

1. **线程概述**
   1. **线程相关概念**

进程

1. 进程（process）是计算机中的程序关于某数据集合上的一次运行活动，是操作系统进行资源分配与调度的基本单位
2. 可以把进程简单的理解为正在操作系统中运行的一个程序

线程

1. 线程（thread）是进程的一个执行单元
2. 一个线程就是进程中一个单一顺序的控制流，进程的一个执行分支

进程是线程的容器，一个进程至少有一个线程，一个进程中也可以有多个线程；

在操作系统中是以进程为单位分配资源，如虚拟存储空间，文件描述符等。每个线程都有各自的线程栈，自己的寄存器环境，自己的线程本地存储；

栈：

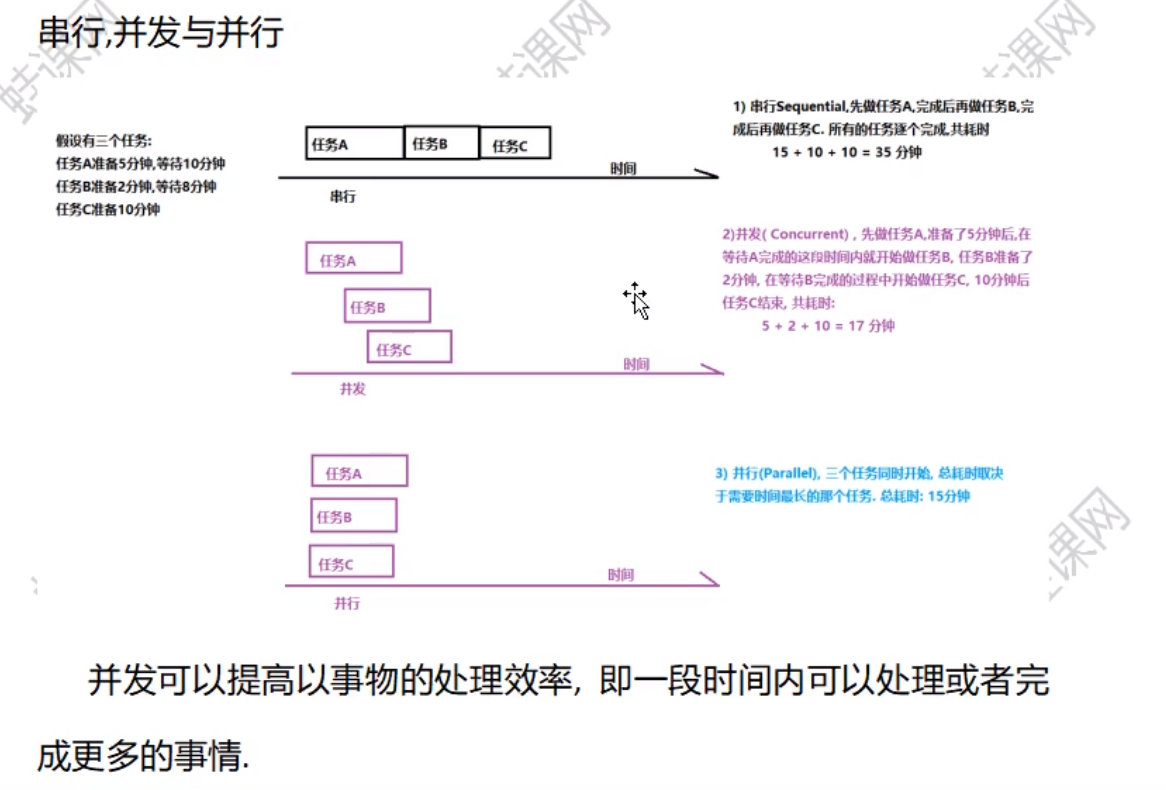
堆：

主线程与子线程

JVM启动的时候会创建一个主线程，该主线程负责执行main方法，主线程就是云顶main方法的线程；

Java中的线程不孤立，线程之间存在一些联系，如果在A线程中创建了B线程，称B线程为A线程的子线程，相应A县城就是B线程的父线程；

串行、并发、并行



并行是一种更为严格，理想的并发。

* 1. **线程相关概念**

在java中，创建一个线程就是创建一个Thread类（子类）的对象（实例）。

Thread类有两个常用的构造方法：Thread（）与Thread（Runnable）对应的创建线程的两种方式：

定义Thread类的子类

定义一个Runnable接口实现类

这两种创建线程的方式没有本质的区别

* 1. **线程相关概念**

**1.3.1 currentThread()方法**

thread.currentThread()方法可以获得当前线程

Java中的任何一段代码都是执行在某个线程当中的，执行当前代码的线程就是当前线程；

同一段代码可能被不同的线程执行，因此当前线程是相对的，Thread.currentThread()方法的返回值是在代码实际运行时候的线程对象；

**1.3.2 setName()/getName()**

tThread.setName(线程名称),设置线程名称

thread.getName（）返回线程名称

**1.3.3 isAlive()**

thread.isAlive（）判断当前线程是否处于活动状态；

活动状态就是线程已启动并且尚未终止；

**1.3.4 isAlive()**

thread.sleep(millis)；让当前线程休眠指定的毫秒数

当前线程是指thread.currentThread()返回的线程

**1.3.5 getId()**

thread.getId()可以获得线程的唯一标识

注意：

某个编号的线程运行结束后，该编号可能被后续创建的线程使用（用完就归还）

重启的JVM后，同一个线程的编号可能不一样

**1.3.6 yield()**

thread.yield()方法的作用是放弃当前的CPU资源，

**1.3.7 setPriority()**

thread.setPriority(sum);设置线程的优先级

java线程的优先级气质范围是1-10，如果超出这个范围会抛出

异常IllegalArgumentException；

在操作系统中，优先级较高的线程获得CPU的资源越多

线程优先级本质上是：只是给线程调度器一个提示信息，以便于调度器决定先调度哪些线程；注意不能保证优先级高的线程先运行。

Java优先级设置不当或者滥用可能会导致某些线程永远无法得到运行，即产生了**线程饥饿**；

线程的优先级并不是设置的越高越好，一般情况下使用普通的优先级即可，即在开发时不必设置线程的优先级；

线程的优先级具有继承性，在A线程中创建了B线程，则B线程的优先级与A线程是一样的；

**1.3.8 interrupt()**

中断线程；

注意调用interrupt（）方法仅仅是在当线程打一个停止标志，并不是真正的停止线程

**1.3.9 setDaemon()**

Java中线程分为用户线程与守护线程

守护线程是为其他线程提供服务的线程，如垃圾回收器（GC）就是一个典型的守护线程

守护线程不能单独运行，当JVM中没有其他用户线程，只有守护线程时，守护线程会自动销毁，JVM会退出；

* 1. **线程的生命周期**

线程的生命周期是线程对象的生来病死，即线程的状态

线程生命周期可以通过getState（）方法获得，线程的状态

Thread.state枚举类型定义的，有以下几种：

NEW，新建状态，创建了线程对象，在调用start（）启动之前的状态；

RUNNABLE，可运行状态，它是一个复合状态，包含：READY和RUNNING两个状态。

READY状态该线程可以被线程调度器进行调度使它处于RUNNING状态，RUNING状态表示该线程正在执行。

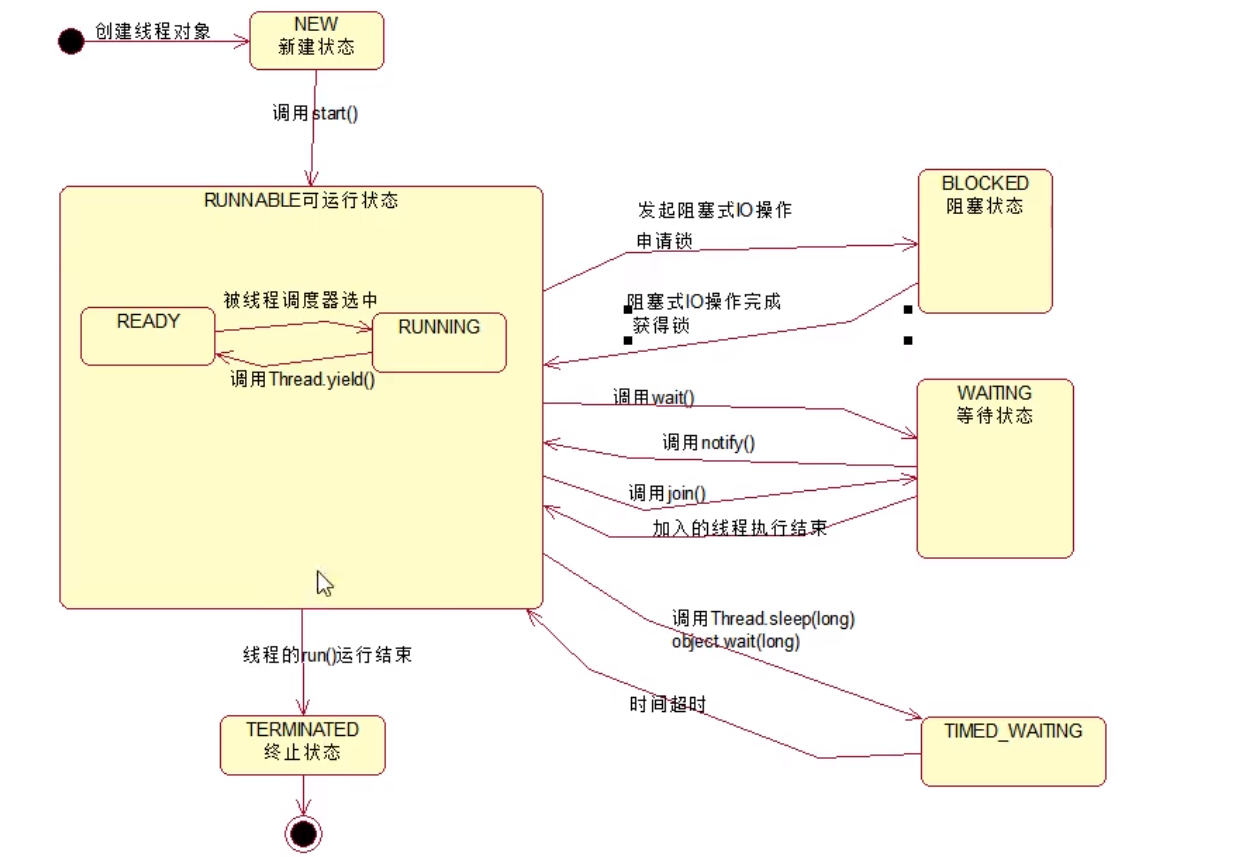
Thread.yield（）方法可以把线程由RUNNING状态转换为READY状态；

BLOCKED阻塞状态，线程发起阻塞的I/O操作，或者申请由其他线程占用的独占资源，线程会转换为BLOCKED阻塞状态，处于阻塞状态的线程不会占用CPU资源。当阻塞I/O操作执行完，或者线程获得了其申请的资源，线程可以转换为RUNNABLE。

WAITING等待状态，线程执行了object.wait()，thread.join()方法会把线程转换为WAITING等待状态，执行object.notify()方法，或者加入的线程执行完毕，当前线程会转换为RUNNABLE状态

TIMED\_WAITING状态，与WAITING状态类似，都是等待状态，区别在于处于该状态的线程不会无限的等待，如果线程没有在指定的时间范围内完成期望的操作，该线程自动转换为RUNNABLE；

TERMINATED终止状态，线程结果处于终止状态；



* 1. **线程的生命周期**

多线程编程具有一下优势：

1. 提高系统的吞吐率（Throughout），多线程编程可以使一个进程有多个并发（concurrent，即同时进行的）的操作
2. 提高响应性（Responsiveness），web服务器会采用一些专门的线程负责用户的请求处理，缩短了用户的等待时间
3. 充分利用多核（Multicore）处理器资源，通过多线程可以充分的利用CPU资源

多线程编程存在的问题与风险：

1. 线程安全（Thread safe）问题，多线程共享数据时，如果没有采取正确的并发访问控制措施，就可能会产生数据一致性问题，如读取脏数据（过期的数据），如丢失数据更新
2. 线程活性问题（thread liveness）问题，由于程序自身的缺陷或者由资源稀缺性导致县城一直处于非RUNNABLE状态，这就是线程活性问题，常见的活性故障有以下几种：
3. 死锁（Deadlock）
4. 锁死（Lockout）
5. 活锁（Livelock）
6. 饥饿（Starvation）
7. 上下文切换（Context Switch），处理器从执行一个线程切换到执行另外一个线程
8. 可靠性，可能会由一个线程导致JVM意外终止，其他的线程也无法执行