**对象的简介**

对（现实）具象事物的抽象

**为什么要定义对象：**

面向对象编程，更加符合现实，所以要定义对象

**五个基础数据类型：**

-Number数字

-String字符串

-Boolean布尔值

-Undefined未定义

-Null空值

只要不是上边的5种，全都是对象

-Object 对象

基本数据类型都是单一的值 "hello” 123 true,

值和值之间没有任何的联系。

在js中来表示一个人的信息. (name gender age) ,

var name ="孙悟空"";

var gender ="男";

var age = 18;

如果使用基本数据类型的数据，我们所创建的变量都是独立，不能成为一个整体。

对象属于一种复合的数据类型，在对象中可以保存多个不同数据类型的属性。

**对象的分类**

**1.内置对象**

js中的内部对象包括Array、Boolean、Date、Function、Global、Math、Number、Object、>RegExp、String以及各种错误类对象，包括Error、EvalError、RangeError、ReferenceError、>SyntaxError和TypeError。

其中Global和Math这两个对象又被称为“内置对象”，这两个对象在脚本程序初始化时被创建，不必实例化这两个对象。

**2.宿主对象**

宿主对象就是执行JS脚本的环境提供的对象。对于嵌入到网页中的JS来说，其宿主对象就是浏览器 提供的对象，所以又称为浏览器对象，如IE、Firefox等浏览器提供的对象。不同的浏览器提供的 宿主对象可能不同，即使提供的对象相同，其实现方式也大相径庭！这会带来浏览器兼容问题， 增加开发难度。浏览器对象有很多，如Window和Document等等。

**3.自定义对象**

顾名思义，就是开发人员自己定义的对象。JS允许使用自定义对象，使JS应用及功能得到扩充

**对象的基本操作**

**创建对象**

使用new关键字调用构造函数constructor

构造函数是专门用来创建对象的函数

**语法：**

var obj = new Object();

**向对象添加属性**

在对象中保存的值成为属性

**语法：**

对象.属性名= 属性值;

**读取对象中的属性**

**语法：**

对象.属性名

如果读取对象中没有的属性，不会报错，而是会返回undefined

**修改对象的属性值**

**语法：**

对象.属性值 = 新值

**删除对象的属性**

不会触及原型链中的任何属性

但有可能让来自原型链中的属性暴露出来

**语法：**

delete dog.name;

delete(dog.name);

**对象的属性名和属性值**

**属性名**

-对象的属性名不强制要求遵守标识符的规范

什么乱七八糟的名字都可以使用

-但是还是尽量按照标识符的规范去命名

如果要使用特殊的属性名，不能采用.的方式来操作

需要使用另一种方式, 用中括号操作这个属性

语法：对象["属性名"] = 属性值

读取时也需要采用这种方式

使用 [ ] 这种形式去操作属性，更加的灵活，

在 [ ] 中可以直按传递一个变量，这样变量值是多少就会读取那个属性

**属性值**

js对象的属性值，可以是任意的数据类型，什么都行

甚至也可以是一个对象，还可以是一个函数

in运算符

通过该运算符可以检查一个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

语法：

"属性名" in 对象

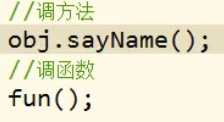
eg. console.log("name" in obj);

函数也可以称为对象的属性,

如果一个函数作为一个对象的属性保存，

那么我们称这个函数是这个对象的方法

调用函数就说调用对象的方法



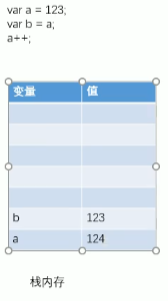
**基本数据类型和引用数据类型**

**基本数据类型**

**String、Number、Boolean、Null、Undefined**

所有变量都保存在栈内存中

基本数据类型的值直接在栈内存中存储



值与值之间相互独立 修改一个变量不会影响其他变量

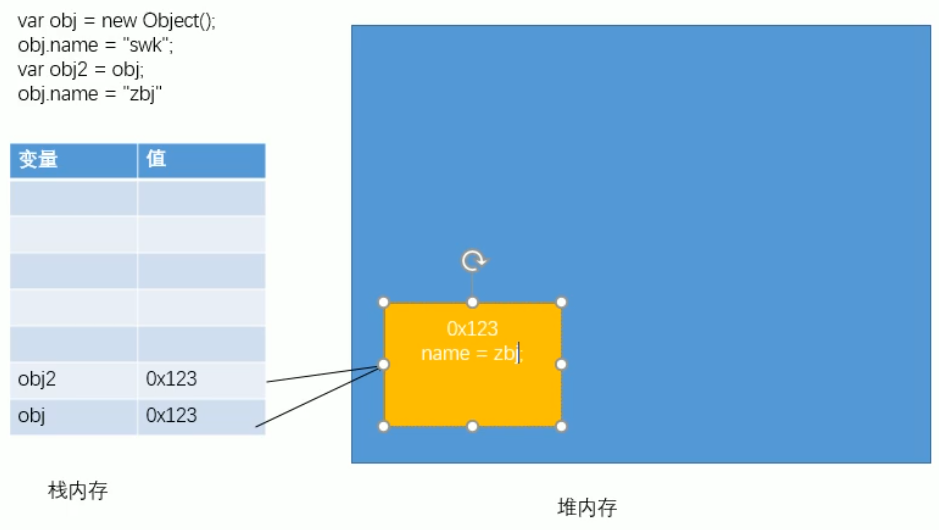
**引用数据类型**

**Object 、Function**

对象是保存到堆内存中的。

每创建（new）一个新的对象，就会在堆内存中开辟出一个新的空间。

变量保存的是对象的内存地址（对象的引用），如果两个变量保存的是同一个对象引用，当一个通过一个变量修改属性时，另一个也会受到影响



当比较两个基本数据类型的值时，就是比较值。

而比较两个引用数据类型时，它是比较的对象的内存地址，

如果两个对象是一模一样的，但是地址不同，它也会返回false

**对象字面量**

使用对象字面量来创建字面量

**语法：**

var obj = { };

用大括号和用new Object（）创建是一样的

**优点： 指定对象中的属性**

使用对象字面量，可以在创建对象时，直接 指定对象中的属性

**语法：**

var obj = {

属性名1∶ 属性值1，

属性名2∶ 属性值2，

属性名3∶ 属性值3

}

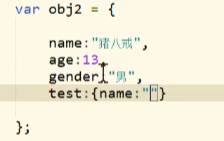
1、对象字面量的属性名可以加引号也可以不加,建议不加

如果要使用一些特殊的名字，则必须加引号

2、属性名和属性值是一组一组的名值对结构，

名和值之间使用冒号":"连接，多个名-值对之间使用逗号","隔开

如果一个属性之后没有其他的属性了，就不要写逗号","



**枚举对象中的属性**

**使用for...in语句**

**语法：**

for(var 变量n in 对象obj) {

console.log("属性名∶ " + n);

console.log("属性值:" + obj[n]);

}

**结果：**



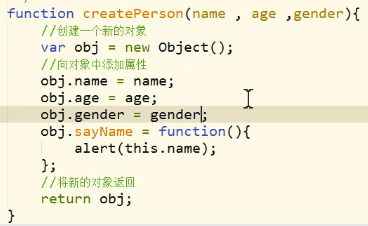
对象中有几个属性，循环体就会执行几次

每次执行时，都会将对象中的一个属性的名字赋值给变量

**使用工厂方法创建对象**

通过该方法可以大批量的创建对象

**模板**



**使用**

截图.png

使用工厂方法创建的对象，使用的构造函数都是Object

所以创建的对象都是Object这个类型，

就导致我们无法区分出多种不同类型的对象

**用构造函数创建某一类对象**

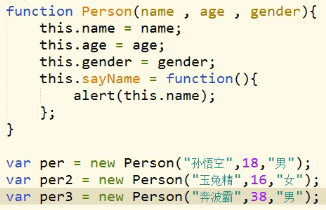
1.构造函数，专门用来创建Person对象

2.构造函数就是一个普通的函数，创建方式和昔通函数没有区别,

不同的是构造函数习惯上首字母大写

3.构造函数和普通函数的区别就是调用方式的不用

普通函数是直接调用，而构造函数需要使用new关键字来调用



**构造函数的执行流程**

**1.立刻创建于一个新的对象**

**2.将新建的对象设置为函数中的this，在构造函数中可以使用this来引用新建的对象**

——理解为在对象方法中的this 指向当前的对象

**3.逐行执行函数中的代码**

**4.将新的对象作为返回值返回**

使用同一个构造函数创建的对象，我们称为一类对象

我们将通过一个构造函数创建的对象，称为是该类的实例

**instanceof**

使用instanceof可以检查一个对象是否是一个类的实例

如果是,则返回true，否则返回false

语法：

对象 instanceof 类名/构造函数名

eg.

var per = new Person("罗小龙",18,"男");

console.log(per instanceof Person);

ps.

所有的对象都是Object的后代，

所以任何对象和Object做instanceof检查时都会返回true

创建一个Person构造函数

-在Person构造函数中，为每一个对象都添加了一个sayName方法，

目前我们的方法是在构造函数内部创建的，

也就是构造函数每执行一次就会创建一个新的sayName方法

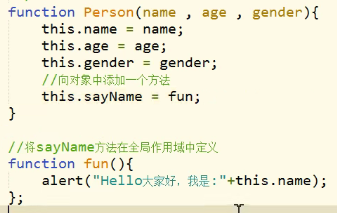
也是所有实例的sayName都是唯一的。

这样就导致了构造函数执行一次就会创建一个新的方法,

执行10000次就会创建10000个新的方法，而10000个方法都是一模一样的

这是完全没有必要,完全可以使所有的对象共享同一个方法

将相同的方法提取到全局中定义，自始至终只有一个



但是将函数定义在全局作用域，污染了全局作用域的命名空间

而且定义在全局作用域中也很不安全

所以尽量不要在全局作用域里声明函数↓

**toString（）**

直接在页面中打印一个对象时，实际上是输出的对象的toString()方法的返回值

如果我们希望在输出对象时不输出[object object]，可以为对象添加一个toString()方法

**//修改Person原型的toString**

Person.prototype.tostring = function( ){

return "Person[name="+this.name+ ",age="+this.age+" ,gender:"+this.gender+"]";

};

输出：Person[ name=孙悟空, age=18,gender=男]

**垃圾回收（GC)**

——就像人生活的时间长了会产生垃圾一样，程序运行过程中也会产生垃圾

这些垃圾积攒过多以后，会导致程序运行的速度过慢,

所以我们需要一个垃圾回收的机制,来处理程序运行过程中产生垃圾

——当一个对象没有任何的变量或属性对它进行引用，此时我们将永远无法操作该对象，

此时这种对象就是一个垃圾，这种对象过多会占用大量的内存空间，导致程序运行变慢

——在JS中拥有自动的垃圾回收机制，会自动将这些垃圾对象从内存中销毁，
我们不需要也不能进行垃圾回收的操作

**包装类：在底层给浏览器自己调用的**

**基本数据类型**

String Number Boolean Null Undefined

**引用数据类型**

object

在]S中为我们提供了三个包装类，通过这三个包装类可以将基本数据类型的数据转换为对象

**String()**

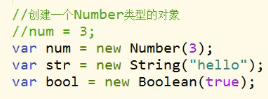
-可以将基本数据类型字符串转换为String对象

**Number()**

-可以将基本数据类型的数字转换为Number对象

**Boolean()**

-可以将基本数据类型的布尔值转换为Boolean对象



此时可以添加属性

但是注意，我们在实际应用中不会使用基本数据类型的对象，

如果使用基本数据类型的对象，在做一些比较时可能会带来一些不可预期的结果



方法和属性之能添加给对象，不能添加给基本数据类型

当我们对一些基本数据类型的值去调用属性和方法时，

浏览器会临时使用包装类将其转换为对象，然后在调用对象的属性和方法

调用完以后，再将其转换为基本数据类型

