# 进程与线程的区别

main 是主线程 如果是自己创建的线程是子线程 gc线程（守护线程，专门回收垃圾）

进程其实是一个应用程序。

进程是所有线程的集合，每一个线程是进程中的一条执行路径。

# 为什么要用多线程

生活问题：

现在有一千克的水，目前只有小明一个人去打水，但是小明每小时打水200千克，现在要求一小时内把水全部打完，请问怎么解决？

如果小明一个人将水全部打完需要五个小时。

解决办法:

在加四个人同时打水，分别为小军、小红、小玲、小小，加上小明一共五个人同时去打水，五个人每小时打水200千克，那么一小时后就可以完成打完一顿水。

主要能体现到多线程提高程序效率。

举例: 迅雷多线程下载、分批发送短信等。

# 多线程创建方式

## ①继承Thread类 重写run方法

|  |
| --- |
| **class CreateThread extends Thread {**  **// run方法中编写 多线程需要执行的代码**  **publicvoid run() {**  **for (inti = 0; i< 10; i++) {**  **System.*out*.println("i:" + i);**  **}**  **}**  **}**  **publicclass ThreadDemo {**  **publicstaticvoid main(String[] args) {**  **System.*out*.println("-----多线程创建开始-----");**  **// 1.创建一个线程**  **CreateThread createThread = new CreateThread();**  **// 2.开始执行线程 注意 开启线程不是调用run方法，而是start方法**  **System.*out*.println("-----多线程创建启动-----");**  **createThread.start();**  **System.*out*.println("-----多线程创建结束-----");**  **}**  **}** |

## ②实现Runnable接口,重写run方法

|  |
| --- |
| **class CreateRunnable implements Runnable {**  **@Override**  **publicvoid run() {**  **for (inti = 0; i< 10; i++) {**  **System.*out*.println("i:" + i);**  **}**  **}**  **}**  **publicclass ThreadDemo2 {**  **publicstaticvoid main(String[] args) {**  **System.*out*.println("-----多线程创建开始-----");**  **// 1.创建一个线程**  **CreateRunnable createThread = new CreateRunnable();**  **// 2.开始执行线程 注意 开启线程不是调用run方法，而是start方法**  **System.*out*.println("-----多线程创建启动-----");**  **Thread thread = new Thread(createThread);**  **thread.start();**  **System.*out*.println("-----多线程创建结束-----");**  **}**  **}** |

## ③使用匿名内部类方式

|  |
| --- |
| **System.*out*.println("-----多线程创建开始-----");**  **Thread thread = new Thread(new Runnable() {**  **public void run() {**  **for (int i = 0; i< 10; i++) {**  **System.*out*.println("i:" + i);**  **}**  **}**  **});**  **thread.start();**  **System.*out*.println("-----多线程创建结束-----");** |

## 使用继承Thread类还是使用实现Runnable接口好？

使用实现实现Runnable接口好，原因实现了接口还可以继续继承，继承了类不能再继承

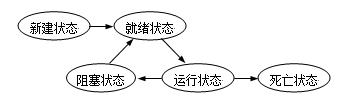
# 在线程中，怎么处理不可捕捉异常

在run方法当中不能抛出异常，只能try捕获

# sleep作用

让当前线程从运行状态变为休眠状态，如果时间到期就回到运行状态。不能释放锁。

# 多线程运行状态



## 新建状态

当用new操作符创建一个线程时， 例如new Thread(r)

## 就绪状态

线程对象调用了start方法，但是并没有获得CPU时间

## 运行状态

线程对象调用了start方法，并获得CPU时间

## 阻塞状态

线程运行过程中，可能由于各种原因进入阻塞状态:  
        1>线程通过调用sleep方法进入睡眠状态；  
        2>线程调用一个在I/O上被阻塞的操作，即该操作在输入输出操作完成之前不会返回到它的调用者；  
        3>线程试图得到一个锁，而该锁正被其他线程持有；  
        4>线程在等待某个触发条件；

## 死亡状态

有两个原因会导致线程死亡：  
   1) run方法正常退出而自然死亡，  
   2) 一个未捕获的异常终止了run方法而使线程猝死

# 多线程分批处理数据

需求:目前蚂蚁课堂有10万个用户，现在蚂蚁课堂需要做活动，给每一个用户发送一条祝福短信。

为了提高数程序的效率，请使用多线程技术分批发送据。

为什么不开10万个线程？

每开一个线程，都会占用CPU资源

服务器（电脑）配置 CPU 核数

## 新建用户实体类

|  |
| --- |
| **publicclass UserEntity {**  **private String userId;**  **private String userName;** |

## 建立多线程UserThread 执行发送短信

|  |
| --- |
| **Class UserThreadextends Thread {**  **private List<UserEntity>list;**  **/\*\***  **\* 通过构造函数 传入每个线程需要执行的发送短信内容**  **\***  **\* @param list**  **\*/**  **public UserThread(List<UserEntity>list) {**  **this.list = list;**  **}**  **\* @methodDesc: 功能描述:(分批发送短信)**  **publicvoid run() {**  **for (UserEntity userEntity : list) {**  **System.*out*.println("threadName:" + Thread.*currentThread*().getName() + "-学员编号:" + userEntity.getUserId()**  **+ "---学员名称:" + userEntity.getUserName());**  **// 调用发送短信具体代码**  **}**  **}**  **}** |

## 初始化数据

|  |
| --- |
| **public static List<UserEntity> init() {**  **List<UserEntity>list = new ArrayList<UserEntity>();**  **for (inti = 1; i<= 140; i++) {**  **UserEntity userEntity = new UserEntity();**  **userEntity.setUserId("userId" + i);**  **userEntity.setUserName("userName" + i);**  **list.add(userEntity);**  **}**  **returnlist;**  **}** |

## 计算分页工具类

|  |
| --- |
| **@classDesc**: 功能描述:(Java集合工具类)  **Public class** ListUtils {  \* **@methodDesc**: 功能描述:(list 集合分批切割)  **Static public**<T> List<List<T>> splitList(List<T>list, **int**pageSize) {  **int**listSize = list.size();  **int**page = (listSize + (pageSize - 1)) / pageSize;  List<List<T>>listArray = **new** ArrayList<List<T>>();  **for** (**int**i = 0; i<page; i++) {  List<T>subList = **new** ArrayList<T>();  **for** (**int**j = 0; j<listSize; j++) {  **int**pageIndex = ((j + 1) + (pageSize - 1)) / pageSize;  **if** (pageIndex == (i + 1)) {  subList.add(list.get(j));  }  **if** ((j + 1) == ((j + 1) \* pageSize)) {  **break**;  }  }  listArray.add(subList);  }  **return**listArray;  }  } |

## 实现发送短信

|  |
| --- |
| **Public staticvoid main(String[] args) {**  **// 1.初始化用户数据**  **List<UserEntity>listUserEntity = *init*();**  **// 2.每一个线程分批跑多少**  **int userThreadPage = 50;**  **// 3.计算创建创建多少个线程并且每一个线程需要执行哪些“分批发送短信用户”**  **List<List<UserEntity>> splitUserList = ListUtils.*splitList*(listUserEntity, userThreadPage);**  **intthreadSaze = splitUserList.size();**  **for (inti = 0; i<threadSaze; i++) {**  **List<UserEntity>list = splitUserList.get(i);**  **UserThread userThread = new UserThread(list);**  **// 4.执行任务发送短信**  **userThread.start();**  **}**  **}** |

# 什么是线程安全问题

当多个线程同时共享，同一个**全局变量或静态变量**，做写的操作时，可能会发生数据冲突问题，也就是线程安全问题。但是做读操作是不会发生数据冲突问题。

# 如何解决多线程之间线程安全问题

使用多线程之间同步synchronized或使用锁(lock)[jdk1.5]

多线程之间同步synchronized分为3种：

①使用同步代码块

什么是同步代码块？

答:就是将可能会发生线程安全问题的代码，给包括起来。

**synchronized(同一个数据){**

**可能会发生线程冲突问题**

**}**

|  |
| --- |
| **private Object mutex = new Object();// 自定义多线程同步锁**  **public void sale() {**  **synchronized (mutex) {**  **if (trainCount > 0) {**  **try {**  **Thread.*sleep*(10);**  **} catch (Exception e) {**  **}**  **System.*out*.println(Thread.*currentThread*().getName() + ",出售 第" + (100 - trainCount + 1) + "张票.");**  **trainCount--; }**  **}**  **}** |

②同步函数

什么是同步函数？

答：在方法上修饰synchronized 称为同步函数

③静态同步函数

什么是静态同步函数？

方法上加上static关键字，使用synchronized 关键字修饰 或者使用类.class文件。

静态的同步函数使用的锁是 该函数所属字节码文件对象

可以用 getClass方法获取，也可以用当前 类名.class 表示。

总结：

synchronized 修饰方法使用锁是当前this锁。

synchronized 修饰静态方法使用锁是当前类的字节码文件

# 为什么使用线程同步或使用锁能解决线程安全问题

将可能会发生数据冲突问题(线程不安全问题)，只能让当前一个线程进行执行。代码执行完成后释放锁，让后才能让其他线程进行执行。这样的话就可以解决线程不安全问题

# 什么是多线程之间同步

当多个线程共享同一个资源,不会受到其他线程的干扰

# 同步函数用的是什么锁及证明方式

同步函数使用this锁

证明：一个线程使用同步代码块(this明锁),另一个线程使用同步函数。如果两个线程抢票不能实现同步，那么会出现数据错误

|  |
| --- |
| **class ThreadTrain5 implements Runnable {**  **// 这是货票总票数,多个线程会同时共享资源**  **private int trainCount = 100;**  **public boolean flag = true;**  **private Object mutex = new Object();**  **@Override**  **public void run() {**  **if (flag) {**  **while (true) {**  **synchronized (mutex) {**  **if (trainCount > 0) {**  **try {**  **Thread.*sleep*(40);**  **} catch (Exception e) {**  **}**  **System.*out*.println(Thread.*currentThread*().getName() + ",出售 第" + (100 - trainCount + 1) + "张票.");**  **trainCount--;**  **}**  **}**  **}**  **} else {**  **while (true) {**  **sale();**  **}**  **}**  **}**  **public synchronized void sale() {**  **if (trainCount > 0) {**  **try {**  **Thread.*sleep*(40);**  **} catch (Exception e) {**  **}**  **System.*out*.println(Thread.*currentThread*().getName() + ",出售 第" + (100 - trainCount + 1) + "张票.");**  **trainCount--;**  **}**  **}**  **}**  **public class ThreadDemo5 {**  **public static void main(String[] args) throws InterruptedException {**  **ThreadTrain5 threadTrain = new ThreadTrain5(); // 定义 一个实例**  **Thread thread1 = new Thread(threadTrain, "一号窗口");**  **Thread thread2 = new Thread(threadTrain, "二号窗口");**  **thread1.start();**  **Thread.*sleep*(40);**  **threadTrain.flag = false;**  **thread2.start();**  **}**  **}** |

# 什么是多线程死锁

同步中嵌套同步,导致锁无法释放