复习：

1. 缓冲流：提供了更大的缓冲区---装饰模式

2. 字节流：以字节形式来操作数据。没有默认的缓冲区

3. 转换流：字节和字符之间转换的桥梁。

4. 合并流：需要将多个流放入一个Vector集合中， 利用Vector来产生一个Enumeration对象，利用Enumeration对象来产生合并流对象

5. 序列化/反序列化流：序列化---将对象的信息进行保存的过程。接口--- Serializable，关键字---static/transient，版本号---serialVersionUID

6. Properties：可以持久化的映射。键和值都是String类型。序列化的时候必须存在properties文件中，这个文件的默认编码是西欧编码，不能存储中文。

7. Junit：在工程中第一次使用的时候需要导入测试库。在使用的时候要求方法无参无返回值非静态。

8. 可变参数：用 ... 定义，本质上是一个数组，因此可以以操作数组的方式来操作可变参数。可变参数只能定义一个而且必须放到参数列表的末尾

9. 枚举：适用于取值固定并且能够一一列举的情况。枚举本身是一个特殊的抽象类，在枚举中构造方法默认私有，枚举常量必须定义在第一行。枚举的顶级父类是Enum。从jdk1.5开始，switch-case开始支持枚举常量

## 断言

对结果进行的肯定的预测。---在Java中，断言不是默认开启的。需要参数-ea

## 线程

进程---计算机中在执行的任务

线程---进程中的某一个小任务---线程可以认为是一个简化版的进程。每一个进程中至少包含一个线程。QQ、微信、下载软件、JVM

注意：在计算机中CPU在每一个时刻只能执行一个进程中的一个线程---宏观上，进程是并行的。微观上，进程是串行的。

引入多线程的意义：

1. 减少响应时间

2. 线程在与硬件进行交互的时候CPU处于空闲状态。---多个线程能够减少CPU的空闲状态

1 0.3 0.7

2 0.51 0.7\*0.7=0.49

3 0.7\*0.7\*0.7

### 定义线程

1. 继承Thread类，将需要执行的逻辑放到run方法中，创建线程对象，调用start方法来启动这个线程

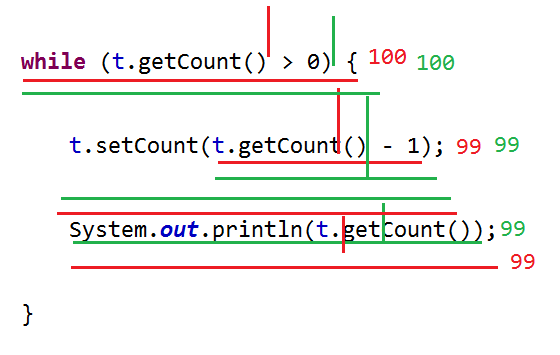
2. 实现Runnable接口，重写run方法。利用Runnable对象来构建一个Thread对象，然后调用start方法来开启这个线程

3. 实现Callable接口，重写call方法---现阶段了解即可

### 多线程的并发安全问题

由于多个线程相互抢占导致出现了不符合常理的数据的情况----多线程的并发安全问题

多个线程在执行的时候是相互抢占资源的，而且抢占是发生在线程执行的每一步过程中。



同步代码块---锁对象，要求必须被所有的线程都认识---共享资源，类的字节码，this

同步：某一个对象或者逻辑在同一个时刻内只能被一个线程操作

异步：某一个对象或者逻辑在同一个时刻内被多个线程操作

宏观上，同步一定安全，不安全一定异步

微观上，同步一定安全，安全一定是同步

由于多个线程之间的锁导致线程相互等着而致使程序无法继续执行---死锁

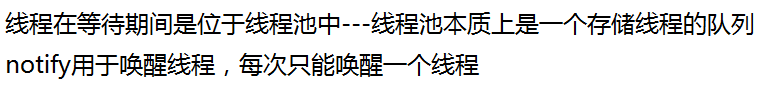
死锁产生的原因：多个线程、锁不统一、锁之间相互嵌套

避免死锁：减少线程数量、统一锁，减少嵌套

活锁的危害：没有使用资源会导致资源浪费，一直在占用CPU导致其他线程无法执行

### 等待唤醒机制

利用wait-notify/notifyAll实现了线程间的交互执行的过程



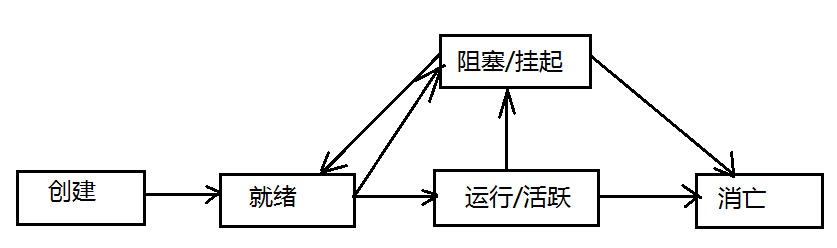
注意：wait/notify/notifyAll必须在同步代码块/同步方法中使用。调用wait的对象和锁对象必须一致。

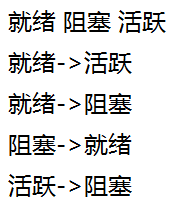
总结：sleep和wait的异同点

sleep使当前线程休眠。需要指定休眠时间，时间一到会自动醒来。释放执行权（无锁状态下），不释放锁。被设计在Thread类上，是一个静态方法

wait使当前线程等待。可以指定等待时间。如果没有指定等待时间就需要手动唤醒。释放执行权，也释放锁。被设计在Object类上，是一个普通的成员方法。

### 线程的状态





### 守护线程

守护其他线程。如果被守护的线程结束，守护线程无论执行完成与否都要随之结束。

一个线程要么是守护线程要么是被守护的线程---直到最后一个被守护的线程结束，所有的守护线程才会随之结束

一个线程只要不手动设置就是被守护的线程。---GC本身是一个守护线程

### 线程的优先级

线程有1-10个优先级。理论上数字越大优先级越高，就意味着这个线程抢占到资源的概率就越高。实际应用中，相邻的两个优先级的差异不明显的。至少需要相差5个优先级及以上才会相对明显一点

### 单例模式

在全局过程只存在一个实例。---往往用于全局的配置

饿汉式的特点是无论是否需要这个对象，需要创建这个对象，这就会导致如果不需要这个对象会使加载时间变长。不会出现多线程的并发安全问题

懒汉式的特点是在对象需要的时候再创建，从而节省加载时间。但是会导致多线程的并发安全问题。

扩展：了解单例模式的七种实现