并发编程基础篇

尹洪亮 | Kevin.Yin

互联网架构师 / 自由讲师

每天都要让自己比别人多努力一分钟

Kevin、让你每天进步一点点

我的微信 liang19871023liang

加微信，获取**项目源码、高清课件**

加微信，所有**新课七折**优惠，职业规划，互动答疑，免费资料

加微信，受邀进入**KEVIN社区**，与大咖和同龄人会面

关注公众号，每周推送**Kevin原创文章**，不定期优惠活动

声明

PPT风格差异、IDE工具差异：Eclipse与Idea

好的课程都是不断的迭代、优化、补充形成的

目的是知识体系完整性、讲真实有用的东西

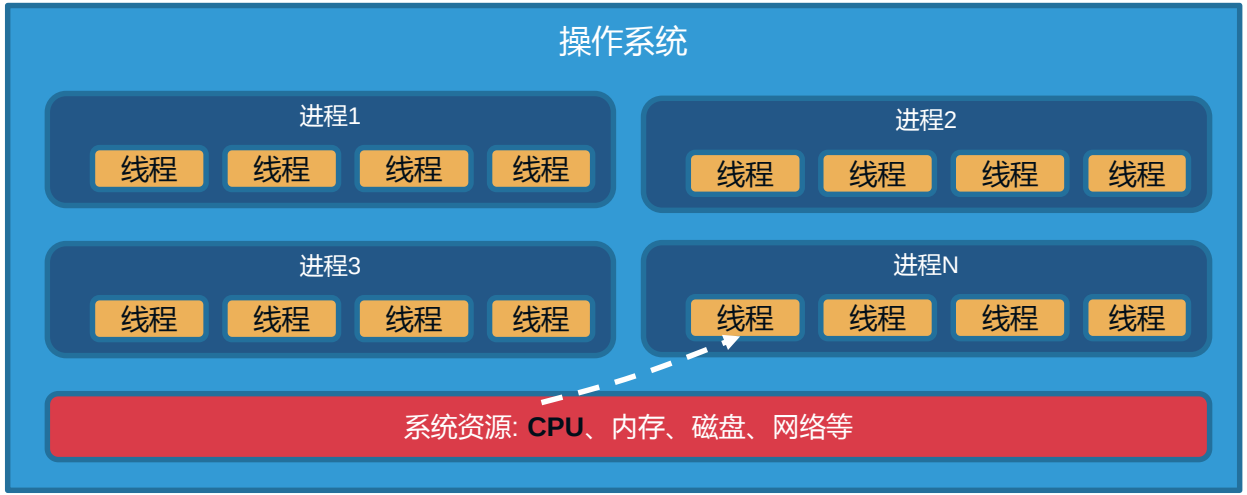
并发编程对于录制时间没有强制要求

进程 与 线程

想要探索线程、必须先理解进程、以及进程与线程之间的关系

进程与线程的图像化

进程与线程的相互依存



进程是系统进行资源分配和调度的基本单位，一个进程中至少有一个线程，进程中的多个线程共享进程的资源。

线程是进程中的一个实体，线程是不会独立存在的！ 所以说，没有进程就就没有线程。

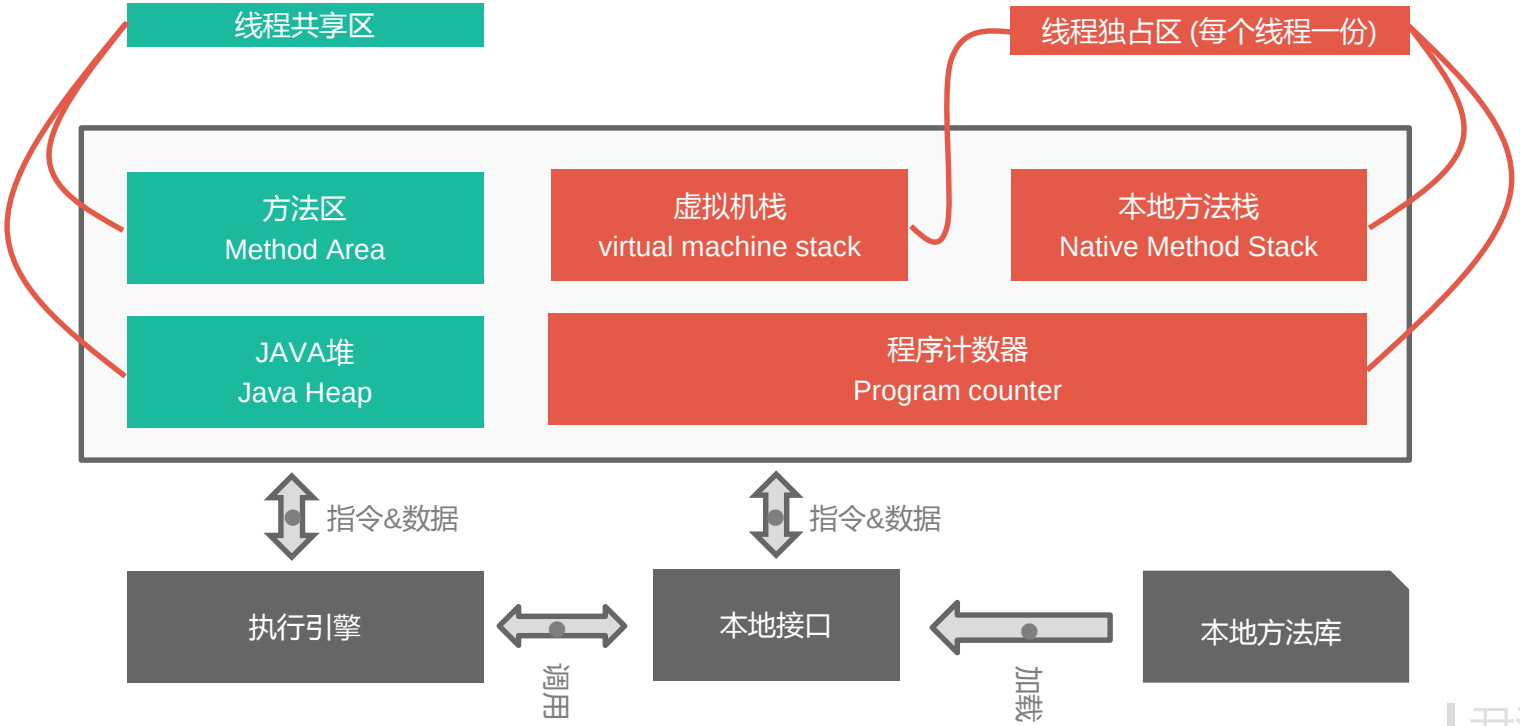
对于CPU资源比较特殊，线程才是CPU分配的基本单位。

main函数启动 》JVM进程 》main函数线程称为主线程

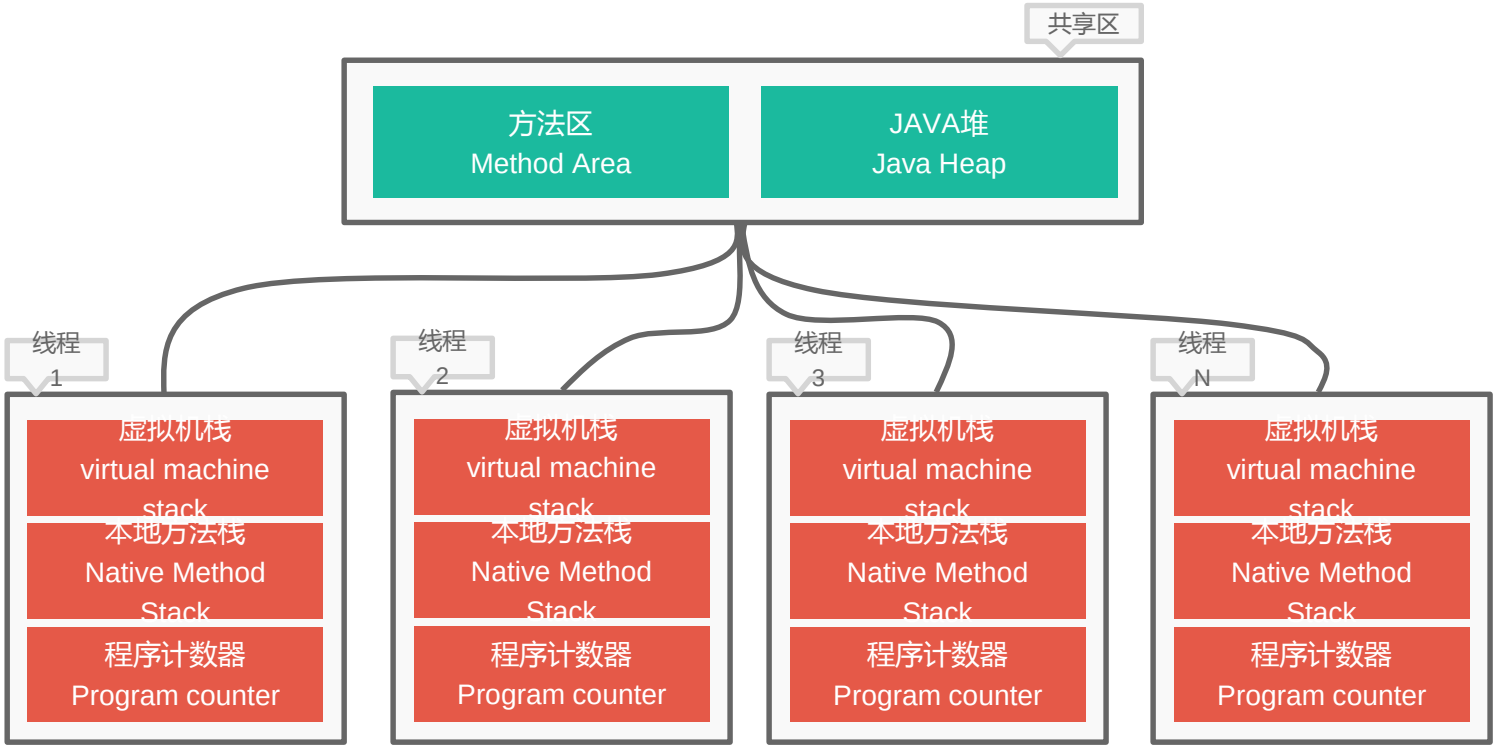
内存 与 线程

内存与线程的关系，主要是指JVM内存模型与线程之间的关系，它也是线程安全问题的主要诱因

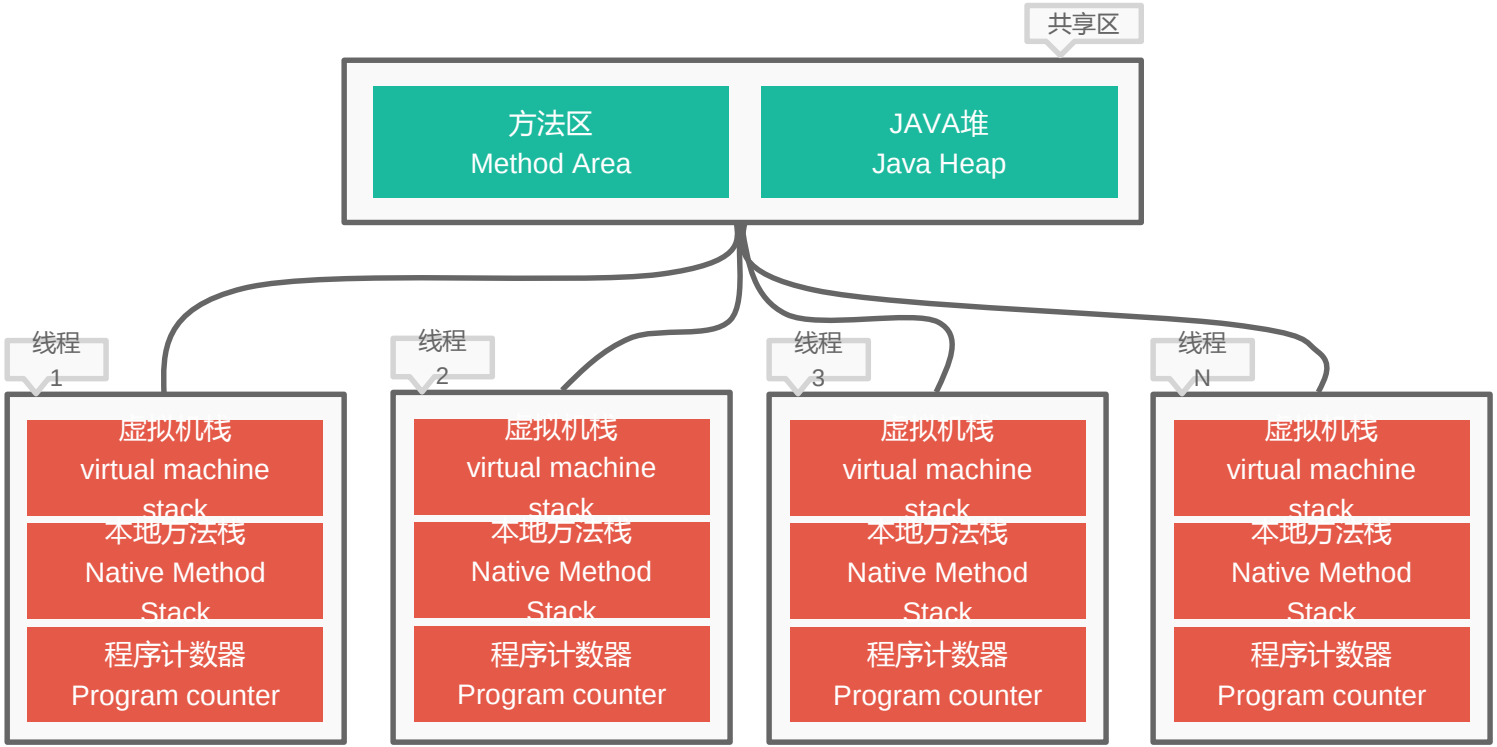
JVM内存模型



线程共享区与线程独享区



线程共享区与线程独享区



使用JDK工具观察线程

使用jdk工具查看线程堆栈、以及图形化跟踪

jcmd 查看线程堆栈

```
[root@dev-yd-jyh01 ~]# jcmd 6835 Thread.print
6835: #进程ID
2019-06-26 08:23:16 #打印时间
Full thread dump Java HotSpot(TM) 64-Bit Server VM (25.45-b02 mixed mode): #虚拟机版本描述

“Attach Listener“ #线程名称 #209 #线程号 daemon #守护进程 prio=9 #线程优先级 os_prio=0 #操作系统内优先级 tid=0x00007fa55402a800 #线程ID nid=0x547 #线程对应的本地线程ID waiting on condition [0x0000000000000000] #等待[某ID]的锁
   java.lang.Thread.State: RUNNABLE  #线程状态

"http-nio-8002-exec-128" #208 daemon prio=5 os_prio=0 tid=0x00007fa498079800 nid=0x1c60 waiting on condition [0x00007fa475ddc000]
   java.lang.Thread.State: WAITING (parking)
        at sun.misc.Unsafe.park(Native Method)
        - parking to wait for  <0x00000000dce0eb08> (a java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject)
        at java.util.concurrent.locks.LockSupport.park(LockSupport.java:175)
        at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await(AbstractQueuedSynchronizer.java:2039)
        at java.util.concurrent.LinkedBlockingQueue.take(LinkedBlockingQueue.java:442)
        at org.apache.tomcat.util.threads.TaskQueue.take(TaskQueue.java:103)
        at org.apache.tomcat.util.threads.TaskQueue.take(TaskQueue.java:31)
        at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1067)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1127)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
        at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)
        at java.lang.Thread.run(Thread.java:745)
```


jstack 查看线程堆栈

```
[root@dev-yd-jyh01 security]# jstack -h
Usage:
    jstack [-l] <pid>
        (to connect to running process)
    jstack -F [-m] [-l] <pid>
        (to connect to a hung process)
    jstack [-m] [-l] <executable> <core>
        (to connect to a core file)
    jstack [-m] [-l] [server_id@]<remote server IP or hostname>
        (to connect to a remote debug server)

Options:
    -F  to force a thread dump. Use when jstack <pid> does not respond (process is hung)
    -m  to print both java and native frames (mixed mode)
    -l  long listing. Prints additional information about locks
    -h or -help to print this help message
```

jstack可以查看或导出 Java 应用程序中线程堆栈信息。

参数说明：

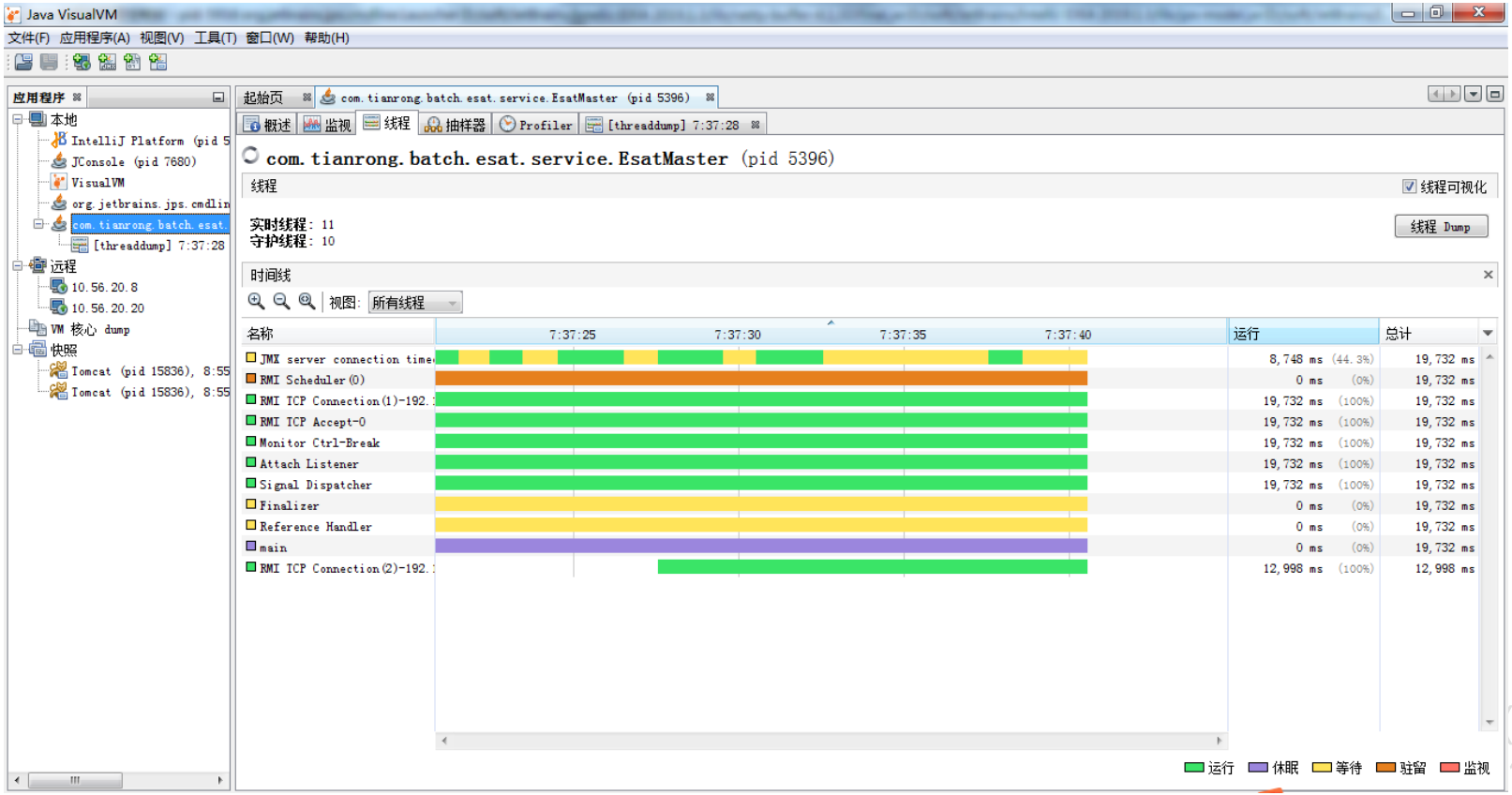
-l 长列表. 打印关于锁的附加信息,例如属于java.util.concurrent 的 ownable synchronizers列表.

-F 当’ jstack [-l] pid’没有相应的时候强制打印栈信息

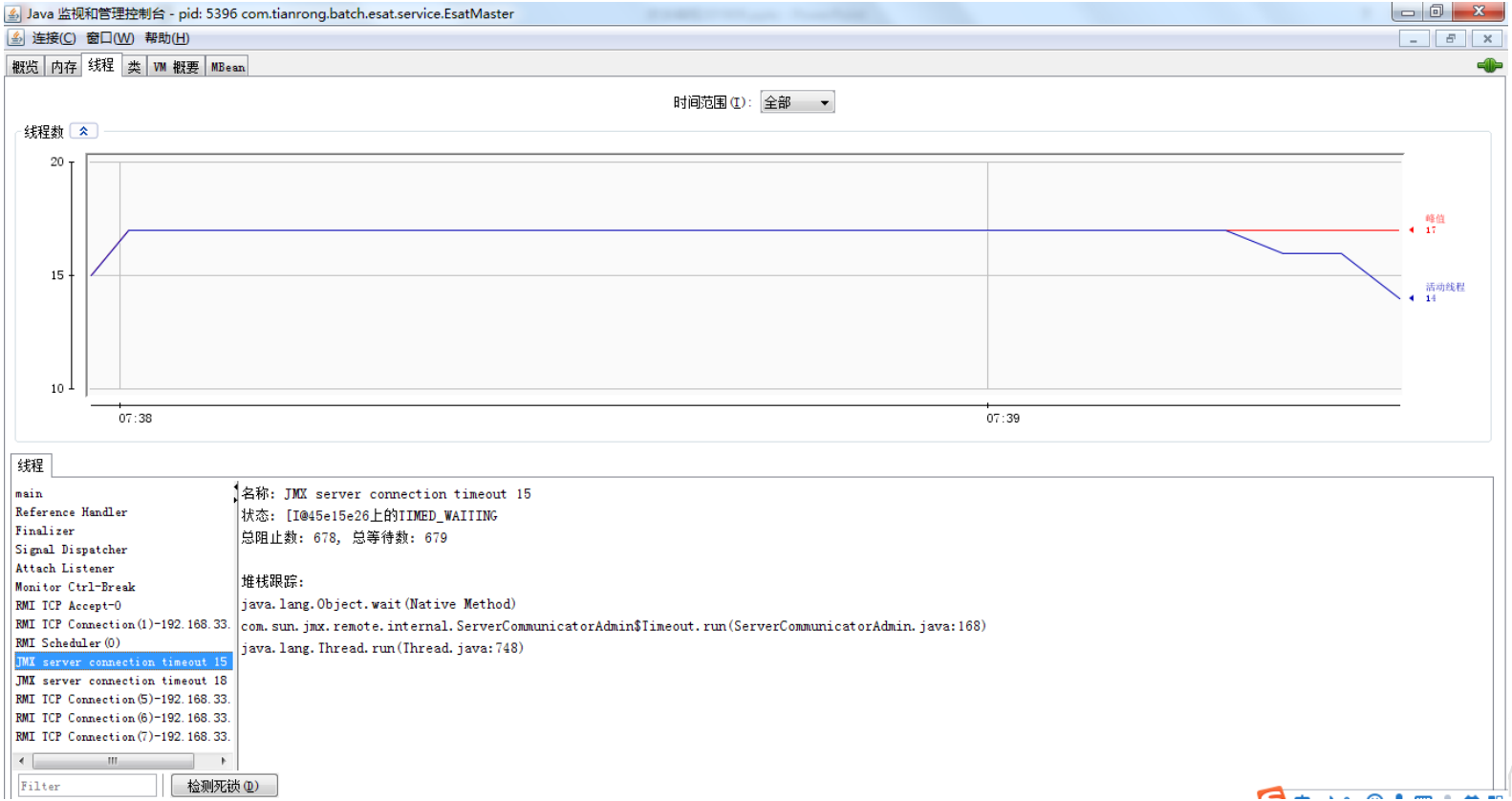
-m 打印java和native c/c++框架的所有栈信息.

-h | -help 打印帮助信息

jvisualvm 可视化查看线程信息



jconsole 可视化查看线程信息



线程创建的3种方法

线程创建有3种方法、继承Thread、实现
Runnable接口、实现Callable接口
优缺点各不相同

创建线程的3种方法

继承Thread类：Java不支持多继承，如果继承了Thread类，那么子类不能再继承其他类
实现Runnable接口：实现了Runnable接口还可以继承其他类。
实现Callable接口：前两种方式都没办法拿到任务的返回结果，但是Callable方式可以。

Demo:
com.mkevin.demo0.CreateThreadExtendsThread
com.mkevin.demo0.CreateThreadImplementsRunnable
com.mkevin.demo0.CreateThreadImplementsCallable

JOIN等待线程执行终止

join方法解析

join方法解析

使用场景：等待线程执行终止之后，继续执行
易混淆知识点：join方法为Thread类直接提供的方法，而wait和notify为Object类中的方法。
扩展知识点：可以使用CountDownLatch也可以达到相同效果

public final void join() throws InterruptedException
public final synchronized void join(long millis) throws InterruptedException
public final synchronized void join(long millis, int nanos) throws InterruptedException

Demo:
com.mkevin.demo1.JoinDemo0
com.mkevin.demo1.JoinDemo1

SLEEP方法解析

SLEEP方法解析

sleep方法解析

知识点：Thread类中的一个静态方法，暂时让出执行权，不参与CPU调度，但是不释放锁。时间到了就进入到就绪状态，一旦获取到CPU时间片，则继续执行。

public static native void sleep(long millis) throws InterruptedException
public static void sleep(long millis, int nanos) throws InterruptedException

Demo:
com.mkevin.demo1.SleepDemo0

YIELD方法解析

虚伪的方法

yield方法解析

知识点：Thread类中的静态native方法；让出剩余的时间片，本身进入就绪状态，CPU再次调度还可能调度到本线程。
易混淆知识点：sleep是在一段时间内进入阻塞状态，cpu不会调度它。而yield是让出执行权，本身还处于就绪状态，cpu还可能立即调度它。

```
public static native void yield();
```

Demo:
com.mkevin.demo1.YieldDemo0
com.mkevin.demo1.YieldDemo1

线程中断

Interrupt等相关方法的解析

线程中断

知识点：线程中断是线程间的一种协作模式，通过设置线程中断标志来实现，线程根据这个标志来自行处理。

public void interrupt()
public boolean isInterrupted()
public static boolean interrupted()

Demo:
com.mkevin.demo1.InterruptDemo0
com.mkevin.demo1.InterruptDemo1

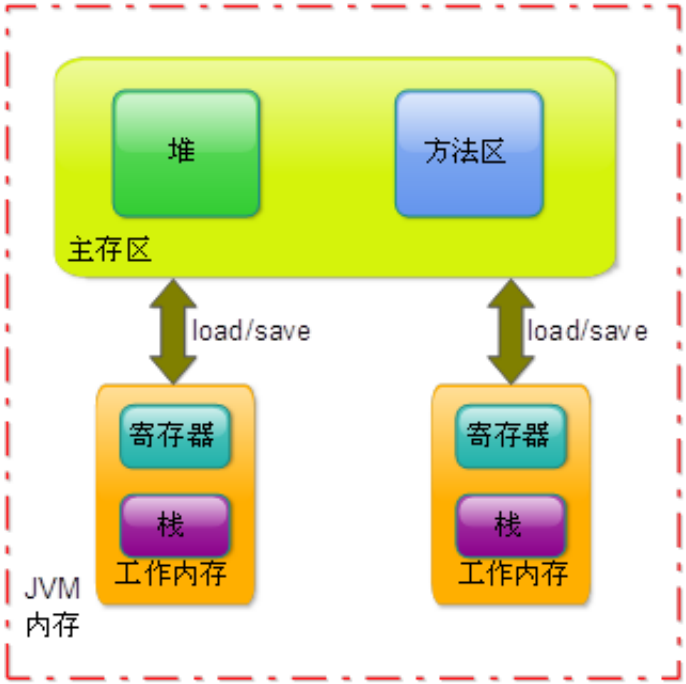
Demo：
com.mkevin.demo1.JoinDemo1
com.mkevin.demo1.WaitDemo1

为什么要学习并发编程

- **为什么要学习并发编程？**
- 互联网行业高速发展、服务能力与响应效率的要求增加。
- 大数据时代的到来、高性能的计算越发重要。
- 高薪职位的必备条件

- 通过本课程你能得到什么？
- 掌握线程安全的原理
- 掌握Concurrent并发编程包的底层原理和使用
- 掌握并发编程设计模式、高性能框架的使用

1、线程安全是怎么产生的?



- JVM内存模型
  - 可见性
  - 原子性
- 同时存款和取款
- 示例：DemoThread00

2、认识线程安全与Synchronized

- 1.线程安全的概念:当多个线程访问某一个类、对象或方法时，这个类、对象或方法都能表现出与单线程执行时一致的行为，那么这个类、对象或方法就是线程安全的。
- 2.线程安全问题都是由全局变量及静态变量引起的。
- 3.若每个线程中对全局变量、静态变量只有读操作，而无写操作，一般来说，这个全局变量是线程安全的；若有多个线程同时执行写操作，一般都需要考虑线程同步，否则的话就可能影响线程安全。
- 示例：DemoThread00

3、 Synchronized

- Synchronized的作用是加锁，所有的synchronized方法都会顺序执行，（这里只占用CPU的顺序）。
- Synchronized方法执行方式：
  - 首先尝试获得锁
  - 如果获得锁，则执行Synchronized的方法体内容。
  - 如果无法获得锁则等待，并且不断的尝试去获得锁，一旦锁被释放，则多个线程会同时去尝试获得所，造成锁竞争问题。
- 锁竞争问题，在高并发、线程数量高时会引起CPU占用居高不下，或者直接宕机。

4、对象锁和类锁

- 示例： DemoThread02

- 示例总结：
- Synchronized作用在非静态方法上代表的对象锁，一个对象一个锁，多个对象之间不会发生锁竞争。
- Synchronized作用在静态方法上则升级为类锁，所有对象共享一把锁，存在锁竞争。

5、同步和异步

- 同步：必须等待方法执行完毕，才能向下执行，共享资源访问的时候，为了保证线程安全，必须同步。
- 异步：不用等待其他方法执行完毕，即可立即执行，例如Ajax异步。

6、对象锁的同步和异步

- 示例： DemoThread03

- 示例总结：
- 对象锁只针对synchronized修饰的方法生效、 对象中的所有synchronized方法都会同步执行、而非synchronized方法异步执行
- 避免误区：类中有两个synchronized方法，两个线程分别调用两个方法，相互之间也需要竞争锁，因为两个方法从属于一个对象，而我们是在对象上加锁

7、脏读

- 由于同步和异步方法的执行个性，如果不从全局上进行并发设计很可能会引起数据的不一致，也就是所谓的脏读。
- 示例：DemoThread04

- 示例总结：
- 多个线程访问同一个资源，在一个线程修改数据的过程中，有另外的线程来读取数据，就会引起脏读的产生。
- 为了避免脏读我们一定要保证数据修改操作的原子性、并且对读取操作也要进行同步控制

8、Oracle如何防止脏读

- 一个数据库有5千万条数据
- 线程A在9:00向数据库发起查询操作，要通过全表扫描来查询最后一条数据，运行耗时需要10分钟。
- 线程B在9:05项数据库发起更新操作，对最后一条数据进行了更新。
- 问题1：线程A得到的数据，是修改前的数据，还是修改后的数据？
- 答：获取修改前的数据，因为Oracle数据修改时，都会先将原数据放入到undo空间中，undo空间可以放入多个版本的快照。
- 问题2：如果有多个线程对数据进行了修改，那么线程A是否还能够获取到修改前的数据？
- 答：可能得到也可能得不到，需要根据undo空间的大小来确定，多次修改如果覆盖了最初的undo数据，则会返回snapshot too old异常。

9、 synchronized锁重入

- 同一个线程得到了一个对象的锁之后，再次请求此对象时可以再次获得该对象的锁。
- 同一个对象内的多个synchronized方法可以锁重入
- 示例： DemoThread05

- 父子类可以锁重入
- 示例： DemoThread06

10、抛出异常释放锁

- 一个线程在获得锁之后执行操作，发生错误抛出异常，则自动释放锁
- 示例： DemoThread07

- 示例总结：
- 1、可以利用抛出异常，主动释放锁
- 2、程序异常时防止资源被死锁、无法释放
- 3、异常释放锁可能导致数据不一致

11、synchronized代码块

- 可以达到更细粒度的控制
- 当前对象锁
- 类锁
- 任意对象锁

- 示例：DemoThread08
- 示例：DemoThread11

- 示例总结：
- 同类型锁之间互斥,不同类型的锁之间互不干扰

12、不要在线程内修改对象锁的引用

- 不要在线程中修改对象锁的引用，引用被改变会导致锁失效。
- 示例： DemoThread09
- 在线程中修改了锁对象的属性,而不修改引用则不会引起锁失效、不会产生线程安全问题。
- 示例： DemoThread10
- 示例总结：
- 线程A修改了对象锁的引用，则线程B实际的到了新的对象锁，而不是锁被释放了,因此引发了线程安全问题。

13、并发与死锁

- 是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。

- 示例： DemoThread12

14、线程之间通讯

- 每个线程都是独立运行的个体，线程通讯能让多个线程之间协同工作
- Object类中的wait/notify方法可以 实现线程间通讯
- Wait/notify必须与synchronized一同使用
- Wait释放锁、notify不释放锁
- 示例：DemoThread17（while方式）
- 示例：DemoThread18（wait/notify方式）

15、notify与notifyAll的区别

- Notify只会通知一个wait中的线程，并把锁给他，不会产生锁竞争问题，但是该线程处理完毕之后必须再次notify或notifyAll，完成类似链式的操作。
- NotifyAll会通知所有wait中的线程，会产生锁竞争问题。

- 示例： DemoThread22

16、线程安全的阻塞队列

- 使用synchronized、wait、notify实现带阻塞的线程安全队列
- 示例： DemoThread20

WAIT方法解析补充

wait方法的一些补充说明

wait方法解析

易混淆知识点：wait和notify为Object类中的方法。而join方法为Thread类直接提供的方法。

public final void wait() throws InterruptedException
public final native void wait(long timeout) throws InterruptedException
public final void wait(long timeout, int nanos) throws InterruptedException

Demo:
com.mkevin.demo1.WaitDemo1
com.mkevin.demo1.WaitDemo0

守护线程与用户线程

Daemon线程与User线程

守护线程与用户线程

知识点

线程分类：daemon线程（守护线程）、user线程（用户线程）
易混淆知识点：main函数所在的线程就是一个用户线程
重要知识点1：最后一个user线程结束时，JVM会正常退出，不管是否有守护线程正在运行。反过来说：只要有一个用户线程还没结束，JVM进程就不会结束。
重要知识点2：父线程结束后，子线程还可以继续存活，子线程的生命周期不受父线程的影响

public final void setDaemon(boolean on)
public final boolean isDaemon()

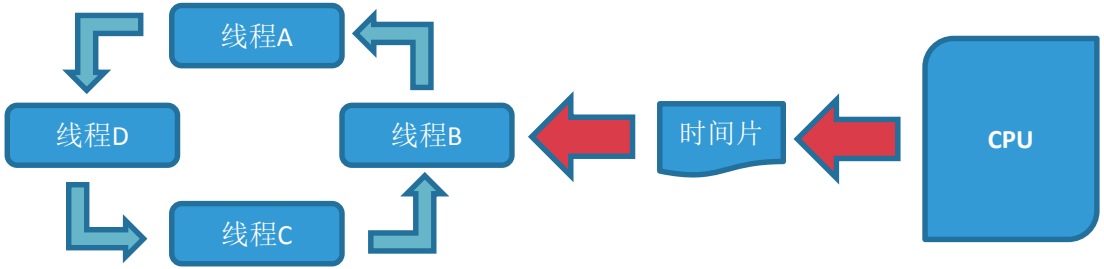
Demo

com.mkevin.demo1.DaemonAndUserThreadDemo0

线程上下文切换

什么是线程上下文切换

线程上下文切换



知识点

当前线程使用完时间片后就会进入就绪状态，让出CPU执行权给其他线程，此时就是从当前线程的上下文切换到了其他线程。

当发生上下文切换的时候需要保存执行现场，待下次执行时进行恢复。

所以频繁的、大量的上下文切换会造成一定资源开销

精品教程

JAVA并发编程系列

一次性搞定数据库事务

Memcached系列

一次性精通JVM

Disruptor高并发框架

SpringCloud微服务架构

RateLimiter访问限流

程序员转型项目经理

合作平台

扫描二维码学习更多教程

腾讯课堂 学习成就梦想

网易云课堂 我的职业课堂

51CTO.com 技术成就梦想