单例模式和多线程

作者：hiquanta
原文链接：https://ld246.com/article/1500810874723
来源网站：链滴

单例模式是23个设计模式中比较简单，也是最常用的模式之一，虽然简单，但在多线程并发访问时如不注意一些使用细节，会引发意想不到的bug。

单例模式

定义：保证一个类只有一个实例，并且自行实例化并向整个系统提供这个实例。

类图：待完成

优点：

- 减少内存开支
- 减少性能开销
- 避免对资源的多重占用
- 提供和共享全局访问量

缺点：

- 扩展性差
- 测试不方便
- 单例模式和单一职责莫设计原则向冲突

单例的变相形式

饿汉模式

```
public class EagerSingleton {
    private static EagerSingleton singleton=new EagerSingleton();
 private EagerSingleton(){

    }
    public static EagerSingleton getSingleton(){
        return singleton;
  }

}
```

懒汉模式

该代码采用了DCL双锁检测(double checked locking)，避免了因为多线程并发下可能出现的异常

```
  public class LazySingleton {
    private volatile static LazySingleton singleton;
 private LazySingleton(){

    }
    public static LazySingleton getSingleton() throws InterruptedException {
        if (singleton != null) {

        } else {
            Thread.sleep(3000);
```

```
 synchronized (LazySingleton.class) {
                if (singleton == null) {
                    singleton = new LazySingleton();
  }
            }
        }
        return singleton;
  }
}
```

测试类：

```
  public class MyThread extends Thread{
    @Override
  public void run() {
        super.run();
  System.out.println("EagerSingleton"+EagerSingleton.getSingleton().hashCode());
 try {
            System.out.println("LazySingleton"+LazySingleton.getSingleton().hashCode());
  } catch (InterruptedException e) {
            e.printStackTrace();
  }
    }
}
public class Test {
    public static void main(String[] args) {
        MyThread t1=new MyThread();
  MyThread t2=new MyThread();
  MyThread t3=new MyThread();
  MyThread t4=new MyThread();
  t1.start();
  t2.start();
  t3.start();
  t4.start();
  }

}
```

运行结果

```
EagerSingleton908288934
EagerSingleton908288934
EagerSingleton908288934
EagerSingleton908288934
LazySingleton1515217202
LazySingleton1515217202
LazySingleton1515217202
LazySingleton1515217202
```

可以看到hash值相同，证明了我们的结论，试着把双重检查中的判空代码去掉，再运行下结果，你会现单例失效了！