懒汉式单例模式

一、问题引出

1、单例：只能通过静态方法获取一个实例，不能通过构造器来构造实例

2、假设有多个线程调用此单例，而调用的获取单例的函数作为操作共享单例的代码块并没有解决线程的安全问题，会导致多个线程都判断实例是否为空，此时就会导致多个实例的产生，也就是单例模式的线程安全问题。

3、解决线程安全问题的思路：

- 将获取单例的方法改写成同部方法，即加上synchronized关键字，此时同步监视器为当前类本身。（当有多个线程并发的获取实例时，同时只能有一个线程获取实例），解决了单例模式的线程安全问题。
- 用同步监视器包裹住同步代码块的方式。

二、代码实现

1、不安全问题

```
public class Bank {
    //私有化构造器
    private Bank(){}
    //初始化静态实例化对象
    private static  Bank instance = null;

    //获取单例实例,此种懒汉式单例模式存在 线程不安全问题（从并发考虑）
    public static  Bank getInstance(){
        if(instance==null){
            instance = new Bank();
        }
        return  instance;
    }
```

2、 同步线程安全

```
    //同步方法模式的线程安全
    public static synchronized Bank getInstance1(){
        if(instance==null){
            instance = new Bank();
```

```
        }
        return  instance;
    }

    //同步代码块模式的线程安全（上锁）
    public  static Bank getInstance2(){
        synchronized (Bank.class){
            if(instance==null){
                instance = new Bank();
            }
            return  instance;
        }
    }
```

3、效率更高的线程安全的懒汉式单例模式

```
    /**
     * 由于当高并发调用单例模式的时候，类似于万人夺宝，只有第一个进入房间的人才能拿到宝物，
     * 当多个人进入这个房间时，第一个人拿走了宝物，也就另外几个人需要在同步代码块外等候，
     * 剩下的人只需要看到门口售罄的牌子即已知宝物已经被夺，可以不用进入同步代码块内，提高了效率。
     *
     *
     * */
    public static Bank getInstance3(){
        if (instance==null){
            synchronized (Bank.class){
                if(instance==null){
                    instance = new Bank();
                }
            }
        }
        return  instance;
    }
}
```