你们项目中那些地方有使用到多线程

注意：在分布式/高并发的项目中，其实多线程使用非常多。

1. 多线程的下载技术
2. 异步发送短信（多线程技术）
3. 异步回调中 采用多线程技术快速响应给支付接口
4. AOP 异步处理日志记录

多线程处理如何获取异步处理结果

主动根据业务的 id 查询

多线程处理技术中会遇到那些问题

线程安全问题：
多个线程同时共享同一个全局变量，可能会被其他的线程干扰 会遇到线程安全的问题。
解决方案：使用 lock 或者 syn、cas 无锁机制保证我们的线程安全问题

获取异步结果：
主动根据业务 id 查询

程序的死锁

可以采用诊断工具检测是否有发送死锁现象。

创建多线程的方式有哪些

1）继承 Thread 类创建线程类

2）通过 Runnable 接口创建线程类

3）通过 Callable 和 Future 创建线程

4）通过线程池创建

Java 线程具有五中基本状态

1)新建状态(New):当线程对象对创建后,即进入了新建状态,如:Thread t = new MyThread();

2）就绪状态（Runnable）：当调用线程对象的 start()方法（t.start();），线程即进入就绪状态。处于就绪状态的线程，只是说明此线程已经做好了准备，随时等待 CPU 调度执行，并不是说执行了 t.start()此线程立即就会执行；

3）运行状态（Running）：当 CPU 开始调度处于就绪状态的线程时，此时线程才得以真正执行，即进入到运行状态。注：就
绪状态是进入到运行状态的唯一入口，也就是说，线程要想进入运行状态执行，首先必须处于就绪状态中；

4）阻塞状态（Blocked）：处于运行状态中的线程由于某种原因，暂时放弃对 CPU 的使用权，停止执行，此时进入阻塞状态，直到其进入到就绪状态，才 有机会再次被 CPU 调用以进入到运行状态。

根据阻塞产生的原因不同，阻塞状态又可以分为三种：

a.等待阻塞：运行状态中的线程执行 wait()方法，使本线程进入到等待阻塞状态；

b.同步阻塞 – 线程在获取 synchronized 同步锁失败(因为锁被其它线程所占用)，它会进入同步阻塞状态；

c.其他阻塞 – 通过调用线程的 sleep()或 join()或发出了 I/O 请求时，线程会进入到阻塞状态。当 sleep()状态超时、join()等待线程终止或者超时、或者 I/O 处理完毕时，线程重新转入就绪状态。

5）死亡状态（Dead）：线程执行完了或者因异常退出了 run()方法，该线程结束生命周期。

synchronized 的作用

在 Java 中， synchronized 关键字是用来控制线程同步的，就是在多线程的环境下，控制 synchronized 代码段不被多个线程同时执行。
synchronized 既可以加在一段代码上，也可以加在方法上。

如何保证线程的安全性

使用 synchronized 或者是 Lock 锁