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，从而解决高并发带来的负载均衡问题。

优点：

- 资源得到更合理的利用
- 程序设计更加简洁
- 程序响应速度更快，运行效率更高

缺点：

- 需要更多的内存空间来支持多线程
- 多线程并发访问可能会影响数据的准确性
- 数据被多线程共享，可能会出现死锁的情况

进程和线程

进程：计算机正在运行的一个独立的应用程序

线程：线程是组成进程的基本单位，一个进程由一个或多个线程组成的

进程和线程都是应用程序在执行过程中的概念，如果应用程序没有执行，则不存在进程和线程的概念，是一个动态的概念。

进程和线程的区别在于进程在运行时拥有独立的内存空间，每个进程所占用的内存都是独立的，互不干扰。

多个线程共享内存空间，彼此之间的运行是独立的。

多线程

在一个进程中，多个线程同时执行。

系统会自动为每个线程分配 CPU 资源，在某个具体的时间段内 CPU 会被一个线程占用，在不同的时间段由不同的线程来使用 CPU 资源。

线程 任务

Java 中线程的使用

Thread 类：用来创建线程对象的

Runable 接口：表示线程要执行的任务

> 继承 Thread 类

Thread 是 JDK 提供的一个类，专门用来创建线程对象，抽象化的描述，具体使用的时候需要实现具体的类

```java
public class MyThread extends Thread {
    @Override
    public void run() {
        for (int i = 0; i < 100; i++) {
            System.out.println("李四打水.............");
        }
    }
}
```

> 实现 Runnable 接口

耦合度过高，一个线程只能执行一个任务，无法做到任务到线程的自由分配

把线程对象和任务进行解耦合

将线程和任务进行分离

1、创建一个任务

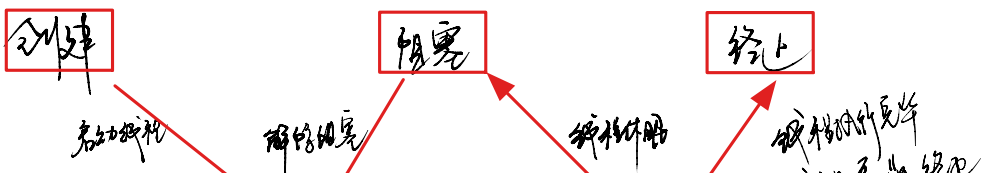
2、创建一个原生的线程对象，并将任务分配给线程对象

```java
public class MyRunnable1 implements Runnable {
    @Override
    public void run() {
        for (int i = 0; i < 100; i++) {
            System.out.println("。。。。。。取快递");
        }
    }
}
```

```java
public class Test {
    public static void main(String[] args) {
        MyRunnable1 myRunnable1 = new MyRunnable1();
        MyRunnable2 myRunnable2 = new MyRunnable2();
        Thread thread1 = new Thread(myRunnable2);
        Thread thread2 = new Thread(myRunnable1);
        thread1.start();
        thread2.start(); = new Thread(new MyRunnable()).start();
    }
}
```

lambda 表达式：函数式编程，可以将方法的具体实现作为参数进行传递 // 由接口名字生成

```java
public class Test {
    public static void main(String[] args) {
        new Thread(()->{
            for (int i = 0; i < 100; i++) {
                System.out.println(i);
            }
        }).start();
    }
}
```



线程的状态

线程共有 5 种状态，在特定的情况吗下，线程可以在不同的状态之间切换，5 种状态如下：

- 创建状态：实例化了一个新的线程对象，还未启动
- 就绪状态：创建好的线程对象调用了 start() 方法完成启动，进入线程池等待抢占 CPU 资源
- 运行状态：线程对象获取了 CPU 资源，在一定的时间内执行任务
- 阻塞状态：正在运行的线程暂停执行任务，释放所占用的 CPU 资源，并在解除阻塞之后也不能直接回到运行状态，而是重新回到就绪状态，等待获取 CPU 资源
- 终止状态：线程运行完毕或因为异常导致线程终止执行

线程调度

线程休眠：让当前线程暂停执行任务，从运行状态进入阻塞状态，将 CPU 资源让给其他线程的一种调度方式，通过调用 sleep 方法来实现，线程休眠需要传入具体的休眠时间。

```
public class Test {
    public static void main(String[] args) {
        new Thread(()->{
            for (int i = 0; i < 10; i++) {
                if(i == 5){
                    try {
                        Thread.sleep(5000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                System.out.println(i+"----------Thread");
            }
        }).start();
    }
}
```

线程合并：将指定的某个线程加入到当前线程中，合并为一个线程，由两个线程交替执行变成一个线程中的两个子线程顺序执行。

合并之后两个任务变成顺序执行，且优先执行合并进来的任务。

```
public class Test {
    public static void main(String[] args) {
        Thread thread = new Thread(()->{
            for (int i = 0; i < 50; i++) {
                System.out.println(i + "----------JoinRunnable");
            }
        });
        thread.start();

        for (int i = 0; i < 100; i++) {
            if(i == 20){
                try {
                    thread.join();
                } catch (InterruptedException e) {
                    e.printStackTrace();
```

```
                }
            }
            System.out.println("main+++++++++++++++" + i);
        }
    }
}
```