单例模式

> 单例模式，顾名思义，就是我们的代码中只实例化出一个对象，就是单例模式，有的人说，为什么用单例模式啊，这个很简单，因为有一些时候，不是所有的对象都可以被实例化多次的，因为，无论是内存空间的大小，和逻辑关系上面都是不允许的，举个最简单的例子，有一件事情，需要皇上去做，但是由于客观事实的要求，我们只能有且只有一个皇上，所以这个时候一定要控制，对象数量的个数，不能够超过1，以下我会为大家介绍单例模式怎么实现。

单例模式，比较重要的应用是，我们可以确保我们的线程安全，因为如果我们用多个对象来操作我们的线程池的时候是很为危险的。

单例模式，同时又会分为饿汉模式，和懒汉模式，他们有一点不同，饿汉模式是在初始化类的时候，就把对象声明好，而懒汉模式，则是在调用的时候再去声明这个对象，不过他们起到的效果是一样的，就是都能确保对象的唯一性。

饿汉模式:

```
public class Singleton {
    private Singleton() {
    }
    private static Singleton iSingleton = new Singleton();
    public static Singleton getInstance() {
        return iSingleton;
    }
}

public class Test {
    public static void main(String[] args) {
        Singleton aSingleton = Singleton.getInstance();
        Singleton bSingleton = Singleton.getInstance();
        if (aSingleton==bSingleton) {
            System.out.println(true);
        }else {
            System.out.println(false);
        }
    }
}
```

当我们将构造方法，变成私有之后，便不能够再通过new的方法来创造对象了，但是我们有暴露一个接口，让用户来获取我们已经写好了对象，这样就能够确保我们所有用到的所有的对象，都是同一个对象了，当然我还加上了测试，当然最终的结果肯定是true了。

懒汉模式：

> 懒汉模式和饿汉模式不同的仅仅是，当第一个人访问这个对象的时候，这个对象是不存在的，只需要新建一个这个对象就可以了。

```
public class Singleton2 {
    private Singleton2() {
    }
    private static Singleton2 singleton2;
    public static Singleton2 getInstance() {
        if (singleton2 == null) {
            singleton2 = new Singleton2();
        }
        return singleton2;
    }
}
```