多线程

本章知识点

- 进程与线程
- 线程的创建与执行
- 线程的生命周期
  - 新建、就绪、运行、阻塞、死亡
- 线程的同步
  - synchronized
  - 同步机制
  - 同步代码块、同步方法
- 线程通信
  - Wait(),notify(),notifyAll()
  - 生产者与消费者模型

多线程

- 很多应用程序工作在多线程环境下，发送和接收文件发生在后台（也就是另一个线程中），当有新的对话要发生时，会开启另一个线程……，这样，所有的处理看起来就像是同时进行，用户始终与应用程序的各部分进行着交互。
- 单线程的程序只有一个顺序执行流，多线程的程序则包含多个顺序执行流，多个执行流彼此之间互不干扰。

11.1 线程的概念

- 线程（thread）：进程的执行单元，线程在进程中是独立的、并发的执行流。
  - 当进程被初始化后，主线程就被创建了，该线程从程序入口main()方法开始运行。
  - 多线程：进程内创建的多个线程，每个线程互相独立。使得同一个进程可以同时并发处理多个任务。

11.1 线程的概念

- 进程和线程的关系
  - 每个线程拥有自己独立的数据区：程序计数器、栈内存等，创建新线程时这些数据区会一并创建。

11.2 线程的创建和执行

- 线程：java.lang.Thread实例
- 创建线程的方式
  - 继承Thread类
  - 实现Runnable接口

11.2.1 继承Thread类创建线程

（1）定义Thread的子类，并重写该类的run()方法。run()方法的方法体代表了线程需要完成的任务。

（2）创建线程对象。

（3）线程对象调用start()方法启动该线程。

11.2.1 继承Thread类创建线程

【例11-1】通过Thread类创建线程。

11.2.2 实现Runnable接口创建线程

- Runnable接口只有一个run()方法。
  - 因为Runnable接口和Thread类之间没有继承关系，所以不能直接赋值。
  - 为了使run()方法中的代码在单独的线程中运行，仍需要一个Thread实例，实现对Runnable对象的包装。这样，线程相当于由两部分代码组成：Thread提供线程的支持，Runnable实现类提供线程执行体，即线程任务部分的代码。
  - Thread(Runnable thread)构造方法用于包装Runnable实现类对象，并创建线程。

11.2.2 实现Runnable接口创建线程

【例11-2】通过Runnable接口创建线程。

11.3 线程的状态与生命周期

- 线程由新建、就绪、运行、阻塞、死亡这些状态构成了它的生命周期，呈现了其工作的过程。

11.3.1 新建和就绪状态

- 创建Thread实例：“新建”（new）。
- start()方法启动线程：“就绪”状态（runnable）。

11.3.2 运行状态

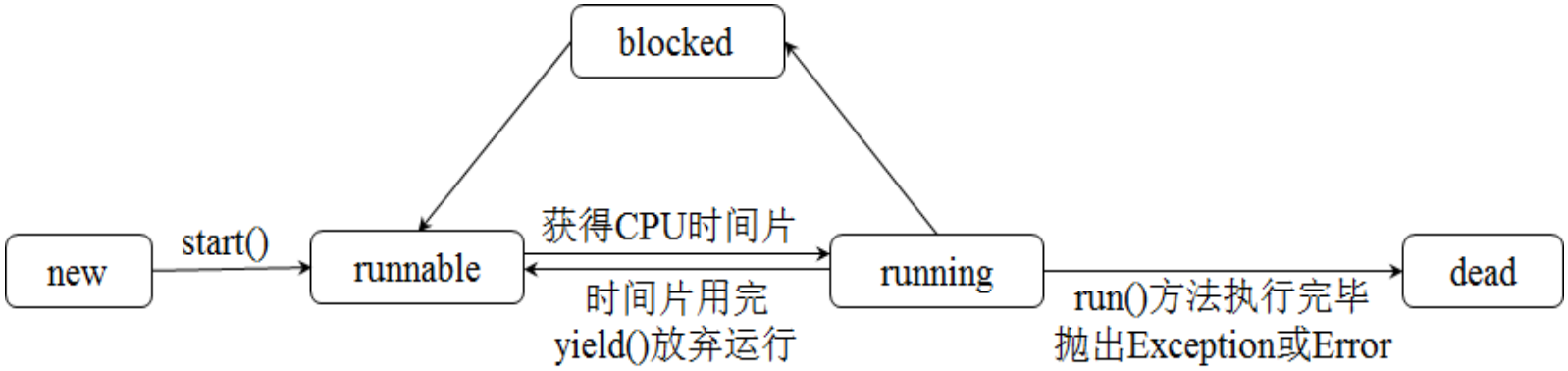
- 处于就绪状态的线程获得了CPU时间片： “运行”状态（running）。
- 操作系统大都采用抢占式的调度策略，即系统给每个可执行的线程一个时间片来处理任务，时间片用完后线程被换下。选择下一个线程时，系统会考虑线程的优先级。

11.3.3 阻塞状态

- 线程在运行状态时，遇到如下状况将会进入各种阻塞状态。
- 睡眠：线程调用sleep()方法睡眠一段时间。
- 资源阻塞：线程在等待一种资源，例如线程调用了阻塞式的I/O方法（等待输入流等），在该方法返回之前该线程被阻止；线程试图获得一个同步锁，但同步锁正被其他线程持有（11.5节详述）。
- 等待：线程调用wait()方法后等候其他线程的唤醒通知（11.6节详述）。

11.3.4 死亡状态

- 线程的run()方法执行完毕，线程正常结束；或者线程执行过程中抛出一个未捕获的异常或错误，线程异常结束。结束后线程处于“死亡”状态（dead）。



11.4 线程优先级与线程调度策略

- 线程调度器：基于优先级的抢先式调度机制。

在设计多线程应用程序时不能依赖线程的优先级。线程调度优先级操作是没有保证的，只能将线程优先级作为一种提高程序效率的方法。

11.5 线程同步

- 当多个线程共享同一个数据时，如果处理不当，很容易出现线程的安全隐患，所以多线程编程时经常需要解决线程同步问题。

11.5.1 数据共享问题

- 【提出问题】夫妻共同进行取钱。
  - ①妻子线程首先执行，检查账户发现账户余额满足取款条件（妻子线程的步骤(1)完成），但在妻子取款之前线程被换下；
  - ②丈夫线程上来后检查账户余额，此时妻子还未取款，丈夫看到的是妻子取款前的账户余额，账户余额也可以满足他的取款要求（丈夫线程的步骤(1)完成），在丈夫取款之前线程被换下；
  - ③妻子线程换上来后接着运行，取走了账户中的全部余额（妻子线程的步骤(2)完成），妻子线程结束；
  - ④丈夫线程换上来后接着运行，因为之前丈夫线程已经确认账户有足够的余额可以支取，于是，丈夫在没有足够余额的情况下仍然进行了取款。

11.5.2 同步和锁机制

- Java使用关键字**synchonized**修饰对象的同步代码块或同步方法。
- “同步”语句块或方法：“原子”操作，让该操作是不可分割的。

```
synchonized 方法签名{
    ……
}
```

```
synchonized (对象) {
    ……
}
```

11.5.2 同步和锁机制

- 同步代码块的处理与之类似，只是同步代码块可以在synchonized后指定加锁的对象，除了this对象外，还可以为其他对象加锁。

锁不属于线程，而是属于对象，一个线程可以拥有多个对象的锁，而只有同一个对象的锁之间才有互斥的作用。

11.5.3 同步代码块

- 因为同步所形成的“原子”操作会损害并发性，所以不要同步原子操作之外的其他代码。如果方法体超出了应同步的操作的范畴，则可以将同步部分缩减为代码块。

11.5.4 同步方法

- 如果一个方法内的所有代码组成“原子”操作，那么可以将该方法定义为同步方法，使用synchronized关键字修饰。

11.6 线程间的通信

- 在多线程环境中，线程之间经常需要协调通信从而共同完成一件任务。Java传统的线程通信是通过Object类中的wait()和notify()方法完成
- Java SE5.0中增加了阻塞队列BlockingQueue等方式控制线程通信。

11.6.1 wait()和notify()方法

- Object类中提供了wait()，notify()和notifyAll()3个方法来操作线程。
- 它们只能在同步代码块或者同步方法内使用，而且只能通过同步监视器（拥有锁的对象）来调用。

对于同步方法，该方法所在类的实例this就是同步监视器，所以可以在同步方法中直接调用这3个方法，如this.wait()直接简写为wait()。

11.6.1 wait()和notify()方法

1. wait()方法

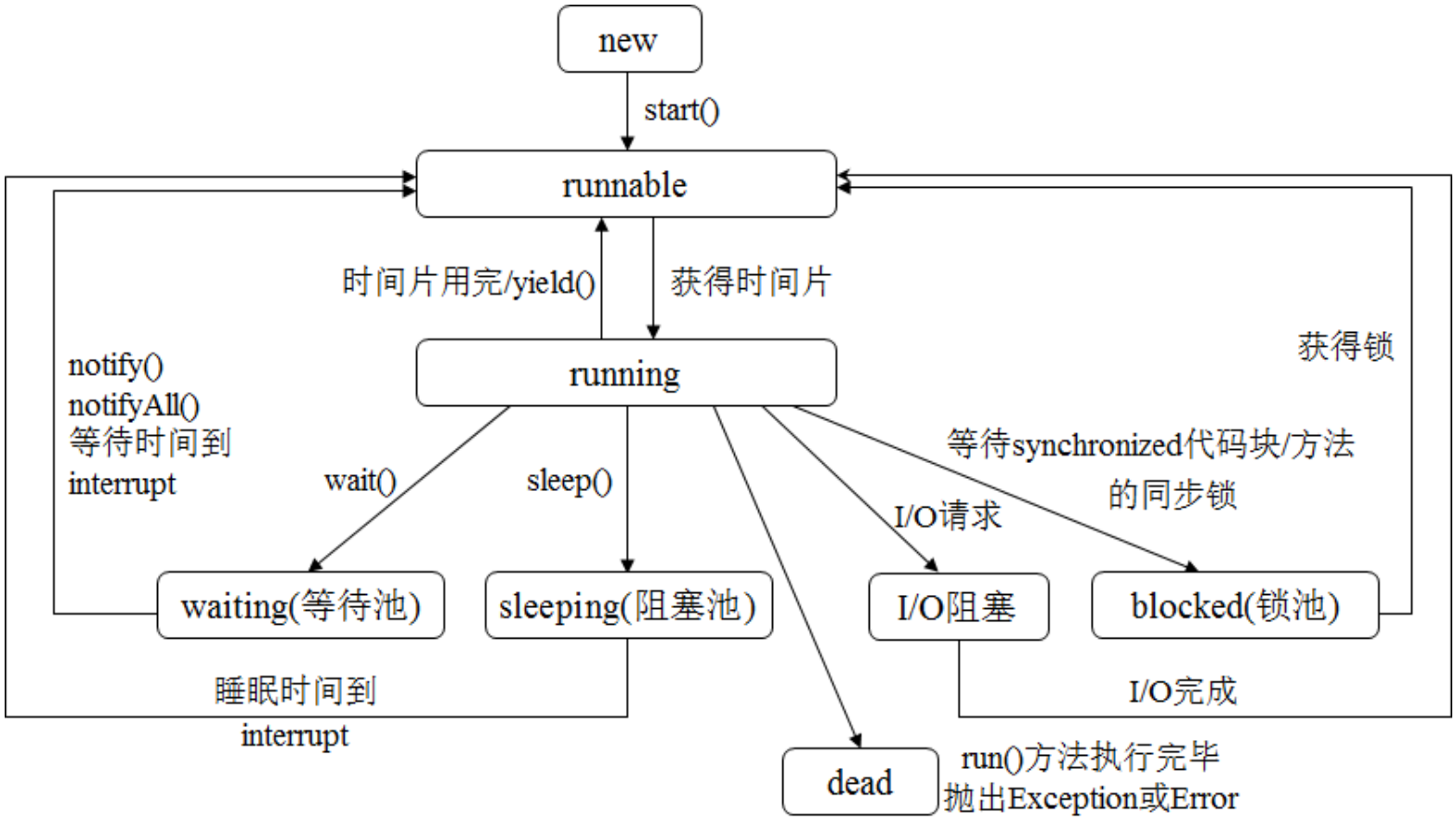
- 在线程已获得对象锁的情形下，如果该线程需要再满足一些条件才能继续执行线程任务，此时该线程可调用wait()方法进入等待池（阻塞状态的一种）。

2. notify()和notifyAll()方法

notify()方法唤醒一个处于等待状态的线程，使之进入runnable状态。

编程时应该用notifyAll()取代notify()，因为用notify()只是从等待池释放出一个线程，至于是哪一个由线程调度器决定，是不可保证的。

11.6.1 wait()和notify()方法



11.6.1 wait()和notify()方法

【例11-8】线程间的通信。写两个线程，线程A“做”10个披萨，线程B“做”20份意大利面，要求线程A每做一个披萨，就通知线程B去做两份意大利面，线程B完成两份意大利面后通知线程A继续做披萨……。

11.6.2 消费者和生产者模型

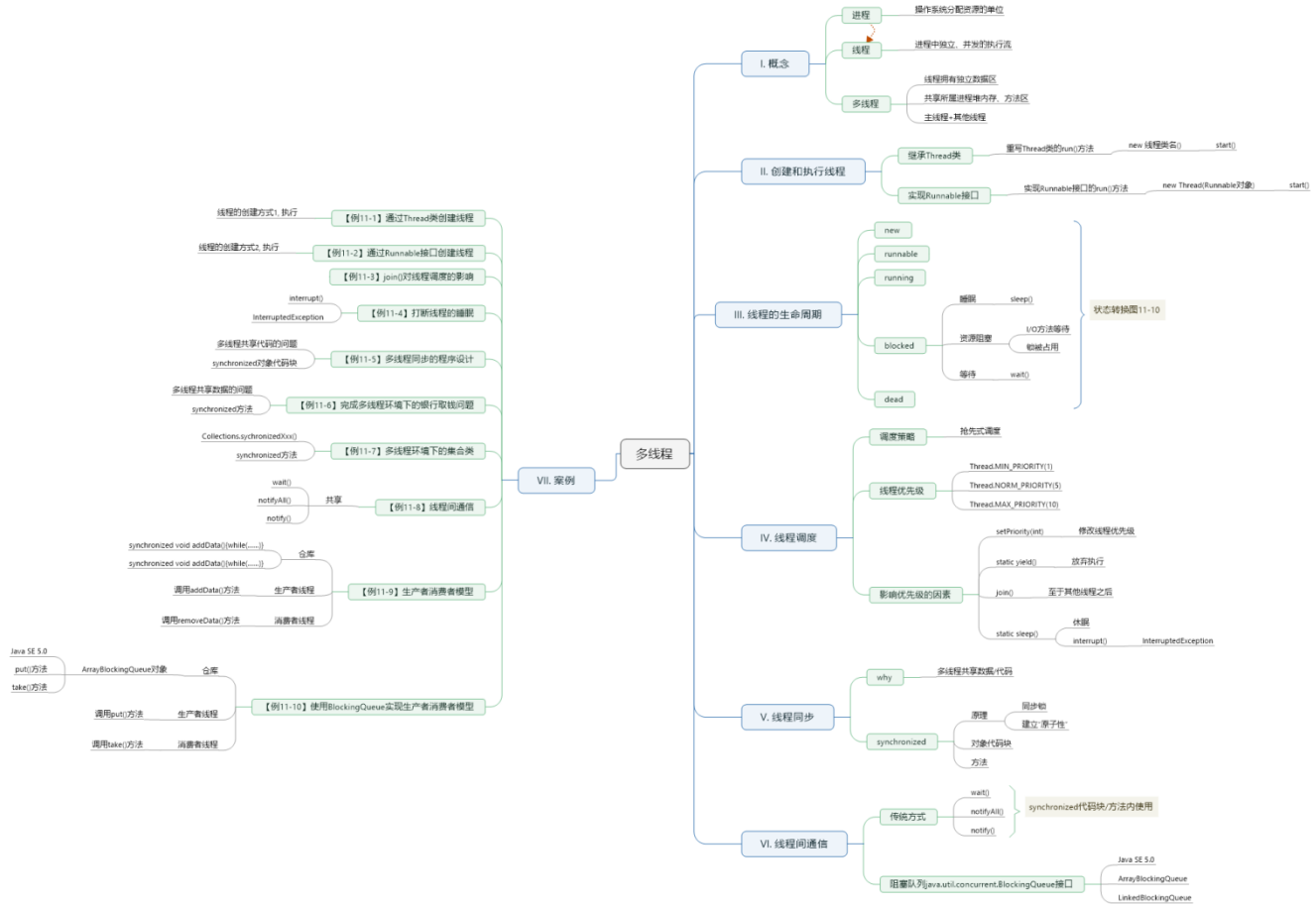
- 【问题描述】生产者和消费者在同一时间段内共用同一个缓冲区（仓库），生产者向缓冲区存放数据，而消费者从缓冲区取用数据。
  - 问题的关键：保证生产者不会在缓冲区满时再加入数据，消费者也不会在缓冲区空时再消耗数据。

11.6.2 消费者和生产者模型

（1）仓库的设计
（2）生产者线程
（3）消费者线程

在生产者消费者模型中，共享缓冲区对象的添加和删除方法必须使用while循环控制临界点。

本章思维导图


多线程
I. 概念
进程
操作系统分配资源的单位
线程
进程中独立、并发的执行流
多线程
线程拥有独立数据区
共享所属进程堆内存、方法区
主线程-其他线程
II. 创建和执行线程
继承Thread类
重写Thread类的run()方法
new 线程类名()
start()
实现Runnable接口
实现Runnable接口的run()方法
new Thread(Runnable对象)
start()
III. 线程的生命周期
new
runnable
running
blocked
睡眠
sleep()
资源阻塞
I/O方法等待
锁被占用
等待
wait()
dead
状态转换图11-10
IV. 线程调度
调度策略
抢先式调度
线程优先级
Thread.MIN_PRIORITY(1)
Thread.NORM_PRIORITY(5)
Thread.MAX_PRIORITY(10)
影响优先级的因素
setPriority(int)
修改线程优先级
static yield()
放弃执行
join()
至于其他线程之后
static sleep()
休眠
interrupt()
InterruptedException
V. 线程同步
why
多线程共享数据/代码
synchronized
原理
同步锁
建立'原子性'
对象代码块
方法
VI. 线程间通信
传统方式
wait()
notifyAll()
notify()
synchronized代码块/方法内使用
阻塞队列java.util.concurrent.BlockingQueue接口
Java SE 5.0
ArrayBlockingQueue
LinkedBlockingQueue
VII. 案例
【例11-1】通过Thread类创建线程
线程的创建方式1, 执行
【例11-2】通过Runnable接口创建线程
线程的创建方式2, 执行
【例11-3】join()对线程调度的影响
【例11-4】打断线程的睡眠
interrupt()
InterruptedException
【例11-5】多线程同步的程序设计
多线程共享代码的问题
synchronized对象代码块
【例11-6】完成多线程环境下的银行取钱问题
多线程共享数据的问题
synchronized方法
【例11-7】多线程环境下的集合类
Collections.sychronizedXxx()
synchronized方法
【例11-8】线程间通信
共享
wait()
notifyAll()
notify()
【例11-9】生产者消费者模型
仓库
synchronized void addData(){while(.....)}
synchronized void addData(){while(.....)}
调用addData()方法
生产者线程
调用removeData()方法
消费者线程
【例11-10】使用BlockingQueue实现生产者消费者模型
Java SE 5.0
put()方法
take()方法
ArrayBlockingQueue对象
仓库
调用put()方法
生产者线程
调用take()方法
消费者线程