**再谈范式**

<http://www.silverna.org/blog/?p=40>  
  
**范式：new C <=> TC = function(){}, TC.prototype = C.prototype, o= new TC, C.apply(o, arguments);**  
  
用文字和伪代码解释就是  
与构造器C构造过程等价的函数调用序列为  
1) 建立一个空的function模板TC  **var TC = function(){}**  
2) 设这个模板TC的原型设为构造器C的原型  **TC.prototype = C.prototype**  
3) 无参数构造TC的对象o  **var o = new TC();**  
4) 以o为this执行C的构造器    **C.apply(o, arguments);**  
5) 得到的o和直接构造C的过程完全等价   **o <=> new C**  
  
这个范式至少有两个作用，第一，用作继承时延迟基类构造函数的调用  
  
   提示：您可以先修改部分代码再运行  
  
第二，实现构造器和函数调用的统一，让对象构造支持apply和call  
  
   提示：您可以先修改部分代码再运行

是的，子类调用父类方法这里没有考虑，主要是实际用的不多  
不过真要考虑，扩展一下，应该也是可以做到的  
apply的问题是这样的，如果用new来构造对象的话，是不能apply的  
比如希望 new Class.apply(null, arguments)是不行的;  
只能用 Class.createInstance(Class, arguments);  
这个范式的本质即对象构造的实质是函数调用，因此可以用函数调用的过程来代替。

，其实我觉得要处理继承，对于脚本来说用几行代码就够了  
关键是多利用js本身的prototype特性，而不是去很全面的描述继承  
对于面向对象来说，prototype-based的思想和class-based的思想一样是完备的  
事实上prototype-based的思想是人类更直接地描绘自然世界的方法，它直接利用相似性来“认知世界”  
这和人类的思维模式是非常的接近的  
新生的儿童是通过相似类比（prototype）来学习和巩固接触到的新知识的，而只有在更高级别上才会对知识进行整理和归类（class）  
  
所以说对prototype-based来说：  
A.prototype = new B(); 的现实语言就是“A很像B”，或者“像B那样对待A就行了”  
Tiger.prototype = new Cat(); //照猫画虎  
  
对class-based来说，就不一样：  
A extends B; 的现实语言是“A是B的一种”  
abstract Felidae extends Animal //猫科动物是一种动物  
Tiger extends Felidae  //老虎是一种猫科动物  
Cat extends Felidae   //猫是一种猫科动物  
  
区别就在上面，对class-based来说，猫和老虎的关系是（必须）通过抽象的“猫科动物”来建立联系的，猫和老虎重用的是猫科动物的通用行为  
而对于prototype-based来说，因为猫本身像老虎，所以这个相似本身就能够让猫建立对老虎的行为重用的可能，至于哪些行为能够重用，交由使用者去处理。  
  
我们认知世界也是同样，既有非常严谨的对世界的归类，也有比较宽松的“相似性”描述，而后者，可能出现得更加频繁。  
人脑的“联想”通常是偏prototype-based的，而逻辑思维同时具有class-based（分析）、prototype-based（推理）两种特质。  
  
本质上说，编程是一种把现实世界的模型抽象化后再利用计算机具现化的行为，所以无论是prototype-based还是class-based，都是思维抽象化的手段，在有prototype的情况下没必要为了继承而继承……

对，或者说JS的prototype根本不是通过考虑事物之间的“系谱”或者说“分类”来联系事物的  
通俗说，只要两个事物有相似性，而希望通过一个已知事物的相似性去描述另一个未知事物的话，那么通过prototype-chain把它们链在一起就行了。  
如果非要深究的话，对js而言“混血”的说法还有一点点不恰当，因为js的prototype本不是去描述“血缘（系谱）关系”的，prototype中没有“固有关系”，因为这个模型不对事物进行分类，所以也根本不存在所谓的“混血”，呵呵

是因为这样的原因，假如p可以以不同的参数来构造，例如 new p(1,2)、new p(3,4)、new p('a','b','c')......，那么不用你说的所谓“中转”的话有时候你无法知道当前的“继承”究竟需要用什么参数来构造  
  
如果你了解的话，回忆一下C++或者Java，在建立继承关系的时候，是不会确定参数构造的，所以说，prototype也有它的缺陷，就是它在建立“继承”关系时，有一个父类对象构造的过程，而这个过程本来应该出现在“子类”的构造时，而不是描述继承关系时。当然如果你非要解决这个问题，你可以强行约定构造器不传入可变参数，并且把初始化操作放在另一个约定的函数内（比如init）去执行，但是那样比较麻烦，而且如果你用的别人的代码不符合这个约定还必须去修改。我的方案则是一个通用的方法，这就是范式的意义所在。