java~理解可重入锁

在Java中，可重入锁（Reentrant Lock）是一种同步机制，允许线程在持有锁的情况下再次获取该锁，而不会被自己所持有的锁所阻塞
说，一个线程可以多次获得同一个锁，而不会出现死锁的情况。

可重入锁在多线程编程中非常有用，它允许线程在访问共享资源时多次获取锁，而不会引发死锁问题。当一个线程第一次获取锁后，会
个计数器，每次成功获取锁后计数器加1，每次释放锁后计数器减1。只有当计数器归零时，锁才会完全释放，其他线程才有机会获取该

可重入锁的一个重要特性是，如果一个线程已经持有了锁，那么它可以重复地获得该锁，而不会被自己所持有的锁所阻塞。这种机制可
的发生，因为线程可以在需要的时候重复获取锁，而不会被自己所持有的锁所阻塞住。

可重入锁的实现在Java中有多种选择，其中最常用的是 `ReentrantLock` 类。使用可重入锁可以通过以下步骤：

1. 创建可重入锁对象：可以使用 `ReentrantLock` 类的构造方法创建一个可重入锁对象，例如：

```
ReentrantLock lock = new ReentrantLock();
```

2. 获取锁：使用 `lock()` 方法获取锁，如果锁不可用，则当前线程将被阻塞，直到获取到锁为止，例如：

```
lock.lock();
```

3. 执行需要保护的临界区代码：获取到锁后，执行需要保护的临界区代码。
4. 释放锁：使用 `unlock()` 方法释放锁，确保在临界区代码执行完毕后释放锁，例如：

```
lock.unlock();
```

使用可重入锁的好处是它提供了更灵活的同步机制，可以方便地控制线程对共享资源的访问。但是需要注意的是，在使用可重入锁时，
锁都有相应的释放操作，以避免造成死锁或资源泄漏的问题。

synchronized是可重入锁吗

是的，Java中的 `synchronized` 关键字实际上是一种可重入锁。当一个线程获得了一个对象的锁之后，它可以多次地进入由这个对象的
`synchronized` 代码块，而不会被自己所持有的锁所阻塞。

可重入性是 `Synchronized` 的一个重要特性，它使得线程可以在持有锁的情况下再次获取相同锁，而不会产生死锁。当一个线程尝试进入
`synchronized` 保护的代码块时，会先尝试获取锁，如果锁已被其他线程持有，该线程会进入阻塞状态，等待锁的释放。然而，如果该
了同一个锁，它可以直接进入临界区，而不会被自己所持有的锁所阻塞。

下面是一个示例，展示了 `synchronized` 的可重入性：

```
class Counter {
    private int count = 0;

    public synchronized void increment() {
        count++;
        anotherMethod();
    }

    public synchronized void anotherMethod() {
        // 可以再次获取相同的锁
        count++;
    }
}
```

在上述示例中， `increment()` 方法和 `anotherMethod()` 方法都使用了 `synchronized` 关键字来实现同步。当一个线程调用 `increment()`
获取到 `Counter` 对象的锁，并执行 `count++` 操作，然后调用 `anotherMethod()` 方法。由于 `anotherMethod()` 方法也使用了相同的锁，
获取该锁并执行相应的操作。

需要注意的是，可重入性是在同一个锁对象上实现的，即线程需要持有同一个锁对象才能再次进入临界区。如果线程尝试在不同的锁对
锁，它仍然会被阻塞，因为每个锁对象是独立的。