主讲老师：Fox

> 有道云笔记链接：https://note.youdao.com/s/7u1Hbkkp

思考： 如何设计一把独占锁?

1. 管程 — Java同步的设计思想

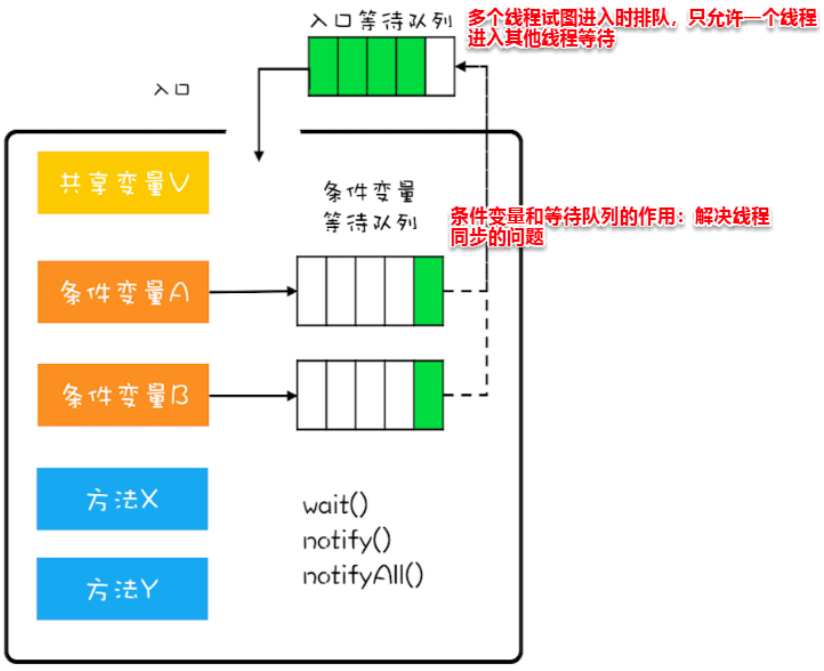
管程：指的是管理共享变量以及对共享变量的操作过程，让他们支持并发。

互斥：同一时刻只允许一个线程访问共享资源；

同步：线程之间如何通信、协作。

MESA模型

在管程的发展史上，先后出现过三种不同的管程模型，分别是Hasen模型、Hoare模型和MESA模型。现在正在广泛使用的是**MESA模型**。



MESA 管程模型

管程中引入了条件变量的概念，而且每个条件变量都对应有一个等待队列。条件变量和等待队列的作用是解决线程之间的同步问题。

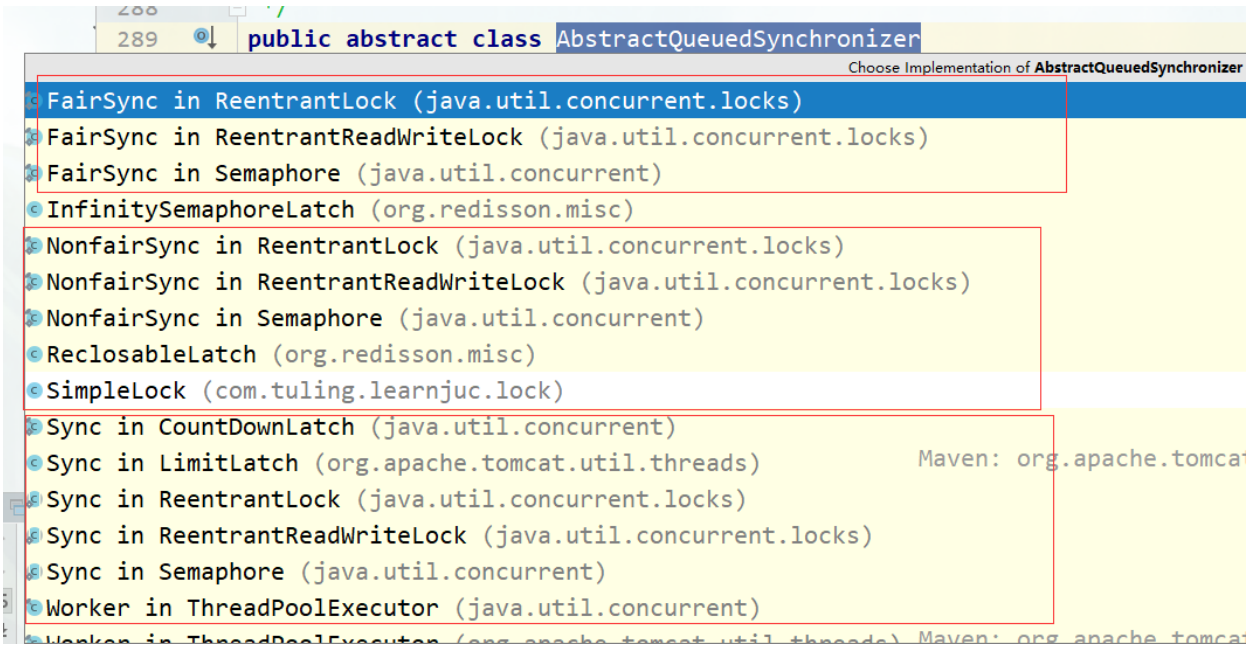
2. AQS原理分析

2.1 什么是AQS

java.util.concurrent包中的大多数同步器实现都是围绕着共同的基础行为，比如等待队列、条件队列、独占获取、共享获取等，而这些行为的抽象就是基于AbstractQueuedSynchronizer（简称AQS）实现的，AQS是一个抽象同步框架，可以用来实现一个依赖状态的同步器。

JDK中提供的大多数的同步器如Lock, Latch, Barrier等，都是基于AQS框架来实现的

- 一般是通过一个内部类Sync继承 AQS
- 将同步器所有调用都映射到Sync对应的方法



AQS具备的特性：

- 阻塞等待队列
- 共享/独占
- 公平/非公平
- 可重入
- 允许中断

2.2 AQS核心结构

AQS内部维护属性volatile int state

- state表示资源的可用状态

State三种访问方式：

- getState()
- setState()
- compareAndSetState()

定义了两种资源访问方式：

- Exclusive-独占，只有一个线程能执行，如ReentrantLock
- Share-共享，多个线程可以同时执行，如Semaphore/CountDownLatch

AQS实现时主要实现以下几种方法：

- isHeldExclusively()：该线程是否正在独占资源。只有用到condition才需要去实现它。
- tryAcquire(int)：独占方式。尝试获取资源，成功则返回true，失败则返回false。
- tryRelease(int)：独占方式。尝试释放资源，成功则返回true，失败则返回false。
- tryAcquireShared(int)：共享方式。尝试获取资源。负数表示失败；0表示成功，但没有剩余可用资源；正数表示成功，且有剩余资源。
- tryReleaseShared(int)：共享方式。尝试释放资源，如果释放后允许唤醒后续等待结点返回true，否则返回false。

2.3 AQS定义两种队列

- 同步等待队列： 主要用于维护获取锁失败时入队的线程。
- 条件等待队列： 调用await()的时候会释放锁，然后线程会加入到条件队列，调用signal()唤醒的时候会把条件队列中的线程节点移动到同步队列中，等待再次获得锁。

AQS 定义了5个队列中节点状态：

1. 值为0，初始化状态，表示当前节点在sync队列中，等待着获取锁。
2. CANCELLED，值为1，表示当前的线程被取消；
3. SIGNAL，值为-1，表示当前节点的后继节点包含的线程需要运行，也就是unpark；
4. CONDITION，值为-2，表示当前节点在等待condition，也就是在condition队列中；
5. PROPAGATE，值为-3，表示当前场景下后续的acquireShared能够得以执行；

同步等待队列

AQS当中的同步等待队列也称CLH队列，CLH队列是Craig、Landin、Hagersten三人发明的一种基于双向链表数据结构的队列，是FIFO先进先出线程等待队列，Java中的CLH队列是原CLH队列的一

个变种,线程由原自旋机制改为阻塞机制。

AQS 依赖CLH同步队列来完成同步状态的管理：

- 当前线程如果获取同步状态失败时，AQS则会将当前线程已经等待状态等信息构造成一个节点（Node）并将其加入到CLH同步队列，同时会阻塞当前线程
- 当同步状态释放时，会把首节点唤醒（公平锁），使其再次尝试获取同步状态。
- 通过signal或signalAll将条件队列中的节点转移到同步队列。（由条件队列转化为同步队列）

条件等待队列

AQS中条件队列是使用单向列表保存的，用nextWaiter来连接:

- 调用await方法阻塞线程；
- 当前线程存在于同步队列的头结点，调用await方法进行阻塞（从同步队列转化到条件队列）

3. ReentrantLock源码分析

ReentrantLock是一种基于AQS框架的应用实现，是JDK中的一种线程并发访问的同步手段，它的功能类似于synchronized是一种互斥锁，可以保证线程安全。

ReentrantLock基本使用方式

```
public class ReentrantLockTest {
    private final ReentrantLock lock = new ReentrantLock();
    // ...

    public void doSomething() {
        lock.lock();  // block until condition holds
        try {
            // ... method body
        } finally {
            lock.unlock();
        }
    }
}
```

源码阅读过程中要关注的问题

1.公平和非公平锁，可重入锁是如何实现的

2.设计的精髓：并发场景下入队和出队操作是如何设计的

- 线程竞争锁失败入队阻塞逻辑实现
- 释放锁的线程唤醒阻塞线程出队竞争锁的逻辑实现

结合课上源码分析流程图理解ReentrantLock的实现

https://www.processon.com/view/link/640de02f5af4953528df4883