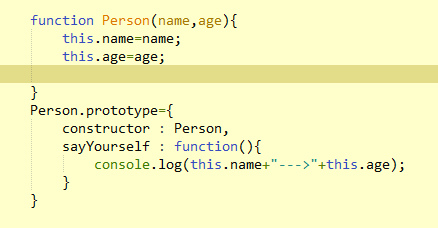
2017-12-29

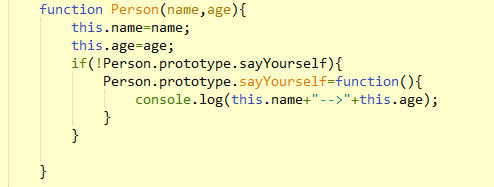
构造函数和原型模式各有利弊，现在最广泛的创建类和对象的方法是将二者结合起来使用



实例方法写在prototype中，所有实例对象共同拥有这个方法，节省资源

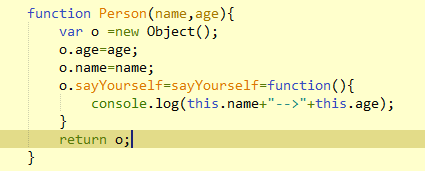
属性写在构造函数中，每个实例对象都拥有一份

为了适应面向对象的思想，也可以将原型方法写在构造函数里面（动态原型模式），例如这样



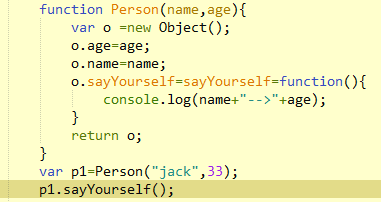
关键在于这个if判断，使得只有一个原型方法。

如果包括前两种模式都不可用的时候，可以使用寄生构造函数模式



Person仅仅是用作封装

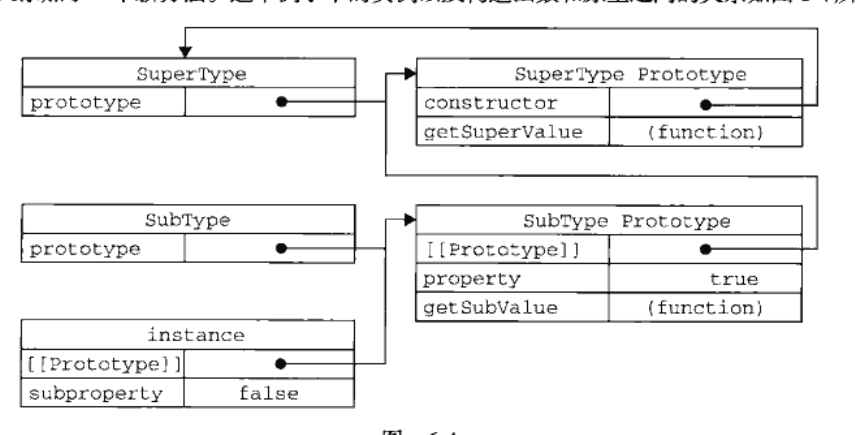
与这个模式类似，有一个叫做稳妥对象的模式，非常适合在安全环境中使用，这个模式和寄生模式最大的不同有2点，第一个是创建实例方法不使用this，第二个是不使用new操作符调用构造函数。**特别要注意的是，稳妥对象没有公共属性！**

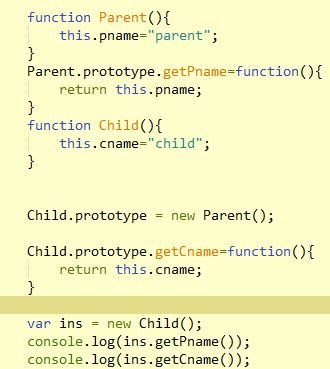


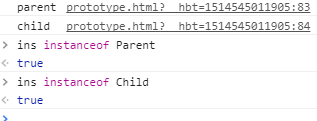


JavaScript的继承，只支持实现继承，并不能像Java等语言一样使用接口、抽象类等等的概念。

而JavaScript实现继承主要是依靠原型链来实现。其基本思想是利用原型让一个类型继承另一个引用类型的属性和方法。







这个例子中，child的原型没有使用默认的原型，而是使用parent实例，parent的原型则是再上一级的原型，层层递进。当然，Object是所有类的原型……

务必结合JavaScript的方法和属性调用的方向去理解，所以只能在定义了child的原型之后才能加入child特有的方法。

**子类型需要重写超类型的某个方法，或者添加某个超类型不存在的方法，必须要在“替换原型语句”之后。**

原型链的最大问题是，在原型链中定义的属性，（尤其是引用类型的属性，这个问题更加严重）会被所有实例共享。所以，**属性一定要在构造函数中定义。**

原型链的第二个问题是，在创建子类型的实例时，不能向超类型的构造函数中传递参数。

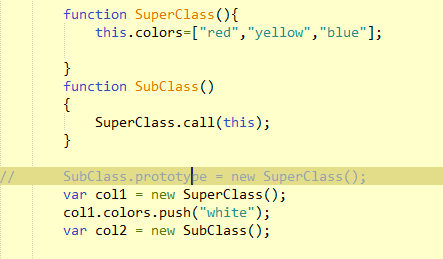
2017-12-30

为了解决原型链的最大问题，可以使用一种叫“借用构造函数”的技术。

这种技术的思想是，在自类型构造函数的内部调用超类型构造函数。

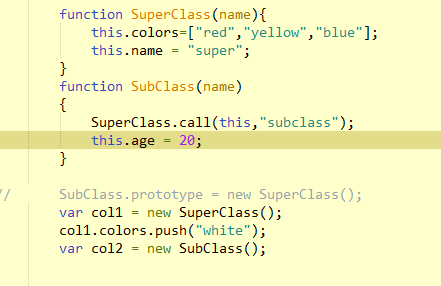
有点类似Java的写法？例如无参构造方法，下面的重载方法中则是带参数的构造方法。

其关键在于理解使用call或者apply来改变函数的执行环境。





相对于原型链，借用构造函数，可以在子类型构造函数中，向超类型构造函数传递参数，这是一个很大的优势。





借用构造函数的问题和构造函数模式一样，方法都在构造函数中定义，不能函数复用(因为每个实例都有一个方法)，而且，超类型的原型中定义的方法，对于子类型来说是不可见的，那么，子类型以及后续的子类型等等，都必须使用构造函数模式了。