第4-6章作业-2020211502-王小龙

作业--思考题

1.

a.

分库分表是为了解决什么问题?

答:

- **性能角度：CPU、内存、磁盘、IO 瓶颈**
  - 随着业务体量扩大，数据规模达到百万行，数据库索引树庞大，查询性能出现瓶颈。
  - 用户并发流量规模扩大，由于单库(单服务器)物理性能限制也无法承载大流量。
- **可用性角度：单机故障率影响面**
  - 如果是单库，数据库宕机会导致 100%服务不可用，N 库则可以将影响面降低 N 倍。

采用了分库分表又会带来哪些新的问题?

答:

- **事务性问题**
  - 方案一：在进行分库分表方案设计过程中，从业务角度出发，尽可能保证一个事务所操作的表分布在一个库中，从而实现数据库层面的事务保证。
  - 方案二：方式一无法实现的情况下，业务层引入分布式事务组件保证事务性，如事务性消息、TCC、Seata 等分布式事务方式实现数据最终一致性。
  - 分库**可能**导致执行一次事务所需的数据分布在不同服务器上，数据库层面无法实现事务性操作，需要更上层业务引入分布式事务操作，难免会给业务带来一定复杂性，那么要想解决事务性问题一般有两种手段：
- **主键(自增 ID)唯一性问题**
  - 在数据库表设计时，经常会使用自增 ID 作为数据主键，这就导致后续在迁库迁表、或者分库分表操作时，会因为主键的变化或者主键不唯一产生冲突，要解决主键不唯一问题，有如下方案：
  - 方案一：自增 ID 做主键时，设置自增步长，采用等差数列递增，避免各个库表的主键冲突。但是这个方案仍然无法解决迁库迁表、以及分库分表扩容导致主键 ID 变化问题
  - 方案二：主键采用全局统一 ID 生成机制：如 UUID、雪花算法、数据库号段等方式。

- **跨库多表 join 问题**
  - 首先来自大厂 DBA 的建议是，线上服务尽可能不要有表的 join 操作，join 操作往往会给后续的分库分表操作带来各种问题，可能导致数据的死锁。可以采用多次查询业务层进行数据组装(需要考虑业务上多次查询的事务性的容忍度)
- **跨库聚合查询问题**

b.

垂直拆分：

- 垂直拆表
  - 即大表拆小表，将一张表中数据不同”字段“分拆到多张表中，比如商品库将商品基本信息、商品库存、卖家信息等分拆到不同库表中。
  - 考虑因素有将**不常用**的，**数据较大**，**长度较长**（比如 text 类型字段）的拆分到“扩展表“，表和表之间通过”主键外键“进行关联。
  - 好处：降低表数据规模，提升查询效率，也避免查询时数据量太大造成的“跨页”问题。
- 垂直拆库
  - 垂直拆库则在垂直拆表的基础上，将一个系统中的不同业务场景进行拆分，比如订单表、用户表、商品表。
  - 好处：降低单数据库服务的压力（物理存储、内存、IO 等）、降低单机故障的影响面

水平拆分：

- 操作：将总体数据按照某种维度(时间、用户)等分拆到多个库中或者表中，典型特征不同的库和表结构完全一下，如订单按照(日期、用户 ID、区域)分库分表。
- 水平拆表
  - 将数据按照某种维度拆分为多张表，但是由于多张表还是从属于**一个库**，其降低**锁粒度**，一定程度提升查询性能，但是仍然会有 IO 性能瓶颈。
- 水平拆库
  - 将数据按照某种维度分拆到多个库中，降低单机单库的压力，提升读写性能。

c.

核心组件：

关系数据库：用于主业务数据存储，提供事务型数据处理，是应用系统的核心数据存储。

高速缓存：对复杂或操作代价昂贵的结果进行缓存，加速访问。

搜索引擎：提供复杂条件查询和全文检索。

队列：用于将数据处理流程异步化，衔接上下游对数据进行实时交换。异构数据存储之间进行上下游对接的核心组件，例如数据库系统与缓存系统或搜索系统间的数据对接。也用于数据的实时提取，在

线存储到离线存储的实时归档。

非结构化大数据存储：用于海量图片或视频等非结构化数据的存储，同时支持在线查询或离线计算的数据访问需求。

结构化大数据存储：在线数据库也可作为结构化数据存储，但这里提到的结构化数据存储模块，更偏在线到离线的衔接，特征是能支持高吞吐数据写入以及大规模数据存储，存储和查询性能可线性扩展。可存储面向在线查询的非关系型数据，或者是用于关系数据库的历史数据归档，满足大规模和线性扩展的需求，也可存储面向离线分析的实时写入数据。

批量计算：对非结构化数据和结构化数据进行数据分析，批量计算中又分为交互式分析和离线计算两类，离线计算需要满足对大规模数据集进行复杂分析的能力，交互式分析需要满足对中等规模数据集实时分析的能力。

流计算：对非结构化数据和结构化数据进行流式数据分析，低延迟产出实时视图。

d.

- 应用层多写：这是实现最简单、依赖最少的一种实现方式，通常采取的方式是在应用代码中先向主存储写数据，后向辅存储写数据。这种方式不是很严谨，通常用在对数据可靠性要求不是很高的场景。因为存在的问题有很多，一是很难保证主与辅之间的数据一致性，无法处理数据写入失效问题；二是数据写入的消耗堆积在应用层，加重应用层的代码复杂度和计算负担，不是一种解耦很好的架构；三是扩展性较差，数据同步逻辑固化在代码中，比较难灵活添加辅存储。
- 异步队列复制：这是目前被应用比较广的架构，应用层将派生数据的写入通过队列来异步化和解耦。这种架构下可将主存储和辅存储的数据写入都异步化，也可仅将辅存储的数据写入异步化。第一种方式必须接受主存储可异步写入，否则只能采取第二种方式。而如果采用第二种方式的话，也会遇到和上一种『应用层多写』方案类似的问题，应用层也是多写，只不过是写主存储与队列，队列来解决多个辅存储的写入和扩展性问题。
- CDC（Change Data Capture）技术：这种架构下数据写入主存储后会由主存储再向辅存储进行同步，对应用层是最友好的，只需要与主存储打交道。主存储到辅存储的数据同步，则可以再利用异步队列复制技术来做。不过这种方案对主存储的能力有很高的要求，必须要求主存储能支持CDC技术。一个典型的例子就是MySQL+Elasticsearch的组合架构，Elasticsearch的数据通过MySQL的binlog来同步，binlog就是MySQL的CDC技术。

2.

a.

缓存一致性

缓存一致性是指业务在引入分布式缓存系统后，业务对数据的更新除了要更新存储以外还需要同时更新缓存，对两个系统进行数据更新就要先解决分布式系统中的隔离性和原子性难题。目前大多数业务在引入分布式缓存后都是通过牺牲小概率的一致性来保障业务性能，因为要在业务层严格保障数据的

一致性，代价非常高，业务引入分布式缓存主要是为了解决性能问题，所以在性能和一致性面前，通常选择牺牲小概率的一致性来保障业务性能。

缓存击穿

缓存击穿是指查询请求没有在缓存层命中而将查询透传到存储 DB 的问题，当大量的请求发生缓存击穿时，将给存储 DB 带来极大的访问压力，甚至导致 DB 过载拒绝服务。空数据查询(黑客攻击)和缓存污染（网络爬虫）是常见的引发缓存击穿的原因。什么是空数据查询？空数据查询通常指攻击者伪造大量不存在的数据进行访问（比如不存在的商品信息、用户信息）。缓存污染通常指在遍历数据等情况下冷数据把热数据驱逐出内存，导致缓存了大量冷数据而热数据被驱逐。缓存污染的场景我们目前还没有发现较好的解决方案，但是在空数据查询问题上我们可以改造业务，通过以下方式防止缓存击穿：

- 通过 bloomfilter 记录 key 是否存在，从而避免无效 Key 的查询；
- 在 Redis 缓存不存在的 Key，从而避免无效 Key 的查询；

解决方案：

1、设置热点数据永远不过期。

2、接口限流与熔断，降级。重要的接口一定要做好限流策略，防止用户恶意刷接口，同时要降级准备，当接口中的某些服务不可用时候，进行熔断，失败快速返回机制。

3、加互斥锁

缓存雪崩

缓存雪崩是指由于大量的热数据设置了相同或接近的过期时间，导致缓存在某一时刻密集失效，大量请求全部转发到 DB，或者是某个冷数据瞬间涌入大量访问，这些查询在缓存 MISS 后，并发的将请求透传到 DB，DB 瞬时压力过载从而拒绝服务。目前常见的预防缓存雪崩的解决方案，主要是通过对 key 的 TTL 时间加随机数，打散 key 的淘汰时间来尽量规避，但是不能彻底规避。

解决方案：

1. 缓存数据的过期时间设置随机，防止同一时间大量数据过期现象发生。
2. 如果缓存数据库是分布式部署，将热点数据均匀分布在不同的缓存数据库中。
3. 设置热点数据永远不过期

b.

1. 排行榜功能

有序集合sorted set经典使用场景。例如视频网站需要对用户上传的视频做排行榜，榜单维护可能是多方面：按照时间、按照播放量、按照获得的赞数等。

2. 共同关注功能

set使用场景。从Redis中读取登录用户的关注列表与查看用户的关注列表，然后进行交集操作，获取共同关注的用户id然后通过用户服务传入用户id数据获取用户基本信息

3. 基于地理位置的“附近的人”功能

使用geospatial（地理位置）数据类型。用户点击获取（授权）自己本身的位置信息，然后以自身的位置信息为中心进行查找附近的人。

c.

将验证码存入Redis中（redisKey对应用户）：

```
redisTemplate.opsForValue().set(redisKey, text, 3, TimeUnit.MINUTES);// text为随机验证码
```

读取验证码：

```
captcha = (String) redisTemplate.opsForValue().get(redisKey);
```

3.

a.

cachePut

更新缓存注解。不管对应key值是否有缓存数据，都执行。

cacheEvict

主动清除缓存注解。

b.

condition:

含义：缓存条件

使用说明：指示满足条件方执行缓存操作，一般使用参数作为条件

unless:

含义：否定缓存

使用说明：当条件为 true ，方法的返回值不会被缓存

4.

a.

消息队列中间件是分布式系统中重要的组件，主要解决应用耦合，异步消息，削峰填谷等问题。实现高性能、高可用、可伸缩和最终一致性架构。它主要用来暂存生产者生产的消息，供后续其他消费者来消费。它的功能主要有两个：a.暂存(存储)、b.队列(有序：先进先出)。其他大部分场景对数据的消费没有顺序要求，主要用它的暂存能力 。从目前互联网应用中使用消息队列的场景来看，主要有以下三个：

1. 异步处理数据
2. 系统应用解耦
3. 业务流量削峰

b.

RoutingKey：指定当前消息被谁接受

BindingKey：指定当前Exchange下，什么样的RoutingKey会被下派到当前绑定的Queue中

c.

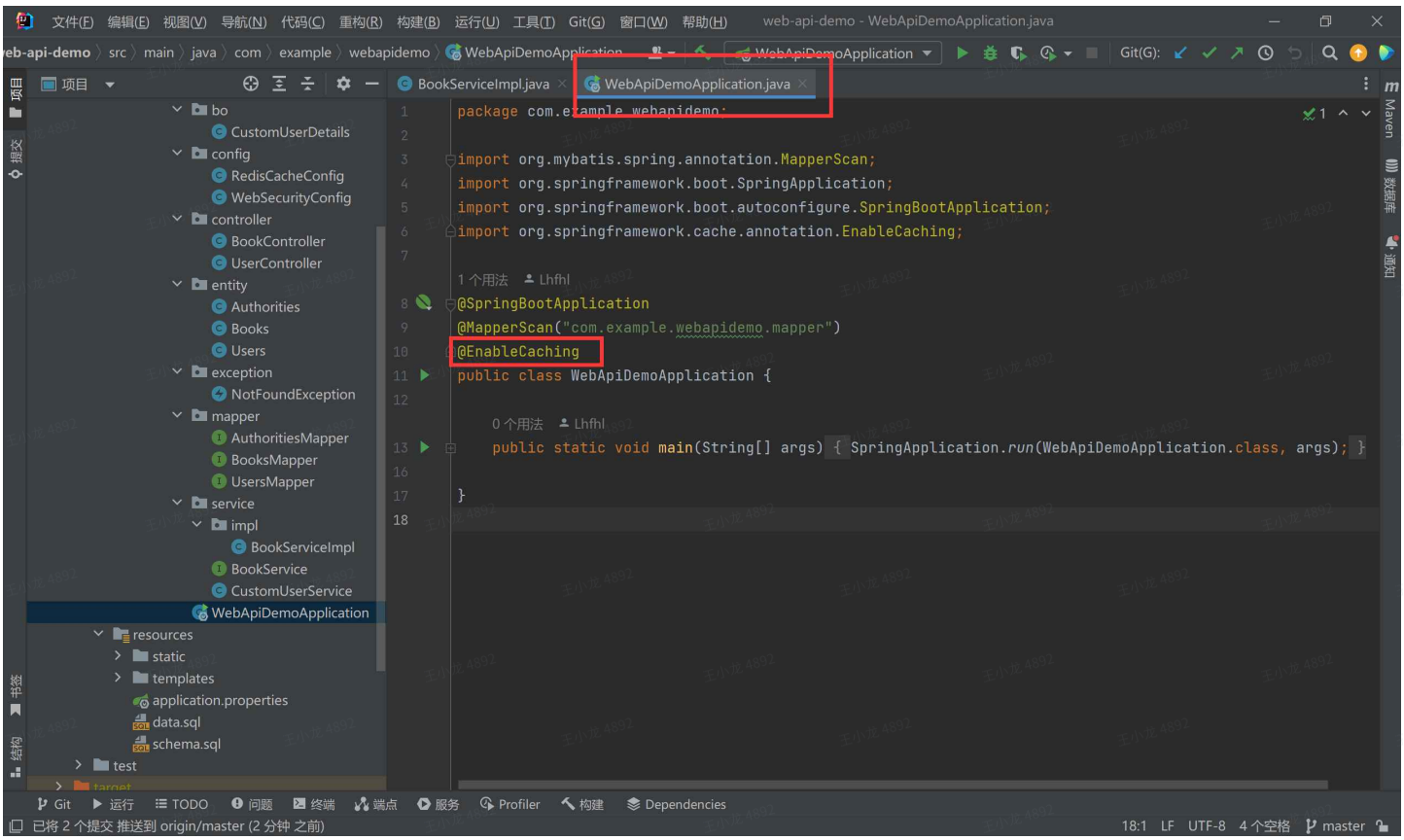
使用RabbitMQ的TTL与死信队列，设置消息的存活时间，在设置的时间内未被消费，即会投递到死信队列，监听死信队列即可。

作业-实验题

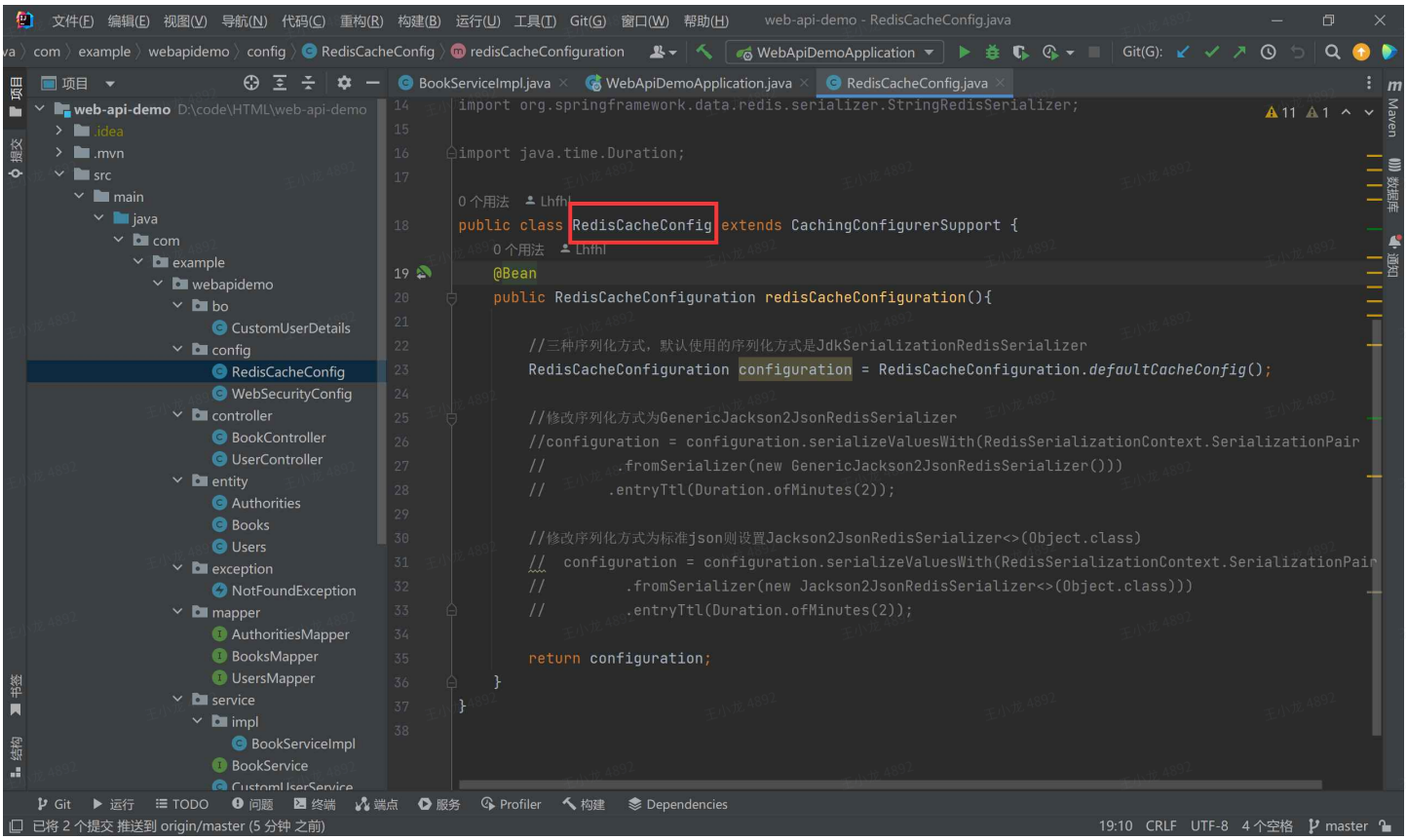
仓库地址（已添加为成员）：

https://gitee.com/lhfhlhfl/web-api-demo.git

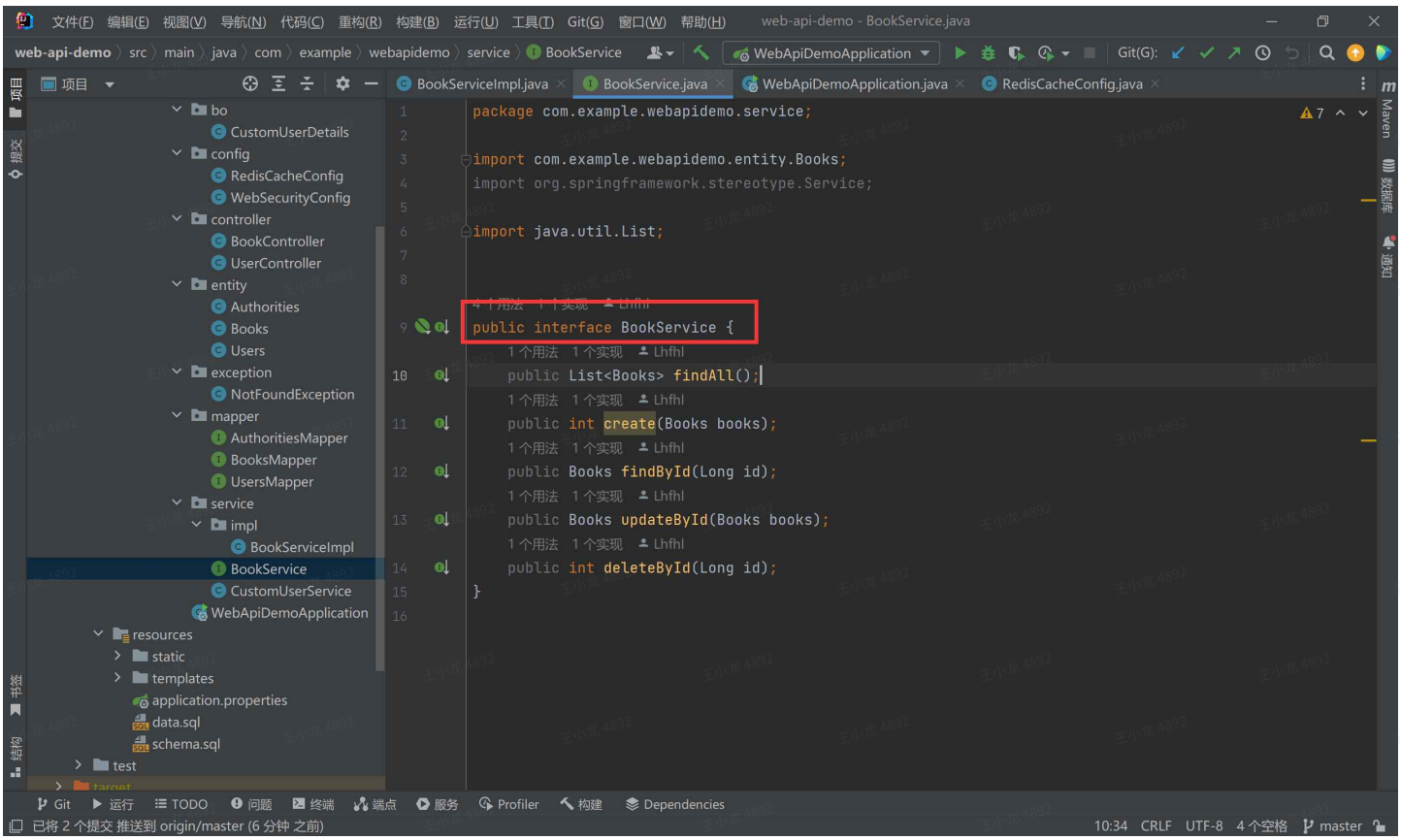
1.在项目启动类加上注解 `@EnableCaching` 开启缓存



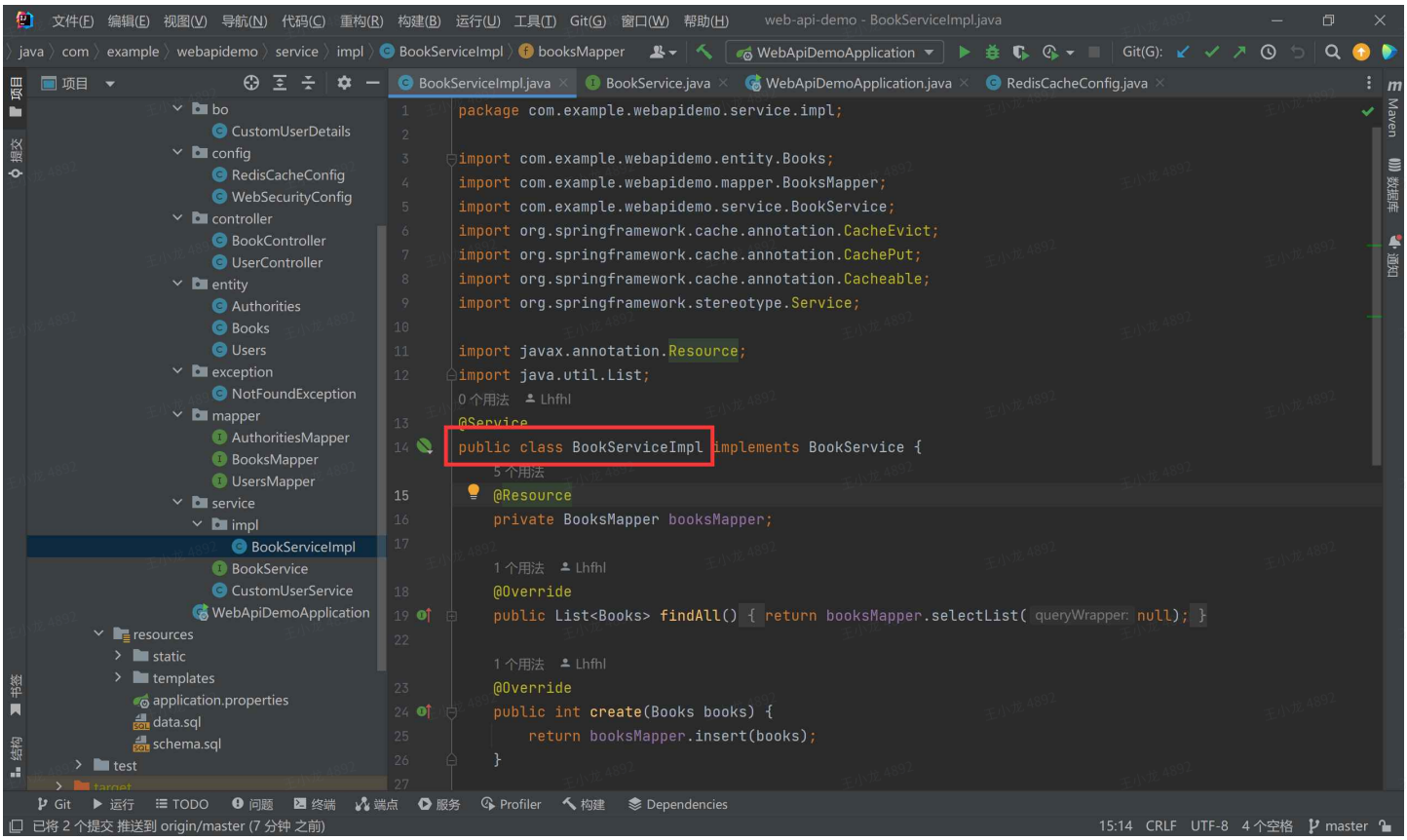
2.添加配置类RedisCacheConfig



3.添加BookService接口



4.添加BookService接口的实现类BookServiceImpl



5.修改BookController