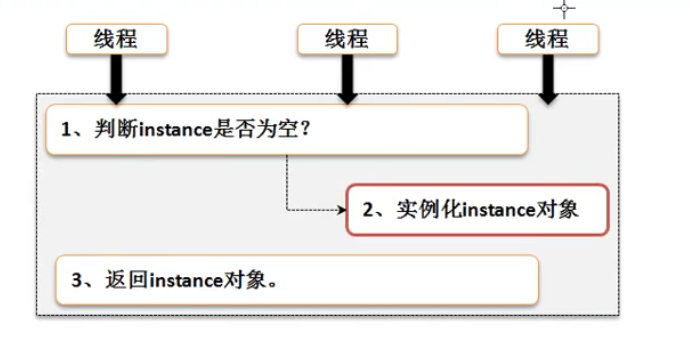
1. 反射与单例设计模式

单例设计模式的核心本质在于:类内部的构造方法私有化，在类的内部产生实例化对象进行类中结构调用，单例模式模式一共有两种:饿汉式，懒汉式，本次主要讨论懒汉式的单例设计模式:

范例:观察懒汉式单例设计模式的问题

|  |
| --- |
| package com.company.反射机制;  /\*\*  \* 观察懒汉式单例设计模式的问题  \* @author JackWen  \*/  public class SingltonDemo {  public static void main(String[] args) {  for (int x = 0 ;x < 3 ;x ++){  new Thread(()->{  Singlton s = Singlton.getInstance();  s.println();  },"线程"+x).start();  }  }  }  class Singlton{  private static Singlton instance = null ;  private Singlton(){  System.out.println("---实例化对象---");  };  public static Singlton getInstance(){  if (instance == null){  return new Singlton();  }  return instance ;  }  public void println(){  System.out.println("\*\*\*Singlton\*\*\*");  }  } |
| 运行结果: |

通过上面的分析我们可以知道，在多线程的时候，单例模式出现多个对象，但是单例设计的目的是在程序允许过程中只允许出现一个对象，因此现在的单例模式不再是单例设计模式。



此时我们发现在问题就在于代码本身的不同步问题，导致多个线程同时判断instance为空，而现在要解决同步问题，但是如果直接加在方法上，则会导致效率非常低。

范例：解决单例设计模式同步问题

当我们实例化单例模式时，应该立刻与主内存中的数据对象保持一致，而不是修改副本。因此在instance 属性前加上volatile，直接操作它本身。

|  |
| --- |
| package com.company.反射机制;  import java.util.concurrent.locks.Lock;  import java.util.concurrent.locks.ReentrantLock;  /\*\*  \* 观察懒汉式单例设计模式的问题  \* @author JackWen  \*/  public class SingltonDemo {  public static void main(String[] args) {  for (int x = 0 ;x < 3 ;x ++){  new Thread(()->{  Singlton s = Singlton.getInstance();  s.println();  },"线程"+x).start();  }  }  }  class Singlton{  private static Lock lock = new ReentrantLock();  private static volatile Singlton instance = null ;  private Singlton(){  System.out.println("---实例化对象---");  };  public static Singlton getInstance(){  lock.lock();  try {  if (instance == null){  return instance = new Singlton();  }  }finally {  lock.unlock();  }  return instance ;  }  public void println(){  System.out.println("\*\*\*Singlton\*\*\*");  }  } |

那么此时就有一道面试题:请编写单例设计模式

* 【100%】直接编写一个饿汉式的单例设计模式，并且实现构造方法私有化。
* 【120%】在Java中那些地方使用到了单例设计模式？Runtime类，Spring框架。
* 【200%】懒汉式设计模式的问题？