**好优租聘公寓**

**1. 实现完整租房流程**

房源发布：

移动端操作：用户在移动端填写房源相关信息，如房屋位置、户型、面积、租金等详细信息，并上传房屋图片。这些信息通过 HTTP 请求发送到后端服务器。

后端处理：Spring Boot 接收请求，调用相应的 Service 层方法，将房源数据封装成对象，通过 MyBatis - Plus 操作 MySQL 数据库，将房源信息插入到对应的房源表中。同时，调用 MinIO 服务，将用户上传的图片存储到 MinIO 服务器，并将图片在 MinIO 中的存储路径保存到数据库中。

看房预约：

移动端操作：用户在移动端浏览房源列表，选择感兴趣的房源后，点击预约看房按钮，填写预约时间、联系方式等信息，发送预约请求。

后端处理：Spring Boot 接收请求，首先通过 JWT 验证用户身份，确保请求合法。然后，在 Service 层检查预约时间是否与其他预约冲突，以及房源是否存在。如果一切正常，将预约信息插入到预约表中，并通过阿里云 SMS 向房东发送预约通知短信。

租赁合同签署：

移动端操作：用户和房东协商好租赁条款后，在移动端生成电子租赁合同。用户确认合同内容无误后，通过电子签名功能签署合同。

后端处理：Spring Boot 接收签署请求，验证用户身份后，将合同内容保存到数据库中的合同表中，并记录签署时间等信息。同时，系统可以生成合同编号，方便后续管理。

租约管理：

移动端操作：用户和房东可以在移动端查看租约详情，包括租赁期限、租金支付记录、房屋状态等信息。用户还可以进行租金支付等操作。

后端处理：Spring Boot 接收请求，根据用户身份和请求类型，从数据库中查询相应的租约信息返回给移动端。在租金支付方面，系统调用第三方支付接口完成支付流程，并更新数据库中的支付记录和租约状态。

**2. 利用 MySQL 存储核心数据，并使用 Redis 作为缓存层**

MySQL 存储核心数据：

设计数据库表结构，如房源表、用户表、预约表、合同表等，以满足业务需求。每个表根据业务逻辑定义相应的字段和约束。

使用 MyBatis - Plus 框架，通过编写 SQL 语句或使用 MyBatis - Plus 提供的内置方法，实现对数据库的增删改查操作。例如，在房源发布时，使用 insert 方法将房源信息插入到房源表中。

Redis 作为缓存层：

在系统启动时，初始化 Redis 连接池，确保与 Redis 服务器建立连接。

对于一些经常查询且不经常变化的数据，如热门房源列表、地区信息等，在查询 MySQL 数据库后，将结果缓存到 Redis 中。设置合适的缓存过期时间，以保证数据的时效性。

当再次查询相同数据时，首先从 Redis 中获取，如果 Redis 中有数据，则直接返回，避免频繁查询数据库，提高系统响应速度。如果 Redis 中没有数据，则查询 MySQL 数据库，将结果存入 Redis 后再返回。

**3. 引入分库分表策略**

分析业务数据特性：

根据公寓租赁平台不同的业务模块进行分库。例如，将房源相关的业务数据存放在一个库（房源库），用户相关业务数据存放在另一个库（用户库），租赁交易相关业务数据放在第三个库（交易库）。这样不同业务模块的数据相互隔离，每个库专注于自身业务功能的数据处理。

选择分库分表方式：

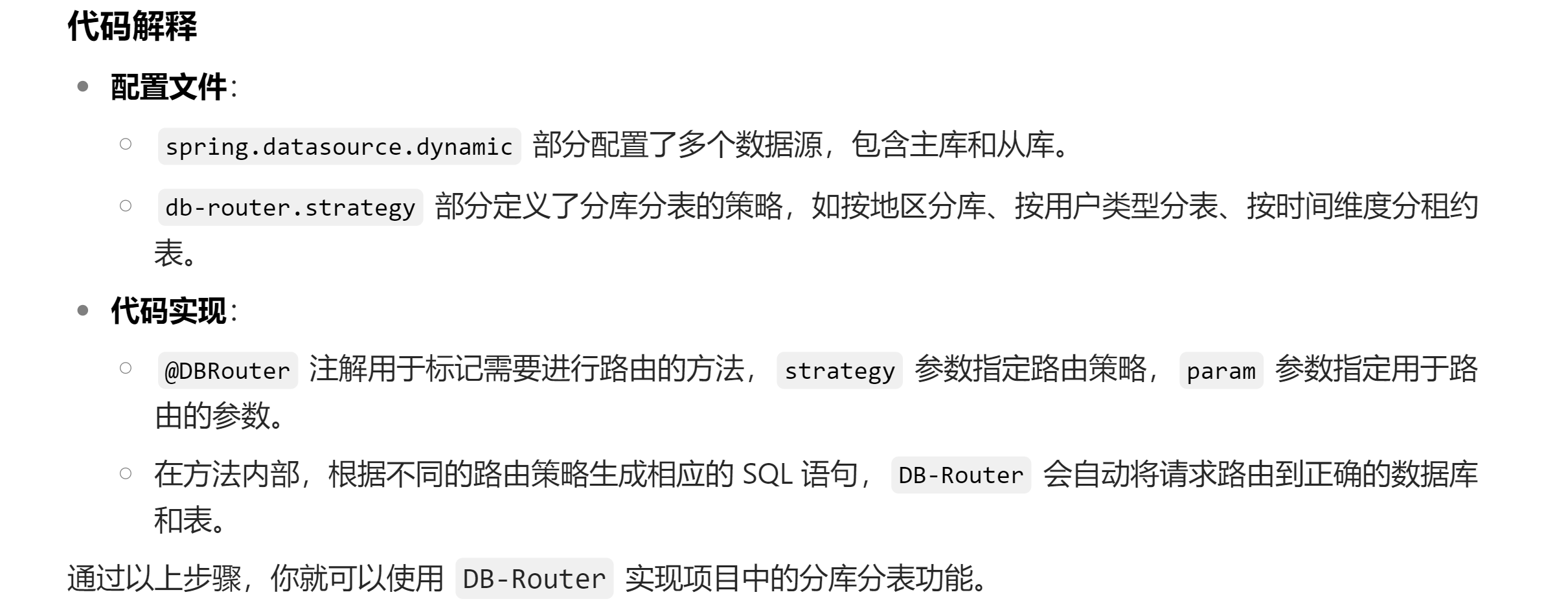
基于业务数据特性，采用按地区进行分库，例如将一线城市的房源数据存储在一个数据库，二线城市的房源数据存储在另一个数据库。

对于用户表，可以按照用户类型进行分表，如个人用户表和企业用户表。同时，对于一些数据量较大的表，如租约表，可以按照时间维度进行分表，例如每个月生成一个新的租约表。

实现分库分表操作：

使用 Sharding – JDBC或者db-rounter 等框架来实现分库分表功能。在配置文件中定义数据源和分库分表规则，例如指定不同地区的数据源和对应的表结构。

在代码中，通过 Sharding - JDBC 提供的 API 进行数据库操作，框架会根据配置的规则自动将请求路由到相应的数据库和表。



**4. 基于阿里云 SMS 和 JWT 搭建用户认证与授权体系**

阿里云 SMS 实现用户注册与找回密码：

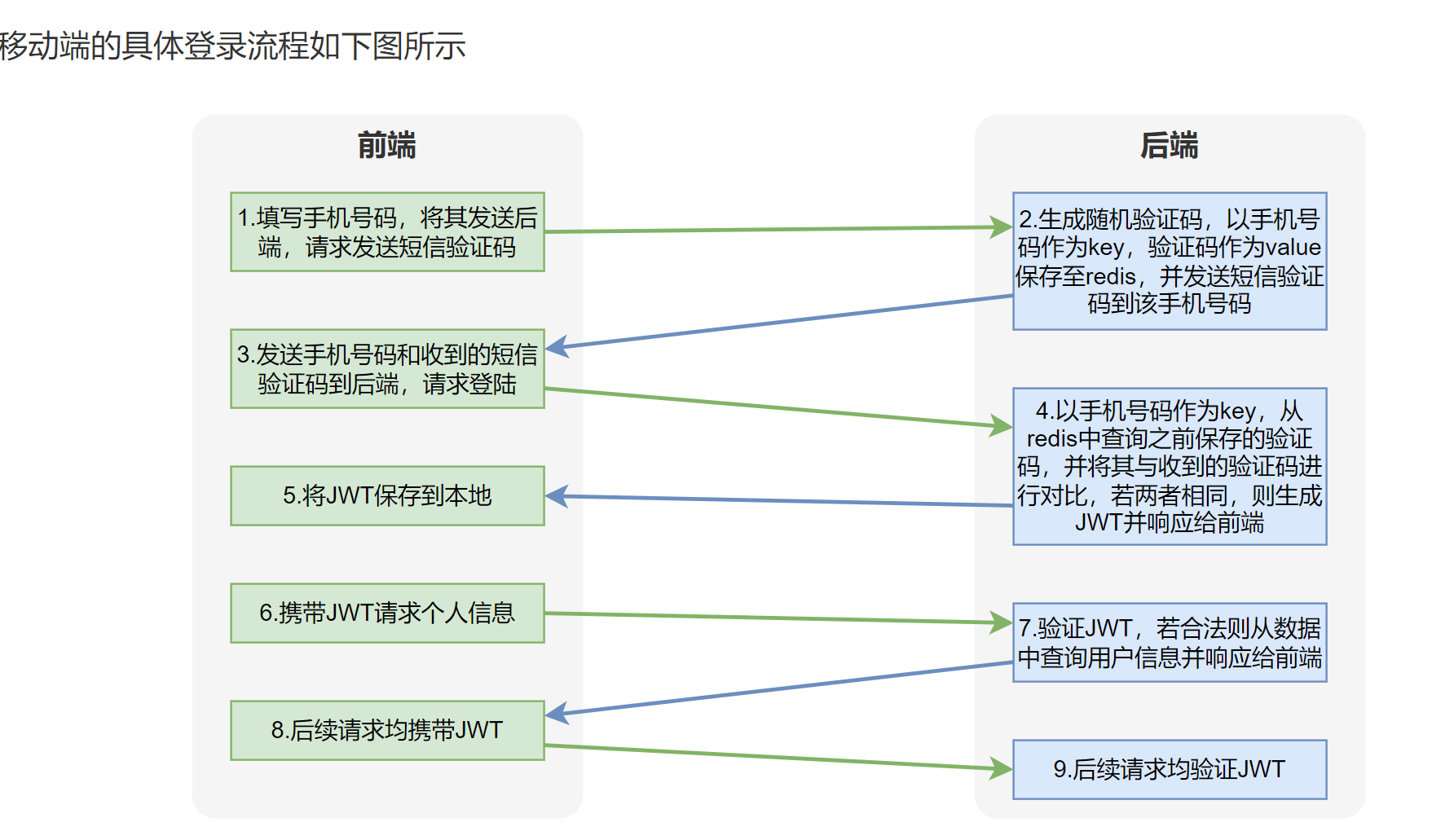
用户注册：用户在移动端填写手机号码，点击获取验证码按钮。后端 Spring Boot 接收到请求后，调用阿里云 SMS 服务，生成随机验证码，并发送到用户手机。同时，将验证码存储到 Redis 中，并设置过期时间。用户在规定时间内输入验证码，后端验证验证码是否正确，若正确则允许用户完成注册流程。

找回密码：用户在忘记密码页面输入手机号码，点击获取验证码。后端同样调用阿里云 SMS 发送验证码，验证通过后，用户可以设置新密码，后端更新数据库中用户的密码信息。

JWT 实现用户认证与授权：

用户登录：用户在移动端输入账号和密码，后端 Spring Boot 接收到登录请求后，查询数据库验证账号密码是否正确。如果正确，生成 JWT Token，Token 中包含用户的基本信息（如用户 ID、用户名、用户角色等）。将 Token 返回给移动端。

后续请求验证：移动端在后续请求中，将 JWT Token 放在请求头中。后端 Spring Boot 接收到请求后，首先通过 JWT 过滤器验证 Token 的合法性。如果 Token 有效，则解析 Token 中包含的用户角色信息，根据角色判断用户是否有权限访问请求的资源。如果权限不足，则返回错误信息，防止非法访问。



**5. 集成 MinIO 实现房源图片存储和管理**

MinIO 环境搭建：

在 Linux 服务器上安装 MinIO 服务，配置 MinIO 的访问地址、端口、用户名和密码等信息。确保 MinIO 服务正常运行。

图片上传：

当用户在移动端上传房源图片时，后端 Spring Boot 接收到图片文件。调用 MinIO Java SDK，将图片上传到 MinIO 服务器指定的存储桶中。在上传过程中，可以设置图片的元数据，如文件名、文件类型等。

上传成功后，MinIO 返回图片在服务器中的存储路径，后端将该路径保存到 MySQL 数据库的房源表中，以便后续查询和展示。

图片访问：

当移动端需要展示房源图片时，后端根据数据库中存储的图片路径，生成 MinIO 图片的访问 URL。将该 URL 返回给移动端，移动端通过该 URL 直接从 MinIO 服务器获取图片并展示。

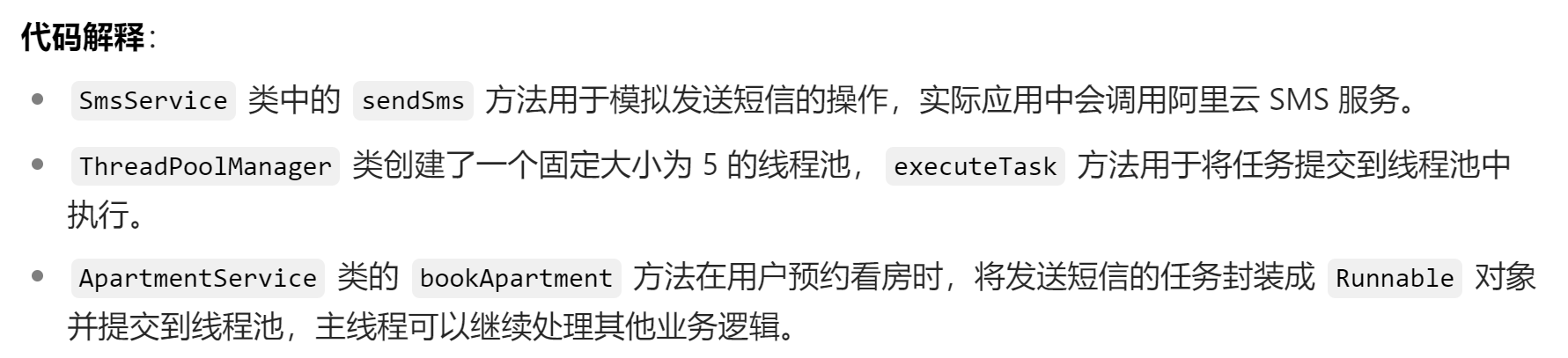
同时，MinIO 支持大文件上传，通过分块上传等技术，确保大尺寸图片能够高效稳定地上传和存储。

在你所负责的公寓租赁平台项目中，线程池技术可以在多个场景发挥重要作用，下面为你详细介绍具体的使用场景和方式：

1. 短信发送任务

使用场景

在项目里，当用户进行看房预约、租赁合同签署等操作时，系统需要通过阿里云 SMS 给房东或者用户发送通知短信。由于短信发送操作可能会涉及网络请求，并且耗时不确定，若在主线程里执行，会阻塞其他业务逻辑，降低系统的响应速度。所以可以使用线程池来异步处理短信发送任务。



2. 图片上传处理

使用场景

用户在移动端上传房源图片时，后端需要将图片存储到 MinIO 服务器。图片上传和存储操作可能会消耗较多的时间和资源，为了避免阻塞主线程，可以使用线程池来异步处理图片上传任务。

大乐透营销系统

1. 设计抽奖模块的领域模型与流程节点实现

抽奖前：人群筛选和黑名单过滤逻辑

人群筛选：依据抽奖活动预先设定的规则，像用户的注册时间、活跃度、所在地区等条件对参与抽奖的人群加以筛选。在代码里，可通过数据库查询或者缓存查询来实现。例如，借助 MyBatis 编写 SQL 语句从 MySQL 数据库里查询符合条件的用户。

黑名单过滤：把处于黑名单中的用户从抽奖人群里排除。可以将黑名单数据存于 Redis 中，在抽奖前快速检查用户是否在黑名单内。

抽奖中：实时奖品库存扣减

在抽奖过程中，当用户中奖时，要立刻扣减奖品的库存。运用 Redis 的原子操作 decr 来实现实时扣减，保证库存数据的一致性。

抽奖后：奖品发放及兜底奖励逻辑

奖品发放：若用户中奖，按照奖品类型进行相应的发放操作，如发放积分、优惠券等。可以借助消息队列（RabbitMQ）异步处理奖品发放，提升系统的响应性能。

2. 抽奖核心算法性能优化

实现 O (1) 时间复杂度

采用哈希表或者数组来存储奖品信息与中奖概率。在抽奖时，借助随机数生成器生成一个随机数，然后依据随机数和奖品概率的映射关系快速确定中奖奖品。

3. 使用责任链模式完成抽奖人群过滤与流程串联

责任链模式实现

定义一系列的处理器（Handler），每个处理器负责一个特定的过滤逻辑，例如、黑名单过滤处理器.权重范围.默认抽奖等。

把这些处理器连接成一个链，请求会依次经过每个处理器进行处理。

4. 高效的奖品库存扣减逻辑设计检查并扣减库存，防止超卖现象的发生。

Redis 分段消费机制结合 decr 操作实现实时扣减

分段设计：将奖品的总库存按照一定规则划分成多个段，每个段对应 Redis 中的一个键。比如，一个奖品总库存为 1000，可均分为 10 段，每段库存 100，每个段在 Redis 里有各自独立的键。

随机选择分段：当有抽奖请求且用户中奖需要扣减库存时，系统会随机挑选一个库存段。随机选择能避免所有请求都集中在同一个库存段上，从而分散 Redis 的并发压力。

decr 操作：使用 Redis 的 decr 操作对选中的库存段进行原子性扣减。decr 操作具有原子性，能保证在高并发场景下库存数据的一致性，即同一时间只有一个请求可以对该库存段进行扣减。若扣减后库存不小于 0，表明扣减成功；若小于 0，则说明该分段库存不足，需重新选择其他分段尝试扣减。

引入加锁兜底策略防止超卖

加锁时机：当 Redis 分段扣减库存失败（所有分段库存都不足）时，会触发加锁兜底策略。

加锁方式：采用 SETNX 命令尝试获取分布式锁。SETNX 会检查锁对应的键是否存在，若不存在则设置该键的值并返回 1，表示成功获取锁；若键已存在则返回 0，意味着锁被其他线程持有。同时，为避免死锁，会给锁设置一个合理的过期时间。

数据库检查与扣减：成功获取锁后，系统会直接访问数据库，再次检查奖品的总库存，并进行扣减操作。由于数据库本身具有事务机制，能保证库存数据的一致性，防止超卖。

释放锁：库存扣减操作完成后，无论成功与否，都会释放锁，使用 DEL 命令删除锁对应的键，以便其他线程可以继续竞争锁。

利用 RabbitMQ 异步处理库存更新

消息发送：当 Redis 分段扣减库存成功后，系统会立即向 RabbitMQ 发送一条消息，消息内容包含奖品 ID、扣减数量等信息。这种异步处理方式可以让系统快速响应抽奖请求，无需等待数据库更新完成。

消息消费：在 RabbitMQ 的消费者端，会有专门的程序监听队列，接收并处理这些消息。消费者接收到消息后，会根据消息内容更新数据库中的库存信息。

定时任务同步至数据库

数据一致性保障：尽管 RabbitMQ 会异步更新数据库库存，但在高并发场景下，可能会存在 Redis 与数据库数据不一致的情况。因此，会设置定时任务，定期将 Redis 中的库存数据同步到数据库中。

同步频率：定时任务的执行频率可根据业务需求和系统负载进行调整。例如，每小时或每天执行一次，确保数据库中的库存数据与 Redis 中的数据最终一致，降低数据库压力。

5. 通过 Zookeeper 监听机制实现活动降级与限流的流程

配置信息存储

将活动的配置信息（如是否开启活动、限流阈值、降级策略等）存储在 Zookeeper 的节点中。这些配置信息可以以 JSON 或 XML 等格式存储，方便应用程序读取和解析。

应用程序监听

在抽奖系统的各个服务节点中，启动 Zookeeper 客户端，连接到 Zookeeper 集群，并监听存储活动配置信息的节点。当应用程序启动时，会首先从 Zookeeper 中读取当前的活动配置信息，并根据这些信息进行初始化。

配置变更通知

当活动需要进行降级或限流时，管理员可以通过 Zookeeper 客户端修改存储在 Zookeeper 中的配置信息。Zookeeper 会检测到节点数据的变化，并将变化通知给所有监听该节点的应用程序。

动态调整系统行为

应用程序接收到 Zookeeper 的通知后，会重新读取配置信息，并根据新的配置信息动态调整系统的行为。例如，如果限流阈值降低，系统会减少处理的请求数量；如果活动需要降级，系统会关闭一些非核心的功能。

结合 Dubbo 远程调用优化通信性能的流程

服务注册

在抽奖系统中，各个服务（如抽奖服务、库存服务、奖品发放服务等）会将自己的服务信息（包括服务名称、地址、端口等）注册到 Dubbo 的服务注册中心（通常使用 Zookeeper 作为注册中心）。这样，其他服务就可以通过服务注册中心发现和调用这些服务。

服务发现

当一个服务需要调用另一个服务时，它会向服务注册中心查询目标服务的信息。服务注册中心会返回可用的服务节点列表，调用方可以根据负载均衡策略选择一个合适的节点进行调用。

远程调用

调用方通过 Dubbo 的远程调用机制，将请求发送到目标服务节点。Dubbo 会自动处理网络通信、序列化和反序列化等细节，使得调用方就像调用本地方法一样简单。

负载均衡和集群容错

Dubbo 提供了多种负载均衡策略（如随机、轮询、最少活跃调用数等），可以根据服务节点的负载情况合理分配请求。同时，Dubbo 还支持集群容错机制，当某个服务节点出现故障时，会自动切换到其他可用的节点，保证服务的高可用性。