河南中医药大学智能医学工程专业《计算机程序设计》课程

第11章：多线程

黄子杰

河南中医药大学信息技术学院（智能医疗行业学院）与河南方和信息科技有限公司 联合建设
河南中医药大学信息技术学院互联网技术教学团队
https://webdev.hactcm.edu.cn
2024/9/13

本章概要

多线程是一种让程序能够同时执行多个任务的技术。它允许一个程序包含多个执行流，这些执行流可以并行地执行不同的代码段。多线程对于提高程序性能、优化资源利用以及开发复杂的并发应用程序至关重要。

11.1 Java 线程的概念

Java多线程是一种使单个程序中的多个任务同时执行的机制。通过创建多个线程，可以充分利用系统资源，提高程序的执行效率。在Java中，线程可以通过继承Thread类或实现Runnable接口来创建。

11.1.1 多进程与多线程

- 概述

多进程和多线程是计算机并发执行任务的两种方式。进程是操作系统资源分配的基本单位，拥有独立的内存空间；线程是进程的执行单元，是CPU调度和执行的单位，多线程共享同一进程的资源。

11.1.1 多进程与多线程

- 详细说明：

多进程：每个进程都有自己独立的内存空间，相互隔离。进程之间的通信开销大，但更加稳定。

多线程：同一进程中的多个线程共享相同的内存空间，因此线程之间的通信更加高效，但需要注意同步问题以避免数据冲突。

11.1.1 多进程与多线程

多进程与多线程区别

- 内存空间差异

多进程与多线程在内存管理上存在本质区别，多进程各自拥有独立的内存空间，而多线程则共享同一进程的内存资源，这影响了程序的隔离性和资源共享。

- 通信开销对比

在多进程和多线程之间，通信机制的不同导致了显著的开销差异，多进程通信需要操作系统介入，增加了复杂性，而多线程通过共享内存实现更高效的通信。

- 稳定性考量

多进程由于内存空间独立，提供了更高的稳定性，单个进程的崩溃不会影响到其他进程；相比之下，多线程中一个线程的错误可能导致整个进程崩溃，影响稳定性。

11.1.1 多进程与多线程

- 生活举例：

多进程：就像每个员工有自己的办公桌，互相独立工作。
多线程：就像一个员工在多个任务之间切换，任务之间可能会有共享的资源，如文件或数据表。

代码实例：


eg11.1.1.txt


eg11.1.1-2.txt

11.1.2 线程的状态

- 概述：

Java线程在生命周期中可以处于五种状态：新建、就绪、运行、阻塞、终止。

11.1.2 线程的状态

- 详细说明：

- 新建（New）：线程对象创建后，未启动前。
- 就绪（Runnable）：线程已经启动，等待CPU资源。
- 运行（Running）：线程正在执行。
- 阻塞（Blocked）：线程被阻塞，等待某种条件（如资源或锁）满足。
- 终止（Terminated）：线程执行完毕或被中断。

11.1.2 线程的状态

- 生活举例：

线程状态：像一个员工的工作状态，从准备工作、开始工作、等待资源、到工作完成。

- 代码示例：

eg.11.1.2.txt

11.1.3 线程调度与优先级

- 概述：

线程调度是操作系统决定何时以及哪一个线程应当运行的过程，确保系统资源的有效利用和任务的合理分配。

线程优先级是一种机制，用于向操作系统的调度器建议哪些线程应优先获得CPU时间，通过设置不同的优先级，可以影响线程的执行顺序，但并不保证执行顺序，还需考虑调度策略如时间片轮转或优先级调度。

11.1.3 线程调度与优先级

- 详细说明：

优先级：Java线程的优先级范围是1到10，Thread.MIN_PRIORITY是1，Thread.NORM_PRIORITY是5，Thread.MAX_PRIORITY是10。高优先级的线程更有可能获得CPU时间。

调度方式：操作系统通常使用时间片轮转（Round-Robin）或优先级调度策略。

11.1.3 线程调度与优先级

- 生活举例：

线程调度：像餐厅中的服务员，优先处理VIP顾客，但也会轮流照顾其他顾客。

- 代码示例：

eg.11.1.3.txt

11.2 Java 多线程编程方法

Java多线程编程方法概念涉及在Java语言中实现并发执行的技术。这包括创建和管理线程，使用线程池来优化资源利用，以及同步机制来控制对共享资源的访问。Java提供了多种方式来实现多线程，如继承Thread类或实现Runnable接口。此外，Java的并发包（java.util.concurrent）提供了更高级的并发工具和框架，使得多线程编程更加安全和高效。

11.2.1 Thread类简介

- 概述：

Thread类是Java中的主要多线程类，它提供了创建和控制线程的方法，包括启动、运行和终止线程的机制，是实现并发执行的关键。通过创建Thread类的实例并调用其start()方法，可以启动一个新的线程。

11.2.1 Thread类简介

- 详细说明：

创建和启动线程：

通过实例化Thread类并调用其start()方法，可以创建并启动一个新的线程，这是在Java中实现多线程编程的第一步，为程序的并行执行奠定了基础。

常用方法：

start()：启动线程。

run()：线程执行的代码。

sleep(long millis)：让当前线程休眠指定时间。

join()：等待线程完成。

interrupt()：中断线程。。

11.2.1 Thread类简介

s

- 生活举例：

Thread类：就像雇佣一个工人来完成特定的任务，你可以指定任务并控制任务的执行。

- 代码示例：

eg.11.2.1.txt

11.2.2 继承Thread类实现多线程

- 概述

通过继承Thread类，我们可以定义自己的线程类，并在其中重写run方法来指定线程执行的任务。创建线程实例时，只需实例化自定义的线程类即可。这是Java中实现多线程的常见方法之一。

11.2.2 继承Thread类实现多线程

- 详细说明：

继承方式：子类继承Thread类，并在run()方法中编写线程的任务逻辑。

重写run()方法：

在继承Thread类的子类中，必须重写run()方法，并在该方法内部编写线程需要执行的任务代码，这是实现多线程任务的关键步骤。

注意事项：创建了继承自Thread类的实例后，应调用其start()方法来启动线程，而不是直接调用run()方法，因为直接调用run()方法不会创建新的线程

11.2.2 继承Thread类实现多线程

- 生活举例：

继承Thread：像招聘员工并直接告诉他们要做什么。

- 代码示例：

eg.11.2.2.txt

11.2.3 实现Runnable接口编写多线程

- 概述

实现Runnable接口是另一种创建线程的方法，通过定义一个类实现Runnable接口，并重写其run方法，可以创建一个线程任务。这种方式使得线程任务与线程本身解耦，提高了代码的灵活性和可扩展性。

11.2.3 实现Runnable接口编写多线程

- 详细说明：

实现方式：实现Runnable接口的run()方法，编写线程逻辑。

优点：相比继承Thread类，实现Runnable接口可以避免Java单继承的局限，并允许对象复用。

11.2.3 实现Runnable接口编写多线程

- 生活举例：

实现Runnable：像雇佣一名承包商，提供具体的任务说明。。

- 代码示例：

eg.11.2.3.txt

11.3 线程资源的同步处理

线程资源的同步处理是指在多线程环境下，对共享资源进行访问控制的一种机制。这种机制可以确保在同一时间只有一个线程能够访问共享资源，从而避免数据不一致的问题。同步处理可以通过互斥锁、信号量等技术实现。

11.3.1 临界资源问题

- 概述：

在多线程环境中，临界资源是那些可能同时被多个线程访问和修改的共享资源。若未妥善处理，这些资源可能会引发数据不一致或竞态条件的问题。为了避免这些问题，需要使用同步机制来确保同一时间只有一个线程可以访问临界资源

11.3.1 临界资源问题

- 详细说明：

竞态条件：当多个线程试图同时访问和修改临界资源时，可能会引发数据不一致或竞态条件，这是因为线程间的操作顺序和时间点无法预测，导致结果不可预期。

解决方法：为避免临界资源导致的并发问题，可以采用同步机制确保每次只有一个线程能够访问临界资源。例如，在Java中使用`synchronized`关键字来同步方法或代码块

11.3.1 临界资源问题

- 生活举例：

临界资源：像多个人同时在同一个共享文档上编辑，可能会互相覆盖对方的修改。

- 代码示例：

eg.11.3.1.txt

11.3.2 wait()和notify()方法

- 概述：

wait()和notify()是用于线程间通信的两种方法。wait()使当前线程等待，直到另一个线程调用notify()或notifyAll()方法来唤醒它。

11.3.2 wait()和notify()方法

- 详细说明：

wait()方法：wait()方法用于让当前线程进入等待状态，直到其他线程调用notify()或notifyAll()方法。此方法常用于线程间同步，确保资源的有效分配和访问。

notify()方法：notify()方法允许一个线程唤醒正在同一对象上等待的其他线程中的一个。被唤醒的线程不会立即获取到锁，直到原调用notify()的线程释放了锁。

notifyAll()方法：notifyAll()方法与notify()类似，但不同之处在于它会唤醒所有在该对象上等待的线程

11.3.2 wait()和notify()方法

- 生活举例：

wait()和notify()：像一个人在某个房间等候（wait()），直到另一个人进入房间并通知（notify()）他继续工作。

- 代码示例：

eg.11.3.2.txt

11.4 Java 的任务定时处理

- 概述：

Java提供了多种方式来进行任务的定时处理，例如Timer类和ScheduledExecutorService接口，可以用来在未来某个时间或以固定间隔重复执行任务。

11.4 Java 的任务定时处理

- 详细说明：

Timer类：允许你安排一个任务在未来的某个时间执行，或者周期性地执行。

ScheduledExecutorService接口：提供更高级的任务调度功能，支持任务的延迟执行和周期性调度。

11.4 Java 的任务定时处理

- 生活举例：

任务定时处理：像设置闹钟，在指定的时间或周期性地提醒你。

- 代码示例：

eg.11.4.txt

本章重点

多进程与多线程
线程的状态
线程调度与优先级
Java 多线程编程方法
Thread类实现多线程
Runnable接口编写多线程

本章作业

作业一：医院挂号系统的多线程实现

目的：通过实现一个医院挂号系统，练习Java多线程编程方法，包括继承Thread类或实现Runnable接口，以及线程的基本操作。

要求：

1 设计一个挂号系统的基本框架，包括挂号窗口（可以视为线程）和患者队列。

2 使用继承Thread类或实现Runnable接口的方式，创建多个挂号窗口线程。

3 每个挂号窗口线程能够处理患者队列中的患者，模拟挂号过程。

4 考虑线程安全问题，确保患者数据在多线程环境下的正确性。

5 编写一个主类，启动挂号系统，并观察多线程工作的效果。

本章作业

作业二：医院药房的线程同步与定时任务处理

目的：通过医院药房的场景，练习线程资源的同步处理、临界资源问题的解决，以及Java的任务定时处理。

要求：

1 设计一个医院药房系统，包括药房库存和多个取药窗口。

2 使用synchronized关键字或ReentrantLock等同步机制，确保药房库存数据在多个取药窗口线程之间的同步。

3 模拟药房库存不足的情况，当库存不足时，取药窗口应等待库存补充。

4 使用wait()和notify()方法或Condition接口，实现库存不足时的等待与通知机制。

5 引入Java的定时任务处理，如使用Timer类或ScheduledExecutorService，模拟定时检查库存并补充药品的过程。

6 编写一个主类，启动药房系统，并观察线程同步和定时任务处理的效果。

计算机程序设计课程学习平台

面向河南中医药大学智能医学工程专业使用

河南中医药大学信息技术学院（智能医疗行业学院）与河南方和信息科技有限公司 联合建设

河南中医药大学信息技术学院互联网技术教学团队