101_回眸再看：通过实战互联网教育系统，你收获了哪些能力？

儒猿架构官网上线，内有石杉老师架构课最新大纲，儒猿云平台详细介绍，敬请浏览

官网：www.ruyuan2020.com（**建议 PC 端访问**）

1、开篇

看到这里相信你是一个坚持学习，不断进步的小伙伴，我们不知不觉来到了第 13 周的课程了，再过去的 12 周里通过对互联网教学系统的搭建让大家熟悉了 Spring 各项技术。本节课作为本次课程的总结包括以下几个部分：

- 互联网教学系统整体架构
- Spring IOC 容器和 AOP 应用
- 使用 JDBC 进行数据库访问
- 使用 Spring Event 开发登陆模块
- 使用 Spring Cache 开发教师模块
- 使用 Spring 事务开发下单模块
- 使用 RocketMQ 开发订单取消和支付模块
- 使用 Spring 任务调度定时更新教师信息

2、互联网教学系统整体架构

作为互联网教学系统的总体架构也是本次课的学习指南，如图 1 所示，图的左边展示的是逻辑架构，主要关注的是服务层中需要实现的功能，包括：查询教师、选择教师、下单付款、评价订单、查看订单、发放优惠券等等。

左边的服务端架构图是技术实现，右边的图是为了实现左边图中的业务而生的。从右边的图可以看出本次课程使用了 Spring Web MVC、Spring IOC/AOP、Spring Event、Spring 事务、Spring 任务调度、Spring batch、RocketMQ、Druid 和 JDBC 等技术。

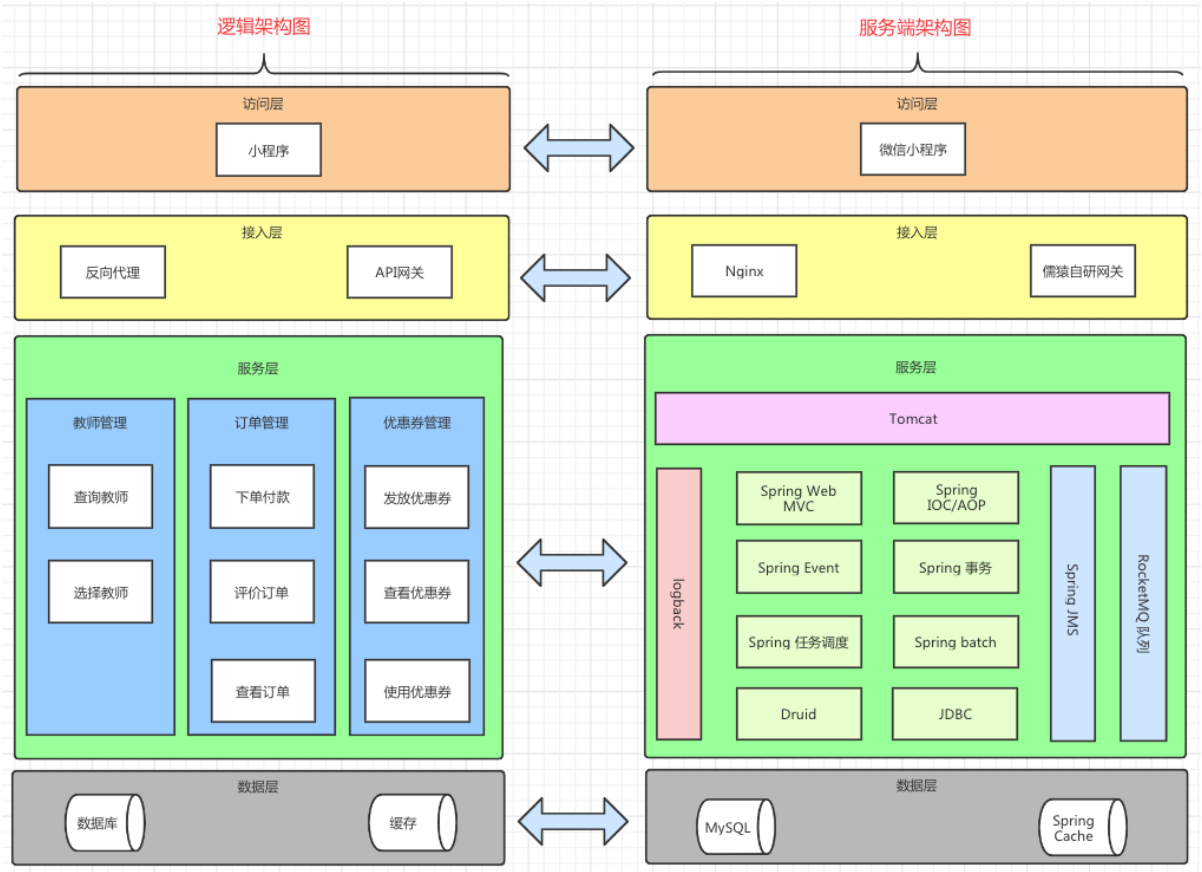

图 1 逻辑架构图和服务端架构图

3、Spring IOC 容器和 AOP 应用

Spring IOC 和 Spring AOP 是 Spring 核心开发的基础，因此从依赖查找、依赖注入以及依赖来源的方法让大家了解了 Spring IOC 的概念和工作原理。同样通过 JDK 动态代理和 CGLIB 动态代理以及 Spring AOP 的核心概念让大家对，Spring AOP 有深入的了解。

Spring IOC 方面我们介绍了依赖查找的方式、依赖注入、依赖来源

- 依赖查找：使用 Bean 实例需要通过某种途径，其中一种就成为依赖查询。具体的依赖查找方式很多，我们主要介绍了根据 Bean 名称查找和根据 Bean 类型查找。
- 依赖注入：也称为自动装载，当两个类存在依赖关系，通过配置<bean> 元素的 autowire 属性自动装配 <bean> 标签的方式获取依赖类的实例。这里我们介绍了根据 Bean 名称注入和根据 Bean 类型注入两种方式。

- 依赖来源：自定义 Bean 作为依赖来源、容器内建 Bean 对象作为依赖来源、容器内建依赖作为依赖来源。

  Spring IOC 在系统中主要负责实体的创建和管理工作。

  Spring AOP 方面介绍了 AOP 概念、CGLib 的动态代理和 Spring AOP。

  AOP （Aspect Orient Programming）,面向切面编程。AOP 是一种编程思想，是面向对象编程（OOP）的一种补充。面向对象编程将程序抽象成各个层次的对象，而面向切面编程是将程序抽象成各个切面。

  CGLib 的动态代理：在运行期间，通过字节码的方式在目标类生成的子类中织入对应的代码完成代理。CGLib 动态代理是 AOP 的一种实现形式。

  Spring AOP 是 Spring 对于 AOP 的一种实现，如图 2 所示，Aspect、Join Points、Pointcuts 以及 Advice，这些都属于 Spring AOP 的核心概念，Spring AOP 是通过这些概念的组合完成代码的织入工作的。

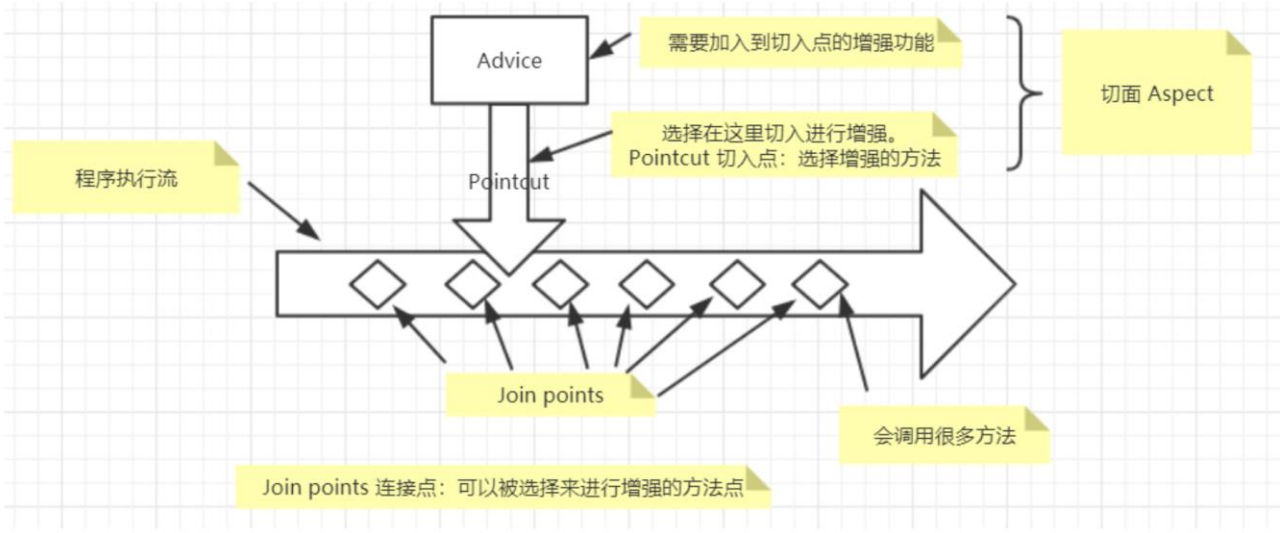

图 2 Spring AOP 架构图

Spring AOP 在系统中主要实现了日志记录和异常处理的工作。

4、使用 Mybatis 进行数据库访问

数据库访问作为系统基础模块，起到很重要的作用，我们先从 JDBC 的方式开始讲起，然后引入 Druid 连接池、JDBCTemplate，最后定格在 Mybatis 的访问方式。

如图 3 所示，MyBatis 应用程序根据 XML 配置文件创建 SqlSessionFactory，然后由 SqlSessionFactory 获取 SqlSession。通过 SqlSession 中的 Executor 执行映射

的 sql 语句，完成对数据的增删改查和事务提交等，用完之后关闭 SqlSession。

Sql 输入映射参数包括 HashMap、基本数据类型以及 POJO，经过 Executor 以后输出结果也是 HashMap、基本数据类型以及 POJO。

图 3 Mybatis 架构图

5、使用 Spring Event 开发登陆模块

在登陆模块的开发中我们遇到了同步还是异步的问题，并且提出 Spring Event 实现异步登陆的实现方式。

如图 4 所示，整个过程通过 ConsumerService、LogonEventPublisher、Spring Event 以及 LogonListener 串联起来。从左往右来看这张图，首先学员通过微信小程序“登陆”系统，通过 ConsumerService 提供的登陆功能，该功能中会判断是否第一次登陆，并且记录用户的登陆信息。

随后，通过 LogonEventPublisher 发布用户的登陆的消息，该消息通过 Spring Event 的消息对立让 LogonListener 的“登陆消息消费者”接收到，然后再更新用户

登陆的次数。整个过程通过 Spring Event 消息队列进行解耦，将登陆验证和登陆次数的更新变成两个异步操作。

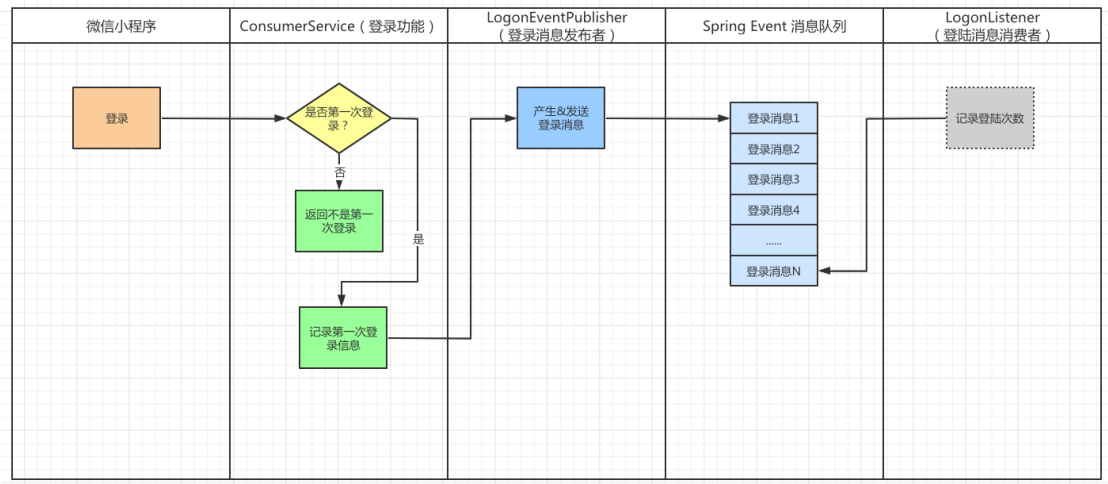

图 4 异步化登录流程

在异步登陆的基础上面还加入了发放优惠券的功能。如图 5 所示，依旧是微信小程序登陆系统，通过 ConsumerService 判断是否是第一次登陆，同时通过 LogonEventPublisher 进行登陆消息的发送。消息生产者会通过 Spring Event 消息队列发送了登陆事件，此时 LogonListener 会监听到登陆事件消息。之前登陆的业务逻辑中是直接，记录登陆的次数。

这里需要加入另外两个处理代码段。第一是判断是否第一次登陆，如果消费者是第一次登陆，那么调用数据库访问方法更新消费者的优惠券信息，也就是给消费者发放优惠券。第二是对消费者的生日进行判断，如果消费者登陆的日期也是他生日当月，那么也给消费者发放优惠券。在发放优惠券之后，用户还可以在“优惠券”菜单中查看优惠券的状态。

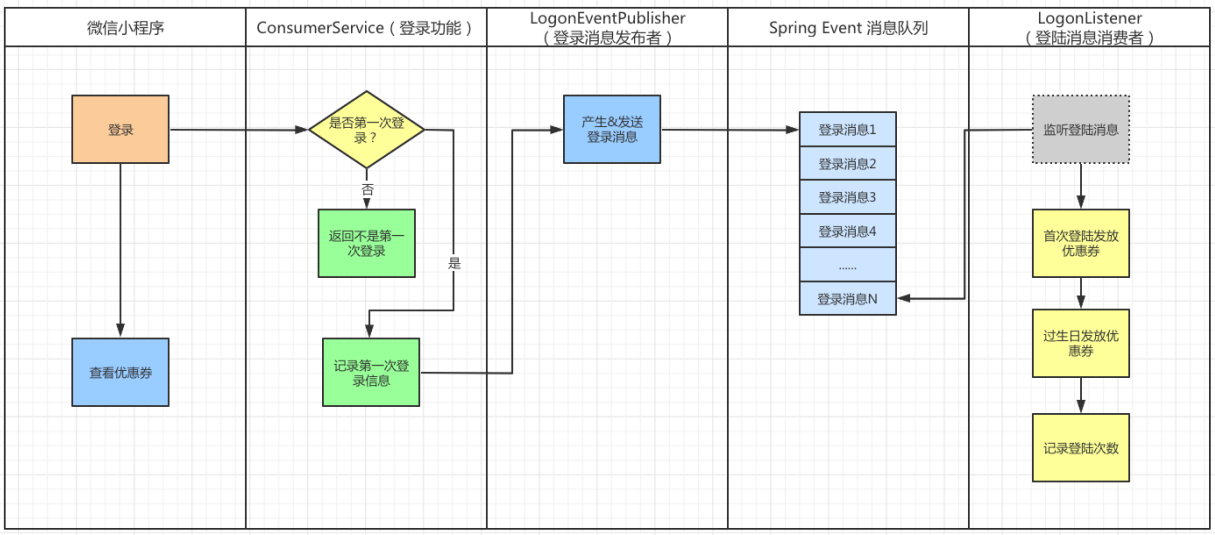

图 5 互联网教育系统优惠券发放流程

6、使用 Spring Cache 开发教师模块

在开发教师查询列表功能是，遇到了高并发下查询教师信息如何处理的问题。我们采用了 Spring Cache 的方式缓存教师信息，这样就不用每次都从数据库中获取教师信息，降低系统性能的损耗，提高用户体验。如图 6 所示，在消费者登陆微信小程序以后，通过“查看教师列表”向后台的 TeacherService 的教师服务发起请求。服务首先判断“是否有教师列表缓存”，第一次请求服务的时候一定是没有任何缓存的。于是就走向下的箭头，“从数据库获取教师列表”并且向消费者“返回教师列表”，于此同时会将教师列表的信息“写入缓存”到 Spring Cache 的服务中，也就是说现在已经将教师列表信息缓存了，下次访问的时候是可以直接使用该缓存中的信息的。

到了第二次访问教师服务时，又判断“是否有教师列表缓存”，此时返回“是”，也就是有缓存信息，于是通过“读取缓存”将 Spring Cache 中的教师列表信息返回给消费者，跳过了“从数据库获取教师列表”的部分。从缓存读取信息就完成了，在高并发场景下，对于教师列表的数据热点查询的问题就得到了解决。

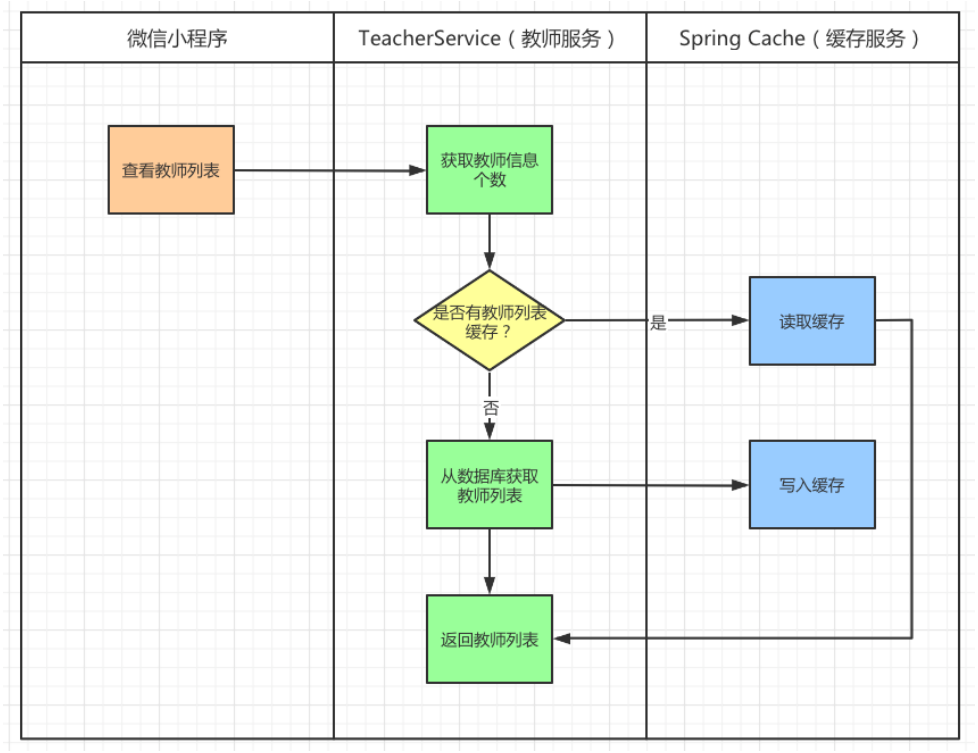

图 6 查看教师列表流程

7、使用 Spring 事务开发下单模块

在下单模块的开发中，我们发现下单不仅包括了创建订单，还包括了使用优惠券以及扣除积分功能。而且这三个操作需要放到同一个事务中处理。如图 7 所示，这个三个操作分别位于订单服务、优惠券服务以及消费者服务中，并且会访问不同的数据库表。

从业务的角度来看，这三个操作都在完成创建订单并且为订单付款，也就是我们前面提到的事务。这三个操作要么不完成，如果完成要一起完成，不可能出现创建了订单，但是没有使用优惠券或者没有扣除积分的情况，满足原子性特性。

同样用户支付的金额加上优惠券优惠的金额以及积分兑换的金额是等于最终的课程后，满足一致性特性。三个操作对数据的操作也是独立的，满足隔离性特性。最后，三个操作对数据会造成永久的改变，满足持久性特性。鉴于以上分析，我们将这三个操作用虚线框起来，把他们放到一个事务中处理。

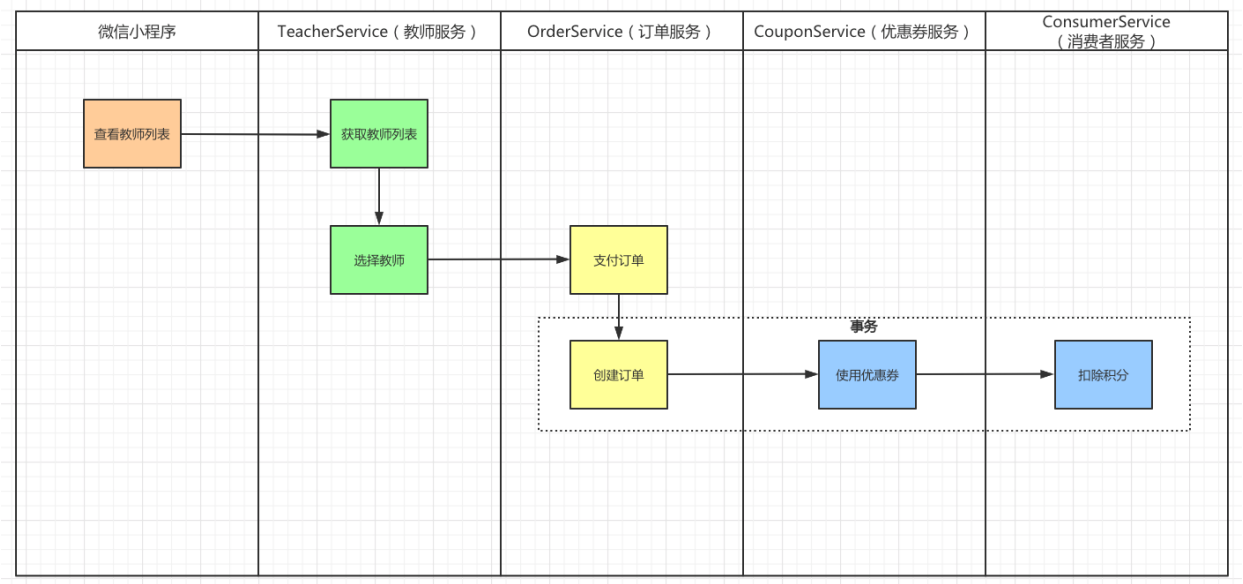

图 7 创建订单流程中的事务

7、使用 RocketMQ 开发订单取消和支付模块

在取消订单的功能中，我们发现如果创建订单一段时间都没有支付的情况，系统需要自动取消掉订单。于是就有了使用 RocketMQ 发送延迟消息的想法，里面涉及到了消息的生产者、消费者、监听器等组件。

如图 8 所示，将上述想法具体化为如下步骤：

1. 在创建订单的时候，就调用消息生产者生成一条延迟消息。
2. 这个延迟消息生产者的任务就是发送一条消息，消息在 30 分钟以后会被消费。
3. 延迟消息消费者会在 30 分钟以后获取消息通知，此时判断“订单是否支付？”
4. 如果此时订单没有支付，就满足了订单创建以后 30 分钟都没有支付的条件，于是调用取消订单的操作。完成系统自动取消订单的功能。

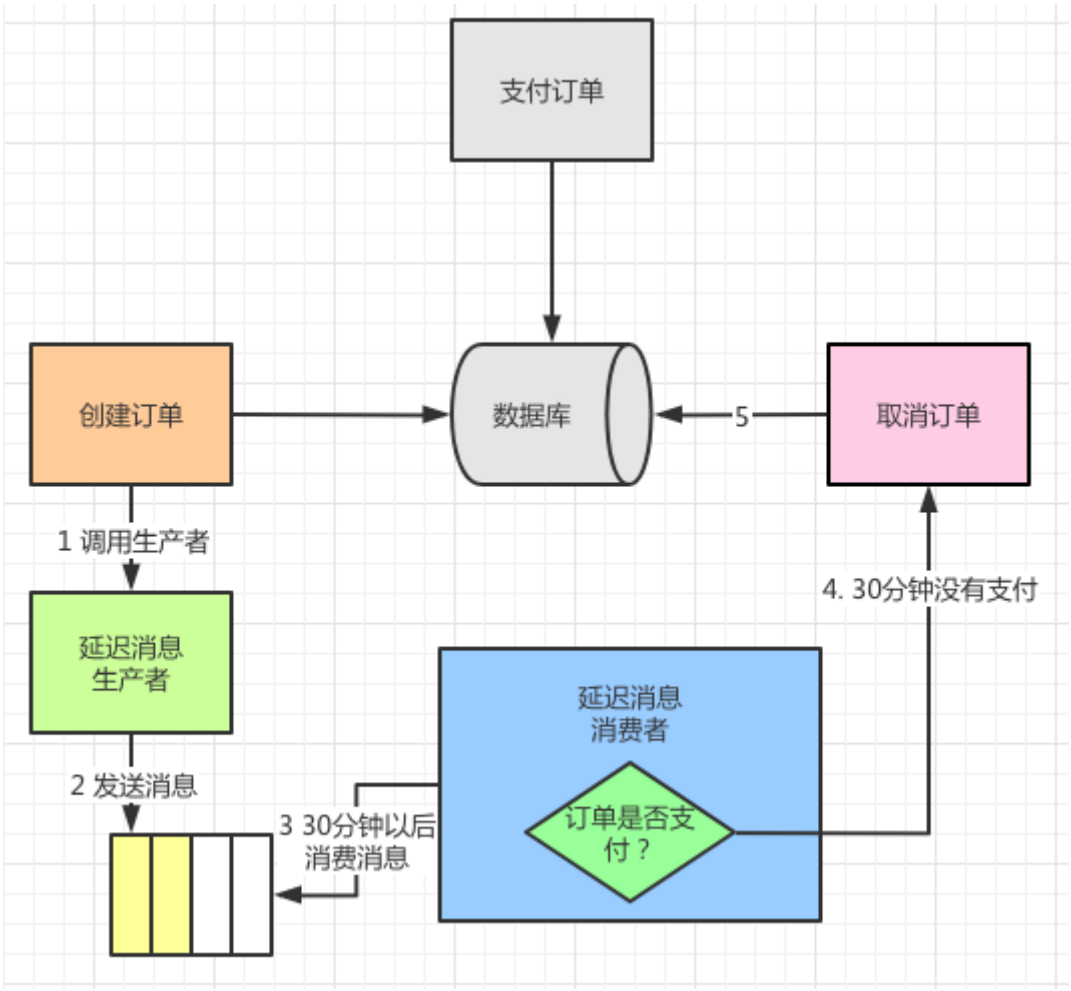

图 8 RocketMQ 的延迟消息机制

支付订单的场景与取消订单很相似也存在处理延迟消息的情况，于是我们如法炮制给支付订单到完成订单也设计了类似的方案。

如图 9 所示，下单流程完毕以后会进入支付订单的环节，这里分为两个部分超时处理和主动处理，我们来一起看看：

1. 支付订单完成以后会发送延迟消息，该消息发送到 RocketMQ 队列中。
2. 延迟消息消费者，在一段时间以后消费队列中的消息。这个时间可能是教学完成的几天以后，也或者更久。
3. 在判断消息类型是完成延迟订单以后，会进行完成订单的操作，包括：发优惠券和增加积分。

4. 如果在支付订单以后，点击“完成订单“按钮也会发送一个消息到 RocketMQ，这是一个即时消息，之所以在这里设计为消息的机制是为了让支付订单和完成订单两个功能进行解耦。
5. 消息消费者会消费队列中的完成订单消息，这里判断消息的类型就与上面不同了，这里判断的是完成订单的消息类型而不是延迟完成订单的消息类型。
6. 当满足条件了以后也会进行完成订单的操作。

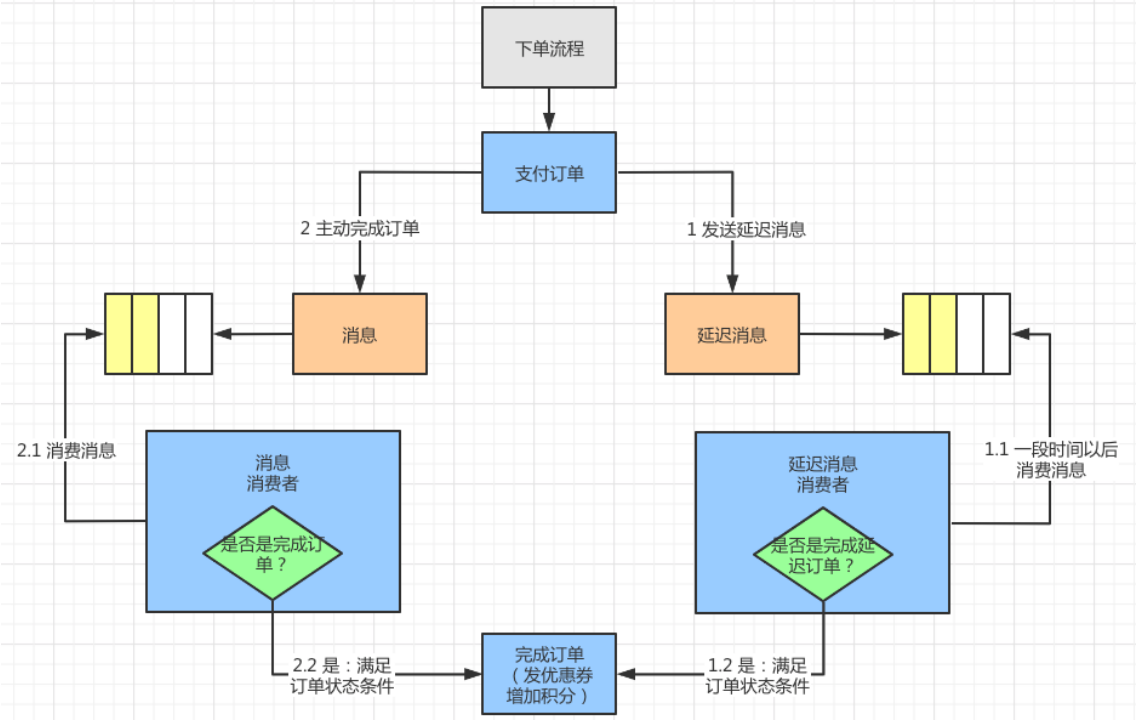

图 9 异步处理支付订单的流程图

8、使用 Spring 任务调度定时更新评分和教学天数

在开发教师评分更新功能的时候，我们对于业务关联性不强的评分信息采取了非实时的更新方式，也就是定期更新。如图 10 所示，在创建订单流程完毕以后，消费者会进行“评价订单”的操作，以及如下操作：

1. 此时有会一个 Task 在指定时间间隔，去获取评价订单信息，根据评价计算教师评分和授课总数。
2. 然后将教学总数和评分信息更新的教师信息的数据库中，便于搜索教师信息。

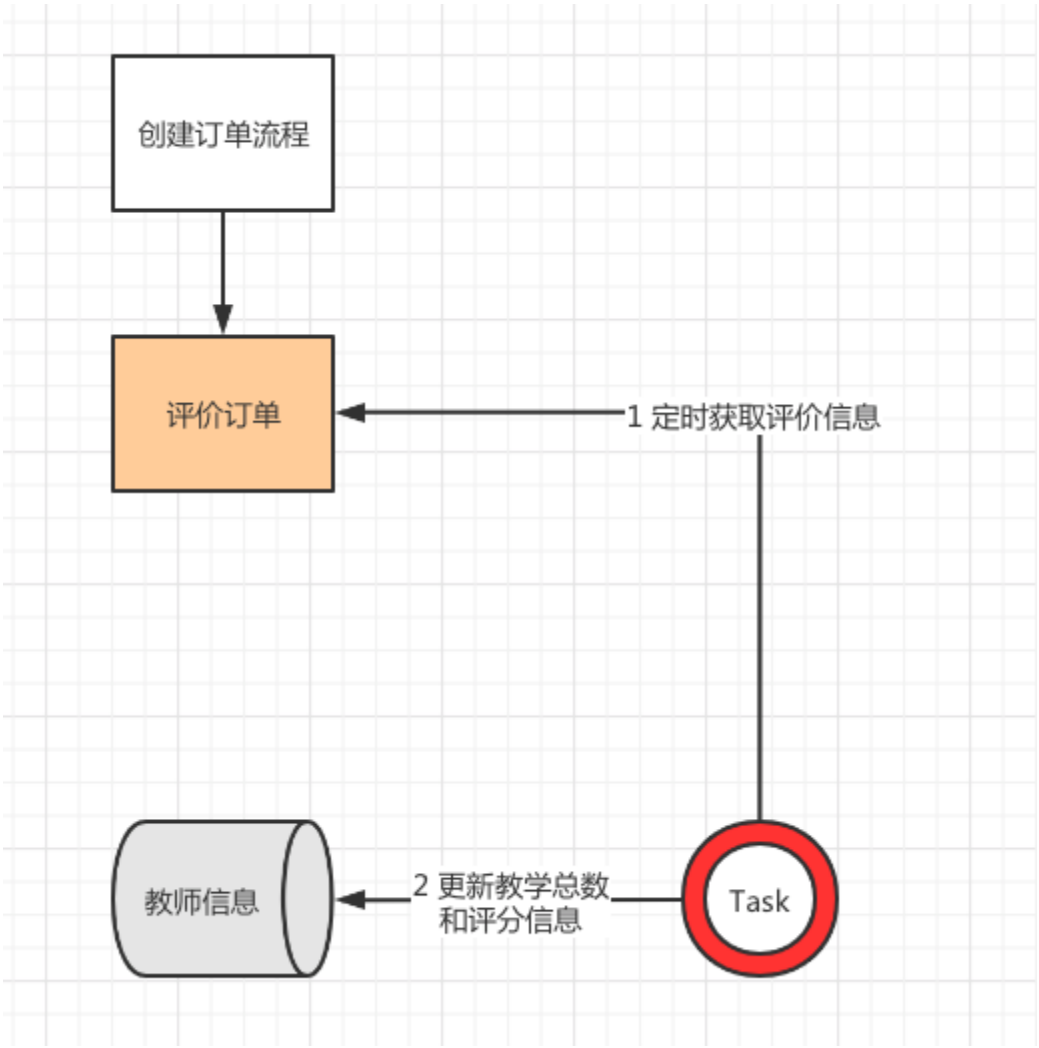

图 10 通过定时任务更新教师信息

有了教师评价的更新经验，在更新教师教学天数的时候就更加有经验了，依旧使用了 Spring 的计划任务，同时加入了 Spring Batch 处理批量数据的更新。这里需要对 Spring Batch 架构有所了解。

如图 11 所示，在整个架构的底部是 JobRepository，它是用来持久化领域对象的组件，它会去配置对应的数据源或者数据库连接池，为存取对象做好准备，也是其他几个组件的基础。在从图的左边看起，首先是 JobLauncher 顾名思义它是用来启动 Job 的组件，可以根据 JobLauncher 对应的配置启动 Job 的实例。

接下来，就是 Job 了，它是实际运行计划任务的具体实现，它和 Step 的关系是一对多。也就是一个 Job 会包括多个 Step，在 Spring Batch 中对 Step 有具体的定

义，从图中最右边黄色的部分可以看出，包括三个部分的操作：ItemReader 数据项读取器；ItemProcessor 数据项处理器；ItemWriter 数据项写入器。

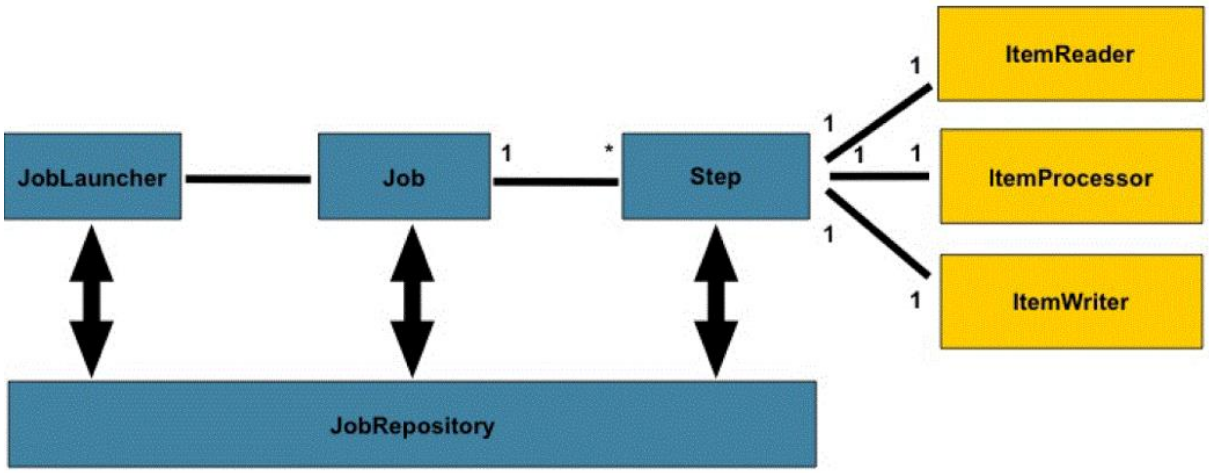

图 11 Spring Batch 的架构图

9、总结

本次针对 Spring 核心组件的开发课程就结束了，上面的内容是我们团队从 14 周课程抽取的技术和解决方案的精华部分。通过这节课的总结，让我们知道了整个互联网教育系统的业务和技术架构，同时也对每个业务场景设计了解决方案，并且利用合理的技术做支持。

这个总结篇可以收藏起来，在回顾知识的时候就不用把每周的课程打开搜索方案了，通过这一遍就能够总览所有课程的信息了。我们下次课程见，拜拜。