医院管理

## 一、注册中心与服务调用

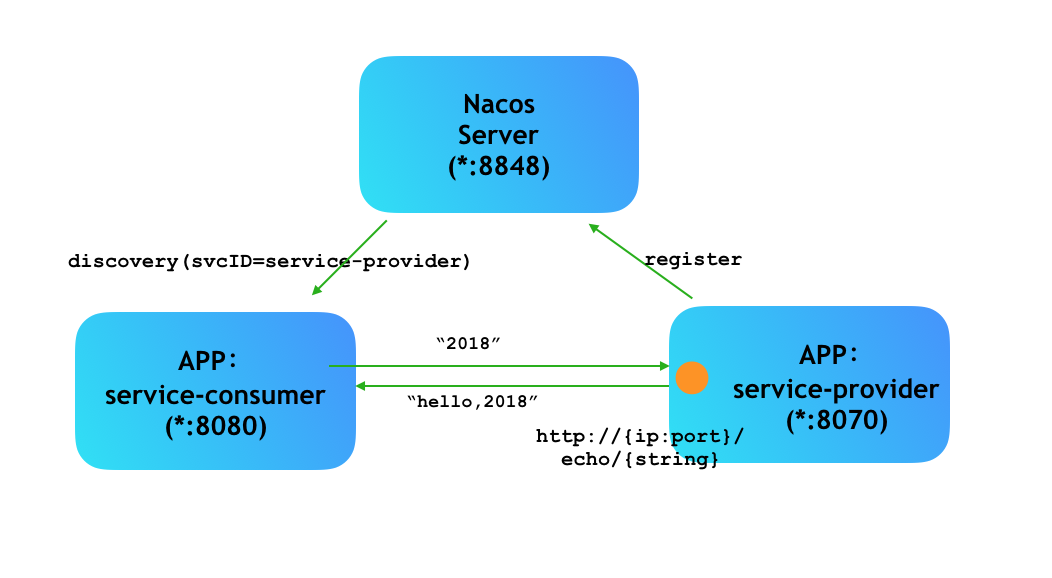
目前在医院列表中需要医院的信息和等级信息,而两段信息属于不同的的模块,service-hosp和service-cmn，所以需要使用到远程调用。

### 1、Nacos

Nacos 是阿里巴巴推出来的一个新开源项目，这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。



### 2、Nacos下载和安装

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

**启动Nacos服务**

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

Windows

启动命令：cmd startup.cmd 或者双击startup.cmd运行文件。（默认为集群模式启动，需要改为单机模式）

访问：<http://localhost:8848/nacos>

用户名：nacos

密码：nacos

### 3、注册服务

1）在service父模块pom文件引入依赖

1. <!-- 服务注册 -->
2. **<dependency>**
3. **<groupId>**com.alibaba.cloud**</groupId>**
4. **<artifactId>**spring-cloud-starter-alibaba-nacos-discovery**</artifactId>**
5. **</dependency>**

2）在service-hosp的配置文件添加nacos服务地址

1. # nacos服务地址
2. spring.cloud.nacos.discovery.server-addr=localhost:8848

3）在service-hosp的启动类添加注解

1. @SpringBootApplication
2. @ComponentScan(basePackages = "com.xlh")
3. @EnableDiscoveryClient
4. **public** **class** ServiceHospApplication {
5. **public** **static** **void** main(String[] args) {
6. SpringApplication.run(ServiceHospApplication.**class**, args);
7. }
8. }

4）启动service-hosp服务，在Nacos管理界面的**服务列表**中可以看到注册的服务

5）使用相同方式注册service-cmn服务



## 二、医院管理实现

### 1、医院列表Api

医院列表分页Controller

1. //医院列表分页
2. @GetMapping("/list/{page}/{limit}")
3. **public** Result listHosp(@PathVariable **int** page, @PathVariable **int** limit, HospitalQueryVo hospitalQueryVo){
4. Page<Hospital> page1=hospitalService.selectHospPage(page-1,limit,hospitalQueryVo);
5. **return**  Result.ok(page1);
6. }

添加service分页接口，在后面实现

### 2、service-cmn模块提供接口

**说明：**提供两个api接口，如省市区不需要上级编码，医院等级需要上级编码。

**Controller:**

1. //根据dictcode和value查询name
2. @GetMapping("/getName/{dictCode}/{value}")
3. **public** String getName(@PathVariable String dictCode,@PathVariable String value){
4. **return** dictService.getName(dictCode,value);
5. }
7. //根据value查询
8. @GetMapping("/getName/{value}")
9. **public** String getName(@PathVariable String value){
10. **return** dictService.getName("",value);
11. }

**Service:**

1. @Override
2. **public** String getName(String dictCode, String value) {
3. **if**(StringUtils.isEmpty(dictCode)){
4. //只根据value查询
5. QueryWrapper<Dict> queryWrapper=**new** QueryWrapper<>();
6. queryWrapper.eq("value",value);
7. Dict dict=baseMapper.selectOne(queryWrapper);
8. **return** dict.getName();
9. }
10. **else**{
11. Dict dict=getDictByDictCode(dictCode);
12. Long parent\_id=dict.getParentId();
13. Dict dictFinal=baseMapper.selectOne(**new** QueryWrapper<Dict>().eq("parent\_id",parent\_id).eq("value",value));
14. **return** dictFinal.getName();
15. }
16. }
18. **private** Dict getDictByDictCode(String dictCode){
19. QueryWrapper<Dict> queryWrapper=**new** QueryWrapper<>();
20. queryWrapper.eq("dict\_code",dictCode);
21. **return** baseMapper.selectOne(queryWrapper);
22. }

### 3、封装Feign服务调用模块

1）创建service-client父模块

创建模块并修改pom.xml文件，作为父模块，修改打包类型：

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<project** xmlns="http://maven.apache.org/POM/4.0.0"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
5. **<parent>**
6. **<artifactId>**yygh\_parent**</artifactId>**
7. **<groupId>**com.xlh**</groupId>**
8. **<version>**0.0.1-SNAPSHOT**</version>**
9. **</parent>**
10. **<modelVersion>**4.0.0**</modelVersion>**
12. **<artifactId>**service\_client**</artifactId>**
13. **<packaging>**pom**</packaging>**
14. **<modules>**
15. **<module>**service\_cmn\_client**</module>**
16. **</modules>**
18. **<dependencies>**
19. **<dependency>**
20. **<groupId>**com.xlh**</groupId>**
21. **<artifactId>**common\_util**</artifactId>**
22. **</dependency>**
23. **<dependency>**
24. **<groupId>**com.xlh**</groupId>**
25. **<artifactId>**model**</artifactId>**
26. **</dependency>**
27. **<dependency>**
28. **<groupId>**org.springframework.boot**</groupId>**
29. **<artifactId>**spring-boot-starter-web**</artifactId>**
30. **<scope>**provided**</scope>**
31. **</dependency>**
33. <!-- 服务调用feign -->
34. **<dependency>**
35. **<groupId>**org.springframework.cloud**</groupId>**
36. **<artifactId>**spring-cloud-starter-openfeign**</artifactId>**
37. **<scope>**provided**</scope>**
38. **</dependency>**
39. **</dependencies>**
40. **</project>**

2）创建service-cmn-client子模块

3）添加Feign接口类

1. **package** com.xlh.yygh.cmn.client;
3. **import** org.springframework.cloud.openfeign.FeignClient;
4. **import** org.springframework.stereotype.Repository;
5. **import** org.springframework.web.bind.annotation.GetMapping;
6. **import** org.springframework.web.bind.annotation.PathVariable;

9. @FeignClient("service-cmn")
10. @Repository
11. **public** **interface** DictFeignClient {
12. //根据dictcode和value查询name
13. @GetMapping("/admin/cmn/dict/getName/{dictCode}/{value}")
14. **public** String getName(@PathVariable("dictCode") String dictCode, @PathVariable("value") String value);
16. //根据value查询
17. @GetMapping("/admin/cmn/dict/getName/{value}")
18. **public** String getName(@PathVariable("value") String value);
19. }

4）在service父模块中添加依赖

1. <!-- 服务调用feign -->
2. **<dependency>**
3. **<groupId>**org.springframework.cloud**</groupId>**
4. **<artifactId>**spring-cloud-starter-openfeign**</artifactId>**
5. **</dependency>**

### 4、操作service-hosp模块

1）引入依赖

1. **<dependency>**
2. **<groupId>**com.xlh**</groupId>**
3. **<artifactId>**service\_cmn\_client**</artifactId>**
4. **<version>**0.0.1-SNAPSHOT**</version>**
5. **</dependency>**

2）启动类开启服务调用

1. @SpringBootApplication
2. @ComponentScan(basePackages = "com.xlh")
3. @EnableDiscoveryClient
4. @EnableFeignClients("com.xlh")
5. public class ServiceHospApplication {
6. public static void main(String[] args) {
7. SpringApplication.run(ServiceHospApplication.class, args);
8. }
9. }

3）调整service方法，修改HospitalServiceImpl类实现分页

1. //根据条件分页查询
2. @Override
3. public Page**<Hospital>** selectHospPage(int page, int limit, HospitalQueryVo hospitalQueryVo) {
4. //创建pageable对象
5. Pageable pageable=PageRequest.of(page,limit);
6. //创建条件匹配器
7. ExampleMatcher exampleMatcher=ExampleMatcher.matching().withStringMatcher(ExampleMatcher.StringMatcher.CONTAINING).withIgnoreCase(true);
8. //对象类型转换
9. Hospital hospital=new Hospital();
10. BeanUtils.copyProperties(hospitalQueryVo,hospital);
11. //创建example
12. Example**<Hospital>** example=Example.of(hospital,exampleMatcher);
13. //调用方法查询
14. Page**<Hospital>** pages=hospitalRepository.findAll(example,pageable);
16. //遍历集合，将Hospital类本不包含的医院等级属性
17. pages.getContent().stream().forEach(item-**>**{
18. this.setHospitalHostype(item);
19. });
20. return pages;
21. }
23. private Hospital setHospitalHostype(Hospital item) {
24. //医院的dictcode为hostype,根据两个参数查询
25. String hosTypeString=dictFeignClient.getName("Hostype",item.getHostype());
26. //查询省市区
27. String provinceString=dictFeignClient.getName(item.getProvinceCode());
28. String cityString=dictFeignClient.getName(item.getCityCode());
29. String districtString=dictFeignClient.getName(item.getDistrictCode());
31. //添加参数
32. //医院级别：三甲、二甲
33. item.getParam().put("hostypeString",hosTypeString);
34. //地址全称
35. item.getParam().put("fullAddress",provinceString+cityString+districtString);
37. return item;
38. }

### 5、添加数据字典显示接口

1）编写controller

1. //根据dictCode查询下级节点
2. @ApiOperation("根据dictCode查询下级节点")
3. @GetMapping("/findByDictCode/{dictCode}")
4. public Result findByDictCode(@PathVariable String dictCode){
5. List**<Dict>** list=dictService.findByDictCode(dictCode);
6. return Result.ok(list);
7. }

2）编写service

根据dicode查询下层节点：

1. //允许缓存
2. @Override
3. @Cacheable(value = "dict",keyGenerator = "keyGenerator")
4. public List**<Dict>** findChildData(long id) {
5. QueryWrapper**<Dict>** wrapper=new QueryWrapper**<>**();
6. wrapper.eq("parent\_id",id);
7. List**<Dict>** dictList=baseMapper.selectList(wrapper);
8. for(Dict dict:dictList){
9. Long dictdictId=dict.getId();
10. boolean isChild=this.hasChildren(dictId);
11. dict.setHasChildren(isChild);
12. }
13. return dictList;
14. }
15. private Dict getDictByDictCode(String dictCode){
16. QueryWrapper**<Dict>** queryWrapper=new QueryWrapper**<>**();
17. queryWrapper.eq("dict\_code",dictCode);
18. return baseMapper.selectOne(queryWrapper);
19. }
21. @Override
22. public List**<Dict>** findByDictCode(String dictCode) {
23. //根据dict获取对应行的数据
24. Dict dict=this.getDictByDictCode(dictCode);
25. //根据id获取子节点
26. List**<Dict>** list=this.findChildData(dict.getId());
27. return list;
28. }

### 6、添加前端代码

### 7、更新医院上线状态

### 8、添加医院详情

## 三、排班管理

### 1、界面效果



排班分成三部分显示：

1、科室信息（大科室与小科室树形展示）

2、排班日期，分页显示，根据上传排班数据聚合统计产生

3、排班日期对应的就诊医生信息

### 2、接口分析

1. 科室数据使用Element-ui el-tree组件渲染展示，需要将医院上传的科室数据封装成两层父子级数据；
2. 聚合所有排班数据，按日期分页展示，并统计号源数据展示；
3. 根据排班日期获取排班详情数据

## 四、排班管理具体实现

### 1、实现科室列表的前后端

### 2、排班日期分页前后端

### 3、根据排班日期获取排班详情列表