【第11话：帮小伙伴们整理一套最全的Java锁，无论面试官怎么问都在这里了】

Hello 小伙伴们，这节课给大家讲解一下：“Java中的锁”。

小伙伴在学习Java的时候关于锁相关的名词可能听过很多：乐观锁、悲观锁、互斥锁、排它锁、共享锁、读锁、写锁、内置锁、显式锁、对象锁、类锁、重入锁、非重入锁、死锁、公平锁、非公平锁、锁升级、锁降级等等，这个感觉像极了相声里面的报菜名：蒸羊羔,蒸熊掌,蒸鹿尾儿, 烧花鸭,烧雏鸡儿,烧子鹅...

别看这么多锁的名称，实际在Java中，我们能见到的落地技术：synchronized、Lock、CAS。这三个技术把上面的这些锁都给包含了，所以每个技术都是身兼数职。这和屏幕前的小伙伴们一样，你在学校里面你就是学生，你的主要任务是学习。你在家里父母眼中，你就是个孩子，只要能健健康康，顺顺利利成长就行。你在公司里面，你就是员工，你的主要任务就是完成领导安排的任务。你还是你，不一样的烟火。

所以这么多词语，他们都是从不同角度去看待进行的分类。下面整理出来大致分为下面几类：

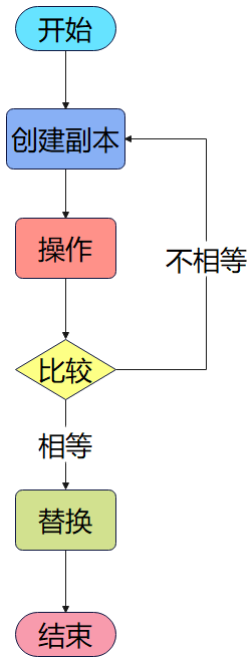
- 是否必须上锁：乐观锁、悲观锁
- 是否能继续获得锁：重入锁、非重入锁
- 锁的特性：互斥锁、排它锁、共享锁、读锁、写锁、锁升级、锁降级
- 锁的资源竞争形势：公平锁、非公平锁
- 是否需要程序员释放锁：内置锁、显式锁

这些锁里面在面试过程中问的最多的就是乐观锁和悲观锁，还有重入锁和非重入锁，内置锁和显式锁（但是问法不是这么问）。其他的锁偶尔可能会被问题。下面我们来一组一组的进行解释。

乐观锁和悲观锁

悲观锁：顾名思义，很悲观的锁。一旦一个线程使用悲观锁时，其他线程只能等待锁释放，才能继续操作。典型代表：synchronized关键字。

乐观锁：通过名称很好记忆，就是很乐观的锁。一个线程操作时，不会锁住，其他线程也可以操作。典型代表：CAS。



既然说到CAS了，就必须好好给小伙伴们说一说：什么是CAS。因为在面试过程中CAS偶尔也会被问到。

CAS 是 Compare And Swap 缩写，取了三个单词首字母。翻译过来：比较和交换算法。

每次在修改值时的思路：

1. 创建内存中旧值副本。
2. 计算最终需要设置的值。
3. 比较副本值是否和内存值相等。

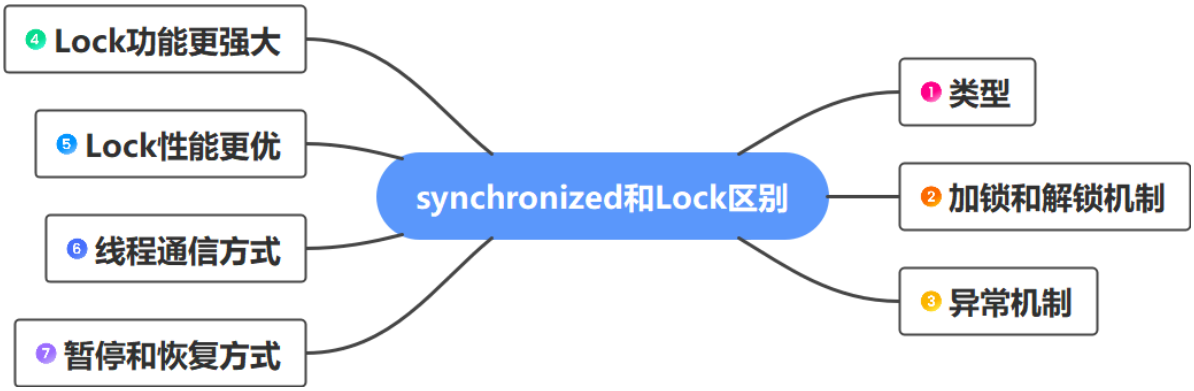
    如果相等，修改内存值为需要设置的值。

    如果不相等，重新创建内存中新值的副本，重新执行全套流程。

内置锁和显式锁

内置锁：所谓的内置锁就是指synchronized关键字。

显式锁：就是指Lock。



它们的区别，帮助小伙伴们总结如下：

1. 类型不同

   synchronized是关键字。修饰方法, 修饰代码块

   Lock是接口

2. 加锁和解锁机制同步

   synchronized是自动加锁和解锁，程序员不需要控制。

   Lock必须由程序员控制加锁和解锁过程, 解锁时, 需要注意出现异常不会自动解锁

3. 异常机制

   synchronized碰到没有处理的异常，会自动解锁，不会出现死锁。

   Lock碰到异常不会自动解锁，可能出现死锁。所以写Lock锁时都是把解锁放入到finally{}中。

4. Lock功能更强大

   Lock里面提供了tryLock()/isLocked()方法，进行判断是否上锁成功。synchronized因为是关键字，所以无法判断。

5. Lock性能更优

   如果多线程竞争锁特别激烈时，Lock的性能更优。如果竞争不激烈，性能相差不大。

6. 线程通信方式不同

   synchronized 使用wait()和notify()线程通信。

   Lock使用Condition的await()和signal()通信。

7. 暂停和恢复方式不同

   synchronized 使用suspend()和resume()暂停和恢复，这俩方法过时了。

   Lock使用LockSupport中park()和unpark()暂停和恢复，这俩方法没有过时。

重入锁和非重入锁

重入锁的概念也是面试过程中可能被问到的问题。

某个线程已经获得了某个锁，允许**再次**获得锁，就是**可重入锁。**如果不允许再次获得锁就称为**不可重入锁。**Java中synchronized和ReenrentLock都是重入锁。

重入锁底层实现：可重入锁底层原理特别简单，就是计数器。

当一个线程第一次持有某个锁时会由monitor（监控器）对持有锁的数量加1，当这个线程再次需要碰到这个锁时，如果是可重入锁就对持有锁数量再次加1（如果是不可重入锁，发现持有锁为1了，就不允许多次持有这个锁了，阻塞），当释放锁时对持有锁数量减1，直到减为0，表示完全释放了这个锁。

synchronized重入锁代码示例

```
public class Demo {
  public static void main(String[] args) {
    Demo demo = new Demo22();
    new Thread(new Runnable() {
      @Override
      public void run() {
        demo.test1();
      }
    }).start();
  }

  public void test1(){
    synchronized (this){
      System.out.println("test1执行");
      test2();
    }
  }
  public void test2(){
    synchronized (this){
      System.out.println("test2执行");
    }
  }
}
```

公平锁、非公平锁

公平锁：严格按照顺序执行。先排队先执行（FIFO(First Input First Output)）。

非公平锁：多线程在等待时，如果发现可以竞争，谁竞争成功，谁获取锁。

非公平锁的效率要高于公平锁

ReentrantLock默认就是非公平锁。提供了一个有参构造方法来控制是否为公平锁。

```
/**
 * Creates an instance of {@code ReentrantLock}.
 * This is equivalent to using {@code ReentrantLock(false)}.
 */
public ReentrantLock() { sync = new NonfairSync(); }

/**
 * Creates an instance of {@code ReentrantLock} with the
 * given fairness policy.
 *
 * @param fair {@code true} if this lock should use a fair ordering policy
 */
public ReentrantLock(boolean fair) {
    sync = fair ? new FairSync() : new NonfairSync();
}
```

互斥锁、排它锁、共享锁、读锁、写锁、锁升级、锁降级

读锁：又叫共享锁。多个读锁可以共同执行。

写锁：又是排它锁/互斥锁。一个写锁线程执行，其他写锁线程等待。

读锁里面又用写锁，叫锁升级。ReadWriteLock不支持锁升级。出现死锁现象。

写锁里面又用读锁，锁降级。支持。

当多个线程又有读锁，又有写锁。读锁可以同时执行，但写锁需要等待读锁执行完成，才能执行。