实验10-实验报告

姓名：郭睿杰

学号：MG21330024

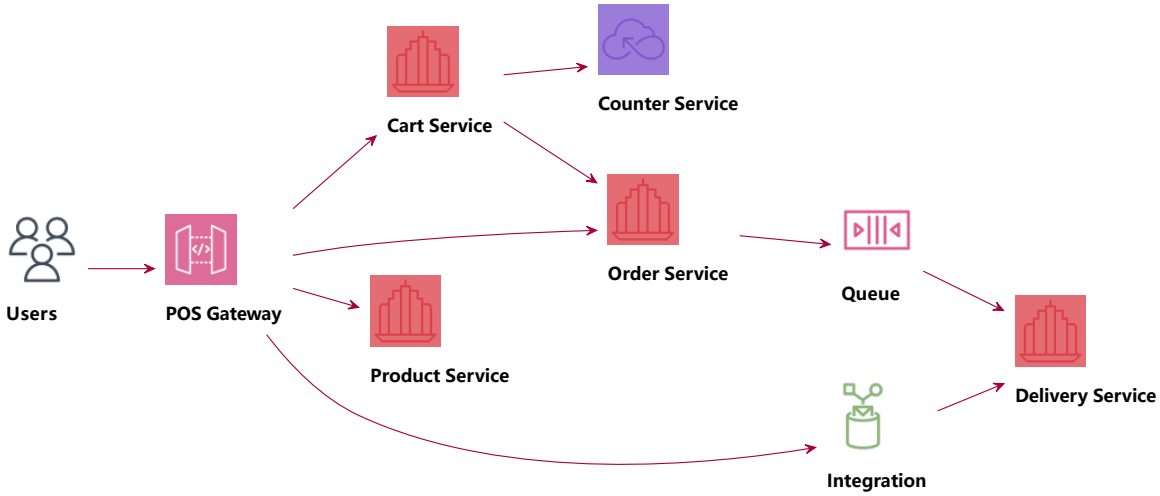
院系：计算机科学与技术

作业完成情况

作业1至10均完成，在本作业10中可得到集中体现。

模块划分和布局

本实验基本上完全参考实验8所提供的模块划分和依赖关系图，进行模块化构建，如下图所示（表达为基本完全参考是因为`Counter Service`，也就是结账服务由`Cart Service`直接提供，其他模块和通信方式完全依照下图进行）。



具体实现细节

- `pos-discovery`：Eureka注册中心。
- `pos-gateway`：网关，用于统一所有服务端口，通信时不需要每个服务给一个端口号。配置文件如下，读者可通过阅读以下配置文件即可得知各个模块所接受的`url前缀`和`端口号`：

```
server:
  port: 8080
spring:
  application:
    name: pos-gateway
  cloud:
    gateway:
      routes:
        - id: products
          uri: http://localhost:8081/
          predicates:
            - Path=/products/**
        - id: carts
          uri: http://localhost:8082/
          predicates:
            - Path=/carts/**
        - id: counter
          uri: http://localhost:8083/
          predicates:
            - Path=/counter/**
        - id: api
          uri: http://localhost:8084/
          predicates:
            - Path=/api/**
        - id: orders
          uri: http://localhost:8085/
          predicates:
            - Path=/orders/**
        - id: delivery
          uri: http://localhost:8086/
          predicates:
            - Path=/delivery/**

eureka:
```

```
client:
  serviceUrl:
    defaultZone: http://localhost:8761/eureka/
```

- `pos-api`：**各类型DTO及对外（对前端）接口暴露**提供者。
  - 各个项目均依赖于项目 `pos-api` 以使用其中的DTO接口；
  - 前端可仅通过 `pos-api` 来访问各个服务（当然也可以通过各个服务自身的接口url访问）；
  - 额外提供账户信息存储功能。
- `pos-products`：提供商品服务。包含两种服务：列举所有商品，并提供按商品Id提供商品信息的查询。本部分通过aw06集成了本地的sql仓库，并通过EhCache进行了性能上的增强。
- `pos-carts`：提供购物车存储和访问服务。该模块具体的工作内容是：存储各个用户当前的购物车，在aw05中通过**mysql**关系型数据库进行存储，而在本作业响应式架构的要求下在aw09之后使用 `mongoDB` 进行存储。
- `pos-orders`：提供订单存储服务。该模块的具体工作内容是：在用户结账后，接受 `pos-carts` 推送来的结账后购物车，根据购物车内物品生成对应的账单。
- `pos-delivery`：提供订单运输信息提供服务。 `pos-orders` 一旦生成一条新订单会通过消息管理中间件 **rabbitmq** 推送一条新的订单信息交给 `pos-delivery`。 `pos-delivery` 生成该订单运输状况信息并存储在数据库。

模块间通信

响应式Feign通信

由于本项目按照响应式架构进行编写，模块间通信也应当遵从响应式架构风格。以下通过举例进行说明：

- 由于 `pos-api` 模块是对前端或用户提供对外服务的核心模块，其是 `pos-carts` 、 `pos-products` 、 `pos-orders` 等模块的消费者。其中，对 `pos-carts` 、 `pos-orders` 模块的消费方式遵从响应式。本代码中使用了一种较新的三方库 `com.playtika.reactivefeign` 所提供的jar包进行响应式通信，下面以购物车消费为例展示。
  - 响应式mongoDB配置文件：

    ```
    spring:
      data:
        mongodb:
          host: localhost
          port: 27017
          database: admin
          username: admin
          password: admin
    ```

  - **ReactiveFeign**购物车消费者核心代码如下：

    ```
    @ReactiveFeignClient(name = "poscarts")
    public interface CartFeignClient {
        @GetMapping("/carts")
        public Flux<CartDto> listCarts();
    ```

```
    @GetMapping("/carts/cart/{cartId}")
    public Mono<CartDto> showCartById(@PathVariable(value = "cartId")
String cartId);

    @GetMapping("/carts/newcart/{accountId}")
    public Mono<CartDto> newCart(@PathVariable(value = "accountId")
String accountId);

    @PostMapping("/carts/cart/{cartId}")
    public Mono<CartDto> addItemToCart(@PathVariable("cartId") String
cartId, @RequestBody ItemDto itemDto);

    @GetMapping("/carts/checkout/{cartId}")
    public Mono<Double> checkout(@PathVariable("cartId") String cartId);
}
```

实验中，当最终创建购物车返回结果到界面，所有响应式代码才会被执行，真正实现了“流”。

- 消费者调用：

```
@GetMapping("/carts/newcart/{accountId}")
public Mono<CartDto> newCart(@PathVariable("accountId") String
accountId) {
    /*为账户accountId创建新购物车。但是，如果account不存在，应当创建失败*/
    return accountRepository.findById(accountId).flatMap(account -> {
        return cartFeignClient.newCart(accountId);
    });
}
```

微服务Feign通信

- 在 `pos-api` 对 `pos-products` 的消费中，仍然使用微服务架构进行消费。使用 **FeignClient**注解。举例如下：

```
@FeignClient(name = "posproducts")
public interface ProductFeignClient {
    @GetMapping("/products")
    public ResponseEntity<List<ProductDto>> listProducts();
    @GetMapping("/products/{productId}")
    public ResponseEntity<ProductDto>
showProductById(@PathVariable(value = "productId") String productId);
}
```

消息驱动通信

- 在 `pos-orders` 生成一条新订单后，自动向 `pos-delivery` 服务推送一条订单信息，推送方式是通过 `RabbitMq` 进行。示例代码如下：
    - 配置文件：

```
rabbitmq:
    host: localhost # rabbitmq docker ip
    port: 5672
    username: guest
    password: guest
    virtual-host: /
```

- `pos-orders` 发送端核心代码：

```
@Autowired
public void setRestTemplate(RestTemplate restTemplate) {
    this.restTemplate = restTemplate;
}
@Override
public void generateDelivery(Order order) {
    rabbitTemplate.convertAndSend("delivery", "order_delivery",
orderMapper.toOrderDto(order));
}
```

- `pos-delivery` 接收端核心代码：

```
/*通过RabbitListener进行监听*/
@RabbitHandler
@RabbitListener(queues = "generateDelivery", containerFactory =
"rabbitListenerContainerFactory")
public void process(Message orderDtoMsg)
{
    OrderDto orderDto = (OrderDto)
jackson2JsonMessageConverter.fromMessage(orderDtoMsg);
    Order order = orderMapper.toOrder(orderDto);
    Delivery delivery = new Delivery();
    delivery.accountId(order.accountId());
    delivery.deliveryId(order.orderId());
    delivery.daysLeft(7);
    delivery.location("Nanjing");
    delivery.items(order.items());
    deliveryRepository.insert(delivery); // 新存储一条delivery信息
}
```

Integration通信

`pos-api` 模块可通过integration访问delivery服务。核心代码：

```
@Bean
public IntegrationFlow outGate() {
    return IntegrationFlows.from("sampleChannel")
        .handle(Http.outboundGateway("http://localhost:8080/delivery")
                .httpMethod(HttpMethod.GET)
                .expectedResponseType(DeliveryDto.class))
        .get();
}
```

性能增强技术的使用

Hystrix熔断

使用熔断机制在服务器出错或商品集合为空的时候返回一条错误信息。使用@Hystrix注解和 **Fallback**等机制：

```
@GetMapping("")
@HystrixCommand(fallbackMethod = "listProductsEmpty")
public ResponseEntity<List<ProductDto>> listProducts(){
    List<ProductDto> productDtos = new ArrayList<>
(productMapper.toProductsDto(this.productService.products()));
    if (productDtos.isEmpty()) {
        return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }
    return new ResponseEntity<>(productDtos, HttpStatus.OK);
}

public ResponseEntity<List<ProductDto>> listProductsEmpty(){
    List<ProductDto> productDtos = new ArrayList<>();
    return new ResponseEntity<>(productDtos, HttpStatus.OK);
}
```

EhCache缓存

使用 **EhCache** 增强products存储库性能。相关配置文件在 `pos-products` 模块的ehcache.xml中。核心代码：

```
@Override
@Cacheable(cacheNames = "products")
public List<Product> products() {
    return Lists.newArrayList(productRepository.findAll());
}
```

实验测试方法

在本地安装mongoDB和rabbitMq。其中请在mongoDB中注册用户admin，密码admin，数据名admin。

- 首先启动注册中心 `pos-discovery`；
- 其次启动网关 `pos-gateway`；
- 然后启动上述介绍的其他服务。
- 具体验证（如url等）可参考test和experiment中相关细节。主要使用postman工具。