链滴

Java 并发编程之多线程基础

作者：wubo8196

原文链接：https://ld246.com/article/1583803575476

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

并发编程线程基础

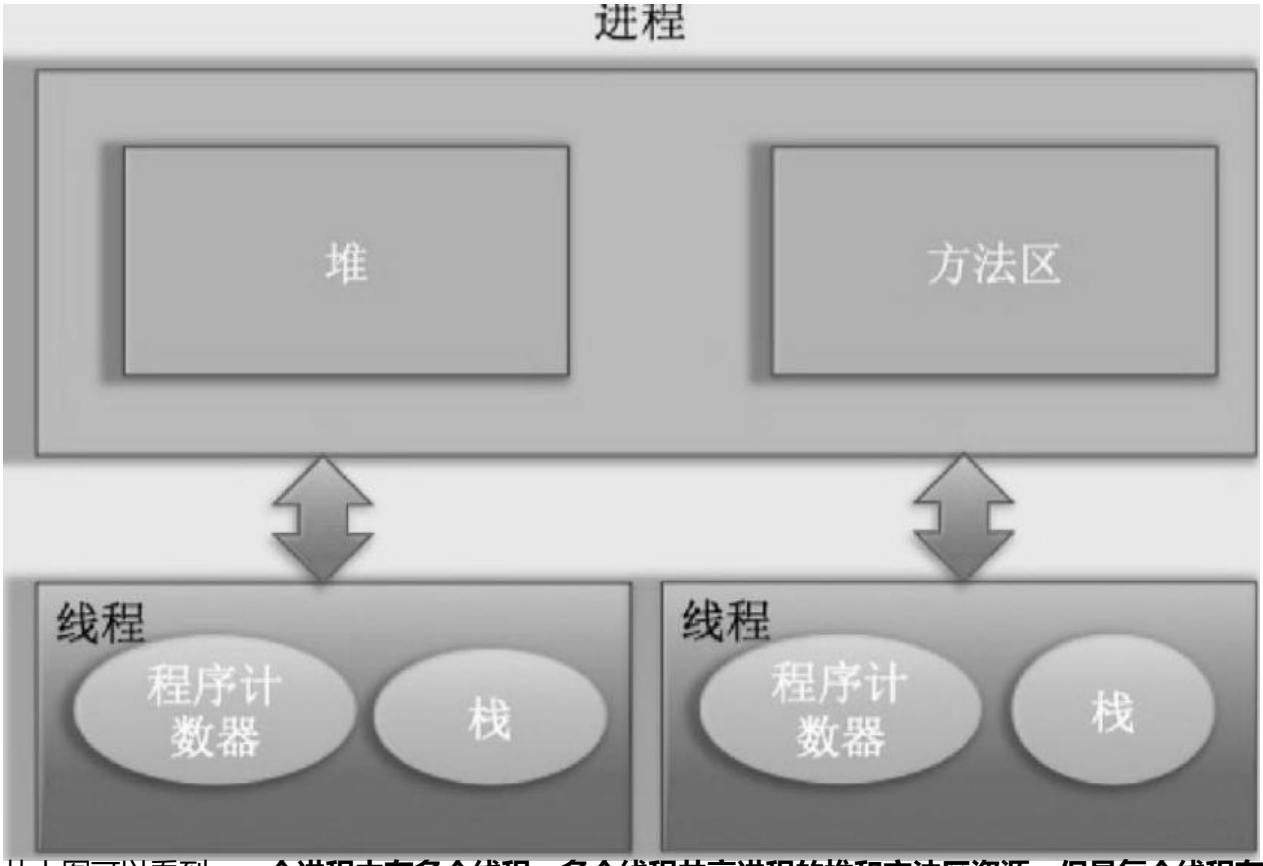
1.1 什么是线程

此处先了解一下进程，线程是进程中的一个实体，线程本身是不会独立存在的。进程是代码在数据集上的一次运行活动，是系统进行资源分配和调度的基本单位，线程则是进程的一个执行路径，一个进中至少有一个线程，进程中的多个线程共享进程的资源。

操作系统在分配资源时是把资源分配给进程的，但是 CPU 资源比较特殊，它是被分配到线程的，因真正要占用 CPU 运行的是线程，所以也说线程是 CPU 分配的基本单位。

在 Java 中，当我们启动 main 函数时其实就启动了一个 JVM 的进程，而 main 函数所在的线程就是个进程中的一个线程，也称主线程。

进程和线程的关系如图所示



从上图可以看到，**一个进程中有多个线程，多个线程共享进程的堆和方法区资源，但是每个线程有自的程序计数器和栈区域**

程序计数器是一块内存区域，用来记录线程当前要执行的指令地址。 记录该线程让出 CPU 时的执行址的，待再次分配到时间片时线程就可以从自己私有的计数器指定地址继续执行。需要注意的是，如执行的是 native 方法，那么 pc 计数器记录的是 undefined 地址，只有执行的是 Java 代码时 pc 计器记录的才是下一条指令的地址。

每个线程都有自己的栈资源，用于存储该线程的局部变量，这些局部变量是该线程私有的，其他线程访问不了的，除此之外栈还用来存放线程的调用栈帧。堆是一个进程中最大的一块内存，堆是被进程的所有线程共享的，是进程创建时分配的，堆里面主要存放使用 new 操作创建的对象实例。方法区用来存放 JVM 加载的类、常量及静态变量等信息，也是线程共享的。

方法区则用来存放 JVM 加载的类、常量及静态变量等信息，也是线程共享的。

1.2 线程的创建与运行

Java 中有以下三种线程创建方式：

- 实现 Runnable 接口的 run 方法
- 继承 Thread 类并重写 run 的方法
- 使用 FutureTask 方式。

继承 Thread 类方式的实现

```
public class ThreadTest {
    //继承Thread类并重写run方法
    public static class MyThread extends Thread {
        @Override
        public void run() {
            System.out.println("I am a child thread");
        }
    }
    public static void main(String[] args) {
        // 创建线程
        MyThread thread = new MyThread();
        // 启动线程
        thread.start();
    }
}
```

如上代码中的 MyThread 类继承了 Thread 类，并重写了 run（）方法。在 main 函数里面创建了一 MyThread 的实例，然后调用该实例的 start 方法启动了线程。

需要注意的是，当创建完 thread 对象后该线程并没有被启动执行，直到调用了 start 方法后才真正动了线程。其实调用 start 方法后线程并没有马上执行而是处于就绪状态，这个就绪状态是指该线程经获取了除 CPU 资源外的其他资源，等待获取 CPU 资源后才会真正处于运行状态。一旦 run 方法行完毕，该线程就处于终止状态。

使用继承方式的好处是，在 run（）方法内获取当前线程直接使用 this 就可以了，无须使用 Thread.c rrentThread（）方法；不好的地方是 Java 不支持多继承，如果继承了 Thread 类，那么就不能再继其他类。另外**任务与代码没有分离**，当多个线程执行一样的任务时需要多份任务代码，而 Runable 没有这个限制。

实现 Runnable 接口的 run 方法方式。

```
    public static class RunableTask implements Runnable{
        @Override
        public void run() {
            System.out.println("I am a child thread");
        }
    }
  public static void main(String[] args) throws InterruptedException{
        RunableTask task = new RunableTask();
```

```
        new Thread(task).start();
        new Thread(task).start();
    }
```

如上面代码所示，两个线程共用一个 task 代码逻辑，如果需要，可以给 RunableTask 添加参数进行务区分。另外，RunableTask 可以继承其他类。但是上面介绍的两种方式都有一个缺点，就是任务没返回值。下面看最后一种方式

使用 FutureTask 的方式。

```
//创建任务类，类似Runable
public static class CallerTask implements Callable<String>{
        @Override
        public String call() throws Exception {
            return "hello";
        }
    }
    public static void main(String[] args) throws InterruptedException {
    // 创建异步任务
        FutureTask<String> futureTask  = new FutureTask<>(new CallerTask());
        //启动线程
        new Thread(futureTask).start();
        try {
          //等待任务执行完毕，并返回结果
            String result = futureTask.get();
            System.out.println(result);
        } catch (ExecutionException e) {
            e.printStackTrace();
        }
}
```

如上代码中的 CallerTask 类实现了 Callable 接口的 call（）方法。在 main 函数内首先创建了一个 FtrueTask 对象（构造函数为 CallerTask 的实例），然后使用创建的 FutrueTask 对象作为任务创建一个线程并且启动它，最后通过 futureTask.get（）等待任务执行完毕并返回结果。

小结：使用继承方式的好处是方便传参，你可以在子类里面添加成员变量，通过 set 方法设置参数或通过构造函数进行传递，而如果使用 Runnable 方式，则只能使用主线程里面被声明为 final 的变量不好的地方是 Java 不支持多继承，如果继承了 Thread 类，那么子类不能再继承其他类，而 Runabl 则没有这个限制。前两种方式都没办法拿到任务的返回结果，但是 Futuretask 方式可以。

1.3 线程的运行状态

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：

1. 初始(NEW)：新创建了一个线程对象，但还没有调用 start()方法。
2. 可运行(RUNNABLE)：Java 虚拟机中将就绪（ready）和运行中（running）两种状态笼统的为“运行”。线程对象创建后，其他线程(比如 main 线程）调用了该对象的 start()方法。等待被线调度选中，获取 CPU 的使用权，此时处于就绪状态（ready）。就绪状态的线程在获得 CPU 时间片变为运行中状态（running）。
3. 阻塞(BLOCKED)：表示线程阻塞于锁。

4. 等待(WAITING)：进入该状态的线程需要等待其他线程做出一些特定动作（通知或中断）。
5. 超时等待(TIMED_WAITING)：该状态不同于 WAITING，它可以在指定的时间后自行返回。
6. 终止(TERMINATED)：该线程已经执行完毕而退出，或者因异常终止。

线程状态间的切换：