

硅谷课堂第十天-整合网关与实现订单和营销管理模块

硅谷课堂第十天-整合网关与实现订单和营销管理模块

一、Spring Cloud 相关概念

1、基本概念

- 1.1、什么是 Spring Cloud
- 1.2、Spring Cloud 和 Spring Boot 关系

- 2、Spring Cloud 相关基础服务组件

二、搭建 Nacos 服务

1、Nacos 概述

- 1.1、基本概念
- 1.2、常见的注册中心
- 1.3、Nacos 结构图

2、Nacos 下载和安装

- 2.1、下载地址和版本
- 2.2、启动 nacos 服务

3、服务注册

- 3.1、在 service 模块配置 pom
- 3.2、配置 service_vod
- 3.3、添加 Nacos 客户端注解
- 3.4、启动客户端微服务

三、整合 Spring Cloud GateWay 网关

1、网关基本概念

- 1.1、Gateway 概述
- 1.2、Gateway 核心概念

2、实现网关转发功能

- 2.1、创建网关模块
- 2.2、引入网关依赖
- 2.3、创建启动类
- 2.4、配置路由规则

3、网关解决跨域问题

- 3.1、跨域概述
- 3.2、创建配置类

4、修改前端配置文件

- 4.1、修改接口为网关地址

四、后台管理系统-订单管理模块

1、环境准备

- 1.1、创建数据库表
- 1.2、创建订单模块
- 1.3、生成订单相关代码
- 1.4、创建启动类
- 1.5、创建配置文件
- 1.6、创建配置类

2、开发订单列表接口

- 2.1、编写 OrderInfoController
- 2.2、编写 Service

3、配置网关

- 3.1、Nacos 注册
- 3.2、配置路由规则

4、开发订单列表前端

- 4.1、定义接口
- 4.2、创建路由
- 4.3、创建 vue 页面

五、后台管理系统-营销管理模块

1、环境准备

- 1.1、创建数据库表
- 1.2、创建营销模块
- 1.3、生成营销相关代码
- 1.4、创建启动类
- 1.5、创建配置文件
- 1.6、创建配置类

2、开发优惠券相关接口

- 2.1、编写 CouponInfoController

一、Spring Cloud 相关概念

1、基本概念

1.1、什么是 Spring Cloud

Spring Cloud 是一系列框架的集合。它利用 Spring Boot 的开发便利性简化了分布式系统基础设施的开发，如服务发现、服务注册、配置中心、消息总线、负载均衡、熔断器、数据监控等，都可以用 Spring Boot 的开发风格做到一键启动和部署。Spring 并没有重复制造轮子，它只是将目前各家公司开发的比较成熟、经得起实际考验的服务框架组合起来，通过 SpringBoot 风格进行再封装屏蔽掉了复杂的配置和实现原理，最终给开发者留出了一套简单易懂、易部署和易维护的分布式系统开发工具包。

1.2、Spring Cloud 和 Spring Boot 关系

Spring Boot 是 Spring 的一套快速配置脚手架，可以基于 Spring Boot 快速开发单个微服务，Spring Cloud 是一个基于 Spring Boot 实现的开发工具；Spring Boot 专注于快速、方便集成的单个微服务个体，Spring Cloud 关注全局的服务治理框架；Spring Boot 使用了默认大于配置的理念，很多集成方案已经帮你选择好了，能不配置就不配置，Spring Cloud 很大一部分是基于 Spring Boot 来实现，必须基于 Spring Boot 开发。可以单独使用 Spring Boot 开发项目，但是 Spring Cloud 离不开 Spring Boot。

2、Spring Cloud 相关基础服务组件

服务发现——Netflix Eureka （Nacos）

服务调用——Netflix Feign

熔断器——Netflix Hystrix

服务网关——Spring Cloud GateWay

分布式配置——Spring Cloud Config （Nacos）

消息总线 —— Spring Cloud Bus （Nacos）

二、搭建 Nacos 服务

1、Nacos 概述

1.1、基本概念

Nacos 是阿里巴巴推出来的一个新开源项目，是一个更易于构建云原生应用的动态服务发现、配置管理和服务管理平台。Nacos 致力于帮助您发现、配置和管理微服务。Nacos 提供了一组简单易用的特性集，帮助您快速实现动态服务发现、服务配置、服务元数据及流量管理。Nacos 帮助您更敏捷和容易地构建、交付和管理微服务平台。Nacos 是构建以“服务”为中心的现代应用架构 (例如微服务范式、云原生范式) 的服务基础设施。

1.2、常见的注册中心

(1) Eureka (原生, 2.0 遇到性能瓶颈, 停止维护)

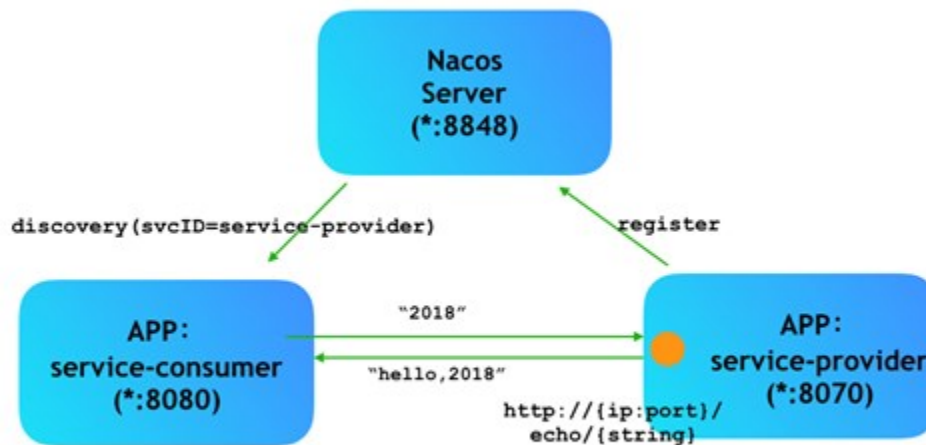
(2) Zookeeper (支持, 专业的独立产品。例如 : dubbo)

(3) Consul (原生, GO 语言开发)

(4) Nacos

- 相对于 Spring Cloud Eureka 来说, Nacos 更强大。Nacos = Spring Cloud Eureka + Spring Cloud Config
- Nacos 可以与 Spring, Spring Boot, Spring Cloud 集成, 并能代替 Spring Cloud Eureka, Spring Cloud Config
- 通过 Nacos Server 和 spring-cloud-starter-alibaba-nacos-discovery 实现服务的注册与发现

1.3、Nacos 结构图



2、Nacos 下载和安装

2.1、下载地址和版本

下载地址：<https://github.com/alibaba/nacos/releases>

下载版本：nacos-server-1.1.4.tar.gz 或 nacos-server-1.1.4.zip，解压任意目录即可

2.2、启动 nacos 服务

(1) Linux/Unix/Mac

启动命令(standalone 代表着单机模式运行，非集群模式)

启动命令：sh `startup.sh` -m standalone

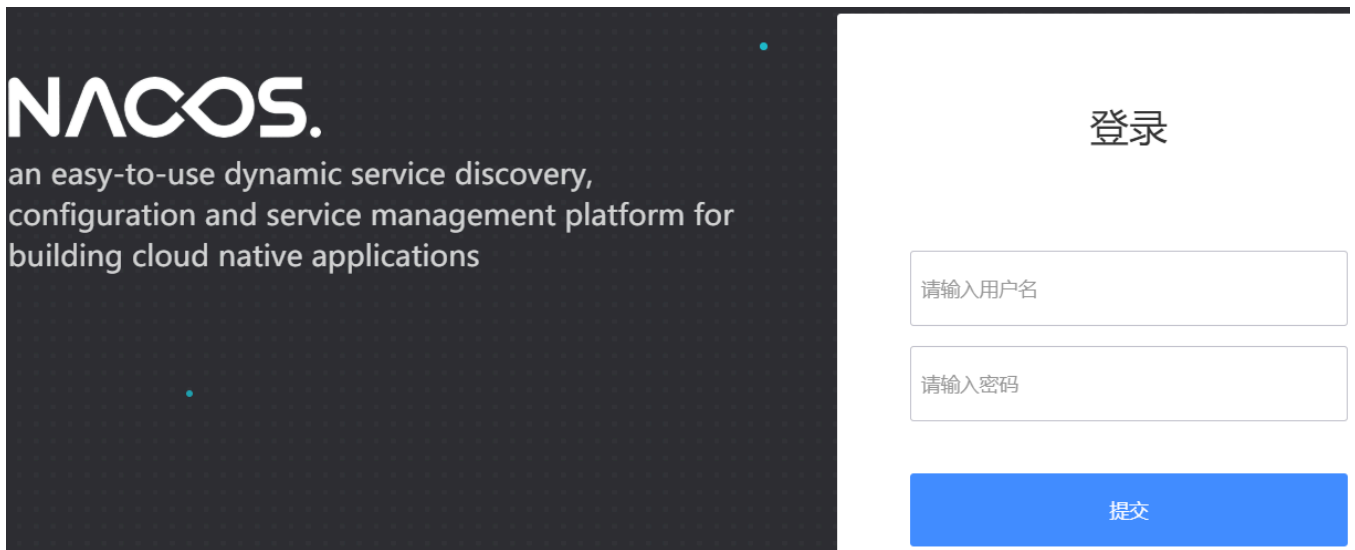
(2) Windows

启动方式，cmd 打开，执行命令：startup.cmd -m standalone。

访问：<http://localhost:8848/nacos>

用户名/密码：nacos/nacos

(3) 登录界面



(4) 主界面



3、服务注册

把 service_vod 微服务注册到注册中心，其他模块注册步骤相同

3.1、在 service 模块配置 pom

配置 Nacos 客户端的 pom 依赖

```
<!-- 服务注册 -->
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
</dependency>

<!-- 服务调用feign -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

3.2、配置 service_vod

配置 application.properties，在客户端微服务中添加注册 Nacos 服务的配置信息

```
# nacos服务地址
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
```

3.3、添加 Nacos 客户端注解

在 service_vod 微服务启动类中添加注解

```
@EnableDiscoveryClient
```

3.4、启动客户端微服务

启动注册中心

启动已注册的微服务，可以在 Nacos 服务列表中看到被注册的微服务

NACOS 1.1.4

配置管理

配置列表

历史版本

监听查询

服务管理

服务列表

订阅者列表

命名空间

集群管理

节点列表

public

服务列表 | public

服务名称

分组名称

隐藏空服务: ☒

查

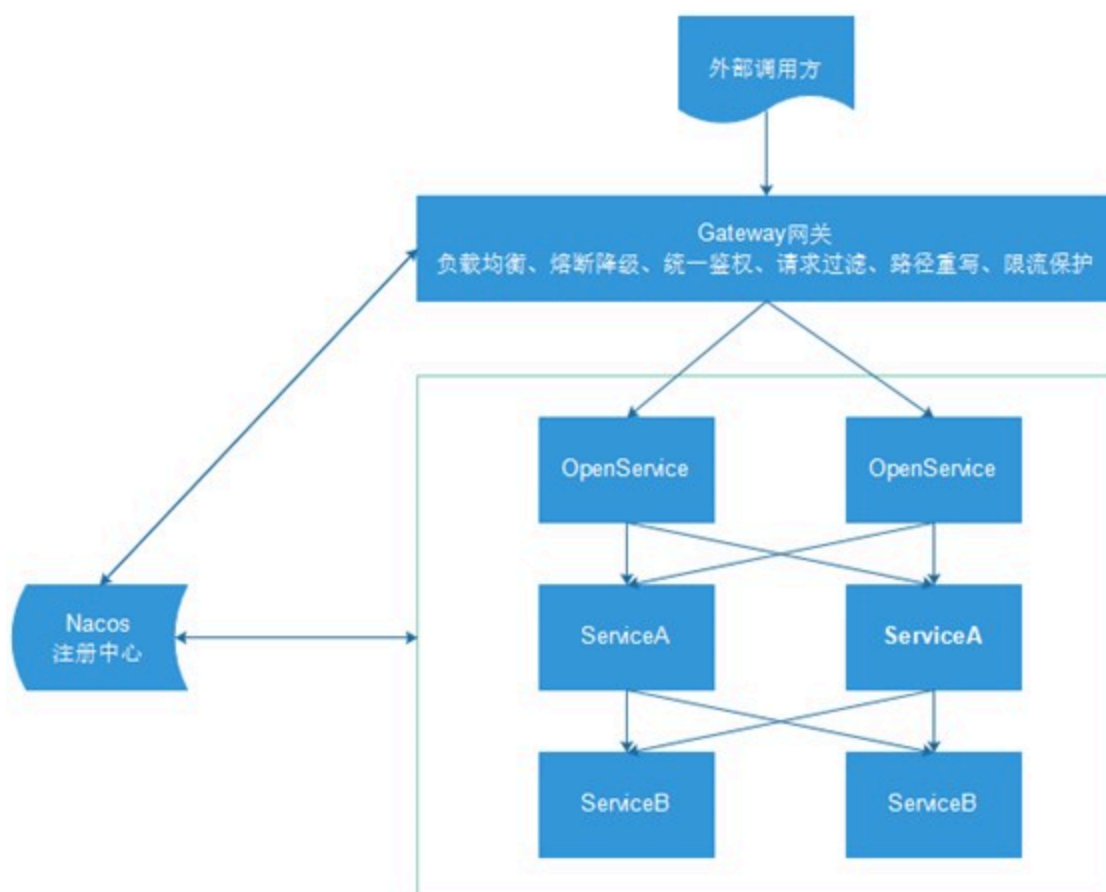
服务名	分组名称	集群数目	实例数	健康实例数
service-vod	DEFAULT_GROUP	1	1	1

三、整合 Spring Cloud GateWay 网关

1、网关基本概念

1.1、Gateway 概述

Spring cloud gateway是 spring 官方基于 Spring 5.0、Spring Boot2.0 和 Project Reactor 等技术开发的网关，Spring Cloud Gateway 旨在为微服务架构提供简单、有效和统一的 API 路由管理方式，Spring Cloud Gateway 作为 Spring Cloud 生态系统中的网关，目标是替代 Netflix Zuul，其不仅提供统一的路由方式，并且还基于 Filter 链的方式提供了网关基本的功能，例如：安全、监控/埋点、限流等。



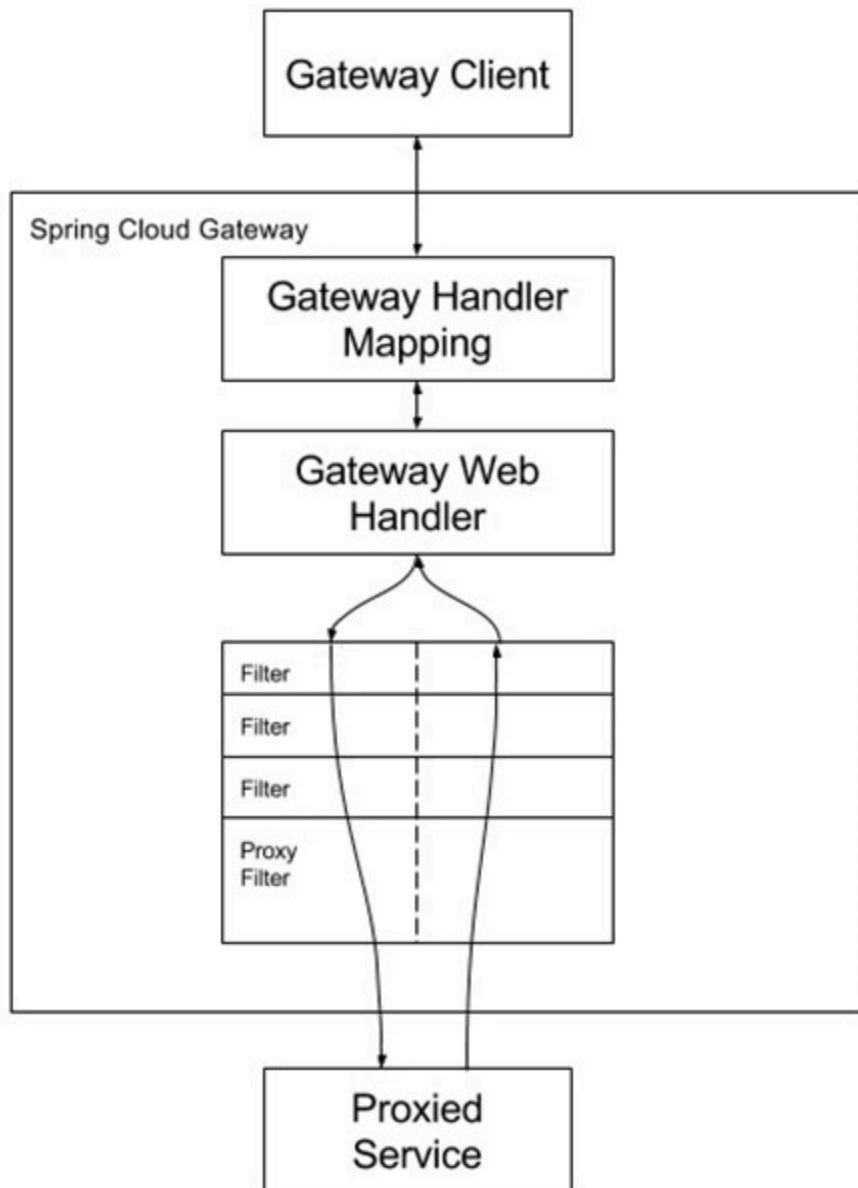
1.2、Gateway 核心概念

网关提供 API 全托管服务，丰富的 API 管理功能，辅助企业管理大规模的 API，以降低管理成本和安全风险，包括协议适配、协议转发、安全策略、防刷、流量、监控日志等。一般来说网关对外暴露的 URL 或者接口信息，我们统称为路由信息。如果研发过网关中间件或者使用过 Zuul 的人，会知道网关的核心是 Filter 以及 Filter Chain（Filter 责任链）。Spring Cloud Gateway 也具有路由和 Filter 的概念。下面介绍一下 Spring Cloud Gateway 中几个重要的概念。

(1) 路由：路由是网关最基础的部分，路由信息有一个 ID、一个目的 URL、一组断言和一组 Filter 组成。如果断言路由为真，则说明请求的 URL 和配置匹配

(2) 断言：Java8 中的断言函数。Spring Cloud Gateway 中的断言函数输入类型是 Spring5.0 框架中的 ServerWebExchange。Spring Cloud Gateway 中的断言函数允许开发者去定义匹配来自于 http request 中的任何信息，比如请求头和参数等

(3) 过滤器：一个标准的 Spring webFilter。Spring cloud gateway 中的 filter 分为两种类型的 Filter，分别是 Gateway Filter 和 Global Filter。过滤器 Filter 将会对请求和响应进行修改处理




如图所示，Spring cloud Gateway 发出请求。然后再由 Gateway Handler Mapping 中找到与请求相匹配的路由，将其发送到 Gateway web handler。Handler 再通过指定的过滤器链将请求发送到实际的服务执行业务逻辑，然后返回。

2、实现网关转发功能

2.1、创建网关模块

(1) 在 ggkt_parent 下创建 service_gateway

 New Module

Add as module to com.atguigu:ggkt_parent:0.0.1-SNAPSHOT

Parent com.atguigu:ggkt_parent:0.0.1-SNAPSHOT

GroupId com.atguigu

ArtifactId **service_gateway**

Version 0.0.1-SNAPSHOT

2.2、引入网关依赖

```
<dependencies>
  <dependency>
    <groupId>com.atguigu</groupId>
    <artifactId>service_utils</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </dependency>

  <!-- 网关 -->
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
  </dependency>

  <!-- 服务注册 -->
  <dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
  </dependency>
</dependencies>
```

2.3、创建启动类

```
@SpringBootApplication
public class ApiGatewayApplication {
    public static void main(String[] args) {
        SpringApplication.run(ApiGatewayApplication.class, args);
    }
}
```

2.4、配置路由规则

(1) 编写 application.properties

```
# 服务端口
server.port=8333

# 服务名
spring.application.name=service-gateway

# nacos服务地址
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848

# 使用服务发现路由
spring.cloud.gateway.discovery.locator.enabled=true

# service-vod模块配置
# 设置路由id
spring.cloud.gateway.routes[0].id=service-vod
# 设置路由的uri
spring.cloud.gateway.routes[0].uri=lb://service-vod
# 设置路由断言,代理servicerId为auth-service的/auth/路径
spring.cloud.gateway.routes[0].predicates= Path=/*/vod/**
```

3、网关解决跨域问题

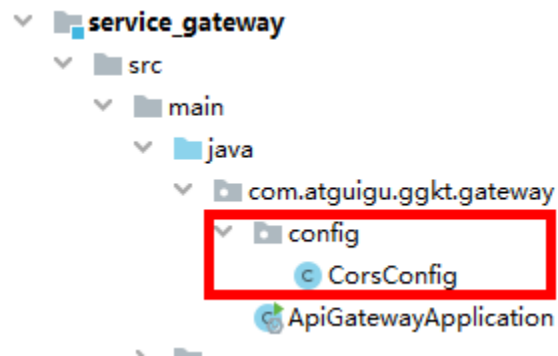
3.1、跨域概述

跨域本质是浏览器对于 ajax 请求的一种安全限制：一个页面发起的 ajax 请求，只能是与当前页

域名相同的路径，这能有效的阻止跨站攻击。因此：跨域问题 是针对 ajax 的一种限制。但是这却给我们的开发带来了不便，而且在实际生产环境中，肯定会有很多台服务器之间交互，地址和端口都可能不同。

之前我们通过服务器添加注解实现，现在我们跨域通过网关来解决跨域问题。

3.2、创建配置类



```
@Configuration
public class CorsConfig {
    //处理跨域
    @Bean
    public CorsWebFilter corsFilter() {
        CorsConfiguration config = new CorsConfiguration();
        config.addAllowedMethod("*");
        config.addAllowedOrigin("*");
        config.addAllowedHeader("*");
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource(new PathPatternParser());
        source.registerCorsConfiguration("/**", config);
        return new CorsWebFilter(source);
    }
}
```

注意：目前我们已经在网关做了跨域处理，那么 service 服务就不需要再做跨域处理了，将之前在 controller 类上添加过@CrossOrigin 标签的去掉

4、修改前端配置文件

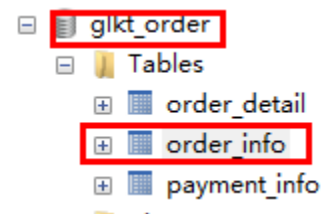
4.1、修改接口为网关地址



四、后台管理系统-订单管理模块

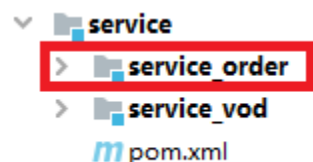
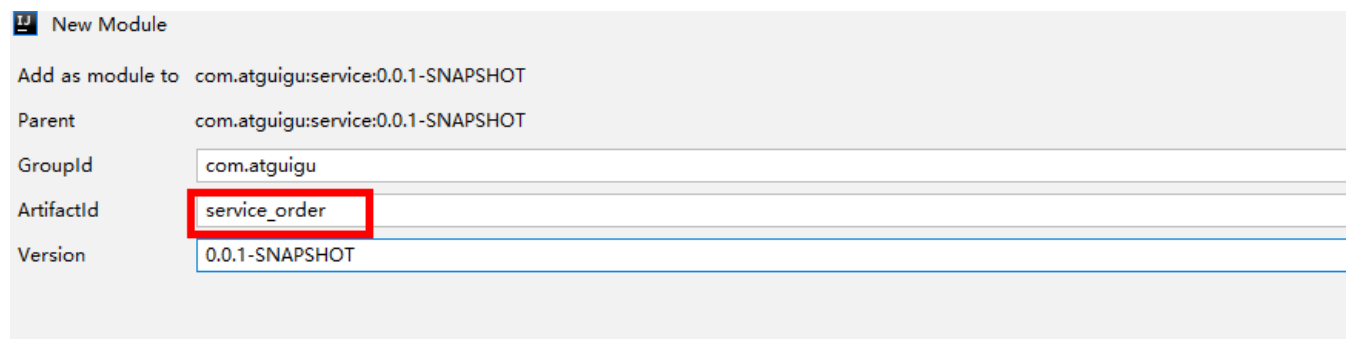
1、环境准备

1.1、创建数据库表

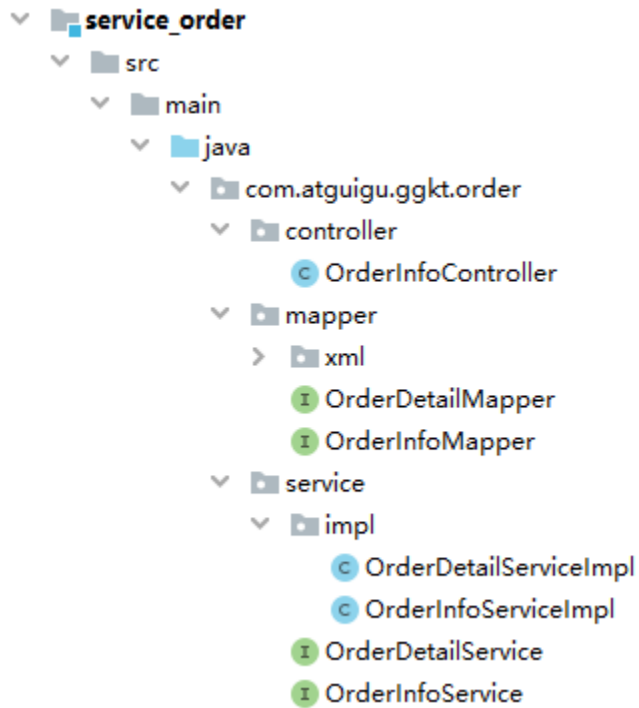


1.2、创建订单模块

(1) service 模块下创建 service_order 模块



1.3、生成订单相关代码



1.4、创建启动类

```
@SpringBootApplication
public class ServiceOrderApplication {
    public static void main(String[] args) {
        SpringApplication.run(ServiceOrderApplication.class, args);
    }
}
```

1.5、创建配置文件

```
# 服务端口
server.port=8302

# 服务名
spring.application.name=service-order

# 环境设置: dev、test、prod
spring.profiles.active=dev

# mysql数据库连接
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/glkt_order?characterEncoding=utf-8&useSSL=false
spring.datasource.username=root
spring.datasource.password=root

# 返回json的全局时间格式
spring.jackson.date-format=yyyy-MM-dd HH:mm:ss
spring.jackson.time-zone=GMT+8

# mybatis日志
mybatis-plus.configuration.log-impl=org.apache.ibatis.logging.stdout.StdoutImpl

mybatis-plus.mapper-locations=classpath:com/atguigu/glkt/order/mapper/xml/*.xml
```


1.6、创建配置类

```
/**
 * 配置类
 */
@Configuration
@MapperScan("com.atguigu.ggkt.order.mapper")
public class OrderConfig {
    /**
     * 分页插件
     */
    @Bean
    public PaginationInterceptor paginationInterceptor() {
        return new PaginationInterceptor();
    }
}
```

2、开发订单列表接口

2.1、编写 OrderInfoController

```
@Api(tags = "订单管理")
@RestController
@RequestMapping(value="/admin/order/orderInfo")
public class OrderInfoController {

    @Autowired
    private OrderInfoService orderInfoService;

    @ApiOperation(value = "获取分页列表")
    @GetMapping("{page}/{limit}")
    public Result index(
        @ApiParam(name = "page", value = "当前页码", required = true)
        @PathVariable Long page,
        @ApiParam(name = "limit", value = "每页记录数", required = true)
        @PathVariable Long limit,
        @ApiParam(name = "orderInfoVo", value = "查询对象", required = false)
        OrderInfoQueryVo orderInfoQueryVo) {
        Page<OrderInfo> pageParam = new Page<>(page, limit);
        Map<String, Object> map = orderInfoService.findPageOrderInfo(pageParam, orderInfoQueryVo);
        return Result.ok(map);
    }
}
```

2.2、编写 Service

(1) OrderInfoService 定义方法

```
public interface OrderInfoService extends IService<OrderInfo> {
    //订单列表
    Map<String, Object> findPageOrderInfo(Page<OrderInfo> pageParam, OrderInfoQueryVo orderInfoQueryVo);
}
```

(2) OrderInfoServiceImpl 实现方法

```

/**
 * <p>
 * 订单表 订单表 服务实现类
 * </p>
 *
 * @author atguigu
 * @since 2022-04-28
 */
@Service
public class OrderInfoServiceImpl extends ServiceImpl<OrderInfoMapper, OrderInfo> implements OrderInfoService {

    @Autowired
    private OrderDetailService orderDetailService;

    //订单列表
    @Override
    public Map<String, Object> selectOrderInfoPage(Page<OrderInfo> pageParam,
                                                    OrderInfoQueryVo orderInfoQueryVo) {

        //orderInfoQueryVo获取查询条件
        Long userId = orderInfoQueryVo.getUserId();
        String outTradeNo = orderInfoQueryVo.getOutTradeNo();
        String phone = orderInfoQueryVo.getPhone();
        String createTimeBegin = orderInfoQueryVo.getCreateTimeBegin();
        String createTimeEnd = orderInfoQueryVo.getCreateTimeEnd();
        Integer orderStatus = orderInfoQueryVo.getOrderStatus();

        //判断条件值是否为空，不为空，进行条件封装
        QueryWrapper<OrderInfo> wrapper = new QueryWrapper<>();
        if(!StringUtils.isEmpty(orderStatus)) {
            wrapper.eq("order_status",orderStatus);
        }
        if(!StringUtils.isEmpty(userId)) {
            wrapper.eq("user_id",userId);
        }
        if(!StringUtils.isEmpty(outTradeNo)) {
            wrapper.eq("out_trade_no",outTradeNo);
        }
    }
}

```

```

        if(!StringUtils.isEmpty(phone)) {
            wrapper.eq("phone",phone);
        }
        if(!StringUtils.isEmpty(createTimeBegin)) {
            wrapper.ge("create_time",createTimeBegin);
        }
        if(!StringUtils.isEmpty(createTimeEnd)) {
            wrapper.le("create_time",createTimeEnd);
        }
        //调用实现条件分页查询
        Page<OrderInfo> pages = baseMapper.selectPage(pageParam, wrapper);
        long totalCount = pages.getTotal();
        long pageCount = pages.getPages();
        List<OrderInfo> records = pages.getRecords();
        //订单里面包含详情内容，封装详情数据，根据订单id查询详情
        records.stream().forEach(item -> {
            this.getOrderDetail(item);
        });

        //所有需要数据封装map集合，最终返回
        Map<String,Object> map = new HashMap<>();
        map.put("total",totalCount);
        map.put("pageCount",pageCount);
        map.put("records",records);
        return map;
    }

    //查询订单详情数据
    private OrderInfo getOrderDetail(OrderInfo orderInfo) {
        //订单id
        Long id = orderInfo.getId();
        //查询订单详情
        OrderDetail orderDetail = orderDetailService.getById(id);
        if(orderDetail != null) {
            String courseName = orderDetail.getCourseName();
            orderInfo.getParam().put("courseName",courseName);
        }
    }

```

```
        return orderInfo;
    }
}
```

3、配置网关

3.1、Nacos 注册

(1) service_order 启动类添加注解

```
@SpringBootApplication
@EnableDiscoveryClient
public class ServiceOrderApplication {
    public static void main(String[] args) {
        SpringApplication.run(ServiceOrderApplication.class, args);
    }
}
```

(2) service_order 配置文件

```
# nacos服务地址
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
```

3.2、配置路由规则

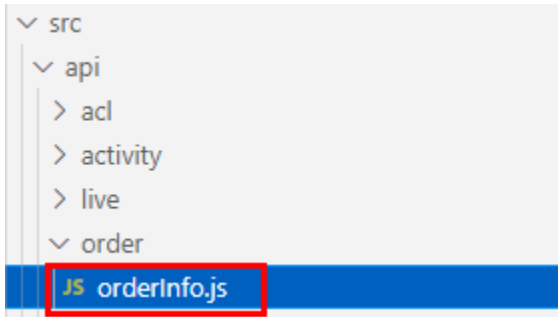
(1) service_gateway 配置文件

```
# service-order模块配置
# 设置路由id
spring.cloud.gateway.routes[1].id=service-order
# 设置路由的uri
spring.cloud.gateway.routes[1].uri=lb://service-order
# 设置路由断言,代理servicerId为auth-service的/auth/路径
spring.cloud.gateway.routes[1].predicates= Path=/*/order/**
```

4、开发订单列表前端

4.1、定义接口

(1) 创建 api/order/orderInfo.js



```
import request from "@/utils/request";

const api_name = "/admin/order/orderInfo";

export default {
  getPageList(page, limit, searchObj) {
    return request({
      url: `${api_name}/${page}/${limit}`,
      method: "get",
      params: searchObj,
    });
  },
};
```

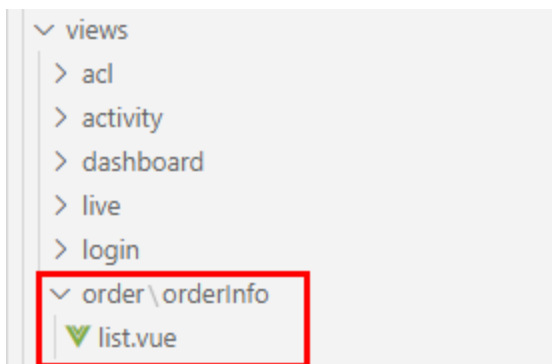
4.2、创建路由

(1) router/index.js 定义路由

```
{
  path: '/order',
  component: Layout,
  redirect: '/order/orderInfo/list',
  name: 'Order',
  meta: { title: '订单管理', icon: 'el-icon-truck' },
  alwaysShow: true,
  children: [
    {
      path: 'orderInfo/list',
      name: 'OrderInfo',
      component: () => import('@/views/order/orderInfo/list'),
      meta: { title: '订单列表' }
    }
  ]
},
```

4.3、创建 vue 页面

(1) 创建 views/order/orderInfo/list.vue



```
<template>
  <div class="app-container">
    <el-card class="operate-container" shadow="never">
      <el-form :inline="true" class="demo-form-inline">
        <el-form-item>
          <el-input v-model="searchObj.outTradeNo" placeholder="订单号" />
        </el-form-item>
        <el-form-item>
          <el-input v-model="searchObj.phone" placeholder="手机" />
        </el-form-item>
        <el-form-item>
          <el-date-picker
            v-model="searchObj.createTimeBegin"
            type="date"
            placeholder="选择下单开始日期"
            value-format="yyyy-MM-dd"
          />
        </el-form-item>
        <el-form-item>
          <el-date-picker
            v-model="searchObj.createTimeEnd"
            type="date"
            placeholder="选择截止日期"
            value-format="yyyy-MM-dd"
          />
        </el-form-item>
        <el-form-item>
          <el-select
            v-model="searchObj.orderStatus"
            placeholder="订单状态"
            class="v-select patient-select"
          >
            <el-option
              v-for="item in statusList"
              :key="item.status"
              :label="item.name"
              :value="item.status"
            />
          </el-select>
        </el-form-item>
      </el-form>
    </el-card>
  </div>
</template>
```



```

        >
        </el-option>
    </el-select>
</el-form-item>
<el-button type="primary" icon="el-icon-search" @click="fetchData()"
>查询</el-button
>
    <el-button type="default" @click="resetData()">清空</el-button>
</el-form>
</el-card>
<!-- 列表 -->
<el-table
    v-loading="listLoading"
    :data="list"
    border
    fit
    highlight-current-row
>
    <el-table-column label="序号" width="60" align="center">
        <template slot-scope="scope">
            {{ (page - 1) * limit + scope.$index + 1 }}
        </template>
    </el-table-column>
    <el-table-column prop="outTradeNo" label="订单号" width="160" />
    <el-table-column prop="courseName" label="课程名称" width="160">
        <template slot-scope="scope">
            {{ scope.row.param.courseName }}
        </template>
    </el-table-column>
    <el-table-column prop="finalAmount" label="订单金额" width="90" />
    <el-table-column prop="nickName" label="下单用户" />
    <el-table-column prop="phone" label="用户手机" />
    <el-table-column prop="payTime" label="支付时间" width="156" />
    <el-table-column prop="orderStatus" label="订单状态">
        <template slot-scope="scope">
            {{ scope.row.orderStatus == 0 ? "未支付" : "已支付" }}
        </template>
    </el-table-column>

```

```

    </el-table-column>
    <el-table-column prop="createTime" label="下单时间" width="156" />
  </el-table>
  <!-- 分页组件 -->
  <el-pagination
    :current-page="page"
    :total="total"
    :page-size="limit"
    :page-sizes="[5, 10, 20, 30, 40, 50, 100]"
    style="padding: 30px 0; text-align: center;"
    layout="sizes, prev, pager, next, jumper, ->, total, slot"
    @current-change="fetchData"
    @size-change="changeSize"
  />
</div>
</template>
<script>
  import orderInfoApi from "@api/order/orderInfo";

  export default {
    data() {
      return {
        listLoading: true, // 数据是否正在加载
        list: null, // banner列表
        total: 0, // 数据库中的总记录数
        page: 1, // 默认页码
        limit: 10, // 每页记录数
        searchObj: {}, // 查询表单对象
        statusList: [
          {
            status: 0,
            name: "未支付",
          },
          {
            status: 1,
            name: "已支付",
          },
        ],
      };
    },
  };

```

```

    ],
  };
},
// 生命周期函数：内存准备完毕，页面尚未渲染
created() {
  this.fetchData();
},
// 生命周期函数：内存准备完毕，页面渲染成功
mounted() {
  console.log("list mounted.....");
},
methods: {
  // 当页码发生改变的时候
  changeSize(size) {
    this.limit = size;
    this.fetchData(1);
  },
  // 加载banner列表数据
  fetchData(page = 1) {
    // 异步获取远程数据 (ajax)
    this.page = page;
    orderInfoApi
      .getPageList(this.page, this.limit, this.searchObj)
      .then((response) => {
        this.list = response.data.records;
        this.total = response.data.total;
        // 数据加载并绑定成功
        this.listLoading = false;
      });
  },
  // 重置查询表单
  resetData() {
    console.log("重置查询表单");
    this.searchObj = {};
    this.fetchData();
  },
},
},

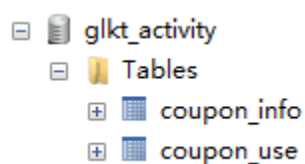
```

```
};  
</script>
```

五、后台管理系统-营销管理模块

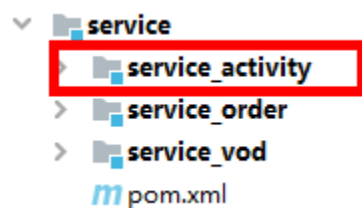
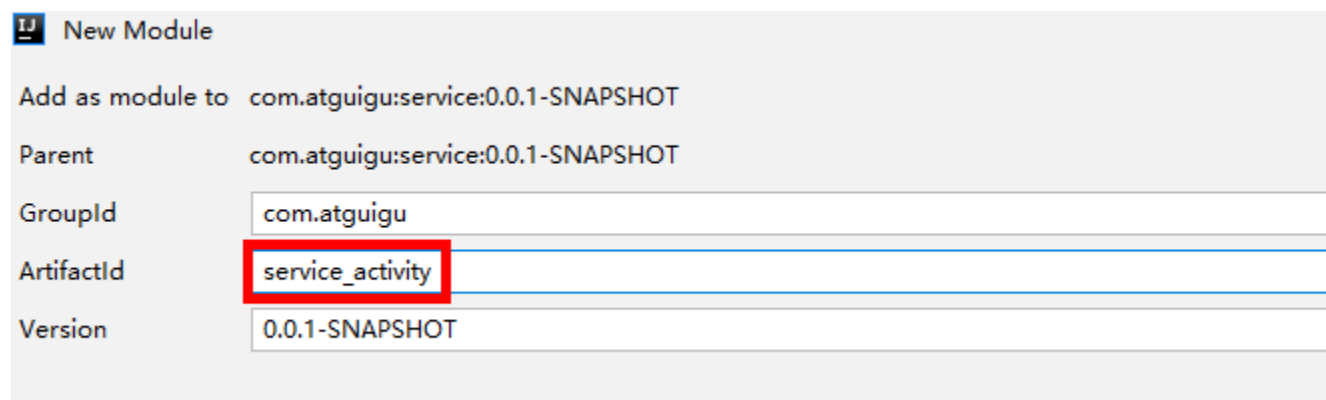
1、环境准备

1.1、创建数据库表

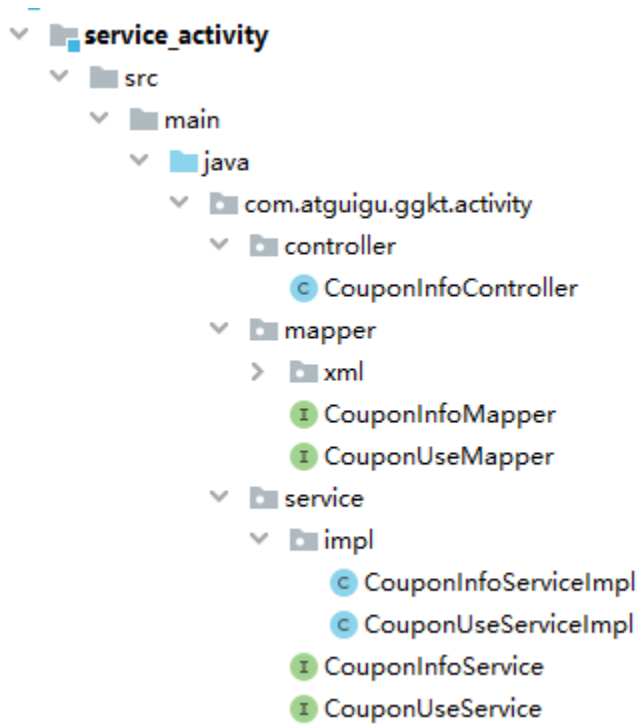


1.2、创建营销模块

(1) service 模块下创建 service_activity 模块



1.3、生成营销相关代码



1.4、创建启动类

```
@SpringBootApplication
@EnableDiscoveryClient
public class ServiceActivityApplication {

    public static void main(String[] args) {
        SpringApplication.run(ServiceActivityApplication.class, args);
    }
}
```

1.5、创建配置文件

```
# 服务端口
server.port=8303

# 服务名
spring.application.name=service-activity

# 环境设置: dev、test、prod
spring.profiles.active=dev

# mysql数据库连接
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/glkt_activity?characterEncoding=utf-8&useSSL=false
spring.datasource.username=root
spring.datasource.password=root

# 返回json的全局时间格式
spring.jackson.date-format=yyyy-MM-dd HH:mm:ss
spring.jackson.time-zone=GMT+8

# mybatis日志
mybatis-plus.configuration.log-impl=org.apache.ibatis.logging.stdout.StdOutImpl

mybatis-plus.mapper-locations=classpath:com/atguigu/glkt/activity/mapper/xml/*.xml

# nacos服务地址
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
```

1.6、创建配置类

```
@Configuration
@MapperScan("com.atguigu.ggkt.activity.mapper")
public class ActivityConfig {
    /**
     * 分页插件
     */
    @Bean
    public PaginationInterceptor paginationInterceptor() {
        return new PaginationInterceptor();
    }
}
```

2、开发优惠券相关接口

2.1、编写 CouponInfoController


```
@RestController
@RequestMapping("/admin/activity/couponInfo")
public class CouponInfoController {

    @Autowired
    private CouponInfoService couponInfoService;

    @ApiOperation(value = "获取分页列表")
    @GetMapping("/{page}/{limit}")
    public Result index(
        @ApiParam(name = "page", value = "当前页码", required = true)
        @PathVariable Long page,
        @ApiParam(name = "limit", value = "每页记录数", required = true)
        @PathVariable Long limit) {
        Page<CouponInfo> pageParam = new Page<>(page, limit);
        IPage<CouponInfo> pageModel = couponInfoService.page(pageParam);
        return Result.ok(pageModel);
    }

    @ApiOperation(value = "获取优惠券")
    @GetMapping("get/{id}")
    public Result get(@PathVariable String id) {
        CouponInfo couponInfo = couponInfoService.getById(id);
        return Result.ok(couponInfo);
    }

    @ApiOperation(value = "新增优惠券")
    @PostMapping("save")
    public Result save(@RequestBody CouponInfo couponInfo) {
        couponInfoService.save(couponInfo);
        return Result.ok();
    }

    @ApiOperation(value = "修改优惠券")
    @PutMapping("update")
    public Result updateById(@RequestBody CouponInfo couponInfo) {
        couponInfoService.updateById(couponInfo);
    }
}
```

```

        return Result.ok();
    }

    @ApiOperation(value = "删除优惠券")
    @DeleteMapping("remove/{id}")
    public Result remove(@PathVariable String id) {
        couponInfoService.removeById(id);
        return Result.ok();
    }

    @ApiOperation(value="根据id列表删除优惠券")
    @DeleteMapping("batchRemove")
    public Result batchRemove(@RequestBody List<String> idList){
        couponInfoService.removeByIds(idList);
        return Result.ok();
    }

    @ApiOperation(value = "获取分页列表")
    @GetMapping("couponUse/{page}/{limit}")
    public Result index(
        @ApiParam(name = "page", value = "当前页码", required = true)
        @PathVariable Long page,

        @ApiParam(name = "limit", value = "每页记录数", required = true)
        @PathVariable Long limit,
        @ApiParam(name = "couponUseVo", value = "查询对象", required = false)
        CouponUseQueryVo couponUseQueryVo) {
        Page<CouponUse> pageParam = new Page<>(page, limit);
        IPage<CouponUse> pageModel = couponInfoService.selectCouponUsePage(pageParam, couponUseQueryVo);
        return Result.ok(pageModel);
    }
}

```

2.2、编写 Service

(1) CouponInfoService 定义方法

```
public interface CouponInfoService extends IService<CouponInfo> {  
    //获取已使用优惠券列表  
    IPage<CouponUse> selectCouponUsePage(Page<CouponUse> pageParam, CouponUseQueryVo couponUseQueryVo)  
}
```