day06_线程、同步

一.线程

1.多线程的原理

```java
public class MyThread extends Thread {

    // 重写run()方法来设置线程任务
    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println("MyThread线程第" + i + "次执行");
        }
    }

}
```

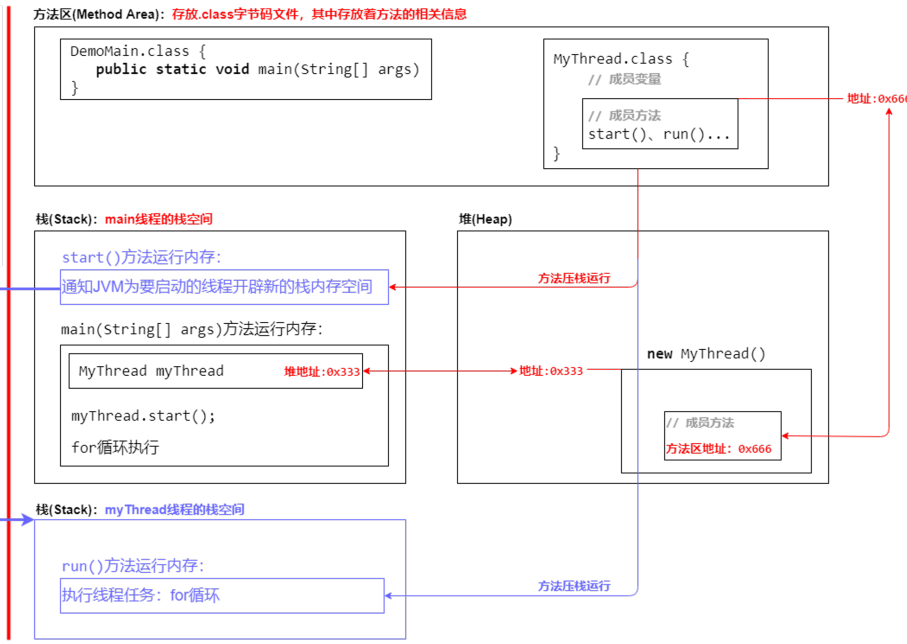
```java
public class DemoMain {
    public static void main(String[] args) {

        // 创建一个自定义线程类MyThread的对象
        MyThread myThread = new MyThread();

        // 调用Thread类中的start()方法启动该线程
        myThread.start();

        for (int i = 0; i < 10; i++) {
            System.out.println("main线程第" + i + "次执行");
        }

    }
}
```



程序启动运行时，java 虚拟机会启动一个进程。在这个进程中：主线程在 main 方法被调用时创建，此时会开辟一个 "main 线程的栈内存空间" 。线程 myThread 调用 start 方法时，会通知 JVM 为要启动的线程 myThread 开辟一个 "myThread 线程的栈内存空间"，并在其中运行 run 方法来执行线程任务。

这样程序就有两个线程在同时运行，每一个执行线程都有自己的栈内存空间，此时两个线程一起抢占 CPU 的执行时间，实现两个线程的并发执行。由于两个线程在不同的栈空间中，所以两个线程之间互不影响。当线程的执行任务结束后，会自动释放线程的栈内存空间，当所有执行线程都结束了，那么 JVM 进程也就结束了。

2.Thread 类的常用方法

3.创建多线程程序的第二种方式——利用 Runnable 接口的实现类创建线程

4.使用匿名内部类创建线程

二.线程安全

1.线程安全问题

2.线程安全问题的解决方法

Java中提供了同步机制 (synchronized) 来解决线程安全问题，即在某个线程修改共享资源的时候，其他线程不能修改该资源，等待该线程修改完毕后，其他线程才能去抢夺 CPU 的执行权，这样就保证了共享数据在多个线程中的同步，解决了线程不安全的问题。有以下三种方式来完成线程同步机制：

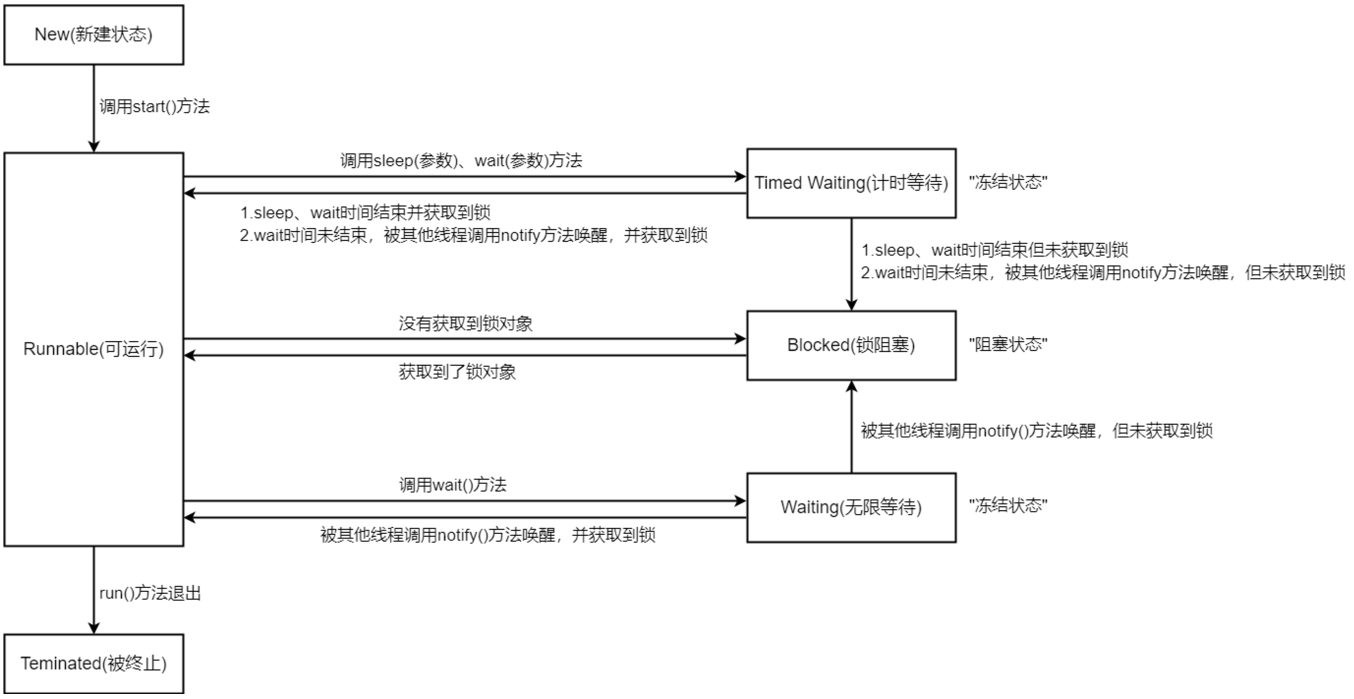
(1).同步代码块

(2).同步方法

(3).Lock 锁

三.线程状态

在 java.lang.Thread.State 枚举中一共定义了六种线程状态，用来描述一个线程的生命周期。线程状态之间的转化关系如下：



- New (新建状态)：线程刚被创建，还没调用 start( ) 方法启动
- Runnable (可运行状态)：线程可以在 JVM 中运行的状态。可能正在执行，也可能没有，但是一旦获得 CPU 的执行权就可以运行(即随时准备好运行的状态)
- Teminated (被终止状态)：run( ) 方法正常退出而死亡，或者因为没有捕获的异常终止了 run( ) 方法而死亡

1.Blocked (锁阻塞状态)

Blocked (锁阻塞状态)：一个正在阻塞等待一个锁对象的线程处于这一状态。例如：线程A、线程B在代码中使用同一个对象锁，当两个线程同时启动并且同时试图获取锁对象时。如果线程A获取到锁就会进入到 Runnable 可运行状态，那么线程B就会进入到 Blocked 锁阻塞状态，等什么时候线程B获取到锁，它才会进入 Runnable 可运行状态。这是线程由 Runnable 状态进入Blocked 状态，除此之外，Timed Waiting、Waiting 状态也会在某种情况下进入锁阻塞状态。

2.Timed Waiting (计时等待状态)

3.Waiting (无限等待状态)