Java多线程新手指南：从零开始学习多线程创建，有两下子！

```
环境说明：Windows 10 + IntelliJ IDEA 2021.3.2 + Jdk 1.8
```

前言

多线程已是众人眼里老生常谈的话题之一了，在日常项目开发中它也是最为熟知且常用的技术之一，毕竟它能够允许程序同时执行多个任务，从而提高程序的响应速度和资源利用率，顾名思义，就是让计算机的多个核心同时工作，就像多个人同时做不同的任务一样。

但在Java这种流行的编程语言中，多线程的使用尤为重要，因为它能帮助我们写出更高效、更流畅的程序，毕竟Java作为一种广泛应用于开发领域的语言，提供了强大的多线程支持，也提供了丰富的多线程支持，使得开发者可以轻松地实现并发编程。所以今天我继续来聊聊它，本文将重点介绍及演示如何使用Java来创建多线程，及衍生穿插知识点，带着大家能零基础轻松入门多线程编辑。

而在前两期中，我也着重的讲解了多线程相关基础知识点及线程生命周期，此期我就一笔带过了，需要重点恶补知识点的同学，可以看下我前两期总结的，都是实打实的干货，篇篇文字并茂，万字长文，一定能够帮助到你。

* 零基础学Java多线程控制：让你轻松掌握线程操作，有两下子！
* 零基础学Java多线程：深入理解线程生命周期，有两下子！

摘要

在Java多线程编程中，掌握线程的创建和管理是至关重要的。，本文将介绍Java中多线程的创建方法，包括继承Thread类和实现Runnable接口两种方式，并通过源代码解析、应用场景案例、优缺点分析等来详细讲解每一种方法的使用和特点，最后通过一个测试案例来完整的复盘一遍，理论+实践 = 百分百掌握。

概述
--

在多线程编程中，线程是程序中执行的独立单元。Java中线程的创建有两种常用的方式：继承`Thread`类和实现`Runnable`接口，这两种是最基础的创建方式之一，比如还有通过线程池的方式、创建定时等。其中，对于继承Thread类，我们需要重写`run()`方法，而实现Runnable接口，我们需要实现`run()`方法。针对如上两者，皆可创建线程对象并调用`start()`方法来启动线程，如下我就通过对此两种方式来零基础教学，并且也会对这两种方式进行优缺点分析，以便于同学们能够优先自我掌握。

为什么需要多线程?

可能很多刚入门的小伙伴就会产生疑问，为啥需要多线程？这个问题想必在座的各位曾经都有思考过，所以现在非常能够理解产生这些疑惑的同学们。这里我先就给大家进行分享下个人的理解，无妨大家可以这样想象一下，你在厨房里准备晚餐，如果只有你一个人，你可能需要先切菜，然后煮饭，接着炒菜；但如果你有帮手，你们可以同时进行这些任务，这样晚餐就能更快准备好。同样，在计算机程序中，如果我们能让不同的任务同时进行，程序就能更快地完成工作，也就是所谓的并行处理，同一时间多人进行，从而加快程序的执行速度。

多线程重要吗?

这点绝对是毋庸置疑的，它非常重要，为啥这么重要？？这里我接着给同学们分享。多线程就像是在厨房里多请了几个帮手，每个人都能同时做不同的活儿，这样饭菜就能更快上桌。在电脑程序里，多线程能让程序同时做很多事，就像多个人一起工作一样，效率自然就高了。

1. **提高效率**：想象一下，如果电脑里只有一个程序在运行，它就只能做一件事。但有了多线程，电脑就能像多核大脑一样，同时处理好几个任务，速度自然就快了。
2. **改善响应性**：就像你在玩游戏时，如果电脑还在下载东西，游戏可能会卡顿。但如果用多线程，下载和游戏可以同时进行，互不影响，游戏就能流畅多了。
3. **资源利用率**：多线程就像是把电脑的CPU和内存都用起来，不让它们闲着。这样，电脑的每个部分都能发挥最大作用。
4. **更好的用户体验**：用户就像是顾客，他们希望点菜后能快速上菜。多线程能让程序快速响应用户的操作，就像快速上菜一样，让顾客满意。
5. **并行处理**：有些任务可以分成很多小块，每块都可以同时处理。多线程

就像是有很多工人同时工作，这样完成任务的速度就快多了。
6. **简化设计**：有时候，程序设计得太复杂，就像厨房里有很多复杂的机器。多线程可以让程序设计更简单，就像用简单的工具就能做出好菜。
7. **避免阻塞**：如果一个任务卡住了，整个程序可能就会停下来。多线程可以避免这种情况，就像一个厨师忙不过来，其他厨师可以接手继续做。
8. **利用现代硬件**：现在的电脑越来越强大，多线程就像是让这些强大的电脑发挥出它们的最大能力。
9. **适应性**：在一些需要快速反应的场合，比如在线游戏或者股票交易，多线程能让程序更加灵活，快速适应变化。
10. **错误隔离**：如果程序中的一个部分出了问题，多线程可以保证其他部分不受影响，就像一个厨师出错了，其他厨师还可以继续工作。

总之，多线程就像是请了一群能干的帮手，让程序能更快、更顺畅、更稳定地运行。掌握了多线程，你的程序就能像高效的厨房一样，快速满足用户的需求。

Java中的多线程是如何工作的?

那么，它究竟是如何工作的呢？这点我们继续往下看。我们都知道Java提供了两种主要的方式来创建线程，这两种方式各有特点，适用于不同的应用场景：

* 继承Thread类：就像你创建一个新食谱，基于一个已有的食谱进行修改。在Java中，你可以通过创建一个新的类，继承自Thread类，并重写它的run方法来定义你的任务。
* 实现Runnable接口：这就像是按照食谱做菜，你不需要自己从头开始创造食谱，只需要按照已有的步骤来。在Java中，你可以创建一个实现了Runnable接口的类，并实现它的run方法，然后将其传递给Thread对象。

如何创建和启动线程?

那么，我们既然了解了线程是如何工作的，那么你们知道它是如何被创建及启动的呢？其实也很简单，创建和启动线程是多线程编程中的基础操作，它们确保了程序能够并行执行多个任务。以下是创建和启动线程的详细步骤，以及它们之间的逻辑联系：

1. **定义任务**：

* 在多线程编程中，每个线程都需要执行一个特定的任务。任务的定义通常通过编写一个`run`方法来实现。这个方法是线程执行的入口点，包含了线程要执行的所有操作。

2. **选择线程创建方式**：

* 根据你的程序需求，你可以选择继承`Thread`类或实现`Runnable`接口来定义你的任务。继承`Thread`类意味着你的类直接扩展了线程的功能，而实现`Runnable`接口则需要将你的任务类传递给一个`Thread`对象。
3. **创建线程对象**：

* 一旦你定义了任务，接下来就是创建线程对象。如果你选择继承`Thread`类，你将创建该类的实例；如果实现`Runnable`接口，你需要创建一个`Runnable`对象，并将该对象作为参数传递给`Thread`类的构造函数，然后创建`Thread`对象的实例。
4. **启动线程**：

* 创建线程对象后，你需要调用`start()`方法来启动线程。这个方法会触发线程的执行，使其进入就绪状态，并最终运行。重要的是要注意，`start()`方法会隐式地调用你的`run`方法，因此无需手动调用`run`。
5. **线程的执行**：

* 一旦线程启动，它将按照定义的`run`方法中的指令执行任务。线程的执行是由Java运行时环境的线程调度器控制的，它会根据系统的线程调度策略来决定何时执行哪个线程。
6. **线程的生命周期管理**：

* 在线程执行过程中，你需要了解线程的生命周期，包括新建、就绪、运行、阻塞和死亡状态。这有助于你更好地控制线程的执行流程，例如，通过适当的同步机制来处理线程间的协作或竞争。
7. **异常处理**：

* 在多线程环境中，异常处理尤为重要。确保你的`run`方法能够妥善处理可能发生的异常，避免线程因未捕获的异常而意外终止。
8. **线程的结束**：

* 线程完成任务后，应当正确结束。在`run`方法的末尾，线程会自然结束。如果需要提前结束线程，可以通过中断机制来实现。

通过上述步骤，你就可以轻松创建和启动线程，使它们能够并行执行任务，从而提高程序的效率和响应性。有一点需要注意，多线程编程需要仔细设计和测试，以确保线程间的同步和资源管理得当，避免出现竞态条件和死锁等问题，这点在日常使用多线程是非常有必要的。

线程的生命周期

线程也有对应的生命周期，就像人一样，有出生、成长、衰老和死亡的过程，只不过它的不是一模一样。

* 新建状态：线程被创建，但还没有开始工作。
* 就绪状态：线程准备好了，等待CPU的调度来执行。
* 运行状态：线程正在执行它的任务。
* 阻塞状态：线程因为某些原因（比如等待输入输出操作完成）暂时不能继续执行。
* 死亡状态：线程的任务执行完毕或者被强制停止。

需要恶补基础知识点的同学可以看我这篇： 零基础学Java多线程：深入理解线程生命周期，有两下子！

线程同步的重要性

当多个线程需要访问共享资源时，如果没有适当的同步机制，就可能出现问题，比如数据不一致。Java提供了多种同步机制，比如synchronized关键字，来确保多个线程不会同时修改同一个资源。

线程池有啥好处?

想象一下，如果你每次做饭都要买一套新的厨具，这将是多么浪费。线程池就像是共享厨具，你可以重复使用它们，这样可以节省资源，提高效率。

何为线程安全?

编写线程安全的代码意味着你的程序在多线程环境下也能正常工作，不会出现数据错乱或者程序崩溃的问题。

线程的高级特性

除了基本的线程操作，Java还提供了一些高级特性，如线程局部变量、中断处理和线程池等，这些特性可以进一步提高多线程程序的性能和可维护性。这些我们就日后再谈。

创建线程方式

那么，有了如上知识点的铺垫，那接下里，我就带着大家从零开始手撸代码，步步为营。

总所周知，在程序的世界里，线程是执行的独立单元，那么我们就来耍耍，它究竟要如何创建？其实我们压根不需要重复造轮子，我们只需要使用原生的就能满足。对于Java本身而言，它就提供了多种方式来创建线程，其中最基本的两种当属继承Thread类和实现Runnable接口。接下来，我们将通过零基础实战教学，逐步深入这两种方法的实现和运用。

方式1：继承Thread类

我们先来介绍第一种，通过定义一个类直接继承Thread类，然后重写run()即可方法即可，上手即会。示例代码如下：

```
/**
 * @Author bug菌
 * @Source 公众号：猿圈奇妙屋
 * @Date 2024-04-15 22:23
 */
public class MyThread extends Thread {

    @Override
    public void run() {
        // 线程执行的代码逻辑
        System.out.println("线程执行中...");
    }
}
```

代码解析：

如上代码我先定义了一个名为`MyThread`的线程类，继承Thread。这个类中我重写了Thread类中的`run()`方法，`run()`方法中包含了线程执行的代码逻辑，实际业务实际修改，我只是做演示。在这个例子中，`run()`方法只是简单地打

印出一条消息”线程执行中...”，而实际项目开发中，则是写对应的业务逻辑即可。

总言而之，通过创建`MyThread`的对象并调用`start()`方法，可启动一个新的线程，并执行其中的代码逻辑。

而如下我再讲一下我提到的另一种创建方式

方式2：实现Runnable接口

我们先来介绍第一种，通过定义一个类直接继承Thread类，然后重写run方法，即可。示例代码如下：

```
/**
 * @Author bug菌
 * @Source 公众号：猿圈奇妙屋
 * @Date 2024-04-15 22:24
 */
public class MyRunnable implements Runnable {

    @Override
    public void run() {
        // 线程执行的代码逻辑
        System.out.println(”线程执行中...”);
    }
}
```

代码解析：

此段代码与上个原理是不太相同，但实现目的是一致的，即创建线程。我先定义了一个名为`MyRunnable`的类，并且实现了`Runnable`接口。`Runnable`接口中只有一个`run()`方法需要实现。

在`MyRunnable`类中，重写了`run()`方法。在`run()`方法中，我们可以定义线程执行的代码逻辑。在这个例子中，我们简单地输出了一条信息”线程执行中...”，同理，对于实际项目开发中，则是写对应的业务逻辑即可。

这样，当我们将`MyRunnable`实例传给一个`Thread`对象，并调用`start()`方法启动线程时，线程就会执行`run()`方法中的代码逻辑。

案例演示

接下来，我们就运用上述理论知识，来实际战斗一波。

方式一：继承Thread类-创建线程

测试代码

示例代码如下：

```
/**
 * @Author bug菌
 * @Source 公众号：猿圈奇妙屋
 * @Date 2024-04-15 22:23
 */
public class MyThreadTest {

    public static void main(String[] args) {
        MyThread thread = new MyThread();
        thread.start();
    }
}
```

案例测试结果

根据如上的测试用例，作者在本地进行测试结果如下，仅供参考，你们也可以自行修改测试用例或者添加其他的测试数据或测试方法，以便于进行熟练学习以此加深知识点的理解。

案例测试代码分析

根据如上测试用例，在此我给大家进行深入详细的解读一下测试代码，以便于更多的同学能够理解并加深印象。

如上代码它是一个简单的多线程程序的示例。下面是对这段代码的分析，希望能帮助到你：

1. **类定义**：先定义一个名为 `MyThreadTest` 的公共类。在Java中，公共类意味着它可以被任何其他类访问。
2. **主方法**：`main` 方法是Java程序的入口点。它是一个静态方法，这意味着它属于类本身而不是类的实例。这个方法接受一个字符串数组 `args` 作为参数，这通常用于命令行参数。
3. **线程创建**：在 `main` 方法中，创建了一个 `MyThread` 类的实例，并且调用了它的 `start()` 方法。`MyThread` 类没有在这段代码中给出，但我们可以推断它是一个继承了 `Thread` 类或者实现了 `Runnable` 接口的类，因为它有一个 `start()` 方法。
4. **线程启动**：调用 `start()` 方法实际上会创建一个新的线程，并在这个新线程中执行 `MyThread` 类的 `run()` 方法。`run()` 方法是 `Thread` 类的一个抽象方法，必须在子类中实现。`MyThread` 类的 `run()` 方法中应该包含了线程执行的代码。
5. **多线程执行**：一旦 `start()` 被调用，`MyThread` 实例将在它自己的线程中并行执行，与主线程（即包含 `main` 方法的线程）同时运行。
6. **代码不完整**：这段代码只是一个框架，缺少 `MyThread` 类的具体实现。为了使程序完整，我们需要 `MyThread` 类的实现，其中至少包含一个重写的 `run()` 方法。
7. **线程的生命周期**：当 `main` 方法执行完毕，程序将结束。如果 `MyThread` 线程还没有执行完，它将继续运行，直到它的 `run()` 方法执行完毕。
8. **线程安全**：这段代码没有显示任何关于线程同步或线程安全的措施。如果 `MyThread` 类的 `run()` 方法中访问共享资源，那么可能需要考虑线程安全问题。
9. **错误处理**：代码中没有显示任何错误处理机制。在实际应用中，线程可能会抛出异常，因此可能需要捕获并适当处理这些异常。
10. **资源管理**：如果 `MyThread` 使用了需要关闭的资源（如文件、网络连接等），则需要确保这些资源被适当地关闭，无论是在 `run()` 方法的末尾还是在 `finally` 块中。

方式二：实现Runnable接口-创建线程

案例测试代码

示例代码如下：

```
/**
 * @Author bug菌
 * @Source 公众号：猿圈奇妙屋
 * @Date 2024-04-15 22:24
 */
public class MyRunnableTest {

    public static void main(String[] args) {
        MyRunnable runnable = new MyRunnable();
        Thread thread = new Thread(runnable);
        thread.start();
    }
}
```

案例测试结果

根据如上的测试用例，作者在本地进行测试结果如下，仅供参考，你们也可以自行修改测试用例或者添加其他的测试数据或测试方法，以便于进行熟练学习以此加深知识点的理解。

本地实际运行结果展示如下：

非常直观可以看到，从执行函数中执行了start方法，皆控制台成功输出打印了线程体中执行run方法的逻辑业务；输出了如下内容：

```
线程执行中...
```

```

#### 案例测试代码分析

根据如上测试用例，在此我给大家进行深入详细的捋一捋测试代码的全过程，以便于更多的同学能够加深印象并且把它吃透。

这段代码展示了如何使用 `Runnable` 接口来创建一个线程。以下是对这段代码案例的详细分析：

1. **类定义**：定义了一个名为 `MyRunnableTest` 的公共类。这个类包含了 `main` 方法，它是程序的入口点。
2. **创建 `Runnable` 实例**：在 `main` 方法中，首先创建了一个 `MyRunnable` 类的实例。`MyRunnable` 应该是一个实现了 `Runnable` 接口的类，这意味着它必须实现 `Runnable` 接口中的 `run` 方法。
3. **创建线程**：接着，使用 `MyRunnable` 实例作为目标，创建了一个新的 `Thread` 对象。`Thread` 构造函数接受一个 `Runnable` 对象作为参数，这个 `Runnable` 对象定义了线程要执行的任务。
4. **启动线程**：通过调用 `thread.start()` 方法，新的线程被启动。这会导致 `MyRunnable` 实例的 `run` 方法在新线程中执行。
5. **线程的并行执行**：一旦 `start()` 方法被调用，`MyRunnable` 的 `run` 方法将在它自己的线程中并行执行，与主线程（即执行 `main` 方法的线程）同时运行。
6. **代码的不完整性**：这段代码没有提供 `MyRunnable` 类的实现。为了使程序完整，我们需要 `MyRunnable` 类的实现，特别是 `run` 方法的实现。
7. **线程生命周期**：`main` 方法执行完毕后，如果 `MyRunnable` 线程还没有执行完，它将继续运行，直到 `run` 方法执行完毕。
8. **线程安全**：如果 `MyRunnable` 类的 `run` 方法中访问了共享资源，那么需要考虑线程安全问题，确保对这些资源的访问是同步的。
9. **错误处理**：这段代码没有显示任何错误处理机制。在实际应用中，`run` 方法可能会抛出异常，因此可能需要在 `run` 方法中添加异常处理逻辑。
10. **资源管理**：如果 `MyRunnable` 使用了需要关闭的资源，如文件、网络连接等，需要确保这些资源在使用完毕后被适当地关闭。
11. **线程的命名**：创建 `Thread` 对象时，可以通过构造函数传递一个字符串参数来为线程命名，这有助于在调试时识别线程。
12. **线程优先级**：可以通过设置线程的优先级来影响线程的调度，但这通常不是线程管理的首选方法。

这段代码是使用 `Runnable` 接口进行多线程编程的一个基本示例。通过这种方式，可以将任务的执行逻辑与线程的创建和管理分离，从而提高代码的可读性和可维护性。

#### 小结
```

那么针对上述两种创建线程的方式，它们之间有何优劣之分呢？其实是有的，这也需要具体分场景，相对而论。

优缺点分析

继承Thread类方式

优点：

* 简单直观，代码量较少。

缺点：

* 由于Java不支持多继承，如果已经继承了其他类，则无法再使用继承Thread类的方式创建线程。

实现Runnable接口方式

优点：

* 避免了单继承的限制，可以同时实现其他接口。
* 实现了解耦，线程对象与线程执行的逻辑分离。

缺点：

* 代码稍微复杂一些。

希望如上优劣分析，同学们在面对第一次学习的过程中，可以优先选择自己最先能掌握的方式，而不是一概而论，有条件的同学，则可以都掌握的熟练。

类代码方法介绍

如下我梳理了下针对如上两种创建线程的方式之间常用到的方法，仅供参考：

Thread类

* start(): 启动线程，使其开始执行run()方法中的代码。
* run(): 线程执行的代码逻辑，需要在子类中重写。

Runnable接口

* run(): 线程执行的代码逻辑，需要在实现类中实现。

测试用例

这里我再通过一个案例测试带着大家把上边的知识点串联起来，进行实战演示一波，同学们看好了。

测试代码

```
/**
 * @Author bug菌
 * @Source 公众号：猿圈奇妙屋
 * @Date 2024-04-15 22:37
 */
public class Test {

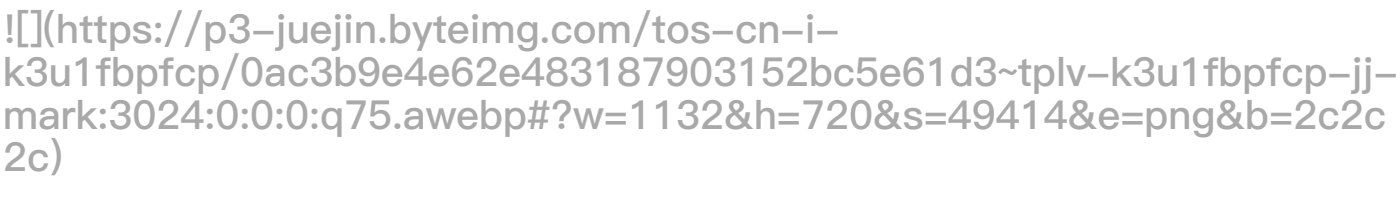
    public static void main(String[] args) {
        MyThread thread1 = new MyThread();
        thread1.start();

        MyRunnable runnable = new MyRunnable();
        Thread thread2 = new Thread(runnable);
        thread2.start();
    }
}
```

```

### 测试结果

针对如上测试代码，这里我本地进行实际测试一波，结果仅供参考，有条件的同学们也可以自己本地实践一下。



### 测试代码解析

针对如上测试代码，这里我再具体给大家讲解下，希望能够更透彻的帮助大家理解。测试用例定义了一个名为`Test`的公共类，其中包含`main`方法，这是Java程序的入口点。在`main`方法中，创建并启动了两个线程：`thread1`和`thread2`。

1. **注释部分**：

* 代码顶部的注释包含了我本地编辑器的信息、来源和日期。这通常用于说明代码的出处和编写时间，也建议大家都设置一下。
2. **main方法**：

* `main`方法是程序执行的起点。它接收一个字符串数组`args`作为参数，这个数组通常用于传递命令行参数给程序（这是学习Java最基础的了，这里我也重提一下）。
3. **创建并启动线程`thread1`**：

* `MyThread`是一个继承自`Thread`的类，这意味着它可能是一个自定义的线程类。在`Test`类的`main`方法中，创建了`MyThread`的一个实例`thread1`，然后调用了它的`start`方法来启动线程。`start`方法会创建一个新的执行线程，并在这个线程中执行`MyThread`类的`run`方法。
4. **创建并启动线程`thread2`**：

* `MyRunnable`是一个实现`Runnable`接口的类。`Runnable`接口是Java中定义线程执行任务的一种方式。在`Test`类的`main`方法中，创建了`MyRunnable`的一个实例`runnable`，然后创建了一个新的`Thread`对象`thread2`，将`runnable`作为目标（target）传递给`Thread`构造函数。最后
```

，调用`thread2`的`start`方法来启动线程。这将导致`Thread`对象在新的执行线程中执行`run`方法，`run`方法将调用`runnable`的`run`方法。

总而言之，这个案例我向大家演示了两种在Java中创建和启动线程的方式：一种是通过继承`Thread`类，另一种是通过实现`Runnable`接口。两种方式都可以定义线程的执行任务，但是实现`Runnable`接口的方式更加灵活，因为它允许你将任务的执行逻辑与线程的创建和控制逻辑分离开来。

但是，这里我也要多提一嘴，需要注意的是，我这个案例中并没有提供`MyThread`和`MyRunnable`类的实现细节，所以我们无法知道这些类的`run`方法具体做了什么。为了完全理解这段代码的行为，我们需要这些类的完整定义，也就是所谓的业务逻辑，希望实战的同学能够把它运用上。

小结

讲解到这里，我对本段进行个收尾工作啦，帮助大家一起梳理下。本文主要通过介绍Java中多线程的创建方法，详细讲解了继承`Thread`类和实现`Runnable`接口两种创建线程方式的使用和特点。同时讲解了每种方式的优缺点，及对比了两者的优缺点，最后通过相应的类代码方法介绍和实战演示该创建线程的代码案例来充分演示这几种创建方式实践，最后对案例代码进行解读及分析，帮助到薄弱的同学更快速的入手掌握。

总结

在此，想必大家应该就能体会到了，多线程是Java开发中常用的技术，能够提高程序的执行效率。通过本文的学习，我们了解了Java中多线程的创建方法，包括继承`Thread`类和实现`Runnable`接口两种方式。在实际开发中，我们可以根据需求选择合适的方式来创建线程，从而提高程序的性能和效率。

结尾

最后，我希望本文能够帮助大家理解并掌握Java中多线程的创建方法，并在实际开发中能够灵活运用它，而不仅仅是为了学而学，一定要把它运用到实际场景中去，发挥线程的功能。多线程它是有一个广阔而有趣的领域，希望大家也能够进一步深入学习和探索，提高自己在多线程编程方面的能力，这对项目开发能力有着极大的提升。

... ...

ok，以上就是我这期的全部内容啦，若想学习更多，你可以持续我，我会把这个多线程篇系统性的更新，保证每篇都是实打实的项目实战经验所撰。只要你每天学习一个奇淫小知识，日积月累下去，你一定能成为别人眼中的大佬的！功不唐捐，久久为功！

「学无止境，探索无界」，期待在技术的道路上与你再次相遇。咱们下期拜拜~~