一、常见的单例模式

静态属性 优点:被使用时,不需要在创建。 饿汉式单例 类加载阶段,就被初始化。 缺点:如果不被调用,就造成空间浪费,可被反射和反序列化破坏。 静态块 线程不安全: 类被调用的时候,才创建对象,缺点是当多个线程调用的时候,会导致创建多个对象,破坏了单例。 类被调用的时候,才创建对象,对外提供的调用方法加上synchronized关键字,达到线程安全的效果— 线程安全 缺点是由于使用synchronized关键字,性能下降,每个线程想获得对象的时候都需要同步。 懒汉式单例 双重检查锁,就是将synchronized关键字移动到方法内部,在其内外加入两次对象是否为空判断, 双重锁 来达到线程安全且性能能效最低的效果。 静态内部类: 类被加载时,首先加载内容类,内部类中初始化外部类,利用内部类的特性,实现单例。 枚举式: 内部是饿汉式实现可以避免多线程同步问题,是线程安全的,也可以防止反序列化破坏单例。 注册式单例 定义一个Map容器,类名作为key,反射获取对象作为vaule来实现单例,为了实现线程安全,要加入, synchronized. 优点:线程安全, ¬ TheadLocal : 同─个线程内保证单例,不同线程创建的对象不同 缺点:应用有一定限制。

内部类

破坏单例的方式

反序列化.

```
public class LazyInnerClassSingleton {
    private LazyInnerClassSingleton(){ }
    public static final LazyInnerClassSingleton getInstance(){
        return LazyHolder.LAZY; }
    private static class LazyHolder{
        private static final LazyInnerClassSingleton LAZY = new LazyInnerClassSingleton();
    }
}

LazyInnerClassSingleton

LazyHolder

构造器

G射 使得私有的构造函数被访问,创建的是新的对象
```

反序列化是通过构造方法导致单例被破坏的,为了保证单例不被破坏重写readResolve()。

jdk层实现了改方法,如果单例中有该方法,则通过反射直接调用该方法,返回实例

优点:性能较好。 : 缺点:线程不安?

缺点:线程不安全,可被反射和反序列化破坏。

优点:线程安全。

缺点:性能较差,可被反射和反序列化破坏。

优点:被访问创建对象,避免空间浪费,线程安全。

缺点:加入synchronized,性能较差,可被反射和反序列化破坏。由

于JVM乱序执行的问题,可能导致线程不安全

优点:被使用时,避免空间浪费,线程安全。

缺点:如果不被调用,就造成空间浪费,可被反射和反序列化破坏。

优点:本身就是现场安全的,在jdk层面就不允许反射和反序列来破坏。 缺点:

优点:对象方便管理。

缺点:为了线程安全加入了synchronized,性能降低了。