**javascript设计模式交流(一) ——Singleton Pattern**

即使是简单的脚本语言，应用良好的模式可以得到非常“优美”的代码和较高的效率。  
尤其是对于交互要求较高的B/S系统，非常有必要用设计模式来优化代码。  
  
单件模式（Singleton Pattern）是一种非常基本和重要的创建型模式。  
“单件”的职责是保证一个类有且只有一个实例，并提供一个访问它的全局访问点。  
在程序设计过程中，有很多情况下需要确保一个类只能有一个实例。  
  
传统的编程语言中为了使一个类只有一个实例，最容易的方法是在类中嵌入静态变量，并在第一个实例中设置该变量，而且每次进入构造函数都要做检查，不管类有多少个实例，静态变量只能有一个实例。为了防止类被多次初始化，要把构造函数声明为私有的，这样只能在静态方法里创建一个实例。  
  
在javascript中，虽然我们仍然可以指定静态方法来构造对象，但由于我们不能利用构造函数的“私有”来禁止多个实例的生成，因此要完全实现Singleton并没有想象中那么简单。  
  
请看下面的例子：  
  
   提示：您可以先修改部分代码再运行  
  
上面的例子试图通过传统的方式来实现Singleton模式，而通过调用SingletonTest.getInstance()来获得对象确实可以保证“唯一实例”，然而，这个例子的失败之处在于它并没有有效地禁止Singleton对象的构造，因此如果我们在程序代码中人工加入new SingletonObject()，仍然可以获得到多个对象而导致模式失败。  
  
一个改进的替代方案如下：  
  
   提示：您可以先修改部分代码再运行  
  
这样当用户试图自己创建多个对象的时候，通过人工抛出异常来阻止。不过这么做还是有一点点违反了"初衷"，即没有满足“必须通过静态方法来构造唯一实例”这个基本条件。因为用户可以在最开始的时候还是可以采用new操作符来构造对象，比如在一开始写var instA = new SingletonObject()来构造instA并不会导致抛出异常，这不能不说是这种方法的一个缺陷。  
  
于是我们进一步思考，得到了下面第三种方法，这种方法巧妙利用了“匿名”函数的特征来禁止对SingletonObject类构造函数的访问，可以说比较好的模拟了私有构造函数的特性，从而比较完美地解决了用javascript实现Singleton Pattern的问题。  
  
   提示：您可以先修改部分代码再运行