第五章:引用类型是一种数据结构也被称为类

## 1 object类型

创建方法 1 new 构造函数（或{}）；2 对象字面量

## 2 Array类型

A创建方法同上

注意1 new Array(3)时会有歧义 2字面量创建数组时,的不同浏览器不同的解析

B检测数组

a instanceof 存在一定的问题:如果网页中包含多个框架(frame,frameset算是么)，那实际上就存在两个以上不同的全局执行环境，从而存在两个以上不同版本的Array 构造函数。如果你从一个框架向另一个框架传入一个数组，那么传入的数组与在第二个框架中原生创建的数组分别具有各自不同的构造函数。

b isArray()该方法不会管是哪一个全局环境中创建的

C转换方法 toLocaleString() toString() valueOf()

["red", "green", "blue"]--》red,green,blue

D 栈方法队列方法 push() pop() shift() unshift()

E 重排序方法reverse() sort()注意按位排序 若想按大小需要传入比较函数

function compare(value1, value2) {

if (value1 < value2) {

**return 1;**

} else if (value1 > value2) {

**return -1;**

} else {

return 0;

}

}

var values = [0, 1, 5, 10, 15];

values.sort(compare);

F 操作方法 concat()数组拼接 slice（start[，end]）返回一个子数组

可以做到数组的深拷贝

var a=[1,2,3];

var b=[];

b=a.slice(b)|| b=a.concat([])

splice（start，num，\*n个插入项）

G 位置方法 indexOf() lastIndexOf() 查找时使用的是全等的操作符

H **迭代方法**

 every()：对数组中的每一项运行给定函数，如果该函数对每一项都返回true，则返回true。

 filter()：对数组中的每一项运行给定函数，返回该函数会返回true 的项组成的数组。

 forEach()：对数组中的每一项运行给定函数。这个方法没有返回值。

 map()：对数组中的每一项运行给定函数，返回每次函数调用的结果组成的数组。

 some()：对数组中的每一项运行给定函数，如果该函数对任一项返回true，则返回true。

arr.every(function(item,index,arr){ })

//对应的元素，对应的下标，操作的数组，相当于所有元素在函数内部操作的结果相与的值，some相当于或的值

每个方法都没有改变原来数**组的值那类似于forEach有什么应用呢**

// 迭代方法

var everyRst = a.every(function(item,index){

if(item>1){ //若此处为>=1，则下面返回true

return true;

}

})

cons(everyRst); //false

var someRst = a.some(function(item,index,arr){

if(item>6){ //若此处为>=1，则下面返回true

return true;

}

// cons(index);

// cons(arr); //对应的传入的数组

})

var cc = [];

a.forEach(function(item,index,arr){

item = item +2;

cc[index] = item;

})

var dd = a.map(function(item,index,arr){

return item = item +2; **//返回值必须自己设置么？return**

// 当为 item = item +2; 打印的所有的dd都是undefined

})

cons(a); //(11) [1, 2, 3, 4, 5, 6, 4, 5, 6, 4, 7]

cons(cc);//(11) [3, 4, 5, 6, 7, 8, 6, 7, 8, 6, 9]

cons(dd);//(11) [3, 4, 5, 6, 7, 8, 6, 7, 8, 6, 9]

cons(someRst); //true

function cons(t){

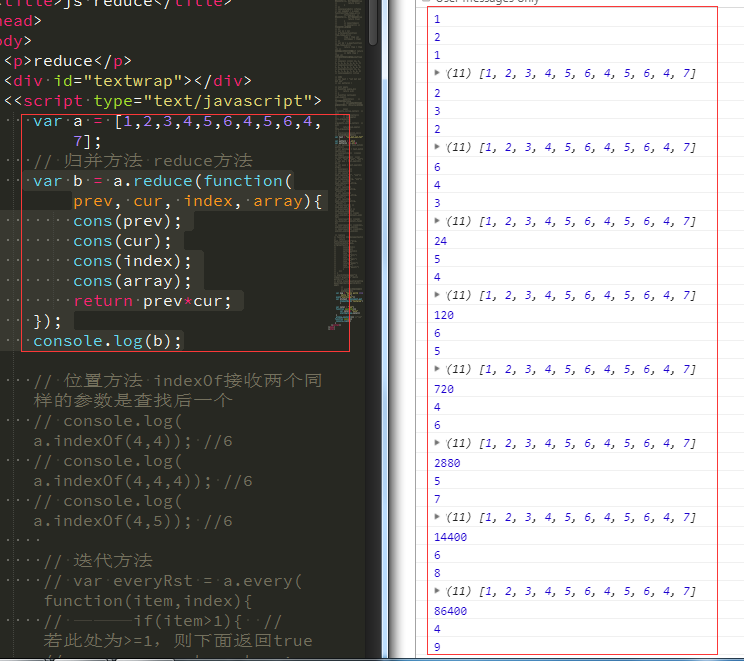
console.log(t);

}

I归并方法 reduce() reduceRight()

两个参数 每一项调用函数的结果值作为归并基础的初始值，function函数（前一个值、当前值、项的索引和数组对象）

怎样到接收两个参数的，如下图明明只是接收了一个函数



## 3 Date类型

**Date**类型使用自UTC（Coordinated Universal Time，国际协调时间）1970 年1 月1 日午夜（零时）开始经过的毫秒数来保存日期

Date.parse() //接收表示日期的字符串参数，返回对应毫秒值

Date.UTC() //同上，~~不同的是月基于0，小时基于0开始~~

Date 类型的valueOf()方法，则根本不返回字符串，而是返回日期的毫秒表示。

还有很多格式化方法和获取详细信息时间的方法；

## 4RegExp

1模式匹配标志 i g m

2实例属性

 **global**：布尔值，表示是否设置了g 标志。

 **ignoreCase**：布尔值，表示是否设置了i 标志。

 **lastIndex**：整数，表示开始搜索下一个匹配项的字符位置，从0 算起。

 **multiline**：布尔值，表示是否设置了m 标志。

 **source**：正则表达式的字符串表示，

3实例方法

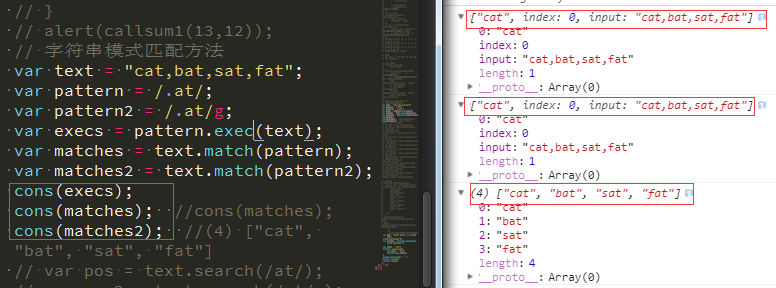
a reg.exec('string')

返回一个含有index(匹配项在字符串中的位置)和input（被应用正则的字符串）两个属性的数组，没有找到结果是返回null，设置g时会继续向下匹配，否则只匹配第一项

在同一个字符串上多次调用exec()将始终返回第一个匹配项的信息。而在设置全局标志的情况下，每次调用exec()则都会在字符串中继续查找新匹配（类似包装类型String中的match方法）

能够循环使用同一个正则表达式的exec()方法，靠的就是lastIndex，因为带全局标志的正则表达式每次匹配后都会更新lastIndex的值作为下次查找匹配的起点。

字符串的正则方法里lastIndex属性是不起作用的，不管正则模式是不是全局的。



b reg.test('text')返回true和false 经常用户用户输入验证的情况下；

c RegExp 实例继承的toLocaleString()和toString()方法，这种方法有什么大的用处么？？

4构造函数属性：input原始字符串；leftContext rightContext lastMatch lastParen

## 5 function类型

1 三种创建方式：

a 函数声明式 function(){} 会有一个函数声明提升的过程

b var sum = function(){}

c var sum=new Function("num1", num2", "return num1 + num2")。

Function 构造函数可以接收任意数量的参数，但最后一个参数始终都被看成是函数体

一定理解函数是对象函数名是指针的概念

使用不带圆括号的函数名是访问函数指针，而非调用函数。

2没有函数的重载，重名函数只会覆盖，重载可以通过arguments实现

3内部属性

arguments的主要用途是保存函数参数，这个对象还有一个名叫callee 的属性，该属性是一个指针，指向拥有这个arguments 对象的函数，如果函数引用被赋值给了另外一个函数，而本函数置空null，内部仍然写函数名的递归将会出错，一定注意，在递归调用时用处很大；

function factorial(num){

if (num <=1) {

return 1;

} else {

**return num \* arguments.callee(num-1)**}}

this 引用的是函数据以执行的环境对象——或者也可以说是this 值（当在网页的全局作用域中调用函数时，this 对象引用的就是window

sayColor()和o.sayColor()指向的是同一个函数，但是this指的是不同的值

函数对象的caller这个属性中保存着调用当前函数的函数的引用，如果是在全局作用域中调用当前函数，它的值为null

总结：

callee自身，可以用于递归调用

caller自身的调用者

this是指当前执行环境

arguments.callee以及arguments.callee.caller可以在递归以及函数互掉里面进行松散的耦合

4函数的属性和方法

a 属性length函数希望接收的参数；prototype保存他们所有实例的方法（不可枚举，不可用for in遍历）

b 方法，每个函数都包含两个非继承而来的方法 ：apply()和call()途都是在特定的作用域中调用函数，实际上等于设置函数体内this 对象的值。

apply(运行函数的作用域，参数数组或arguments对象)

this is determined by how a function is called

function callsum1(num1,num2){

cons(this); //为何指的是window

// this当前函数的作用域么？（也是一个对象 同caller的区别）

cons(arguments.callee.caller) //null alert不是相当于window.alert么

// cons(caller);

return sum.apply(this,arguments);

}

alert(callsum1(13,12));

情况1：如果一个函数中有this，但是它没有被上一级的对象所调用，那么this指向的就是window，这里需要说明的是在js的严格版中this指向的不是window

情况2：如果一个函数中有this，这个函数有被上一级的对象所调用，那么this指向的就是上一级的对象。

　　情况3：如果一个函数中有this，**这个函数中包含多个对象，尽管这个函数是被最外层的对象所调用，this指向的也只是它上一级的对象，而如下情况有些不同**

var o = {

a:10,

b:{

a:12,

fn:function(){

console.log(this.a); //undefined

console.log(this); //window

}}}

var j = o.b.fn;

j();

this永远指向的是最后调用它的对象，也就是看它执行的时候是谁调用的，例子中虽然函数fn是被对象b所引用，但是在将fn赋值给变量j的时候并没有执行所以最终指向的是window

对象中的this如上逻辑，而构造函数中的this有些不同，new关键字可以改变this的指向

指向构造函数，当构造函数中有return时需要注意，this将指向返回的对象。

传递参数并非apply()和call()真正的用武之地；它们真正强大的地方是能够扩充函数

赖以运行的作用域。

window.color = "red";

var o = { color: "blue" };

function sayColor(){

alert(this.color);

}

sayColor(); //red

sayColor.call(this); //red

sayColor.call(window); //red

sayColor.call(o); //blue

使用call()（或apply()）来扩充作用域的最大好处就是对象不需要与方法有任何耦合关系

bind()：这个方法会创建一个函数的实例，其this 值会被绑定到传给bind()函数的值

## 六 基本包装类型

为了便于操作基本类型，定义了3种特殊的引用类型：Boolean Number String

引用类型与基本包装类型的主要区别就是对象的生存期。使用new 操作符创建的引用类型的实例，在执行流离开当前作用域之前都一直保存在内存中。而自动创建的基本包装类型的对象，则只存在于一行代码的执行瞬间，然后立即被销毁。这意味着我们不能在运行时为基本类型值添加属性和方法。对基本包装类型的实例调用typeof 会返回"object"，

1 boolean使用instanceof操作符测试Boolean 对象会返回true，而测试基本类型的布尔值则返回false

2 Number

toFixed()方法会按照指定的小数位返回数值的字符串表示

toExponential()，该方法返回以指数表示法（也称e 表示法）表示的数值的字符串形式。

3 String

访问字符串中特定字符的方法是：charAt()和charCodeAt()

concat() slice()、substr()和substring(); indexOf() lastIndexOf() ；trim()

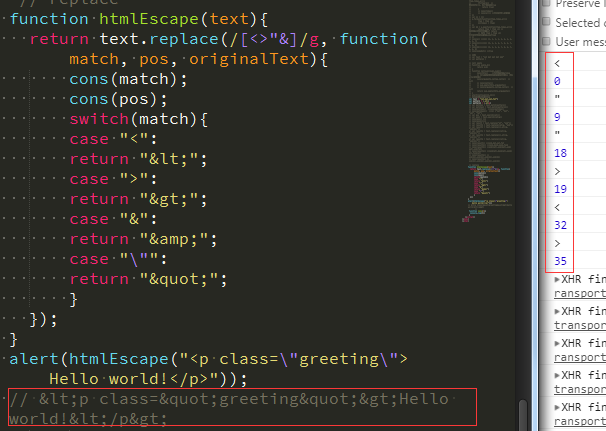
toLowerCase()、toLocaleLowerCase()、toUpperCase()和toLocaleUpperCase()

match()（同上exec方法只不过一个是正则定义的一个是String定义的）

search()方法返回字符串中第一个匹配项的索引

replace（//,要被替换的值）的模式组匹配

replace()方法的第二个参数也可以是一个函数。在只有一个匹配项（即与模式匹配的字符串）的情况下，会向这个函数传递3 个参数：模式的匹配项、模式匹配项在字符串中的位置和原始字符串。在正则表达式中定义了多个捕获组的情况下，传递给函数的参数依次是模式的匹配项、第一个捕获组的匹配项、第二个捕获组的匹配项……，但最后两个参数仍然分别是模式的匹配项在字符串中的位置和原始字符串。



split()方法可以接受可选的第二个参数，用于指定数组的大小

localeCompare()方法比较两个字符串

var stringValue = "yellow";

alert(stringValue.localeCompare("brick")); //1

比较与众不同的地方，就是实现所支持的地区（国家和语言）决定了这个方法的行为

fromCharCode()是接收一或多个字符编码，然后将它们转换成一个字符串。从本质上来看，这个方法与实例方charCodeAt()执行的是相反的操作

七 单体内置对象（不依赖于宿主环境的对象，这些对象在ECMAScript 程序执行之前就已经存在了）

1global对象（在js中体现的是window对象）

a URI编码方法

URI，是uniform resource identifier，统一资源标识符，用来唯一的标识一个资源。而URL是uniform resource locator，统一资源定位器，它是一种具体的URI，即URL可以用来标识一个资源，而且还指明了如何locate这个资源。而URN，uniform resource name，统一资源命名，是通过名字来标识资源，比如mailto:java-net@java.sun.com。也就是说，URI是以一种抽象的，高层次概念定义统一资源标识，而URL和URN则是具体的资源标识的方式。URL和URN都是一种URI。

URL是一种具体的URI，它不仅唯一标识资源，而且还提供了定位该资源的信息。URI是一种语义上的抽象概念，可以是绝对的，也可以是相对的，而URL则必须提供足够的信息来定位，所以，是绝对的，而通常说的relative URL，则是针对另一个absolute URL，本质上还是绝对的。

encodeURI()编码后的结果是除了空格之外的其他字符都原封不动，只有空格被替换成了%20 encodeURIComponent()方法则会使用对应的编码替换所有非字母数字字符

decodeURI()和decodeURIComponent()是两个对应的逆方法

b ~~eval()可以用来保留字符换换行操作~~并不是eval的作用而是alert的作用,alert以及console.log都会识别换行符

var msg = "hello world \r\n hello tom";

eval("alert(msg)"); //会换行

alert(msg); //会换行

cons(msg); //会换行

c global对象属性

d window对象

2math对象

a 对象的属性定义的一些常量eg pi

b min() max()

c ceil() floor() round()

d abs() log() exp()等其他方法

疑问总结:

1 indexOf接收两个参数是什么意思

var a = [1,2,3,4,5,6,4,5,6,4,7];

console.log(a.indexOf(4,4)); //6

*str*.indexOf(*searchValue*[, *fromIndex*])

console.log(a.indexOf(4,4,4)); //6

console.log(a.indexOf(4,5)); //6

2[^...] 不在[]中的字符：[^abc] 匹配除了a,b,c之外的字符。

var colorText = "red,blue,green,yellow";

var colors3 = colorText.split(/[^\,]+/); //["", ",", ",", ",", ""]

3 127 -- 145page

$$ $

$& 匹配整个模式的子字符串。与RegExp.lastMatch的值相同

$' 匹配的子字符串之前的子字符串。与RegExp.leftContext的值相同

$` 匹配的子字符串之后的子字符串。与RegExp.rightContext的值相同

$n 匹配第n个捕获组的子字符串，其中n等于0～9。例如，$1是匹配第一个捕获组的子字符串，

$2是匹配第二个捕获组的子字符串，以此类推。如果正则表达式中没有定义捕获组，则使

用空字符串

$nn 匹配第nn个捕获组的子字符串，其中nn等于01～99。例如，$01是匹配第一个捕获组的子字

符串，$02是匹配第二个捕获组的子字符串，以此类推。如果正则表达式中没有定义捕获组，

则使用空字符串

var result = text.replace("at", "ond");

var result2 = text.replace(/at/g, "ond");

var result3 = text.replace(/(.at)/g, "ond($1)");

var result4 = text.replace(/(at)/g, "ond($1)");

var result5 = text.replace(/(.at)/g, "ond($2)");

var result6 = text.replace(/(at)/g, "ond($2)");

cons(result); //cond,bat,sat,fat

cons(result2); //cond,bond,sond,fond

cons(result3); //ond(cat),ond(bat),ond(sat),ond(fat)

cons(result4); //cond(at),bond(at),sond(at),fond(at)

cons(result5); //ond($2),ond($2),ond($2),ond($2)

cons(result5); //ond($2),ond($2),ond($2),ond($2)

不知道这些是干什么的？？？？？？？？？？

4 this和caller区别

第六章 面向对象(Object-Oriented)的程序设计

## 一 理解对象

1理解对象,new Object实例或者对象的字面量的形式,每个对象包含属性和方法

2属性类型

A 数据属性:包含一个数据值得位置,在这个位置上可以读取和写入值;

拥有四个可以描述其行为的属性:

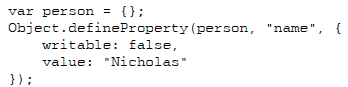
a [[Configurable]]:值为true(默认)或false 表示是否可以通过delete删除属性从而进行重新的定义,能否修改属性的特性;

b [[Enumerable]]: 值为true(默认)或false,表示能否通过for-in循环返回属性;

c [[writable]]: 值为true(默认)或false,表示能否修改属性的值;

d [[value]]:默认值为undefined,包含这个属性的数据值,就是在此处写入和读出的;

可以利用Object.defineProperty(属性所在的对象,属性的名字,描述符对象)方法修改默认的特性,其中描述符对象必须是configurable、enumerable、writable 和value。如



需要注意的是:一旦将[[Configurable]] 设置为false不可配置的,就不能在变回可配置的了;

B 访问器属性:

访问器属性有如下4 个特性:

a [[Configurable]]同上

b [[Enumerable]]同上

c [[Get]]：在读取属性时调用的函数。默认值为undefined。

d [[Set]]：在写入属性时调用的函数。默认值为undefined。

访问器属性不能直接定义，必须使用Object.defineProperty()来定义

3 定义多个属性：

Object.defineProperties(实例对象，{name:{value:''}，set:function(){}，\*\*\*})

4 读取属性的特性

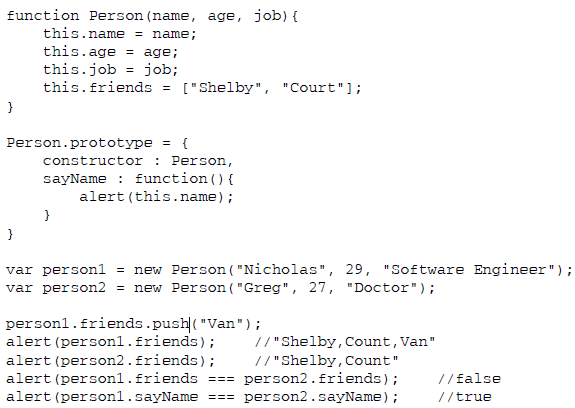
Object.getOwnPropertyDescriptor(对象，变量名)

## 二 创建对象

|  |  |  |
| --- | --- | --- |
| 1工厂模式 | 2构造函数模式 | 3原型模式 |
|  |  |  |
| 工厂模式虽然解决了创建  多个相似对象的问题，但却没有解决对象识别的问题（即怎样知道一个对象的类型)  比如Object我可以new多个，然后在绑定不同的对象和方法不是一样的么（同）工程设计模式 | 注意:1 使用new;2P大写；3使用this；4没有return；5没有显式的创建对象;  其中new实现了：1创建新对象；2将构造函数的作用域赋值给新对象（this只想这个新对象）；3执行代码返回对象；  有一个constructor（构造函数）属性，该属性指向Person，但一般利用instanceof检测对象；  任何函数，只要通过new 操作符来调用，那它就可以作为构造函数；而任何函数，如果不通过new 操作符来调用，那它跟普通函数也不会有什么两样  存在的问题：每个方法都要在每个实例上重新创建一遍 | 1-注意:A变量之间先从内部查找;B对原型对象所做的任何修改都能够立即从实例上反映出来（不论创建实例的先后顺序）；C一定不要先创建实例，在重写整个原型对象如下（会抱错）    因为把原型修改为另外一个对象就等于切断了构造函数与最初原型之间的联系  2-使用原型对象的好处是可以让所有对象实例共享它所包含的属性和方法  3-利用hasOwnProperty()方法检测实例的固有属性(非原型中的属性);  4-可以利用hasPrototypeProper(实例对象,属性名)遍历到原型属性的,返回true/false  5-在做对象的深拷贝时可以如下实现  var cloneObject = {};  for(var key in children){  if(children,hasOwnProperty(key)){  cloneObject[key] = children[key];  }  }  上面是原生方法进行克隆，在angularjs中，利用angular.copy()就可以实现功能  6- Object.keys(实例对象Person.Prototype)实例固有的，（不是从原型中继承来的）可以枚举的  7通过原生对象的原型，不仅可以取得所有默认方法的引用，而且也可以定义新方法；定义的新方法，有效范围是什么，比如String.prototype.sayhello();之后再整个项目中所有的字符串都可以访问这个方法么。还是在一定的范围之内；  8-存在的问题：当原型对象中包含引用对象属性时，其实例中改变对应属性的值将改变其他实例的值，就是深浅拷贝的问题； |

4针对原型模式和构造函数模式对应的缺点可以使用两者组合的模式。

构造函数模式用于定义实例属性，而原型模式用于定义方法和共享的属性



5 动态的原型模式，4的变形，可以通过检查某个应该存在的方法是否有效，来决定是否需要初始化原型。

6 寄生构造函数模式

7 稳妥构造函数模式（适用于安全模式）

## 三 继承

1-函数签名包括：

参数和参数类型；

一个返回值和类型；可能抛出或回传的异常；

一个面向对象程序方法的有效性信息；

而上面的这些信息可以用来实现类型的检测，函数重载，接口等等，这些都是js所不具备的东西

js是一种松散型的动态语言，不需要提前声明变量的类型，这些类型在程序运行时将会自动处理；

2-原型链 作为继承的主要方法

默认原型都是Object实例

一个构造函数A通过原型继承另外一个构造函数B(B的实例对象),那么A的实例包含构造函数B的属性和方法,依次继续下去产生的链条就是原型链

SubType.prototype = new SuperType();

3-使用instanceof 操作符确定原型和实例的关系，只要用这个操作符来测试实例与原型链中出现过的构造函数，结果就会返回true

alert(instance instanceof Object);//true

也可以使用 isPrototypeOf（）返回值同上一样，注意和上面用法的不同

alert(instance instanceof Object);

alert(Object.prototype.isPrototypeOf(instance));//true

4-注意

A 定义方法或重写方法一定在替换原型的语句之后;

//继承了SuperType

SubType.prototype = new SuperType(); //替换原型的语句

//添加新方法

**SubType.prototype.getSubValue = function (){**

**return this.subproperty;**

**};**

//重写超类型中的方法

**SubType.prototype.getSuperValue = function (){**

**return false;**

**};**

B 通过原型链实现继承时,一定注意不要使用字面量对象创建原型

//继承了SuperType

**SubType.prototype = new SuperType();**

//使用字面量添加新方法，会导致上一行代码无效,因为重新定义之后切断了指向

**SubType.prototype = {**

**getSubValue : function (){**

**return this.subproperty;**

**},**

**someOtherMethod : function (){**

**return false;**

**}**

**};**

C 原型链最大的问题就是,原型中包含引用类型的属性(会被所有实例共享)

5借用构造函数:在子类型构造函数的内部调用父类型构造函数

function SubType(){

//继承了SuperType,下面可传递参数

SuperType.call(this);//这句话都作了什么操作

}

问题:没有避免构造函数每个方法都在每个实例中创建一遍的缺点,而且超类对子类是不可见的(不会被单独使用)

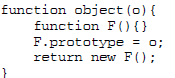
6组合继承

是使用原型链实现对原型属性和方法的继承，而通过借用构造函数来实现对实例属性的继承

组合继承避免了原型链和借用构造函数的缺陷，融合了它们的优点，成为JavaScript 中最常用的继承模式, instanceof 和isPrototypeOf()也能够用于识别基于组合继承创建的对象

7原型式继承

object()对传入其中的对象执行了一次浅复制



var person = {\*\*};

var anotherPerson = object(person);

var yetAnotherPerson = object(person);

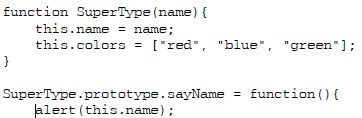
ES5 规范化使用object.create(用作新对象原型的对象,为新对象定义额外属性的对象)

8 寄生式继承

9 寄生组合是继承

## 附:

1 最常用的组合继承例子





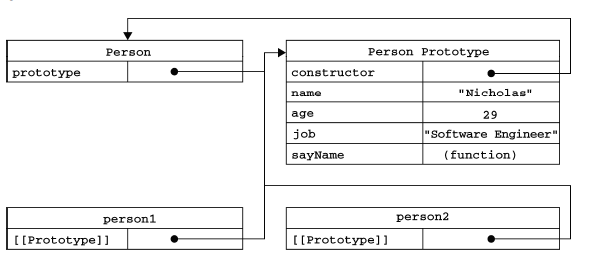
2:原型文字解析和图解

var person = new Person()

创建函数的同时创建一个prototype属性,指向原型对象(叫做\_\_proto\_\_指针,存在于实例和原型对象之间)

默认所有原型对象自动获得一个constructor属性,包含指向prototype属性所在函数的指针;即person.prototype.constructor 指向person

实例



4原型链图解

