1. Synchronized关键字
   1. 说说自己对synchronized关键字

Synchronized关键字解决的是多个线程之间访问资源的同步性，它可以保证被它修饰的方法或代码块在任意时刻只能有一个线程执行。

* 1. 说说自己是怎么使用synchronized关键字

1. 修饰实例方法（对象）

作用于当前对象实例加锁，进入同步代码前要获得当前对象实例的锁。

1. 修饰静态方法（类）

也就是给当前类加锁，进入同步代码块前要获得当前class的锁。

如果一个线程 A 调用一个实例对象的非静态 synchronized 方法，而线程 B 需要调用这个实例对象所属类的静态 synchronized 方法，是允许的，不会发生互斥现象，**因为访问静态 synchronized 方法占用的锁是当前类的锁，而访问非静态 synchronized 方法占用的锁是当前实例对象锁**。

1. 修饰代码块

需要指定加锁对象，对给定对象/类加锁。synchronized(this|object) 表示进入同步代码库前要获得**给定对象的锁**。synchronized(类.class) 表示进入同步代码前要获得 **当前 class 的锁**

双重校验锁实现对象单例：

public class Singleton {

private volatile static Singleton uniqueInstance;

private Singleton() {

}

public static Singleton getUniqueInstance() {

//先判断对象是否已经实例过，没有实例化过才进入加锁代码

if (uniqueInstance == null) {

//类对象加锁

synchronized (Singleton.class) {

if (uniqueInstance == null) {

uniqueInstance = new Singleton();

}

}

}

return uniqueInstance;

}

}

Volatile可以禁止指令重排序优化。

双重检验锁模式（double checked locking pattern），是一种使用同步块加锁的方法。程序员称其为双重检查锁，因为会有两次检查 instance == null，一次是在同步块外，一次是在同步块内。为什么在同步块内还要再检验一次？因为可能会有多个线程一起进入同步块外的 if，如果在同步块内不进行二次检验的话就会生成多个实例了。

* 1. 构造方法可以使用synchronized关键字修饰吗？

不能，由于构造方法本身就是线程安全的，不存在同步构造方法。

* + 1. synchronized关键字底层原理

属于JVM层面的

synchronized 同步语句块的实现使用的是 monitorenter 和 monitorexit 指令，其中 monitorenter 指令指向同步代码块的开始位置，monitorexit 指令则指明同步代码块的结束位置。

synchronized 修饰的方法并没有 monitorenter 指令和 monitorexit 指令，取得代之的确实是 ACC\_SYNCHRONIZED 标识，该标识指明了该方法是一个同步方法。

**不过两者的本质都是对对象监视器 monitor 的获取。**

* 1. 谈谈synchronized和reentrantlock的区别
     1. 两者都是可重入锁

可重入锁是指自己可以再次获得自己的内部锁。比如**一个线程**获得某个**对象锁**，但是这个对象锁还没释放，**当再次向获得这个对象的锁的时候，还是可以获取的。**如果不可锁重入的话，就能造成死锁。同一个线程获取锁，锁的计数器都自增一，所以要等待锁的计数器将为0时，才能释放锁。

* + 1. synchronized依赖于JVM而reentrantlock依赖于api

synchronized是依赖于jvm实现的，并没有直接暴露给我们。

Reetrantlock是jdk层面实现的（也就是api层面，需要lock()和unlock()方法，配合try/finally语句块完成）所以，我们可以查看它的源代码，如何实现。

* + 1. reentrantlock比synchronized增加了一些高级功能

1. 等待可中断：也就是说正在等待的线程可以选择放弃等待，改为处理其他事情。
2. 可实现公平锁：reentrantlock可以指定是公平锁还是非公平锁。而synchronzied只能是非公平锁。所谓公平锁就是先等待的线程先获得锁。而reentrantlock默认情况下是非公平的。
3. 可实现选择性通知

Synchronized关键字于wait()和notify/notify方法结合可以实现等待/通知机制。ReentrantLock类当然也可以实现，但是需要借助于Condition接口与newCondition()方法

1. Volatile关键字
   1. CPU缓存模型

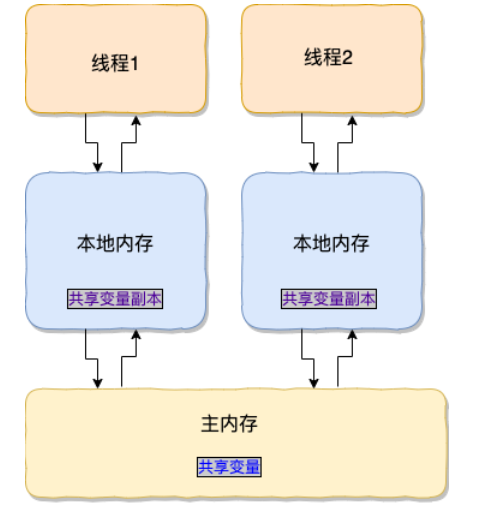
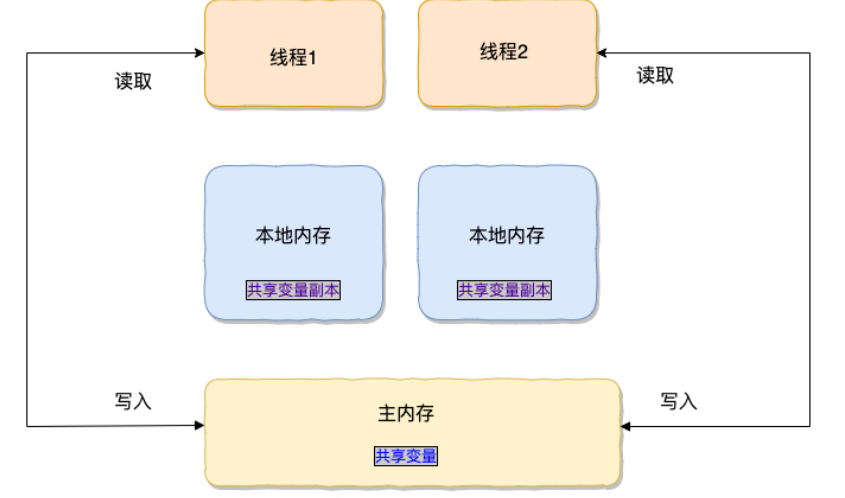
CPU缓存时为了解决CPU处理速度和内存处理速度不对等的问题。

我们可以把内存看成外存的告诉缓存，程序运行的时候，我们把外存的数据复制到内存，由于内存的处理速度远远高于外存，这样提高了处理速度。

**CPU 为了解决内存缓存不一致性问题可以通过制定缓存一致协议或者其他手段来解决。**

* 1. **讲一下JMM（JAVA内存模型）**

**Java的内存模型实现总是从主存读取变量，是不需要进行特别的注意的。但是在当前的java内存模型下，线程可以把变量保存在本地内存，而不直接在主存进行读写。这就可能造成，一个线程在主存中修改了一个变量的值，而另外一个线程还继续使用本地内存中的值，造成数据不一致。**

**因此，我们需要把变量声明为volatile（防止指令重排）， 保证变量的可见性。**

* 1. **并发编程的三个重要特性**

1. **原子性：**一个的操作或者多次操作，要么所有的操作全部都得到执行并且不会收到任何因素的干扰而中断，要么所有的操作都执行，要么都不执行。synchronized 可以保证代码片段的原子性。
2. 可见性：当一个变量对共享变量进行修改，那么另外的线程都是立即可以看到修改后的最新值的。Volatile关键字可以保证共享变量的可见性。
3. **有序性：代码在执行过程中的先后顺序，Java在编译器以及运行期间的优化，代码的执行顺序未必就是编写代码时候的顺序。Volatile关键字可以禁止之列进行重排序优化。**
   1. **说说synchronized关键字和volatile关键字的的区别**

**两者是互补的存在，不是对立的**

1. **Volatile关键字是线程同步的轻量级实现，但是它只能用于变量，而synchronized关键字可以修饰方法和代码块。**
2. **Volatile关键字能保证数据的可见性，但不能保证数据的原子性。Synchronized关键字两者都能保证。**
3. volatile 关键字主要用于解决变量在多个线程之间的可见性，而 synchronized 关键字解决的是多个线程之间访问资源的同步性。

**线程池：**

**主要思想是为了减少每次获取资源的消耗，提高对资源的利用率。**

**4.1 为什么要使用线程池？**

**降低资源消耗：通过重复利用已创建的线程降低线程创建和销毁造成的消耗。**

**提高响应速度：当任务到达时，任务可以不需要的等到线程创建就能立即执行。**

**提高线程的可管理性：**线程是稀缺资源，如果无限制的创建，不仅会消耗系统资源，还会降低系统的稳定性，使用线程池可以进行统一的分配，调优和监控。

4.2 实现runnable接口和callable接口的区别

**Runnable 接口**不会返回结果或抛出检查异常，但是**Callable 接口**可以。

所以，如果任务不需要返回结果或抛出异常推荐使用 **Runnable 接口**，这样代码看起来会更加简洁。

**4.3 执行execute()方法和submit方法的区别是什么？**

**1. execute()方法用于提交不需要返回值的任务，所以无法判断任务是否被线程池执行成功与否。**

**2. submit()方法用于提交需要返回值的任务。线程池会返回一个 Future 类型的对象，通过这个 Future 对象可以判断任务是否执行成功**，

**4.4 如何创建线程池**

**方式一：通过构造方法实现**

**方式二：通过 Executor 框架的工具类 Executors 来实现** 我们可以创建三种类型的