# Javascript基础

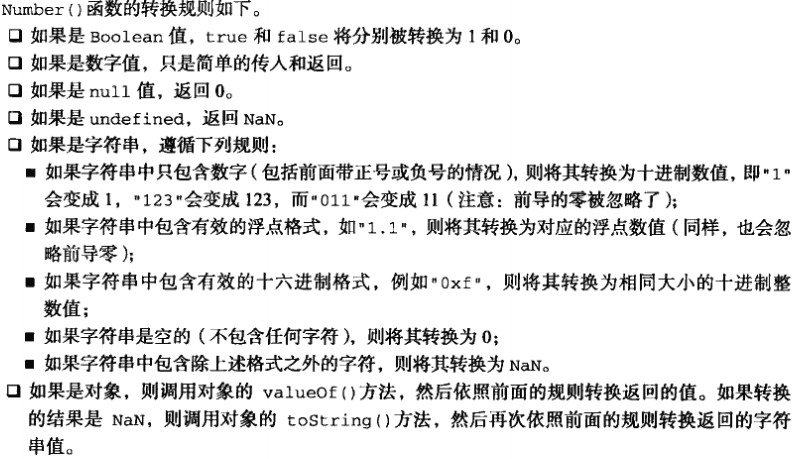
## 数据类型

### Boolean

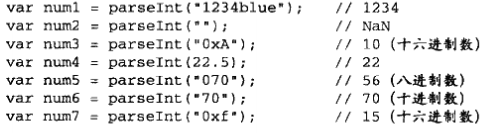
* Boolean类型可以和任意的其他类型进行转换，调用的函数式Boolean。如下所示：
  + (String)非空字符串true，空字符串为false
  + (Number)任何非零数字为true, 0和NaN为false
  + (Object)任何对象都为true，null为false.
  + (Undefined)为false.

### Number

* + 永远不要测试某个特定的浮点数值。浮点数的最高精度为17. 0.1+0.2=0.30000000000000004，并不是等于0.3。这是使用IEEEE754数值的浮点计算的通病。
  + Number.MAX\_VALUE和Number.MIN\_VALUE分别返回最大值和最小值。超过这个值会被看做是infinite或者-infinite。可以使用函数isFinite来判断是否是有穷数字，也就是介于最大值和最小值直接的数字。
  + NaN: 非数字。使用isNaN函数来判断，判断时会进行转换，然后再判断。10和”10”都是非数字。如果isNaN(object), 会首先调用object.valueOf方法来判断，然后通过toString来判断。
  + 数字转换函数：Number，parseFloat，parseInt.



* + parseInt转换时，如果开始字符是数字，会忽略掉其他的字符，比如parseInt(“123abc”) = 123.

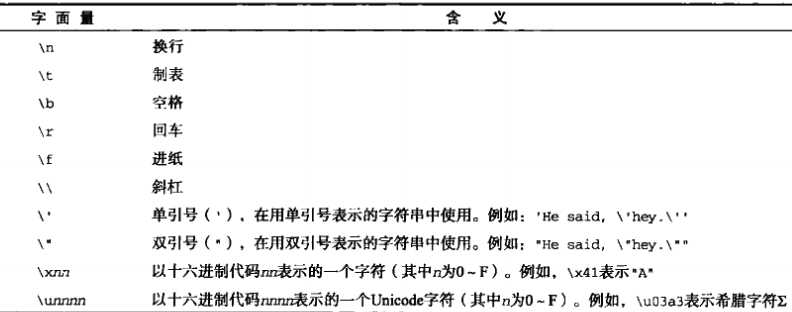


parseInt第二个参数为进制数，比如8进制或者16进制。

* + parseFloat只能解析10进制数字

### String

* + 转义符



* + toString转换：不能转换null和undefined.字符串是不可变的。
  + String函数可以转换任意的对象，String(null)转换为”null”

### Object对象

* + Constructor：构造器函数
  + HasOwnProperty：属性是否在实例类型中
  + isPropertyOf：传入的对象是否是另外一个对象的原型
  + propertyIsEnumerable：检查给定的属性是否可以使用for in循环
  + toLocaleString：地区字符串
  + toString：转换到字符串。
  + valueOf：转换

### Array

* + 数组的栈访问方法，后进先出, push+pop
  + 数组的队列访问方法，先进先出, push+shift
  + 数组的队列访问方法，相反的方向，从数组前面加入数据，后端删除数据, unshift+pop
  + Join, reverse, sort, concat, slice, splice

## 变量与作用域

### 变量

* 变量可以保存两种类型，基本类型和引用类型。基本类型包括undefined，boolean，string，number和null。
* 基本类型占据固定的内存大小，被保存在栈内存中。基本类型的赋值会创建一个副本。
* 引用类型的值是对象，保存在**堆内存**中。引用类型的变量实际上是一个指向对象的指针，而不是对象本身。
* 从一个对象向另一个对象赋值引用类型时，实际上赋值的是指针。
* 基本类型的判断使用typeof，引用类型的判断使用instanceof

### 执行环境

* 执行环境有全局执行环境和函数执行环境。
* 每次进入一个新的执行环境，都会创建一个用于搜索变量和函数的作用链。
* 函数的局部环境可以访问函数作用于链中的变量，也可以访问父环境和全局环境中的变量。
* 全局环境不能访问局部环境中的变量。
* 变量的执行环境有助于确定何时释放内存。

### 垃圾回收

* 离开作用域的值就被标记为回收，然后被删除。
* 标记清除或者引用计数是两种垃圾回收清理算法。
* 解除变量的引用有助于消除循环引用，而且对垃圾回收期有好处，应该及时的解除不再使用的全局变量。

## 引用对象

### 数组

* 动态的增加或者更改数组的长度
* Typeof arr & instanceof arr的区别
* Push/pop方法，先进后出，栈
* Push/shift方法，先进先出，队列
* Unshift/pop方法，反向队列
* Reverse/sort/compare方法，排序与比较
* Concat/slice方法：链接和分离数组
* Splice方法：增加或者删除数组
* Every/foreach/map/some/filter等过滤方法
* Reduce/reduceRight方法

### 正则表达式

* 正则表达式内部属性
* 检测方法match和test

### 函数

* 函数本身是对象，函数名是指向对象的指针。
* 正因为函数名是指向函数对象的指针，所以函数不存在重载。
* 函数声明和函数表达式的先后顺序。
* 函数内部属性arguments和this. Argument中有属性callee, 指向拥有这个arguments对象的函数。另外一个属性是this，this为执行环境的上下文对象。一定要注意，函数名是个指针。函数本身的callee指的是调用当前函数的函数引用。
* 函数length属性，指的是函数的参数个数。Prototype为函数的原型对象。
* Apply和call基本没啥区别就是参数的传递方式不一致。

### 全局对象

* IsNaN非数字
* isFinite: 有穷数字
* parseInt：转换为整型
* parseFloat：转换为浮点数
* encodeURI:对URI进行编码，整个url编码
* encodeURIComponent：对URI进行编码，某部分进行编码

### Math

* Math.min/max
* Math.ceil/floor/round
* Math.random

# 面向对象设计

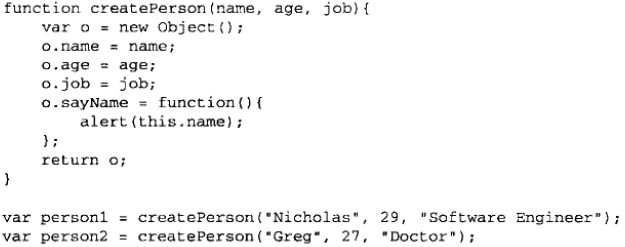
## 对象

### 属性类型（元元数据）

* Configurable: 是否可以通过delete属性删除
* Enumrable:是否可通过for in循环
* Writable：是否能修改属性的值
* Value：属性的值
* Set：可以给属性定义set
* Get：可以给属性定义get
* 修改这些属性类型的默认值通过Object.defineProperty或者是Object.defineProperties函数来修改。（IE9+才支持）
* 读取这些属性的特性通过: Object.getOwnPropertyDescriptor方法来获取某个属性所定义的属性类型值。

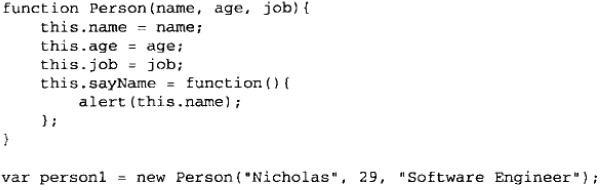
## 创建对象

### 工厂模式



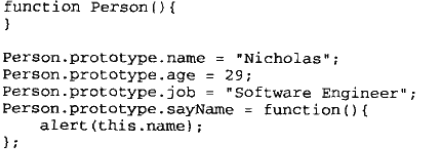
缺点是无法识别具体的对象，instanceof还是Object.

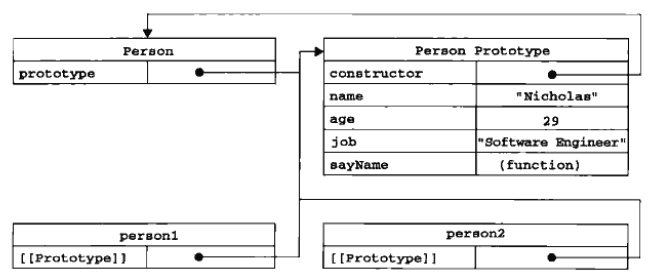
### 构造函数模式



优点是可以通过instanceof来识别，缺点是每个方法都要在实例上单独创建一遍。

### 原型模式





原型，构造函数和实例之间的关系

优点：让所有实例共享属性和方法。

* IsPrototype: 通过这个方法来判断是否XXXX的原型。

Person.prototype.isPrototype(person1)

* 在实例中增加一个同名的方法或者属性后，会覆盖原型中的对应的内容。
* 原型链的搜索
* Hasownproperty是否存在于实例中
* PropertyIsEnumrable: 属性是否可以枚举
* 判断属性在原型中，而不是在实例中。



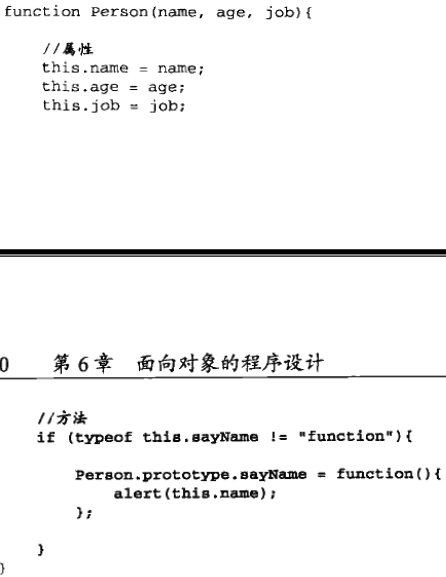
* Object.keys获取对象上可以枚举的属性，for in循环时
* 不能以字面量的方式直接更改原型对象，这样contructor对象会发生变化。可以直接更改他的contructor属性。
* 原型声明后还可以动态的增加方法，即时他所有的实例都已经创建了。

缺点：原型最大的缺点是由于共享导致的问题。

### 组合使用构造函数和原型

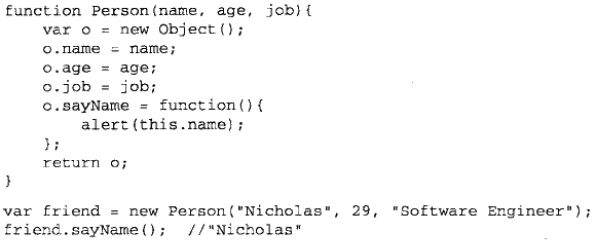
* 构造函数用于定义实例属性，原型用于定义方法和共享的属性。

### 动态原型模式

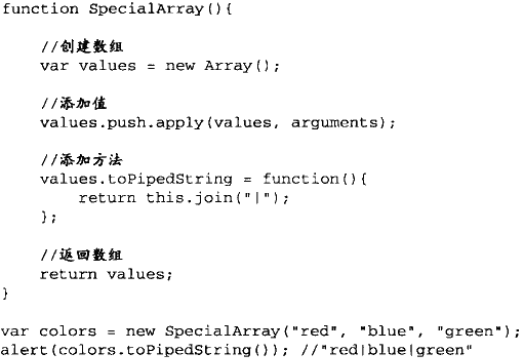


这个方法的设置只判断一次，只判断一个就可以了。

### 寄生构造函数

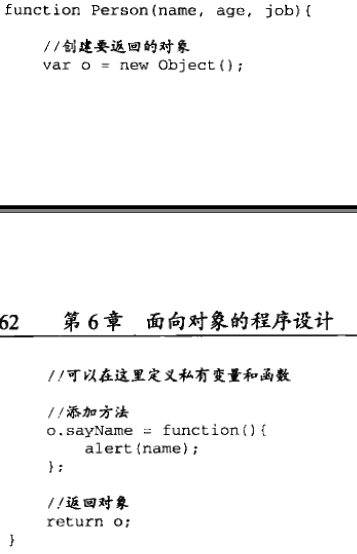


这种情况可以在特殊的情况下为对象创建函数，比如：



缺点是不能依赖instanceof判断对象。

### 稳妥构造函数模式



稳妥指的是没有公共属性，只有私有属性，并且私有属性是通过方法来进行访问的。无法直接访问属性，不使用new操作符号。

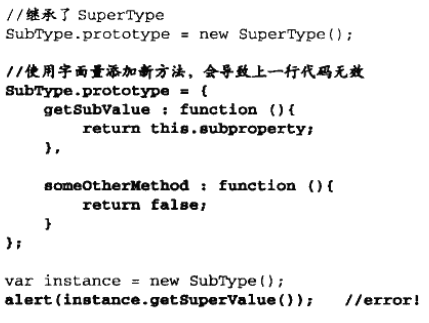
## 继承

### 原型链



通过上面的原型链条实现继承。

不能使用字面量来创建原型，比如下面：

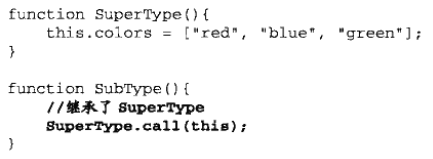


因为这样会重写原型链。

问题：

* 共享引起的问题
* 不能向超类的构造函数中传递参数

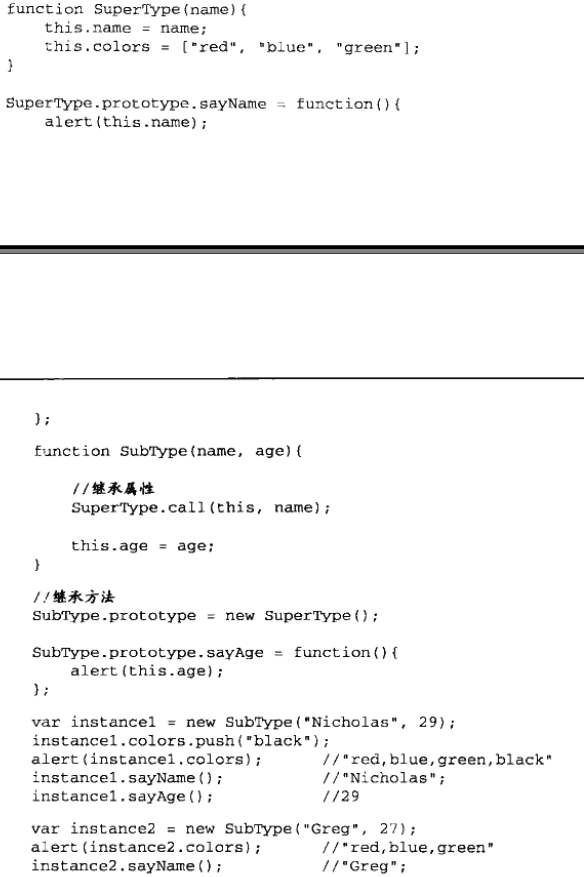
### 借用构造函数



在子类中调用父类的构造函数，可以向父类传递参数。

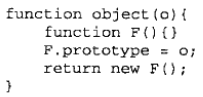
### 组合继承

借用原型链和借用构造函数的优点，然后组合处理。



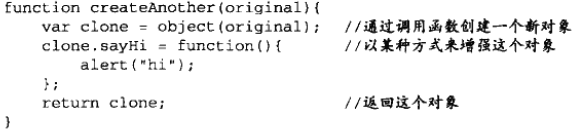
最大问题是调用两次构造函数。

### 原型式继承



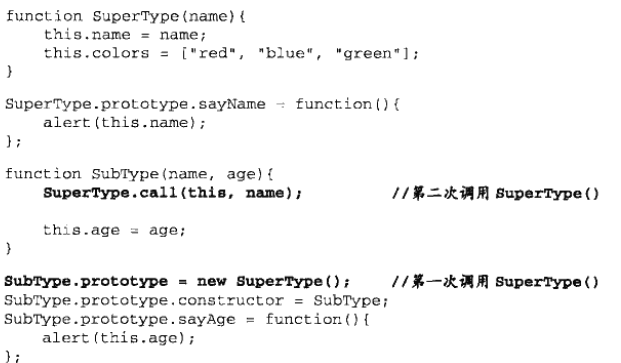
新对象基于某个对象进行创建，将参数o作为一个新函数的原型对象，因此这种继承方式会造成的object的各个实例之间是共享属性和方法的。

### 寄生式继承

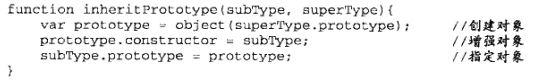


在主要考虑对象而不是自定义类型和构造函数的情况下，可以使用这种继承模式。

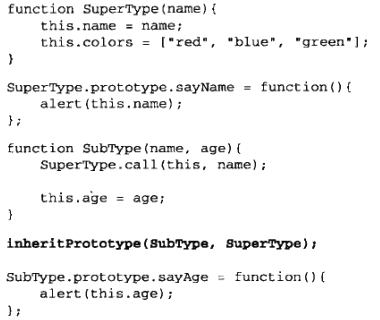
### 寄生组合式继承



以上为组合式继承



两个参数分别是子类型构造函数和超类型构造函数。



175