中间件技术 实验5

22920212204392 黄勖、22920212204385 胡翼翔、22920212204492 张靖源

1 实验目的

- 确定实验大作业选题以及初步完成项目构思。

2 实验环境

系统：Windows 10

软件：

- VSCode、IDEA
- SpringBoot
- RabbitMQ

3 实验步骤

在电商行业中，秒杀活动因其极高的折扣和限时购买特性吸引大量用户。这种活动通常导致短时间内的高并发请求，给传统的单数据库系统带来巨大压力，常见问题包括系统响应缓慢、事务处理能力不足等。为解决这些问题，本项目采用中间件技术——Sharding-JDBC，实现数据库的分库分表和读写分离，以优化系统性能和数据处理能力，有效应对大规模用户同时参与秒杀活动时产生的高并发数据访问需求。系统将支持商品的展示、秒杀活动的进行、订单的生成和用户管理等核心功能，以提高处理速度、降低系统延迟，并保证数据的一致性和可靠性。

3.1 引言

随着电子商务的迅猛发展，促销活动如“秒杀”已成为吸引客户和增加销售额的重要手段。秒杀活动特点是商品数量有限、折扣深，且购买时间限制严格，这常常在极短的时间内引发大量用户的并发访问。这种高并发场景对电商平台的数据处理能力和系统响应速度提出了极高的要求。

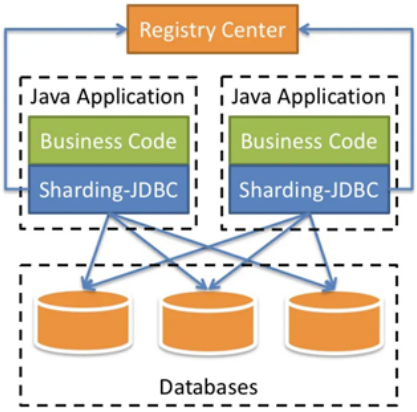
传统的单一数据库系统在处理如此高并发的请求时面临着诸多挑战，包括数据访问延迟、事务阻塞、服务器过载等，这些问题极可能导致系统崩溃，从而影响用户体验和企业信誉。为了克服这些技术障碍，必须采用更为高效的数据管理和处理策略。

本项目《基于Sharding-JDBC的高并发秒杀系统设计与实现》旨在通过实施中间件技术——Sharding-JDBC，来实现数据的有效分片和读写分离，提高数据库处理能力，确保系统在高并发条件下的稳定性和高效性。Sharding-JDBC作为一个轻量级的中间件集成方案，不仅可以减少系统的复杂性，还能提高数据操作的灵活性和效率。

通过本项目的设计与实现，我们将详细探索Sharding-JDBC在真实秒杀场景中的应用效果，验证其在分布式数据库环境下对于提升系统性能、增强数据一致性和可靠性的能力。此外，本项目也将为相关领域的开发者和研究人员提供宝贵的实践经验和技术洞见，助力未来电商系统的技术革新和优化。

3.2 系统概述

本项目开发的“基于Sharding-JDBC的高并发秒杀系统”旨在提供一个稳定、高效的电商秒杀平台，能够处理大量用户在限定时间内对限定商品的购买请求。该系统利用Sharding-JDBC实现数据库的分库分表和读写分离，以优化数据处理速度和系统响应时间，确保在高并发环境下的性能和稳定性。



3.2.1 主要功能

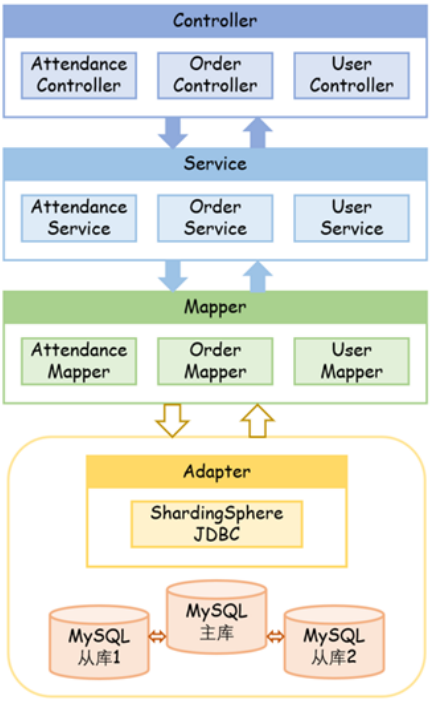
1. **商品管理**：
   - **商品添加**：允许管理员上传新商品信息，包括商品名称、描述、价格、库存数量及秒杀开始和结束时间。
   - **商品浏览**：用户可以浏览可用的秒杀商品列表，查看商品详细信息。
2. **用户管理**：
   - **用户注册与登录**：用户需注册并登录后才能参加秒杀活动。系统提供基本的用户认证功能，包括用户名和密码验证。
   - **用户信息管理**：用户可以查看和更新自己的个人信息。
3. **秒杀功能**：
   - **秒杀参与**：在活动时间内，用户可以对感兴趣的商品发起购买请求。系统需实时处理这些请求，并即时反馈购买结果。
   - **库存管理**：系统需要实时更新库存状态，确保不会超卖。
4. **订单处理**：
   - **订单生成**：成功秒杀后，系统自动为用户生成订单。
   - **订单查询**：用户可以查询自己的订单详情，包括订单状态、商品信息、支付状态等。

3.2.2 系统特点

- **高并发处理能力**：通过Sharding-JDBC的分库分表功能，系统能够在多数据库实例间分散请求压力，提高数据操作的并行度，从而有效应对秒杀场景下的高并发请求。
- **读写分离**：通过配置Sharding-JDBC的读写分离策略，系统可以将查询操作和事务性写操作分别路由到不同的数据库节点，进一步优化性能。
- **数据一致性保证**：尽管数据分布在多个数据库实例中，Sharding-JDBC确保跨分片的数据操作维持一致性，对外提供一致的数据视图。
- **容错与可扩展性**：系统设计考虑了容错机制，如数据库实例宕机的情况下自动切换到备用实例。同时，支持动态扩展数据库实例以应对业务增长。

3.2.3 技术架构

系统采用分层架构设计，主要包括：



- **前端层**：使用HTML, CSS, JavaScript构建用户界面，提供动态交互式的用户体验。
- **业务逻辑层**：后端采用Spring Boot框架，负责处理业务逻辑，如用户认证、商品管理、订单处理等。
- **数据访问层**：通过Sharding-JDBC中间件实现与后端MySQL数据库的交互，包括数据的CRUD操作及事务处理。
- **数据存储层**：使用MySQL作为关系数据库管理系统，存储用户数据、商品数据和订单数据等。

3.3 安装指南

安装指南提供了“基于Sharding-JDBC的高并发秒杀系统”的详细安装步骤，包括软件环境要求、所需依赖和配置过程。

3.3.1 环境要求

为确保系统的正常运行和最佳性能，需要预先准备以下环境：

- **操作系统**：Windows 10或更高版本、Linux (Ubuntu 18.04 或 CentOS 7 或更高版本)
- **Java**：Java JDK 11 或以上
- **数据库**：MySQL 5.7 或更高版本
- **开发工具**：IntelliJ IDEA 或 Eclipse（带Spring Boot插件）
- **其他软件**：Git（用于代码管理和版本控制）

3.3.2 安装步骤

1. **安装Java开发环境**

   - 确保安装Java JDK 11或以上版本。可以从Oracle官网下载并安装。
   - 设置环境变量`JAVA_HOME`，并将Java的bin目录添加到系统的PATH变量。

2. **安装MySQL数据库**

   - 从MySQL官网下载并安装MySQL 5.7或更高版本。
   - 设置root用户密码并记住，后续配置中将使用此密码连接数据库。
   - 开启MySQL服务，并确保服务在系统重启后能自动启动。

3. **配置数据库**

   - 登录MySQL数据库，创建所需的数据库和表。可以使用项目提供的SQL脚本进行数据库和表的创建。

```
CREATE DATABASE seckill;
USE seckill;
-- 运行项目中提供的SQL脚本来创建表和初始化数据
```

4. **安装并配置Sharding-JDBC**

   - 在项目的`pom.xml`文件中添加Sharding-JDBC的依赖。

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>sharding-jdbc-spring-boot-starter</artifactId>
    <version>最新版本号</version>
</dependency>
```

   - 配置`application.yml`文件，设置Sharding-JDBC的数据源和分片规则。

5. **构建和运行项目**

   - 使用Git克隆项目仓库到本地开发环境。
   - 打开IntelliJ IDEA或Eclipse，导入项目。
   - 构建项目：在IDE中运行`mvn clean install`（确保已安装Maven）。
   - 运行项目：在IDE中运行主程序类或使用命令行`java -jar target/项目名.jar`。

6. **验证安装**

   - 访问`http://localhost:8080/`来查看是否能够访问系统界面。
   - 进行几项基本操作，如用户注册、商品浏览等，以确保系统功能正常。

通过以上步骤，应能成功安装并运行“基于Sharding-JDBC的高并发秒杀系统”。若在安装过程中遇到任何问题，请参考附录中的常见问题解答，或联系技术支持获取帮助。

3.4 用户指南

本用户指南旨在帮助用户熟悉并有效使用“基于Sharding-JDBC的高并发秒杀系统”。以下内容将详细介绍系统的主要功能，包括用户注册、登录、商品浏览、参与秒杀和查看订单等操作。

3.4.1 用户注册与登录

1. **用户注册**

   - 访问系统首页，在页面右上角点击“注册”按钮。
   - 在注册表单中填写用户名、密码、电子邮箱等信息。
   - 提交表单后，系统将发送一封验证邮件到邮箱。请访问邮件中的链接以完成账户激活。
   - 账户激活后，即可使用注册的用户名和密码登录系统。

2. **用户登录**

   - 在系统首页点击“登录”按钮。
   - 在登录页面，输入用户名和密码。
   - 点击“登录”按钮，若信息无误，系统将导航至用户主界面。

3.4.2 商品浏览

- 在登录后的用户主界面，将看到当前可参与秒杀的商品列表。
- 每个商品项将显示商品图片、名称、价格和剩余库存。
- 点击商品可以查看其详细信息，包括商品描述、秒杀开始时间和结束时间。

3.4.3 参与秒杀

1. **选择商品**

   - 浏览感兴趣的商品，点击“秒杀”按钮。按钮仅在秒杀时间内可点击。
   - 系统会自动检查商品库存，如果库存不足，将显示“已售罄”状态。

2. **提交订单**

   - 若库存充足，系统将自动创建订单并跳转至订单确认页面。
   - 在此页面确认购买信息并选择支付方式。
   - 完成支付后，系统将显示订单成功页面。

3.4.4 订单查询

- 在用户主界面，点击“我的订单”查看所有订单。
- 查看每个订单的详细状态，包括订单号、商品详情、支付状态和下单时间。
- 若需要帮助或有疑问，可以通过订单页面的联系方式联系客服。

3.4.5 系统操作注意事项

- 请确保在参与秒杀前，账户已登录且账户信息准确无误。
- 秒杀活动通常时间敏感且库存有限，请及时参与以增加成功的机会。
- 订单一旦生成，将无法取消，请在下单前确认好所有信息。

3.5 数据库设计

3.5.1 E-R 分析

在设计“基于Sharding-JDBC的高并发秒杀系统”的数据库之前，我们进行实体-关系(E-R)分析来确定各数据实体间的关系。以下是涉及的主要实体及其关系：

3.5.1.1 实体

1. **用户（User）**
2. **商品（Product）**
3. **订单（Order）**

3.5.1.2 关系

- **用户与订单**：一个用户可以有多个订单，每个订单属于一个用户（一对多关系）。
- **商品与订单**：一个订单对应一个商品，但一个商品可以对应多个订单（一对多关系）。

3.5.1.3 E-R图描述

用户(User) <--> 创建 --> 订单(Order) <--> 包含 --> 商品(Product)

3.5.2 数据库表设计

基于上述E-R分析，我们可以具体设计数据库表格。以下表格详细描述了各表的字段和属性：

3.5.2.1 用户表（`users`）

字段名	数据类型	约束	描述
user_id	INT	PRIMARY KEY, AUTO_INCREMENT	用户唯一标识
username	VARCHAR(255)	UNIQUE	用户名
password	VARCHAR(255)		加密后的密码
email	VARCHAR(255)	UNIQUE	邮箱
created_at	DATETIME		账户创建时间
updated_at	DATETIME		账户最后更新时间

3.5.2.2 商品表（`products`）

字段名	数据类型	约束	描述
product_id	INT	PRIMARY KEY, AUTO_INCREMENT	商品唯一标识
name	VARCHAR(255)		商品名称
description	TEXT		商品描述
price	DECIMAL(10,2)		价格
total_stock	INT		总库存
seckill_stock	INT		秒杀活动库存
start_time	DATETIME		秒杀开始时间
end_time	DATETIME		秒杀结束时间
created_at	DATETIME		商品录入时间
updated_at	DATETIME		最后更新时间

3.5.2.3 订单表（`orders`）

字段名	数据类型	约束	描述
order_id	INT	PRIMARY KEY, AUTO_INCREMENT	订单唯一标识
user_id	INT	FOREIGN KEY	用户ID
product_id	INT	FOREIGN KEY	商品ID
status	VARCHAR(50)		订单状态
quantity	INT		购买数量
total_price	DECIMAL(10,2)		总金额
created_at	DATETIME		订单创建时间
updated_at	DATETIME		订单更新时间

这种设计充分体现了实体之间的关系，并为系统提供了处理高并发秒杀活动所需的数据结构支持。

3.6 性能优化

在“基于Sharding-JDBC的高并发秒杀系统”中，性能优化是确保系统在极端负载下能够稳定运行的关键。本节将介绍针对不同系统层面的优化策略，包括数据库优化、应用层优化、缓存策略以及并发控制技术。

3.6.1 数据库优化

1. **索引优化**：
   - 在频繁查询的字段上（如`user_id`，`product_id`在订单表中）建立索引，加快查询速度。
   - 使用复合索引优化查询效率，例如在商品表中根据`start_time`和`end_time`字段创建索引以快速查询活动商品。
2. **分库分表**：

- 利用Sharding-JDBC进行分库分表，分散数据库负载。按照订单ID或时间进行分片，保证数据分布均匀。
- 配置适当的分片策略以减少跨分片查询和事务，提高执行效率。

3. **读写分离**：

- 配置读写分离，将读操作分流到从服务器，减轻主服务器负载，提高查询性能。

3.6.2 应用层优化

1. **异步处理**：

- 使用异步编程模型处理用户请求，如使用Spring WebFlux或其他响应式编程框架，以非阻塞方式提升系统吞吐量。

2. **代码优化**：

- 优化循环、查询和算法，减少不必要的计算和资源消耗。
- 确保核心处理逻辑尽可能轻量，特别是在处理秒杀核心功能时。

3.6.3 缓存策略

1. **热点数据缓存**：

- 将热门商品的信息、库存和秒杀状态等数据缓存于内存中，如使用Redis。
- 设计高效的缓存失效和更新策略，确保数据一致性。

2. **请求合并**：

- 对于秒杀开始时的高并发请求，使用请求合并技术，减少对后端服务和数据库的压力。

3.6.4 并发控制

1. **限流策略**：

- 实现API限流，如使用令牌桶或漏桶算法，防止流量突增导致服务不可用。

2. **队列机制**：

- 使用消息队列管理秒杀请求，如Kafka或RabbitMQ，保证请求的有序处理，并能够应对流量高峰。

3. **分布式锁**：

- 在处理关键操作，如库存扣减时，使用分布式锁确保操作的原子性和一致性。

通过上述多层面的性能优化策略，可显著提高系统的响应速度和处理能力，确保在高并发场景下的稳定性和效率。这些优化措施将使“基于Sharding-JDBC的高并发秒杀系统”能够更好地服务于用户，提供顺畅和公平的秒杀体验。

3.7 测试与评估

为确保“基于Sharding-JDBC的高并发秒杀系统”在实际部署前能够达到预期的性能标准和功能要求，进行全面的测试和评估是必不可少的。本节将详细介绍针对系统进行的各类测试方法及评估标准。

3.7.1 测试类型

1. **单元测试**：
   - 针对系统中的每个独立模块进行测试，确保模块内部逻辑正确性。
   - 使用JUnit或TestNG等测试框架来实现自动化测试。
2. **集成测试**：
   - 验证各个模块之间的接口和数据交互是否符合设计。
   - 使用Mockito或Spring Boot Test等工具模拟外部服务，确保模块间集成正确无误。
3. **压力测试**：
   - 模拟高并发访问，确保系统在极端负载下的响应速度和稳定性。
   - 使用工具如JMeter或LoadRunner来生成大量请求，测试系统的极限性能。
4. **性能测试**：
   - 评估系统处理请求的速度、吞吐量及资源消耗等指标。
   - 关注数据库操作的响应时间和事务处理速度。
5. **安全测试**：
   - 检测系统对于潜在的安全威胁（如SQL注入、跨站请求伪造等）的防御能力。
   - 使用OWASP ZAP、Burp Suite等安全测试工具。
6. **用户接受测试** (UAT)：
   - 邀请目标用户参与，确保系统功能满足用户需求，并且用户界面友好。
   - 收集用户反馈，优化系统功能和操作流程。

3.7.2 评估标准

- **功能完整性**：系统功能是否全面实现了设计要求。
- **系统稳定性**：系统是否能在高并发情况下持续稳定运行。
- **性能指标**：系统的响应时间、并发处理能力和资源使用效率是否达标。
- **用户满意度**：最终用户对系统操作界面和功能的满意程度。
- **安全性**：系统是否有足够的安全保护措施，以防止数据泄露和其他安全风险。

3.7.3 测试结果与反馈

- **结果分析**：对测试结果进行详细分析，标识出性能瓶颈和功能缺陷。
- **问题修正**：针对发现的问题制定改进措施，并进行相应的优化。
- **回归测试**：对进行修改或优化后的系统再次执行测试，确保问题已被解决，且没有引入新的问题。

通过上述全面的测试与评估，可以确保“基于Sharding-JDBC的高并发秒杀系统”在投入实际运营前，已达到既定的质量和性能标准，具备处理真实环境下高并发请求的能力。这不仅提升了系统的可靠性，也增强了用户的信任和满意度。

3.8 问题处理与维护

确保“基于Sharding-JDBC的高并发秒杀系统”在长期运营中保持高性能和稳定性，问题处理与维护是必不可少的。本节将介绍系统的问题处理流程、定期维护策略和持续改进措施。

3.8.1 问题处理流程

1. **问题识别**：
    - 监控系统运行状态，使用工具如Prometheus和Grafana实时跟踪系统性能指标。
    - 用户和操作员可通过报告工具提交系统故障或性能下降的问题。
2. **问题分类与分派**：
    - 根据问题的紧急程度和类型进行分类（如系统崩溃、性能问题、功能故障等）。
    - 分派问题到相应的技术团队或个人进行处理。
3. **问题调查与分析**：
    - 通过日志分析、数据库查询和代码审查等方式，确定问题的根本原因。
    - 如果问题复杂，可能需要团队协作来确定解决方案。
4. **问题修复**：
    - 开发合适的修复方案，如代码修改、配置调整或硬件升级。
    - 在测试环境中验证修复方案的有效性。
5. **发布与监控**：
    - 将修复方案部署到生产环境，并严密监控其效果。
    - 确保修复措施不会引发新的问题。
6. **问题记录与回顾**：
    - 记录每个问题的详细信息、处理过程和结果，用于未来的问题预防和培训。
    - 定期回顾问题处理流程和解决方案，评估其效率和效果，进行必要的流程优化。

3.8.2 定期维护策略

1. **系统更新与升级**：

   - 定期检查并更新系统所依赖的软件库和框架，确保安全性和兼容性。
   - 按计划升级系统硬件和网络设施，以提升性能和扩展性。

2. **性能优化**：

   - 定期进行性能评估，根据评估结果优化数据库、应用服务器和前端性能。
   - 优化缓存策略和数据存储方案，以应对数据增长和访问量增加。

3. **数据备份与恢复**：

   - 实施定期数据备份计划，包括全备份和增量备份。
   - 测试数据恢复流程，确保在数据丢失或损坏时能迅速恢复系统运行。

3.8.3 持续改进措施

1. **技术监控与创新**：

   - 持续关注新技术和行业趋势，评估其在系统中的应用潜力。
   - 鼓励技术创新和实验，以持续提升系统的技术竞争力。

2. **用户反馈与参与**：

   - 建立有效的用户反馈机制，定期收集和分析用户的使用体验和改进建议。
   - 鼓励用户参与新功能的测试和评估，确保系统的功能满足用户需求。

通过以上的问题处理与维护策略，可以有效地预防和解决系统中的问题，保持系统的长期稳定和高效运行。这些措施将加强系统的可靠性，提高用户满意度，并为系统的持续改进和优化提供支持。

4 实验总结

通过本次实验，我加深了对中间件的理解。

5 实验分工

排序如下：

1 黄勖

2 胡翼翔、张靖源