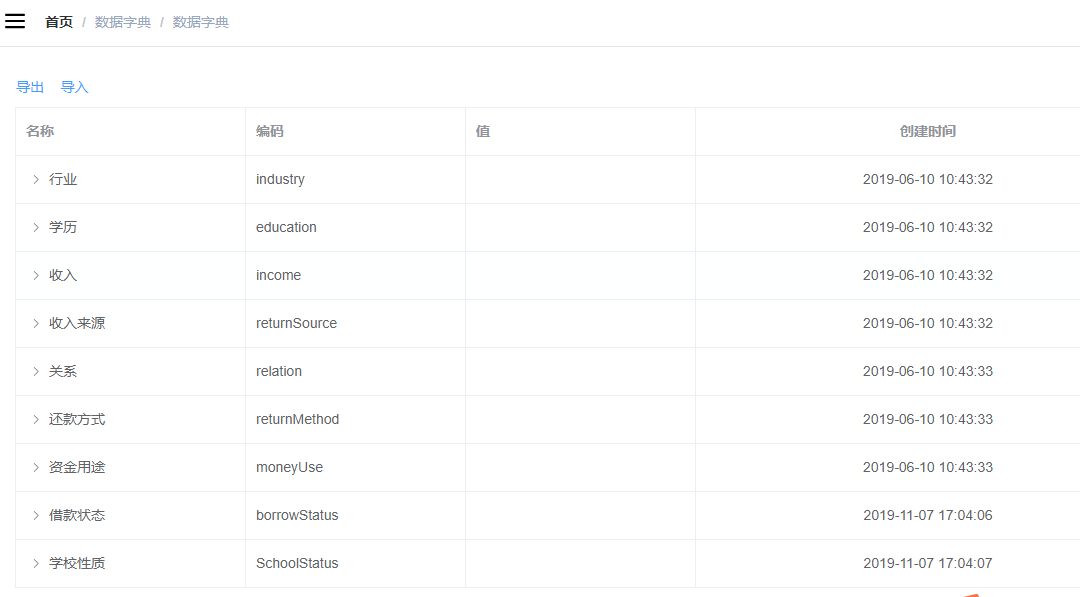
数据字典

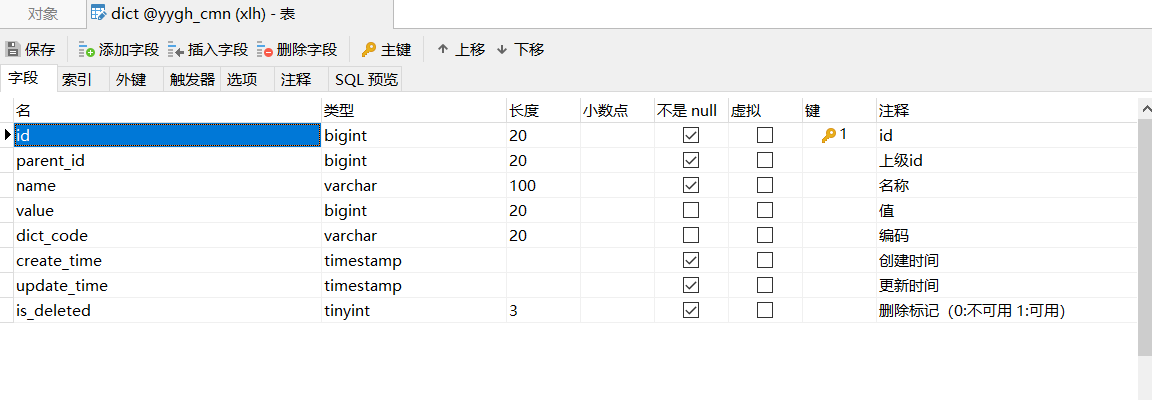
## 一、简介

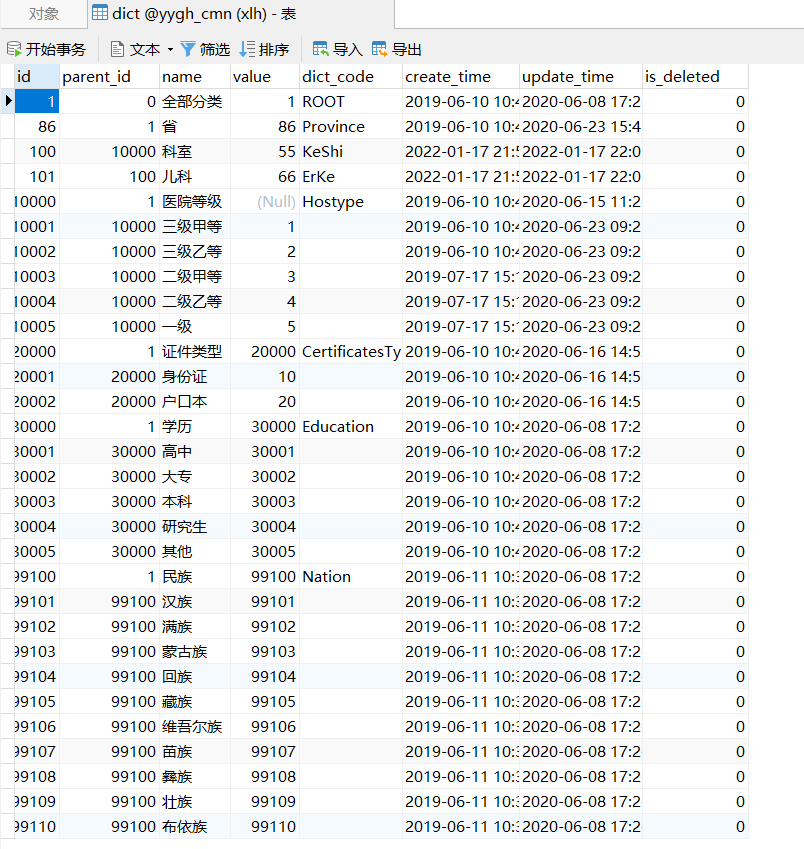
数据字典就是管理系统常用的分类数据或者一些固定数据，例如：省市区三级联动数据、民族数据、行业数据、学历数据等，由于该系统大量使用这种数据，所以我们要做一个数据管理方便管理系统数据，一般系统基本都会做数据管理。

## 二、页面效果



## 三、数据表设置





**说明：**

**parent\_id：**

上级id，通过id与parent\_id构建上下级关系，例如：我们要获取所有行业数据，那么只需要查询parent\_id=20000的数据

name：名称，例如：填写用户信息，我们要select标签选择民族，“汉族”就是数据字典的名称

value：值，例如：填写用户信息，我们要select标签选择民族，“1”（汉族的标识）就是数据字典的值

dict\_code：编码，编码是我们自定义的，全局唯一，例如：我们要获取行业数据，我们可以通过parent\_id获取，但是parent\_id是不确定的，所以我们可以根据编码来获取行业数据

说明：系统中会使用省市区三级联动数据，该数据我们来自“国家统计局”官方数据，地址：<http://www.stats.gov.cn/tjsj/tjbz/tjyqhdmhcxhfdm/2019/index.html>

## 四、页面设计

数据字典是树形展示，由于数据众多，我们使用“树形数据与懒加载”的方式展现数据列表，其他就是对数据的新增、修改与删除操作，因此需要提供的接口如下：

1，根据上级id获取下级数据（构造树形数据），参考文档：https://element.eleme.cn/#/zh-CN/component/table，页面搜索：树形数据与懒加载

2，导入接口

3，导出接口

## 五、数据字典开发

### 1、创建service-cmn模块

**Pom文件：**

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>service</artifactId>
7. <groupId>com.xlh</groupId>
8. <version>0.0.1-SNAPSHOT</version>
9. </parent>
10. <modelVersion>4.0.0</modelVersion>
12. <artifactId>service\_cmn</artifactId>
13. <dependencies>
14. <dependency>
15. <groupId>com.xlh</groupId>
16. <artifactId>model</artifactId>
17. <version>0.0.1-SNAPSHOT</version>
18. </dependency>
20. <!-- https://mvnrepository.com/artifact/com.alibaba/easyexcel -->
21. <dependency>
22. <groupId>com.alibaba</groupId>
23. <artifactId>easyexcel</artifactId>
24. <version>2.1.1</version>
25. </dependency>
27. </dependencies>
29. </project>

**application.properties文件：**

1. # 服务端口
2. server.port=8202
3. # 服务名
4. spring.application.name=service-cmn
6. # 环境设置：dev、test、prod
7. spring.profiles.active=dev
9. # mysql数据库连接
10. spring.datasource.driver-**class**-name=com.mysql.jdbc.Driver
11. spring.datasource.url=jdbc:mysql://localhost:3306/yygh\_cmn?characterEncoding=utf-8&useSSL=false
12. spring.datasource.username=root
13. spring.datasource.password=root
15. #返回json的全局时间格式
16. spring.jackson.date-format=yyyy-MM-dd HH:mm:ss
17. spring.jackson.time-zone=GMT+8
19. #redisw配置
20. spring.redis.host=192.168.243.129
21. spring.redis.port=6379
22. spring.redis.database= 0
23. spring.redis.timeout=1800000
25. spring.redis.lettuce.pool.max-active=20
26. spring.redis.lettuce.pool.max-wait=-1
27. #最大阻塞等待时间(负数表示没限制)
28. spring.redis.lettuce.pool.max-idle=5
29. spring.redis.lettuce.pool.min-idle=0

32. #配置mapper xml文件的路径
33. #mybatis-plus.mapper-locations=classpath:com/xlh/yygh/mapper/xml/\*.xml
34. #mybatis-plus.mapper-locations=classpath:com/xlh/yygh/mapper/xml/\*.xml
35. # nacos服务地址
36. #spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
38. #开启sentinel
39. #feign.sentinel.enabled=true
40. #设置sentinel地址
41. #spring.cloud.sentinel.transport.dashboard=http://127.0.0.1:8858
43. #mongodb地址
44. #spring.data.mongodb.host=192.168.44.163
45. #spring.data.mongodb.port=27017
46. #spring.data.mongodb.database=yygh\_hosp
48. #rabbitmq地址
49. #spring.rabbitmq.host=127.0.0.1
50. #spring.rabbitmq.port=5672
51. #spring.rabbitmq.username=guest
52. #spring.rabbitmq.password=guest

### 2、数据字典列表接口

创建实体类(必须包含hasChildren属性)

1. package com.xlh.yygh.model.cmn;
3. import com.xlh.yygh.model.base.BaseEntity;
4. import com.baomidou.mybatisplus.annotation.\*;
5. import com.fasterxml.jackson.annotation.JsonFormat;
6. import io.swagger.annotations.ApiModel;
7. import io.swagger.annotations.ApiModelProperty;
8. import lombok.Data;
10. import java.util.Date;
11. import java.util.HashMap;
12. import java.util.Map;
14. /\*\*
15. \* <p>
16. \* Dict
17. \* </p>
18. \*
19. \* @author qy
20. \*/
21. @Data
22. @ApiModel(description = "数据字典")
23. @TableName("dict")
24. **public** **class** Dict {
26. **private** **static** final **long** serialVersionUID = 1L;
28. @ApiModelProperty(value = "id")
29. **private** Long id;
31. @ApiModelProperty(value = "创建时间")
32. @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")
33. @TableField("create\_time")
34. **private** Date createTime;
36. @ApiModelProperty(value = "更新时间")
37. @TableField("update\_time")
38. **private** Date updateTime;
40. @ApiModelProperty(value = "逻辑删除(1:已删除，0:未删除)")
41. @TableLogic
42. @TableField("is\_deleted")
43. **private** Integer isDeleted;
45. @ApiModelProperty(value = "其他参数")
46. @TableField(exist = **false**)
47. **private** Map<String,Object> param = **new** HashMap<>();
49. @ApiModelProperty(value = "上级id")
50. @TableField("parent\_id")
51. **private** Long parentId;
53. @ApiModelProperty(value = "名称")
54. @TableField("name")
55. **private** String name;
57. @ApiModelProperty(value = "值")
58. @TableField("value")
59. **private** String value;
61. @ApiModelProperty(value = "编码")
62. @TableField("dict\_code")
63. **private** String dictCode;
65. @ApiModelProperty(value = "是否包含子节点")
66. @TableField(exist = **false**)
67. **private** boolean hasChildren;
69. }

### 3、添加数据字典mapper

1. package com.xlh.yygh.cmn.mapper;
3. import com.baomidou.mybatisplus.core.mapper.BaseMapper;
4. import com.xlh.yygh.model.cmn.Dict;
6. **public** interface DictMapper extends BaseMapper<Dict> {
8. }

### 4、添加数据字典service（需使用EasyExcel）

1. package com.xlh.yygh.cmn.service.impl;
3. import com.alibaba.excel.EasyExcel;
4. import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
5. import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
6. import com.fasterxml.jackson.databind.util.BeanUtil;
7. import com.xlh.yygh.cmn.listener.DictListener;
8. import com.xlh.yygh.cmn.mapper.DictMapper;
9. import com.xlh.yygh.cmn.service.DictService;
10. import com.xlh.yygh.common.result.Result;
11. import com.xlh.yygh.enums.DictEnum;
12. import com.xlh.yygh.model.cmn.Dict;
13. import com.xlh.yygh.vo.cmn.DictEeVo;
14. import org.springframework.beans.BeanUtils;
15. import org.springframework.cache.annotation.CacheEvict;
16. import org.springframework.cache.annotation.Cacheable;
17. import org.springframework.stereotype.Service;
18. import org.springframework.web.multipart.MultipartFile;
20. import javax.servlet.http.HttpServletResponse;
21. import java.io.IOException;
22. import java.util.ArrayList;
23. import java.util.List;
25. @Service
26. **public** **class** DictServiceImpl extends ServiceImpl<DictMapper, Dict> implements DictService {
28. //允许缓存
29. @Override
30. @Cacheable(value = "dict",keyGenerator = "keyGenerator")
31. **public** List<Dict> findChildData(**long** id) {
32. QueryWrapper<Dict> wrapper=**new** QueryWrapper<>();
33. wrapper.eq("parent\_id",id);
34. List<Dict> dictList=baseMapper.selectList(wrapper);
35. **for**(Dict dict:dictList){
36. Long dictId=dict.getId();
37. boolean isChild=**this**.hasChildren(dictId);
38. dict.setHasChildren(isChild);
39. }
40. **return** dictList;
41. }
43. //判断id下是否有子节点
44. **private** boolean hasChildren(**long** id){
45. QueryWrapper<Dict> wrapper=**new** QueryWrapper<>();
46. wrapper.eq("parent\_id",id);
47. Integer count=baseMapper.selectCount(wrapper);
48. **return** count>0;
49. }
51. //导出数据
52. @Override
53. **public** **void** exportDictData(HttpServletResponse httpServletResponse) {
54. //设置下载信息
55. httpServletResponse.setContentType("application/vnd.ms-excel");
56. httpServletResponse.setCharacterEncoding("utf-8");
57. String filename="dict";
58. httpServletResponse.setHeader("Content-disposition","attachment;filename="+filename+".xlsx");
59. //查询数据库
60. List<Dict> dictList=baseMapper.selectList(null);
61. //将Dict转为DictEeVo
62. List<DictEeVo> dictEeVos=**new** ArrayList<>();
63. **for**(Dict dict:dictList){
64. DictEeVo dictEeVo=**new** DictEeVo();
65. BeanUtils.copyProperties(dict,dictEeVo);
66. dictEeVos.add(dictEeVo);
67. }
68. //写操作
69. **try** {
70. EasyExcel.write(httpServletResponse.getOutputStream(), DictEeVo.**class**).sheet("dict").doWrite(dictEeVos);
71. }**catch** (IOException e){
72. e.printStackTrace();
73. }
74. }
76. //导入数据
77. @Override
78. @CacheEvict(value = "dict", allEntries=**true**)
79. **public** **void** importDictData(MultipartFile file) {
80. **try** {
81. EasyExcel.read(file.getInputStream(), DictEeVo.**class**,**new** DictListener(baseMapper)).sheet().doRead();
82. }
83. **catch** (IOException e){
84. e.printStackTrace();
85. }
86. }
87. }

### 5、添加数据字典controller

## 六、Spring Cache + Redis 缓存数据

### 1、service-util中添加依赖

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>common</artifactId>
7. <groupId>com.xlh</groupId>
8. <version>0.0.1-SNAPSHOT</version>
9. </parent>
10. <modelVersion>4.0.0</modelVersion>
12. <artifactId>service\_util</artifactId>
14. <dependencies>
15. <!-- redis -->
16. <dependency>
17. <groupId>org.springframework.boot</groupId>
18. <artifactId>spring-boot-starter-data-redis</artifactId>
19. </dependency>
21. <!-- spring2.X集成redis所需common-pool2-->
22. <dependency>
23. <groupId>org.apache.commons</groupId>
24. <artifactId>commons-pool2</artifactId>
25. <version>2.6.0</version>
26. </dependency>
27. </dependencies>

30. </project>

### 2、service-util添加配置类

1. package com.xlh.yygh.common.config;
3. import com.fasterxml.jackson.annotation.JsonAutoDetect;
4. import com.fasterxml.jackson.annotation.PropertyAccessor;
5. import com.fasterxml.jackson.databind.ObjectMapper;
6. import org.springframework.cache.CacheManager;
7. import org.springframework.cache.annotation.EnableCaching;
8. import org.springframework.cache.interceptor.KeyGenerator;
9. import org.springframework.context.annotation.Bean;
10. import org.springframework.context.annotation.Configuration;
11. import org.springframework.data.redis.cache.RedisCacheConfiguration;
12. import org.springframework.data.redis.cache.RedisCacheManager;
13. import org.springframework.data.redis.connection.RedisConnectionFactory;
14. import org.springframework.data.redis.core.RedisTemplate;
15. import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;
16. import org.springframework.data.redis.serializer.RedisSerializationContext;
17. import org.springframework.data.redis.serializer.RedisSerializer;
18. import org.springframework.data.redis.serializer.StringRedisSerializer;
20. import java.lang.reflect.Method;
21. import java.time.Duration;
23. //标明配置类
24. @Configuration
25. //允许缓存
26. @EnableCaching
27. **public** **class** RedisConfig {
28. /\*\*
29. \* 自定义key规则
30. \* @return
31. \*/
32. @Bean
33. **public** KeyGenerator keyGenerator() {
34. **return** **new** KeyGenerator() {
35. @Override
36. **public** Object generate(Object target, Method method, Object... params) {
37. StringBuilder sb = **new** StringBuilder();
38. sb.append(target.getClass().getName());
39. sb.append(method.getName());
40. **for** (Object obj : params) {
41. sb.append(obj.toString());
42. }
43. **return** sb.toString();
44. }
45. };
46. }
48. /\*\*
49. \* 设置RedisTemplate规则
50. \* @param redisConnectionFactory
51. \* @return
52. \*/
53. @Bean
54. **public** RedisTemplate<Object, Object> redisTemplate(RedisConnectionFactory redisConnectionFactory) {
55. RedisTemplate<Object, Object> redisTemplate = **new** RedisTemplate<>();
56. redisTemplate.setConnectionFactory(redisConnectionFactory);
57. Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = **new** Jackson2JsonRedisSerializer(Object.**class**);
59. //解决查询缓存转换异常的问题
60. ObjectMapper om = **new** ObjectMapper();
61. // 指定要序列化的域，field,get和set,以及修饰符范围，ANY是都有包括private和public
62. om.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
63. // 指定序列化输入的类型，类必须是非final修饰的，final修饰的类，比如String,Integer等会跑出异常
64. om.enableDefaultTyping(ObjectMapper.DefaultTyping.NON\_FINAL);
65. jackson2JsonRedisSerializer.setObjectMapper(om);
67. //序列号key value
68. redisTemplate.setKeySerializer(**new** StringRedisSerializer());
69. redisTemplate.setValueSerializer(jackson2JsonRedisSerializer);
70. redisTemplate.setHashKeySerializer(**new** StringRedisSerializer());
71. redisTemplate.setHashValueSerializer(jackson2JsonRedisSerializer);
73. redisTemplate.afterPropertiesSet();
74. **return** redisTemplate;
75. }
77. /\*\*
78. \* 设置CacheManager缓存规则
79. \* @param factory
80. \* @return
81. \*/
82. @Bean
83. **public** CacheManager cacheManager(RedisConnectionFactory factory) {
84. RedisSerializer<String> redisSerializer = **new** StringRedisSerializer();
85. Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = **new** Jackson2JsonRedisSerializer(Object.**class**);
87. //解决查询缓存转换异常的问题
88. ObjectMapper om = **new** ObjectMapper();
89. om.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
90. om.enableDefaultTyping(ObjectMapper.DefaultTyping.NON\_FINAL);
91. jackson2JsonRedisSerializer.setObjectMapper(om);
93. // 配置序列化（解决乱码的问题）,过期时间600秒
94. RedisCacheConfiguration config = RedisCacheConfiguration.defaultCacheConfig()
95. .entryTtl(Duration.ofSeconds(600))
96. .serializeKeysWith(RedisSerializationContext.SerializationPair.fromSerializer(redisSerializer))
97. .serializeValuesWith(RedisSerializationContext.SerializationPair.fromSerializer(jackson2JsonRedisSerializer))
98. .disableCachingNullValues();
100. RedisCacheManager cacheManager = RedisCacheManager.builder(factory)
101. .cacheDefaults(config)
102. .build();
103. **return** cacheManager;
104. }
106. }

**说明：**

@EnableCaching：标记注解 @EnableCaching，开启缓存，并配置Redis缓存管理器。@EnableCaching 注释触发后置处理器, 检查每一个Spring bean 的 public 方法是否存在缓存注解。如果找到这样的一个注释, 自动创建一个代理拦截方法调用和处理相应的缓存行为。

### 3、service-cmn添加redis配置

1. #redis配置
2. spring.redis.host=192.168.243.129
3. spring.redis.port=6379
4. spring.redis.database= 0
5. spring.redis.timeout=1800000

### 4、常用缓存介绍

#### 4.1 @Cacheable

根据方法对其返回结果进行缓存，下次请求时，如果缓存存在，则直接读取缓存数据返回；如果缓存不存在，则执行方法，并把返回的结果存入缓存中。一般用在查询方法上。

#### 4.2 @CachePut

使用该注解标志的方法，每次都会执行，并将结果存入指定的缓存中。其他方法可以直接从响应的缓存中读取缓存数据，而不需要再去查询数据库。一般用在新增方法上。

#### 4.3 @CacheEvict

使用该注解标志的方法，会清空指定的缓存。一般用在更新或者删除方法上。

## 七、配置nginx

由于我们后端有很多服务模块，每个模块都有对应的访问路径与端口，为了提供统一的api接口，所以使用nginx作为反向代理服务器；

反向代理，其实客户端对代理是无感知的，因为客户端不需要任何配置就可以访问，我们只需要将请求发送到反向代理服务器，由反向代理服务器去选择目标服务器获取数据后，在返回给客户端，此时反向代理服务器和目标服务器对外就是一个服务器，暴露的是代理服务器地址，隐藏了真实服务器IP地址。

1. server {
2. listen       9001;
3. server\_name  localhost;
5. location ~ /hosp/ {
6. proxy\_pass http://localhost:8201;
7. }
8. location ~ /cmn/ {
9. proxy\_pass http://localhost:8202;
10. }
11. }