# 线程的介绍

## 程序、进程、线程

程序：一段静态的代码，是静态的对象。

进程：程序的一次执行过程，或是正在进行的一个程序。进程是资源分配的单位。系统为线程分配内存。

线程：进程可以进一步地细化为线程。是一个程序内部的一个执行路径。一个进程内可能有多个线程。线程是调度和执行的单位，每个线程有独立地运行栈和程序计数器。同一个进程中的线程共享堆，在通信上更方便，但是有安全隐患。

一个Java应用程序java.exe，至少有3个线程：main()主线程，gc()垃圾回收线程，异常处理线程。

## 并行、并发

并行：多个CPU同时执行多个任务。

并发：一个CPU[采用时间片策略]“同时”执行多个任务。

## 使用多线程的优点

提高应用程序的响应。对图像化界面更加有意义。

提高CPU的利用率。

改善程序结构。将长且复杂的进程分为多个线程，利于理解和修改。

## 什么时候使用多线程

程序需要同时执行两个及以上的任务。

程序需要执行需要等待的任务。

一些后台运行的程序。

# 线程的创建与使用\*

## 线程的创建方式一——继承于Thread类

1. 创建一个继承于Thread类的子类
2. 重写Thread类的run()，将此线程的操作写在run()方法中
3. 创建Thread类的子类的对象
4. 通过这个对象调用start()

Start()有两个作用：启动当前线程，调用当前线程的run()方法。

public class ThreadTest {

public static void main(String[] args) {

// 3. 创建子类的对象

MyThread t1 = new MyThread();

// 4. 调用start()

// 不能直接调用run()

// 不可以重复执行start()，需要重新创建一个对象，要想创建多个线程就要创建多个对象

t1.start();

for (int i = 0; i < 1000; ++i) {

System.*out*.println("main " + i);

}

}

}

// 1. 创建一个Thread类的子类

class MyThread extends Thread {

// 2. 重写run()方法

@Override

public void run() {

for (int i = 0; i < 1000; ++i) {

System.*out*.println(i);

}

}

}

## 线程的创建方式二——实现Runnable接口

1. 创建一个实现了Runnable接口的类
2. 实现类去实现Runnable中的抽象方法run()
3. 创建实现类的对象
4. 将此对象作为参数传递到Thread类的构造器中，创建Thread类的对象
5. 通过Thread类的对象调用start()

// 1. 创建一个实现了Runnable接口的类

class MyThread implements Runnable {

// 2. 实现Runnable中的抽象方法run()

@Override

public void run() {

for (int i = 0; i < 100; i++) {

System.*out*.println(Thread.*currentThread*().getName() + i);

}

}

}

public class ThreadTest1 {

public static void main(String[] args) {

// 3. 创建实现类的对象

MyThread myThread = new MyThread();

// 4. 将此对象作为参数传递到Thread类的构造器中，创建Thread类的对象

Thread thread1 = new Thread(myThread);

// 5. 通过Thread类的对象调用start()

thread1.start();

// 再创建一个线程

Thread thread2 = new Thread(myThread);

thread2.start();

}

}

## 比较方式一和方式二

开发中优先选择方式二：

继承的方式受限于单继承

实现的方式天然地可以共享数据

Thread类本身也是现实的Runnable接口

## 线程的调度

1. 时间片
2. 优先级

线程优先级1到5，默认是5

getPriority(int p)方法获得优先级

setPriority(int p)方法设置优先级

# 线程的生命周期

## 线程的状

# 线程的同步\*

解决的是线程安全问题。

# 线程的通信

# Thread类的常用方法

## start()方法

1. 启动当前线程
2. 调用当前线程的run()

## run()方法

创建的线程要执行的操作在run()中重写

## currentThread()方法

静态方法，返回执行当前代码的线程

## getName()方法和setName()方法

获得和设置当前线程的名字

## yield()方法

释放当前CPU的执行权

## join()方法

在线程a中调用线程b的join()，此时线程a进入阻塞状态，直到线程b完全执行之后，a结束阻塞状态。