深入Java多线程：基础篇，你必须掌握！| 多线程篇(一)

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

前言

今天，让我们深入探讨一个经典话题——Java多线程。这个主题对于许多开发者来说并不陌生，因为它在实际项目开发中的应用非常广泛。无论是Java新手还是有经验的开发者，都适合阅读和学习这个主题。对于初学者来说，这是一个深入了解多线程概念的好机会；而对于进阶者，这将是一个深入了解多线程原理、设计和实现的绝佳途径。

以下是我为系统性学习多线程而整理的教程大纲。我计划在接下来的几个月里，分阶段逐步更新这个系列，直至完全完成。我的目的是帮助更多的开发者系统性地掌握多线程知识。

引言：

多线程编程是Java开发中的一项核心技能，它不仅能够提升程序的性能，还能优化资源的使用。无论你是刚步入Java世界的新手，还是已经积累了一定经验的开发者，深入理解多线程都将为你的职业生涯增添宝贵的财富。

教程概览：

本系列教程将从基础概念出发，逐步深入到多线程的高级主题。每一期更新都将围绕一个核心主题，确保你能够系统性地掌握多线程的各个方面。

更新计划：

我将在未来几个月内，持续更新本系列教程。每期内容都将精心编排，确保知识点的连贯性和深度。完成本系列后，我将根据反馈和需求，考虑启动新的系

列。

目标读者：

* Java初学者：希望通过系统性学习，快速掌握多线程的基本概念和应用。
* 进阶Java开发者：希望深化对多线程原理的理解，探索更高效的编程模式。

内容预告：

* 多线程基础：线程的创建、生命周期和同步机制。
* 线程安全与锁：深入理解线程安全的概念，学习如何使用锁来保证线程安全。
* 并发工具类：探索Java并发API中的各种工具类，如Executor框架、Future、Callable等。
* 高级主题：包括线程池、并发集合、原子变量等高级特性的深入讲解。
* ...

希望这期系统教学内容能够更好地吸引读者，并帮助他们系统性地学习Java多线程。

摘要
--

在未来三个月的更新计划中，我将深入探讨Java多线程的核心概念和应用。对于初学者来说，理解多线程的基础知识至关重要。虽然这些知识点可能听起来有些枯燥，但我会尽力用生动有趣的语言来阐述，确保同学们能够愉快地学习，能够在我这片净土中培养学习的热情与初衷。

在Java多线程的世界里，基础概念就好比是构建高楼大厦的基石。虽然这些基础知识点可能显得有些枯燥，但它们是理解复杂多线程应用的关键。以下是本期教程的内容大纲，将从多线程的基本概念开始，逐步引导大家进入多线程编程的大门：

* **线程与进程：**

+ 我们将深入了解线程和进程之间的区别与联系，为理解多线程编程打下坚实的基础。
* **线程的生命周期：**

+ 我们将探索线程从创建到消亡的各个状态，包括新建、就绪、运行、阻塞和死亡，帮助大家全面理解线程的生命周期。
* **线程优先级：**

+ 我们将学习线程优先级的概念，以及如何设置线程优先级来优化程序性能。

虽然多线程的基础知识可能需要一些时间去消化，但请相信，随着学习的深入，你将能够更加灵活地运用这些知识来解决实际问题，随着你有一定的基础，日积月累，这些自然也就通了。

正文
--

在开始编写多线程程序之前，理解线程和进程的基本概念至关重要。我们将从这里出发，先从理解多线程开始再到进程，逐步深入到线程的生命周期管理，以及如何通过设置线程优先级来优化程序性能。

线程与进程

何为线程?

首先，作为科班出身的同学们都知道，线程和进程是计算机中的两个基本概念，它们在程序的执行和资源管理中扮演着重要的角色。那么我就来探探你们的底子，看看厚不厚，是不是学的都还给老师了？何为线程，请3秒内作答！！

何为线程？这个问题其实很简单，但如果想要通过简洁的话来把它讲清楚，那是有点难度，这里就听我来理解一遍，你们再自己梳理一下。

线程，或称为线程，是程序执行的最小单元，是操作系统能够进行运算调度的最小单位。在多线程环境中，一个程序可以同时运行多个线程，每个线程可以执行不同的任务，或者多个线程可以协同完成一个复杂的任务。

以下是线程的几个关键特点：

1. **轻量级**：线程比进程更轻量，创建和切换的开销较小。
2. **独立性**：每个线程拥有自己的执行栈，但通常与其他线程共享同一进程

的资源，如内存空间、文件句柄等。
3. **并发性**：线程可以在多核处理器上并行执行，提高程序的执行效率。
4. **共享资源**：同一进程内的线程可以共享进程的资源，这使得线程间的通信更为便捷，但也需要注意线程安全问题。
5. **同步与通信**：线程之间可能需要同步操作，以避免数据竞争和确保数据一致性。线程间通信（IPC）通常比进程间通信（IPC）更简单。

在Java中，线程可以通过继承`Thread`类或实现`Runnable`接口来创建。Java提供了丰富的API来管理线程的生命周期，包括线程的创建、启动、同步、中断以及优先级设置等。

以下用一个简单的比喻来描述线程：

想象一下，线程就像是一个乐队中的乐手。每个乐手（线程）都有自己的乐器（执行栈），但他们共同演奏同一首曲子（程序）。乐手们可以同时演奏（并发执行），但他们都使用同一套乐谱（共享内存空间）。乐队指挥（操作系统）负责协调乐手们的工作，确保音乐的和谐与流畅。

我这样讲，屏幕前的你是否能够理解透了！

何为进程?

那何为进程呢？这又是一个烧脑壳的问题？大脑中瞬间空白，虽然大概知道，但是要用话来描述，跟讲线程一样，这是有难度。那你还是来听我来讲讲吧。

进程是计算机中的一个基本概念，指的是在操作系统中正在运行的一个程序的实例。每个进程都有其独立的内存空间，至少一个线程（通常称为主线程），以及操作系统分配给它的资源，如CPU时间、文件句柄、网络连接等。

以下是进程的一些关键特点：

1. **独立性**：每个进程都有自己独立的内存空间和系统资源，进程间的信息交换需要通过进程间通信（IPC）机制来实现。
2. **资源分配**：进程是资源分配的基本单位，操作系统为每个进程分配必要的资源以保证其正常运行。
3. **执行环境**：进程提供了程序执行所需的环境，包括代码、数据和堆栈等。
4. **调度**：进程是操作系统进行CPU调度的基本单位，操作系统根据进程的

优先级和调度算法决定哪个进程获得CPU时间。
5. **生命周期**：进程有其生命周期，包括创建、就绪、运行、阻塞和终止等状态。

用一个简单的比喻来描述进程：

想象一下，进程就像是一个独立的剧院，每个剧院（进程）都有自己的舞台（内存空间）、演员（线程）、道具（资源）和剧本（程序代码）。不同的剧院可以同时上演不同的剧目，它们互不干扰，各自拥有自己的观众（CPU时间）和表演时间。

以下用一个简单的比喻来描述**进程**：

* **剧院**：代表进程，是程序运行的独立环境。
* **舞台**：代表进程的内存空间，是演员表演的地方。
* **演员**：代表线程，是剧院中实际执行表演的个体。
* **道具**：代表进程的资源，如灯光、音响等。
* **剧本**：代表程序代码，是演员表演的依据。

怎么样，通过如上形象比喻，你们肯定能更直观地理解进程的概念以及它在计算机系统中的作用了吧，再喊看不懂，我就要打你们小手心了。

区别与联系

既然如上内容作为非科班的同学学完也大概能够理解，那么它两之间究竟有何区别之处？这个问题值得我们审视。这里我用一个简单的比喻来解读下它们之间的区别和联系，非常形象，希望能够帮助你理解：

你无妨再想象一下，进程它就像是一个工厂，而线程则是工厂里的工人。每个工厂（进程）都有它自己的资源，比如机器、原材料等。但是，工厂的运作需要工人（线程）来完成具体的工作。

* **进程**：每个进程就像是独立的工厂，拥有自己的资源和空间。进程是操作系统进行资源分配和调度的一个基本单位。它确保了各个工厂之间的独立性，即使一个工厂出现问题，也不会影响到其他工厂的运作。
* **线程**：线程则像是工厂里的工人，它们是程序执行的最小单元。每个工人（线程）可以同时做不同的工作（执行不同的任务），这样可以提高工厂的效率。而且，这些工人可以共享工厂的资源，比如使用同一台机器或者同一批原材料。

在Java中，这种概念被具体实现为：

* **Java线程**：Java程序中的每个线程都是一个独立的执行流，它们可以并行地执行不同的任务。
* **资源共享**：Java线程之间共享同一进程的内存空间和资源，这使得线程间的通信和数据共享变得更加容易。

用通俗易懂的话来理解就是：

* **线程**：就像是家里的兄弟姐妹，大家共享同一个家（进程）的资源，比如电视、玩具等，但是每个人都可以同时做自己的事情，比如一个在看电视，一个在玩玩具。
* **进程**：就像是一个家，每个家都有自己的房子、家具等资源。不同的家（进程）之间是独立的，不会互相干扰。

通过这种方式，Java能够有效地利用多线程来提高程序的执行效率和响应速度，同时保持资源的高效管理和利用。

如下我再简单给同学们演示下，如何通过代码来创建一个多线程，示例代码如下：

```
// Java线程示例
public class MyThread extends Thread {
    public void run() {
        // 线程执行的代码
    }
}
```

生命周期

概念

这里要提到另一个概念，生命周期，那该生命周期究竟为何？顾名思义，线程的生命周期通常指的是一个对象或实体从创建到消亡的整个过程。在计算机科

学中，这个概念被广泛应用于不同的上下文中，比如软件工程、操作系统和多线程编程。对于不同的实体，生命周期的具体阶段可能有所不同，但基本的框架是相似的。线程的生命周期它包括新建、就绪、运行、阻塞和死亡五个状态。理解这些状态有助于我们更好地管理线程。

进程的生命周期

对于进程而言，生命周期通常包括以下几个阶段：

1. **创建（New）**：进程被操作系统创建，分配必要的资源，如内存、文件句柄等。
2. **就绪（Ready）**：进程已经准备好执行，等待操作系统调度分配CPU时间。
3. **运行（Running）**：进程获得CPU时间，开始执行指令。
4. **阻塞（Blocked）**：进程因为等待某些事件（如I/O操作完成）而暂停执行。
5. **终止（Terminated）**：进程执行完毕或被操作系统强制终止，释放所有资源。

线程的生命周期

线程作为进程中的执行单元，其生命周期与进程类似，但更细致：

1. **新建（New）**：线程对象被创建，但尚未启动。
2. **就绪（Runnable）**：线程已经准备好运行，等待CPU时间。
3. **运行（Running）**：线程正在执行其任务。
4. **阻塞（Blocked）**：线程因等待资源（如锁）或其他线程的操作而暂停。
5. **等待（Waiting）**：线程因等待某个条件的发生而暂停执行，需要其他线程的协作才能继续。
6. **超时等待（Timed Waiting）**：线程在等待某个条件的同时，设定了一个最长等待时间。
7. **死亡（Terminated）**：线程完成任务或被显式终止。

软件工程中的生命周期

在软件工程中，生命周期通常指的是软件从概念化到退役的整个过程，包括：

1. **需求分析**：确定软件需要满足的功能和性能要求。

2. **设计**：根据需求制定软件的架构和详细设计。
3. **实现**：编写代码，实现软件设计。
4. **测试**：验证软件的功能和性能是否符合要求。
5. **部署**：将软件发布到生产环境供用户使用。
6. **维护**：对软件进行持续的改进和修复。
7. **退役**：软件停止使用并从生产环境中移除。

生命周期的概念帮助我们理解一个实体从开始到结束的完整过程，以及在这个过程中可能经历的不同状态。在不同的领域，生命周期的具体内容和阶段可能有所不同，但其核心思想是一致的。

线程优先级

概念

继续，保证学完这个概念，我们就下课。跟着我思考，何为线程优先级？这也得掌握，不得马虎。好理解的，见名知意。线程优先级它就是操作系统在进行线程调度时考虑的一个因素，它决定了线程获得CPU时间的相对顺序。线程优先级影响线程的调度策略，但并不保证高优先级的线程一定会先于低优先级的线程执行，因为线程的执行还受到其他调度因素（如线程状态、系统调度策略等）的影响。

特点

1. **相对性**：线程优先级是相对的，它只在同一进程内的线程之间比较有效。
2. **范围**：在Java中，线程优先级的范围通常是1到10，其中1是最低优先级，10是最高优先级。
3. **默认优先级**：大多数线程在创建时，如果没有显式设置优先级，会继承其父线程（通常是main线程）的优先级。
4. **调度影响**：高优先级的线程可能会更频繁地获得CPU时间，但这并不是绝对的，因为实际的调度还受到操作系统的调度策略和当前系统负载的影响。

如何设置线程优先级?

这里我通过演示个简单的伪代码示例，以辅助大家理解下如何在代码层面上设置线程优先级，实践出真知，索性我就通过代码的形式来演示一波，示例代码如下：

```
Thread thread = new Thread(new Runnable() {
    public void run() {
        // 线程要执行的任务
    }
});

// 设置线程优先级
thread.setPriority(Thread.MAX_PRIORITY); // 设置为最高优先级
// 或者
thread.setPriority(Thread.MIN_PRIORITY); // 设置为最低优先级
// 或者设置为默认优先级
thread.setPriority(Thread.NORM_PRIORITY); // 默认优先级
```

setPriority()方法

在Java中，`setPriority` 方法用于设置线程的优先级。线程优先级是一个整数，它影响线程的调度顺序。Java 线程优先级的范围从 1 到 10，其中 1 是最低优先级，10 是最高优先级。默认情况下，线程的优先级是 5，称为 `Thread.NORM_PRIORITY`。

以下我给大家总结些 `setPriority()` 方法的一些关键点，以便于大家在日后用到时能够如鱼得水。

1. **相对性**：线程优先级是相对的，只在同一进程内的线程之间比较有效。
2. **默认优先级**：新线程如果没有显式设置优先级，将继承创建它的线程（通常是主线程）的优先级。
3. **调度影响**：高优先级的线程可能会更频繁地获得 CPU 时间，但这并不是绝对的，因为实际的调度还受到操作系统的调度策略和当前系统负载的影响。

使用 `setPriority` 设置线程优先级的示例代码：

```
package com.example.javase.bugTest.thread.day1;

/**
 * @Author bug菌
 * @Source 公众号：猿圈奇妙屋
```

```
 * @Date 2024-06-25 21:11
 */
public class PriorityExample {
    public static void main(String[] args) {
        Thread currentThread = Thread.currentThread();
        System.out.println("当前线程优先级: " + currentThread.getPriority());

        // 创建并启动一个新线程
        Thread highPriorityThread = new Thread(new Runnable() {
            public void run() {
                // 执行一些任务
            }
        });

        // 设置新线程的优先级为最高
        highPriorityThread.setPriority(Thread.MAX_PRIORITY);

        // 启动线程
        highPriorityThread.start();
    }
}
```

代码解析：

还是老配方，这里我简单对上述代码做个解释：

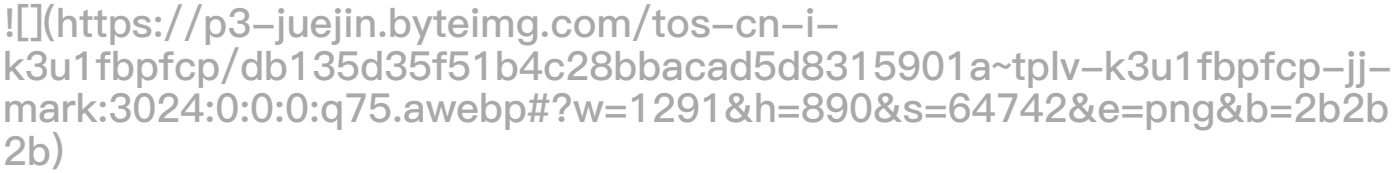
如上这段代码演示了如何使用 `setPriority()` 方法设置线程的优先级。下面是代码的逐行解释：

1. `public class PriorityExample`：定义了一个名为 `PriorityExample` 的公共类。
2. `public static void main(String[] args)`：`main` 方法是程序的入口点，`args` 是传递给 `main` 方法的参数数组。
3. `Thread currentThread = Thread.currentThread()`：通过调用 `Thread.currentThread()` 静态方法获取当前执行的线程（即主线程）的引用。
4. `System.out.println("当前线程优先级: " + currentThread.getPriority())`：打印当前线程（主线程）的优先级。
5. `Thread highPriorityThread = new Thread(new Runnable() {...});`：创建一个新的 `Thread` 对象，该线程将执行一个实现了 `Runnable` 接口的匿名类。这个匿名类定义了线程要执行的任务。
6. `highPriorityThread.setPriority(Thread.MAX_PRIORITY);`：设置新创建的线程 `highPriorityThread` 的优先级为最高，即

`Thread.MAX_PRIORITY`，这个常量值为10。
7. `highPriorityThread.start();`：调用 `start()` 方法启动新线程。一旦线程启动，它将调用 `Runnable` 接口的 `run` 方法来执行其任务。

这个案例展示了如何创建一个线程，设置其优先级，并启动它。然而，同学们需要注意的是，线程优先级的调度是由操作系统控制的，并且实际效果可能因操作系统和当前系统负载而异。此外，过度依赖线程优先级可能导致优先级反转和优先级继承问题，因此在实际应用中应谨慎使用，我们知道可以设置优先级，但是别滥用就行。

通过如上案例代码，这里我将本地案例结果截图展示给大家看下：



注意事项

* **优先级反转**：如果一个高优先级的线程等待一个低优先级的线程同步完成，而低优先级的线程又在等待其他线程，这可能导致优先级反转问题。
* **优先级继承**：一些操作系统允许低优先级的线程执行高优先级任务，这可能导致优先级继承问题。
* **操作系统支持**：线程优先级的实际效果依赖于操作系统的支持，不同的操作系统可能会有不同的表现。

使用 `setPriority` 时，应该谨慎考虑线程优先级的设置，以避免潜在的问题，并确保程序的稳定性和效率。

线程优先级的应用场景

* **实时系统**：在需要快速响应的实时系统中，线程优先级用于确保关键任务能够及时执行。
* **用户界面**：在GUI应用程序中，提高事件处理线程的优先级可以改善用户界面的响应性。
* **后台任务**：对于不需要立即完成的后台任务，可以设置较低的优先级，以避免它们抢占前台任务的CPU时间。

注意事项

* 线程优先级不应该作为确保线程执行顺序的唯一手段，因为多线程环境中存在许多不可预测的因素。
* 过度依赖线程优先级可能导致优先级反转和优先级继承问题，需要谨慎使用。
* 在多核处理器上，线程优先级的影响可能会降低，因为每个核心可以独立调度线程。

线程优先级是一个有用的工具，但开发者需要根据具体的应用场景和系统环境来合理设置和使用。

案例演示

我们将通过具体的Java代码示例，深入分析线程的创建、同步、通信以及如何避免常见的并发问题。

这里我先通过例子，然后再代码解析一波，以帮助大家更好地理解代码的功能、逻辑和结构。在多线程编程中，源码解析尤为重要，因为它可以帮助同学们理解线程的创建、同步、通信和调度等方面，所以有基础的同学可以略过，但是零基础的同学务必认真。

下面是一个简单的Java多线程程序的示例，包括源代码和对应的解析，如下：

```
package com.example.javase.bugTest.thread.day1;

/**
 * @Author bug菌
 * @Source 公众号：猿圈奇妙屋
 * @Date 2024-06-25 21:13
 */
public class ThreadExample extends Thread{

    // 主方法，程序的入口点
    public static void main(String[] args) {

        // 创建ThreadExample类的实例，即创建一个线程对象
        Thread thread1 = new Thread();

        // 启动线程，调用run方法
```

```
        thread1.start();

        // 在主线程中执行一些操作
        for (int i = 0; i < 5; i++) {
            System.out.println("主线程执行：" + (i + 1));
        }
    }

    // 覆写了Thread类的run方法，定义线程要执行的任务
    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println("线程执行：" + (i + 1));
        }
    }
}
```

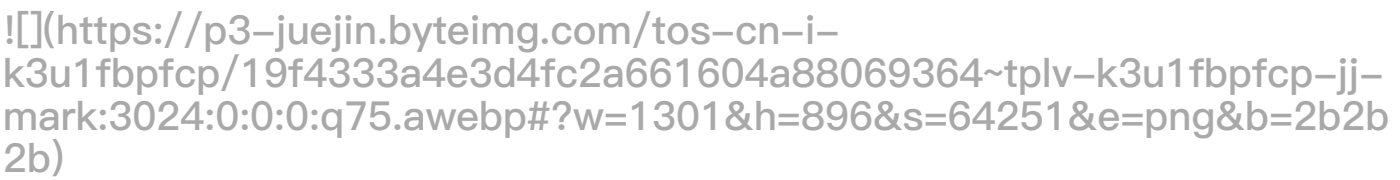
案例代码解析

1. **导入Thread类**：首先，程序通过`import java.lang.Thread;`导入了Java的标准库中的`Thread`类，这是进行多线程编程的基础。
2. **定义ThreadExample类**：这个类是`Thread`类的子类，它将创建一个新的线程。
3. **主方法（main）**：这是Java程序的入口点。在这个方法中，我们创建了`ThreadExample`的一个实例，这个实例同时也是一个线程对象。
4. **创建线程对象**：通过`new ThreadExample()`创建了`ThreadExample`的一个实例，这个实例包含了要执行的代码。
5. **启动线程**：通过调用`thread1.start()`方法启动了线程。这会导致自动调用`run`方法，线程开始执行。
6. **主线程中的循环**：在`main`方法中，有一个循环，打印了5次"主线程执行"的信息。
7. **覆写run方法**：`ThreadExample`类覆写了`Thread`类的`run`方法，定义了线程要执行的具体任务。在这个例子中，线程的任务是打印5次"线程执行"的信息。
8. **线程执行**：当`start()`被调用后，线程会并行地执行它的`run`方法，与主线程同时运行。

如上这个简单的示例展示了如何在Java中创建和启动线程，以及如何在主线程和新线程中执行不同的任务。通过源代码解析，我们可以清晰地看到线程的创建、启动和执行过程，以及它们是如何与主线程并发运行的。

案例结果演示：

通过如上案例代码，这里我将本地案例结果截图展示给大家看下：



优缺点分析

优点

* 提高程序的响应性：多线程可以使得程序在等待某些操作完成时继续执行其他任务。
* 提高资源利用率：多线程可以更有效地利用多核处理器。

缺点

* 增加编程复杂性：多线程编程需要考虑线程安全、死锁等问题。
* 资源竞争：线程之间可能会竞争共享资源。

小结
--

在本教程的最后，我们做一个简单的回顾。我们从多线程的基础知识讲起，逐步深入到线程的生命周期、优先级设置，以及如何通过Java代码实现多线程。我们通过具体的代码示例，展示了线程的创建和启动过程，以及如何在多线程环境中进行任务的同步和通信。

我们强调了多线程编程的优点，比如提高程序性能和资源利用率，同时也指出了它的缺点，如增加了编程的复杂性和潜在的资源竞争问题。通过测试用例，我们学习了如何验证多线程程序的正确性，并确保线程行为符合预期。

总结
--

随着本教程的深入，我们一同探索了Java多线程。从线程与进程的基本概念，到线程的生命周期和优先级，我们不仅学习了理论知识，还通过代码示例和实际应用案例，加深了对多线程编程的认识。多线程编程是一种强大的工具，它能够帮助我们构建更加高效和响应迅速的应用程序。然而，它也带来了新的挑战，比如线程安全和死锁问题，这要求我们在设计和实现多线程程序时必须更加谨慎(这点我会在后几期给大家仔细讲解的，这里就提一嘴)。

通过本教程，我们希望能够帮助大家：

1. **理解多线程的基本概念**：包括线程与进程的区别、线程的生命周期和优先级等。
2. **掌握多线程编程的基本技巧**：如何创建线程、如何管理线程的执行和调度。
3. **学会使用Java提供的多线程工具和API**：比如`Thread`类、同步机制、并发工具类等。
4. **认识到多线程编程的复杂性**：了解线程安全、死锁等概念，并学会如何避免这些问题。

多线程是一个不断学习和实践的过程。随着技术的不断进步，我们也需要不断更新自己的知识和技能，以适应新的编程挑战，在日常的开发中，这些基础是我们务必需要掌握的，否则真会被社会所淘汰，假设那天面试，可别再说这方面我没用上，那就真太拉了。

最后，我希望通过本教程，能够激发大家对多线程编程的兴趣，帮助同学们在实际开发中更好地应用多线程技术。记住，多线程编程是一项宝贵的技能，但也需要细心和耐心来掌握。不断实践，不断学习，你将能够精通这门艺术。

在此，感谢您的陪伴，希望本教程对您有所帮助。如果您有任何问题或想要深入探讨的话题，请随时与我联系，或者评论区留言都行，三人行，必有我师焉，咱们一起学习。

... ...

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~~