面向对象设计模式　单件

曹东刚
caodg@pku.edu.cn

北京大学信息学院研究生课程 - 面向对象的分析与设计
http://sei.pku.edu.cn/~caodg/course/oo

单件模式 (Singleton Pattern)

单件模式用于创建唯一的对象实例

- 很多对象只需一个，如线程池、缓存、日志、打印机驱动等
- 用全局变量或静态变量也能保证确保创建唯一的实例，但单件模式提供了全局的访问点，和全局变量一样方便，又没有全局变量的缺点
- 单件模式可以允许在需要的时候才创建对象

核心问题：如何保证一个对象只被实例化一次？

通常如何创建对象

常见的创建对象的方式

```
1    MyObject o = new MyObject() ;
```

问题: 对象的创建不受限制，可创建任意多个

办法: 限制用new操作符创建对象

限制用 new 创建对象

将构造函数声明为私有的

```
1  public MyClass {
2    private MyClass() {}
3  }
```

只有 MyClass 的实例才能调用该构造函数，但由于其他类无法实例化 MyClass，故产生“鸡生蛋、蛋生鸡”的问题

私有构造函数 + 公有类方法

用静态方法避开必须实例化的问题

```
1  public MyClass {
2    private MyClass() {}
3
4    public static MyClass getInstance() {
5      return new MyClass() ;
6    }
7  }
```

getInstance() 是一个类方法 (静态方法)，不需要对象实例

经典的单件模式实现

一个初步的实现

```
1  public class Singleton {
2    private static Singeleton uniqueInstance ;
3
4    private Singleton () {}
5
6    public static Singleton getInstance() {
7      if (uniqueInstance == null)
8        uniqueInstance = new Singleton() ;
9      return uniqueInstance ;
10   }
11 }
```

定义单件模式

单件模式（Singleton Pattern）

单件模式确保一个类只有一个实例，并提供一个全局的访问点

要点：类肩负了管理实例的额外责任!

Singleton
static uniqueInstance
static getInstance()

经典的单件模式实现在多线程下的表现

多线程下有问题

```
1  public class Singleton {
2    private static Singeleton uniqueInstance ;
3
4    private Singleton {}
5
6    public static Singleton getInstance() {
7      if (uniqueInstance == null)
8        uniqueInstance = new Singleton() ;
9      return uniqueInstance ;
10   }
11 }
```

利用 synchronize 处理多线程问题

synchronized 保护整个方法，性能会有影响

```
1  public class Singleton {
2    private static Singeleton uniqueInstance ;
3
4    private Singleton {}
5
6    public static synchronized Singleton getInstance() {
7      if (uniqueInstance == null)
8        uniqueInstance = new Singleton() ;
9      return uniqueInstance ;
10   }
11 }
```

提早创建实例

在类加载时就创建实例

```
1  public class Singleton {
2    private static Singeleton uniqueInstance =
3      new Singleton() ;
4
5    private Singleton {}
6
7    public static Singleton getInstance() {
8      return uniqueInstance ;
9    }
10 }
```

用双重检查加锁方法减少同步

只有第一次会同步

```
1  public class Singleton {
2    private volatile static Singeleton uniqueInstance ;
3    private Singleton {}
4    public static Singleton getInstance() {
5      if (uniqueInstance == null) {
6        synchronized (Singleton.class) {
7          if (uniqueInstance ==  null)
8            uniqueInstance = new Singleton() ;
9        }
10     }
11     return uniqueInstance ;
12   }
13 }
```

单件模式小结

- 单件模式确保程序中一个类最多只有一个实例
- 单件模式也提供这个实例的全局访问点
- 考虑性能和可用资源的因素小心设计单件
- 实现时要考虑运行环境的支持，如 JVM 的垃圾回收机制、类加载器特性等

关于设计原则的小结

1. 封装变化
2. 多用聚合、少用继承
3. 针对接口编程，不针对实现编程
4. 尽最大可能将要交互的对象设计为松耦合的
5. 对扩展开放，对修改封闭
6. 依赖抽象，不要依赖具体类