Java语言程序设计

第12章 多线程机制

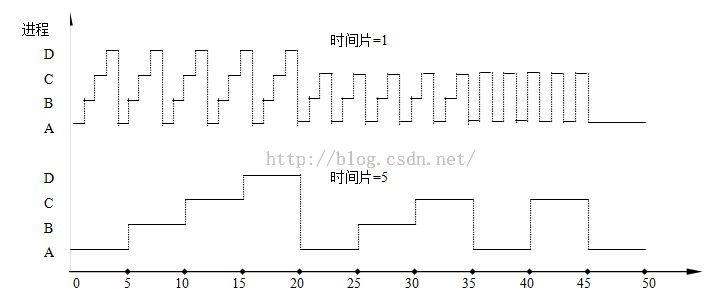
 进程与线程

“多任务”操作系统

多任务处理是指用户可以在同一时间内运行多个应用程序,每个应用程序被称作一个任务。

任务调度机制确保不同任务之间的切换速度十分迅速，因此感觉多个任务同时运行。

主频：CPU内核工作的时钟频率。



进程（Process）

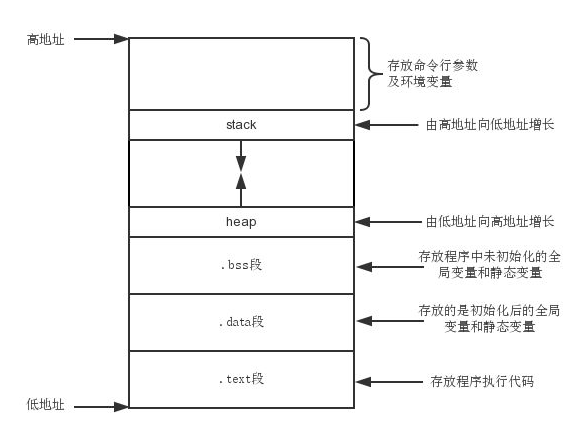
进程（process）是程序的一次动态执行过程，是操作系统分配资源的基本单位。同一个程序可以被加载到不同内存区域分别执行，形成多个进程。进程特性：

独立性：进程是系统中独立存在的实体，可以拥有自己独立的资源，拥有自己私有的内存空间；

动态性：进程是系统中正在执行的指令集合，进程拥有自己的生命周期和状态；

并发性：多个进程可以在单个处理器上并发执行，多个进程之间不会相互影响。

多个进程可以共享资源：剪切板、输入输出设备、打印机等。



“JVM虚拟机进程”

1.Java运行命令“java 类名”启动一个Java虚拟机进程。其中该类中含有main()方法。

2.终止当前正在运行的Java虚拟机：

System.exit(n)等效于Runtime.getRuntime().exit(n)

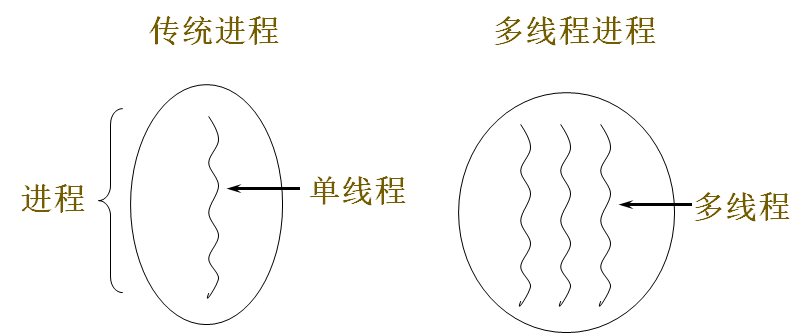
非0 的状态码表示异常终止。

线程（Thread）

1.线程必须依附于进程。一个进程可以拥有多个（≥1）线程。

2.每一个线程都可以启动和停止其它线程，多个线程之间可以并发执行（抢占时间片），线程之间可以共享数据。

3.线程的创建、撤销、切换和数据共享等操作，代价远小于进程，效率也高于进程。

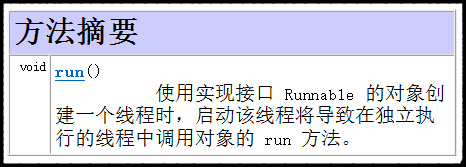


 Java的线程

线程（Thread）也是一种类，每个具体的线程是其对象。



Runnable接口：



一个进程可以拥有多个线程

主线程（main方法线程）

线程分为两种：守护(Daemon)线程和非守护线程。

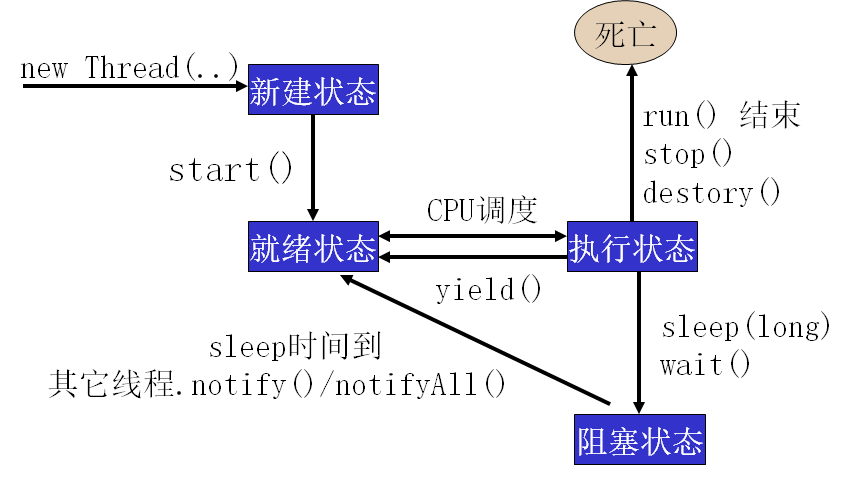
主线程是一种非守护线程，可以创建其它线程，当所有的非守护线程执行结束时，当前进程执行结束。

垃圾回收线程是一种守护线程。

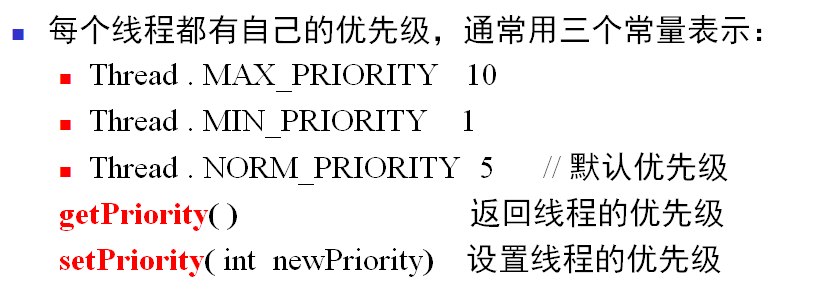
一个线程在运行过程中可以设置其它线程为守护(或非守护)线程。

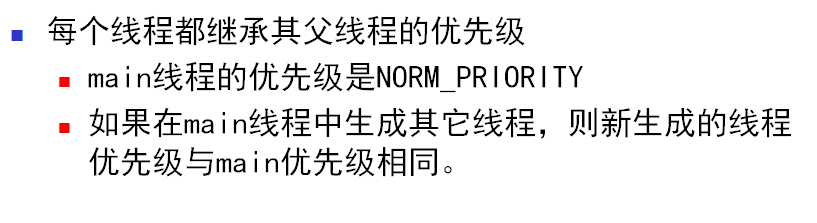
线程的生命周期

**线程在其生命周期中的状态：**



线程的优先级（Priority）





 创建线程

继承Thread类

Thread()

构造方法：

Thread()

Thread(String threadName)

------------------------------------

常用方法：

String getName()

void setName(String newName)

int getPriority()

void setPriority(int newPriority)

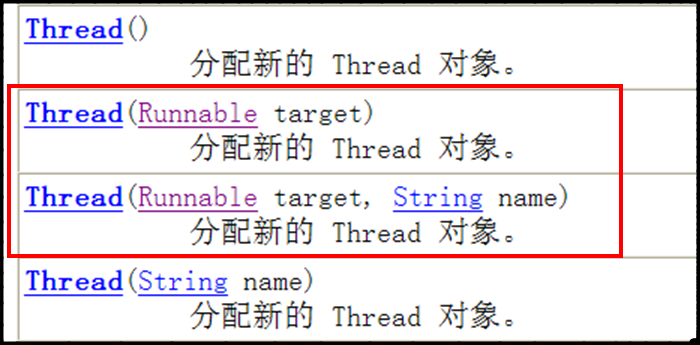
Thread.State getState()

void run() //线程的执行体，需改写

void yield()

实现Runnable接口

构造方法



创建线程过程:

1. class Target implements Runnable

Target类重写 run()方法，作为执行任务

2. 实例化Target类，得到target:"目标对象"

3. new Thread(target, "ThreadName")

“模拟车站售票”

假设：只有一辆火车，座位:1000个

情形1：

只有一个售票窗口。

情形2：

有5个售票窗口。

-----------------------------------

目标对象：售票任务：1000次

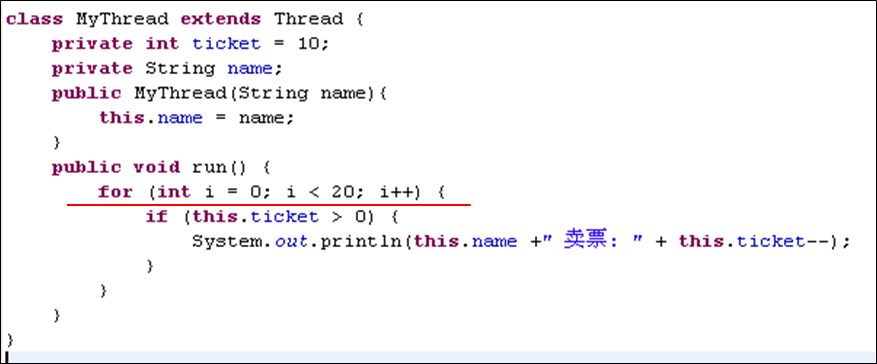
线程对象：5个售票窗口，共享目标对象

 线程的常用方法

start()

run()

注意：不同线程run()中的局部变量在各自的线程中被分配内存空间。这些局部变量互不干扰



sleep(int millsecond)

isAlive()

currentThread()

12.7 线程联合

join()

12.10 守护线程

isDaemon()

setDaemon(boolean)

setDaemon(true) : 设置当前线程为守护线程

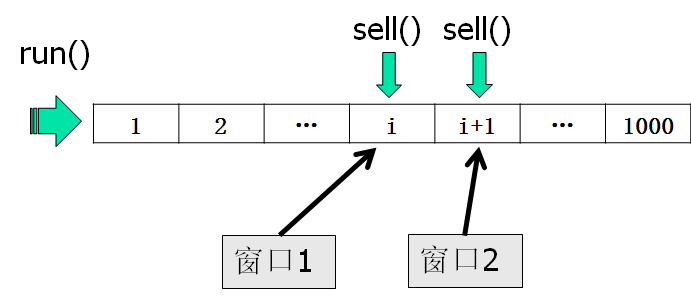
守护线程从接收命令到做出响应需要一定时间

 线程同步

多个线程访问（尤其是有修改操作时）同一个资源（变量或者对象）时，需要互配合：确保同一时刻只有一个线程在修改当前对象。

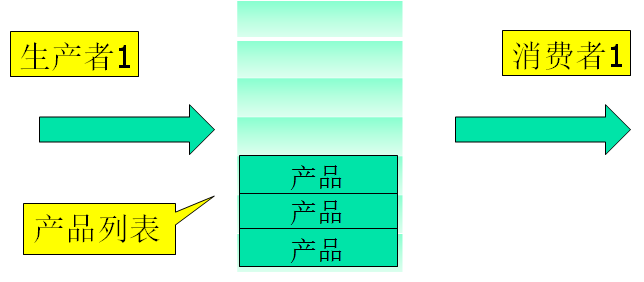
-----------------------------------------

**“火车站售票”问题：**



-----------------------------------------

**“生产者消费者”问题：**



**讨论：多个生产者和多个消费者的情况**

synchronized关键字（“同步”）

Java采用synchronized关键字给需要访问（修改）共享数据的方法加锁，被加锁的方法称为**同步方法。**

即，如果多个线程都可能执行某个方法 f(),为了确保在同一时刻只有一个线程访问f()，该方法可以声明为：

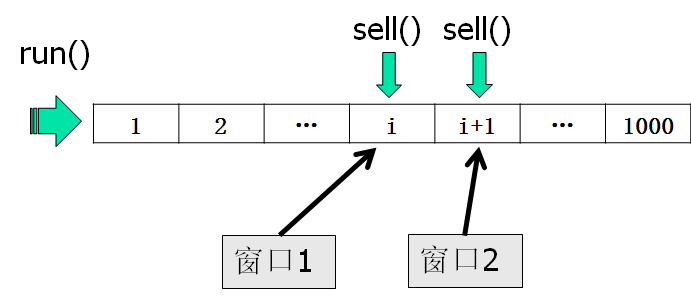
**public synchronized void f() {**

**/\* 修改多个线程共享的数据 \*/**

**}**

修改TicketTask.java

**“火车站售票”问题：**



思路：

1：售票任务类 实现 Runnable接口

成员变量 n = 1000 被多个窗口共享

run() 循环调用 sell()

sell()只卖出一张票：n--，用synchronized修饰

2. 实例化目标对象，多个线程模仿售票窗口。

多个同步方法的情况

若一个类中有多个同步方法中，一次只能有一个同步方法被某个线程执行。

**“会计&出纳”问题：**

账户（Account）：初始值200

会计（Accountant）：存钱:100 \* 3次，间隔1秒

出纳（Cashier）：取钱：50 \* 3次，间隔1秒

思路：

1. P374 Example12.7

**class Bank implements Runnable{**

**int money = 200;**

**public void run(){**

**if(会计线程) saveOrTake(300);**

**if(出纳线程) saveOrTake(150);**

**}**

**public synchronized void saveOrTake(int){**

**if(会计线程) 分三次修改money;**

**if(出纳线程) 分三次修改money;**

**}**

**}**

2. 其它

Account int money

添加两个同步方法：save(int) & take(int)

两个线程类：

Accountant"会计" 执行三次存钱

Cashier“出纳” 执行三次取钱

同步代码块

synchronized关键字不仅可以修饰方法，也可以修饰方法里的某一段代码块。

注意：不能修饰构造方法和成员变量。

方式：

**void f(){**

**synchronized(object){**

**//修改共享变量的代码段**

**//称作"同步代码块"**

**}**

**}**

这里使用的对象object是“同步监视器”，用来防止不同线程执行同步代码块。

一般使用被修改的对象作为同步代码块。

--------------------------------------

思路：

Account 两个非同步方法：save(int) & take(int)

两个线程类：

Accountant 执行存钱（存钱操作作为同步代码块）

Cashier 执行取钱（取钱操作作为同步代码块），如：

**public** **void** run() {

**for**(**int** i=0;i<3;i++) {

//使用account作为监视器，任何线程进入同步代码块之前

//必须先获得account对象的锁。

//--其它对象无法获得对象锁，就无法修改该对象

//过程："获得锁"--"修改对象"--"释放锁"

**synchronized**(account) {

account.take(50);

}

}

}

注：同步方法使用this作为同步监视器

 协调同步线程

设想一种场景：

* 父亲给儿子寄生活费
* 儿子从银行卡中提取
* 如果仅仅使用同步对账号进行存款、取款操作，当儿子急需用钱而余额不足的时候……

多个线程修改同一个变量，线程的执行需要该变量满足一定条件时，需要协调线程

1.synchronized方法(如取钱)中，同步监视器对象调用wait(),当前线程将被中断运行，并且放弃对象的锁。同时该线程进入同步监视器的等待列表。

2.获得锁的线程调用notify(),唤醒等待列表的第一个线程。调用notifyAll(),唤醒等待列表的所有线程。

3.重新获得锁的线程从该线程调用wait()处继续执行。

Object.wait()

Object.notify()

Object.notifyAll()

Father.java

账户（Account）：初始值0，具有同步方法：

存钱deposit(**int** fund)

取钱withDraw(**int** draw)

父亲（Father）：存钱:每隔10s存入1000，存入钱后通知儿子

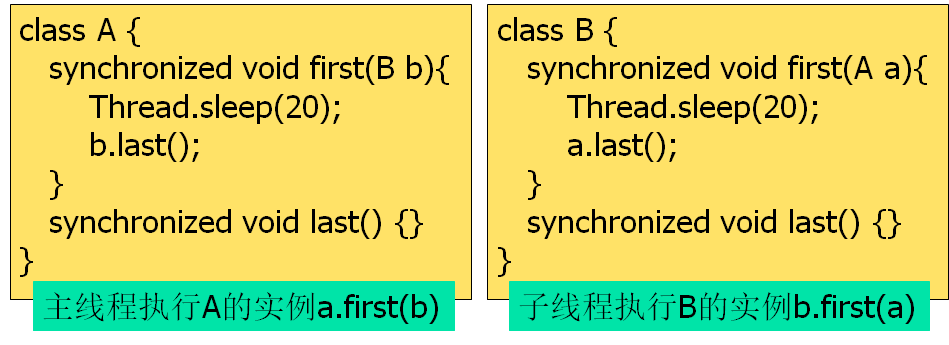
儿子（Son）：取钱：每隔5s取出600，余额不足时等待

死锁

两个线程互相等待对方释放同步监视器时会发生死锁，线程保持监视器对象的锁

死锁时不发生异常，线程处于阻塞状态

JVM不会采取措施处理死锁，只能在编程时规避



其它线程知识点

1.实现Callable接口生成线程

2.同步锁：Lock接口、ReadWriteLock接口等

3.使用Condition控制线程通信（替代同步监视器）

4.使用阻塞队列BlockingQueue控制线程通信

5.线程组ThreadGroup：对一批线程进行管理

6.线程池 ExecutorService pool =

Executors.newFixedThreadPool(int)

7.线程安全的集合类

 GUI线程

垃圾回收机制

Garbage Collection，GC

（1）发现无用的信息对象；System.gc()

（2）回收将无用对象占用的内存空间。

https://www.cnblogs.com/andy-zcx/p/5522836.html