中间件技术 实验6

22920212204392 黄勖、22920212204385 胡翼翔、22920212204492 张靖源

1 实验目的

- 05：确定实验大作业选题以及初步完成项目构思。
- 06：完成了项目的脚手架，提供了秒杀系统的后端接口，以便下一步部署具体中间件

2 实验环境

系统：Windows 10

软件：

- VSCode、IDEA
- SpringBoot

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。

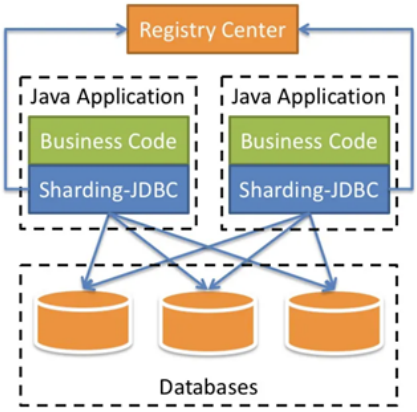
传统的单一数据库系统在处理如此高并发的请求时面临着诸多挑战，包括数据访问延迟、事务阻塞、服务器过载等，这些问题极可能导致系统崩溃，从而影响用户体验和企业信誉。为了克服这些技术障碍，必须采用更为高效的数据管理和处理策略。

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

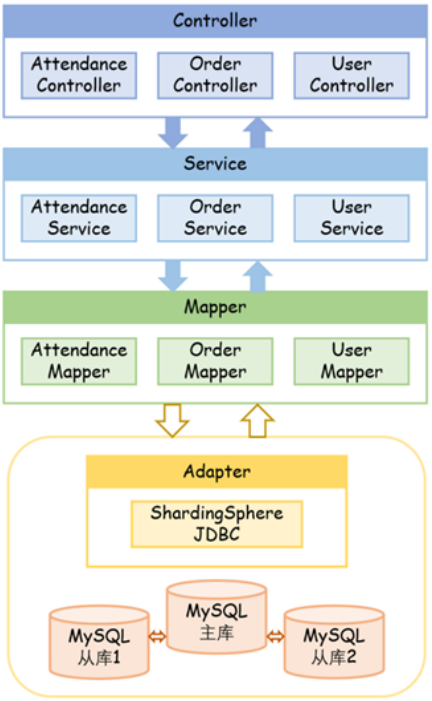
1. **商品管理**：
   - **商品添加**：允许管理员上传新商品信息，包括商品名称、描述、价格、库存数量及秒杀开始和结束时间。
   - **商品浏览**：用户可以浏览可用的秒杀商品列表，查看商品详细信息。
2. **用户管理**：
   - **用户注册与登录**：用户需注册并登录后才能参加秒杀活动。系统提供基本的用户认证功能，包括用户名和密码验证。
   - **用户信息管理**：用户可以查看和更新自己的个人信息。
3. **秒杀功能**：
   - **秒杀参与**：在活动时间内，用户可以对感兴趣的商品发起购买请求。系统需实时处理这些请求，并即时反馈购买结果。
   - **库存管理**：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

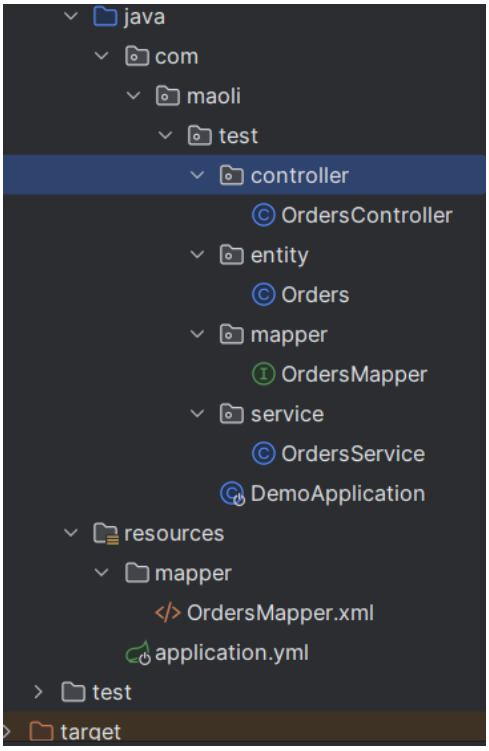
系统采用分层架构设计，主要包括：



- **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 具体编写

项目的结构如下，采用了mvc架构：



Controller:

```
package com.maoli.test.controller;


import com.maoli.test.entity.Orders;
import com.maoli.test.service.OrdersService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/order")
public class OrdersController {
    @Autowired
    private OrdersService ordersService;

    @GetMapping("/{id}")
    public Orders findOrderById(@PathVariable Integer id){
        return ordersService.findOrderById(id);
    }

    @PostMapping
    public void insert(@RequestBody Orders order){
        ordersService.insert(order);
    }

}
```

Model:

```java
package com.maoli.test.entity;

public class Orders {
    private Integer id;
    private String customer;
    private Integer price;

    public Integer getId() {
        return id;
    }

    public Orders(Integer id, String customer, Integer price) {
        this.id = id;
        this.customer = customer;
        this.price = price;
    }

    public void setId(Integer id) {
        this.id = id;
    }


    public String getCustomer() {
        return customer;
    }

    public void setCustomer(String customer) {
        this.customer = customer;
    }

    public Integer getPrice() {
        return price;
    }

    public void setPrice(Integer price) {
        this.price = price;
    }
}
```

Service:

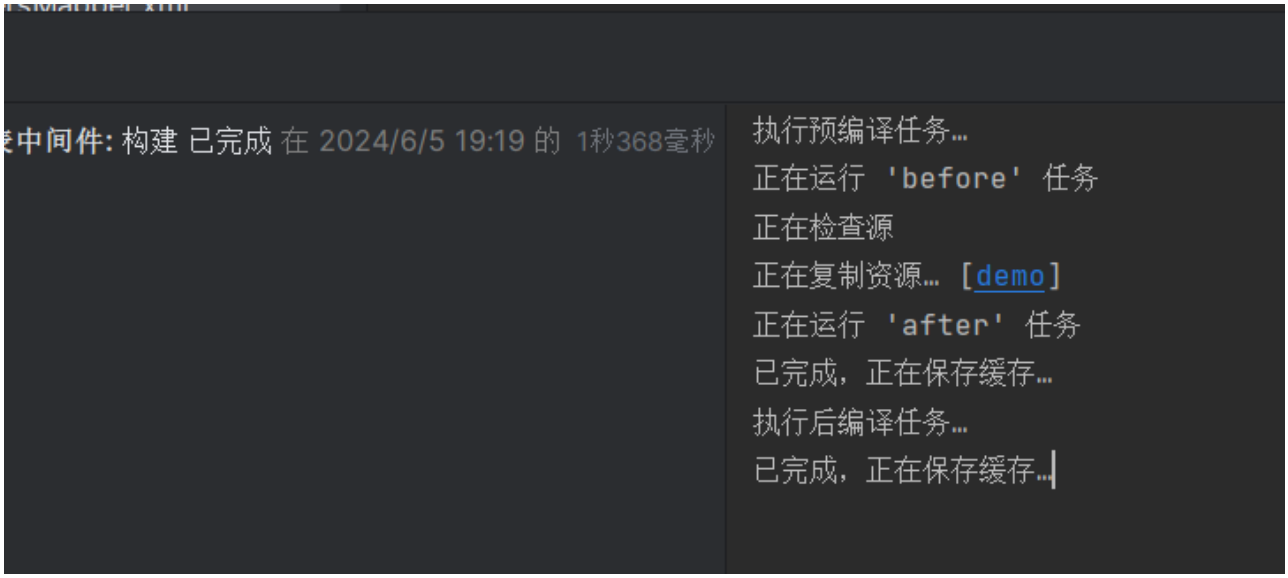
```java
package com.maoli.test.service;


import com.maoli.test.entity.Orders;
import com.maoli.test.mapper.OrdersMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```java
@Service
public class OrdersService {
    @Autowired
    private OrdersMapper ordersMapper;

    public Orders findOrderById(Integer id){
        return ordersMapper.findById(id);
    }

    public void insert(Orders order){
        ordersMapper.insert(order);
    }
}
```

Mapper:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.maoli.test.mapper.OrdersMapper">

    <resultMap id="ordersMap" type="com.maoli.test.entity.Orders">
        <id column="id" property="id"></id>
        <result column="customer" property="customer"></result>
        <result column="price" property="price"></result>
    </resultMap>

    <insert id="insert" parameterType="com.maoli.test.entity.Orders">
        insert into orders values(#{id},#{customer},#{price})
    </insert>

    <select id="findById" resultType="com.maoli.test.entity.Orders"
parameterType="Integer">
        select * from orders where id = #{id}
    </select>

</mapper>
```

编写运行后，可以针对sql进行插入查询操作。



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