链滴

Java 多线程介绍

作者：RustFisher

原文链接：https://ld246.com/article/1631001736567

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)

运行环境与工具

- jdk1.8.0
- macOS 11.4
- IDEA

操作系统可以在同一时刻运行多个程序。例如一边播放音乐，一边下载文件和浏览网页。操作系统将cpu的时间片分配给每一个进程，给人一种并行处理的感觉。

一个多线程程序可以同时执行多个任务。通常，每一个任务称为一个线程（thread），它是线程控制简称。可以同时运行一个以上线程的程序成为多线程程序（multithreaded）。

多进程和**多线程**有哪些区别呢?

本质区别在于进程每个进程有自己的一整套变量，而线程则共享数据。

线程比进程更轻量级，创建、销毁一个线程比启动新进程的开销要小。

实际应用中，多线程非常有用。例如应用一边处理用户的输入指令，一遍联网获取数据。

本文我们介绍Java中的**Thread**类。

Thread

Thread类属于java.lang包。

要创建一个线程很简单，新建一个**Thread**对象，并传入一个**Runnable**，实现run()方法。

调用start()方法启动线程。

```
Thread t1 = new Thread(new Runnable() {
    @Override
    public void run() {
        System.out.println("rustfisher said: hello");
    }
});
t1.start();
```

Java lambda

```
Thread t1 = new Thread(() -> System.out.println("rustfisher said: hello"));
t1.start();
```

不要直接调用run()方法

直接调用run()方法不会启动新的线程，而是直接在当前线程执行任务。

我们来看一个使用了Thread.sleep()方法的例子。

```
Thread t1 = new Thread(() -> {
    for (String a : "rustfisher said: hello".split("")) {
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        }
        System.out.print(a);
    }
});
t1.start();
```

sleep(int)方法会让线程睡眠一个指定的时间（单位毫秒）。并且需要try-catch捕获InterruptedException异常。

中断线程

当run()方法执行完最后一条语句后，或者return，或者出现了未捕获的异常，线程将会终止。

使用Thread的interrupt方法也可以终止线程。调用interrupt方法时，会修改线程的中断状态为true。用isInterrupted()可以查看线程的中断状态。

但如果线程被阻塞了，就没法检测中断状态。当在一个被阻塞的线程（sleep或者wait）上调用interrupt方法，阻塞调用将会被InterruptedException中断。

被中断的线程可以决定如何响应中断。可以简单地将中断作为一个终止请求。比如我们主动捕获InterruptedException。

```
Thread t2 = new Thread(() -> {
    for (String a : "rustfisher said: hello".split("")) {
        try {
            Thread.sleep(200);
        } catch (InterruptedException e) {
            e.printStackTrace();
            System.out.println("被中断 退出线程");
            return;
        }
        System.out.print(a);
    }
});
t2.start();

new Thread(() -> {
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    t2.interrupt();
}).start();
```

上面这个小例子展示了用interrupt()来中断线程t2。而线程t2的run()方法中捕获InterruptedException，可以进行自己的处理。

线程的状态

线程有6种状态，用枚举类**State**来表示：

- NEW（新创建）
- RUNNABLE（可运行）
- BLOCKED（被阻塞）
- WAITING（等待）
- TIMED_WAITING（计时等待）
- TERMINATED（被终止）

用getState()方法可以获取到线程的状态。

新创建线程

new一个线程的时候，线程还没开始运行，此时是NEW（新创建）状态。在线程可以运行前，还有一工作要做。

可运行线程

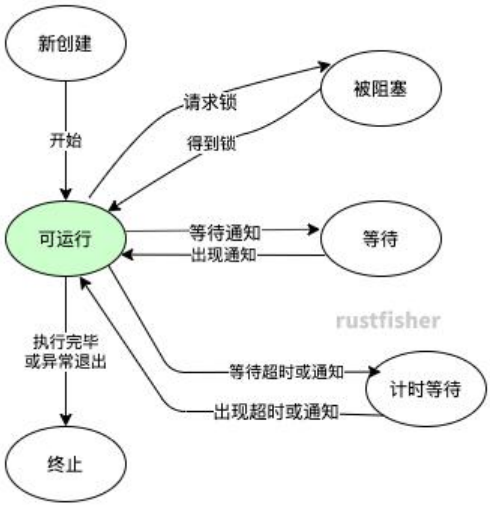
一旦调用start()方法，线程处于RUNNABLE（可运行）状态。调用start()后并不保证线程会立刻运行而是要看操作系统的安排。

一个线程开始运行后，它不一定时刻处于运行状态。操作系统可以让其他线程获得运行机会。一个可行的线程可能正在运行也可能没在运行。

被阻塞和等待

线程处于被阻塞和等待状态时，它暂时不活动。不运行代码，且只消耗最少的资源。直到线程调度器新激活它。

- 一个线程试图获取一个内部的对象锁，而该锁被其他线程持有，则这个线程进入阻塞状态。当这个被释放，并且线程调度器允许这个线程持有它，该线程变成非阻塞状态。
- 当线程等待另一个线程通知调度器，它自己进入等待状态。例如调用 Object.wait()或者Thread.join()方法。
- 带有超时参数的方法可让线程进入超时等待状态。例如 Thread.sleep()，Object.wait(long)，Thread.join(long)，Lock.tryLock(long time, TimeUnit unit)



上面这个图展示了状态之间的切换。

被终止

终止的原因：

- run方法正常退出
- 出现了没有捕获的异常而终止了run方法

线程属性

线程优先级，守护线程，线程组以及处理未捕获异常的处理器。

线程优先级

Java中每个线程都有一个优先级。默认情况下，线程继承它的父线程的优先级。

可用setPriority(int)方法设置优先级。优先级最大为MAX_PRIORITY = 10，最小为MIN_PRIORITY = 1，普通的是NORM_PRIORITY = 5。

线程调度器有机会选新线程是，会优先选高优先级的线程。

守护线程

调用setDaemon(true)可以切换为守护线程（daemon thread）。守护线程的用途是为其他线程提服务。例如计时线程。

当只剩下守护线程是，虚拟机就退出了。

守护线程不应该去访问固有资源，如文件和数据库。

未捕获异常处理器

run()方法里抛出一个未捕获异常，在线程死亡前，异常被传递到一个用于未捕获异常的处理器。

要使用这个处理器，需要实现接口Thread.UncaughtExceptionHandler，并且用setUncaughtExceptonHandler(Thread.UncaughtExceptionHandler)方法把它交给线程。

```
Thread t3 = new Thread(() -> {
    try {
        Thread.sleep(5);
    } catch (InterruptedException e) {
        e.printStackTrace();
        return;
    }
    int x = 0, y = 3;
    int z = y / x; // 故意弄一个异常
});
t3.setUncaughtExceptionHandler(new Thread.UncaughtExceptionHandler() {
    @Override
    public void uncaughtException(Thread t, Throwable e) {
        System.out.println(t + "有未捕获异常");
        e.printStackTrace();
```

```
    }
});
t3.start();
```

运行后，run()方法里抛出ArithmeticException异常

```
Thread[Thread-0,5,main]有未捕获异常
java.lang.ArithmeticException: / by zero
    at Main.lambda$main$0(Main.java:15)
    at java.lang.Thread.run(Thread.java:748)
```

也可以用静态方法Thread.setDefaultUncaughtExceptionHandler(Thread.UncaughtExceptionHandler)给所有的线程安装一个默认处理器。可以在这个默认处理器里做一些工作，例如记录日志。

ThreadGroup代表着一组线程。也可以包含另外的线程组。

ThreadGroup类实现了**UncaughtExceptionHandler**接口。它的uncaughtException(Thread t, Throwable e)方法会有如下操作

- 如果该线程组有父线程组，则父线程组的 uncaughtException被调用。
- 否则，如果 Thread.getDefaultUncaughtExceptionHandler()返回一个非空处理器，则使用这个理器。
- 否则，如果抛出的 **Throwable**是**ThreadDeath**对象，就什么也不做。
- 否则，线程的名字和 **Throwable**的栈踪迹输出到System.err上。

参考书籍： Core Java, Volume II - Advanced Features 7th Edition

建议参考： Concurrent Programming in Java