链滴

来动手写一个安全的单例

作者：JackHoo

原文链接：https://ld246.com/article/1509881604755

来源网站：链滴

序言

犹记得当初老师跟我们讲设计模式时候问我们对象的创建有两种模式，一种是饿汉模式，还有一种呢我们认为是饱汉模式。。。

饿汉模式

也就是立即加载，当使用类的时候对象已经创建完毕

```
public class StarveObject {
    private static StarveObject starveObject = new StarveObject();

    public StarveObject() {
    }
    public static StarveObject getInstance(){
        return starveObject;
    }
}
```

懒汉模式

延迟加载，当调用get方法时候对象才被创建

```
public class LazyObject {
    private static LazyObject lazyObject;

    public LazyObject() {

    }
    public static LazyObject getInstance(){
        if (lazyObject != null) {

        } else {
            lazyObject = new LazyObject();
        }
        return lazyObject;
    }
}
```

但是上面的单例设计存在问题，当在多线程环境下，就很可能创建出多个实例。于是我想到了加synchonized关键字

```
synchronized public static LazyObject getInstance(){
    if (lazyObject != null) {

    } else {
        lazyObject = new LazyObject();
    }
    return lazyObject;
}
```

这样做就解决了问题，但是每个线程想要获取该单例对象的时候都需要等待锁，浪费了很多时间。

DCL双检查锁创建单例

```
public static LazyObject getInstance(){
    if (lazyObject != null) {

    } else {
        synchronized (LazyObject.class) {
            if (lazyObject == null) {
                lazyObject = new LazyObject();
            }
        }
    }
    return lazyObject;
}
```

使用静态内置类实现单例

```
public static LazyObject getInstance(){

    if (lazyObject != null) {

    } else {
        synchronized (LazyObject.class) {
            if (lazyObject == null) {
                lazyObject = new LazyObject();
            }
        }
    }
    return lazyObject;
}
```

利用enum枚举实现单例

```
public class MyEnum {
    private Connection connection;

    public MyEnum(MyEnum myEnum) {
        connection = new Connection();
    }
    public Connection getConnection(){
        return connection;
    }
}
```