进程

概念

就是现在正在运行的程序。也就是代表了程序锁占用的内存区域

特点

独立性进程是系统中独立存在的实体它可以拥有自己的独立的资源，每一个进程都拥有自己私有的地址的地址空间

动态性 进程和程序的去表在于程序只是一个静态的指令集和，而进程是一个正在系统中

活动的指令集合 在进程种加入了时间的概念进程具有自己的生命周期

并发想：多个进程可以在单个处理器上并发执行多个进程之间不会互相影响

工程绘图

描述已自动生成

CPU的分时调度

时间片，即CPU分配给各个程序的时间每个进程被分配一个时间段 称作它的时间片

即该程序允许运行的时间使各个程序从表面上看是同时进行的

如果时间片结束时进程还在运行则CPU将被剥削并分配给另一个进程 将当前进程挂起

如果进程在时间片结束前阻塞或结束 则CPU当即进行切换 而不会造成CPU的资源浪费

当又切换到之前执行的进程把现场恢复 继续执行

在宏观上：我们可以同时打开多个应用程序每个程序并行，同时运行

在微观上由于只有一个CPU一次只能处理程序要求的一部分如何处理公平一种方法就是引入时间片 每个程序轮流执行多核提高了并发性

图形用户界面, Word

描述已自动生成

线程的概念

线程是操作系统能够进行运算调度的最小单位

它被包含在进程中，是进程种的实际运作单位

一个进程可以开启多个线程

多线程扩展了多进程的概念 使得同一个进程可以同时并发处理多个任务

简而言之 一个程序运作后至少一个进程 一个进程包含了一个线程（）

图形用户界面, 图示

描述已自动生成

图形用户界面

描述已自动生成

并行和 并发

CPU处理器

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

多线程的特性

线程执行的随机性

在同一时刻只能有一个程序在执行

我们感觉这些程序像是在同时进行实际上是因为CPU在高校的切换着

时间：1/?ms 切换速度是纳秒的非常快

图示

描述已自动生成日程表

描述已自动生成

多线程的创建1：继承Thread

概述

Thread类本质上实现了Runnable接口的是一个实例，代表一个线程的实例。启动线程的 唯一方法就是通过Thread类的strat()实列方法。Start()方法是一个native方法

它通知底层操作系统最终由操作系统启动一个新的线程操作系统将执行run()方法

这种方式实现多线程很简单。通过自己的类直接extend Thread并复写run()方法，就可以启动新线程并执行自己定义的run()方法



图形用户界面, 文本, 应用程序

描述已自动生成

图形用户界面, 文本, 应用程序, 表格, 电子邮件

描述已自动生成

图形用户界面

中度可信度描述已自动生成

文本

描述已自动生成

多线程创建 实现Runnable接口

概述

如果自己的类已经extends另一个类 ，就无法多继承此时可以实现一个Runnable接口

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

常用方法

方法摘要

Run()使用实现接口Runnable的对象创建一个线程时,启动该线程导致在独立执行的线程种调用对象的run方法

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

测试

图形用户界面, 文本, 应用程序

描述已自动生成

多线程的同步锁

是吧有可能出现问题的代码抱起来 一次只让一个线程执行 通过sychronized关键字实现同步

当多个对象操作共享数据时,可以使用同步锁解决线程的安全问题

Synchronized

Synchronized(对象){

需要同步的代码

}

同步：同一时刻只能有一个线程独占资源没有权利的排队

异步：就是体现了多线程的效果就是互相不等待，互相争抢资源好处就是效率高

Synchronized

特点

前提1：同步需要两个或者两个以上的线程

前提2：多个线程间必须时使用同一个锁

同步的缺点是会降低程序的执行效率 为了保证线程的安全 必须牺牲性能

可以修饰方法成为同步方法使用锁对象是this

可以修饰代码块称为同步代码快 锁对象可以任意

图形用户界面, 文本, 应用程序

描述已自动生成

线程创建的其他方式

ExecutorService/Executors



Api

ExecutorService接口

Execute(Runnable任务对象)把任务丢到线程池

Executors辅助创建线程池的工具类

NewFixedThreadPool(5)最多5个线程的线程池

newCachedThreadPool()足够多的线程使任务不必等待

newSingleThreadExecutor()只有一个线程的线程池

线程锁

悲观锁

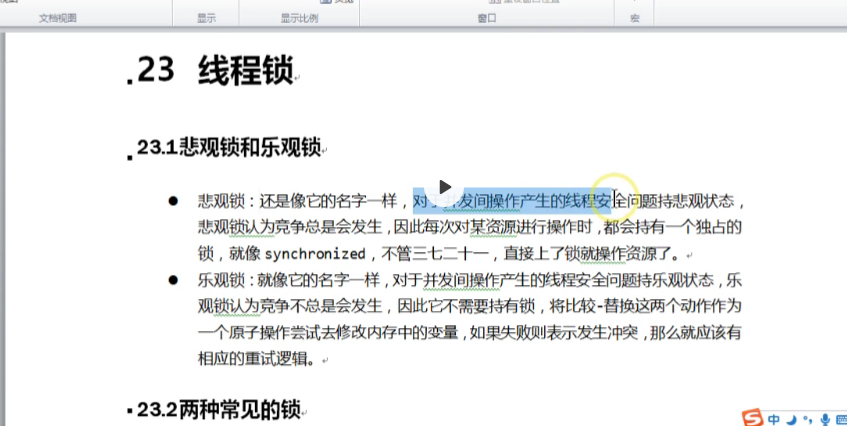
悲观锁：还是像他的名字一样对于，并发间操作产生的线程安全问题保持悲观状态

悲观锁认为竞争总是会发生 因此每次对资源进行操作时，都会持有一个独占的锁

叫做synchronized,不管三七二十一直接上了锁就操作资源了

乐观锁就和他的名字一样 ，对于并发之间的操作的线程安全保持乐观状态，乐观锁认为

竞争不总是会发生因此它不需要锁 将比较-替换这两个动作为一个原子操作尝试去修改内存中的变量如果失败则标识发生冲突，那么久应该有相应的重试逻辑



Synchronized互斥锁（悲观锁，有罪假设）

采用synchronized修饰实现的同步计划之叫做互斥锁机制，它锁获得的锁叫做互斥锁

每个对象都有一个monitor（锁标记），当线程拥有这个锁标记时才能访问这个资源

没有锁标记变进入锁池，任何一个对象系统都会为其创建一个互斥锁

这个锁是为了分配给线程的防止打断原子操作 。每个对象的锁只能分配给一个线程

因此叫做互斥锁

ReentrantReadWriteLock读写锁（乐观锁，无罪假设）

ReentrantLock是排他锁，排他锁在同一时刻仅有一个线程可以进行访问

实际上独占锁是一种相对比较保守的锁策略，在这种清苦那个在任何“读/读”、“读写”

，“写/写”操作都不能同时发生，这在一定程度上降低了吞吐量

然而读操作之间不存在数据竞争问题，如果”读/读”操作能够以共享锁的方式进行

那会进一步提升性能，因此引入了ReentranReadWriteLock，顾名思义

ReentrantReadWriteLock是Reentrant(可重入)Read（读）Write(写)

Lock(锁)我们下面称为读写锁

读写锁内部又分为读锁和写锁。读锁可以在没有写锁的时候被多个线程同时

持有，写锁是独占的读锁和写锁分离从而提升了程序性能读写锁主要应用

在读多写少的场景

