# JavaScript进阶

第4天课堂笔记

# 目录

[JavaScript进阶 1](#_Toc25229)

[目录 2](#_Toc26957)

[一、object类型 3](#_Toc18719)

[1.1 对象的创建 3](#_Toc21541)

[1.1.1 字面量的创建方式 3](#_Toc24469)

[1.1.2 new Object()创建对象 5](#_Toc29680)

[1.2 对象的属性值 6](#_Toc7753)

[二、对象的方法 7](#_Toc1708)

[三、构造函数 8](#_Toc32300)

[四、原型prototype 11](#_Toc4322)

[4.1 原型的定义 11](#_Toc2962)

[4.2 原型的用途 13](#_Toc26557)

[4.3 JavaScript原型链机制 14](#_Toc119)

[五、面向对象的思维方式 17](#_Toc14532)

# 一、object类型

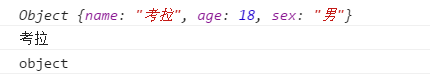
对象类型。

## 1.1 对象的创建

创建一个对象，有两种方法，第一种比较简单，叫做字面量；第二种是用new Object()。

### 1.1.1 字面量的创建方式

|  |
| --- |
| 1. var obj = { 2. name : "考拉", 3. age : 18, 4. sex : "男" 5. }; 6. console.log(obj); 7. console.log(obj.name); 8. console.log(typeof obj); |



{}就是对象的界定符，就是对象的字面量。对象有属性，所谓的属性就是这个对象的特点、特性，name、age、sex都是这个obj对象的属性（preperty）。

**哲学上讲，什么是对象？对象就是属性的无序集合。**

我们可以用.点语法、方括号法来获得一个对象的属性。那么你会发现，和数组有点相似，只不过数组的下标只能是数字0、1、2……，而我们的对象，可以用任何的词儿来当做属性名。

公式：

|  |
| --- |
| 1. { 2. k : v, 3. k : v, 4. k : v, 5. k : v 6. } |

JSON和对象字面量的区别：

之前学习过JSON，JSON要求所有的k必须加引号，而对象字面量不需要加引号，当然加引号也不错。

JSON = JavaScript Object Notation，JS对象表示法。JSON是一个用于交换的格式，所以JSON不仅仅JavaScript用，后台语言比如PHP、Java、ASP等等都要识别JSON，为了最大的兼容，k必须加引号。也就是说，JSON里面的k加引号，不是因为JS，而是因为后台的那些语言。

JSON要比对象字面量，要严格，严格在哪儿呢？就是所有的k，必须加引号。

JSON：

|  |
| --- |
| 1. { 2. "k" : v, 3. "k" : v, 4. "k" : v, 5. "k" : v 6. } |

但是，下面的特殊情况，这个k必须加引号：

k是特殊字符

k是数字

k是有空格

k是关键字、保留字

先说一下，上面这些情况，也同时不能使用点语法来访问属性了，必须使用方括号：

|  |
| --- |
| 1. var obj = { 2. name : "考拉" 3. } 4. console.log(obj["name"]); |

特殊形式的k，必须要加上引号，检索属性的时候，必须用方括号：

|  |
| --- |
| 1. var obj = { 2. **"**24&\*$&#@@)@!**"** : "哈哈", 3. **"**all name**"** : "考拉", 4. **"**++++%%%%**"** : "嘻嘻", 5. **"**var**"** : "么么哒", 6. **"**function**"** : "嘻嘻" 7. } 8. console.log(obj**["24&\*$&#@@)@!"]**); 9. console.log(obj**["all name"]**); 10. console.log(obj**["++++%%%%"]**); 11. console.log(obj**["var"]**); 12. console.log(obj**["function"]**); |

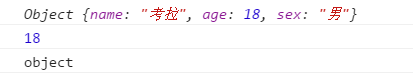
你会发现，JS会一个对象的属性名，没有特殊的规定。这是因为属性名不是标识符，没有那些规定。

对象的属性的访问，点语法是有局限的，它不能访问上面的特殊的那些情况。也不能访问以变量保存的k：

|  |
| --- |
| 1. var obj = { 2. name : "考拉", 3. age : 18, 4. sex : "男", 5. "study score" : 100 6. } 7. //console.log(obj."study score");//错误的 8. console.log(obj["study score"]); //正确的 9. console.log(obj["a" + "ge"]); //正确的 10. var a = "sex"; 11. console.log(obj[a]); //正确的 12. </script> |

### 1.1.2 new Object()创建对象

|  |
| --- |
| 1. var obj = new Object(); //这是一个空对象，里面没有任何属性 2. obj.name = "考拉"; 3. obj.age = 18; 4. obj.sex = "男"; 5. console.log(obj); 6. console.log(obj.age); 7. console.log(typeof obj); |



new是一个运算符，你没有看错，和+-\*/一样是一个运算符。表示新创建一个对象。一会儿我们学习构造函数，实际上你将了解到new是一个函数调用的方式。Object()大写字母O，这是一个系统内置的构造函数，什么是构造函数，我们稍后讲到。

下面就可以用obj.k = v ;来追加属性了：

|  |
| --- |
| 1. obj.name = "考拉"; |

这是一条语句，如同：

|  |
| --- |
| 1. oDiv.className = "current"; |

事实上，工程师更喜欢用字面量的方式来创建对象。因为更直观：

字面量方式：

|  |
| --- |
| 1. var obj = { 2. name : "考拉", 3. age : 18, 4. sex : "男" 5. }; |

构造函数方式：

|  |
| --- |
| 1. var obj = new Object(); 2. obj.name = "考拉"; 3. obj.age = 18; 4. obj.sex = "男"; |

上面两种方式创建出来的对象，是相同的。字面量的方式直观、简单、并且有“封装”的感觉。所以我们鼓励大家用字面量来创建对象。

但是，不要杂糅：

|  |
| --- |
| 1. var obj = {}; //的确能创建一个空对象 2. obj.name = "考拉"; //追加属性 3. obj.age = 18; //追加属性 4. obj.sex = "男"; //追加属性 |

## 1.2 对象的属性值

对象属性值，可以是任何东西。比如数字、字符串、布尔值、正则表达式、对象、数组、函数……

|  |
| --- |
| 1. var kaola = { 2. name : "考拉", 3. age : 18, 4. peiou : { 5. name : "母考拉", 6. age : 19 7. } 8. } 9. console.log(kaola.peiou.age); |

|  |
| --- |
| 1. var obj = { 2. a : 1**,** 3. b : "哈哈"**,** 4. c : true**,** 5. d : /[A-Z]/g**,** 6. e : function(){ 7. alert(1+2); 8. }**,** 9. f : { 10. p : 1 11. } 12. } |

特别的，当对象的属性的值是一个函数的时候，我们称这个函数是对象的方法。

# 二、对象的方法

当一个对象的属性的值，是一个函数，那么这个函数我们就称为对象的“方法”(method)。

方法就是一个对象能够做的事情，一般来说，就是一个动词。比如小明打招呼、长大、变性方法。

|  |
| --- |
| 1. var xiaoming = { 2. name : "小明", 3. age : 18, 4. sex : "男", 5. sayHello : function(){ 6. alert("你好，我是" + this.name); 7. alert("今年" + this.age + "岁了"); 8. alert("我是可爱的小" + this.sex + "生"); 9. } 10. } 11. xiaoming.sayHello(); |

比如上面的案例，sayHello就是一个属性，只不过它的值是一个函数，所以我们就可以说xiaoming这个对象，有sayHello方法。

一个对象，方法函数里面的this指的是这个对象。

那我们复习一下现在，我们调用函数的方式有哪些？这些调用函数的方式，里面的this又是谁？

● 直接用()运算符来调用函数，那么函数里面的this指的是window对象

● 函数如果绑定给了某个HTML元素的事件上，那么函数里面的this就是这个HTML对象

● 用定时器调用函数，函数内部的this就是window对象

● 用对象打点（方法）来调用函数，函数里面的this指的是这个对象

● 用apply、call，可以人工设置this是谁了，百变。

说白了，我们学习“方法”，无非就是学习了一种函数的调用方式。这个函数里面的this指的是这个对象。

函数里面的this到底是谁，在函数定义的时候并不知道，要看函数如何被调用。

对象的方法的哲学，就是操作自己的属性。如果一个对象的方法，不操作自己的属性，那干嘛还要是方法呢？

zhangda方法，就是让自己的age++：

|  |
| --- |
| 1. var xiaoming = { 2. name : "小明", 3. age : 18, 4. sex : "男", 5. sayHello : function(){ 6. alert("你好，我是" + this.name); 7. alert("今年" + this.age + "岁了"); 8. alert("我是可爱的小" + this.sex + "生"); 9. }, 10. zhangda : function(){ 11. this.age++; 12. } 13. } |

# 三、构造函数

JavaScript规定，一个函数可以用new关键字来调用。那么此时将按顺序发生四件事情：

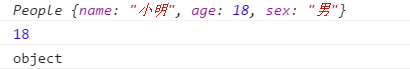
1）隐秘的创建一个新的空对象

2）将这个函数里面的this绑定到刚才创建隐秘新对象上

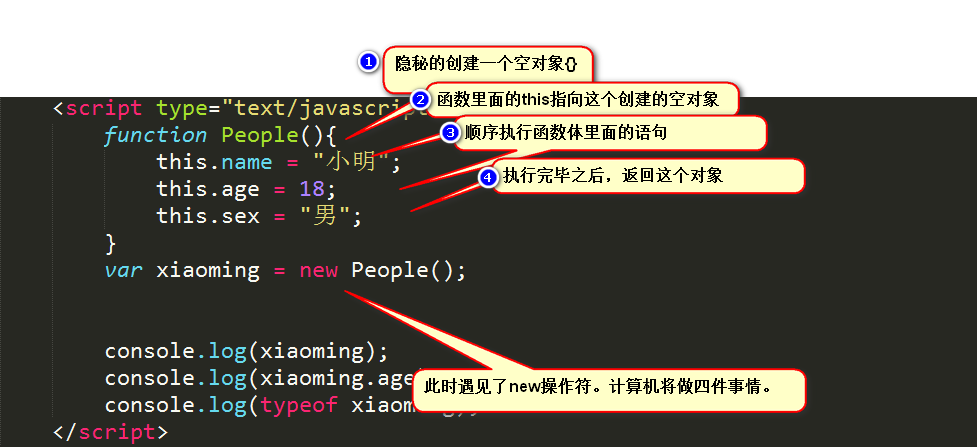
3）执行函数体里面的语句

4）返回这个新的对象

|  |
| --- |
| 1. function People(){ 2. this.name = "小明"; 3. this.age = 18; 4. this.sex = "男"; 5. } 6. var xiaoming = **new** People(); 7. console.log(xiaoming); 8. console.log(xiaoming.age); 9. console.log(typeof xiaoming); |



上面的函数的机理：



好了，也就是说，我们现在学习了函数的新的调用方式：使用new关键字来调用。

此时很有意思，函数不仅仅能够执行，还能返回出来一个对象。也就是说，对象是函数“生”出来的，对象是函数“new”出来的。

我们称呼这个函数，叫做构造函数，一般的，构造函数用大写字母开头。也叫作People“类”。

我们称呼xiaoming这个对象是People类的实例。

真实社会，真实自然界，万物分为各种各样的“种类”，比如有“人类”、“猪类”、“鸟类”、“计算机类”。为什么能分出类别呢？哲学上讲，是因为他们有不同的属性群。换句话说，有相同属性群的东西，就可以称为一“类”。

所以我们的构造函数，也可以看成类的定义：

|  |
| --- |
| 1. //类，People类 2. function People(name,age,sex){ 3. this.name = name; 4. this.age = age; 5. this.sex = sex; 6. } 7. //new关键字造出来的xiaoming、xiaohong，我们称为People类的实例。 8. var xiaoming = new People("小明",18,"男"); 9. var xiaohong = new People("小红",17,"女"); 10. console.log(xiaoming); 11. console.log(xiaohong); |

People这个函数一会儿将用new关键字来调用，所以称为构造函数。

构造函数里面的语句将执行，并且将返回一个对象。所以new多少次，里面的语句就会执行多少次。**宏观的看，返回的对象，就都有name属性、age属性、sex属性了**。我们说，**这些返回的对象都有相同的属性群了，所以可以看做是一类东西**。那么People这个构造函数，**也可以看成类的定义**。

但是JavaScript中，没有类的概念。是通过构造函数的4步走机制来创建类似的对象，可以看为类。JS这个语言是“基于对象”(base Object)的语言，不能叫做“面向对象”(orinted object)的语言。

注意，我们学习Java、C++、C#语言的惯例，构造函数以大写字母开头。

|  |
| --- |
| 1. function People(){ 2. } |

哦，这是一个People类，这个函数一会儿要用new来调用。能返回对象。

类就是蓝图，上帝要根据这个蓝图造小人儿。造出的小人，就是People类的实例。

构造函数中，肯定会出现this语句，如果没有this语句，那就相当于不能给创建出来的空对象绑定属性。

构造函数，在JS中，就是一个普通的函数，JS没有对里面要书写什么进行任何规定。只不过我们工程师习惯的，先用this.\*\*\* = \*\*\*来罗列所有属性，然后罗列所有的方法而已。你一定要深刻理解：

**new一个函数的时候，函数里面的语句会执行。**

我们刚才学啥了？学习了一个新的函数调用的方式。

当一个函数用()调用的时候，this就是window

当一个函数用对象方法调用的时候，this就是这个对象

当一个函数绑定给一个HTML元素事件的时候，this就是这个HTML元素

当一个函数用定时器调用的时候，this就是window

当一个函数用apply、call调用的时候，this就是你指定这个东西

当一个函数用new调用的时候，this就是隐秘创建的空对象，函数里面的语句将被执行，并且返回新对象

宏观的看，new出来的东西都有相同的属性群、方法群。此时我们就发现了：哇塞这个构造函数像模板一样，可以非常快速的制作类似的实例。这个构造函数可以称为类，它new出来的东西叫做类的实例。

# 四、原型prototype

## 4.1 原型的定义

我们先来看一个事儿：

|  |
| --- |
| 1. function fun(){ 2. alert("你好"); 3. } 4. console.log(fun.prototype); 5. console.log(typeof fun.prototype); |



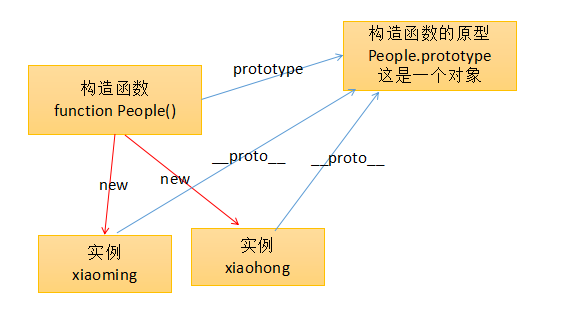
在JavaScript中，任何一个函数，都有一个prototype属性，指向一个对象。我们输出了一个函数的prototype属性，你会发现是一个空对象。输出这个prototype的类型，发现是object类型。

prototype就是英语“原型”的意思。每个函数都有原型，原型是一个对象。

**一个函数的原型，对于普通函数来说，没任何鸟用。但是如果函数是一个构造函数，那么函数的原型，用处极大！**

|  |
| --- |
| 1. //构造函数，构造函数里面没有任何语句，也就是说，这个构造函数在执行的时候，不会给创建出来的对象添加任何属性。 2. function People(){ 3. } 4. //构造函数的原型，我们更改了构造函数的原型，为一个新的对象： 5. People.prototype = { 6. name : "考拉", 7. sex : "男", 8. age : 18 9. } 10. //当一个对象被new出来的时候，不仅仅执行了构造函数里面的语句，也会把这个函数的\_\_proto\_\_指向构造函数的prototype。 11. var xiaoming = new People(); 12. console.log(xiaoming.\_\_proto\_\_); 13. console.log(xiaoming.\_\_proto\_\_ == People.prototype); 14. //当我们试图访问name、sex、age属性的时候，身上没有。那么就去查找原型，原型身上有，就当做了自己的属性返回了。 15. console.log(xiaoming.name); 16. console.log(xiaoming.sex); 17. console.log(xiaoming.age); |

prototype一定是函数的属性！当这个函数是一个构造函数的时候，那么它new出来的对象，将以它的原型那个对象为new出来的实例的原型对象。



注意，任何一个对象，都有\_\_proto\_\_属性，这个属性是Chrome自己的属性，别的浏览器不兼容，但是别的浏览器也有原型对象，只不过不能通过\_\_proto\_\_进行访问而已。

这是属性指向自己的原型对象。

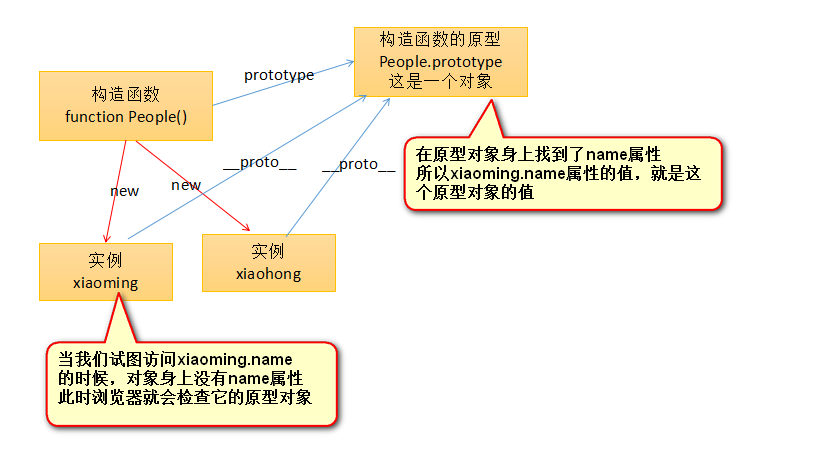
我们的JavaScript有一个非常牛逼的机制：原型链查找。

当我们试图访问一个对象身上的属性的时候，如果这个对象身上有这个属性，则返回它的值。如果它身上没有这个属性，那么将访问它的原型对象，检测它的原型对象身上是否有这个值，如果有返回它原型对象身上的这个值。

也就是说，我们刚才讲解了2个对象和一个函数的故事。任何一个函数都有原型，原型是一个对象，用prototype来访问。当这个函数是构造函数的时候，new出来的对象，它们的原型对象就是这个构造函数的原型。

prototype我们称为“原型”，只有函数有原型

\_\_proto\_\_我们称为“原型对象”，任何对象都有原型对象。



## 4.2 原型的用途

我们定义一个方法的时候，如果写在构造函数里面：

|  |
| --- |
| 1. function People(name,age){ 2. this.name = name; 3. this.age = age; 4. this.sayHello = function(){ 5. alert("你好,我是" + this.name + "我今年" + this.age + "岁了"); 6. } 7. } 8. var xiaoming = new People("小明",12); 9. var xiaohong = new People("小红",11); 10. xiaoming.sayHello(); 11. xiaohong.sayHello(); |

实际上这个函数被复制了两份，一份给了xiaoming，一份给了xiaohong。xiaoming和xiaohong这两个实例身上有了相同功能的函数，但是这个函数不是同一个函数！

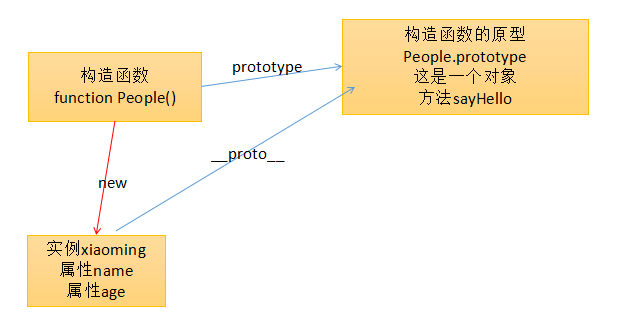
而函数天生要被复用，你没有复用函数：

|  |
| --- |
| 1. alert(xiaoming.sayHello == xiaohong.sayHello); **//false** |

xiaoming身上的函数，和xiaohong身上的函数，不是同一个函数。

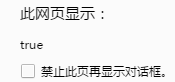
那应该怎么办呢？一句话：所有的属性要绑在对象身上，而所有的方法，定义在对象的原型对象中：

|  |
| --- |
| 1. function People(name,age){ 2. //构造函数里面，负责定义一些属性，随着构造函数的执行，这些属性将绑定到new出来的对象身上 3. this.name = name; 4. this.age = age; 5. } 6. //把所有的方法，定义在原型对象身上： 7. People.prototype.sayHello = function(){ 8. alert("你好,我是" + this.name + "我今年" + this.age + "岁了"); 9. } |



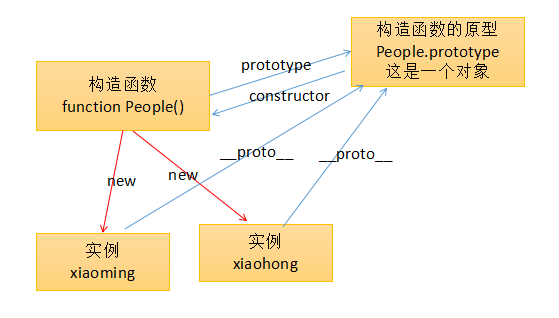
|  |
| --- |
| 1. alert(xiaoming.sayHello == xiaohong.sayHello); |

证明了xiaoming的sayHello方法和xiaohong的，是同一个函数。内存消耗小很多。



## 4.3 JavaScript原型链机制

先学一个属性，constructor，函数的原型有constructor属性，指向构造函数。



|  |
| --- |
| 1. People.prototype.constructor //指向构造函数 |

我们之前说过，一个对象的原型身上有什么，那么实例对象就也可以打点调用什么，所以：

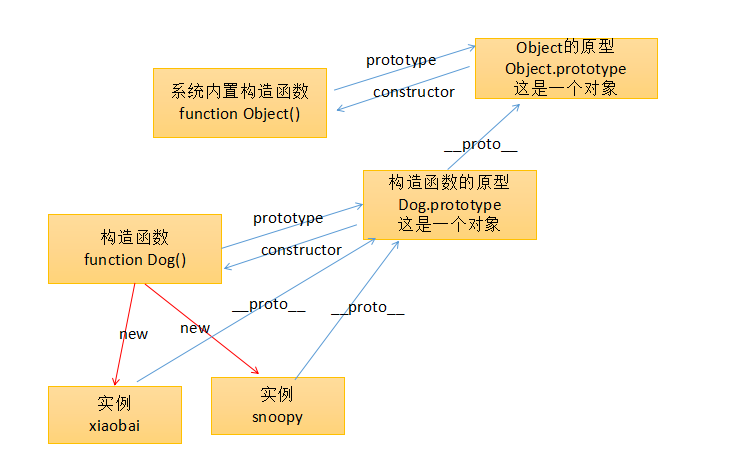
|  |
| --- |
| 1. xiaoming.constructor //指向构造函数 |

只要是对象，一定有原型对象，就是说只要这个东西是个对象，那么一定有\_\_proto\_\_属性。

世界上只有一个对象没有原型对象，这个对象就是Object.prototype。

Object是一个函数，是系统内置的构造函数，用于创造对象的。Object.prototype是所有对象的原型链终点。

所以，当我们在一个对象上打点调用某个方法的时候，系统会沿着原型链去寻找它的定义，一直找到Object.prototype。



所有的引用类型值，都有内置构造函数。比如

new Object()

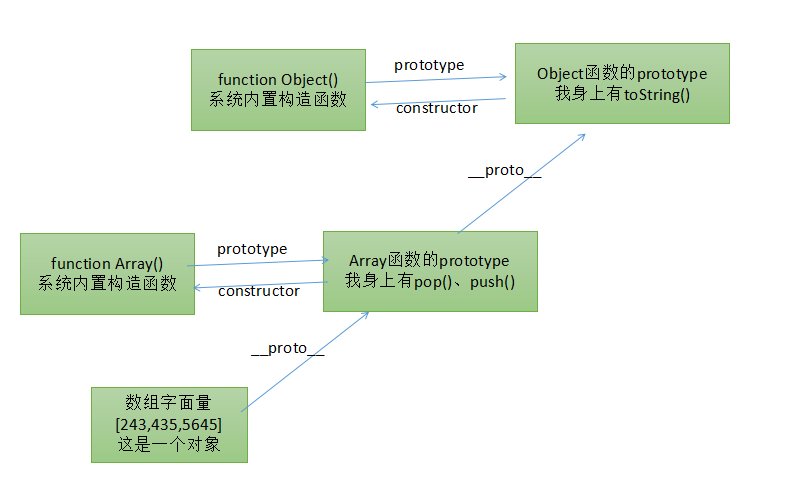
new Array()

new Function()

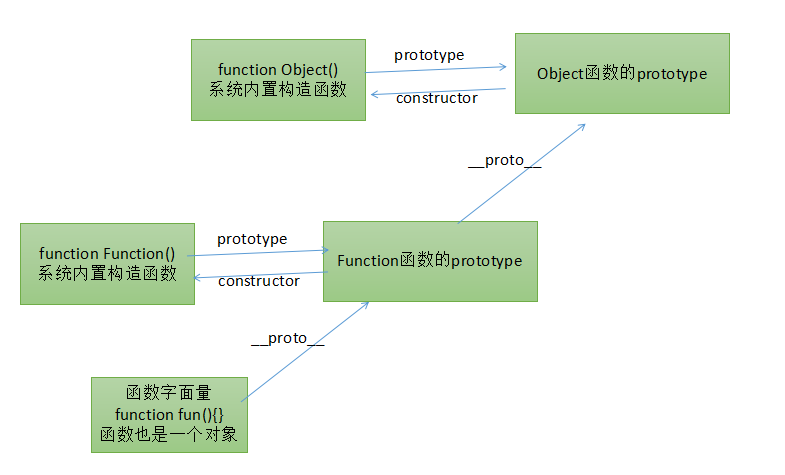
new RegExp()

new Date()

我们来看看数组的情况：



函数也是对象。JavaScript中函数是一等公民，函数是对象。函数也是对象，只不过自己能()执行。

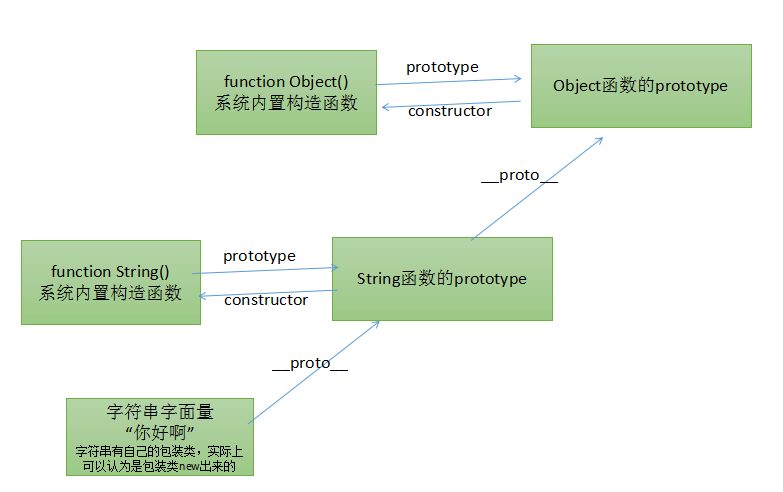


基本类型值，也有包装类型。所谓包装类型，就是它的构造函数。

new Number()

new String()

new Boolean()



所以你就能明白，为毛毛”abc”能够调用indexOf()方法，是因为String.prototype身上有这个方法。

更多的理论：继承、in操作符、hasOwnPreperty、delete明天讲。理论和实际结合起来。

# 五、面向对象的思维方式

实际上，在JS中面向对象技术，已经变为了组织代码的方式。

比如页面上有一个轮播图，就一个，我们用jQuery嗷嗷写就行了。但是页面上有三个轮播图，代码维护、共用的困难。所以就用面向对象来组织代码，这就是组件开发。

为了让大家体会面向对象的好处，或者说必须用面向对象的思路来解决问题，我们用游戏来引导你。

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |