StoreCloud电商平台配置文档

一、中间件配置

1. nacos

1.1 dockercompose 配置

```
nacos:
    image: nacos/nacos-server:v2.1.0-slim
    container_name: nacos
    env_file:
      - ./nacos/custom.env
```

```
    ports:
      - "8848:8848"
      - "9848:9848"
      - "9849:9849"
    restart: always
    networks:
      - sc-net
```

1.2 gateway 配置

```
spring:
  application:
    name: gateway
  cloud:
    nacos:
      server-addr: 47.100.81.138:8848
    gateway:
      routes:
        - id: item # 路由规则id，自定义，唯一
          uri: lb://item-service # 路由的目标服务，lb代表负载均衡，会从注册中心拉取服务列表
          predicates: # 路由断言，判断当前请求是否符合当前规则，符合则路由到目标服务
            - Path=/items/**,/search/** # 这里是以请求路径作为判断规则
        - id: cart
          uri: lb://cart-service
          predicates:
            - Path=/carts/**
        - id: user
          uri: lb://user-service
          predicates:
            - Path=/users/**,/addresses/**
        - id: trade
          uri: lb://trade-service
          predicates:
            - Path=/orders/**
        - id: pay
          uri: lb://pay-service
          predicates:
            - Path=/pay-orders/**
```

2. mysql

2.1 dockercompose 配置

```yaml
mysql:
    image: mysql:latest
    container_name: mysql
    ports:
      - "3306:3306"
    environment:
      TZ: "Asia/Shanghai"
      MYSQL_ROOT_PASSWORD: "123"
    volumes:
      - /root/mysql/data:/var/lib/mysql
      - /root/mysql/conf:/etc/mysql/conf.d
      - /root/mysql/init:/docker-entrypoint-initdb.d
    networks:
      - sc-net
```

2.2 共享配置

```yaml
spring:
  datasource:
    url:
jdbc:mysql://${hm.db.host:192.168.150.101}:${hm.db.port:3306}/${hm.db.database}
?useUnicode=true&characterEncoding=UTF-
8&autoReconnect=true&serverTimezone=Asia/Shanghai
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: ${hm.db.un:root}
    password: ${hm.db.pw:123}
mybatis-plus:
  configuration:
    default-enum-type-handler:
com.baomidou.mybatisplus.core.handlers.MybatisEnumTypeHandler
  global-config:
    db-config:
      update-strategy: not_null
      id-type: auto
```

3. seata

3.1 dockercompose配置

```yaml
  seata:
    image: seataio/seata-server:1.5.2
    container_name: seata
    ports:
```

```
      - "8099:8099"
      - "7099:7099"
    environment:
      SEATA_IP: "192.168.150.101"
    volumes:
      - ./seata:/seata-server/resources
    privileged: true
    networks:
      - sc-net
```

3.2 共享配置

```
seata:
  registry: # TC服务注册中心的配置，微服务根据这些信息去注册中心获取tc服务地址
    type: nacos # 注册中心类型 nacos
    nacos:
      server-addr: 192.168.150.101:8848 # nacos地址
      namespace: "" # namespace，默认为空
      group: DEFAULT_GROUP # 分组，默认是DEFAULT_GROUP
      application: seata-server # seata服务名称
      username: nacos
      password: nacos
  tx-service-group: hmall # 事务组名称
  service:
    vgroup-mapping: # 事务组与tc集群的映射关系
      hmall: "default"
```

4. mq

4.1 dockercompose配置

```
mq:
    image: rabbitmq:3.8-management
    container_name: mq
    hostname: mq
    ports:
      - "15672:15672"
      - "5672:5672"
    environment:
      RABBITMQ_DEFAULT_USER: "storecloud"
      RABBITMQ_DEFAULT_PASS: "123321"
    volumes:
      - mq-plugins:/plugins
```

```
    networks:
      - sc-net
```

二、API文档

1. User Service 用户微服务

1.1 地址管理接口 (/addresses)

1.1.1 根据ID查询地址

请求方式: GET

路径: /addresses/{addressId}

参数:

- addressId (path参数): 地址ID

功能: 查询指定ID的地址信息

返回: AddressDTO对象

响应示例:

```
{
    "city": "",
    "contact": "",
    "id": 0,
    "isDefault": 0,
    "mobile": "",
    "notes": "",
    "province": "",
    "street": "",
    "town": ""
}
```

1.1.2 查询当前用户地址列表

请求方式: GET

路径: /addresses

参数: 无

功能: 查询当前登录用户的所有地址

返回: List<AddressDTO>

响应示例:

```
[
    {
        "city": "",
        "contact": "",
        "id": 0,
        "isDefault": 0,
        "mobile": "",
        "notes": "",
        "province": "",
        "street": "",
        "town": ""
    }
]
```

1.2 用户管理接口 (/users)

1.2.1 用户登录

请求方式: POST

路径: /users/login

参数:

- loginFormDTO (请求体): 登录表单数据

功能: 用户登录认证

返回: UserLoginVO对象

响应示例:

```
{
    "balance": 0,
    "token": "",
    "userId": 0,
    "username": ""
}
```

1.2.2 扣减用户余额

请求方式: PUT

路径: /users/money/deduct

参数:

- pw (query参数): 支付密码
- amount (query参数): 支付金额

功能: 扣减用户账户余额

返回: void

响应示例：无

2. Item Service 商品微服务

2.1 商品管理接口 (/items)

2.1.1 分页查询商品

请求方式: GET

路径: /items/page

参数:

- ids (query参数): 商品ID列表

功能: 分页查询商品列表

返回: PageDTO<ItemDTO>

响应示例:

```
{
  "brand": "",
  "category": "",
  "commentCount": 0,
  "id": 0,
  "image": "",
  "isAD": true,
  "name": "",
  "price": 0,
  "sold": 0,
  "spec": "",
  "status": 0,
  "stock": 0
}
```

2.1.2 批量查询商品

请求方式: GET

路径: /items

参数:

- ids (query参数): 商品ID列表

功能: 根据ID批量查询商品信息

返回: List<ItemDTO>

响应示例:

```
{
    "list": [
        {
            "brand": "",
            "category": "",
            "commentCount": 0,
            "id": 0,
            "image": "",
            "isAD": true,
            "name": "",
            "price": 0,
            "sold": 0,
            "spec": "",
            "status": 0,
            "stock": 0
        }
    ],
    "pages": 0,
    "total": 0
}
```

2.1.3 查询单个商品

请求方式: GET

路径: /items/{id}

参数:

- id (path参数): 商品ID

功能: 查询指定ID的商品详情

返回: ItemDTO

响应示例:

```
{
```

```
    "brand": "",
    "category": "",
    "commentCount": 0,
    "id": 0,
    "image": "",
    "isAD": true,
    "name": "",
    "price": 0,
    "sold": 0,
    "spec": "",
    "status": 0,
    "stock": 0
}
```


2.1.4 新增商品

请求方式: POST

路径: `/items`

参数:

- item (请求体): 商品信息DTO

功能: 新增商品

返回: void

2.1.5 更新商品状态

请求方式: PUT

路径: `/items/status/{id}/{status}`

参数:

- id (path参数): 商品ID

status (path参数): 商品状态

功能: 更新商品状态

返回: void

响应示例：无响应

2.1.6 更新商品信息

请求方式: PUT

路径: `/items`

参数:

- item (请求体): 商品信息DTO

功能: 更新商品信息（不包含状态）

返回: void

响应示例：无响应

2.1.7 删除商品

请求方式: DELETE

路径: `/items/{id}`

参数:

- id (path参数): 商品ID

功能: 删除指定商品

返回: void

响应示例：无响应

2.1.8 批量扣减库存

请求方式: PUT

路径: `/items/stock/deduct`

参数:

- items (请求体): List<OrderDetailDTO订单商品详情列表

功能: 批量扣减商品库存

返回: void

响应示例：无响应

2.2 搜索接口 (`/search`)

2.2.1 搜索商品

请求方式: GET

路径: `/search/list`

参数:

- query (query参数): ItemPageQuery查询参数
- key: 关键字
- brand: 品牌
- category: 分类
- minPrice: 最低价格
- maxPrice: 最高价格

功能: 根据条件搜索商品

返回: PageDTO<ItemDTO>

响应示例：


```
{
    "list": [
        {
            "brand": "",
            "category": "",
            "commentCount": 0,
            "id": 0,
            "image": "",
            "isAD": true,
            "name": "",
            "price": 0,
            "sold": 0,
            "spec": "",
            "status": 0,
            "stock": 0
        }
    ],
    "pages": 0,
    "total": 0
}
```


3. Cart Service 购物车微服务

3.1 购物车管理接口 (/carts)

3.1.1 添加商品到购物车

请求方式: POST

路径: /carts

参数:

- cartFormDTO (请求体): 购物车表单数据，需要经过验证

功能: 将商品添加到购物车

返回: void

响应示例：无

3.1.2 更新购物车数据

请求方式: PUT

路径: /carts

参数:

- cart (请求体): 购物车数据

功能: 更新购物车中的商品信息

返回: void

响应示例：无

3.1.3 删除购物车商品

请求方式: DELETE

路径: /carts/{id}

参数:

- id (path参数): 购物车条目ID

功能: 删除购物车中的单个商品

返回: void

3.1.4 查询购物车列表

请求方式: GET

路径: /carts

参数: 无

功能: 查询当前用户的购物车列表

返回: List<CartVO>

响应示例：


```
[
    {
        "createTime": "",
        "id": 0,
        "image": "",
        "itemId": 0,
        "name": "",
        "newPrice": 0,
        "num": 0,
        "price": 0,
        "spec": "",
        "status": 0,
        "stock": 0
    }
```

```
]
```

3.1.5 批量删除购物车商品

请求方式: DELETE

路径: /carts

参数:

- ids (query参数): 购物车条目ID集合

功能: 批量删除购物车中的商品

返回: void

响应示例：无

4. Pay Service 支付微服务

4.1 支付管理接口 (/pay-orders)

4.1.1 查询支付单列表

请求方式: GET

路径: /pay-orders

参数: 无

功能: 查询支付单列表

返回: List<PayOrderVO>

```
[
    {
        "amount": 0,
        "bizOrderNo": 0,
        "bizUserId": 0,
        "createTime": "",
        "expandJson": "",
        "id": 0,
        "payChannelCode": "",
        "payOrderNo": 0,
        "payOverTime": "",
        "paySuccessTime": "",
        "payType": 0,
        "qrCodeUrl": "",
        "resultCode": "",
        "resultMsg": "",
        "status": 0,
```

```
        "updateTime": ""
    }
]
```

4.1.2 生成支付单

请求方式: POST

路径: /pay-orders

参数:

- applyDTO (请求体): 支付申请数据

功能: 生成新的支付单

返回: String (支付单号)

响应示例：

```
{
  "amount": 0,
  "bizOrderNo": 0,
  "orderInfo": "",
  "payChannelCode": "",
  "payType": 0
}
```

4.1.3 余额支付

请求方式: POST

路径: /pay-orders/{id}

参数:

- id (path参数): 支付单ID
- payOrderFormDTO (请求体): 支付表单数据

功能: 使用用户余额支付指定支付单

返回: void

响应示例：无

5. Trade Service 交易微服务

5.1 订单管理接口 (/orders)

5.1.1 查询订单详情

请求方式: GET

路径: `/orders/{id}`

参数:

- orderId (path参数): 订单ID

功能: 根据ID查询订单详细信息

返回: OrderVO

响应示例：


```
{
    "closeTime": "",
    "commentTime": "",
    "consignTime": "",
    "createTime": "",
    "endTime": "",
    "id": 0,
    "payTime": "",
    "paymentType": 0,
    "status": 0,
    "totalFee": 0,
    "userId": 0
}
```


5.1.2 创建订单

请求方式: POST

路径: `/orders`

参数:

- orderFormDTO (请求体): 订单创建表单数据

功能: 创建新订单

返回: Long (订单ID)

响应示例：无

5.1.3 标记订单支付成功

请求方式: PUT

路径: `/orders/{orderId}`

参数:

- orderId (path参数): 订单ID

功能: 将指定订单标记为已支付状态

返回: void

响应示例：无

6. OpenFeign 客户端API

6.1 CartClient (cart-service)

6.1.1 批量删除购物车商品

接口: deleteCartItemByIds

路径: /carts

方法: DELETE

参数:

- ids (RequestParam): Collection<Long购物车条目ID集合

功能: 批量删除购物车中的商品

返回: void

响应示例：无

6.2 ItemClient (item-service)

6.2.1 批量查询商品

接口: queryItemByIds

路径: /items

方法: GET

参数:

- ids (RequestParam): Collection<Long> 商品ID集合

功能: 批量查询商品信息

返回: List<ItemDTO>

响应示例:


```
{
    "list": [
        {
            "brand": "",
            "category": "",
            "commentCount": 0,
            "id": 0,
```

```
            "image": "",
            "isAD": true,
            "name": "",
            "price": 0,
            "sold": 0,
            "spec": "",
            "status": 0,
            "stock": 0
        }
    ],
    "pages": 0,
    "total": 0
}
```

6.2.2 扣减库存

接口: deductStock

路径: /items/stock/deduct

方法: PUT

参数:

- items (RequestBody): List<OrderDetailDTO> 订单商品详情列表

功能: 批量扣减商品库存

返回: void

特性: 具有服务降级功能（ItemClientFallbackFactory）

响应示例：无

6.3 TradeClient (trade-service)

6.3.1 标记订单支付成功

接口: markOrderPaySuccess

路径: /orders/{orderId}

方法: PUT

参数:

- orderId (PathVariable): Long 订单ID

功能: 将指定订单标记为支付成功状态

返回: void

响应示例：无

6.4 UserClient (user-service)

6.4.1 扣减用户余额

接口: deductMoney

路径: /users/money/deduct

方法: PUT

参数:

- pw (RequestParam): String 支付密码
- amount (RequestParam): Integer 扣减金额

功能: 扣减用户账户余额

返回: void

三、主要maven依赖

```
<dependencyManagement>
    <dependencies>
        <!--spring cloud-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
        <!--spring cloud alibaba-->
        <dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-alibaba-dependencies</artifactId>
            <version>${spring-cloud-alibaba.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
        <!-- 数据库驱动包管理 -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
```

```
            <version>${mysql.version}</version>
        </dependency>
        <!-- mybatis plus 管理 -->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>${mybatis-plus.version}</version>
        </dependency>
        <!--hutool工具包-->
        <dependency>
            <groupId>cn.hutool</groupId>
            <artifactId>hutool-all</artifactId>
            <version>${hutool.version}</version>
        </dependency>
    </dependencies>
</dependencyManagement>
<dependencies>
    <!-- lombok 管理 -->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>${org.projectlombok.version}</version>
    </dependency>
    <!--单元测试-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

这些 Maven 依赖主要用于管理 Spring Cloud、Spring Cloud Alibaba、数据库连接、MyBatis Plus、Hutool 工具包、Lombok 和单元测试等相关功能。具体解读如下：

1. spring-cloud-dependencies

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-dependencies</artifactId>
    <version>${spring-cloud.version}</version>
    <type>pom</type>
    <scope>import</scope>
</dependency>
```

- **作用**：这是 Spring Cloud 依赖的 BOM（Bill Of Materials），它为项目中使用的 Spring Cloud 相关组件提供统一的版本控制。通过引入此依赖，可以简化 Spring Cloud 相关库的版本管理，避免每个组件的版本冲突。
- **配置项**：`spring-cloud.version` 是项目中定义的变量，表示 Spring Cloud 版本号。

2. spring-cloud-alibaba-dependencies

```xml
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-alibaba-dependencies</artifactId>
    <version>${spring-cloud-alibaba.version}</version>
    <type>pom</type>
    <scope>import</scope>
</dependency>
```

- **作用**：这是 Spring Cloud Alibaba 相关依赖的 BOM，用于管理 Spring Cloud Alibaba 系列工具包和框架的版本。Spring Cloud Alibaba 提供了一些用于分布式系统的解决方案，例如 Nacos、RocketMQ 等。
- **配置项**：`spring-cloud-alibaba.version` 是项目中定义的版本变量，表示 Spring Cloud Alibaba 版本号。

3. mysql-connector-java

```xml
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>${mysql.version}</version>
</dependency>
```

- **作用**：这是 MySQL 数据库的 JDBC 驱动，用于连接和操作 MySQL 数据库。
- **配置项**：`mysql.version` 是项目中定义的版本变量，表示 MySQL Connector/J 驱动的版本号。

4. mybatis-plus-boot-starter

```xml
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>${mybatis-plus.version}</version>
</dependency>
```

- **作用**：MyBatis-Plus 是 MyBatis 的增强工具包，提供了简化 SQL 操作的功能，例如自动生成 SQL、分页查询、批量操作等。这是 MyBatis-Plus 的 Spring Boot 启动器。
- **配置项**：`mybatis-plus.version` 是项目中定义的版本变量，表示 MyBatis-Plus 的版本号。

5. hutool-all

```
<dependency>
    <groupId>cn.hutool</groupId>
    <artifactId>hutool-all</artifactId>
    <version>${hutool.version}</version>
</dependency>
```

- **作用**：Hutool 是一个实用的 Java 工具库，提供了大量常用工具类，涵盖了日期、文件、加密、网络、JSON 等多种功能。
- **配置项**：`hutool.version` 是项目中定义的版本变量，表示 Hutool 工具库的版本号。

6. lombok

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>${org.projectlombok.version}</version>
</dependency>
```

- **作用**：Lombok 是一个 Java 编译时注解处理工具，可以减少 Java 代码中的样板代码（如 getter、setter、toString 等）。通过注解自动生成这些常见方法，简化代码编写。
- **配置项**：`org.projectlombok.version` 是项目中定义的版本变量，表示 Lombok 的版本号。

7. spring-boot-starter-test

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

- **作用**：这是 Spring Boot 的测试启动器，提供了单元测试相关的依赖，例如 JUnit、Mockito、Spring Test 等，帮助进行 Spring Boot 应用的单元测试和集成测试。
- **配置项**：`scope` 设置为 `test`，表示这个依赖只在测试阶段有效，不会包含在最终的生产环境包中。

8. 总结

- **Spring Cloud 和 Spring Cloud Alibaba** 用于构建分布式系统。
- **MySQL 驱动** 和 **MyBatis-Plus** 用于数据访问和 ORM 操作。
- **Hutool** 提供常用的工具类。
- **Lombok** 简化代码书写。
- **Spring Boot Test** 提供单元测试支持。