需要函数中的代码执行，则需要对函数进行调用

语法：函数对象+()

fun();

创建函数的方式一：

\* - 使用函数声明来创建一个函数

\* - 语法：

\* function 函数的名字([形参1,形参2,...形参n]){

\* 语句...

\* }

中括号[]表示可选的，写不写都行

创建函数的方式二：

\* - 使用函数表达式来创建一个函数

\* - 语法:

\* var 变量 = function([形参1,形参2,...形参n]){

\* 语句...

\* };

在定义函数时，可以在()中来指定数量不等的形参(形式参数)

\* 多个形参之间使用,隔开

\* 在函数中定义几个形参，就相当于在函数中声明了几个变量，但是并不赋值

在调用函数时，可以指定实参（实际参数），

\* 和形参类似，我们也可以传递多个实参，多个实参之间使用,隔开

\* 实参会赋值给对应的形参，第一个实参赋值个第一个形参 第二个实参赋值给第二个形参。。。

JS函数在调用时，解析器（浏览器）不会去检查实参的类型和个数

\* - 也可以传递任意数量的实参

\* - 如果实参的数量小于形参，则没有对应实参的形参将会是undefined

\* - 如果实参的数量大于形参，则多余的实参不会使用

\* - 可以传递任意类型的实参

\* - 实参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

## 返回值

- 返回值简单来说就是函数的执行结果

- 我们可以通过return来设置函数的返回值

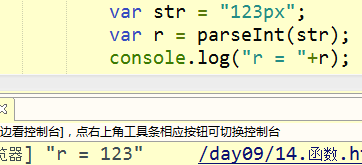
- 语法：return 值;

return后的代码将不会执行

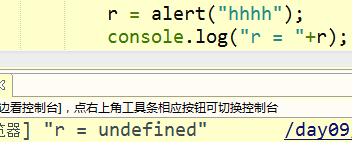
- 函数的返回值可以是任意的数据类型

也可以是一个对象，也可以是一个函数

- 如果return后不写值或者不写return，则相当于 return undefined;



123就是parseInt的返回值



alert没有返回值

# break、continue和return

