老齐的IT加油站

为热爱编程的你量身打造

老齐的IT加油站

Java多线程与并发编程

小白轻松学系列课程

齐毅

15年软件研发与教育经验

在多家一线软件公司任职高级职位

- 【用友软件】高级工程师
- 【财政部】金财系统项目经理
- 【京东】架构师
- 【宜信】高级研发经理
- 【尚学堂】金牌讲师、教学总监

2016年评选腾讯网"中国好老师"。

授课风格幽默风趣，内容干练实用，说话“节操尽碎”，童鞋们喜欢亲切的称呼我“老齐”、“齐老湿”。

课程目标


基本概念
Basic
原子对象
Atomic
多线程
Thread
并发编程
同步工具
Tools
线程池
Executor
并发容器
Collections

课程特点

1. 说人话，小白也能听得懂
2. 面试把关，老齐帮你划重点
3. 图文并茂，深入浅出

适合人群

- 求职应聘者
- 编程爱好者
- Java软件工程师
- 初级架构师

老齐的IT加油站

Java多线程与并发编程

小白轻松学系列课程

老齐的IT加油站

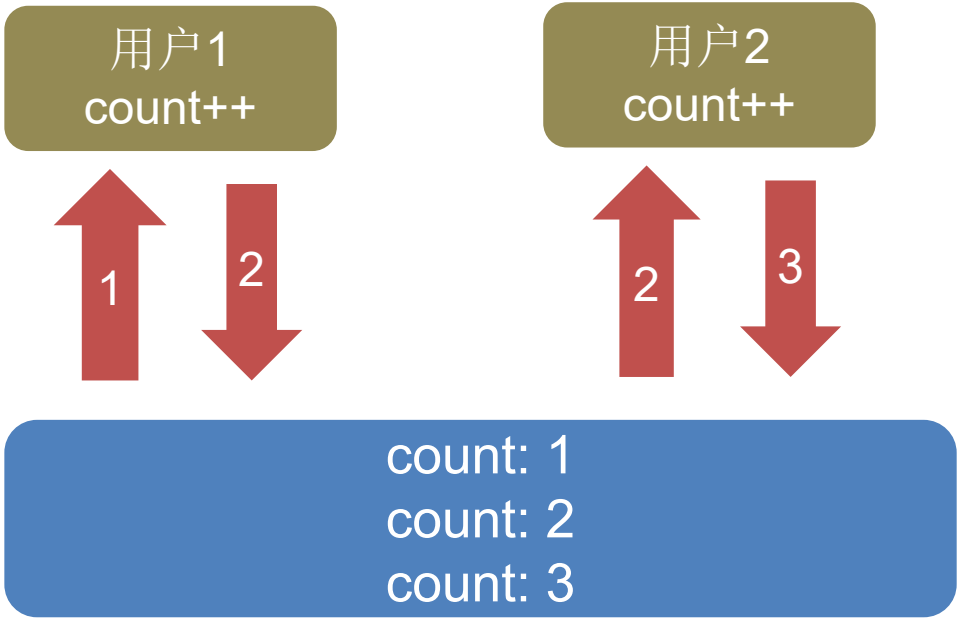
并发背后的故事

什么是并发

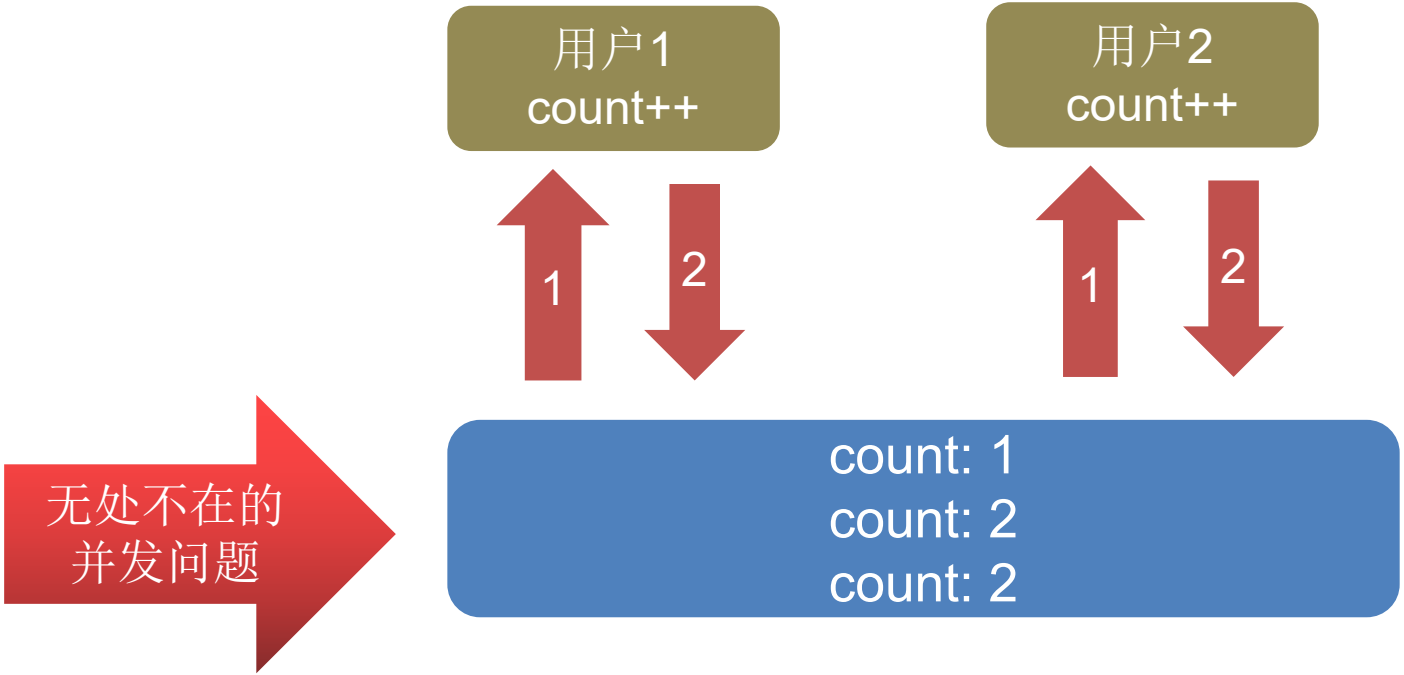
- 并发就是指程序同时处理多个任务的能力。
- 并发编程的根源在于对多任务情况下对访问资源的有效控制。

你的程序在并发环境下一定是正确的吗?

同时只有一个用户时运行过程



同时有两个(以上)个用户时的运行过程



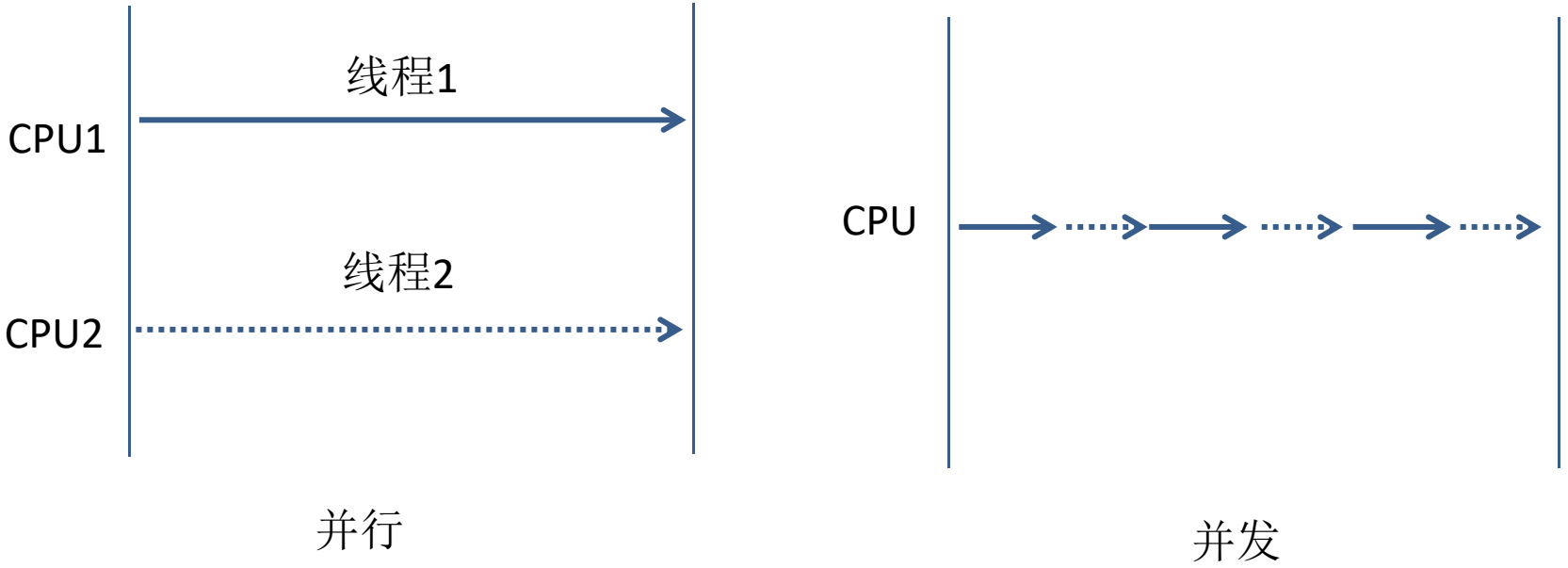
老齐的IT加油站

你必须知道的概念

程序、进程与线程

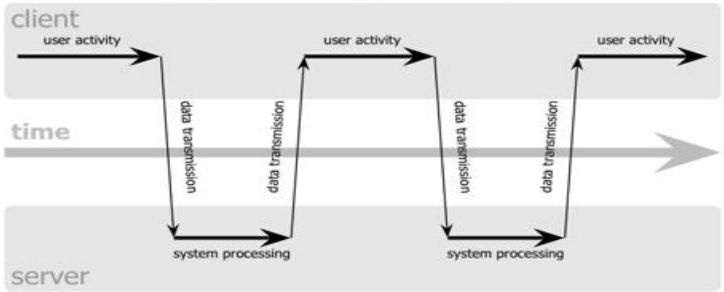
- 程序是静态的概念，windows下通常指exe文件。
- 进程是动态的概念，是程序在运行状态，进程说明程序在内存中的边界。
- 线程是进程内的一个”基本任务”,每个线程都有自己的功能，是CPU分配与调度的基本单位。

并发与并行

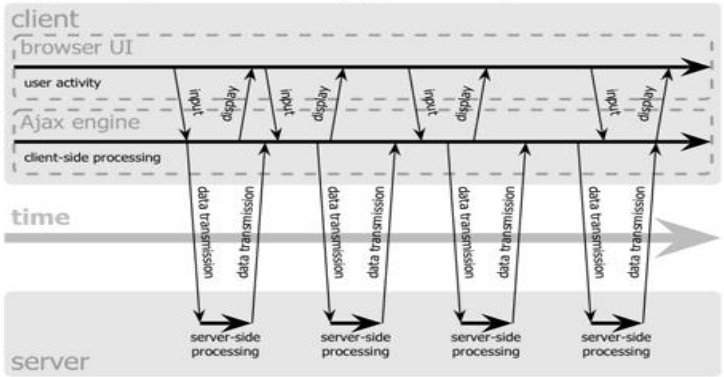


同步和异步

classic web application model (synchronous)



Ajax web application model (asynchronous)



临界区

- 临界区用来表示一种公共资源与共享数据，可以被多个线程使用。
- 同一时间只能有一个线程访问临界区（阻塞状态），其他资源必须等待。

死锁、饥饿、活锁

死锁

饥饿

活锁

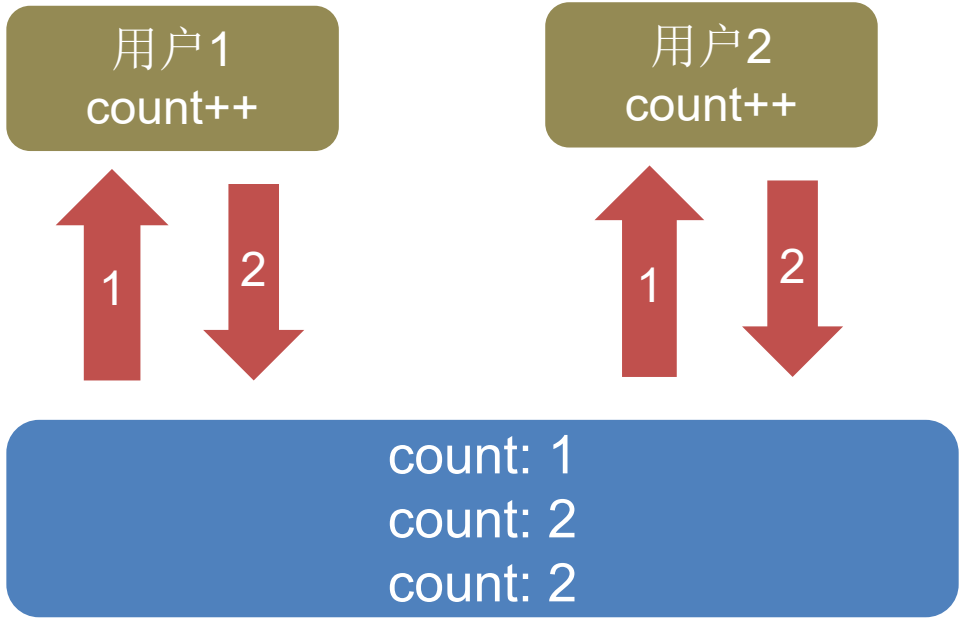
线程安全

- 在拥有共享数据的多条线程并行执行的程序中，线程安全的代码会通过同步机制保证各个线程都可以正常且正确的执行，不会出现数据污染等意外情况。

线程安全三大特性

- **原子性**
  - 即一个操作或者多个操作 要么全部执行并且执行的过程不会被任何因素打断，要么就都不执行。i = i + 1
- **可见性**
  - 当多个线程访问同一个变量时，一个线程修改了这个变量的值，其他线程能够立即看得到修改的值。
- **有序性**
  - 如果在本线程内观察，所有的操作都是有序的；如果在一个线程观察另一个线程，所有的操作都是无序的。

线程可见性不足的案例



可使用”锁”的解决线程的可见性问题

有序性的案例

```
int a = 10;    //语句1
int r = 2;    //语句2
a = a + 3;    //语句3
r = a*a;     //语句4
```

- 则因为重排序，他还可能执行顺序为2-1-3-4，1-3-2-4
- 但绝不可能2-1-4-3，因为这打破了依赖关系。
- 我们可以使用volatile关键字来防止重排序

老齐的IT加油站

创建线程-继承Thread

Java中创建线程三种方式

1. 继承Thread类创建线程
2. 实现Runnable接口创建线程
3. 使用Callable和Future创建线程

案例中Java共创建的几个线程?

进程

main
主线程

并发工具包-Concurrent

- JDK1.5以后为我们专门提供了一个并发工具包java.util.concurrent。
- java.util.concurrent 包含许多线程安全、测试良好、高性能的并发构建块。创建 concurrent 的目的就是要实现 Collection 框架对数据结构所执行的并发操作。通过提供一组可靠的、高性能并发构建块，开发人员可以提高并发类的线程安全、可伸缩性、性能、可读性和可靠性，

创建线程的三种方式对比

	继承Thread	实现Runnable	利用线程池
优点	编程简单 执行效率高	面向接口编程 执行效率高	容器管理线程 允许返回值与异常
缺点	单继承 无法对线程组有效控制	无法对线程组有效控制 没有返回值、异常	执行效率相对低 编程麻烦
使用场景	不推荐使用	简单的多线程程序	企业级应用 推荐使用

老齐的IT加油站

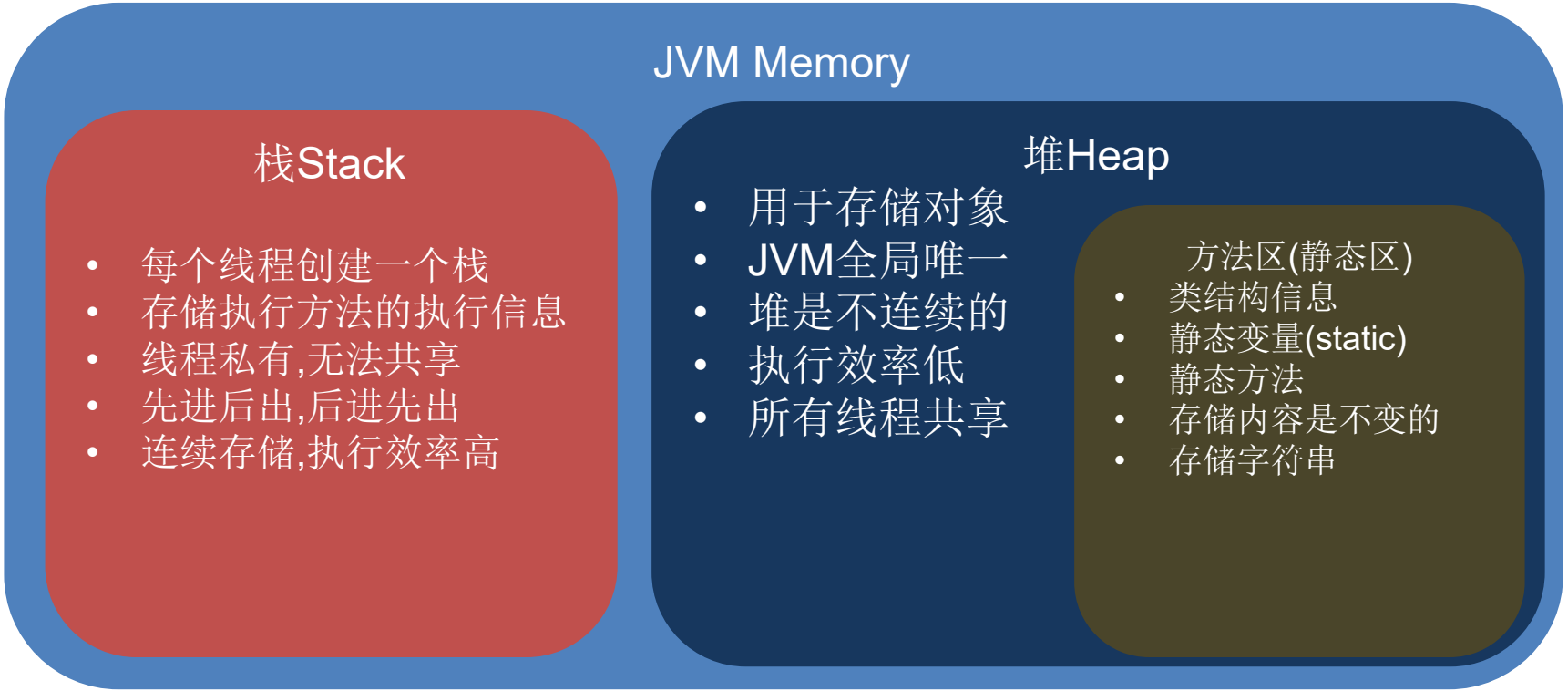
创建线程-实现Runnable

创建线程-利用线程池

老齐的IT加油站

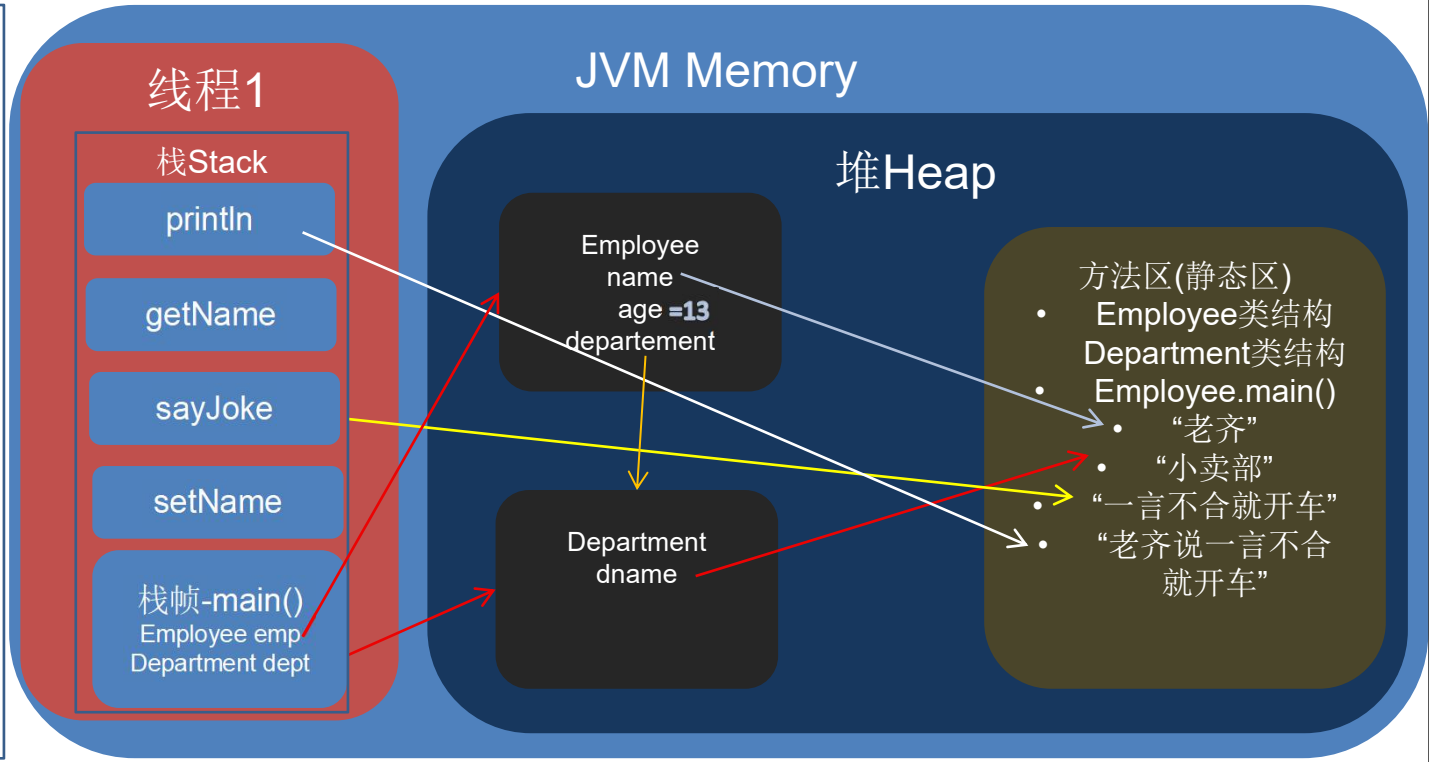
Java内存模型-JMM

Java内存模型
Java Memory Model



执行流程

1. 加载类(ClassLoader)
2. 启动main
3. emp = new Employee()
4. emp.setName("老齐");
5. emp.setAge(13);
6. dept = new Department()
7. dept.setDname("小卖部");
8. emp.setDepartment(dept);
9. emp.sayJoke("一言不合...")
10. 方法执行完成

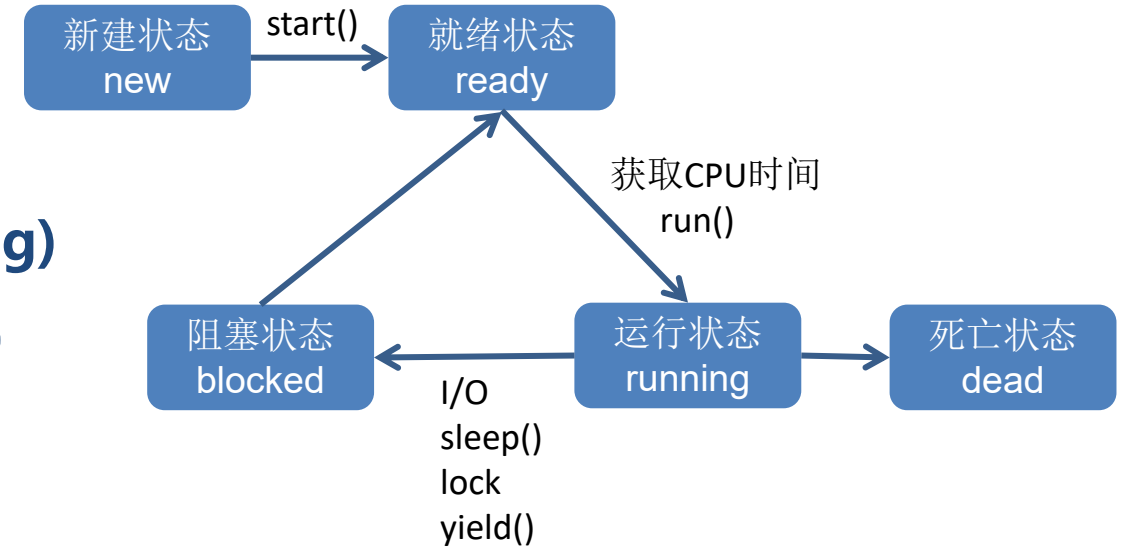


老齐的IT加油站

面试题：线程的五种状态

线程的五种状态

1. 新建（new）
2. 就绪(ready)
3. 运行中(running)
4. 阻塞(blocked)
5. 死亡(dead)



老齐的IT加油站

多线程的同步机制

现实中的”同步“机制

代码中的同步机制

- synchronized（同步锁）关键字的作用就是利用一个特定的对象设置一个锁lock（绣球），在多线程（游客）并发访问的时候，同时只允许一个线程（游客）可以获得这个锁，执行特定的代码（迎娶新娘）。执行后释放锁，继续由其他线程争抢。

Synchronize的使用场景

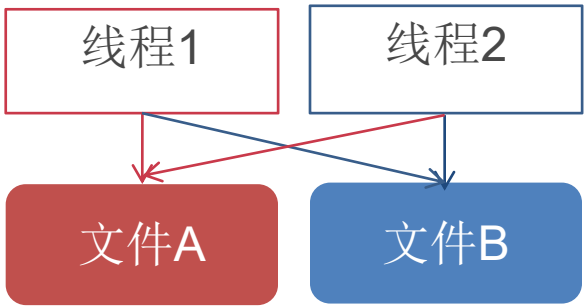
- Synchronize可以使用在以下三种场景，对应不同锁对象：
  - synchronized代码块 - 任意对象即可
  - synchronized方法 - this当前对象
  - synchronized静态方法 - 该类的字节码对象

老齐的IT加油站

死锁的产生

死锁产生的原因

- 死锁是在多线程情况下最严重的问题，在多线程对公共资源（文件、数据）等进行操作时，彼此不释放自己的资源，而去试图操作其他线程的资源，而形成交叉引用，就会产生死锁。
- 解决死锁最根本的建议是：
  - 尽量减少公共资源的引用，用完马上释放
  - 用完马上释放公共资源
  - 减少synchronized使用，采用“副本”方式替代



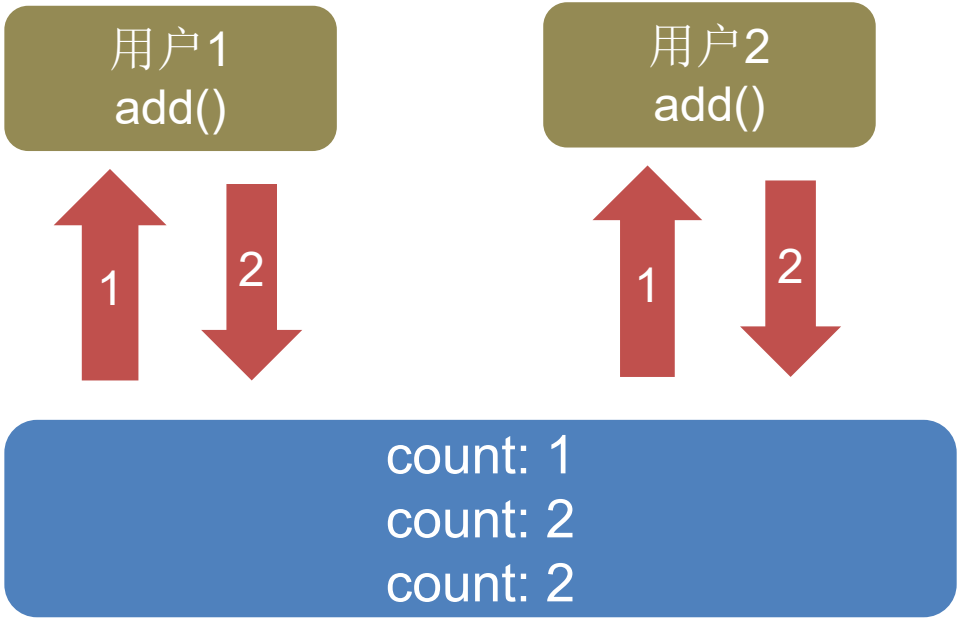
老齐的IT加油站

重新认识线程安全

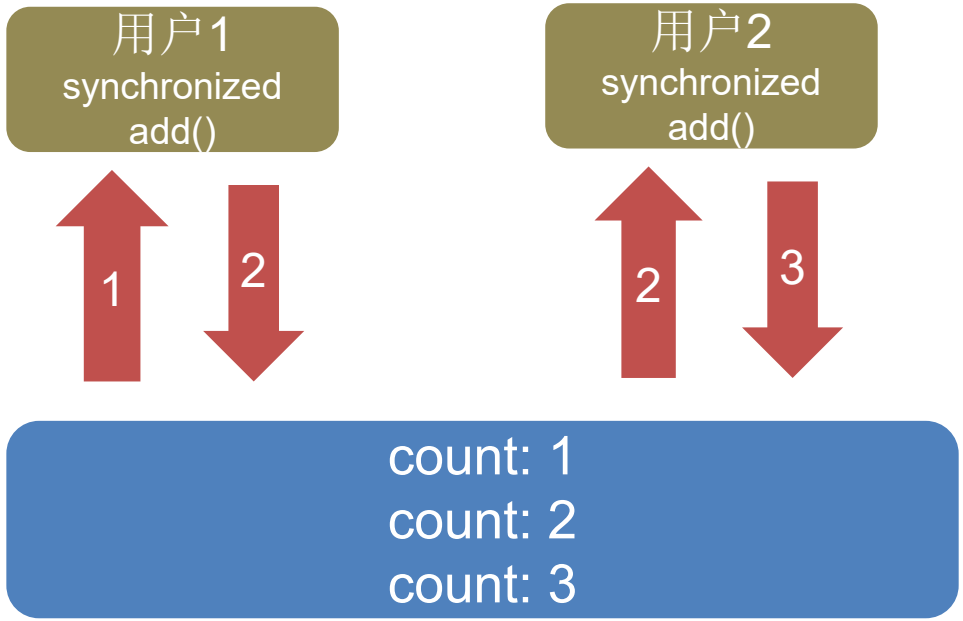
线程安全

- 在拥有共享数据的多条线程并行执行的程序中，线程安全的代码会通过同步机制保证各个线程都可以正常且正确的执行，不会出现数据污染等意外情况。

线程不安全的程序



通过synchronized使线程变得安全



线程安全与不安全的区别

◆ 线程安全

- 优点：可靠
- 缺点：执行速度慢
- 使用建议：需要线程共享时使用

◆ 线程不安全

- 优点：速度快
- 缺点：可能与预期不符
- 使用建议：在线程内部使用，无需线程间共享

请写出线程(不）安全的类

- Vector是线程安全的,ArrayList、LinkedList是线程不安全的
- Properties是线程安全的，HashSet、TreeSet是不安全的
- StringBuffer是线程安全的,StringBuilder是线程不安全的
- HashTable是线程安全的,HashMap是线程不安全的

老齐的IT加油站

JDK并发工具包-线程池

java.util.concurrent

- 并发是伴随着多核处理器的诞生而产生的，为了充分利用硬件资源，诞生了多线程技术。但是多线程又存在资源竞争的问题，引发了同步和互斥的问题，JDK 1.5推出的java.util.concurrent(并发工具包）来解决这些问题。

什么是线程池

new Thread的弊端

- new Thread()新建对象，性能差
- 线程缺乏统一管理，可能无限制的新建线程，相互竞争，严重时会占用过多系统资源导致死机或OOM

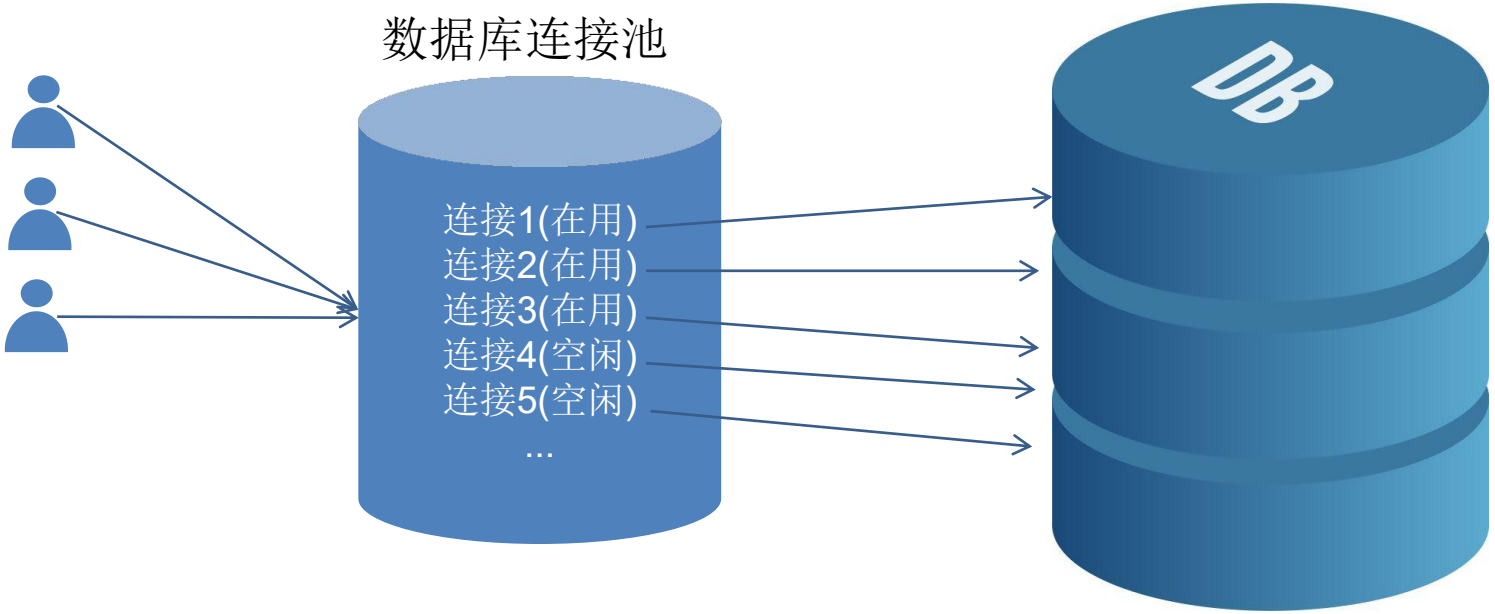
ThreadPool - 线程池

- 重用存在的线程，减少对象对象、消亡的开销
- 线程总数可控，提高资源的利用率
- 避免过多资源竞争，避免阻塞
- 提供额外功能，定时执行、定期执行、监控等。

线程池的种类

- 在java.util.concurrent中，提供了工具类Executors（调度器）对象来创建线程池，可创建的线程池有四种：
  1. CachedThreadPool - 可缓存线程池
  2. FixedThreadPool - 定长线程池
  3. SingleThreadExecutor - 单线程池
  4. ScheduledThreadPool - 调度线程池

线程池的经典应用



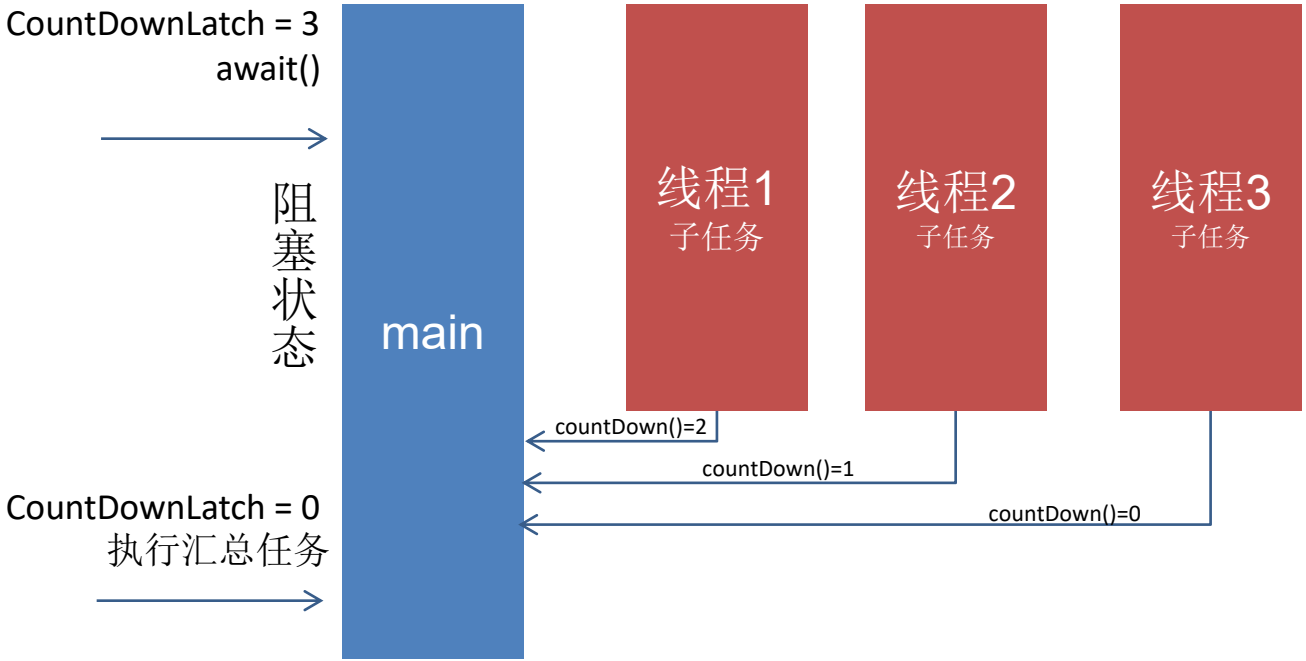
老齐的IT加油站

CountDownLatch - 倒计时锁

CountDownLatch - 倒计时锁

- CountDownLatch倒计时锁特别适合”总-分任务”，例如多线程计算后的数据汇总
- CountDownLatch类位于java.util.concurrent (J.U.C) 包下，利用它可以实现类似计数器的功能。比如有一个任务A，它要等待其他3个任务执行完毕之后才能执行，此时就可以利用CountDownLatch来实现这种功能了。

执行原理


CountDownLatch = 3
await()
阻塞状态
CountDownLatch = 0
执行汇总任务
main
线程1
子任务
线程2
子任务
线程3
子任务
countDown()=2
countDown()=1
countDown()=0

老齐的IT加油站

JUC之Semaphore信号量

Semephore信号量的作用

- Semaphore信号量经常用于限制获取某种资源的线程数量。下面举个例子，比如说操场上有5个跑道，一个跑道一次只能有一个学生在上面跑步，一旦所有跑道在使用，那么后面的学生就需要等待，直到有一个学生不跑了

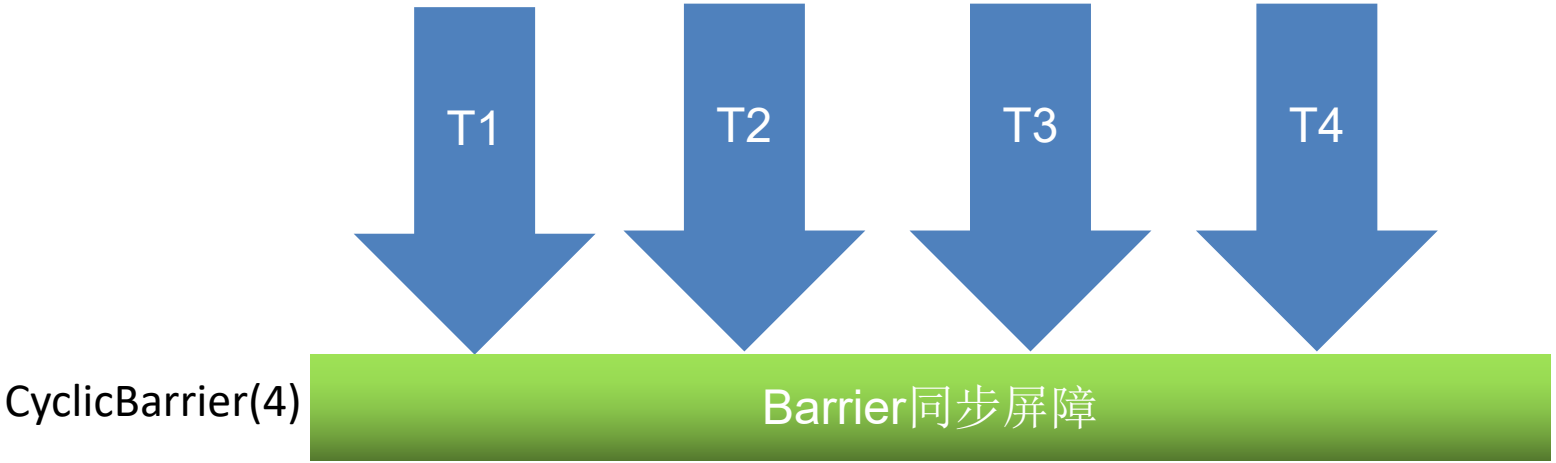
老齐的IT加油站

JUC之CyclicBarrier循环屏障

CyclicBarrier循环屏障

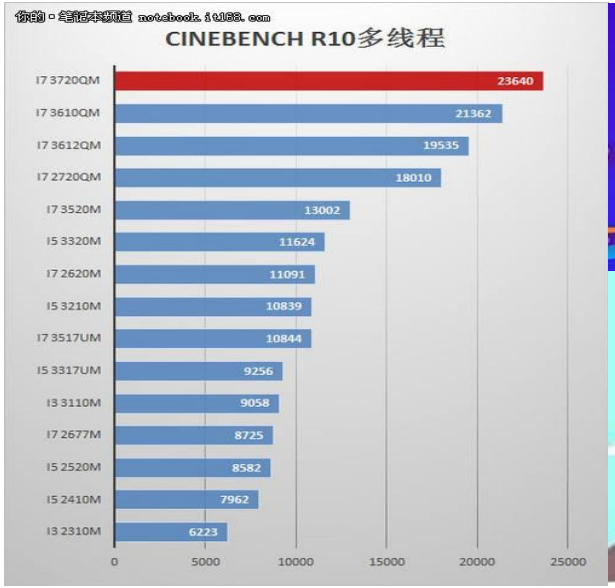
- CyclicBarrier是一个同步工具类，它允许一组线程互相等待，直到到达某个公共屏障点。与CountDownLatch不同的是该barrier在释放等待线程后可以重用，所以称它为循环（Cyclic）的屏障（Barrier）。

CyclicBarrier用于让线程必须运行



CyclicBarrier的应用场景

- CyclicBarrier适用于多线程必须同时开始的场景



老齐的IT加油站

JUC之ReentrantLock重入锁

什么是重入锁

- 重入锁是指任意线程在获取到锁之后,再次获取该锁而不会被该锁所阻塞
- ReentrantLock设计的目标是用来替代synchronized关键字

ReentrantLock与synchronized的区别

特征	synchronized（推荐）	reentrantLock
底层原理	JVM实现	JDK实现
性能区别	低->高（JDK5+)	高
锁的释放	自动释放(编译器保证)	手动释放(finally保证)
编码程度	简单	复杂
锁的粒度	读写不区分	读锁、写锁
高级功能	无	公平锁、非公平锁唤醒 Condition分组唤醒 中断等待锁

老齐的IT加油站

JUC之Condition等待与唤醒

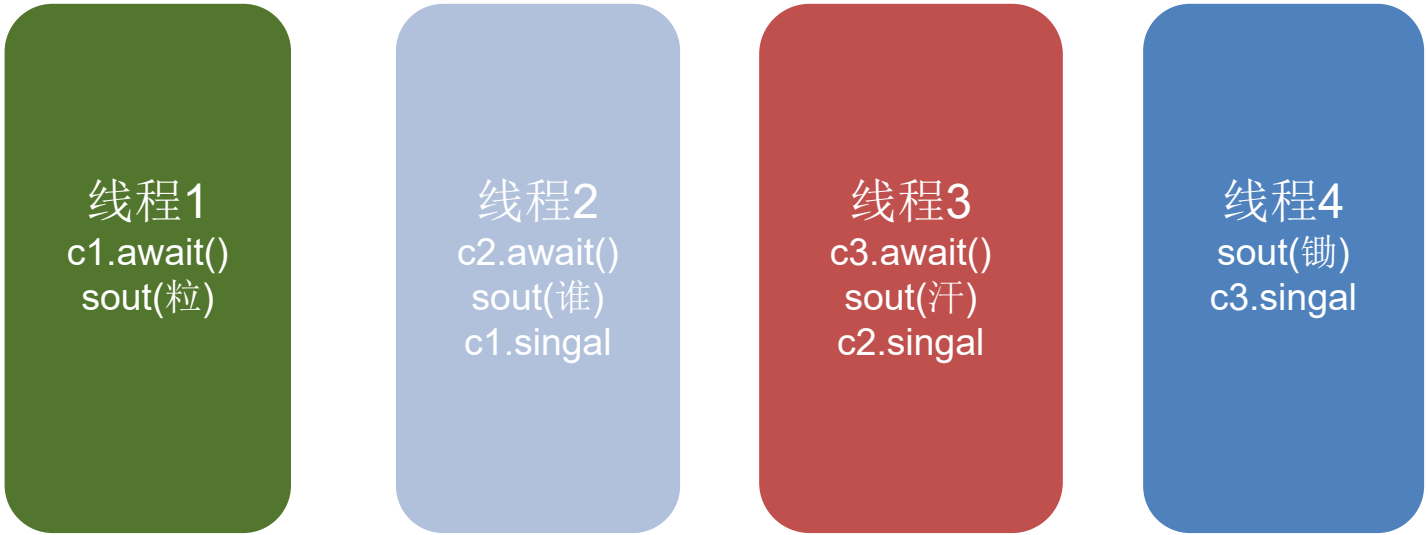
Condition条件唤醒

- 我们在并行程序中，避免不了某些线程要预先规定好的顺序执行，例如：先新增再修改,先买后卖，先进后出……，对于这类场景，使用JUC的Condition对象再合适不过了。
- JUC中提供了Condition对象，用于让指定线程等待与唤醒，按预期顺序执行。它必须和ReentrantLock重入锁配合使用。
- Condition用于替代wait()/notify()方法
  - notify只能随机唤醒等待的线程，而Condition可以唤醒指定的线程，这有利于更好的控制并发程序。

Condition核心方法

- await() - 阻塞当前线程，直到singal唤醒
- signal() - 唤醒被await的线程，从中断处继续执行
- signalAll() - 唤醒所有被await()阻塞的线程

执行过程


线程1
c1.await()
sout(粒)
线程2
c2.await()
sout(谁)
c1.singal
线程3
c3.await()
sout(汗)
c2.singal
线程4
sout(锄)
c3.singal

老齐的IT加油站

JUC之Callable&Future

Callable&Future

- Callable和Runnable一样代表着任务，区别在于Callable有返回值并且可以抛出异常。
- Future 是一个接口。它用于表示异步计算的结果。提供了检查计算是否完成的方法，以等待计算的完成，并获取计算的结果。

老齐的IT加油站

JUC之并发容器

请写出线程安全的类

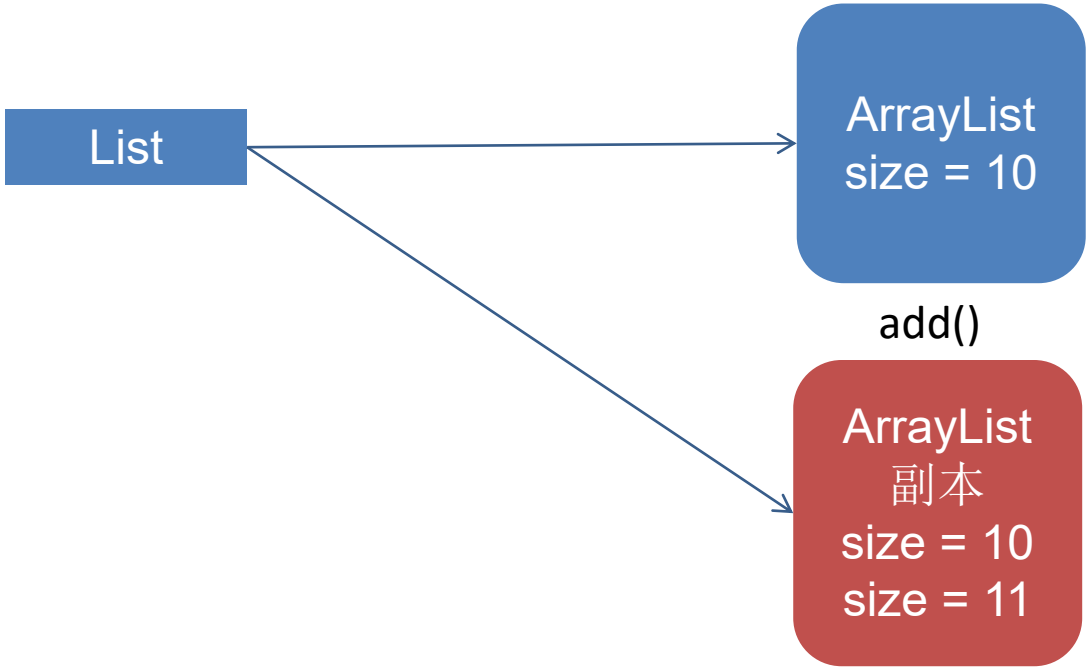
- Vector是线程安全的,ArrayList、LinkedList是线程不安全的
- Properties是线程安全的，HashSet、TreeSet是不安全的
- StringBuffer是线程安全的,StringBuilder是线程不安全的
- HashTable是线程安全的,HashMap是线程不安全的

线程安全 - 并发容器

- ArrayList -> CopyOnWriteArrayList - 写复制列表
- HashSet -> CopyOnWriteArraySet - 写复制集合
- HashMap -> ConcurrentHashMap - 分段锁映射

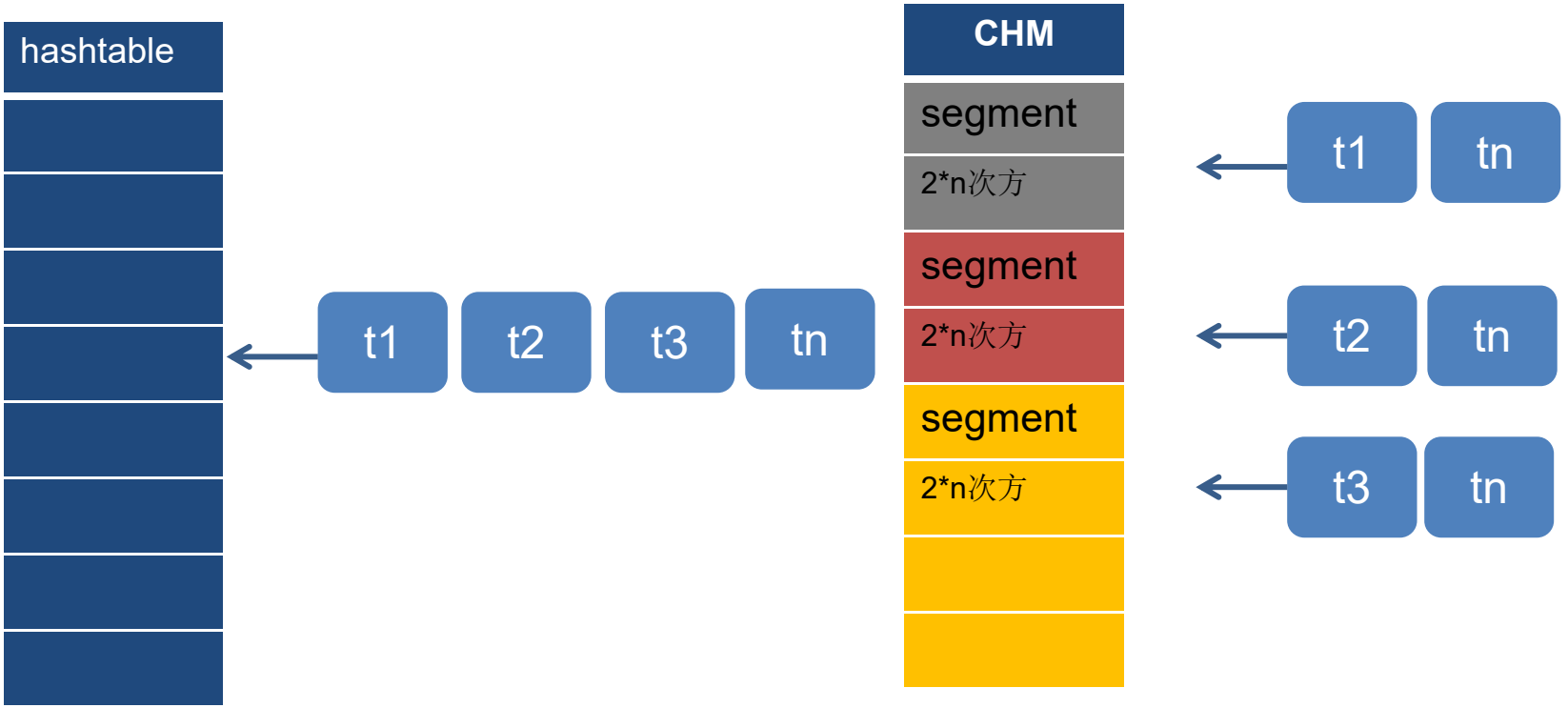
CopyOnWriteArrayList并发原理

- CopyOnWriteArrayList通过“副本”解决并发问题



ConcurrentHashMap

- ConcurrentHashMap 采用”分段锁“的方式



老齐的IT加油站

JUC之Atomic包与CAS算法

回顾原子性

- 原子性：是指一个操作或多个操作要么全部执行，且执行的过程不会被任何因素打断，要么就都不执行。

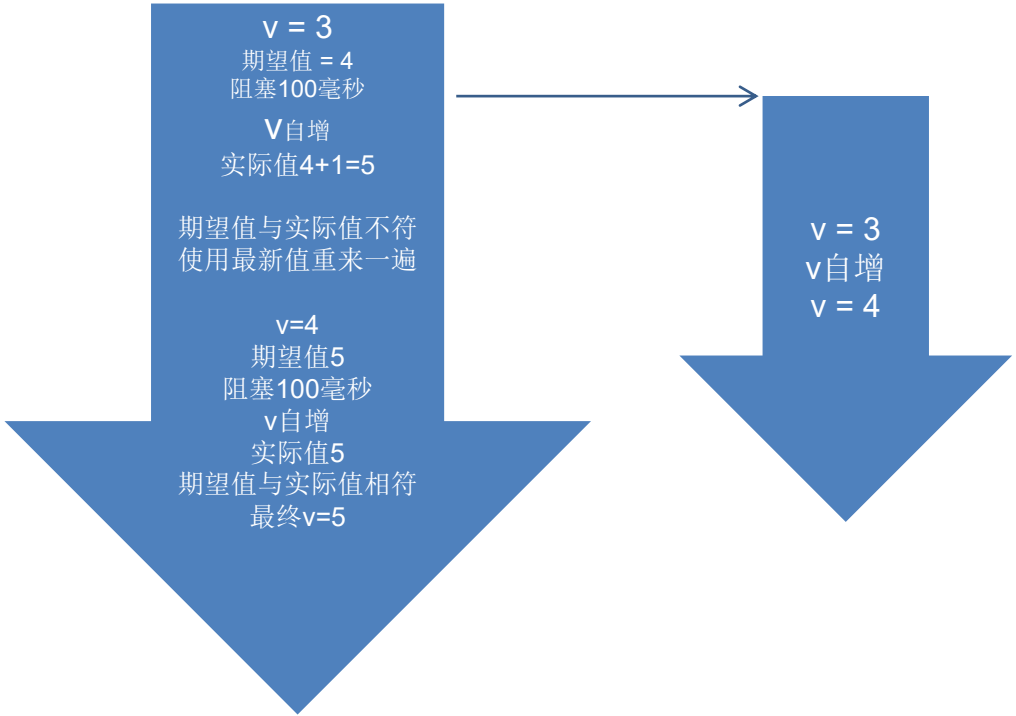
Atomic包

- Atomic包是java.util.concurrent下的另一个专门为线程安全设计的Java包，包含多个原子操作类。
- Atomic常用类
  - AtomicInteger
  - AtomicIntegerArray
  - AtomicBoolean
  - AtomicLong
  - AtomicLongArray

白话CAS算法

- 锁是用来做并发最简单的方式，当然其代价也是最高的。独占锁是一种悲观锁，synchronized就是一种独占锁，它假设最坏的情况，并且只有在确保其它线程不会造成干扰的情况下执行，会导致其它所有需要锁的线程挂起，等待持有锁的线程释放锁。
- 所谓乐观锁就是，每次不加锁而是假设没有冲突而去完成某项操作，如果因为冲突失败就重试，直到成功为止。其中CAS（比较与交换，Compare And Swap）是一种有名的无锁算法。

白话C(Compare)A(And)Swap算法

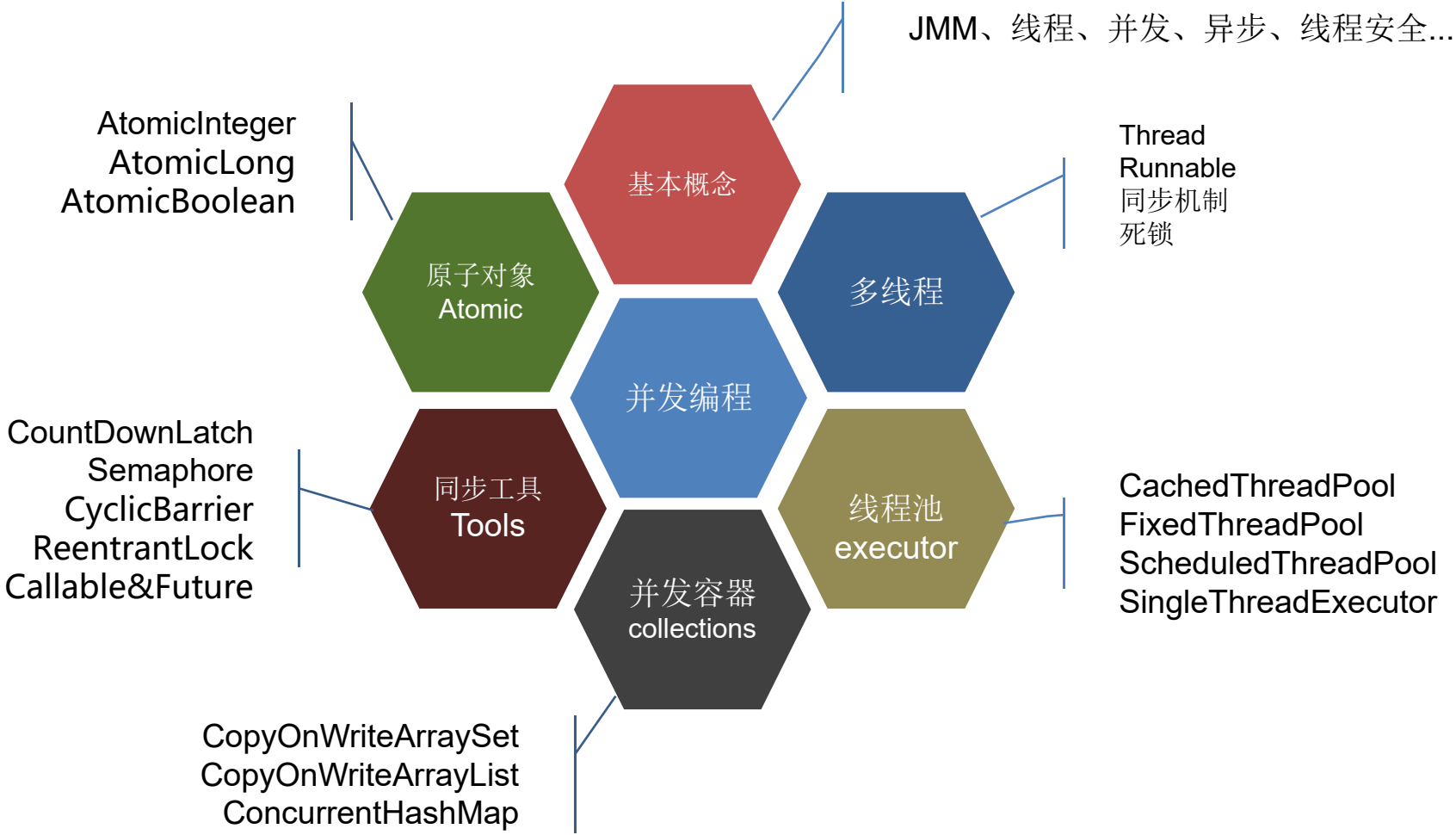


Atomic的应用场景

- 虽然基于CAS的线程安全机制很好很高效，但要说的是，并非所有线程安全都可以用这样的方法来实现，这只适合一些粒度比较小型,如计数器这样的需求用起来才有效，否则也不会有锁的存在了。

老齐的IT加油站

课程总结


JMM、线程、并发、异步、线程安全...
AtomicInteger
AtomicLong
AtomicBoolean
基本概念
Thread
Runnable
同步机制
死锁
原子对象
Atomic
多线程
并发编程
CountDownLatch
Semaphore
CyclicBarrier
ReentrantLock
Callable&Future
同步工具
Tools
线程池
executor
CachedThreadPool
FixedThreadPool
ScheduledThreadPool
SingleThreadExecutor
并发容器
collections
CopyOnWriteArraySet
CopyOnWriteArrayList
ConcurrentHashMap