OS进程状态 VS JVM线程状态

第一章 OS 操作系统进程（传智播客版权）

1.1 进程的引入

提起进程这个概念，让很多经常使用计算机的人感到陌生，其实我们经常和它打交道，只要在计算机上运行一个程序，相应的一个进程就诞生了，而且它伴随着整个操 作过程，直到程序终止。进程在操作系统中是一个非常抽象、非常重要、非常难以理解的概念。对进程概念的深入透彻的理解，有助于理解操作系统中的各种机制原 理。

1.2 什么是进程

进程：进程（Process）是计算机中的程序关于某数据集合上的一次运行活动，是系统进行资源分配和调度的基本单位。简单理解为当一个程序进入到内存就形成了进程，进程也就是程序的一次执行过程。

进程状态：反映进程执行过程的变化。这些状态随着进程的执行和外界条件的变化而转换。进程状态即体现一个进程的生命状态。

进程状态反映进程执行过程的变化。这些状态随着进程的执行和外界条件的变化而转换。在三态模型中，进程状态分为三个基本状态，即运行态，就绪态，阻塞态。在五态模型中，进程分为新建态、终止态，运行态，就绪态，阻塞态。

对应英文即：new，ready，running，waiting，terminated。

1.3 三态模型

在一个进程的执行过程中，从创建到消亡的整个生命期间，有时占有CPU处理器执行，有时虽可运行但分不到CPU处理器、有时虽有空闲处理器但因等待某个事件的发生而无法执行，这一切都说明进程和程序不相同，代表着程序的生存状态，它是活动的且有状态变化的，这可以用一组状态加以刻画。为了便于管理进程，一般来说，按进程在执行过程中的不同情况至少要定义三种不同的进程状态：

- 就绪态(ready):进程具备了运行条件，等待CPU分配才能运行。
- 运行态(running):进行正在执行，CPU正在处理该程序。
- 等待态(waiting):又称为阻塞态(blocked)，指进程正在等待某件事情完成，不具备运行的状态。

通常，一个进程在创建后将处于就绪状态。每个进程创建之后，在执行过程中，任意时刻当且仅当处于上述三种状态之一。当然最终都会被终止，进入死亡状态。

从理论上分析有6种状态转换，我们就一一来说一下：

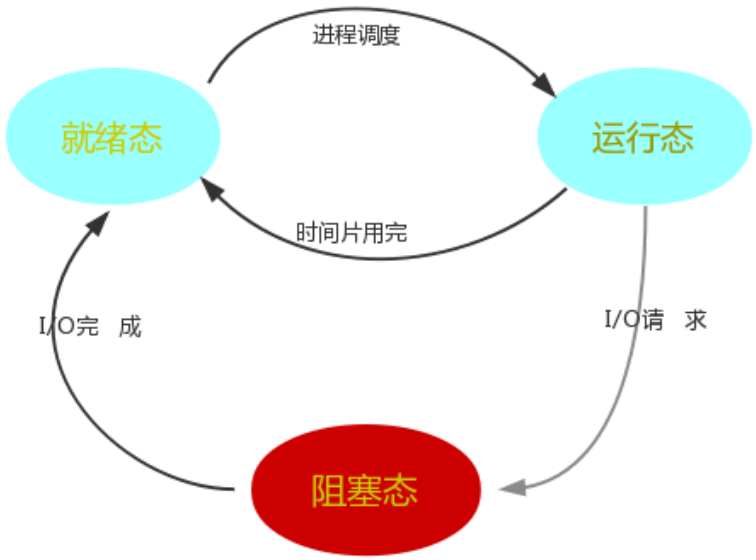
存在的：

1. 就绪态---->运行态：其他进程时间片用完，CPU空闲时被调度选中一个就绪进程执行。
2. 运行态---->就绪态：分配给每个进程的时间片是有限的，运行时间片到了就进入到就绪状态，或出现有更高优先权进程。
3. 运行态---->等待(阻塞)态：正在执行的进程因发生某等待事件而无法执行，则进程由执行状态变为阻塞状态，如发生了I/O请求(等待外设传输)。
4. 等待(阻塞)态---->就绪态：进程所等待的事件已经发生，就进入就绪队列。

还有两种不可能存在的转换：

1. 等待(阻塞)态---->运行态：即使给阻塞进程分配CPU，也无法执行，操作系统在进行调度时不会从阻塞队列进行挑选，而是从就绪队列中选取。
2. 就绪态---->等待(阻塞)态：就绪态根本就没有执行，谈不上进入等待态。

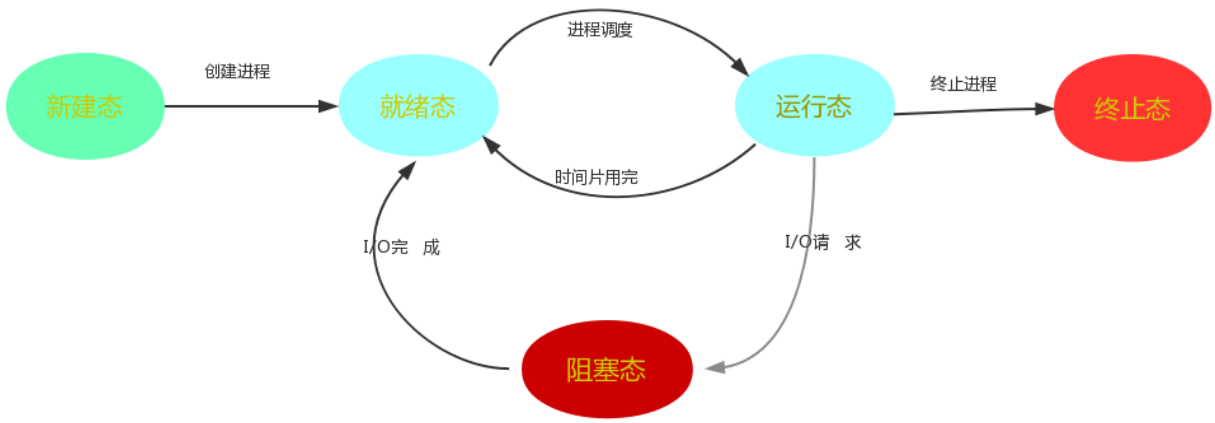
三态之间转换如图：



1.4 五态模型

对于一个实际的系统，进程的状态及其转换更为复杂。引入**新建态**和**终止态**构成了进程的五态模型。

这五种状态就如下图所示：



这五种状态详细说明如下：

- **新建状态**：进程在创建时需要申请一个空白进程管理块，向其中填写控制和管理进程的信息，完成资源分配。如果创建工作无法完成，比如资源无法满足，就无法被调度运行，把此时进程所处状态称为创建状态。创建进程时分为两个阶段，第一个阶段为一个新进程创建必要的管理信息，第二个阶段让该进程进入就绪状态。由于有了新建态，操作系统往往可以根据系统的性能和主存容量的限制推迟新建态进程的提交。

- **就绪状态**：进程已经准备好，已分配到所需资源，只要分配到CPU就能够立即运行。进程这时的状态称为就绪状态。在一个系统中处于就绪状态的进程可能有多个，通常将它们排成一个队列，称为就绪队列。例如，当一个进程由于时间片用完而进入就绪状态时，排入低优先级队列；当进程由I / O操作完成而进入就绪状态时，排入高优先级队列。
- **运行状态**：进程已获得CPU，其程序正在执行。在单核系统中，只有一个进程处于执行状态； 在多核机系统中，则有多个进程处于执行状态。在没有其他进程可以执行时（如所有进程都在阻塞状态），通常会自动执行系统的空闲进程。
- **等待状态**：正在执行的进程由于某些事件而暂时无法运行，CPU便暂时不再处理，进程处于暂停状态，也就是进程受到阻塞，这种暂停状态称为阻塞状态，有时也称为等待状态。致使暂停的事件有申请缓冲空间失败，I/O请求等。通常将这种处于阻塞状态的进程也排成一个队列。有的系统则根据阻塞原因的不同而把处于阻塞状态的进程排成多个队列。在再满足请求时再进入就绪状态。
- **终止状态**：进程结束，或出现错误，或被系统终止，进入终止状态。等待操作系统进行善后处理，然后将其进程管理块清零，就无法再执行。类似的，进程的终止也可分为两个阶段，第一个阶段等待操作系统进行善后处理，第二个阶段释放主存。

1.5 为什么引入五态模型呢

引入新建态和终止态对于进程管理来说是非常有用的。新建态对应于进程刚刚被创建的状态，创建进程要通过两个步骤，第一，是为新进程创建进程管理块；第二，让该进程进入就绪态。此时进程将处于新建态，它并没有被提交执行，而是在等待操作系统完成创建进程的必要操作。必须指出的是，操作系统有时将根据系统性能或主存容量的限制推迟新建态进程的提交。

类似地，进程的终止也要通过两个极端，第一，是等待操作系统进行善后处理；第二，释放内存。当一个进程到达了自然结束点，或是出现了无法克服的错误，或是被操作系统所终结，或是被其他有终止权的进程所终结，它将进入终止态。进入终止态的进程以后不再执行，但依然保留在操作系统中等待善后。一旦其他进程完成了对终止态进程的信息抽取之后，操作系统将删除该进程。引起进程状态转换的具体原因如下：

1. NULL---->新建态: 执行程序，创建一个子进程。
2. 新建态---->就绪态:当操作系统完成了进程创建的必要操作，并且当前系统的性能和内存的容量均允许。
3. 运行态---->终止态:当进程到达程序结点，或是出现了无法克服的错误，亦或是被操作系统所终结，又或者被其他有终止权的进程所终结。
4. 终止态---->NULL:完成善后操作，释放内存。
5. 就绪态---->终止态:但某些操作系统允许父进程终结子进程。
6. 等待(阻塞)态---->终止态:但某些操作系统允许父进程终结子进程。

第二章 JVM线程

2.1 线程

线程：进程内部的一个独立执行单元；一个进程可以同时并发的运行多个线程，可以理解为一个进程便相当于一个单 CPU 操作系统，而线程便是这个系统中运行的多个任务。

进程与线程的区别

- 进程：有独立的内存空间，进程中的数据存放空间（堆空间和栈空间）是独立的，至少有一个线程。
- 线程：堆空间是共享的，栈空间是独立的，线程消耗的资源比进程小的多。

注意：

1. 因为一个进程中的多个线程是并发运行的，那么从微观角度看也是有先后顺序的，哪个线程执行完全取决于 CPU 的调度，程序员是干涉不了的。而这也就造成的多线程的随机性。

2. Java 程序的进程里面至少包含两个线程，主进程也就是 main()方法线程，另外一个是垃圾回收机制线程。每当使用 java 命令执行一个类时，实际上都会启动一个 JVM，每一个 JVM 实际上就是在操作系统中启动了一个线程，java 本身具备了垃圾的收集机制，所以在 Java 运行时至少会启动两个线程。
3. 由于创建一个线程的开销比创建一个进程的开销小的多，那么我们在开发多任务运行的时候，通常考虑创建多线程，而不是创建多进程。

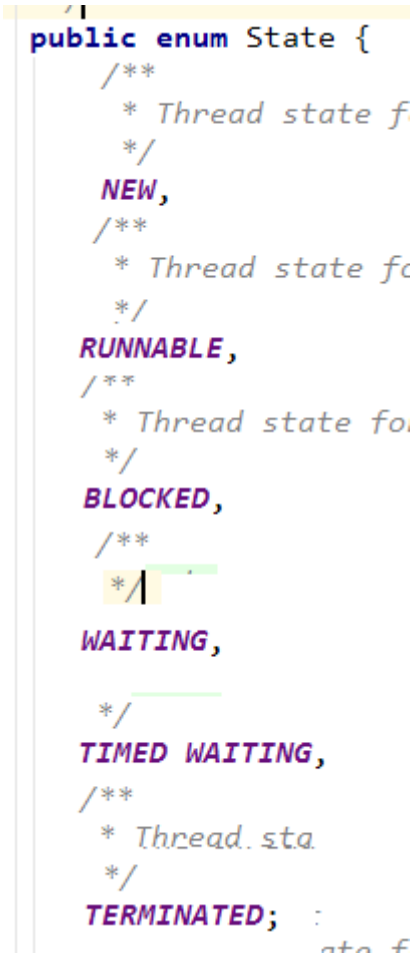
线程调度:

计算机通常只有一个CPU时,在任意时刻只能执行一条计算机指令,每一个进程只有获得CPU的使用权才能执行指令。所谓多进程并发运行,从宏观上看,其实是各个进程轮流获得CPU的使用权,分别执行各自的任务。那么,在可运行池中,会有多个线程处于就绪状态等到CPU,JVM就负责了线程的调度。JVM采用的是**抢占式调度**,没有采用分时调度,因此可以能造成多线程执行结果的的随机性。

2.2 官方线程状态

当线程被创建并启动以后，它既不是一启动就进入了执行状态，也不是一直处于执行状态。在线程的生命周期中，有几种状态呢？在API中`java.lang.Thread.State`这个枚举中给出了六种线程状态：

源码：



很明显，在`Thread`类中的 枚举类型`Thread.State`中明确的定义了6中线程状态。

API中介绍如下：

- `public static enum Thread.State extends Enum<Thread.State>`

  线程状态。

线程可以处于以下状态之一：

- `NEW`
  尚未启动的线程处于此状态。
- `RUNNABLE`
  在Java虚拟机中执行的线程处于此状态。
- `BLOCKED`
  被阻塞等待监视器锁定的线程处于此状态。
- `WAITING`
  正在等待另一个线程执行特定动作的线程处于此状态。
- `TIMED_WAITING`
  正在等待另一个线程执行动作达到指定等待时间的线程处于此状态。
- `TERMINATED`
  已退出的线程处于此状态。

一个线程可以在给定时间点处于一个状态。 这些状态是不反映任何操作系统线程状态的虚拟机状态。

- - 从以下版本开始：

    1.5

  - 另请参见：

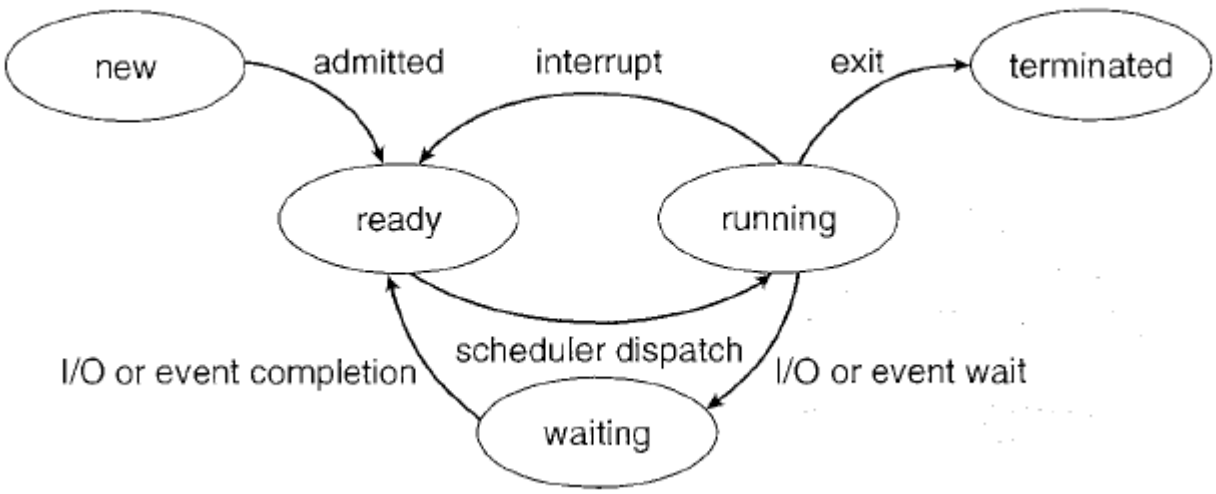
    `Thread.getState()`

由此可以看出这六种状态的说法是来源于JDK1.5开始，并且给出了一个提示在Thread里有个静态方法`Thread.getState()`方法可以获取线程的状态。

以上都是官方给出关于Thread线程的说明。

2.3 民间流传

网上流传了很久的线程具备5种状态，这样是不贴切JDK中描述的，JDK中描述线程状态只有6种，而网络流传的5种状态就是我们之前讲的进程的五态模型。那张广为流传的来自网络的图如下：



很明显这是操作系统中进程的5种状态，在很多操作系统书中也由介绍分别为new，ready，running，waiting，terminated。不幸的是，有很多的书上常常把这些进程状态，线程状态与Java线程状态混在一起谈。

当然，Java线程在Windows及Linux平台上的实现方式，是内核线程的实现方式。**这样的方式实现的线程，是直接由操作系统内核支持的——由内核完毕线程切换，内核通过操纵调度器（Thread Scheduler）实现线程调度，并将线程任务反映到各个处理器上。**

> 因为Java线程与操作系统存在内核映射关系，很多人就混为一谈了，这样是不科学的。映射，可以是一一映射，也可以是1对多，多对多映射，谁说一定是一对一了。

操作系统自然有它自己的线程状态，JVM有6种，Linux有7种，到了其他系统，还可能编程了N种。

> 不同的操作系统，运行系统对线程的控制以及生命周期都有自己的规则。

不管其他操作系统如何，我们在JAVA中线程就只有6种，阅读下面这句话：

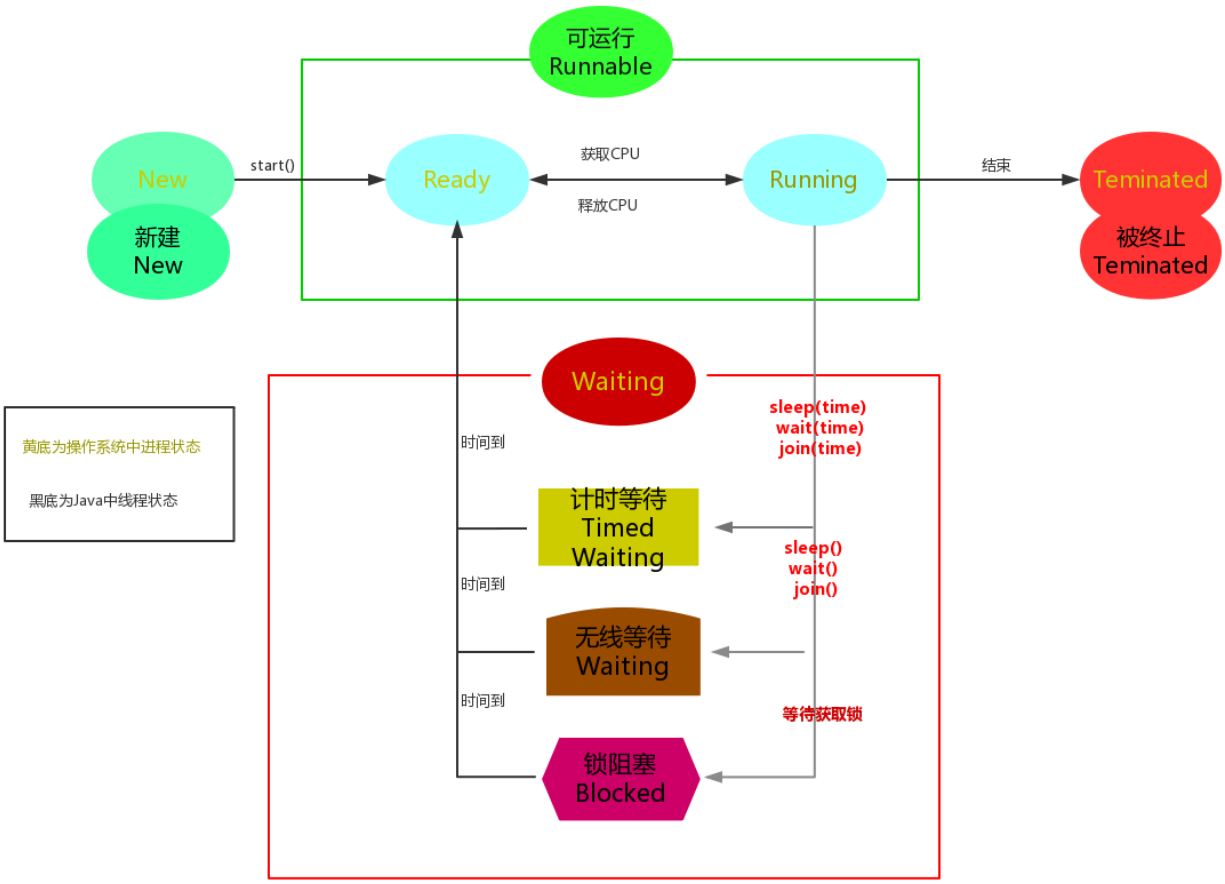
一个线程可以在给定时间点处于一个状态。 这些状态是不反映任何操作系统线程状态的虚拟机状态。

很明显，JVM中规定的线程状态不受其他操作系统影响。

现在看来网上说的五种线程状态，其实就是所谓“**进程状态**”指早期的那种“**单线程进程**”的状态。

对于现在普遍的“多线程进程”，显然，准确的说应该是某个进程状态下的线程状态，或者直接成为线程状态。它们之间还是有交集的，本次公开课就先分说线程状态，最后在跟系统层面的进程状态进行结合，达到对线程更完美的理解。

进程与线程的区分总图：



> 一些小问题
>
> 1：很多人觉得在JVM线程中应该有，Running运行状态。对JAVA而言，Runnable包含了就绪与运行，那为什么JAVA不区分开呢？这跟CPU分配的时间片有关，而且JAVA进行的是抢占式轮转调度，由于我们的JVM线程是服务于监控，线程又是切换的如此之快，那么区分ready与running又没有多大意义了。

> 再者，我们都知道现在使用的很多JVM底层都将线程映射到操作系统上了，JVM本身没有做什么调度，因为虚拟机看到的都是底层的映射与封装，故而将ready与running映射来也没有太大意义，不如统一为Runnable
>
> 2：总之还是有些乱的，我们不妨就拿Windows系统为例，用的就是"进程"和"线程"这两种较为标准的叫法，这时一个进程下至少有一个线程，线程是CPU调度的基本单位，进程不参与CPU调度，CPU根本不知道进程的存在。
>
> 3 : 为了避免混乱，下面说的线程状态，只是站在JVM层面上。

我们先来看下

这里先列出各个线程状态发生的条件，下面将会对每种状态进行详细解析

线程状态	导致状态发生条件
NEW(新建)	线程刚被创建，但是并未启动。
Runnable(可运行)	线程可以在java虚拟机中运行的状态，可能正在运行自己代码，也可能没有，这取决于操作系统处理器。
Blocked(锁阻塞)	当一个线程试图获取一个对象锁，而该对象锁被其他的线程持有，则该线程进入Blocked状态；当该线程持有锁时，该线程将变成Runnable状态。
Waiting(无限等待)	一个线程在等待另一个线程执行一个（唤醒）动作时，该线程进入Waiting状态。进入这个状态后是不能自动唤醒的，必须等待另一个线程调用notify或者notifyAll方法才能够唤醒。
Timed Waiting(计时等待)	同waiting状态，有几个方法有超时参数，调用他们将进入Timed Waiting状态。这一状态将一直保持到超时期满或者接收到唤醒通知。带有超时参数的常用方法有Thread.sleep 、Object.wait。
Teminated(被终止)	因为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。

我们不需要去研究这几种状态的实现原理，我们只需知道在做线程操作中存在这样的状态。那我们怎么去理解这几个状态呢，新建与被终止还是很容易理解的，我们就研究一下线程从Runnable（可运行）状态与非运行状态之间的转换问题。

第三章 新建与被终止线程

我们先来研究一下新建状态，也就是NEW，顾名思义，就是new 线程对象。

```
Thread state for a thread which has not yet started.
```

这句话的意思就是说这个线程状态是在线程创建之后，还未启动的时候，属于新建状态。

那怎么创建新的线程，有怎么启动呢？我们先来研究下线程的开启方式：

API中有提到：

> 创建一个新的执行线程有两种方法。 一个是将一个类声明为Thread的子类。 另一种方法来创建一个线程是声明实现类Runnable接口。

那我们就用过代码先来完成线程的创建，再来研究线程的状态。

3.1 创建线程方式一

Java使用`java.lang.Thread`类代表**线程**，所有的线程对象都必须是Thread类或其子类的实例。每个线程的作用是完成一定的任务，实际上就是执行一段程序流即一段顺序执行的代码。Java使用线程执行体来代表这段程序流。Java中通过继承Thread类来**创建**并**启动多线程**的步骤如下：

1. 定义Thread类的子类，并重写该类的run()方法，该run()方法的方法体就代表了线程需要完成的任务,因此把run()方法称为线程执行体。
2. 创建Thread子类的实例，即创建了线程对象
3. 调用线程对象的start()方法来启动该线程

代码如下：

测试类：

```
public class Demo01 {
    public static void main(String[] args) {
        //创建自定义线程对象
        MyThread mt = new MyThread("新的线程！");
       // 创建线程对象之后 启动之前
       System.out.println("线程状态为："+mt.getState());
        //开启新线程
        mt.start();
        //在主方法中执行for循环
        for (int i = 0; i < 10; i++) {
            System.out.println("main线程！"+i);
        }
    }
}
```

自定义线程类：

```
public class MyThread extends Thread {
    //定义指定线程名称的构造方法
    public MyThread(String name) {
        //调用父类的String参数的构造方法，指定线程的名称
        super(name);
    }
    /**
     * 重写run方法，完成该线程执行的逻辑
     */
    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(getName()+"：正在执行！"+i);
        }
    }
}
```

3.2 创建线程方式二

采用`java.lang.Runnable`也是非常常见的一种，我们只需要重写run方法即可。

步骤如下：

1. 定义Runnable接口的实现类，并重写该接口的run()方法，该run()方法的方法体同样是该线程的线程执行体。
2. 创建Runnable实现类的实例，并以此实例作为Thread的target来创建Thread对象，该Thread对象才是真正的线程对象。
3. 调用线程对象的start()方法来启动线程。

代码如下：

```
public class MyRunnable implements Runnable{
    @Override
    public void run() {
        for (int i = 0; i < 20; i++) {
            System.out.println(Thread.currentThread().getName()+" "+i);
        }
    }
}
```

```
public class Demo {
    public static void main(String[] args) {
        //创建自定义类对象  线程任务对象
        MyRunnable mr = new MyRunnable();
        //创建线程对象
        Thread t = new Thread(mr, "小强");
        t.start();
        for (int i = 0; i < 20; i++) {
            System.out.println("旺财 " + i);
        }
    }
}
```

3.3 查看线程状态

接下来 我们就在代码中测试线程的状态

```
public class Demo01 {
    public static void main(String[] args) {
        //创建自定义线程对象
        MyThread mt = new MyThread("新的线程！");
        // 创建线程对象之后 启动之前
        System.out.println("线程状态为："+mt.getState());
        //开启新线程
        mt.start();
        //在主方法中执行for循环
        for (int i = 0; i < 10; i++) {
            System.out.println("main线程！"+i);
        }
    }
}
```

```
public class MyThread extends Thread {
    //定义指定线程名称的构造方法
    public MyThread(String name) {
        //调用父类的String参数的构造方法，指定线程的名称
        super(name);
    }
    /**
     * 重写run方法，完成该线程执行的逻辑
     */
    @Override
    public void run() {
        System.out.println("线程状态为："+mt.getState());
        for (int i = 0; i < 10; i++) {
            System.out.println(getName()+"：正在执行！"+i);
        }
    }
}
```

这里强调两点：

- 线程对象创建之后，还未开启时候，就处于NEW的状态。
- 开启线程，指的是调用start方法，并不是run方法，run方法仅仅作为一个普通方法存在。

> 线程对象调用run()方法不开启线程，仅是对象调用方法。线程对象调用start()方法开启线程，并让jvm调用run()方法在开启的线程中执行。

3.4 多线程原理

昨天的时候我们已经写过一版多线程的代码，很多同学对原理不是很清楚，那么我们今天先画个多线程执行时序图来体现一下多线程程序的执行流程。

代码如下：

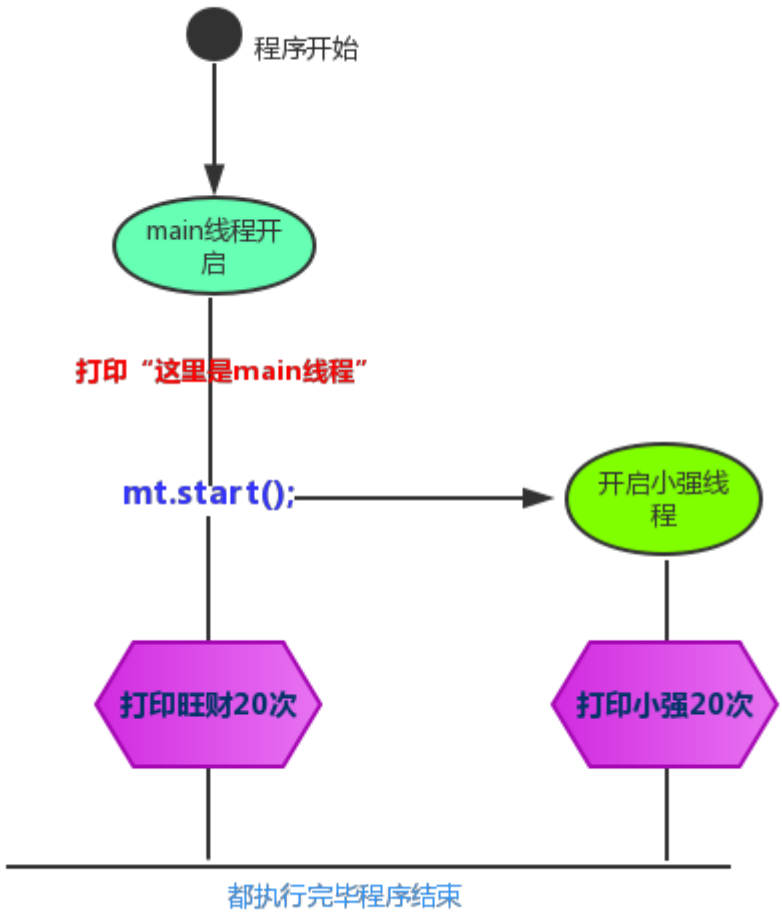
自定义线程类：

```
public class MyThread extends Thread{
    /*
     * 利用继承中的特点
     *   将线程名称传递  进行设置
     */
    public MyThread(String name){
        super(name);
    }
    /*
     * 重写run方法
     *  定义线程要执行的代码
     */
    public void run(){
        for (int i = 0; i < 20; i++) {
            //getName()方法 来自父亲
            System.out.println(getName()+i);
        }
    }
}
```

测试类：

```
public class Demo {
    public static void main(String[] args) {
        System.out.println("这里是main线程");
        MyThread mt = new MyThread("小强");
        mt.start();//开启了一个新的线程
        for (int i = 0; i < 20; i++) {
            System.out.println("旺财:"+i);
        }
    }
}
```

流程图：

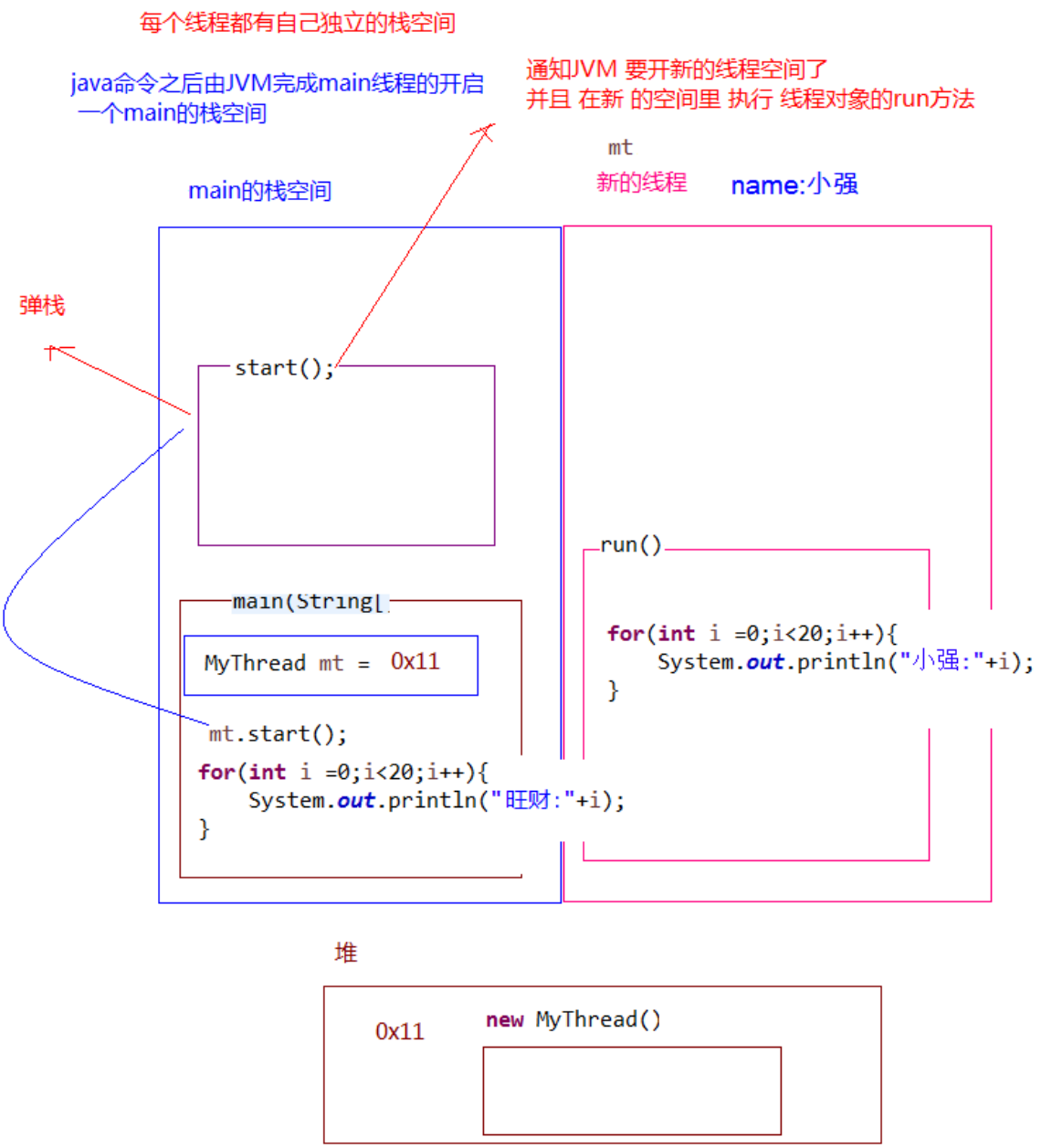


程序启动运行main时候，java虚拟机启动一个进程，主线程main在main()调用时候被创建。随着调用mt的对象的start方法，另外一个新的线程也启动了，这样，整个应用就在多线程下运行。

通过这张图我们可以很清晰的看到多线程的执行流程，那么为什么可以完成并发执行呢？我们再来讲一讲原理。

多线程执行时，到底在内存中是如何运行的呢？以上个程序为例，进行图解说明：

多线程执行时，在栈内存中，其实**每一个执行线程都有一片自己所属的栈内存空间**。进行方法的压栈和弹栈。



当执行线程的任务结束了，线程自动在栈内存中释放了。但是当所有的执行线程都结束了，那么进程就结束了。

通过学习多线程的原理，就知道什么时候处于NEW与TERMINATED两种状态了。

- NEW：当线程对象创建之后，还没有开始执行就处于NEW的状态，一旦执行了start方法，那么就进入RUNNABLE状态，之后还可能继续转换成其他状态。
- TERMINATED：终止状态，也没什么好说的，完成了执行后，或者说是退出了。线程就进入终止状态。