链滴

# Java 设计模式 (1) 单例模式

作者：wthfeng

原文链接：https://ld246.com/article/1480035821021

来源网站：链滴

# 概述

单例模式是23种设计模式之一。顾名思义，即在应用环境中一个类只能创建一个对象实例，这样处理因为有些场景如：线程池、缓存、日志对象、打印机对象等只能有一个实例，若有多个，则会产生程、资源等等问题。单例模式的简要定义为：**确保一个类只有一个实例，并提供一个全局访问点。 **

注意：

1. 以下讨论均在一个类加载器、一个JVM前提下；
2. 以下讨论不涉及反射机制。

# 方法一：延迟实例化方式（非线程安全）

单例模式的实现方式很简单，即将构造器设为私有（private），不允许其他类通过new方式创建对象并对外提供一个返回该类实例的接口。具体代码如下：

```
public class Singleton{
  private Singleton(){}
  //将构造器设为私有
  private static Singleton singleton;
  public Singleton getSingleton(){
     if(singleton==null){
        singleton=new Singleton();
     }  return singleton;
  }
}
```

该类将构造器设为private，则只有该类内部可以通过new创建实例，同时对外提供getSingleton()方，调用该方法返回一个实例。（若实例未创建，则创建，创建后始终返回该实例） 此种方式**在需要类实例时才会创建，可以称之为“延迟实例化“的方式（也称为“懒汉模式“）。另外重要的是，此方法是非线程安全** 的。 试想若有2个线程同时到达 if(singleton==null)的判断中，并且都得到true，则就会产生两个实例，破坏单例模式的约束。

# 方法二：双重检查加锁

若想在多线程环境中应用单例模式，还需要对刚才的方法加以改变，将创建的实例看做共享的资源，需要创建资源时加锁控制访问，操作完成后释放锁。

```
public class Singleton{
  private Singleton(){}
  private volatile static Singleton singleton;
  public static Singleton getSingleton(){
      if(singleton==null){  //检查实例，若未创建进入同步块
        synchronized(Singleton.class){
          //同步块开始
           if(singleton==null){ //进入同步块后再检查一次，防止出现2个线程同时通过第一次检查的况（当时还未同步）
              singleton= new Singleton();
           }
        } //同步块结束
```

```
        }
        return singleton;
    }
}
```

volatile 关键字的含义，简单来说即：告诉JVM，对于这个成员变量不能保存它的私有拷贝，而应直与共享成员变量交互。 singleton= new Singleton()这句话不是原子操作，其主要包含3步：

1. 给Singleton分配空间;
2. 初始化Singleton的构造函数;
3. 将分配的空间地址返回给实例。

java编译器允许处理器乱序执行以提高执行效率，因此在实际执行中可能是1-2-3，也可能是1-3-2。是1-3-2，若当线程1执行完3后轮到线程2执行，此时线程1由于执行了3，singleton已不为空，而未行构造函数等操作，因此得到的为不完整的实例，线程2使用该实例时就会出错。 另外需要注意的是：**重检查加锁方法必须是JDK1.5或以后版本才支持。**

## 方法三：初始实例化（“饿汉模式”）

以上使用线程加锁的方式可以解决多线程的方式，不过有些麻烦，还要考虑性能的一些影响。JVM可保证一个类只被加载一次，因此可以考虑下面的方法

```
public class Singleton{

  private Singleton(){}

  private static Singleton singleton = new Singleton();
  public Singleton getSingleton(){
     if(singleton==null){
        singleton = new Singleton();
      }
      return singleton;
      }
  }
```

此种方法在类被初次记载时即初始化了该类实例，不存在线程安全问题。但如果该类创建需配置文件该方法就无能为力了。

参考文献

1、《Head First 设计模式》

2、 http://crud0906.iteye.com/blog/576321