单例模式解析

1，概述

单例模式通俗的讲就是单例对象的类必须保证只有一个实例存在。整个系统中就只有一个类的实例。

2，实现方式

- 饿汉式单例：在类加载的过程中。就主动创建实例。

  饿汉式单例代码：

```
public class singleclass {
    public static void main(String[] args) {
        T x = T.getT();
    }
}

class T {
    private static T singleT = new T();
    public T(){}
    public static T getT() {
        return singleT;
    }
}
```

- 懒汉式单例：等到真正使用的时候才会去创建实例。

```
public class singleclass2 {
    public static void main(String[] srgs) {
        T x = T.getT();
    }
}

class T {
    private static T singleT;

    public T() {
    }

    public static T getT() {
        if (singleT == null)
            singleT = new T();
        return singleT;
    }

}
```

3,单例模式的优点

- 在内存当中，只有一个对象，节省空间。
- 避免频繁的创建销毁对象，可以提高性能。
- 避免对共享资源的多重占用。
- 为整个系统提供一个全局的访问点。

4，单例模式的线程安全问题

- 由于饿汉式是再类加载的时候就创建出一个类实例对象，因此这种方式具有线程安全的特性。
- 但是对于懒汉式来说，在多线程的环境下则不宜动就是安全的。因为在在懒汉式实现单例模式的时候，有这样一句话：if(singleT==null) singleT=new T()；这句话当处于多个线程的时候，可能会在if判断句的地方发生歧义，不能保证线程安全。

  解决办法：

  双重校验线程安全

```
import com.sun.xml.internal.txw2.output.StaxSerializer;

public class DoubleCheckThreadsafe {
    public static void main(String[] args) {
        for(int i=1;i<=10;i++){
            new Thread(){
                @Override
                public void run() {
                    System.out.println(Test.getTest());
                }
            }.start();
        }
    }
}

// 饿汉模式
class Test {

    public static volatile Test test = null;
    public static Test getTest() {
        if (test == null) {                //第一次检查
            synchronized (Test.class) {
                if (test == null)          //第二次检查
                    test = new Test();
                return test;
            }
        }
        return test;
    }

}
```