多线程

多线程是提升程序性能非常重要的一种方式，使用多线程可以让程序充分利用CPU资源，提高CPU的使用效率，从而到达解决高并发所带来的负载均衡的问题。

多线程的优点：

1.CPU资源得到更合理的利用。

2.程序设计更加简洁。

3.程序响应更快，运行效率更高。

多线程的缺点：

1.需要更多的内存空间来支持多线程。

2.多线程并发访问的情况可能会影响数据的准确性。

3.一个资源被多个线程共享时，可能会出现死锁的情况。

线程与进程

什么是进程?

进程：计算机在某个数据集合上进行的一次运行活动，是系统进行资源分配和调度的基本单位。

简单理解，进程就是计算机正在运行的一个具体的应用程序，一个应用程序至少有一个进程，也可以是多个进程。

什么是线程?

线程是程序执行的最小单元，线程是进程中的一个基本单位，为独立完成程序中的某一个功能而存在的，一个进程是由一个或多个线程组成的。

进程和线程是应用程序在执行过程中所产生的概念，即如果一个应用程序没有运行，就没有线程和进程的概念，应用程序是一个静态概念，进程和线程是动态概念，只有当静态的应用程序运行起来，才会产生对应的动态的进程和线程，有创建有销毁，存在也是暂时的，不可能永远存在。

进程和线程的区别：

进程在运行时拥有独立的内存空间，即每个进程所占用的内存都是独立的，互不干扰。而多线程是共享内存空间的，线程不能独立执行，必须依赖与进程，由进程提供多线程的执行控制。

我们通常所说的多线程是指在一个进程中，多个线程同时执行，注意：这里的同时执行并不是真正意义上的同时执行，系统会自动为每一个线程分配CPU资源，在某一个具体的时间段内CPU被一个线程所占用，不同的时间段内不同的线程占用CPU资源，所以多线程实际上是多个线程在交替执行，CPU运行速度很快，感觉上是同时在执行。

Java使用线程

- 继承Thread类

1.自定义一个类，继承Thread类。

2.重写Thread类中的run方法，将相应的业务逻辑在run方法中实现。

定义好了一个线程类之后，我们就可以通过该类来实例化对象，对象就可以描述一个线程。

实例化该对象之后，必须通过调用start()来开启该线程，这样该线程才会和其他线程来抢占CPU资源，不能调用run()方法，调用run()相当于普通的实例对象方法调用，并不是多线程。

- 实现Runnable接口

  1.自定义一个类，实现Runnable接口。

  2.实现run方法。

  MyRunnable的使用与MyThread略有不同，MyRunnable相当于定义了线程的业务逻辑，但是它本身不是一个线程，所以需要实例化Thread类的对象作为线程，然后将MyRunnable对象赋给Thread对象，这样Thread就拥有了MyRunnable中定义的业务逻辑，再通过调用Thread对象的start方法来启动线程。

线程的状态

线程一共有5种状态，在特定的情况下，线程可以在不同的状态之间进行切换，5种状态如下：

1.创建状态：实例化了一个新的线程对象，还未启动。

2.就绪状态：线程对象创建好之后，调用了该对象的start方法启动该线程，该状态下的线程位于可运行线程池中，等待系统为其分配CPU资源。

3.运行状态：就绪状态的线程在某个时间段内获取到了CPU资源，执行相应的业务逻辑（run方法的逻辑）。

4.阻塞状态：运行状态的线程因某些原因暂时放弃CPU资源，停止执行程序，解除阻塞之后不能直接回到运行状态，而是重新回到就绪状态，等待下一次的CPU资源。

5.死亡状态：线程执行完毕或因为异常退出，该线程的生命周期结束了。