# Java多线程与并发编程深入浅出

## 第一章 多线程、并发基础知识

### 课程简介

### 基本概念

**1、进程（process）**

**狭义定义：**进程就是一段程序的执行过程。

**简单的来讲进程的概念主要有两点**：**第一**，进程是一个实体。每一个进程都有它自己的系统资源。**第二**，进程是一个“执行中的程序”。程序是一个没有生命的实体，只有处理器赋予程序生命时，它才能成为一个活动的实体，我们称其为进程。

**2、线程**

通常在一个进程中可以包含若干个线程，当然一个进程中至少有一个线程，不然没有存在的意义。线程可以利用进程所拥有的资源，在引入线程的操作系统中，通常都是把进程作为分配资源的基本单位，而把线程作为独立运行和独立调度的基本单位，由于线程比进程更小，基本上不拥有系统资源，故对它的调度所付出的开销就会小得多，能更高效的提高系统多个程序间并发执行的程度。

**3、进程与线程的区别：**

1) 简而言之,一个程序至少有一个进程,一个进程至少有一个线程.

2) 线程的划分尺度小于进程，使得多线程程序的并发性高。

3) 另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。

4) 线程在执行过程中与进程还是有区别的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。

5) 从逻辑角度来看，多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。这就是进程和线程的重要区别。

### java内存模型（JMM）

## 第二章 java多线程

### 2-1 继承Thread类

1. 声明一个父类是Thread类的类Demo1类。
2. 重写run方法。
3. 创建Demo1类的对象即Tread对象，对象执行run方法即创建一个新线程执行run方法里的程序逻辑。
4. 多个线程同时运行run方法里的程序时，只能保证每个线程是顺序执行的，多个线程的执行顺序是无法保证的。

### 2-2 实现Runnable接口

1. 声明一个实现了Runnable接口的类Demo2类。
2. 实现run方法。
3. 创建入参为Demo2类对象的Thread对象，对象执行run方法即创建一个新线程执行run方法里的程序逻辑。
4. 无论是继承Thread类，还是实现Runnable接口，创建线程的本质：创建一个Thread对象，并执行start方法。
5. 只是执行run方法不会创建新线程。

### 2-3 实现Callable接口

1. Callable位于java.util.concurrent包下，它也是一个接口，只声明了一个call()方法。
2. FutureTask实现了RunnableFuture接口，次接口集成了Callable、Future接口。
3. Future接口能获取线程执行的结果，能中断线程，能判断线程是否执行完成。
4. 线程执行FutureTask的run方法实际执行的是call方法，并将结果赋值给outcome
5. 主线程在调用get方法获取结果时，如果线程未执行完毕，主线程将以死循环的形式，等待结果产生。call方法执行完成后，FutureTask会将执行结果赋值给outcome，get方法将其返回。

### 2-4 多线程总结

1. 本质上有集成Thread类、实现Runnable接口两种方式编写线程逻辑。
2. Callable接口是为了获取线程任务执行结果的封装，需要同FutureTask配合使用。
3. 无论以何种方式创建线程，一定是创建一个Thread对象，并且调用这个Thread对象的start方法。
4. 多个线程是并发执行的，执行任务(run方法)的快慢是无序的。

## 第三章 线程安全

### 3-1 Synchronized线程同步机制

问题引入：多个线程去操作一个公共对象，会出现什么现象？

解决办法：把一个对象当作锁对象，用这个锁对操作公共对象的代码加锁，让多个线程去争抢这个锁对象，抢到这个锁对象的线程才能够执行加锁的代码，其他线程必须等待这个线程执行完毕释放锁对象后，才能抢到锁对象，才能执行加锁的代码。

1. 线程安全：多个线程去操作一个公共对象（或其他公共资源）时，这个对象始终都能表现出正常的行为，那么这个对象的类就是线程安全的类。
2. synchronized：可以在任意方法、代码块上加锁，而加锁的这段代码称为"互斥区"或"临界区"。这段代码只能有一个线程处于执行状态，其他线程必须等待当前线程执行完毕，才能获取锁后，执行代码。
3. 方法锁（synchronized修饰方法时） 对象锁（synchronized修饰方法或代码块），琐是当前对象。有两种形式。synchronized method或synchronized block、synchronized(锁对象) { }。
4. 类锁(synchronized 修饰静态的方法或代码块)，锁对象为类的Class对象。

### 线程的状态

1. **新建(NEW)**：新创建了一个线程对象。
2. **可运行(RUNNABLE)**：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。该状态的线程位于可运行线程池（也叫锁池）中，等待被线程调度选中，获取cpu 的使用权。
3. **运行(RUNNING)**：可运行状态(runnable)的线程获得了cpu 时间片（timeslice） ，执行程序代码。
4. **阻塞(BLOCKED)**：阻塞状态是指线程因为某种原因放弃了cpu 使用权，也即让出了cpu timeslice，暂时停止运行。直到线程进入可运行(runnable)状态，才有机会再次获得cpu timeslice 转到运行(running)状态。
5. **死亡(DEAD)**：线程run()、main() 方法执行结束，或者因异常退出了run()方法，则该线程结束生命周期。死亡的线程不可再次复生。
6. <https://www.cnblogs.com/GooPolaris/p/8079490.html>

### 3-3 Synchronized的实现原理

### 3-4 volatile线程可见性

### 3-5 volatile实现原理

## 第四章 JDK并发工具包-JUC

### 4-1 Atomic与CAS算法

### 4-2 CAS算法的ABA问题

### 4-3 ReentrantLock重入锁

### 4-4 Condition线程等待与唤醒

### 4-5 CountDownLatch倒计时锁

### 4-6 Semaphore信号量

### 4-7 CyclicBarrier信号量

### 4-8 Synchronized和Lock的区别

## 第五章 阻塞队列与ThreadPool线程池

### 5-1 阻塞队列基础知识

### 5-2 阻塞队列接口和实现类

### 5-3 阻塞队列代码与原理分析

### 5-4 线程池使用及优势

### 5-5 线程池3个使用方式

### 5-6 线程池7大参数及原理

### 5-7 线程池4中拒绝策略

### 5-8 课程总结