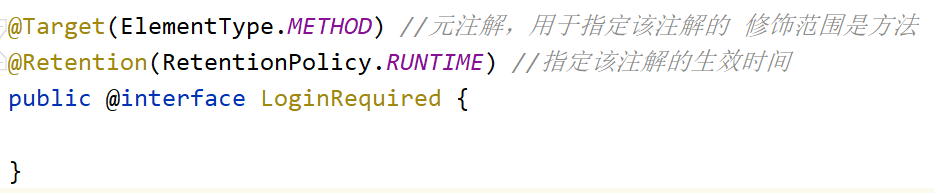
## 注解是怎么实现的？

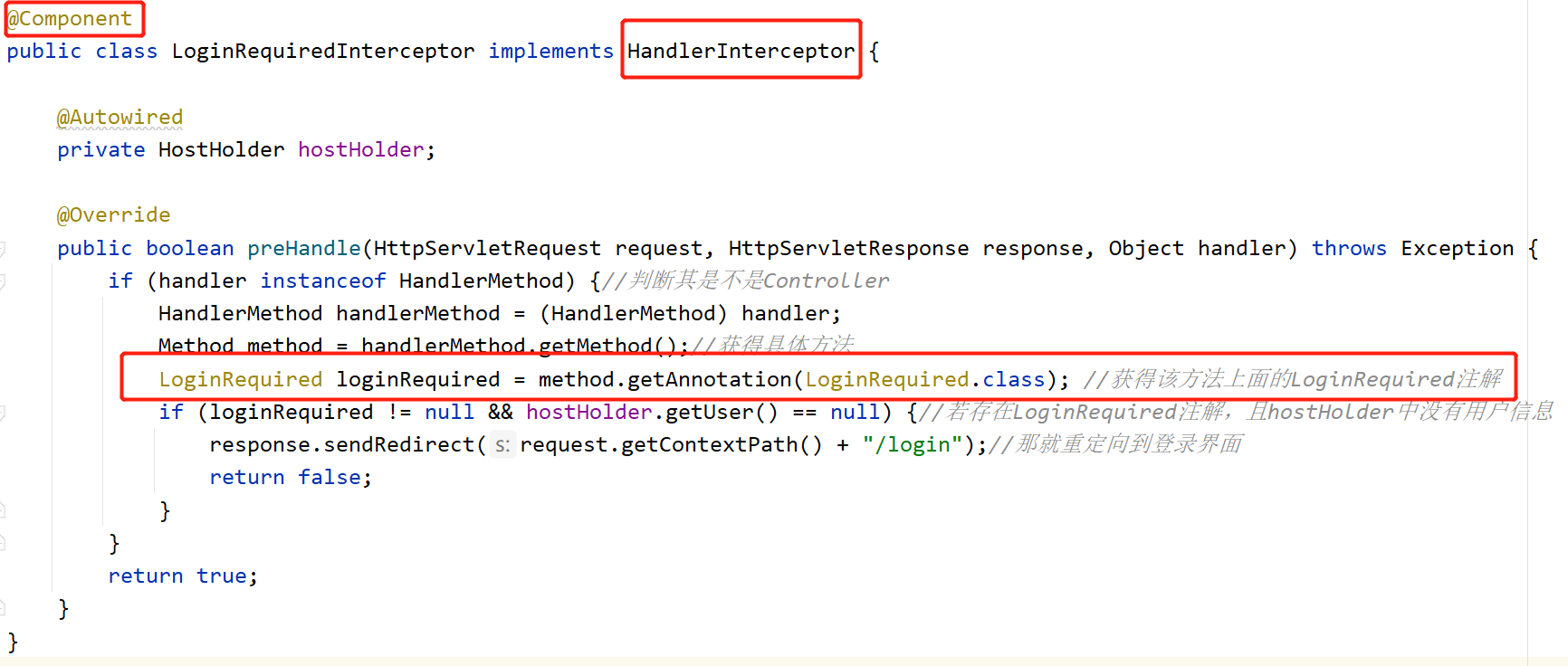
注解其实是代码里面的特殊标记，这些标记可以在编译、类加载、运行时被读取，并执行相应的处理。

注解是一个接口，程序可以通过反射来获取制定程序元素的Annotation对象，以此来取得注解里面的元数据。

自定义注解：



拦截器，拦击带有LoginRequired注解的controller方法。带有此注解意味着需要登录检查。



## ThreadLoca+Redis替代Session，在线程内保存用户信息



对一个请求的处理是在一个线程内实现的。

因此可以使用HostHolder保存用户信息，使用拦截器，在访问到达controller之前就根据请求中的cookie中的sessionid查找redis中对应的用户信息（包含用户名、过期时间等等），保存在Hostholder中，因此不论是在controller还是service处需要使用用户信息的时候，就直接注入HostHolder对象即可。

且该对象内的成员变量是ThreadLocal，因此，对应不同的线程都是不同的用户。

这样就避免了每次需要用户信息就去Redis中查找了。也不需要想Service中单独传入用户信息或者sessionid了。

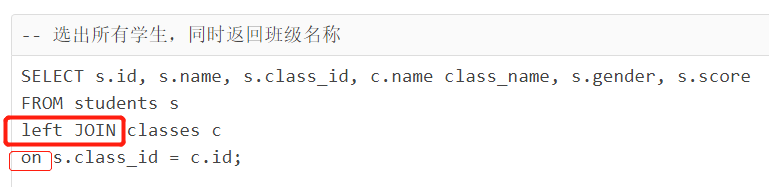
## 连接查询：

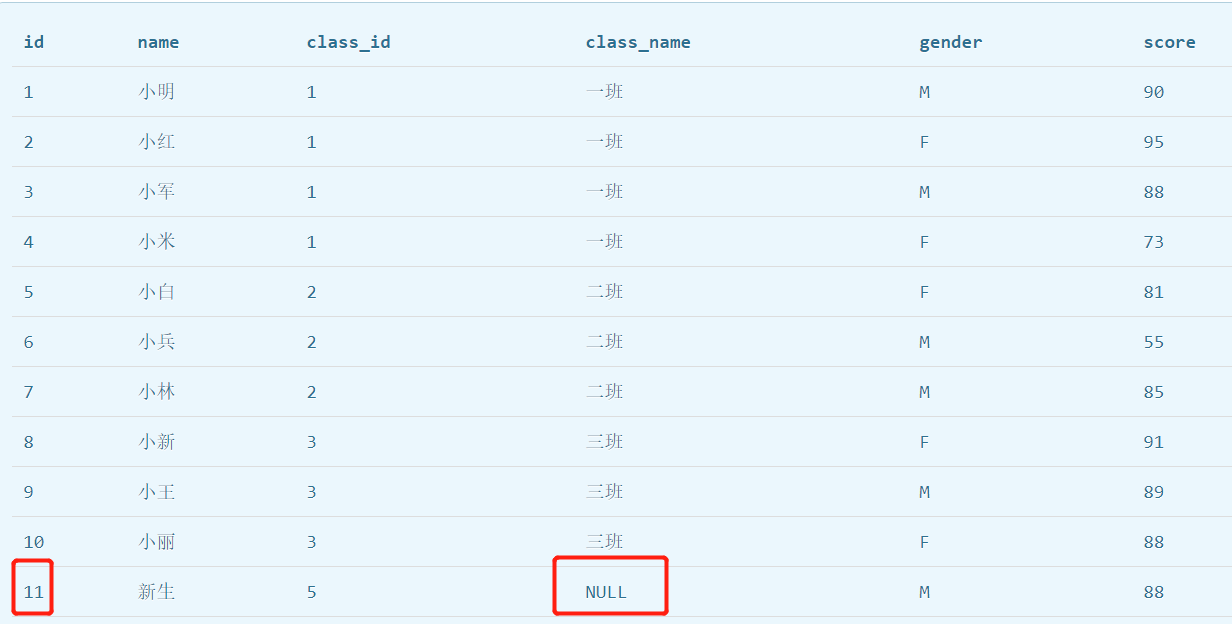
https://www.liaoxuefeng.com/wiki/1177760294764384/1179610888796448

**SELECT** ... **FROM** tableA ??? **JOIN** tableB **ON** tableA.column1 = tableB.column2

注意：区分以下两个查询语句的区别：

1、

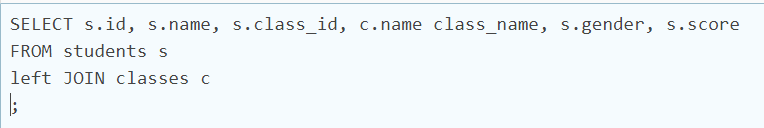




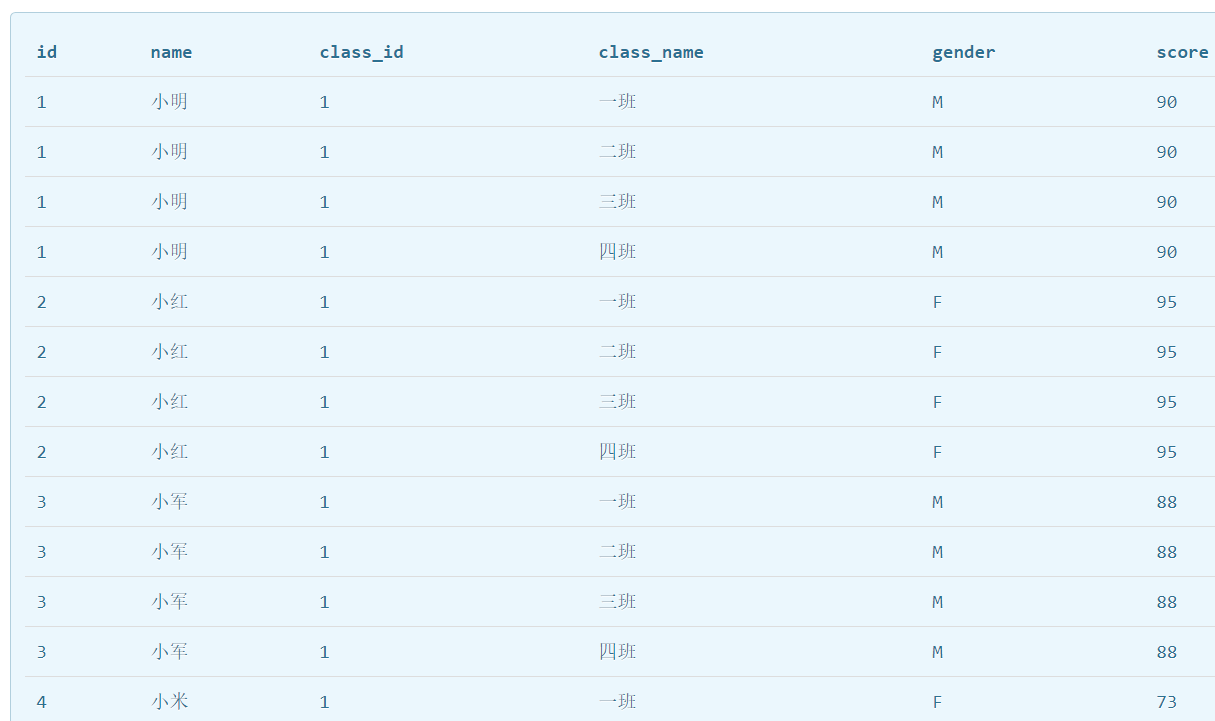
语句意义为：以s.class\_id=c.lcass\_id为条件，将表c左连接至表s当中。

显然，包含左表的全部数据。注意第11条，其class\_name列为空，意味着在表c中，没查到符合s.class\_id=c.lcass\_id条件的。就没有记录可以接到s此行的右侧。

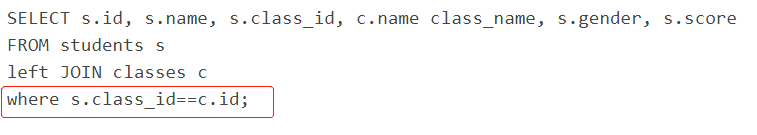
2、



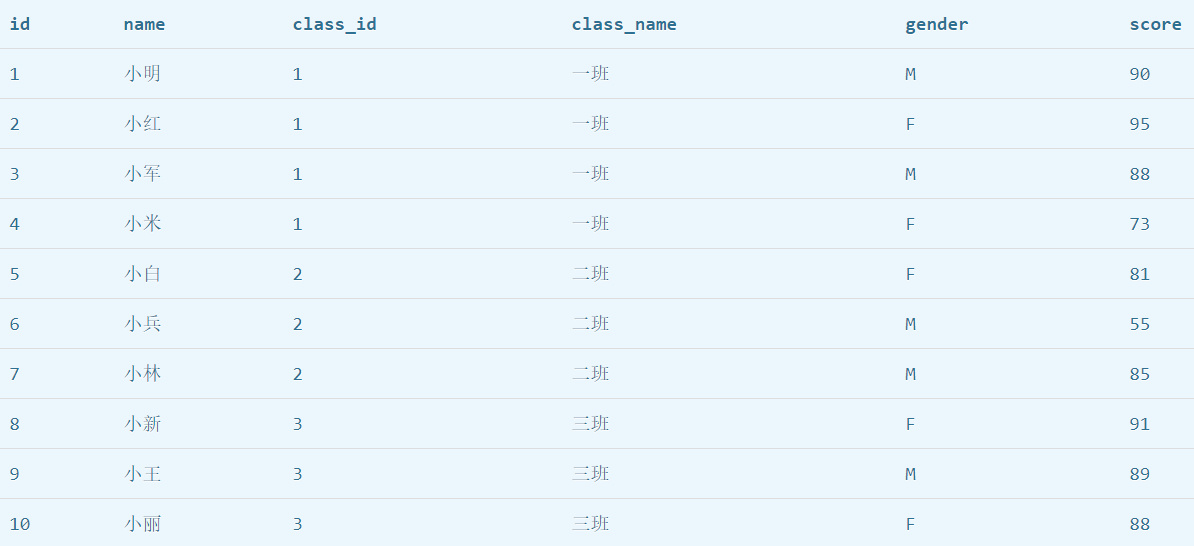
无条件进行左连接，即笛卡尔积。得到结果如下：

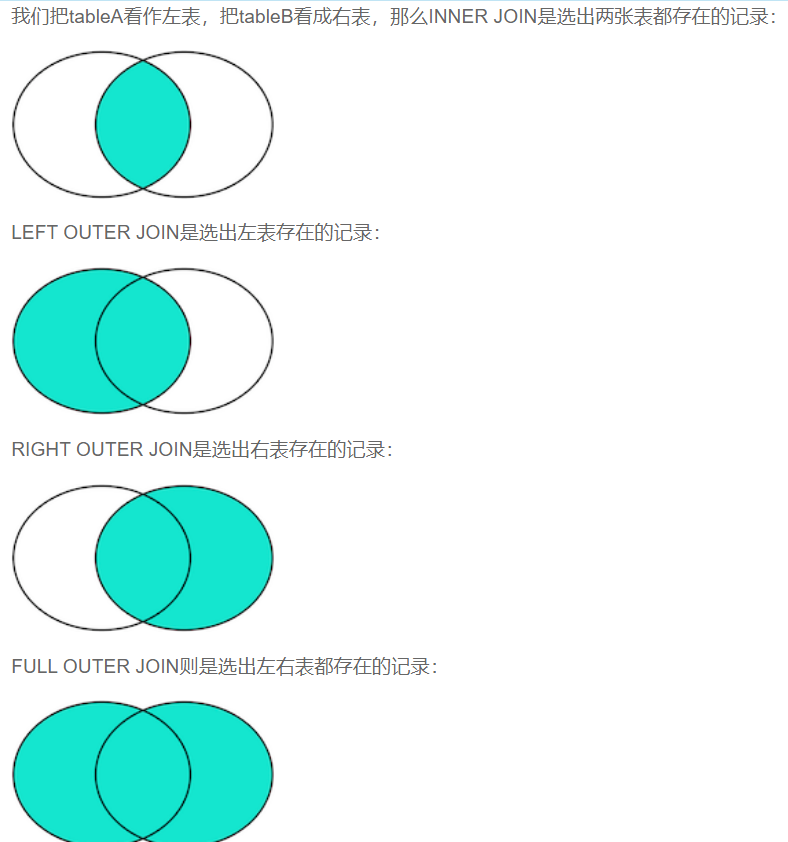


3、



在刚才笛卡尔积的基础上，进行了条件查询。由于在新生那一行的class\_id != c.id，因此此时，不包含第十一行。且**查询效率低于直接on条件**。





## 创建线程的方式

<http://www.cyc2018.xyz/Java/Java%20%E5%B9%B6%E5%8F%91.html#%E5%AE%9E%E7%8E%B0-callable-%E6%8E%A5%E5%8F%A3>

4种方式：

1. 继承Thread
2. 实现Runable
3. 实现Callable
4. 线程池

实现 Runnable 和 Callable 接口的类只能当做一个可以在线程中运行的任务，不是真正意义上的线程，因此最后还需要通过 Thread 来调用。可以理解为任务是通过线程驱动从而执行的。







实现接口 VS 继承 Thread

实现接口会更好一些，因为：

* Java 不支持多重继承，因此继承了 Thread 类就无法继承其它类，但是可以实现多个接口；
* 类可能只要求可执行就行，继承整个 Thread 类开销过大。

# vivo提前批后端开发一面 2021/6/22

## 看服务器有多少个tcp连接的命令

<https://blog.csdn.net/he_jian1/article/details/40787269>

netstat 命令

## 所有的对象都是建立在堆上的吗？

Java中的对象不一定是在堆上分配的，因为JVM通过逃逸分析，能够分析出一个**新对象的使用范围**，并以此确定是否要将这个对象分配到堆上。

逃逸分析就是：一种确定指针动态范围的静态分析，它可以分析在程序的哪些地方可以访问到指针。

对象逃逸示例

一种典型的对象逃逸就是：对象被复制给成员变量或者静态变量，可能被外部使用，此时变量就发生了逃逸。



另一种典型的场景就是：对象通过return语句返回。如果对象通过return语句返回了，此时的程序并不能确定这个对象后续会不会被使用，外部的线程可以访问到这个变量，此时对象也发生了逃逸。



逃逸分析的优点：

1. 对象可分配在栈上

JVM通过逃逸分析，分析出新对象的使用范围，就可能将对象在栈上进行分配。栈分配可以快速地在栈帧上创建和销毁对象，不用再将对象分配到堆空间，可以有效地减少 JVM 垃圾回收的压力。

1. 分离对象或标量替换

当JVM通过逃逸分析，确定要将对象分配到栈上时，即时编译可以将对象打散，将对象替换为一个个很小的**局部变量**，我们将这个打散的过程叫做标量替换。将对象替换为一个个局部变量后，就可以非常方便的在栈上进行分配了。

1. 同步锁消除

如果JVM通过逃逸分析，发现一个对象只能从一个线程被访问到，则访问这个对象时，可以不加同步锁。如果程序中使用了synchronized锁，则JVM会将synchronized锁消除。

这里，需要注意的是：这种情况针对的是synchronized锁，而对于Lock锁，则JVM并不能消除。

## TCP的状态有哪些？

LISTEN： 侦听来自远方的TCP端口的连接请求

SYN-SENT： 再发送连接请求后等待匹配的连接请求

SYN-RECEIVED：再收到和发送一个连接请求后等待对方对连接请求的确认

ESTABLISHED： 代表一个打开的连接

FIN-WAIT-1： 等待远程TCP连接中断请求，或先前的连接中断请求的确认

FIN-WAIT-2： 从远程TCP等待连接中断请求

CLOSE-WAIT： 等待从本地用户发来的连接中断请求

CLOSING： 等待远程TCP对连接中断的确认

LAST-ACK： 等待原来的发向远程TCP的连接中断请求的确认

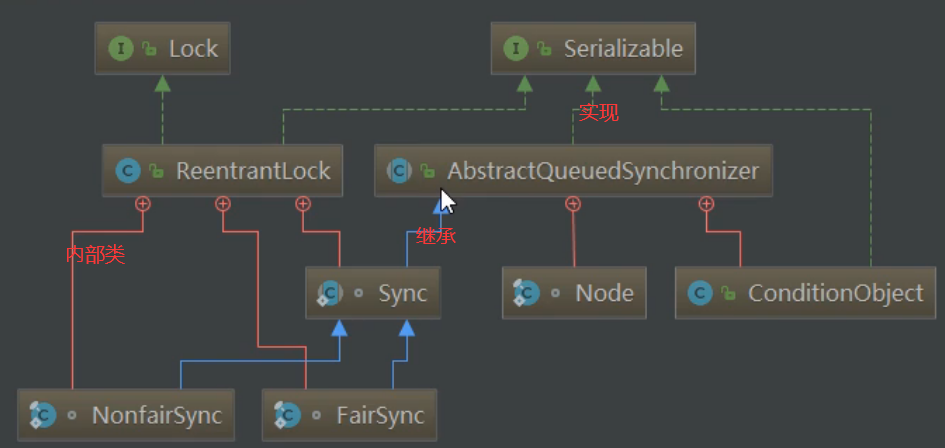
TIME-WAIT： 等待足够的时间以确保远程TCP接收到连接中断请求的确认

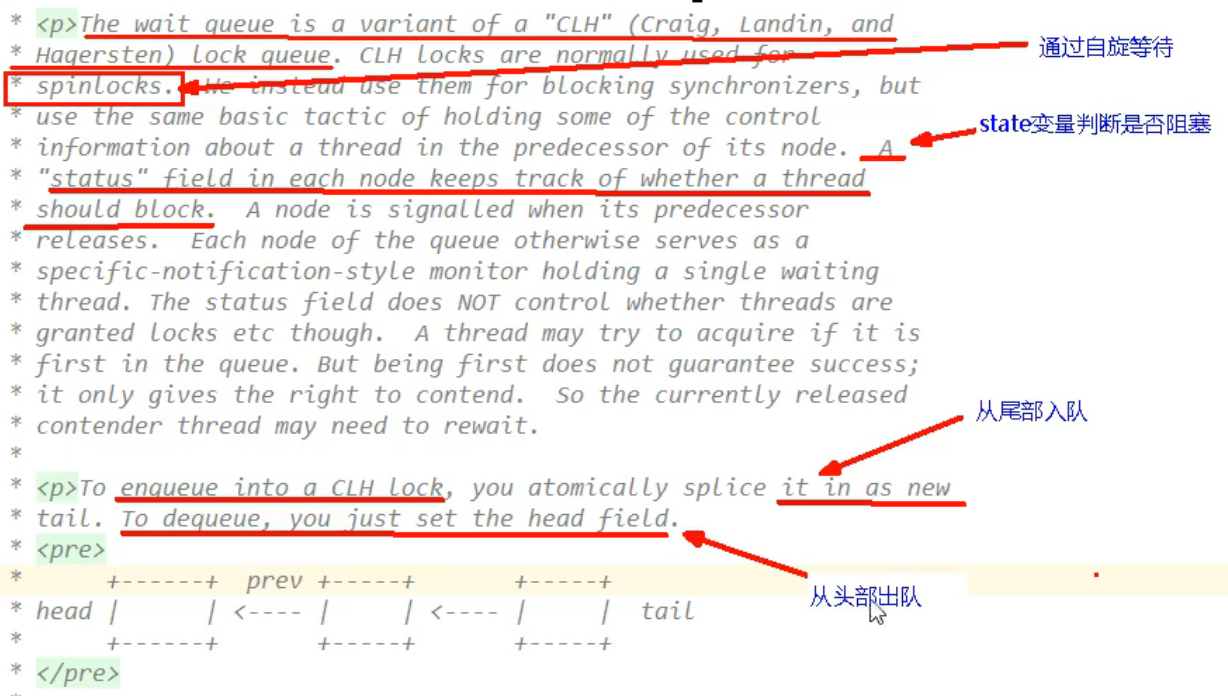
CLOSED： 没有任何连接状态

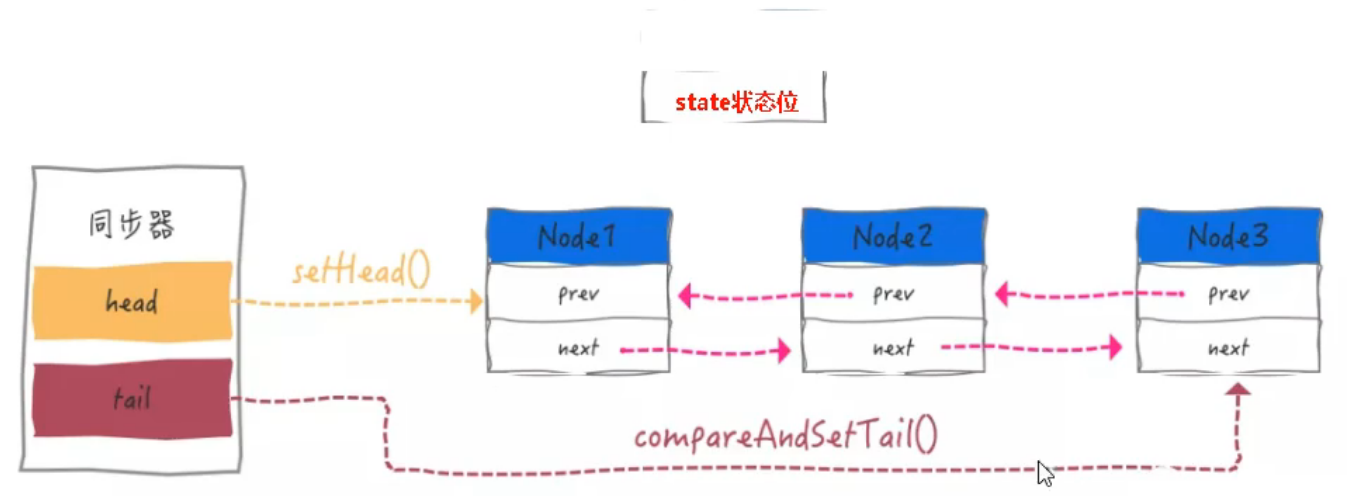
# 高并发

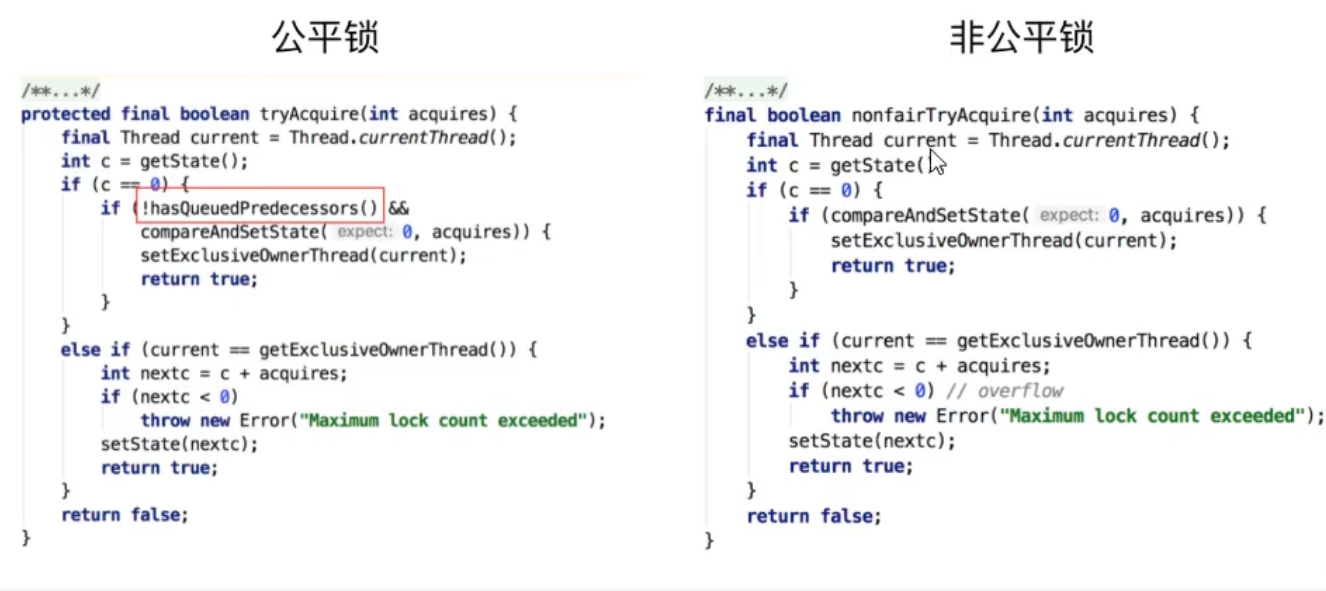
## AQS

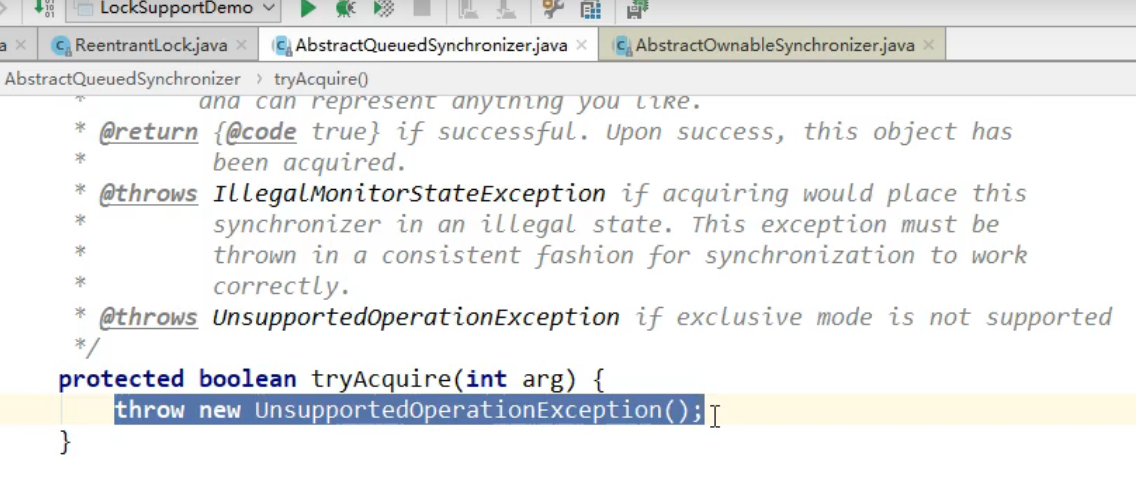
AbstractQueuedSynchronizer，是用来构建锁和同步器的框架。例如：ReentrantLock，Semaphore，CountDownLatch，CyclicBarrier均是由AQS实现的。











模板方法：逼着子类必须去实现这个方法，否则我就直接抛出异常。

