# day01 项目介绍与MyBatis Plus

## 尚医通介绍

### 核心技术

SpringBoot：简化新Spring应用的初始搭建以及开发过程

SpringCloud：基于Spring Boot实现的云原生应用开发工具，SpringCloud使用的技术：（SpringCloudGateway、Spring Cloud Alibaba Nacos、Spring Cloud Alibaba Sentinel、SpringCloud Task和SpringCloudFeign等）

MyBatis-Plus：持久层框架

Redis：内存缓存

RabbitMQ：消息中间件

HTTPClient: Http协议客户端

Swagger2：Api接口文档工具

Nginx：负载均衡

Lombok

Mysql：关系型数据库

MongoDB：面向文档的NoSQL数据库

Vue.js：web 界面的渐进式框架

Node.js： JavaScript 运行环境

Axios：Axios 是一个基于 promise 的 HTTP 库

NPM：包管理器

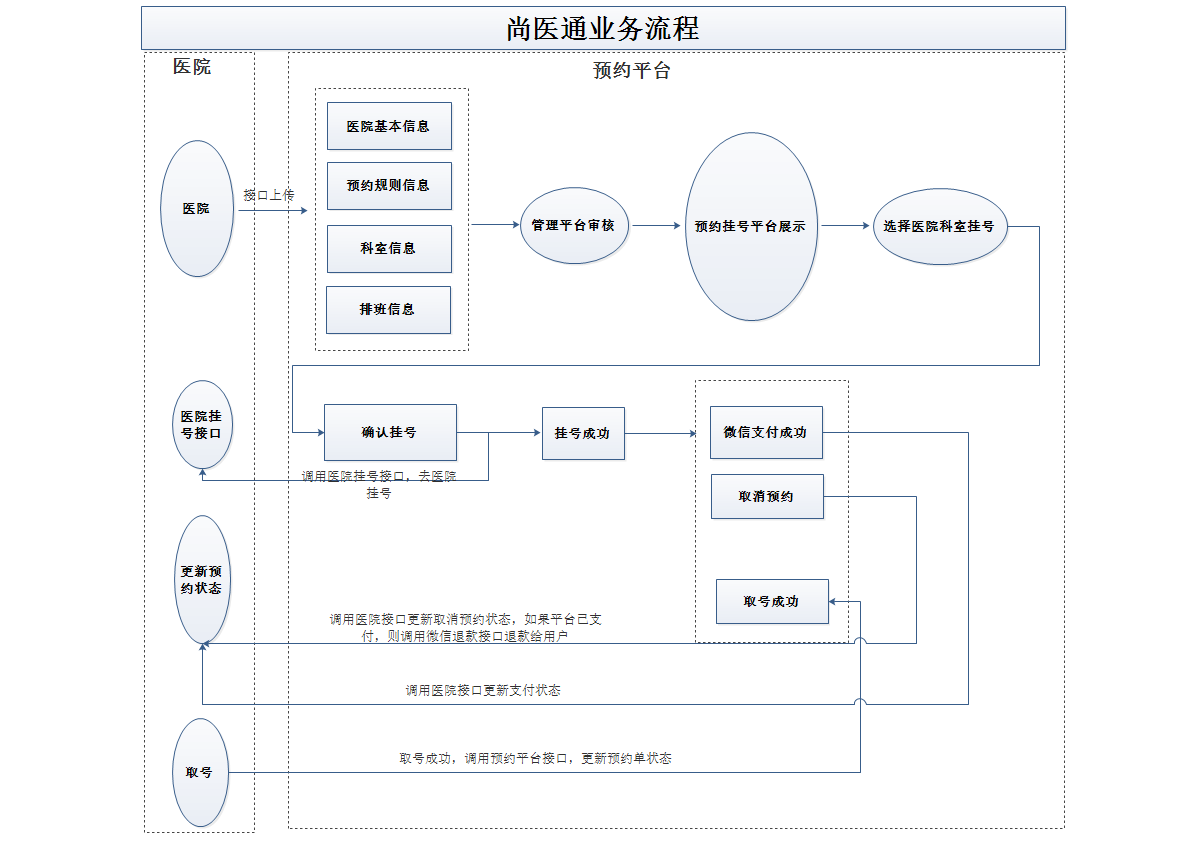
Babel：转码器

Webpack：打包工具

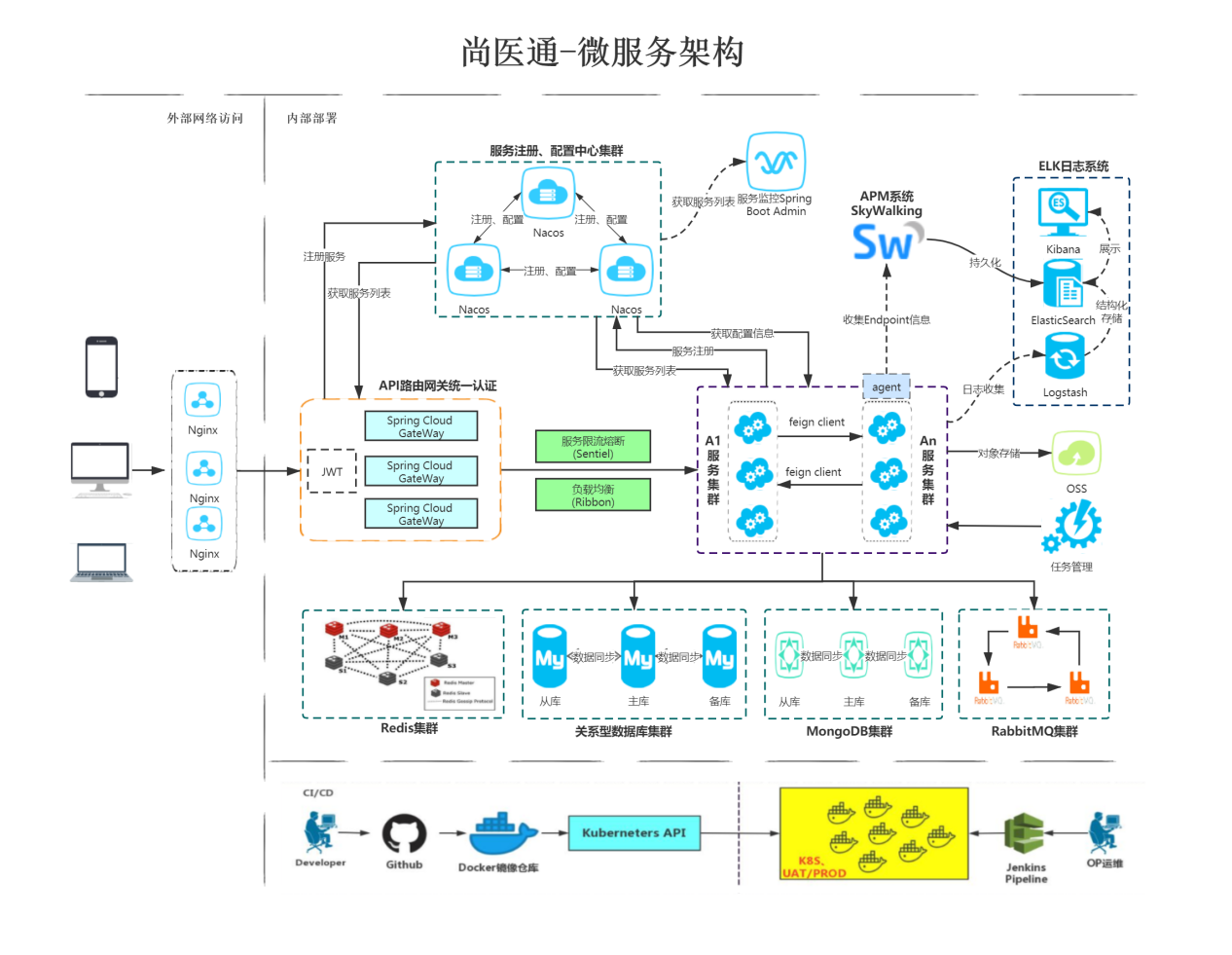
Docker ：容器技术

Git：代码管理工具

### 业务流程



### 服务架构



## MyBatis-Plus入门

### 简介

MyBatis-Plus（简称 MP）是一个 MyBatis 的增强工具，在 MyBatis 的基础上只做增强不做改变，为简化开发、提高效率而生。

1）润物无声

只做增强不做改变，引入它不会对现有工程产生影响，如丝般顺滑。

2）效率至上

只需简单配置，即可快速进行 CRUD 操作，从而节省大量时间。

3）丰富功能

热加载、代码生成、分页、性能分析等功能一应俱全。

### 创建并初始化数据库

#### 2.1 创建数据库

mybatis\_plus

#### 2.2 创建User表

其对应的数据库 Schema 脚本如下：

CREATE TABLE USER

(

id BIGINT(20)NOT NULL COMMENT '主键ID',

NAME VARCHAR(30)NULL DEFAULT NULL COMMENT '姓名',

age INT(11)NULL DEFAULT NULL COMMENT '年龄',

email VARCHAR(50)NULL DEFAULT NULL COMMENT '邮箱',

PRIMARY KEY (id)

);

其对应的数据库 Data 脚本如下：

INSERT INTO user (id, name, age, email)VALUES

(1, 'Jone', 18, 'test1@baomidou.com'),

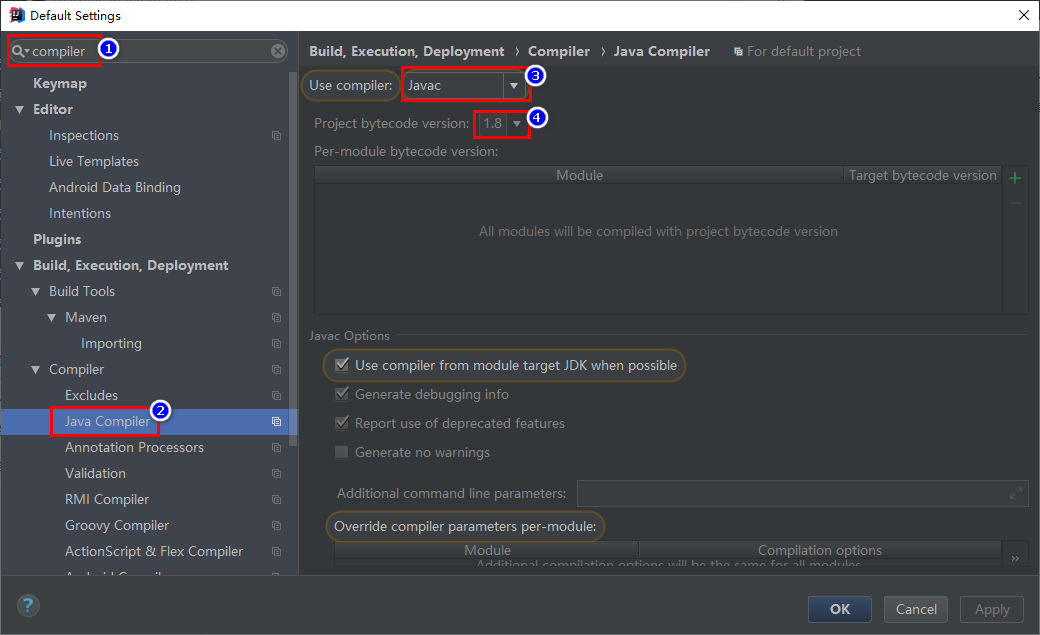
(2, 'Jack', 20, 'test2@baomidou.com'),

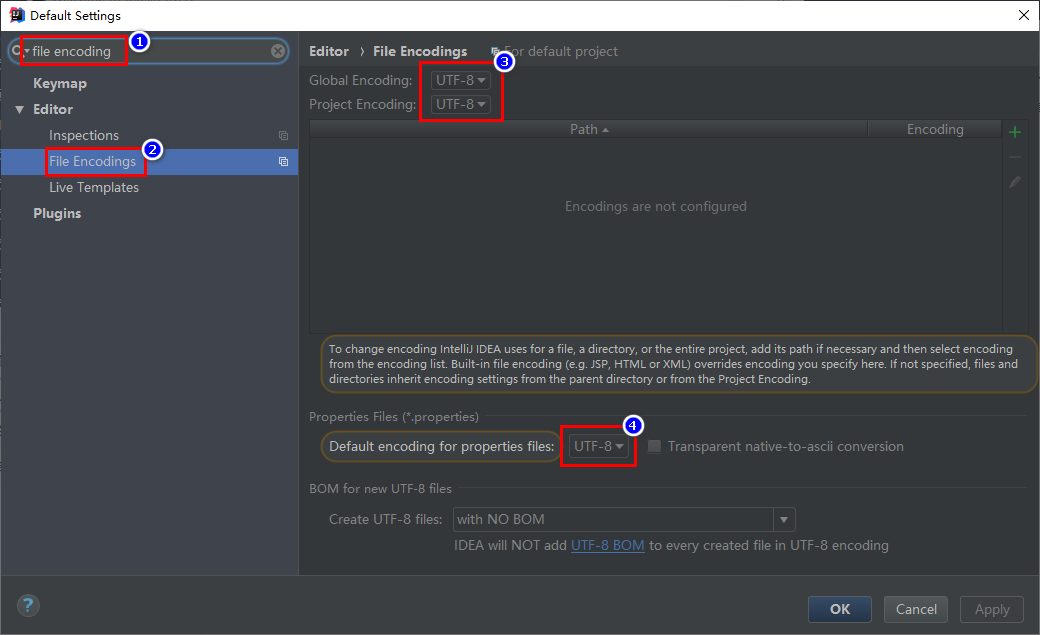
(3, 'Tom', 28, 'test3@baomidou.com'),

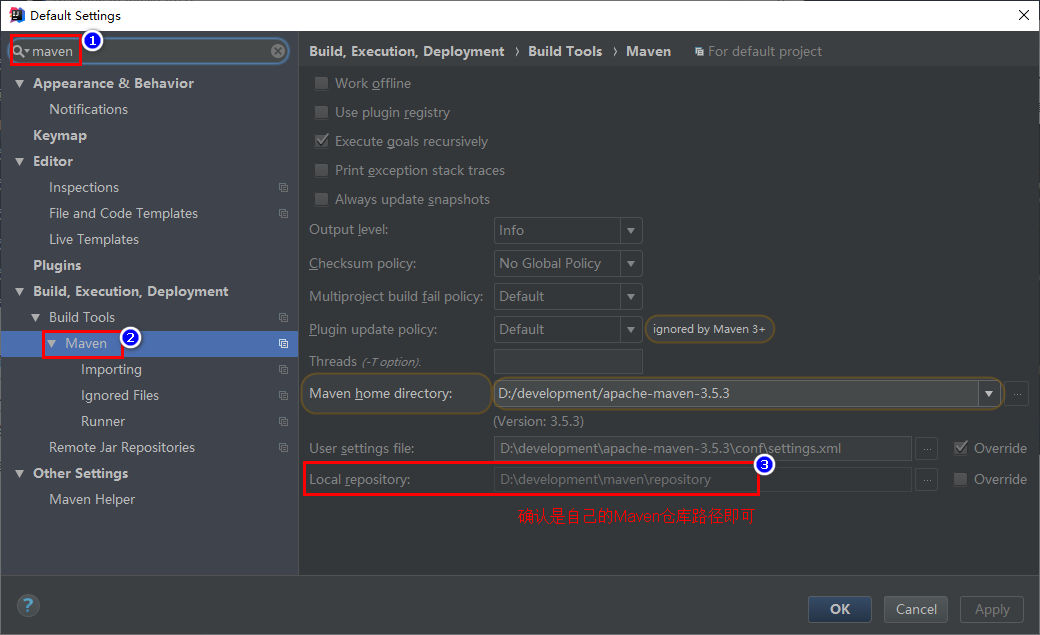
(4, 'Sandy', 21, 'test4@baomidou.com'),

(5, 'Billie', 24, 'test5@baomidou.com');

### 3、确认idea配置







### 4、创建项目

#### 4.1 初始化工程

使用 Spring Initializr 快速初始化一个 Spring Boot 工程

Group：com.atguigu

Artifact：mybatis\_plus

版本：2.2.1.RELEASE

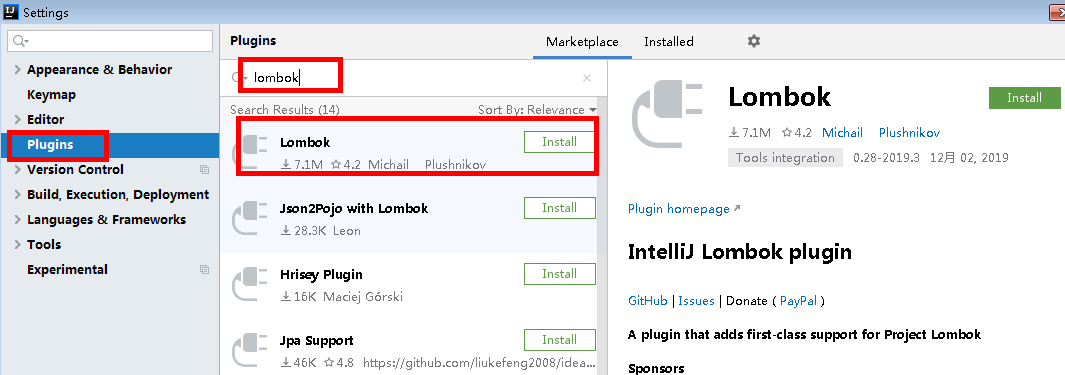
#### 4.2 引入依赖

注意：引入 MyBatis-Plus 之后请不要再次引入 MyBatis，以避免因版本差异导致的问题

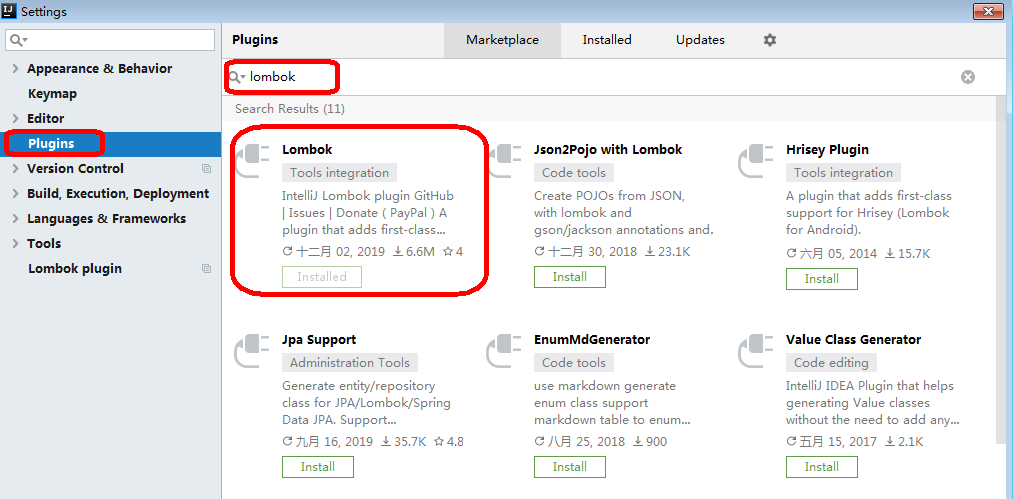
<**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter</**artifactId**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-test</**artifactId**>  
 <**scope**>test</**scope**>  
 <**exclusions**>  
 <**exclusion**>  
 <**groupId**>org.junit.vintage</**groupId**>  
 <**artifactId**>junit-vintage-engine</**artifactId**>  
 </**exclusion**>  
 </**exclusions**>  
 </**dependency**>  
  
 *<!--mybatis-plus-->* <**dependency**>  
 <**groupId**>com.baomidou</**groupId**>  
 <**artifactId**>mybatis-plus-boot-starter</**artifactId**>  
 <**version**>3.3.1</**version**>  
 </**dependency**>  
  
 *<!--mysql依赖-->* <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 </**dependency**>  
 *<!--lombok用来简化实体类-->* <**dependency**>  
 <**groupId**>org.projectlombok</**groupId**>  
 <**artifactId**>lombok</**artifactId**>  
 <**optional**>true</**optional**>  
 </**dependency**>  
  
</**dependencies**>

#### 4.3 idea中安装lombok插件

1）idea2019版本



2）idea2018版本



### 5、编写代码

#### 5.1 配置

在application.properties 配置文件中添加 MySQL 数据库的相关配置：

spring boot 2.0（内置jdbc5驱动）

#mysql数据库连接

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/mybatis\_plus?characterEncoding=utf-8&useSSL=false

spring.datasource.username=root

spring.datasource.password=root

spring boot 2.1及以上（内置jdbc8驱动）

注意：driver和url的变化

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/mybatis\_plus?serverTimezone=GMT%2B8

spring.datasource.username=root

spring.datasource.password=root

**注意：**

1、这里的 url 使用了 ?serverTimezone=GMT%2B8 后缀，因为8.0版本的jdbc驱动需要添加这个后缀，否则运行测试用例报告如下错误：

java.sql.SQLException: The server time zone value 'ÖÐ¹ú±ê×¼Ê±¼ä' is unrecognized or represents more

2、这里的 driver-class-name使用了com.mysql.cj.jdbc.Driver ，在 jdbc 8 中 建议使用这个驱动，否则运行测试用例的时候会有 WARN 信息

#### 5.2 启动类

在 Spring Boot 启动类中添加 @MapperScan 注解，扫描 Mapper文件夹

@SpringBootApplication

@MapperScan("com.atguigu.demomptest.mapper")

public class DemomptestApplication {

public static void main(String[] args) {

SpringApplication.run(DemomptestApplication.class, args);

}

}

#### 5.3 添加实体

创建包 entity 编写实体类 User.java（此处使用了 Lombok 简化代码）

@Data

public class User {

private Long id;

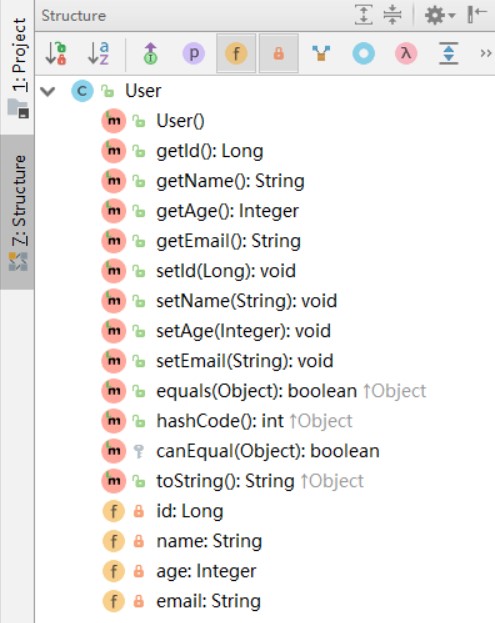
private String name;

private Integer age;

private String email;

}

查看编译结果



#### 5.4 添加mapper

创建包mapper，编写Mapper接口：UserMapper.java

@Repository

public interface UserMapper extends BaseMapper<User> {

}

#### 5.5 测试

添加测试类，进行功能测试：

@SpringBootTest

class DemomptestApplicationTests {

@Autowired

private UserMapper userMapper;

@Test

public void findAll() {

List<User> users = userMapper.selectList(null);

System.out.println(users);

}

}

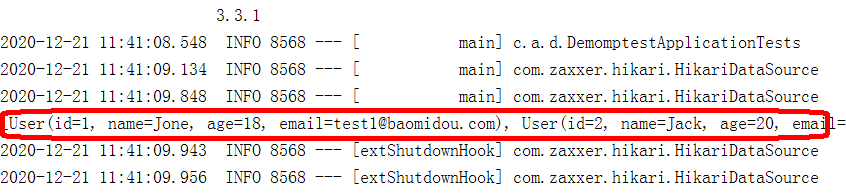
注意：

IDEA在 userMapper 处报错，因为找不到注入的对象，因为类是动态创建的，但是程序可以正确的执行。

为了避免报错，可以在 dao 层 的接口上添加 @Repository 注

通过以上几个简单的步骤，我们就实现了 User 表的 CRUD 功能，甚至连 XML 文件都不用编写！

查看控制台输出：



#### 5.6 查看sql输出日志

application.properties中配置：

#mybatis日志

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

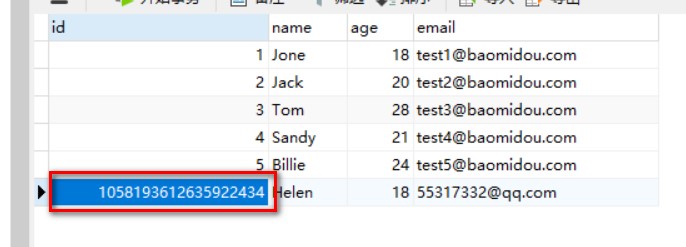
## 主键策略

### 添加操作

@SpringBootTest  
**class** DemomptestApplicationTests {  
  
 @Autowired  
 **private** UserMapper **userMapper**;  
  
 *//添加* @Test  
 **public void** testAdd() {  
 User user = **new** User();  
 user.setName(**"王五"**);  
 user.setAge(20);  
 user.setEmail(**"1243@qq.com"**);  
 **int** insert = **userMapper**.insert(user);  
 System.***out***.println(insert);  
 }

}

注意：数据库插入id值默认为：全局唯一id



### MP的主键策略

#### 2.1 ASSIGN\_ID

MyBatis-Plus默认的主键策略是：ASSIGN\_ID（使用了雪花算法生成19位的值）

@TableId(type = IdType.ASSIGN\_ID)

private String id;

**雪花算法：分布式ID生成器**

雪花算法是由Twitter公布的分布式主键生成算法，它能够保证不同表的主键的不重复性，以及相同表的主键的有序性。

**核心思想：**

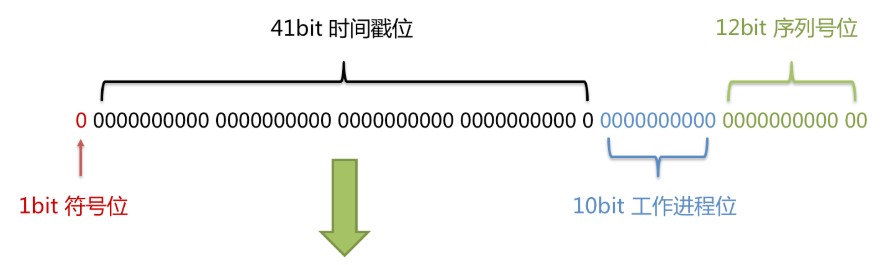
长度共64bit（一个long型）。

首先是一个符号位，1bit标识，由于long基本类型在Java中是带符号的，最高位是符号位，正数是0，负数是1，所以id一般是正数，最高位是0。

41bit时间截(毫秒级)，存储的是时间截的差值（当前时间截 - 开始时间截)，结果约等于69.73年。

10bit作为机器的ID（5个bit是数据中心，5个bit的机器ID，可以部署在1024个节点）。

12bit作为毫秒内的流水号（意味着每个节点在每毫秒可以产生 4096 个 ID）。



优点：整体上按照时间自增排序，并且整个分布式系统内不会产生ID碰撞，并且效率较高。

#### 2.2 AUTO 自增策略

需要在创建数据表的时候设置主键自增

实体字段中配置 @TableId(type = IdType.AUTO)

@TableId(type = IdType.AUTO)

private Long id;

要想影响所有实体的配置，可以设置全局主键配置

#全局设置主键生成策略

mybatis-plus.global-config.db-config.id-type=auto

## 自动填充和乐观锁

### 1、更新操作

注意：update时生成的sql自动是动态sql：UPDATE user SET age=? WHERE id=?

@SpringBootTest  
**class** DemomptestApplicationTests {  
  
 @Autowired  
 **private** UserMapper **userMapper**;  
  
 *//修改* @Test  
 **public void** testUpdate() {  
 User user = **new** User();  
 user.setId(1340874606495080450L);  
 user.setName(**"lucymaryupup"**);  
 **int** count = **userMapper**.updateById(user);  
 System.***out***.println(count);  
 }  
}

### 2、自动填充

需求描述：

项目中经常会遇到一些数据，每次都使用相同的方式填充，例如记录的创建时间，更新时间等。

我们可以使用MyBatis Plus的自动填充功能，完成这些字段的赋值工作

#### 2.1数据库修改

在User表中添加datetime类型的新的字段：create\_time、update\_time

#### 2.2实体类修改

实体上增加字段并添加自动填充注解

@Data  
**public class** User {  
 *// @TableId(type = IdType.ASSIGN\_ID)* **private** Long **id**;  
  
 **private** String **name**;  
 **private** Integer **age**;  
 **private** String **email**;  
  
 @TableField(fill = FieldFill.***INSERT***) **//在添加的时候设置值**  
 **private** Date **createTime**; *//create\_time* @TableField(fill = FieldFill.***INSERT\_UPDATE***) **//在添加、修改的时候设置值**  
 **private** Date **updateTime**; *//update\_time* @Version  
 @TableField(fill = FieldFill.***INSERT***)  
 **private** Integer **version**;  
  
 @TableLogic  
 @TableField(fill = FieldFill.***INSERT***)  
 **private** Integer **deleted**;  
}

#### 2.3实现元对象处理器接口

注意：不要忘记添加 @Component 注解

**package** com.atguigu.demomptest.handler;  
  
@Component //交给spring管理  
**public class** MyMetaObjectHandler **implements** MetaObjectHandler {  
  
 *//mp执行添加操作，这个方法执行* @Override  
 **public void** insertFill(MetaObject metaObject) {  
 **this**.setFieldValByName(**"createTime"**,**new** Date(),metaObject);  
 **this**.setFieldValByName(**"updateTime"**,**new** Date(),metaObject);  
 **this**.setFieldValByName(**"version"**,1,metaObject);  
 **this**.setFieldValByName(**"deleted"**,0,metaObject);  
 }  
  
 *//mp执行修改操作，这个方法执行* @Override  
 **public void** updateFill(MetaObject metaObject) {  
 **this**.setFieldValByName(**"updateTime"**,**new** Date(),metaObject);  
 }  
}

### 3、乐观锁

主要适用场景：当要更新一条记录的时候，希望这条记录没有被别人更新，也就是说实现线程安全的数据更新

**乐观锁实现方式：**

* 取出记录时，获取当前version
* 更新时，带上这个version
* 执行更新时， set version = newVersion where version = oldVersion
* 如果version不对，就更新失败

接下来介绍如何在Mybatis-Plus项目中，使用乐观锁：

### 4、乐观锁实现流程

#### 4.1修改实体类

添加 @Version 注解

@Data  
**public class** User {  
@Version  
 @TableField(fill = FieldFill.***INSERT***)  
 **private** Integer **version**;  
}

#### 4.2创建配置文件

创建包config，创建文件MybatisPlusConfig.java

此时可以删除主类中的 @MapperScan 扫描注解

**package** com.atguigu.demomptest.config;  
  
@Configuration //注明这是一个配置类  
@MapperScan(**"com.atguigu.demomptest.mapper"**)  
**public class** MpConfig {  
  
 */\*\*  
 \* 乐观锁插件  
 \*/* @Bean  
 **public** OptimisticLockerInterceptor optimisticLockerInterceptor() {  
 **return new** OptimisticLockerInterceptor();  
 }  
}

**package** com.atguigu.demomptest.handler;  
  
@Component //交给spring管理  
**public class** MyMetaObjectHandler **implements** MetaObjectHandler {  
  
 *//mp执行添加操作，这个方法执行* @Override  
 **public void** insertFill(MetaObject metaObject) {  
  
 **this**.setFieldValByName(**"version"**,1,metaObject);  
 }  
}

#### 4.3注册乐观锁插件

在 MybatisPlusConfig 中注册 Bean

#### 4.4测试乐观锁

@SpringBootTest  
**class** DemomptestApplicationTests {  
  
 @Autowired  
 **private** UserMapper **userMapper**;  
  
 *//测试乐观锁* @Test  
 **public void** testOptimisticLocker() {  
 *//根据id查询* User user = **userMapper**.selectById(1340899349885612034L);  
 *//修改* user.setName(**"张三"**);

*//不需要手动将版本加1，mybatisplus已经自动完成了加1*

*//user.setVersion(user.getVersion() + 1);*  
 **userMapper**.updateById(user);  
 }  
}

## 查询

### 1、通过多个id批量查询

完成了动态sql的foreach的功能

//多个id批量查询

@Test

public void testSelect1() {

List<User> users = userMapper.selectBatchIds(Arrays.asList(1, 2, 3));

System.out.println(users);

}

@SpringBootTest  
**class** DemomptestApplicationTests {  
  
 @Autowired  
 **private** UserMapper **userMapper**;  
  
 *//多个id批量查询* @Test  
 **public void** testSelect1() {  
 List<User> users = **userMapper**.selectBatchIds(Arrays.*asList*(1, 2, 3));  
 System.***out***.println(users);  
 }

}

### 2、简单的条件查询

通过map封装查询条件

注意：map中的key对应数据库中的列名。如：数据库user\_id，实体类是userId，这时map的key需要填写user\_id

@SpringBootTest  
**class** DemomptestApplicationTests {  
  
 @Autowired  
 **private** UserMapper **userMapper**;  
  
 *//简单条件查询* @Test  
 **public void** testSelect2() {  
 Map<String, Object> columnMap = **new** HashMap<>();  
 columnMap.put(**"name"**,**"Jack"**);  
 columnMap.put(**"age"**,20);  
 List<User> users = **userMapper**.selectByMap(columnMap);  
 System.***out***.println(users);  
 }  
}

### 3、分页查询

#### 3.1添加分页插件

MyBatis Plus自带分页插件，只要简单的配置即可实现分页功能。

配置类中添加@Bean配置

**package** com.atguigu.demomptest.config;  
  
@Configuration //注明这是一个配置类  
@MapperScan(**"com.atguigu.demomptest.mapper"**)  
**public class** MpConfig {  
  
 */\*\*  
 \* 分页插件  
 \*/* @Bean  
 **public** PaginationInterceptor paginationInterceptor() {  
 **return new** PaginationInterceptor();  
 }  
}

#### 3.2测试selectPage分页

测试：最终通过page对象获取相关数据

@SpringBootTest  
**class** DemomptestApplicationTests {  
  
 @Autowired  
 **private** UserMapper **userMapper**;  
  
 *//分页查询* @Test  
 **public void** testSelectPage() {

Page<User> page = **new** Page(1,3); *//1：当前页，3：总页数*   
 Page<User> userPage = **userMapper**.selectPage(page, **null**); *//null查询条件*  
 *//返回对象得到分页所有数据* **long** pages = userPage.getPages(); *//总页数* **long** current = userPage.getCurrent(); *//当前页* List<User> records = userPage.getRecords(); *//查询数据集合* **long** total = userPage.getTotal(); *//总记录数* **boolean** hasNext = userPage.hasNext(); *//下一页* **boolean** hasPrevious = userPage.hasPrevious(); *//上一页* System.***out***.println(pages);  
 System.***out***.println(current);  
 System.***out***.println(records);  
 System.***out***.println(total);  
 System.***out***.println(hasNext);  
 System.***out***.println(hasPrevious);  
 }  
}

#### 3.3测试selectMapsPage分页

当指定了特定的查询列时，希望分页结果列表只返回被查询的列，而不是很多null值

测试selectMapsPage分页：结果集是Map

@SpringBootTest  
**class** DemomptestApplicationTests {  
  
 @Autowired  
 **private** UserMapper **userMapper**;  
  
 @Test  
 **public void** testSelectMapsPage() {  
   
 *//Page不需要泛型* Page<Map<String, Object>> page = newPage<>(1, 5);  
 Page<Map<String, Object>> pageParam = **userMapper**.selectMapsPage(page, **null**);  
 List<Map<String, Object>> records = pageParam.getRecords();  
 records.forEach(System.***out***::println);  
 System.***out***.println(pageParam.getCurrent());  
 System.***out***.println(pageParam.getPages());  
 System.***out***.println(pageParam.getSize());  
 System.***out***.println(pageParam.getTotal());  
 System.***out***.println(pageParam.hasNext());  
 System.***out***.println(pageParam.hasPrevious());  
 }

}

## 删除与逻辑删除

### 1、删除

#### 1.1根据id删除记录

@SpringBootTest  
**class** DemomptestApplicationTests {  
  
 @Autowired  
 **private** UserMapper **userMapper**;  
  
 *//根据id删除* @Test  
 **public void** testDeleteId() {  
 **int** rows = **userMapper**.deleteById(1340936420025077762L);  
 System.***out***.println(rows);  
 }  
}

#### 1.2批量删除

@SpringBootTest  
**class** DemomptestApplicationTests {  
  
 @Autowired  
 **private** UserMapper **userMapper**;  
  
 *//批量删除* @Test  
 **public void** testDeleteBatchIds() {  
 **int** result = **userMapper**.deleteBatchIds(Arrays.*asList*(8, 9, 10));  
 System.***out***.println(result);  
 }

}

#### 1.3简单条件删除

@SpringBootTest  
**class** DemomptestApplicationTests {  
  
 @Autowired  
 **private** UserMapper **userMapper**;  
  
 *//简单条件删除* @Test  
 **public void** testDeleteByMap() {  
 HashMap<String, Object> map = **new** HashMap<>();  
 map.put(**"name"**, **"Helen"**);  
 map.put(**"age"**, 18);  
 **int** result = **userMapper**.deleteByMap(map);  
 System.***out***.println(result);  
 }

}

### 2、逻辑删除

#### 2.1物理删除和逻辑删除

物理删除：**真实删除**，将对应数据从数据库中删除，之后查询不到此条被删除数据

逻辑删除：**假删除**，将对应数据中代表是否被删除字段状态修改为“被删除状态”，之后在数据库中仍旧能看到此条数据记录

逻辑删除的使用场景：

可以进行数据恢复

有关联数据，不便删除

#### 2.2 逻辑删除实现流程

##### 2.2.1数据库修改

添加 deleted字段

ALTER TABLE `user` ADD COLUMN `deleted` boolean DEFAULT false;

##### 2.2.2实体类修改

添加deleted 字段，并加上 @TableLogic 注解

@Data  
**public class** User {  
**private** Long **id**;  
 **private** String **name**;  
 **private** Integer **age**;  
 **private** String **email**;  
  
 @TableField(fill = FieldFill.***INSERT***)   
 **private** Date **createTime**; *//create\_time* @TableField(fill = FieldFill.***INSERT\_UPDATE***)   
 **private** Date **updateTime**; *//update\_time* @Version  
 @TableField(fill = FieldFill.***INSERT***)  
 **private** Integer **version**;  
  
 @TableLogic  
 @TableField(fill = FieldFill.***INSERT***)  
 **private** Integer **deleted**;  
}

**package** com.atguigu.demomptest.handler;  
  
@Component //交给spring管理  
**public class** MyMetaObjectHandler **implements** MetaObjectHandler {  
  
 *//mp执行添加操作，这个方法执行* @Override  
 **public void** insertFill(MetaObject metaObject) {  
  
 **this**.setFieldValByName(**"deleted"**,0,metaObject);  
 }  
}

##### **2.2.3配置（可选）**

application.properties 加入以下配置，此为默认值，如果你的默认值和**mybatis-plus**默认的一样,该配置可无

**mybatis-plus.global-config.db-config.logic-delete-value=1**

**mybatis-plus.global-config.db-config.logic-not-delete-value=0**

##### **2.2.4 测试**

测试后发现，数据并没有被删除，deleted字段的值由0变成了1

**测试后分析打印的sql语句，是一条update**

注意：被删除前，数据的deleted 字段的值必须是 0，才能被选取出来执行逻辑删除的操作

**@Test**

**public void testLogicDelete() {**

**int result = userMapper.deleteById(1L);**

**System.out.println(result);**

**}**

##### **2.2.5测试逻辑删除后的查询**

MyBatis Plus中查询操作也会自动添加逻辑删除字段的判断，查询结果会过滤掉被逻辑删除的数据

**@Test**

**public void testLogicDeleteSelect() {**

**List<User> users = userMapper.selectList(null);**

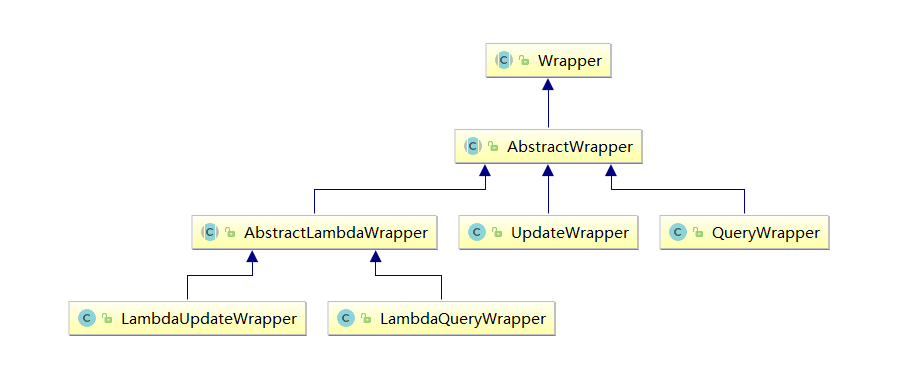
**users.forEach(System.out::println);**

**}**

## 条件构造器和常用接口

MyBatis Plus中查询操作也会自动添加逻辑删除字段的判断

### **1、wapper**介绍



Wrapper： 条件构造抽象类，最顶端父类

    AbstractWrapper ： 用于查询条件封装，生成 sql 的 where 条件

**QueryWrapper ： 查询条件封装（重点使用）**

        UpdateWrapper ： Update 条件封装

    AbstractLambdaWrapper ： 使用Lambda 语法

        LambdaQueryWrapper ：用于Lambda语法使用的查询Wrapper

        LambdaUpdateWrapper ： Lambda 更新封装Wrapper

### 2、测试用例

#### 2.1 ge、gt、le、lt、isNull、isNotNull

**@Test**

**public void testQuery() {**

**QueryWrapper<User> queryWrapper = new QueryWrapper<>();**

**queryWrapper**

**.isNull("name")**

**.ge("age", 12)**

**.isNotNull("email");**

**int result = userMapper.delete(queryWrapper);**

**System.out.println("delete return count = " + result);**

**}**

#### 2.2 eq、ne

注意：seletOne()返回的是一条实体记录，当出现多条时会报错

**@Test**

**public void testSelectOne() {**

**QueryWrapper<User> queryWrapper = new QueryWrapper<>();**

**queryWrapper.eq("name", "Tom");**

**Useruser = userMapper.selectOne(queryWrapper);//只能返回一条记录，多余一条则抛出异常**

**System.out.println(user);**

**}**

#### 2.3 between、notBetween

包含大小边界

**@Test**

**public void testSelectCount() {**

**QueryWrapper<User> queryWrapper = new QueryWrapper<>();**

**queryWrapper.between("age", 20, 30);**

**Integer count = userMapper.selectCount(queryWrapper); //返回数据数量**

**System.out.println(count);**

**}**

#### 2.4 like、notLike、likeLeft、likeRight

selectMaps()返回Map集合列表，通常配合select()使用

**@Test**

**public void testSelectMaps() {**

**QueryWrapper<User> queryWrapper = new QueryWrapper<>();**

**queryWrapper**

**.select("name", "age")**

**.like("name", "e")**

**.likeRight("email", "5");**

**List<Map<String, Object>>maps = userMapper.selectMaps(queryWrapper);//返回值是Map列表**

**maps.forEach(System.out::println);**

**}**

#### 2.5 orderBy、orderByDesc、orderByAsc

**@Test**

**public void testSelectListOrderBy() {**

**QueryWrapper<User> queryWrapper = new QueryWrapper<>();**

**queryWrapper.orderByDesc("age", "id");**

**List<User>users = userMapper.selectList(queryWrapper);**

**users.forEach(System.out::println);**

**}**

### 查询方式

|  |  |
| --- | --- |
| 查询方式 | 说明 |
| **setSqlSelect** | 设置 SELECT 查询字段 |
| **where** | WHERE 语句，拼接 + WHERE 条件 |
| **and** | AND 语句，拼接 + AND 字段=值 |
| **andNew** | AND 语句，拼接 + AND (字段=值) |
| **or** | OR 语句，拼接 + OR 字段=值 |
| **orNew** | OR 语句，拼接 + OR (字段=值) |
| **eq** | 等于= |
| **allEq** | 基于 map 内容等于= |
| **ne** | 不等于<> |
| **gt** | 大于> |
| **ge** | 大于等于>= |
| **lt** | 小于< |
| **le** | 小于等于<= |
| **like** | 模糊查询 LIKE |
| **notLike** | 模糊查询 NOT LIKE |
| **in** | IN 查询 |
| **notIn** | NOT IN 查询 |
| **isNull** | NULL 值查询 |
| **isNotNull** | IS NOT NULL |
| **groupBy** | 分组 GROUP BY |
| **having** | HAVING 关键词 |
| **orderBy** | 排序 ORDER BY |
| **orderAsc** | ASC 排序 ORDER BY |
| **orderDesc** | DESC 排序 ORDER BY |
| **exists** | EXISTS 条件语句 |
| **notExists** | NOT EXISTS 条件语句 |
| **between** | BETWEEN 条件语句 |
| **notBetween** | NOT BETWEEN 条件语句 |
| **addFilter** | 自由拼接 SQL |
| **last** | 拼接在最后，例如：last(“LIMIT 1”) |

# day02 搭建环境和医院设置

## 预约挂号微服务模块搭建

### 项目模块构建

hospital-manage：医院接口模拟端（已开发，直接使用）

yygh-parent：根目录，管理子模块：

common：公共模块父节点

common-util：工具类模块，所有模块都可以依赖于它

rabbit-util：rabbitmq业务封装

service-util：service服务的工具包，包含service服务的公共配置类，

所有service模块依赖于它

server-gateway：服务网关

model：实体类模块

service：api接口服务父节点

service-hosp：医院api接口服务

service-cmn：公共api接口服务

service-user：用户api接口服务

service-order：订单api接口服务

service-oss：文件api接口服务

service-sms：短信 api接口服务

service-task：定时任务服务

service-statistics：统计api接口服务

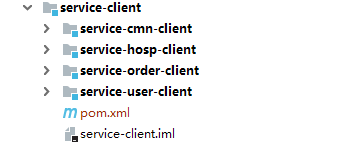
service-client：feign服务调用父节点

service-cmn-client：公共api接口

service-hosp-client：医院api接口

service-order-client：订单api接口

service-user：用户api接口

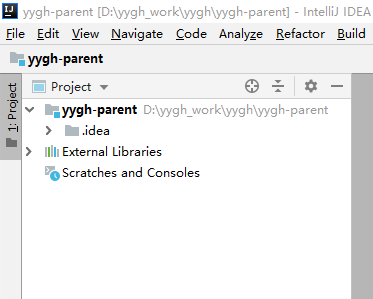


### 2、sql资源

导入sql：资料/sql/\*.sql

### 3、构建父工程（yygh-parent）

父工程目录我们已经通过git引入



#### 3.1 添加配置pom.xml

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
 <**modules**>  
 <**module**>common</**module**>  
 <**module**>model</**module**>  
 <**module**>service</**module**>  
 <**module**>service\_client</**module**>  
 <**module**>service\_gateway</**module**>  
 </**modules**>  
 <**parent**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-parent</**artifactId**>  
 <**version**>2.2.1.RELEASE</**version**>  
 <**relativePath**/> *<!-- lookup parent from repository -->* </**parent**>  
 <**groupId**>com.atguigu</**groupId**>  
 <**artifactId**>yygh\_parent</**artifactId**>  
 *<!-- 标注这是一个父工程 -->* <**packaging**>pom</**packaging**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 <**name**>yygh\_parent</**name**>  
 <**description**>Demo project for Spring Boot</**description**>  
  
 <**properties**>  
 <**java.version**>1.8</**java.version**>  
 <**cloud.version**>Hoxton.RELEASE</**cloud.version**>  
 <**alibaba.version**>2.2.0.RELEASE</**alibaba.version**>  
 <**mybatis-plus.version**>3.3.1</**mybatis-plus.version**>  
 <**mysql.version**>5.1.46</**mysql.version**>  
 <**swagger.version**>2.7.0</**swagger.version**>  
 <**jwt.version**>0.7.0</**jwt.version**>  
 <**fastjson.version**>1.2.29</**fastjson.version**>  
 <**httpclient.version**>4.5.1</**httpclient.version**>  
 <**easyexcel.version**>2.2.0-beta2</**easyexcel.version**>  
 <**aliyun.version**>4.1.1</**aliyun.version**>  
 <**oss.version**>3.9.1</**oss.version**>  
 <**jodatime.version**>2.10.1</**jodatime.version**>  
 </**properties**>  
  
 *<!--配置dependencyManagement锁定依赖的版本-->* <**dependencyManagement**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-dependencies</**artifactId**>  
 <**version**>${cloud.version}</**version**>  
 <**type**>pom</**type**>  
 <**scope**>import</**scope**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>com.alibaba.cloud</**groupId**>  
 <**artifactId**>spring-cloud-alibaba-dependencies</**artifactId**>  
 <**version**>${alibaba.version}</**version**>  
 <**type**>pom</**type**>  
 <**scope**>import</**scope**>  
 </**dependency**>  
  
 *<!--mybatis-plus 持久层-->* <**dependency**>  
 <**groupId**>com.baomidou</**groupId**>  
 <**artifactId**>mybatis-plus-boot-starter</**artifactId**>  
 <**version**>${mybatis-plus.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 <**version**>${mysql.version}</**version**>  
 </**dependency**>  
  
 *<!--swagger-->* <**dependency**>  
 <**groupId**>io.springfox</**groupId**>  
 <**artifactId**>springfox-swagger2</**artifactId**>  
 <**version**>${swagger.version}</**version**>  
 </**dependency**>  
 *<!--swagger ui-->* <**dependency**>  
 <**groupId**>io.springfox</**groupId**>  
 <**artifactId**>springfox-swagger-ui</**artifactId**>  
 <**version**>${swagger.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>io.jsonwebtoken</**groupId**>  
 <**artifactId**>jjwt</**artifactId**>  
 <**version**>${jwt.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.apache.httpcomponents</**groupId**>  
 <**artifactId**>httpclient</**artifactId**>  
 <**version**>${httpclient.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>fastjson</**artifactId**>  
 <**version**>${fastjson.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>easyexcel</**artifactId**>  
 <**version**>${easyexcel.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>com.aliyun</**groupId**>  
 <**artifactId**>aliyun-java-sdk-core</**artifactId**>  
 <**version**>${aliyun.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>com.aliyun.oss</**groupId**>  
 <**artifactId**>aliyun-sdk-oss</**artifactId**>  
 <**version**>${oss.version}</**version**>  
 </**dependency**>  
  
 *<!--日期时间工具-->* <**dependency**>  
 <**groupId**>joda-time</**groupId**>  
 <**artifactId**>joda-time</**artifactId**>  
 <**version**>${jodatime.version}</**version**>  
 </**dependency**>  
 </**dependencies**>  
 </**dependencyManagement**>  
  
 *<!-- 项目打包时会将java目录中的\*.xml文件也进行打包 -->* <**build**>  
 <**resources**>  
 <**resource**>  
 <**directory**>src/main/java</**directory**>  
 <**includes**>  
 <**include**>\*\*/\*.xml</**include**>  
 </**includes**>  
 <**filtering**>false</**filtering**>  
 </**resource**>  
 </**resources**>  
 </**build**>  
</**project**>

### 4、搭建common父模块

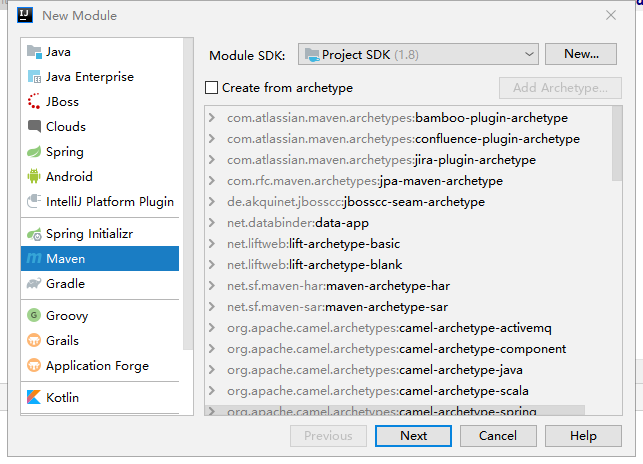
common：公共模块父节点

common-util：工具类模块，所有模块都可以依赖于它

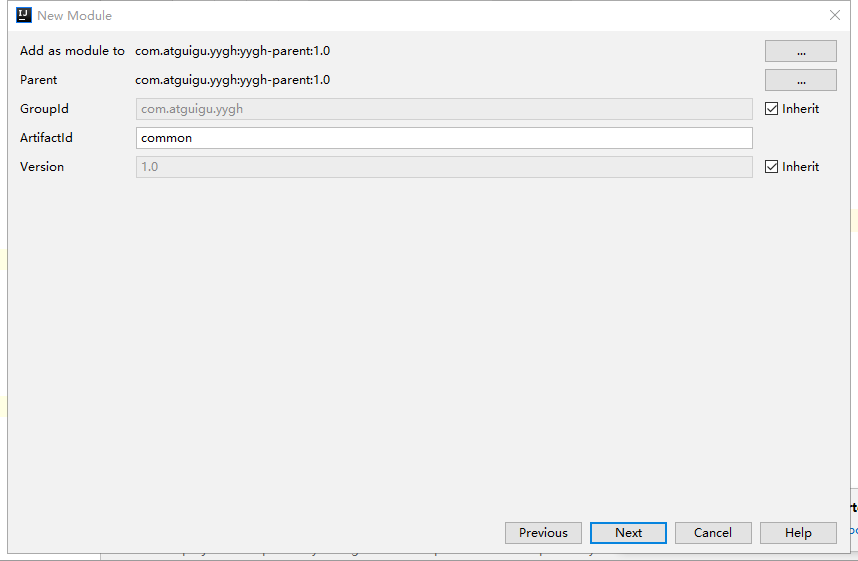
service-util：service服务的工具包，包含service服务的公共配置类，所有 service模块依赖于它

#### 4.1 搭建common

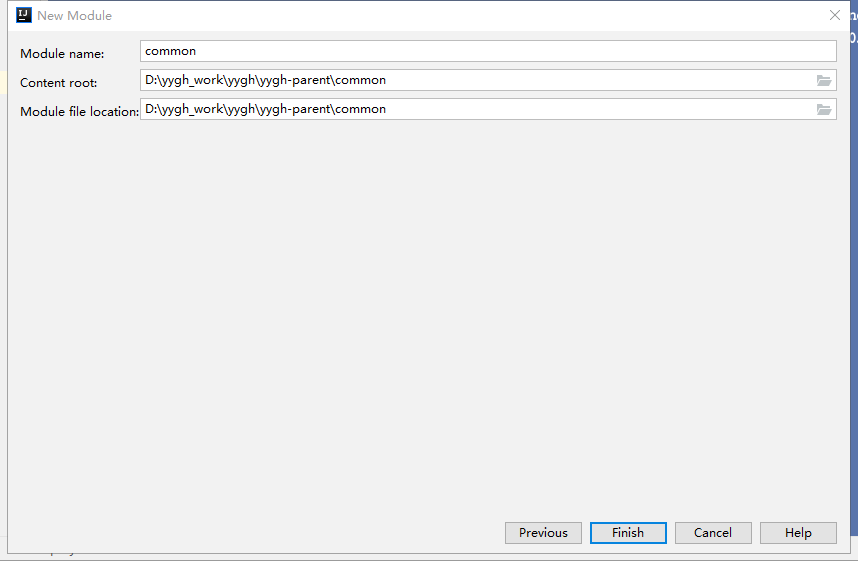
点击yygh-parent，选择New–>Module,操作如下



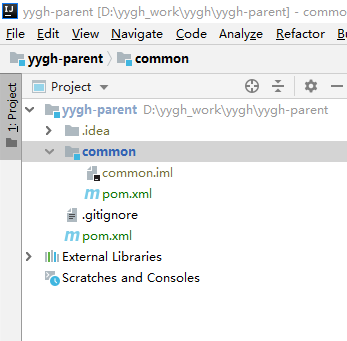
选择下一步



选择下一步



完成，删除src目录，结构如下



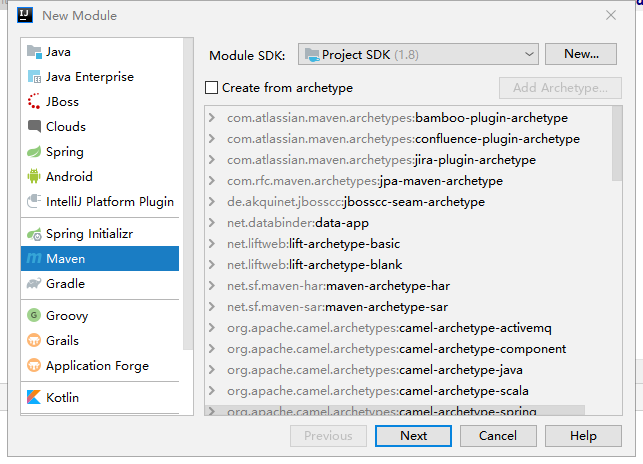
#### 4.2 修改配置pom.xml

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**parent**>  
 <**artifactId**>yygh\_parent</**artifactId**>  
 <**groupId**>com.atguigu</**groupId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 </**parent**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**artifactId**>common</**artifactId**>  
 <**packaging**>pom</**packaging**>  
 <**modules**>  
 <**module**>common\_util</**module**>  
 <**module**>service\_util</**module**>  
 <**module**>rabbit\_util</**module**>  
 </**modules**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 <**scope**>provided </**scope**>  
 </**dependency**>  
  
 *<!--mybatis-plus-->* <**dependency**>  
 <**groupId**>com.baomidou</**groupId**>  
 <**artifactId**>mybatis-plus-boot-starter</**artifactId**>  
 <**scope**>provided </**scope**>  
 </**dependency**>  
  
 *<!--lombok用来简化实体类：需要安装lombok插件-->* <**dependency**>  
 <**groupId**>org.projectlombok</**groupId**>  
 <**artifactId**>lombok</**artifactId**>  
 </**dependency**>  
  
 *<!--swagger-->* <**dependency**>  
 <**groupId**>io.springfox</**groupId**>  
 <**artifactId**>springfox-swagger2</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>io.springfox</**groupId**>  
 <**artifactId**>springfox-swagger-ui</**artifactId**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>fastjson</**artifactId**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

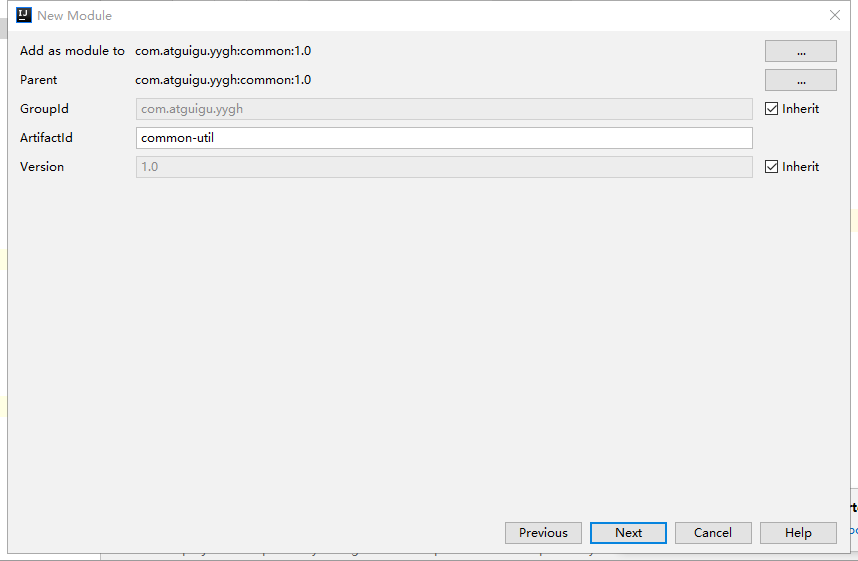
### 5、搭建common-util模块

#### 5.1 搭建common-util

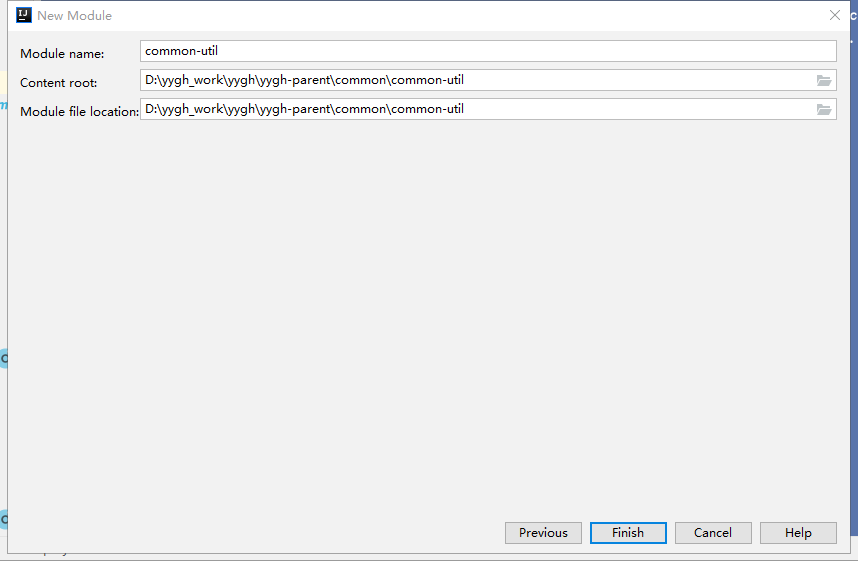
点击common，选择New–>Module,操作如下



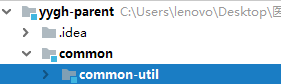
选择下一步



选择下一步



完成，结构如下



#### 5.2 修改配置pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>com.atguigu.yygh</groupId>

<artifactId>common</artifactId>

<version>1.0</version>

</parent>

<artifactId>common-util</artifactId>

<version>1.0</version>

<packaging>jar</packaging>

<name>common-util</name>

<description>common-util</description>

<dependencies>

<dependency>

<groupId>org.apache.httpcomponents</groupId>

<artifactId>httpclient</artifactId>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>easyexcel</artifactId>

</dependency>

<!-- 日期工具栏依赖 -->

<dependency>

<groupId>joda-time</groupId>

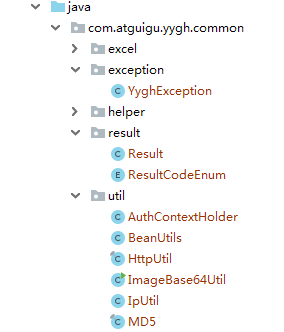
<artifactId>joda-time</artifactId>

</dependency>

</dependencies>

</project>

#### 5.3 添加公共工具类



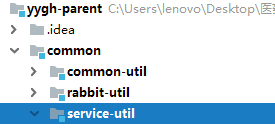
|  |  |
| --- | --- |
| YyghException | 自定义全局异常 |
| Result | API统一返回结果封装类 |
| ResultCodeEnum | API统一返回结果状态信息 |
| BeanUtils | 实体bean copy工具类 |
| MD5 | MD5加密工具类 |

### 6、搭建service-util模块

#### 6.1 搭建service-util

搭建过程同common-util

如图



#### 6.2 修改配置pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>com.atguigu.yygh</groupId>

<artifactId>common</artifactId>

<version>1.0</version>

</parent>

<artifactId>service-util</artifactId>

<version>1.0</version>

<packaging>jar</packaging>

<name>service-util</name>

<description>service-util</description>

<dependencies>

<dependency>

<groupId>com.atguigu.yygh</groupId>

<artifactId>common-util</artifactId>

<version>1.0</version>

</dependency>

<!-- redis -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-redis</artifactId>

</dependency>

<!-- spring2.X集成redis所需common-pool2-->

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-pool2</artifactId>

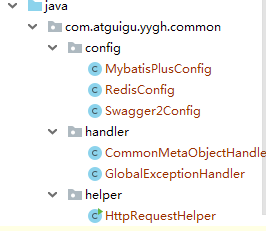
<version>2.6.0</version>

</dependency>

</dependencies>

</project>

#### 4.3 添加service-util公共类



|  |  |
| --- | --- |
| MybatisPlusConfig | MybatisPlus配置类 |
| CommonMetaObjectHandler | Mybatis plus Handler配置类 |

package com.atguigu.yygh.common.config;

/\*\*

\* MybatisPlus配置类

\*/

@EnableTransactionManagement //事务处理

@Configuration

@MapperScan("com.atguigu.yygh.\*.mapper")

public class MybatisPlusConfig {

/\*\*

\* 分页插件

\*/

@Bean

public PaginationInterceptor paginationInterceptor() {

PaginationInterceptor paginationInterceptor = new PaginationInterceptor();

// paginationInterceptor.setLimit(你的最大单页限制数量，默认 500 条，小于 0 如 -1 不受限制);

return paginationInterceptor;

}

/\*\*

\* 乐观锁配置

\*/

@Bean

public OptimisticLockerInterceptor optimisticLockerInterceptor() {

return new OptimisticLockerInterceptor();

}

}

package com.atguigu.yygh.common.handler;

/\*\*

\* Mybatis plus Handler配置类

\*/

@Component

public class CommonMetaObjectHandler implements MetaObjectHandler {

@Override

public void insertFill(MetaObject metaObject) {

this.setFieldValByName("createTime", new Date(), metaObject);

this.setFieldValByName("updateTime", new Date(), metaObject);

}

@Override

public void updateFill(MetaObject metaObject) {

this.setFieldValByName("updateTime", new Date(), metaObject);

}

}

### 7、搭建model模块

#### 7.1 搭建model

搭建过程同common-util父模块

#### 7.2 修改配置pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>com.atguigu.yygh</groupId>

<artifactId>yygh-parent</artifactId>

<version>1.0</version>

</parent>

<artifactId>model</artifactId>

<version>1.0</version>

<packaging>jar</packaging>

<name>model</name>

<description>model</description>

<dependencies>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</dependency>

<!--mybatis-plus-->

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-boot-starter</artifactId>

<scope>provided </scope>

</dependency>

<!--swagger-->

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

<scope>provided </scope>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>easyexcel</artifactId>

<scope>provided </scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-mongodb</artifactId>

<scope>provided </scope>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<scope>provided </scope>

</dependency>

</dependencies>

</project>

#### 7.3 引入实体类

将实体类都引入进来，后续不再单独引入，实体类没有业务逻辑，提前引入，方便后续开发。

### 8、搭建service父模块

service：service模块父节点

service-user：会员服务模块

...

#### 8.1 搭建service

搭建过程同common父模块

#### 8.2 修改配置pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<parent>

<artifactId>yygh-parent</artifactId>

<groupId>com.atguigu.yygh</groupId>

<version>1.0</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<artifactId>service</artifactId>

<packaging>pom</packaging>

<version>1.0</version>

<dependencies>

<dependency>

<groupId>com.atguigu.yygh</groupId>

<artifactId>service-util</artifactId>

<version>1.0</version>

</dependency>

<dependency>

<groupId>com.atguigu.yygh</groupId>

<artifactId>model</artifactId>

<version>1.0</version>

</dependency>

<!--web-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!--mybatis-plus-->

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-boot-starter</artifactId>

</dependency>

<!--mysql-->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<!--开发者工具-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

<!-- 服务调用feign -->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-openfeign</artifactId>

</dependency>

<!-- 服务注册 -->

<dependency>

<groupId>com.alibaba.cloud</groupId>

<artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>

</dependency>

<!-- 流量控制 -->

<dependency>

<groupId>com.alibaba.cloud</groupId>

<artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

<resources>

<resource>

<directory>src/main/java</directory>

<includes>

<include>\*\*/\*.yml</include>

<include>\*\*/\*.properties</include>

<include>\*\*/\*.xml</include>

</includes>

<filtering>false</filtering>

</resource>

<resource>

<directory>src/main/resources</directory>

<includes> <include>\*\*/\*.yml</include>

<include>\*\*/\*.properties</include>

<include>\*\*/\*.xml</include>

</includes>

<filtering>false</filtering>

</resource>

</resources>

</build>

</project>

## 二、gitee使用

### 1.创建仓库



### 2.点击创建

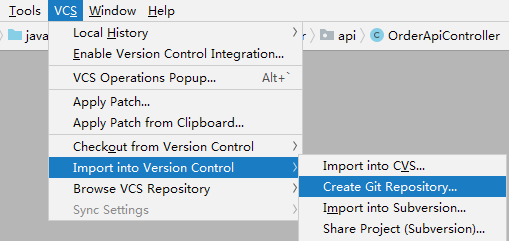


### 3复制gitee码云的HttpS连接

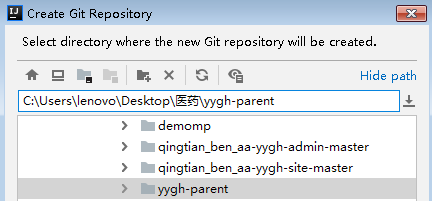


### 4 提交上传

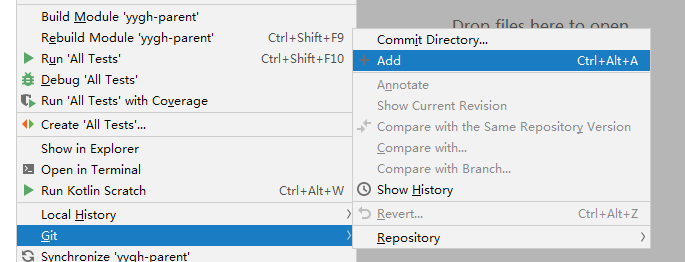
打开项目并点击菜单栏上的【CVS】--》【Import into version control】--》【Create Git Repository】**创建本地仓库**



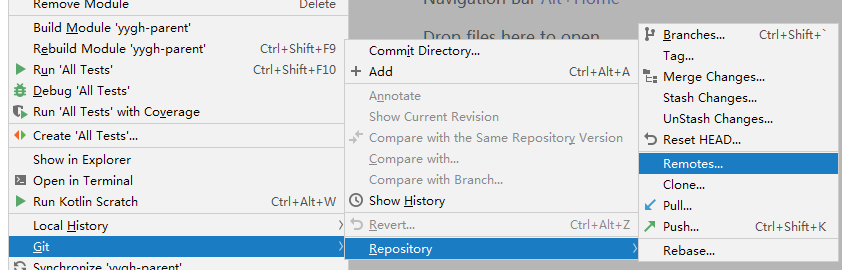
在打开的【Create Git Repository】对话框内选择本地仓库的位置，这里我选择项目的根目录。

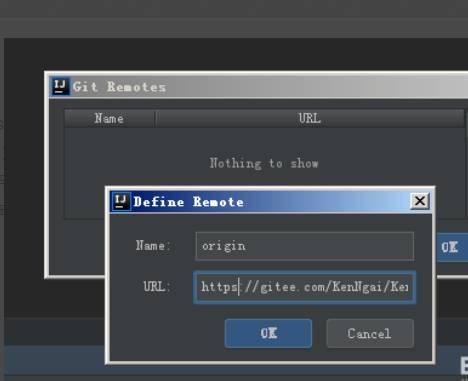


右击项目点击【Git】--》【Add】，将代码**提交到本地仓库内**。

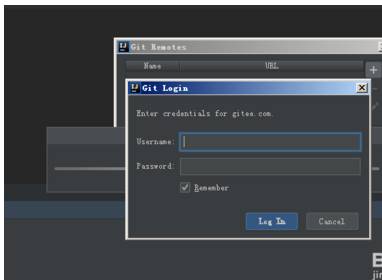


右击项目点击【Git】--》【Repository】--》【Remotes...】。在打开的【Git Remotes】窗口中**添加码云的远程仓库地址**。

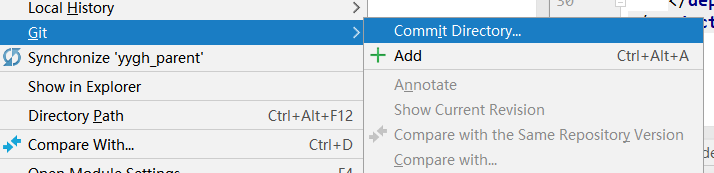


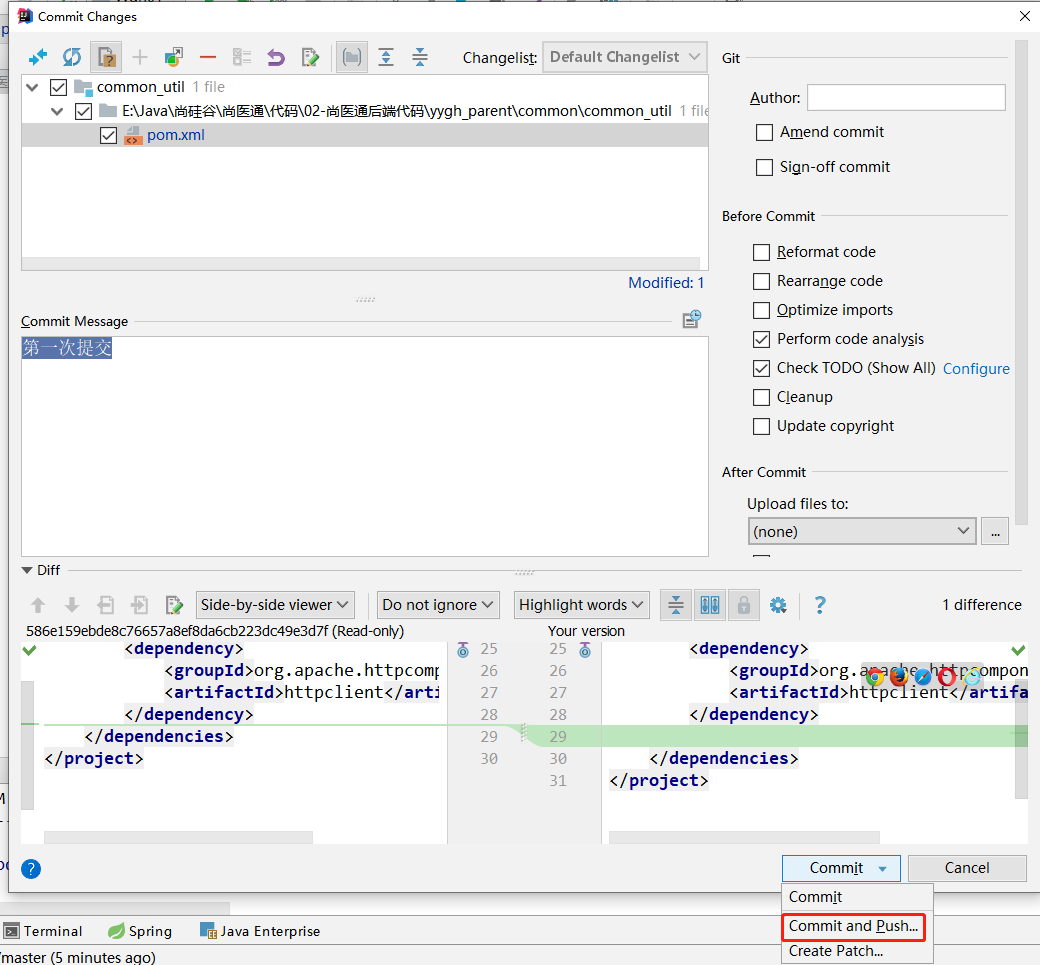


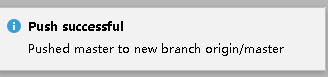
点击【OK】后接着输入码云的账号密码。



上传代码到码云：右击项目点击接着点击【Git】--》【Commit Directory】，在打开的【Push commits】内可以看到已提交到本地仓库的提交信息。点击【Commit and Push】按钮将本地仓库的代码上传到码云上，上传成功后就可以在码云上看到。







## 三、医院设置需求

### 1、需求

医院设置主要是用来保存开通医院的一些基本信息，每个医院一条信息，保存了医院编号（平台分配，全局唯一）和接口调用相关的签名key等信息，是整个流程的第一步，只有开通了医院设置信息，才可以上传医院相关信息。

我们所开发的功能就是基于单表的一个CRUD、锁定/解锁和发送签名信息这些基本功能。

### 2、表结构





hosname：医院名称

hoscode：医院编号（平台分配，全局唯一，api接口必填信息）

api\_url：医院回调的基础url（如：预约下单，我们要调用该地址去医院下单）

sign\_key：双方api接口调用的签名key，有平台生成

contacts\_name：医院联系人姓名

contacts\_phone：医院联系人手机

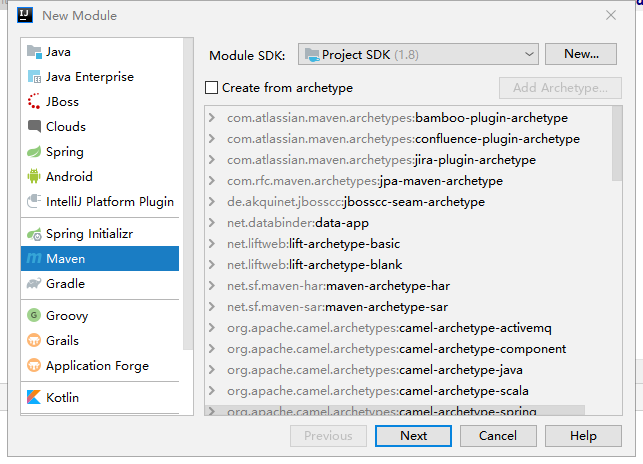
status：状态（锁定/解锁）

## 四、医院模块开发

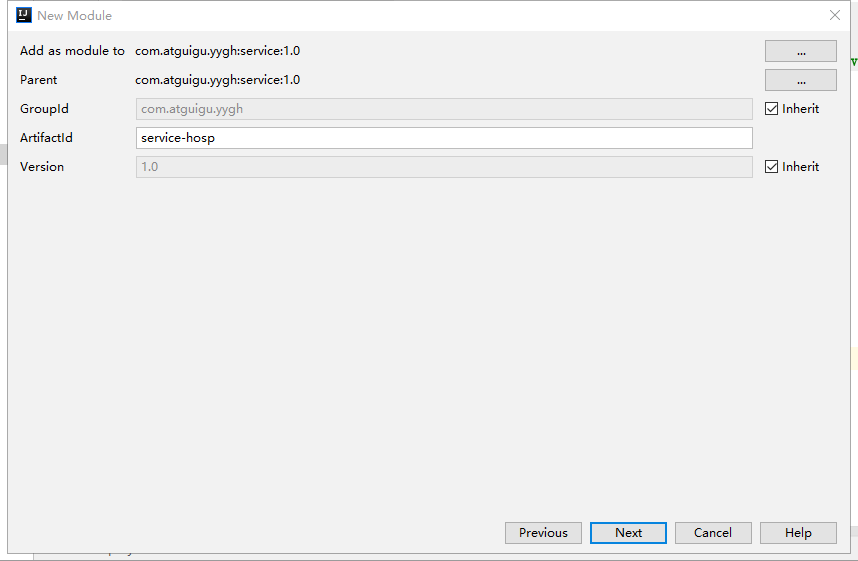
### 1、搭建医院模块service-hosp

#### 1.1搭建service-hosp

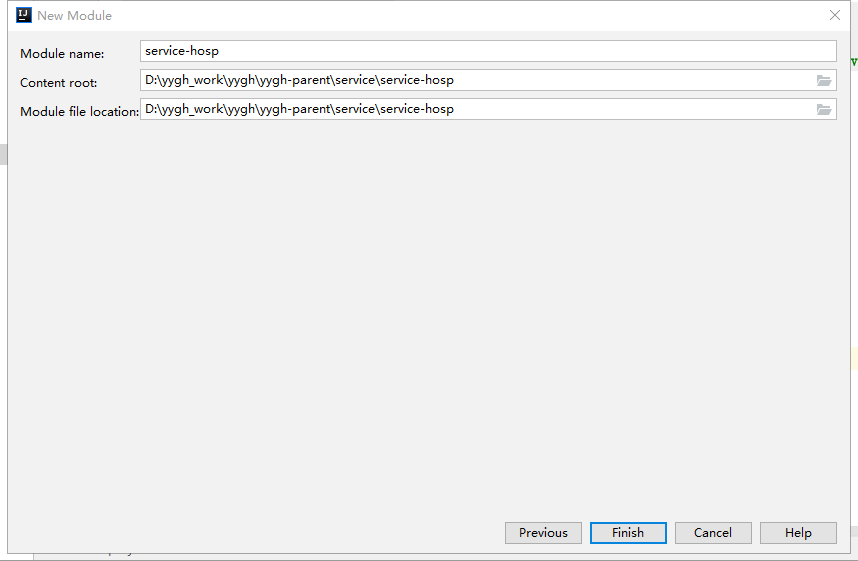
点击service，选择New–>Module,操作如下



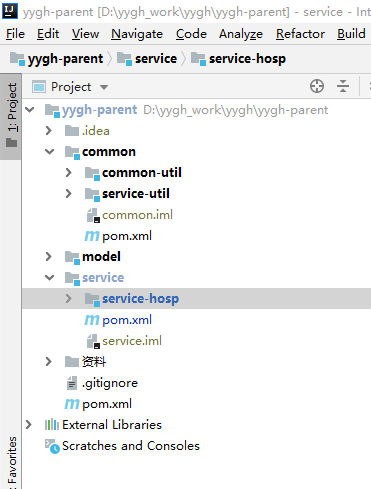
选择下一步



选择下一步



完成，结构如下：



#### 1.2修改配置

1）修改pom.xml

<**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-data-mongodb</**artifactId**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>com.atguigu</**groupId**>  
 <**artifactId**>service\_cmn\_client</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>com.atguigu</**groupId**>  
 <**artifactId**>rabbit\_util</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 </**dependency**>  
  
</**dependencies**>  
  
<**build**>  
 <**finalName**>service-hosp</**finalName**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-maven-plugin</**artifactId**>  
 </**plugin**>  
 </**plugins**>  
</**build**>

2）添加配置文件application.properties

*# 服务端口***server.port**=**8201***# 服务名***spring.application.name**=**service-hosp***# 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://192.168.44.163:3306/yygh\_hosp?characterEncoding=utf-8&useSSL=false  
spring.datasource.username**=**root  
spring.datasource.password**=**root123***#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***#配置mapper xml文件的路径  
#mybatis-plus.mapper-locations=classpath:com/atguigu/yygh/mapper/xml/\*.xml***mybatis-plus.mapper-locations**=**classpath:com/atguigu/yygh/mapper/xml/\*.xml***# nacos服务地址***spring.cloud.nacos.discovery.server-addr**=**127.0.0.1:8848***#开启sentinel***feign.sentinel.enabled**=**true***#设置sentinel地址***spring.cloud.sentinel.transport.dashboard**=**http://127.0.0.1:8858***#mongodb地址***spring.data.mongodb.host**=**192.168.44.163  
spring.data.mongodb.port**=**27017  
spring.data.mongodb.database**=**yygh\_hosp***#rabbitmq地址***spring.rabbitmq.host**=**127.0.0.1  
spring.rabbitmq.port**=**5672  
spring.rabbitmq.username**=**guest  
spring.rabbitmq.password**=**guest**

#### 1.3 添加启动类

添加com.atguigu.yygh.ServiceHospApplication类

package com.atguigu.yygh.hosp;

@SpringBootApplication

public class ServiceHospApplication {

public static void main(String[] args) {

SpringApplication.run(ServiceHospApplication.class, args);

}

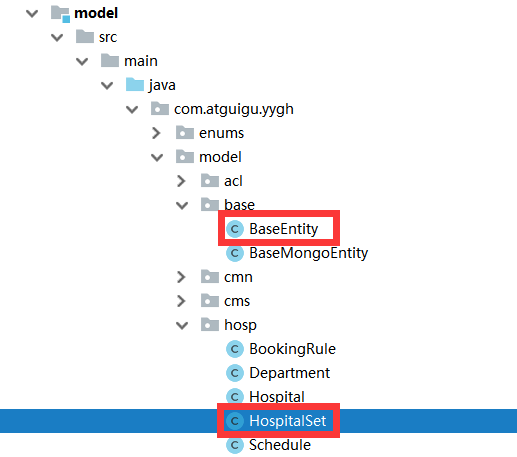
}

### 2、添加医院设置CURD

#### 2.1 添加model

说明：由于实体对象没有逻辑，我们已经统一导入

com.atguigu.yygh.model.hosp.HospitalSet



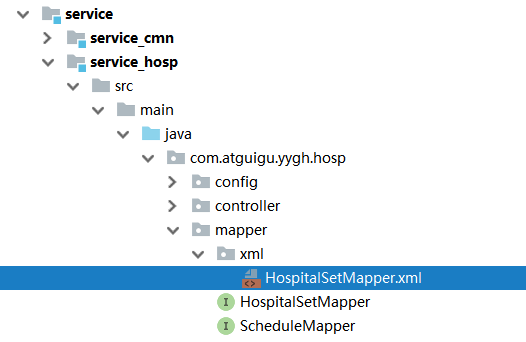
#### 2.2 添加Mapper

添加com.atguigu.yygh.hosp.mapper.HospitalSetMapper

**package** com.atguigu.yygh.hosp.mapper;  
  
**public interface** HospitalSetMapper **extends** BaseMapper<HospitalSet> {  
  
}

在mapper/xml下添加HospitalSetMapper.xml

*<?***xml version="1.0" encoding="UTF-8"** *?>***<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  
 "http://mybatis.org/dtd/mybatis-3-mapper.dtd"*>***<**mapper namespace="com.atguigu.yygh.hosp.mapper.HospitalSetMapper"**>  
  
</**mapper**>



#### 2.3 添加service接口及实现类

1、添加com.atguigu.yygh.hosp.service.HospitalSetService接口

package com.atguigu.yygh.hosp.service;

public interface HospitalSetService extends IService<HospitalSetService> {

}

2、添加com.atguigu.yygh.hosp.service.impl.HospitalSetServiceImpl接口实现

package com.atguigu.yygh.hosp.service.impl;

@Service

Public class HospitalSetServiceImpl extends ServiceImpl<

HospitalSetMapper, HospitalSet> implements HospitalSetService {

@Autowired

private HospitalSetMapper hospitalSetMapper;

}

#### 2.4 添加controller

添加com.atguigu.yygh.hosp.controller.HospitalSetController类

package com.atguigu.yygh.hosp.controller;

@RestController

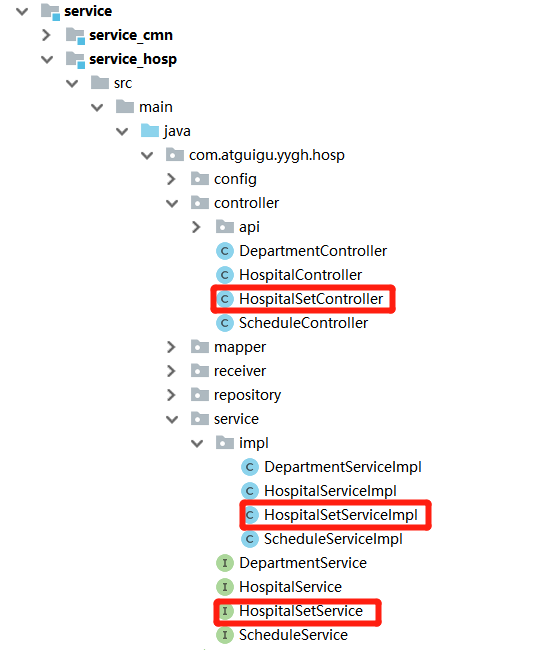
@RequestMapping("/admin/hosp/hospitalSet")

public class HospitalSetController {

@Autowired

private HospitalSetService hospitalSetService;

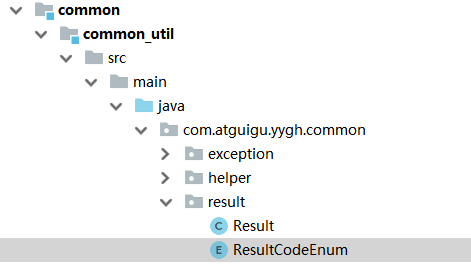
}



#### 2.5 医院设置CRUD

由于com.baomidou.mybatisplus.extension.service.impl.ServiceImpl类已经默认实现了单表的CRUD，分页查询也有默认实现，能够更加灵活和代码简洁把分页查询功能实现。

#### 2.6 全局统一返回结果定义



##### 1）统一返回结果状态信息类

**package** com.atguigu.yygh.common.result;  
  
**import** lombok.Getter;  
  
*/\*\*  
 \* 统一返回结果状态信息类  
 \*/*@Getter  
**public enum** ResultCodeEnum {  
  
 ***SUCCESS***(200,**"成功"**),  
 ***FAIL***(201, **"失败"**),  
 ***PARAM\_ERROR***( 202, **"参数不正确"**),  
 ***SERVICE\_ERROR***(203, **"服务异常"**),  
 ***DATA\_ERROR***(204, **"数据异常"**),  
 ***DATA\_UPDATE\_ERROR***(205, **"数据版本异常"**),  
  
 ***LOGIN\_AUTH***(208, **"未登陆"**),  
 ***PERMISSION***(209, **"没有权限"**),  
  
 ***CODE\_ERROR***(210, **"验证码错误"**),  
*// LOGIN\_MOBLE\_ERROR(211, "账号不正确"),* ***LOGIN\_DISABLED\_ERROR***(212, **"改用户已被禁用"**),  
 ***REGISTER\_MOBLE\_ERROR***(213, **"手机号已被使用"**),  
 ***LOGIN\_AURH***(214, **"需要登录"**),  
 ***LOGIN\_ACL***(215, **"没有权限"**),  
  
 ***URL\_ENCODE\_ERROR***( 216, **"URL编码失败"**),  
 ***ILLEGAL\_CALLBACK\_REQUEST\_ERROR***( 217, **"非法回调请求"**),  
 ***FETCH\_ACCESSTOKEN\_FAILD***( 218, **"获取accessToken失败"**),  
 ***FETCH\_USERINFO\_ERROR***( 219, **"获取用户信息失败"**),  
 *//LOGIN\_ERROR( 23005, "登录失败"),* ***PAY\_RUN***(220, **"支付中"**),  
 ***CANCEL\_ORDER\_FAIL***(225, **"取消订单失败"**),  
 ***CANCEL\_ORDER\_NO***(225, **"不能取消预约"**),  
  
 ***HOSCODE\_EXIST***(230, **"医院编号已经存在"**),  
 ***NUMBER\_NO***(240, **"可预约号不足"**),  
 ***TIME\_NO***(250, **"当前时间不可以预约"**),  
  
 ***SIGN\_ERROR***(300, **"签名错误"**),  
 ***HOSPITAL\_OPEN***(310, **"医院未开通，暂时不能访问"**),  
 ***HOSPITAL\_LOCK***(320, **"医院被锁定，暂时不能访问"**),  
 ;  
  
 **private** Integer **code**;  
 **private** String **message**;  
  
 **private** ResultCodeEnum(Integer code, String message) {  
 **this**.**code** = code;  
 **this**.**message** = message;  
 }  
}

##### 全局统一返回结果类

**package** com.atguigu.yygh.common.result;  
  
*/\*\*  
 \* 全局统一返回结果类  
 \*/*@Data  
@ApiModel(value = **"全局统一返回结果"**)  
**public class** Result<T> {  
  
 @ApiModelProperty(value = **"返回码"**)  
 **private** Integer **code**;  
  
 @ApiModelProperty(value = **"返回消息"**)  
 **private** String **message**;  
  
 @ApiModelProperty(value = **"返回数据"**)  
 **private** T **data**;  
  
 **public** Result(){}  
  
 **protected static** <T> Result<T> build(T data) {  
 Result<T> result = **new** Result<T>();  
 **if** (data != **null**)  
 result.setData(data);  
 **return** result;  
 }  
  
 **public static** <T> Result<T> build(T body, ResultCodeEnum resultCodeEnum) {  
 Result<T> result = *build*(body);  
 result.setCode(resultCodeEnum.getCode());  
 result.setMessage(resultCodeEnum.getMessage());  
 **return** result;  
 }  
  
 **public static** <T> Result<T> build(Integer code, String message) {  
 Result<T> result = *build*(**null**);  
 result.setCode(code);  
 result.setMessage(message);  
 **return** result;  
 }  
  
 **public static**<T> Result<T> ok(){  
 **return** Result.*ok*(**null**);  
 }  
 */\*\*  
 \* 操作成功  
 \** ***@param data*** *\** ***@param <T>*** *\** ***@return*** *\*/* **public static**<T> Result<T> ok(T data){  
 Result<T> result = *build*(data);  
 **return** *build*(data, ResultCodeEnum.***SUCCESS***);  
 }  
  
 **public static**<T> Result<T> fail(){  
 **return** Result.*fail*(**null**);  
 }

*/\*\*  
 \* 操作失败  
 \** ***@param data*** *\** ***@param <T>*** *\** ***@return*** *\*/* **public static**<T> Result<T> fail(T data){  
 Result<T> result = *build*(data);  
 **return** *build*(data, ResultCodeEnum.***FAIL***);  
 }  
  
 **public** Result<T> message(String msg){  
 **this**.setMessage(msg);  
 **return this**;  
 }  
  
 **public** Result<T> code(Integer code){  
 **this**.setCode(code);  
 **return this**;  
 }  
  
 **public boolean** isOk() {  
 **if**(**this**.getCode().intValue() == ResultCodeEnum.***SUCCESS***.getCode().intValue()) {  
 **return true**;  
 }  
 **return false**;  
 }  
}

#### 2.7 添加controller方法

**package** com.atguigu.yygh.hosp.controller;  
  
@Api(tags = **"医院设置管理"**)  
@RestController  
@RequestMapping(**"/admin/hosp/hospitalSet"**)**public class** HospitalSetController {  
  
 *//注入service* @Autowired  
 **private** HospitalSetService **hospitalSetService**;  
  
 *// http://localhost:8201/admin/hosp/hospitalSet/findAll  
  
 //1 查询医院设置表所有信息* @ApiOperation(value = **"获取所有医院设置"**)  
 @GetMapping(**"findAll"**)  
 **public** Result findAllHospitalSet() {  
 *//调用service的方法* List<HospitalSet> list = **hospitalSetService**.list();  
 **return** Result.*ok*(list);  
 }  
  
 *//2 逻辑删除医院设置* @ApiOperation(value = **"逻辑删除医院设置"**)  
 @DeleteMapping(**"{id}"**)  
 **public** Result removeHospSet(@PathVariable Long id) {  
 **boolean** flag = **hospitalSetService**.removeById(id);  
 **if**(flag) {  
 **return** Result.*ok*();  
 } **else** {  
 **return** Result.*fail*();  
 }  
 }  
  
 *//3 条件查询带分页*

*先创建HospitalSetQueryVo，属性为：hosname、hoscode* @PostMapping(**"findPageHospSet/{current}/{limit}"**)  
 **public** Result findPageHospSet(@PathVariable **long** current,  
 @PathVariable **long** limit,  
 @RequestBody(required = **false**) HospitalSetQueryVo hospitalSetQueryVo) {

*//创建page对象，传递当前页，每页记录数* Page<HospitalSet> page = **new** Page<>(current,limit);  
 *//构建条件* QueryWrapper<HospitalSet> wrapper = **new** QueryWrapper<>();  
 String hosname = hospitalSetQueryVo.getHosname(); *//医院名称* String hoscode = hospitalSetQueryVo.getHoscode(); *//医院编号*

**if**(!StringUtils.*isEmpty*(hosname)) {

*//模糊查询*  
 wrapper.like(**"hosname"**,hospitalSetQueryVo.getHosname());  
 }  
 **if**(!StringUtils.*isEmpty*(hoscode)) {

*//精确查询*  
 wrapper.eq(**"hoscode"**,hospitalSetQueryVo.getHoscode());  
 }  
  
 *//调用方法实现分页查询* IPage<HospitalSet> pageHospitalSet = **hospitalSetService**.page(page, wrapper);  
  
 *//返回结果* **return** Result.*ok*(pageHospitalSet);  
 }  
  
 *//4 添加医院设置* @PostMapping(**"saveHospitalSet"**)  
 **public** Result saveHospitalSet(@RequestBody HospitalSet hospitalSet) {

*//设置状态：1 使用，0 不能使用* hospitalSet.setStatus(1);

*//签名秘钥* Random random = **new** Random();  
 hospitalSet.setSignKey(MD5.*encrypt*(System.*currentTimeMillis*()+**""**+random.nextInt(1000)));  
 *//调用service* **boolean** save = **hospitalSetService**.save(hospitalSet);  
 **if**(save) {  
 **return** Result.*ok*();  
 } **else** {  
 **return** Result.*fail*();  
 }  
 }  
  
 *//5 根据id获取医院设置* @GetMapping(**"getHospSet/{id}"**)  
 **public** Result getHospSet(@PathVariable Long id) {  
*// try {  
// //模拟异常  
// int a = 1/0;  
// }catch (Exception e) {  
// throw new YyghException("失败",201);  
// }* HospitalSet hospitalSet = **hospitalSetService**.getById(id);  
 **return** Result.*ok*(hospitalSet);  
 }  
  
 *//6 修改医院设置* @PostMapping(**"updateHospitalSet"**)  
 **public** Result updateHospitalSet(@RequestBody HospitalSet hospitalSet) {  
 **boolean** flag = **hospitalSetService**.updateById(hospitalSet);  
 **if**(flag) {  
 **return** Result.*ok*();  
 } **else** {  
 **return** Result.*fail*();  
 }  
 }  
  
 *//7 批量删除医院设置* @DeleteMapping(**"batchRemove"**)  
 **public** Result batchRemoveHospitalSet(@RequestBody List<Long> idList) {  
 **hospitalSetService**.removeByIds(idList);  
 **return** Result.*ok*();  
 }  
  
 *//8 医院设置锁定和解锁* @PutMapping(**"lockHospitalSet/{id}/{status}"**)  
 **public** Result lockHospitalSet(@PathVariable Long id,  
 @PathVariable Integer status) {  
 *//根据id查询医院设置信息* HospitalSet hospitalSet = **hospitalSetService**.getById(id);  
 *//设置状态* hospitalSet.setStatus(status);  
 *//调用方法* **hospitalSetService**.updateById(hospitalSet);  
 **return** Result.*ok*();  
 }  
  
 *//9 发送签名秘钥* @PutMapping(**"sendKey/{id}"**)  
 **public** Result lockHospitalSet(@PathVariable Long id) {  
 HospitalSet hospitalSet = **hospitalSetService**.getById(id);  
 String signKey = hospitalSet.getSignKey();  
 String hoscode = hospitalSet.getHoscode();  
 *//****TODO 发送短信* return** Result.*ok*();  
 }  
}

**package** com.atguigu.yygh.hosp.config;  
  
@Configuration //自定义配置类  
@MapperScan(**"com.atguigu.yygh.hosp.mapper"**) **//开启扫描，可以加到启动类、配置类上**  
**public class** HospConfig {  
  
}

说明：一个完整的CRUD就实现了

### 3、Swagger2介绍与集成

#### 3.1 swagger2介绍

编写和维护接口文档是每个程序员的职责，根据Swagger2可以快速帮助我们编写最新的API接口文档，再也不用担心开会前仍忙于整理各种资料了，间接提升了团队开发的沟通效率。

常用注解：

swagger通过注解表明该接口会生成文档，包括接口名、请求方法、参数、返回信息的等等。

@Api：修饰整个类，描述Controller的作用

@ApiOperation：描述一个类的一个方法，或者说一个接口

@ApiParam：单个参数描述

@ApiModel：用对象来接收参数

@ApiModelProperty：用对象接收参数时，描述对象的一个字段

@ApiImplicitParam：一个请求参数

@ApiImplicitParams：多个请求参数

#### 3.2 swagger2集成

##### 1）项目整合swagger2

在common模块下的servic-util下的pom.xml引入依赖

<!--swagger-->

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

</dependency>

<dependency>

<groupId>io.springfox</groupId>

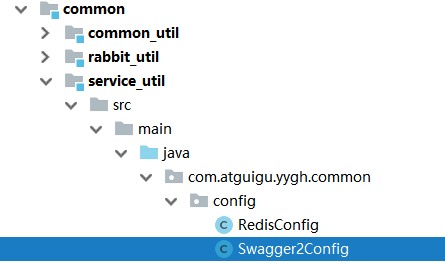
<artifactId>springfox-swagger-ui</artifactId>

</dependency>

说明：我们在yygh-parent中的pom.xml中添加了版本控制，这里不需要添加版本，已引入就忽略

##### 2）添加swagger2配置类

在service-util模块添加配置类：



**package** com.atguigu.yygh.common.config;  
  
*/\*\*  
 \* Swagger2配置信息  
 \*/*@Configuration  
@EnableSwagger2  
**public class** Swagger2Config {  
  
 @Bean  
 **public** Docket webApiConfig(){  
  
 **return new** Docket(DocumentationType.***SWAGGER\_2***)  
 .groupName(**"webApi"**)  
 .apiInfo(webApiInfo())  
 .select()  
 *//只显示api路径下的页面* .paths(Predicates.*and*(PathSelectors.*regex*(**"/api/.\*"**)))  
 .build();  
 }  
 @Bean  
 **public** Docket adminApiConfig(){  
  
 **return new** Docket(DocumentationType.***SWAGGER\_2***)  
 .groupName(**"adminApi"**)  
 .apiInfo(adminApiInfo())  
 .select()  
 *//只显示admin路径下的页面* .paths(Predicates.*and*(PathSelectors.*regex*(**"/admin/.\*"**)))  
 .build();  
 }  
 **private** ApiInfo webApiInfo(){  
  
 **return new** ApiInfoBuilder()  
 .title(**"网站-API文档"**)  
 .description(**"本文档描述了网站微服务接口定义"**)  
 .version(**"1.0"**)  
 .contact(**new** Contact(**"atguigu"**, **"http://atguigu.com"**, **"493211102@qq.com"**))  
 .build();  
 }  
 **private** ApiInfo adminApiInfo(){  
  
 **return new** ApiInfoBuilder()  
 .title(**"后台管理系统-API文档"**)  
 .description(**"本文档描述了后台管理系统微服务接口定义"**)  
 .version(**"1.0"**)  
 .contact(**new** Contact(**"atguigu"**, **"http://atguigu.com"**, **"49321112@qq.com"**))  
 .build();  
 }  
}

#### 3.3 使用swagger2测试

**package** com.atguigu.yygh.hosp;  
  
@SpringBootApplication  
@ComponentScan(basePackages = **"com.atguigu"**)

**//扫描com.atguigu下所有的文件，包括Swagger2Config**  
**public class** ServiceHospApplication {  
  
 **public static void** main(String[] args) {  
  
 SpringApplication.*run*(ServiceHospApplication.**class**, args);  
 }  
}



### 4、医院锁定与解锁

医院锁定后不能再上传数据

#### 4.1 添加controller方法

在HospitalSetController类添加方法

**package** com.atguigu.yygh.hosp.controller;  
  
@Api(tags = **"医院设置管理"**)  
@RestController  
@RequestMapping(**"/admin/hosp/hospitalSet"**)**public class** HospitalSetController {  
  
 *//注入service* @Autowired  
 **private** HospitalSetService **hospitalSetService**;  
  
 *//8 医院设置锁定和解锁* @PutMapping(**"lockHospitalSet/{id}/{status}"**)  
 **public** Result lockHospitalSet(@PathVariable Long id,  
 @PathVariable Integer status) {  
 *//根据id查询医院设置信息* HospitalSet hospitalSet = **hospitalSetService**.getById(id);  
 *//设置状态* hospitalSet.setStatus(status);  
 *//调用方法* **hospitalSetService**.updateById(hospitalSet);  
 **return** Result.*ok*();  
 }  
}

### 5、发送签名key

医院信息配置后，可以通过短信的形式发送医院编号与签名key给联系人，联系人拿到该信息就可以参考《尚医通API接口文档.docx》对接接口了。

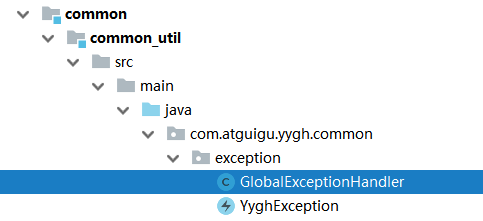
#### 5.1 添加controller方法

在HospitalSetController类添加方法

**package** com.atguigu.yygh.hosp.controller;  
  
@Api(tags = **"医院设置管理"**)  
@RestController  
@RequestMapping(**"/admin/hosp/hospitalSet"**)**public class** HospitalSetController {  
  
 *//注入service* @Autowired  
 **private** HospitalSetService **hospitalSetService**;  
  
 *//9 发送签名秘钥* @PutMapping(**"sendKey/{id}"**)  
 **public** Result lockHospitalSet(@PathVariable Long id) {  
 HospitalSet hospitalSet = **hospitalSetService**.getById(id);  
 String signKey = hospitalSet.getSignKey();  
 String hoscode = hospitalSet.getHoscode();  
 *//****TODO 发送短信* return** Result.*ok*();  
 }  
}

### 6、全局异常处理

spring boot 默认情况下会映射到 /error 进行异常处理，但是提示并不十分友好，下面自定义异常处理，提供友好展示。



#### 6.1 自定义异常类

我们在搭建模块时在common-util模块已经添加了YyghException类，这里不做解释

**package** com.atguigu.yygh.common.exception;  
  
*/\*\*  
 \* 自定义全局异常类  
 \*/*@Data  
@ApiModel(value = **"自定义全局异常类"**)  
**public class** YyghException **extends** RuntimeException {  
  
 @ApiModelProperty(value = **"异常状态码"**)  
 **private** Integer **code**;  
  
 */\*\*  
 \* 通过状态码和错误消息创建异常对象  
 \** ***@param message*** *\** ***@param code*** *\*/* **public** YyghException(String message, Integer code) {  
 **super**(message);  
 **this**.**code** = code;  
 }  
  
 */\*\*  
 \* 接收枚举类型对象  
 \** ***@param resultCodeEnum*** *\*/* **public** YyghException(ResultCodeEnum resultCodeEnum) {  
 **super**(resultCodeEnum.getMessage());  
 **this**.**code** = resultCodeEnum.getCode();  
 }  
  
 @Override  
 **public** String toString() {  
 **return "YyghException{"** +  
 **"code="** + **code** +  
 **", message="** + **this**.getMessage() +  
 **'}'**;  
 }  
}

#### 6.2 添加全局异常处理类

在service-util模块添加全局异常处理类

**package** com.atguigu.yygh.common.exception;

*/\*\**

*\* 全局异常处理类*

*\**

*\*/*  
@ControllerAdvice  
**public class** GlobalExceptionHandler {  
  
 @ExceptionHandler(**Exception.class**)  
 @ResponseBody  
 **public** Result error(Exception e) {  
 e.printStackTrace();  
 **return** Result.*fail*();  
 }

*/\*\**

*\* 自定义异常处理方法*

*\* @param e*

*\*/*  
 @ExceptionHandler(**YyghException.class**)  
 @ResponseBody  
 **public** Result error(YyghException e) {  
 e.printStackTrace();  
 **return** Result.*fail*();  
 }  
}

#### 6.3 集成测试

手动在controller任意方法制造异常（int i = 1/0），添加GlobalExceptionHandler类与不添加这个类区别

*//5 根据id获取医院设置*@GetMapping(**"getHospSet/{id}"**)  
**public** Result getHospSet(@PathVariable Long id) {  
 try {  
 //模拟异常  
 int a = 1/0;  
 }catch (Exception e) {  
 **throw new YyghException("失败",201); //自定义异常类，需要手动抛出异常**  
 }HospitalSet hospitalSet = **hospitalSetService**.getById(id);  
 **return** Result.*ok*(hospitalSet);  
}

### 7、日志

#### 7.1配置日志级别

日志记录器（Logger）的行为是分等级的。如下表所示：

分为：OFF、FATAL、ERROR、WARN、INFO、DEBUG、ALL

默认情况下，spring boot从控制台打印出来的日志级别只有INFO及以上级别，可以配置日志级别

# 设置日志级别

logging.level.root=WARN

这种方式只能将日志打印在控制台上

#### 7.2 Logback日志

spring boot内部使用Logback作为日志实现的框架。

Logback和log4j非常相似，如果你对log4j很熟悉，那对logback很快就会得心应手。

#### 7.3 配置日志

resources/logback-spring.xml

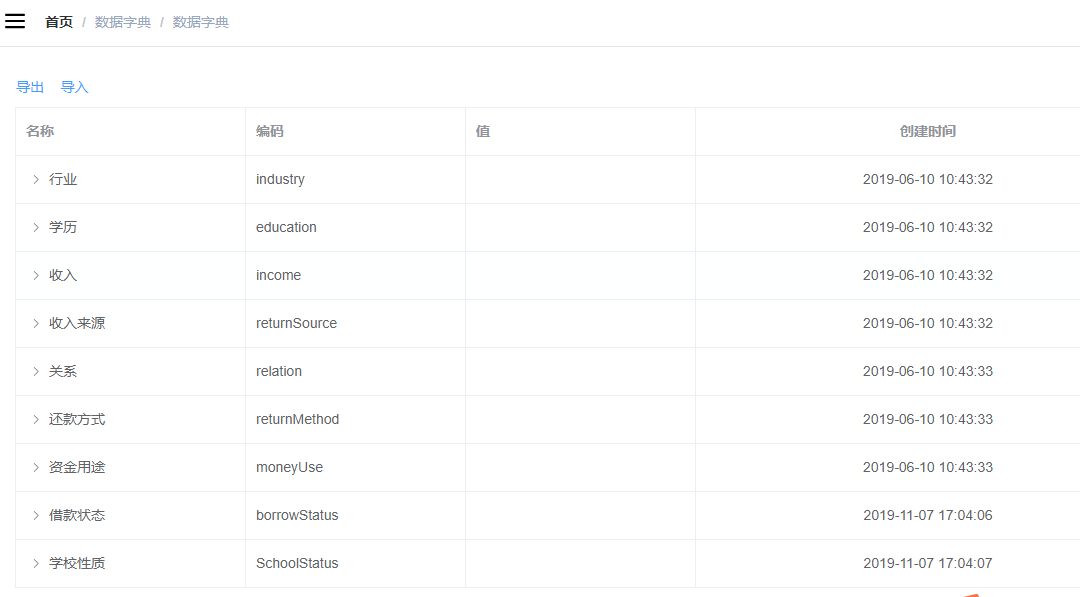


# day06 数据字典

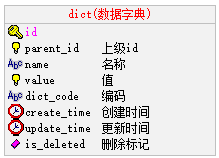
## 一、数据字典介绍

何为数据字典？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。

### 1、页面效果



### 2、表设计



### 3、数据分析



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

### 4、根据页面效果分析数据接口

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

接下来我们封装服务器端数据接口，接口测试通过后再做页面渲染

## 二、数据字典开发

### 1、搭建service-cmn模块

#### 1.1 搭建service-cmn模块

搭建过程参考service-hosp模块

#### 1.2 修改配置

**修改pom.xml**

<?xml version="1.0"encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>com.atguigu.yygh</groupId>

<artifactId>service</artifactId>

<version>1.0</version>

</parent>

<version>1.0</version>

<artifactId>service-cmn</artifactId>

<packaging>jar</packaging>

<name>service-cmn</name>

<description>service-cmn</description>

<dependencies>

</dependencies>

<build>

<finalName>service-cmn</finalName>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**添加配置文件application.properties**

# 服务端口

server.port=8202

# 服务名

spring.application.name=service-cmn

# 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://192.168.44.165:3306/yygh\_cmn?characterEncoding=utf-8&useSSL=false

spring.datasource.username=root

spring.datasource.password=root123

#返回json的全局时间格式

spring.jackson.date-format=yyyy-MM-dd HH:mm:ss

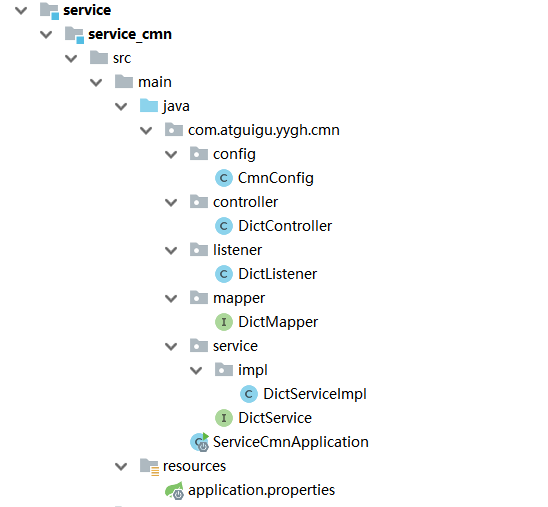
spring.jackson.time-zone=GMT+8

#### 1.3 启动类

**package** com.atguigu.yygh.cmn;  
  
@SpringBootApplication  
@ComponentScan(basePackages = {**"com.atguigu"**}) **//扫描所有com.atguigu下的文件**  
**public class** ServiceCmnApplication {

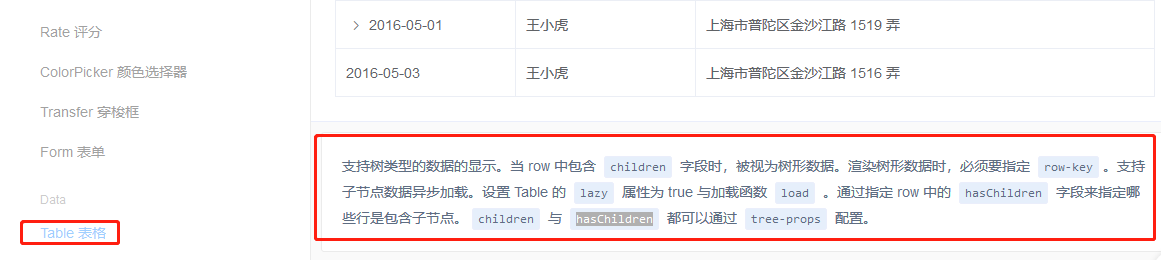
**public static void** main(String[] args) {  
 SpringApplication.*run*(ServiceCmnApplication.**class**, args);  
 }  
}

### 数据字典列表



根据element组件要求，返回列表数据必须包含hasChildren字典，如图：

https://element.eleme.cn/#/zh-CN/component/table



#### 2.1 数据字典列表接口

##### 2.1.1 model模块添加数据字典实体

在model模块查看实体：com.atguigu.yygh.model.cmn.Dict

@Data

@ApiModel(description = "数据字典")

@TableName("dict")

public class Dict extends BaseEntity {

private static final long serialVersionUID = 1L;

@ApiModelProperty(value = "上级id")

@TableField("parent\_id")

private Long parentId;

@ApiModelProperty(value = "名称")

@TableField("name")

private String name;

@ApiModelProperty(value = "值")

@TableField("value")

private String value;

@ApiModelProperty(value = "编码")

@TableField("dict\_code")

private String dictCode;

@ApiModelProperty(value = "是否包含子节点")

@TableField(exist = false) **//标识为数据库表不存在该字段**

private boolean hasChildren;

}

**说明：hasChildren为树形组件所需字典，标识为数据库表不存在该字段**

##### 2.1.2 添加数据字典mapper

添加com.atguigu.yygh.cmn.mapper.DictMapper

public interface DictMapper extends BaseMapper<Dict> {

}

##### 2.1.3 添加数据字典service

1、添加com.atguigu.yygh.cmn.service.DictService

public interface DictService extends IService<Dict> {

//根据数据id查询子数据列表

List<Dict> findChlidData(Long id);

}

2、添加com.atguigu.yygh.cmn.service.impl.DictServiceImpl接口实现

**package** com.atguigu.yygh.cmn.service.impl;  
  
@Service  
**public class** DictServiceImpl **extends** ServiceImpl<DictMapper, Dict> **implements** DictService {  
  
 *//根据数据id查询子数据列表* @Override  
 *//@Cacheable(value = "dict",keyGenerator = "keyGenerator")* **public** List<Dict> findChlidData(Long id) {  
 QueryWrapper<Dict> wrapper = **new** QueryWrapper<>();  
 wrapper.eq(**"parent\_id"**,id);  
 List<Dict> dictList = **baseMapper**.selectList(wrapper);  
 *//向list集合每个dict对象中设置hasChildren* **for** (Dict dict:dictList) {  
 Long dictId = dict.getId();  
 **boolean** isChild = **this**.isChildren(dictId);  
 dict.setHasChildren(isChild);  
 }  
 **return** dictList;  
 }  
  
 *//导出数据字典接口* @Override  
 **public void** exportDictData(HttpServletResponse response) {  
 *//设置下载信息* response.setContentType(**"application/vnd.ms-excel"**);  
 response.setCharacterEncoding(**"utf-8"**);  
 String fileName = **"dict"**;  
 response.setHeader(**"Content-disposition"**, **"attachment;filename="**+ fileName + **".xlsx"**);  
 *//查询数据库* List<Dict> dictList = **baseMapper**.selectList(**null**);  
 *//Dict -- DictEeVo* List<DictEeVo> dictVoList = **new** ArrayList<>();  
 **for**(Dict dict:dictList) {  
 DictEeVo dictEeVo = **new** DictEeVo();  
 *// dictEeVo.setId(dict.getId());* BeanUtils.*copyProperties*(dict,dictEeVo);  
 dictVoList.add(dictEeVo);  
 }  
 *//调用方法进行写操作* **try** {  
 EasyExcel.*write*(response.getOutputStream(), DictEeVo.**class**).sheet(**"dict"**)  
 .doWrite(dictVoList);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 *//导入数据字典* @Override  
 @CacheEvict(value = **"dict"**, allEntries=**true**)  
 **public void** importDictData(MultipartFile file) {  
 **try** {  
 EasyExcel.*read*(file.getInputStream(),DictEeVo.**class**,**new** DictListener(**baseMapper**)).sheet().doRead();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 *//根据dictcode和value查询* @Override  
 **public** String getDictName(String dictCode, String value) {  
 *//如果dictCode为空，直接根据value查询* **if**(StringUtils.*isEmpty*(dictCode)) {  
 *//直接根据value查询* QueryWrapper<Dict> wrapper = **new** QueryWrapper<>();  
 wrapper.eq(**"value"**,value);  
 Dict dict = **baseMapper**.selectOne(wrapper);  
 **return** dict.getName();  
 } **else** {*//如果dictCode不为空，根据dictCode和value查询  
 //根据dictcode查询dict对象，得到dict的id值* Dict codeDict = **this**.getDictByDictCode(dictCode);  
 Long parent\_id = codeDict.getId();  
 *//根据parent\_id和value进行查询* Dict finalDict = **baseMapper**.selectOne(**new** QueryWrapper<Dict>()  
 .eq(**"parent\_id"**, parent\_id)  
 .eq(**"value"**, value));  
 **return** finalDict.getName();  
 }  
 }  
  
 *//根据dictCode获取下级节点* @Override  
 **public** List<Dict> findByDictCode(String dictCode) {  
 *//根据dictcode获取对应id* Dict dict = **this**.getDictByDictCode(dictCode);  
 *//根据id获取子节点* List<Dict> chlidData = **this**.findChlidData(dict.getId());  
 **return** chlidData;  
 }  
  
 **private** Dict getDictByDictCode(String dictCode) {  
 QueryWrapper<Dict> wrapper = **new** QueryWrapper<>();  
 wrapper.eq(**"dict\_code"**,dictCode);  
 Dict codeDict = **baseMapper**.selectOne(wrapper);  
 **return** codeDict;  
 }  
  
 *//判断id下面是否有子节点* **private boolean** isChildren(Long id) {  
 QueryWrapper<Dict> wrapper = **new** QueryWrapper<>();  
 wrapper.eq(**"parent\_id"**,id);  
 Integer count = **baseMapper**.selectCount(wrapper);  
 *// 0>0 1>0* **return** count>0;  
 }  
}

##### 2.1.4 添加数据字典controller

添加com.atguigu.yygh.cmn.controller.DictController

**package** com.atguigu.yygh.cmn.controller;  
  
@Api(description = **"数据字典接口"**)  
@RestController  
@RequestMapping(**"/admin/cmn/dict"**)**public class** DictController {  
  
 @Autowired  
 **private** DictService **dictService**;  
  
 *//导入数据字典* @PostMapping(**"importData"**)  
 **public** Result importDict(MultipartFile file) {  
 **dictService**.importDictData(file);  
 **return** Result.*ok*();  
 }  
  
 *//导出数据字典接口* @GetMapping(**"exportData"**)  
 **public void** exportDict(HttpServletResponse response) {  
 **dictService**.exportDictData(response);  
 }  
  
 *//根据dictCode获取下级节点* @ApiOperation(value = **"根据dictCode获取下级节点"**)  
 @GetMapping(**"findByDictCode/{dictCode}"**)  
 **public** Result findByDictCode(@PathVariable String dictCode) {  
 List<Dict> list = **dictService**.findByDictCode(dictCode);  
 **return** Result.*ok*(list);  
 }  
  
 *//根据数据id查询子数据列表* @ApiOperation(value = **"根据数据id查询子数据列表"**)  
 @GetMapping(**"findChildData/{id}"**)  
 **public** Result findChildData(@PathVariable Long id) {  
 List<Dict> list = **dictService**.findChlidData(id);  
 **return** Result.*ok*(list);  
 }  
  
 *//根据dictcode和value查询* @GetMapping(**"getName/{dictCode}/{value}"**)  
 **public** String getName(@PathVariable String dictCode,  
 @PathVariable String value) {  
 String dictName = **dictService**.getDictName(dictCode,value);  
 **return** dictName;  
 }  
  
 *//根据value查询* @GetMapping(**"getName/{value}"**)  
 **public** String getName(@PathVariable String value) {  
 String dictName = **dictService**.getDictName(**""**,value);  
 **return** dictName;  
 }  
}

#### 2.2 数据字典列表前端

##### 2.2.1 添加路由

在 src/router/index.js 文件添加路由

  {

    path: '/cmn',

    component: Layout,

    redirect: '/cmn/list',

    name: '数据管理',

    alwaysShow: true,

    meta: { title: '数据管理', icon: 'example' },

    children: [

      {

        path: 'list',

        name: '数据字典',

        component: () => import('@/views/dict/list'),

        meta: { title: '数据字典', icon: 'table' }

      }

    ]

  },

说明：列表与查看都添加了

##### 2.2.2 定义api

创建文件 src/api/cmn/dict.js

export default {

  dictList(id) {//数据字典列表

    return request ({

      url: `/admin/cmn/dict/findChildData/${id}`,

      method: 'get'

    })

  }

}

##### 2.2.2 方法调用

<script>

import dict from '@/api/dict'

export default {

    data() {

        return {

            list:[] //数据字典列表数组

        }

    },

    created() {

        this.getDictList(1)

    },

    methods: {

        //数据字典列表

        getDictList(id) {

            dict.dictList(id)

                .then(response => {

                    this.list = response.data

                })

        },

        getChildrens(tree, treeNode, resolve) {

            dict.dictList(tree.id).then(response => {

                resolve(response.data)

            })

        }

    }

}

</script>

##### 2.2.3 表格渲染

<template>

    <div class="app-container">

        <el-table

        :data="list"

        style="width: 100%"

        row-key="id"

        border

        lazy

        :load="getChildrens"

        :tree-props="{children: 'children', hasChildren: 'hasChildren'}">

            <el-table-column label="名称" width="230" align="left">

            <template slot-scope="scope">

            <span>{{ scope.row.name }}</span>

            </template>

            </el-table-column>

            <el-table-column label="编码" width="220">

            <template slot-scope="{row}">

                    {{ row.dictCode }}

            </template>

            </el-table-column>

            <el-table-column label="值" width="230" align="left">

            <template slot-scope="scope">

            <span>{{ scope.row.value }}</span>

            </template>

            </el-table-column>

            <el-table-column label="创建时间" align="center">

            <template slot-scope="scope">

            <span>{{ scope.row.createTime }}</span>

            </template>

            </el-table-column>

        </el-table>

    </div>

</template>

### 3、EasyExcel介绍

Java解析、生成Excel比较有名的框架有Apache poi、jxl。但他们都存在一个严重的问题就是非常的耗内存，poi有一套SAX模式的API可以一定程度的解决一些内存溢出的问题，但POI还是有一些缺陷，比如07版Excel解压缩以及解压后存储都是在内存中完成的，内存消耗依然很大。easyexcel重写了poi对07版Excel的解析，能够原本一个3M的excel用POI sax依然需要100M左右内存降低到几M，并且再大的excel不会出现内存溢出，03版依赖POI的sax模式。在上层做了模型转换的封装，让使用者更加简单方便。

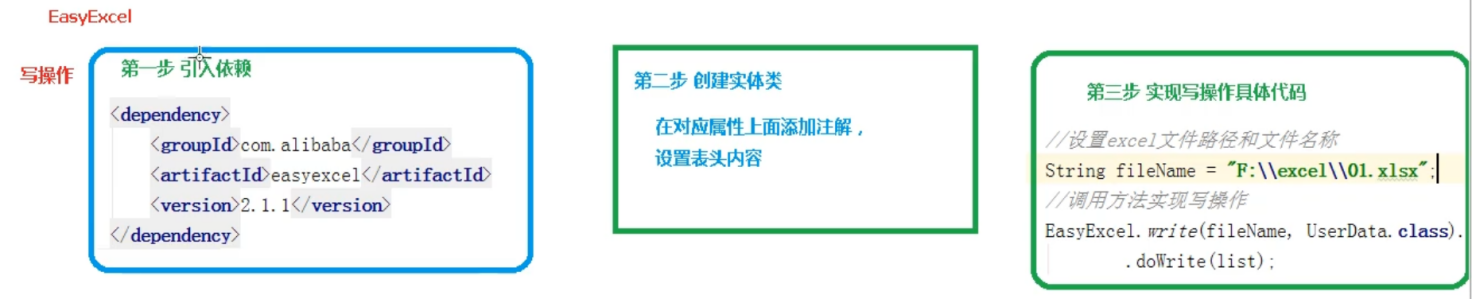
EasyExcel是一个基于Java的简单、省内存的读写Excel的开源项目。在尽可能节约内存的情况下支持读写百M的Excel。

文档地址：https://alibaba-easyexcel.github.io/index.html

github地址：https://github.com/alibaba/easyexcel

#### 3.1 导出示例

示例链接：<https://alibaba-easyexcel.github.io/quickstart/write.html>



**package** com.atguigu.easyexcel;  
  
@Data  
**public class** UserData {  
  
 @ExcelProperty(value = **"用户编号"**)  
 **private int uid**;  
  
 @ExcelProperty(value = **"用户名称"**)  
 **private** String **username**;  
  
}  
  
**public class** TestWrite {  
  
 **public static void** main(String[] args) {  
  
 *//构建数据list集合* List<UserData> list = **new** ArrayList();  
 **for** (**int** i=0;i<10;i++) {  
 UserData data = **new** UserData();  
 data.setUid(i);  
 data.setUsername(**"lucy"**+i);  
 list.add(data);  
 }  
  
 *//设置excel文件路径和文件名称* String fileName = **"F:\\excel\\01.xlsx"**;  
 *//调用方法实现写操作* EasyExcel.*write*(fileName, UserData.**class**).sheet(**"用户信息"**)  
 .doWrite(list);  
 }  
}

#### 3.2 导入示例

示例链接：<https://alibaba-easyexcel.github.io/quickstart/read.html>

**package** com.atguigu.easyexcel;  
  
@Data  
**public class** UserData {  
  
 @ExcelProperty(value = **"用户编号"**,index = 0)  
 **private int uid**;  
  
 @ExcelProperty(value = **"用户名称"**,index = 1)  
 **private** String **username**;  
  
}

**public class** ExcelListener **extends** AnalysisEventListener<UserData> {  
  
 *//一行一行读取excel内容，从第二行读取* @Override  
 **public void** invoke(UserData userData, AnalysisContext analysisContext) {  
 System.***out***.println(userData);  
 }  
  
 @Override  
 **public void** invokeHeadMap(Map<Integer, String> headMap, AnalysisContext context) {  
 System.***out***.println(**"表头信息："**+headMap);  
 }  
  
 *//读取之后执行* @Override  
 **public void** doAfterAllAnalysed(AnalysisContext analysisContext) {  
  
 }  
}  
  
**public class** TestRead {  
  
 **public static void** main(String[] args) {  
  
 *// 读取文件路径* String fileName = **"F:\\excel\\01.xlsx"**;  
  
 *//调用方法实现读取操作* EasyExcel.*read*(fileName, UserData.**class**,**new** ExcelListener()).sheet().doRead();  
 }  
}

#### 3.3 EasyExcel集成

##### 3.3.1 添加依赖

1、添加依赖

<dependencies>

<!-- https://mvnrepository.com/artifact/com.alibaba/easyexcel -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>easyexcel</artifactId>

<version>2.1.1</version>

</dependency>

</dependencies>

说明：我们已经在yygh-parent中的pom.xml中添加了所有依赖管理

2、导入导出需要定义对象，对象上需要引用easyexcel标签，所以model模块需要引入，scope：provided

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>easyexcel</artifactId>

**<scope>provided </scope>**

</dependency>

3、导入导出我们会把它封装成工具类，放在common-util中，所有模块公用，所以该模块也得引入

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>easyexcel</artifactId>

</dependency>

### 4、数据字典导出

#### 4.1 导出接口封装

##### 4.1.1 在model模块添加导出实体

在model模块查看实体：com.atguigu.yygh.vo.cmn.DictEeVo

**package** com.atguigu.yygh.vo.cmn;  
@Data  
**public class** DictEeVo {  
  
 @ExcelProperty(value = **"id"** ,index = 0)  
 **private** Long **id**;  
  
 @ExcelProperty(value = **"上级id"** ,index = 1)  
 **private** Long **parentId**;  
  
 @ExcelProperty(value = **"名称"** ,index = 2)  
 **private** String **name**;  
  
 @ExcelProperty(value = **"值"** ,index = 3)  
 **private** String **value**;  
  
 @ExcelProperty(value = **"编码"** ,index = 4)  
 **private** String **dictCode**;  
  
}

##### 4.1.2 在service-cmn模块添加service方法

1、在DictService类添加接口

/\*\*

\* 导出

\* @param response

\*/

void exportData(HttpServletResponse response);

2、在DictServiceImpl类添加接口实现类

**package** com.atguigu.yygh.cmn.service.impl;  
  
@Service  
**public class** DictServiceImpl **extends** ServiceImpl<DictMapper, Dict> **implements** DictService {  
  
 *//导出数据字典接口* @Override  
 **public void** exportDictData(HttpServletResponse response) {  
 *//设置下载信息* response.setContentType(**"application/vnd.ms-excel"**);

*//这里URLEncoder.encode可以防止中文乱码 当然和easyexcel没有关系*  
 response.setCharacterEncoding(**"utf-8"**);  
 String fileName = **"dict"**;  
 response.setHeader(**"Content-disposition"**, **"attachment;filename="**+ fileName + **".xlsx"**);  
 *//查询数据库* List<Dict> dictList = **baseMapper**.selectList(**null**);  
 *//Dict -- DictEeVo* List<DictEeVo> dictVoList = **new** ArrayList<>();

**for**(Dict dict:dictList) {  
 DictEeVo dictEeVo = **new** DictEeVo();  
 *// dictEeVo.setId(dict.getId());* BeanUtils.*copyProperties*(dict,dictEeVo);  
 dictVoList.add(dictEeVo);  
 }  
 *//调用方法进行写操作* **try** {  
 EasyExcel.*write*(response.getOutputStream(), DictEeVo.**class**).sheet(**"dict"**)  
 .doWrite(dictVoList);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

说明：直接复制示例代码中的“web中的写”，改造即可

##### 4.1.3 在service-cmn模块添加controller方法

在DictController类添加方法

**package** com.atguigu.yygh.cmn.controller;  
  
@Api(description = **"数据字典接口"**)  
@RestController  
@RequestMapping(**"/admin/cmn/dict"**)**public class** DictController {  
  
 @Autowired  
 **private** DictService **dictService**;  
  
 *//导出数据字典接口*

@ApiOperation(value = "**导出**")@GetMapping(**"exportData"**)  
 **public void** exportDict(HttpServletResponse response) {

**dictService**.exportDictData(response);  
 }  
}

##### 4.1.4 测试

直接通过浏览器导出数据：http://localhost:8202/admin/cmn/dict/exportData

#### 4.2 导出前端实现

##### 4.2.1 列表页面添加导出按钮

src/views/cmn/dict/list.vue

<div class="el-toolbar">

<div class="el-toolbar-body"style="justify-content: flex-start;">

<el-button type="text"@click="exportData"><i class="fa fa-plus"/> 导出</el-button>

</div>

</div>

##### 4.2.2 添加导出方法

exportData() {

window.location.href = 'http://localhost:8202/admin/cmn/dict/exportData'

}

##### 4.2.3 测试

### 5、数据字典导入

#### 5.1 导入接口封装

##### 5.1.1 创建回调监听器

**package** com.atguigu.yygh.cmn.listener;  
  
**public class** DictListener **extends** AnalysisEventListener<DictEeVo> {

*//注入DictMapper 与@Autowired作用一样*  
 **private** DictMapper **dictMapper**;  
 **public** DictListener(DictMapper dictMapper) {  
 **this**.**dictMapper** = dictMapper;  
 }  
  
 *//一行一行读取，第二行开始* @Override  
 **public void** invoke(DictEeVo dictEeVo, AnalysisContext analysisContext) {  
 *//调用方法添加数据库* Dict dict = **new** Dict();  
 BeanUtils.*copyProperties*(dictEeVo,dict);  
 **dictMapper**.insert(dict);  
 }  
  
 @Override  
 **public void** doAfterAllAnalysed(AnalysisContext analysisContext) {  
  
 }  
}

##### 5.1.2 在service-cmn模块添加service方法

**package** com.atguigu.yygh.cmn.service.impl;  
  
@Service  
**public class** DictServiceImpl **extends** ServiceImpl<DictMapper, Dict> **implements** DictService {  
  
 *//导入数据字典* @Override  
 **public void** importDictData(MultipartFile file) {  
 **try** {  
 EasyExcel.*read*(file.getInputStream(),DictEeVo.**class**,**new** DictListener(**baseMapper**)).sheet().doRead();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

##### 5.1.3 在service-cmn模块添加controller方法

在DictController类添加方法

**package** com.atguigu.yygh.cmn.controller;  
  
@Api(description = **"数据字典接口"**)  
@RestController  
@RequestMapping(**"/admin/cmn/dict"**)**public class** DictController {  
  
 @Autowired  
 **private** DictService **dictService**;  
  
 *//导入数据字典*

@ApiOperation(value = "**导入**")@PostMapping(**"importData"**)  
 **public** Result importDict(MultipartFile file) { *//MultipartFile得到上传的文件*

**dictService**.importDictData(file);  
 **return** Result.*ok*();  
 }  
}

#### 5.2 导入前端实现

##### 5.2.1 列表页面添加导入按钮

src/views/cmn/dict/list.vue

<el-button type="text"@click="importData"><i class="fa fa-plus"/> 导入</el-button>

说明：按钮位置与导出并列

##### 5.2.2 添加导入弹出层

<el-dialog title="导入":visible.sync="dialogImportVisible"width="480px">

<el-form label-position="right"label-width="170px">

<el-form-item label="文件">

<el-upload

:multiple="false"

:on-success="onUploadSuccess"

:action="'http://localhost:8202/admin/cmn/dict/importData'"

class="upload-demo">

<el-button size="small"type="primary">点击上传</el-button>

<div slot="tip"class="el-upload\_\_tip">只能上传xls文件，且不超过500kb</div>

</el-upload>

</el-form-item>

</el-form>

<div slot="footer"class="dialog-footer">

<el-button @click="dialogImportVisible = false">

取消

</el-button>

</div>

</el-dialog>

##### 5.2.3 添加弹出可见模型

// 定义数据

data() {

return {

list: [],

listLoading: true,

dialogImportVisible: false

}

}

##### 5.2.4 添加方法

importData() {

this.dialogImportVisible = true

},

onUploadSuccess(response, file) {

this.$message.info('上传成功')

this.dialogImportVisible = false

this.fetchData()

}

## 三、Spring Cache + Redis 缓存数据

Spring Cache 是一个非常优秀的缓存组件。自Spring 3.1起，提供了类似于@Transactional注解事务的注解Cache支持，且提供了Cache抽象，方便切换各种底层Cache（如：redis）

使用Spring Cache的好处：

* 提供基本的Cache抽象，方便切换各种底层Cache；
* 通过注解Cache可以实现类似于事务一样，缓存逻辑透明的应用到我们的业务代码上，且只需要更少的代码就可以完成；
* 提供事务回滚时也自动回滚缓存；
* 支持比较复杂的缓存逻辑；

### 1、项目集成Spring Cache + Redis

因为缓存也是公共使用，所有的service模块都有可能使用缓存，所以我们把依赖与部分配置加在service-util模块，这样其他service模块都可以使用了

#### 1.1 service-util添加依赖

在service-util模块的pom.xml添加依赖

*<!-- redis -->*<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-data-redis</**artifactId**>  
</**dependency**>  
  
*<!-- spring2.X集成redis所需common-pool2-->*<**dependency**>  
 <**groupId**>org.apache.commons</**groupId**>  
 <**artifactId**>commons-pool2</**artifactId**>  
 <**version**>2.6.0</**version**>  
</**dependency**>

#### 1.2 service-util添加配置类

创建com.atguigu.yygh.common.config.RedisConfig

**package** com.atguigu.yygh.common.config;  
  
**import** java.lang.reflect.Method;  
**import** java.time.Duration;  
  
@Configuration  
@EnableCaching //开启缓存处理  
**public class** RedisConfig {

*/\*\*  
 \* 自定义生成 key的规则  
 \** ***@return*** *\*/* @Bean  
 **public** KeyGenerator keyGenerator() {

**return new** KeyGenerator() {  
 @Override  
 **public** Object generate(Object target, Method method, Object... params) {  
 StringBuilder sb = **new** StringBuilder();  
 sb.append(target.getClass().getName());  
 sb.append(method.getName());  
 **for** (Object obj : params) {  
 sb.append(obj.toString());  
 }  
 **return** sb.toString();  
 }  
 };  
 }  
 */\*\*  
 \* 设置RedisTemplate规则  
 \** ***@param redisConnectionFactory*** *\** ***@return*** *\*/* @Bean  
 **public** RedisTemplate<Object, Object> redisTemplate(RedisConnectionFactory redisConnectionFactory) {

RedisTemplate<Object, Object> redisTemplate = **new** RedisTemplate<>();  
 redisTemplate.setConnectionFactory(redisConnectionFactory);  
 Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = **new** Jackson2JsonRedisSerializer(Object.**class**);

*//解决查询缓存转换异常的问题* ObjectMapper om = **new** ObjectMapper();  
*// 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* om.setVisibility(PropertyAccessor.***ALL***, JsonAutoDetect.Visibility.***ANY***);

*//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* om.enableDefaultTyping(ObjectMapper.DefaultTyping.***NON\_FINAL***);  
 jackson2JsonRedisSerializer.setObjectMapper(om);

*//序列号key value* redisTemplate.setKeySerializer(**new** StringRedisSerializer());  
 redisTemplate.setValueSerializer(jackson2JsonRedisSerializer);  
 redisTemplate.setHashKeySerializer(**new** StringRedisSerializer());  
 redisTemplate.setHashValueSerializer(jackson2JsonRedisSerializer);  
  
 redisTemplate.afterPropertiesSet();  
 **return** redisTemplate;  
 }  
 */\*\*  
 \* 设置CacheManager缓存规则  
 \** ***@param factory*** *\** ***@return*** *\*/* @Bean  
 **public** CacheManager cacheManager(RedisConnectionFactory factory) {

RedisSerializer<String> redisSerializer = **new** StringRedisSerializer();  
 Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = **new** Jackson2JsonRedisSerializer(Object.**class**);

*//解决查询缓存转换异常的问题* ObjectMapper om = **new** ObjectMapper();  
 om.setVisibility(PropertyAccessor.***ALL***, JsonAutoDetect.Visibility.***ANY***);  
 om.enableDefaultTyping(ObjectMapper.DefaultTyping.***NON\_FINAL***);  
 jackson2JsonRedisSerializer.setObjectMapper(om);

*//配置序列化（解决乱码的问题）,过期时间600秒* RedisCacheConfiguration config = RedisCacheConfiguration.*defaultCacheConfig*()  
 .entryTtl(Duration.*ofSeconds*(600))  
 .serializeKeysWith(RedisSerializationContext.SerializationPair.*fromSerializer*(redisSerializer))  
 .serializeValuesWith(RedisSerializationContext.SerializationPair.*fromSerializer*(jackson2JsonRedisSerializer))  
 .disableCachingNullValues();  
  
 RedisCacheManager cacheManager = RedisCacheManager.*builder*(factory)  
 .cacheDefaults(config)  
 .build();  
 **return** cacheManager;  
 }  
}

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。

#### 1.3 service-cmn添加redis配置

spring.redis.host=192.168.44.165

spring.redis.port=6379

spring.redis.database= 0

spring.redis.timeout=1800000

spring.redis.lettuce.pool.max-active=20

spring.redis.lettuce.pool.max-wait=-1

#最大阻塞等待时间(负数表示没限制)

spring.redis.lettuce.pool.max-idle=5

spring.redis.lettuce.pool.min-idle=0

### 2、使用Spring Cache

#### 2.1 常用缓存标签

##### 2.1.2 缓存@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。

查看源码，属性值如下：

| 属性/方法名 | 解释 |
| --- | --- |
| value | 缓存名，必填，它指定了你的缓存存放在哪块命名空间 |
| cacheNames | 与 value 差不多，二选一即可 |
| key | 可选属性，可以使用 SpEL 标签自定义缓存的key |

##### 2.1.2 缓存@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。

查看源码，属性值如下：

| 属性/方法名 | 解释 |
| --- | --- |
| value | 缓存名，必填，它指定了你的缓存存放在哪块命名空间 |
| cacheNames | 与 value 差不多，二选一即可 |
| key | 可选属性，可以使用 SpEL 标签自定义缓存的key |

##### 2.1.3 缓存@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

查看源码，属性值如下：

| 属性/方法名 | 解释 |
| --- | --- |
| value | 缓存名，必填，它指定了你的缓存存放在哪块命名空间 |
| cacheNames | 与 value 差不多，二选一即可 |
| key | 可选属性，可以使用 SpEL 标签自定义缓存的key |
| allEntries | 是否清空所有缓存，默认为 false。如果指定为 true，则方法调用后将立即清空所有的缓存 |
| beforeInvocation | 是否在方法执行前就清空，默认为 false。如果指定为 true，则在方法执行前就会清空缓存 |

#### 2.2 数据字典应用

改造com.atguigu.yygh.cmn.service.impl.DictServiceImpl类方法

@Service  
**public class** DictServiceImpl **extends** ServiceImpl<DictMapper, Dict> **implements** DictService {  
  
 *//根据数据id查询子数据列表* @Override  
 @Cacheable(value = "**dict**",keyGenerator = "**keyGenerator**")**public** List<Dict> findChlidData(Long id) {  
 QueryWrapper<Dict> wrapper = **new** QueryWrapper<>();  
 wrapper.eq(**"parent\_id"**,id);  
 List<Dict> dictList = **baseMapper**.selectList(wrapper);  
 *//向list集合每个dict对象中设置hasChildren* **for** (Dict dict:dictList) {  
 Long dictId = dict.getId();  
 **boolean** isChild = **this**.isChildren(dictId);  
 dict.setHasChildren(isChild);  
 }  
 **return** dictList;  
 }  
  
 *//导出数据字典接口* @Override  
 **public void** exportDictData(HttpServletResponse response) {  
 *//设置下载信息* response.setContentType(**"application/vnd.ms-excel"**);  
 response.setCharacterEncoding(**"utf-8"**);  
 String fileName = **"dict"**;  
 response.setHeader(**"Content-disposition"**, **"attachment;filename="**+ fileName + **".xlsx"**);  
 *//查询数据库* List<Dict> dictList = **baseMapper**.selectList(**null**);  
 *//Dict -- DictEeVo* List<DictEeVo> dictVoList = **new** ArrayList<>();  
 **for**(Dict dict:dictList) {  
 DictEeVo dictEeVo = **new** DictEeVo();  
 *// dictEeVo.setId(dict.getId());* BeanUtils.*copyProperties*(dict,dictEeVo);  
 dictVoList.add(dictEeVo);  
 }  
 *//调用方法进行写操作* **try** {  
 EasyExcel.*write*(response.getOutputStream(), DictEeVo.**class**).sheet(**"dict"**)  
 .doWrite(dictVoList);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 *//导入数据字典* @Override  
 @CacheEvict(value = **"dict"**, allEntries=**true**)  
 **public void** importDictData(MultipartFile file) {  
 **try** {  
 EasyExcel.*read*(file.getInputStream(),DictEeVo.**class**,**new** DictListener(**baseMapper**)).sheet().doRead();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }

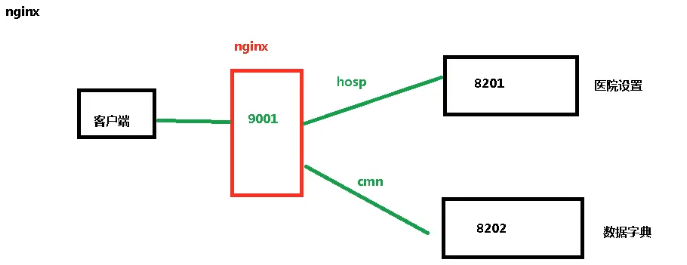
}

# day07 mongodb

## 一、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、下载安装nginx（window版）

2、配置nginx

server {

listen 9001;

server\_name localhost;

location ~ /hosp/ {

proxy\_pass http://localhost:8201;

}

location ~ /cmn/ {

proxy\_pass http://localhost:8202;

}

}

3、调整/config/dev.env.js中的BASE\_API

BASE\_API: 'http://localhost:9001'

说明：

后续我们会使用Spring Cloud Gateway网关，将替代nginx网关

## 二、MongoDB简介

### 1、NoSQL简介

NoSQL(NoSQL = Not Only SQL)，意即反SQL运动，指的是非关系型的数据库，是一项全新的数据库革命性运动，早期就有人提出，发展至2009年趋势越发高涨。NoSQL的拥护者们提倡运用非关系型的数据存储，相对于目前铺天盖地的关系型数据库运用，这一概念无疑是一种全新的思维的注入

**为什么使用NoSQL:**

* 对数据库高并发读写。
* 对海量数据的高效率存储和访问。
* 对数据库的高可扩展性和高可用性。

**弱点：**

* 数据库事务一致性需求
* 数据库的写实时性和读实时性需求
* 对复杂的SQL查询，特别是多表关联查询的需求

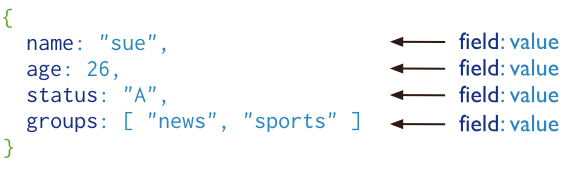
### 2、什么是MongoDB

MongoDB 是由C++语言编写的，是一个基于分布式文件存储的开源数据库系统。

在高负载的情况下，添加更多的节点，可以保证服务器性能。

MongoDB 旨在为WEB应用提供可扩展的高性能数据存储解决方案。

**MongoDB 将数据存储为一个文档**，数据结构由键值(key=>value)对组成。MongoDB 文档类似于 JSON 对象。字段值可以包含其他文档，数组及文档数组。



### 3、MongoDB 特点

1、MongoDB 是一个面向文档存储的数据库，操作起来比较简单和容易。

2、你可以在MongoDB记录中设置任何属性的索引 (如：FirstName="Sameer",Address="8 Gandhi Road")来实现更快的排序。

3、你可以通过本地或者网络创建数据镜像，这使得MongoDB有更强的扩展性。

4、如果负载的增加（需要更多的存储空间和更强的处理能力） ，它可以分布在计算机网络中的其他节点上这就是所谓的分片。

5、Mongo支持丰富的查询表达式。查询指令使用JSON形式的标记，可轻易查询文档中内嵌的对象及数组。

6、MongoDb 使用update()命令可以实现替换完成的文档（数据）或者一些指定的数据字段 。

7、Mongodb中的Map/reduce主要是用来对数据进行批量处理和聚合操作。

8、Map和Reduce。Map函数调用emit(key,value)遍历集合中所有的记录，将key与value传给Reduce函数进行处理。

9、Map函数和Reduce函数是使用Javascript编写的，并可以通过db.runCommand或mapreduce命令来执行MapReduce操作。

10、GridFS是MongoDB中的一个内置功能，可以用于存放大量小文件。

11、MongoDB允许在服务端执行脚本，可以用Javascript编写某个函数，直接在服务端执行，也可以把函数的定义存储在服务端，下次直接调用即可。

12、MongoDB支持各种编程语言:RUBY，PYTHON，JAVA，C++，PHP，C#等多种语言。

13、MongoDB安装简单。

### 4、安装mongodb

#拉取镜像

docker pull mongo:latest

#创建和启动容器

docker run -d --restart=always -p 27017:27017 --name mymongo -v /data/db:/data/db -d mongo

#进入容器

docker exec -it mymongo/bin/bash

#使用MongoDB客户端进行操作

mongo

> show dbs #查询所有的数据库

admin 0.000GB

config 0.000GB

local 0.000GB

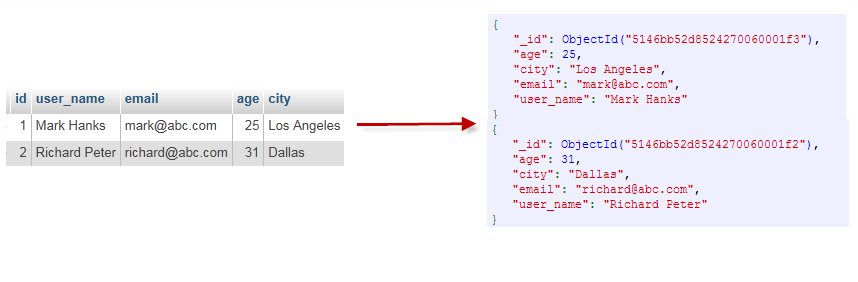
### 5、MongoDB 概念解析

不管我们学习什么数据库都应该学习其中的基础概念，在mongodb中基本的概念是文档、集合、数据库，下面我们挨个介绍。

下表将帮助您更容易理解Mongo中的一些概念：

|  |  |  |
| --- | --- | --- |
| SQL术语/概念 | MongoDB术语/概念 | 解释/说明 |
| database | database | 数据库 |
| table | collection | 数据库表/集合 |
| row | document | 数据记录行/文档 |
| column | field | 数据字段/域 |
| index | index | 索引 |
| table joins |  | 表连接,MongoDB不支持 |
| primary key | primary key | 主键,MongoDB自动将\_id字段设置为主键 |

通过下图实例，我们也可以更直观的的了解Mongo中的一些概念：



#### 5.1 数据库

一个mongodb中可以建立多个数据库

常用操作：

1、 Help查看命令提示

db.help();

2、 切换/创建数据库

use test

如果数据库不存在，则创建数据库，否则切换到指定数据库

3、 查询所有数据库

show dbs;

4、 删除当前使用数据库

db.dropDatabase();

5、 查看当前使用的数据库

db.getName();

6、 显示当前db状态

db.stats();

7、 当前db版本

db.version();

8、 查看当前db的链接机器地址

db.getMongo〇;

#### 5.2 文档

文档是一组键值(key-value)对(即BSON)。MongoDB 的文档不需要设置相同的字段，并且相同的字段不需要相同的数据类型，这与关系型数据库有很大的区别，也是 MongoDB 非常突出的特点。

下表列出了 RDBMS 与 MongoDB 对应的术语：

|  |  |
| --- | --- |
| RDBMS | MongoDB |
| 数据库 | 数据库 |
| 表格 | 集合 |
| 行 | 文档 |
| 列 | 字段 |
| 表联合 | 嵌入文档 |
| 主键 | 主键 (MongoDB 提供了 key 为 \_id ) |

**需要注意的是：**

1、文档中的键/值对是有序的。

2、文档中的值不仅可以是在双引号里面的字符串，还可以是其他几种数据类型（甚至可以是整个嵌入的文档)。

3、MongoDB区分类型和大小写。

4、MongoDB的文档不能有重复的键。

5、文档的键是字符串。除了少数例外情况，键可以使用任意UTF-8字符。

**文档键命名规范：**

1、键不能含有\0 (空字符)。这个字符用来表示键的结尾。

2、.和$有特别的意义，只有在特定环境下才能使用。

3、以下划线"\_"开头的键是保留的(不是严格要求的)。

#### 5.3 集合

集合就是 MongoDB 文档组，类似于 RDBMS （关系数据库管理系统：Relational Database Management System)中的表格。

集合存在于数据库中，集合没有固定的结构，这意味着你在对集合可以插入不同格式和类型的数据，但通常情况下我们插入集合的数据都会有一定的关联性。

常用命令：

1、 创建一个集合（table)

db.createCollection( "collName");

2、 得到指定名称的集合（table )

db.getCollection("user");

#### 5.4 MongoDB 数据类型

下表为MongoDB中常用的几种数据类型：

|  |  |
| --- | --- |
| 数据类型 | 描述 |
| String | 字符串。存储数据常用的数据类型。在 MongoDB 中，UTF-8 编码的字符串才是合法的。 |
| Integer | 整型数值。用于存储数值。根据你所采用的服务器，可分为 32 位或 64 位。 |
| Boolean | 布尔值。用于存储布尔值（真/假）。 |
| Double | 双精度浮点值。用于存储浮点值。 |
| Min/Max keys | 将一个值与 BSON（二进制的 JSON）元素的最低值和最高值相对比。 |
| Arrays | 用于将数组或列表或多个值存储为一个键。 |
| Timestamp | 时间戳。记录文档修改或添加的具体时间。 |
| Object | 用于内嵌文档。 |
| Null | 用于创建空值。 |
| Symbol | 符号。该数据类型基本上等同于字符串类型，但不同的是，它一般用于采用特殊符号类型的语言。 |
| Date | 日期时间。用 UNIX 时间格式来存储当前日期或时间。你可以指定自己的日期时间：创建 Date 对象，传入年月日信息。 |
| Object ID | 对象 ID。用于创建文档的 ID。 |
| Binary Data | 二进制数据。用于存储二进制数据。 |
| Code | 代码类型。用于在文档中存储 JavaScript 代码。 |
| Regular expression | 正则表达式类型。用于存储正则表达式。 |

### 5、适用场景

**适用场景：**

1、网站数据：Mongo非常适合实时的插入，更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。

2、缓存：由于性能很高，Mongo也适合作为信息基础设施的缓存层。在系统重启之后，由M ongo搭建的持久化缓存层可以避免下层的数据源过载。

3、大尺寸，低价值的数据：使用传统的关系型数据库存储一些数据时可能会比较昂贵， 在此之前，很多时候程序员往往会选择传统的文件进行存储。

4、高伸缩性的场景：Mongo非常适合由数十或数百台服务器组成的数据库。Mongo的路线图中已经包含对Map Reduce弓摩的内置支持。

5、用于对象及 JSON数据的存储：Mongo的BSON数据格式非常适合文档化格式的存储 及查询。

**不适用场合：**

1、高度事务性的系统：例如银行或会计系统。传统的关系型数据库目前还是更适用于需要大量原子性复杂事务的应用程序。

2、传统的商业智能应用：针对特定问题的BI数据库会对产生高度优化的查询方式。对于此类应用，数据仓库可能是更合适的选择。

## 三、MongoDB入门

### 1、常用操作

#### 1.1 INSERT

> db.User.save({name:'zhangsan',age:21,sex:true})

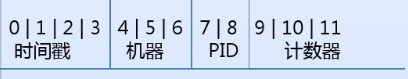
> db.User.find()

{"\_id": Objectld("4f69e680c9106ee2ec95da66"), "name": "zhangsan", "age": 21,

"sex": true}

\_id组合

Objectld是、id”的默认类型。Objectld使用12字节的存储空间，每个字节二位十六进制数字， 是一个24位的字符串



1. 时间戳：时间不断变化的

2. 机器：主机的唯\_标识码。通常是机器主机名的散列值，这样可以确保不同主机

生成不同的Objectld ,不产生冲突。

3. PID:为了确保在同一台机器上并发的多个进程产生的Objectld是唯一的，

所以加上进程标识符(PID).

4. 计数器：前9个字节保证了同一秒钟不同机器不同进程产生的Objectld是唯一的。

后3个字节就是一个自动增加的计数器，确保相同进程同一秒产生的Objectld也是

不一样。同一秒最多允许每个进程拥有IS 777 2托个不同的Objectld。

#### 1.2 Query

1、WHERE

# select \* from User where name = 'zhangsan'

> db.User.find({name:"zhangsan"})

2、FIELDS

# select name, age from User where age = 21

> db.User.find({age:21}, {'name':1, 'age':1})

3、SORT

在 MongoDB 中使用 sort() 方法对数据进行排序，sort() 方法可以通过参数指定排序的字段，并使用 1 和 -1 来指定排序的方式，其中 1 为升序排列，而 -1 是用于降序排列。

# select \* from User order by age

> db.User.find().sort({age:1})

4、SUCE

在 MongoDB 中使用 limit()方法来读取指定数量的数据，skip()方法来跳过指定数量的数据

# select \* from User skip 2 limit 3

> db.User.find().skip(0).limit(3)

5、IN

# select \* from User where age in (21, 26, 32)

> db.User.find({age:{$in:[21,26,32]}})

6、COUNT

# select count(\*) from User where age >20

> db.User.find({age:{$gt:20}}).count()

7、0R

# select \* from User where age = 21 or age = 28

> db.User.find({$or:[{age:21}, {age:28}]})

#### 1.3 Update

可直接用类似T-SQL条件表达式更新，或用SaveO更新从数据库返回到文档对象。

# update Userset age = 100, sex = 0 where name = 'user1'

> db.User.update({name:"zhangsan"}, {$set:{age:100, sex:0}})

Update()有几个参数需要注意。

db.collection.update(criteria, objNew, upsert, mult)

criteria:需要更新的条件表达式

objNew:更新表达式

upsert:如FI标记录不存在，是否插入新文档。

multi:是否更新多个文档。

#### 1.4 Remove

removeO用于删除单个或全部文档，删除后的文档无法恢复。

> db.User.remove(id)

//移除对应id的行

> db.User.remove({})

//移除所有

#### 1.5 aggregate

MongoDB中聚合(aggregate)主要用于处理数据(诸如统计平均值,求和等)，并返回计算后的数据结果。有点类似sql语句中的 count(\*)

##### 1.5.1 插入数据

>db.article.insert({

title: 'MongoDB Overview',

description: 'MongoDB is no sql database',

by\_user: 'runoob.com',

url: 'http://www.runoob.com',

tags: ['mongodb', 'database', 'NoSQL'],

likes: 100

})

>db.article.insert({

title: 'NoSQL Overview',

description: 'No sql database is very fast',

by\_user: 'runoob.com',

url: 'http://www.runoob.com',

tags: ['mongodb', 'database', 'NoSQL'],

likes: 10

})

>db.article.insert({

title: 'Neo4j Overview',

description: 'Neo4j is no sql database',

by\_user: 'Neo4j',

url: 'http://www.neo4j.com',

tags: ['neo4j', 'database', 'NoSQL'],

likes: 750

})

##### 1.5.2 统计sum

现在我们通过以上集合计算每个作者所写的文章数

# select by\_user, count(\*) from article group by by\_user

> db.article.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : 1}}}])

{

"result" : [

{

"\_id" : "runoob.com",

"num\_tutorial" : 2

},

{

"\_id" : "Neo4j",

"num\_tutorial" : 1

}

],

"ok" : 1

}

在上面的例子中，我们通过字段 by\_user 字段对数据进行分组，并计算 by\_user 字段相同值的总和。

##### 1.5.3 常见的聚合表达式

|  |  |  |
| --- | --- | --- |
| **表达式** | **描述** | **实例** |
| $sum | 计算总和 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : "$likes"}}}]) |
| $avg | 计算平均值 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$avg : "$likes"}}}]) |
| $min | 获取集合中所有文档对应值得最小值。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$min : "$likes"}}}]) |
| $max | 获取集合中所有文档对应值得最大值。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$max : "$likes"}}}]) |
| $push | 在结果文档中插入值到一个数组中。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", url : {$push: "$url"}}}]) |
| $addToSet | 在结果文档中插入值到一个数组中，但不创建副本。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", url : {$addToSet : "$url"}}}]) |
| $first | 根据资源文档的排序获取第一个文档数据。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", first\_url : {$first : "$url"}}}]) |
| $last | 根据资源文档的排序获取最后一个文档数据 | db.mycol.aggregate([{$group : {\_id : "$by\_user", last\_url : {$last : "$url"}}}]) |

### 2 索引

索引通常能够极大的提高查询的效率，如果没有索引，MongoDB在读取数据时必须扫描集合中的每个文件并选取那些符合查询条件的记录。

这种扫描全集合的查询效率是非常低的，特别在处理大量的数据时，查询可以要花费几十秒甚至几分钟，这对网站的性能是非常致命的。

索引是特殊的数据结构，索引存储在一个易于遍历读取的数据集合中，索引是对数据库表中一列或多列的值进行排序的一种结构。

>db.User.createIndex({"name":1})

语法中 name值为你要创建的索引字段，1 为指定按升序创建索引，如果你想按降序来创建索引指定为 -1 即可

## 四、Spring boot集成mongodb

### 1、集成简介

spring-data-mongodb提供了MongoTemplate与MongoRepository两种方式访问mongodb，MongoRepository操作简单，MongoTemplate操作灵活，我们在项目中可以灵活适用这两种方式操作mongodb，MongoRepository的缺点是不够灵活，MongoTemplate正好可以弥补不足。

### 2、搭建开发环境

#### 2.1 初始化工程

使用 Spring Initializr 快速初始化一个 Spring Boot 工程

Group：com.atguigu

Artifact：mongodb

#### 2.2 引入依赖

修改pom.xml

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-mongodb</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</dependency>

<dependency>

<groupId>joda-time</groupId>

<artifactId>joda-time</artifactId>

<version>2.10.1</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</dependency>

</dependencies>

#### 2.3 添加配置

在application.properties文件添加配置

spring.data.mongodb.uri=mongodb://47.93.118.241:27017/test

### 3、基于MongoTemplate 开发CRUD

#### 3.1 添加实体

添加com.atguigu.mongodb.entity.User类

@Data

@Document("User") //指定操作的集合或数据库表

public class User {

@Id //自动生成的id

private String id;

private String name;

private Integer age;

private String email;

private String createDate;

}

#### 3.2 实现

常用方法

mongoTemplate.findAll(User.class): 查询User文档的全部数据

mongoTemplate.findById(<id>, User.class): 查询User文档id为id的数据

mongoTemplate.find(query, User.class);: 根据query内的查询条件查询

mongoTemplate.upsert(query, update, User.class): 修改

mongoTemplate.remove(query, User.class): 删除

mongoTemplate.insert(User): 新增

Query对象

1、创建一个query对象（用来封装所有条件对象)，再创建一个criteria对象（用来构建条件）

2、 精准条件：criteria.and(“key”).is(“条件”)

模糊条件：criteria.and(“key”).regex(“条件”)

3、封装条件：query.addCriteria(criteria)

4、大于（创建新的criteria）：Criteria gt = Criteria.where(“key”).gt（“条件”）

小于（创建新的criteria）：Criteria lt = Criteria.where(“key”).lt（“条件”）

5、Query.addCriteria(new Criteria().andOperator(gt,lt));

6、一个query中只能有一个andOperator()。其参数也可以是Criteria数组。

7、排序 ：query.with（new Sort(Sort.Direction.ASC, "age"). and(new Sort(Sort.Direction.DESC, "date")))

#### 3.3 添加测试类

在/test/java下面添加测试类：

@SpringBootTest

class DemomogoApplicationTests {

@Autowired

private MongoTemplate mongoTemplate;

//添加

@Test

public void createUser() {

User user = new User();

user.setAge(20);

user.setName("test");

user.setEmail("4932200@qq.com");

User user1 = mongoTemplate.insert(user);

System.out.println(user1);

}

//查询所有

@Test

public void findUser() {

List<User> userList = mongoTemplate.findAll(User.class);

System.out.println(userList);

}

//根据id查询

@Test

public void getById() {

User user =

mongoTemplate.findById("5ffbfa2ac290f356edf9b5aa", User.class);

System.out.println(user);

}

//条件查询

@Test

public void findUserList() {

Query query = new Query(Criteria

.where("name").is("test")

.and("age").is(20));

List<User> userList = mongoTemplate.find(query, User.class);

System.out.println(userList);

}

//模糊查询

@Test

public void findUsersLikeName() {

String name = "est";

String regex = String.format("%s%s%s", "^.\*", name, ".\*$");

Pattern pattern = Pattern.compile(regex, Pattern.CASE\_INSENSITIVE);

Query query = new Query(Criteria.where("name").regex(pattern));

List<User> userList = mongoTemplate.find(query, User.class);

System.out.println(userList);

}

//分页查询

@Test

public void findUsersPage() {

String name = "est";

int pageNo = 1;

int pageSize = 10;

//条件构建

Query query = new Query();

String regex = String.format("%s%s%s", "^.\*", name, ".\*$");

Pattern pattern = Pattern.compile(regex, Pattern.CASE\_INSENSITIVE);

query.addCriteria(Criteria.where("name").regex(pattern));

//分页构建

//查询记录数

int totalCount = (int) mongoTemplate.count(query, User.class);

//分页

List<User> userList = mongoTemplate.find(query.skip((pageNo - 1) \* pageSize).limit(pageSize), User.class);

Map<String, Object> pageMap = new HashMap<>();

pageMap.put("list", userList);

pageMap.put("totalCount",totalCount);

System.out.println(pageMap);

}

//修改

@Test

public void updateUser() {

User user = mongoTemplate.findById("5ffbfa2ac290f356edf9b5aa", User.class);

user.setName("test\_1");

user.setAge(25);

user.setEmail("493220990@qq.com");

Query query = new Query(Criteria.where("\_id").is(user.getId()));

Update update = new Update();

update.set("name", user.getName());

update.set("age", user.getAge());

update.set("email", user.getEmail());

UpdateResult result = mongoTemplate.upsert(query, update, User.class);

long count = result.getModifiedCount();

System.out.println(count);

}

//删除操作

@Test

public void delete() {

Query query =

new Query(Criteria.where("\_id").is("5ffbfa2ac290f356edf9b5aa"));

DeleteResult result = mongoTemplate.remove(query, User.class);

long count = result.getDeletedCount();

System.out.println(count);

}

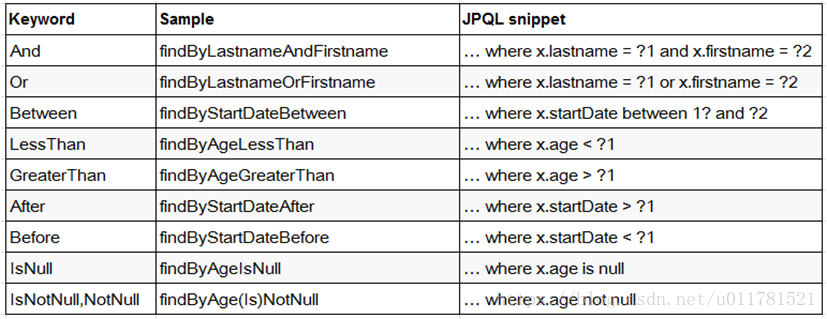
}

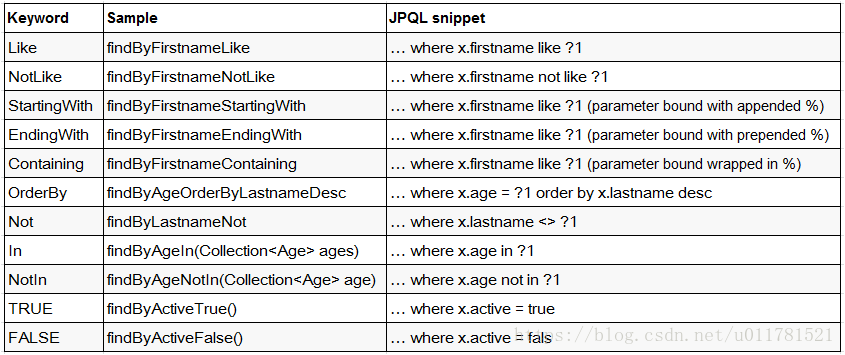
### 4、基于MongoRepository开发CRUD

#### 4.1 实现

Spring Data提供了对mongodb数据访问的支持，我们只需要继承MongoRepository类，按照Spring Data规范就可以了

SpringData 方法定义规范





1、不是随便声明的，而需要符合一定的规范

2、 查询方法以find | read | get开头

3、 涉及条件查询时，条件的属性用条件关键字连接

4、 要注意的是：条件属性首字母需要大写

5、 支持属性的级联查询，但若当前类有符合条件的属性则优先使用，而不使用级联属性,若需要使用级联属性，则属性之间使用\_强制进行连接

#### 4.2 添加Repository类

添加com.atguigu.mongodb.repository.UserRepository类

package com.atguigu.mongodb.repository;

import com.atguigu.mongodb.entity.User;

import org.springframework.data.mongodb.repository.MongoRepository;

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface UserRepository extends MongoRepository<User, String> {

}

#### 4.3 添加测试类

在/test/java下面添加测试类：

@SpringBootTest

class DemomogoApplicationTests1 {

@Autowired

private UserRepository userRepository;

//添加

@Test

public void createUser() {

User user = new User();

user.setAge(20);

user.setName("张三");

user.setEmail("3332200@qq.com");

User user1 = userRepository.save(user);

}

//查询所有

@Test

public void findUser() {

List<User> userList = userRepository.findAll();

System.out.println(userList);

}

//id查询

@Test

public void getById() {

User user = userRepository.findById("5ffbfe8197f24a07007bd6ce").get();

System.out.println(user);

}

//条件查询

@Test

public void findUserList() {

User user = new User();

user.setName("张三");

user.setAge(20);

Example<User> userExample = Example.of(user);

List<User> userList = userRepository.findAll(userExample);

System.out.println(userList);

}

//模糊查询

@Test

public void findUsersLikeName() {

//创建匹配器，即如何使用查询条件

ExampleMatcher matcher = ExampleMatcher.matching() //构建对象

.withStringMatcher(ExampleMatcher.StringMatcher.CONTAINING) //改变默认字符串匹配方式：模糊查询

.withIgnoreCase(true); //改变默认大小写忽略方式：忽略大小写

User user = new User();

user.setName("三");

Example<User> userExample = Example.of(user, matcher);

List<User> userList = userRepository.findAll(userExample);

System.out.println(userList);

}

//分页查询

@Test

public void findUsersPage() {

Sort sort = Sort.by(Sort.Direction.DESC, "age");

//0为第一页

Pageable pageable = PageRequest.of(0, 10, sort);

//创建匹配器，即如何使用查询条件

ExampleMatcher matcher = ExampleMatcher.matching() //构建对象

.withStringMatcher(ExampleMatcher.StringMatcher.CONTAINING) //改变默认字符串匹配方式：模糊查询

.withIgnoreCase(true); //改变默认大小写忽略方式：忽略大小写

User user = new User();

user.setName("三");

Example<User> userExample = Example.of(user, matcher);

//创建实例

Example<User> example = Example.of(user, matcher);

Page<User> pages = userRepository.findAll(example, pageable);

System.out.println(pages);

}

//修改

@Test

public void updateUser() {

User user = userRepository.findById("5ffbfe8197f24a07007bd6ce").get();

user.setName("张三\_1");

user.setAge(25);

user.setEmail("883220990@qq.com");

User save = userRepository.save(user);

System.out.println(save);

}

//删除

@Test

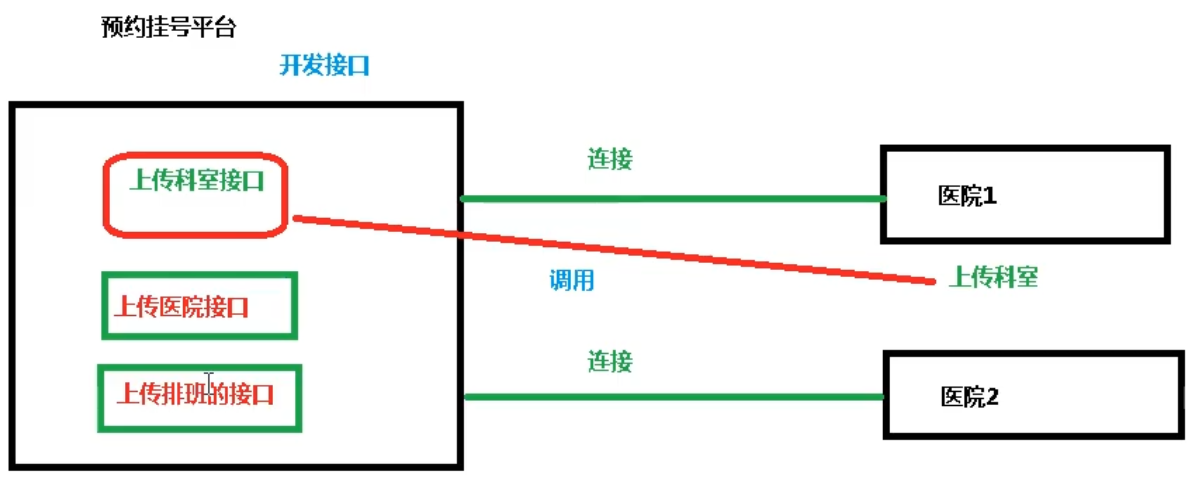
public void delete() {

userRepository.deleteById("5ffbfe8197f24a07007bd6ce");

}

}

# day08 医院上传接口



**准备工具类：**



## 上传医院接口

参考《尚医通API接口文档.docx》业务接口4.1上传医院

参考《医院接口模拟系统.docx》进行接口测试与数据上传

### 集成mongodb

#### 1.1 添加依赖

service-hosp模块pom.xml添加依赖

|  |
| --- |
| <**dependency**> <**groupId**>org.springframework.boot</**groupId**> <**artifactId**>spring-boot-starter-data-mongodb</**artifactId**>  </**dependency**> |

#### 1.2 添加配置

service-hosp模块在application.properties文件添加配置

|  |
| --- |
| **spring.data.mongodb.uri**=**mongodb://192.168.44.165:27017/yygh\_hosp** |

说明：改为自己安装mongodb的ip地址

### 添加医院基础类

#### 2.1 添加model

说明：由于实体对象没有逻辑，我们已经统一导入

**package** com.atguigu.yygh.model.hosp;  
@Data  
@ApiModel(description = **"Hospital"**)  
@Document(**"Hospital"**)  
**public class** Hospital **extends** BaseMongoEntity {  
   
 **private static final long *serialVersionUID*** = 1L;  
   
 @ApiModelProperty(value = **"医院编号"**)  
 @Indexed(unique = **true**) *//唯一索引* **private** String **hoscode**;  
  
 @ApiModelProperty(value = **"医院名称"**)  
 @Indexed *//普通索引* **private** String **hosname**;  
  
 @ApiModelProperty(value = **"医院类型"**)  
 **private** String **hostype**;  
  
 @ApiModelProperty(value = **"省code"**)  
 **private** String **provinceCode**;  
  
 @ApiModelProperty(value = **"市code"**)  
 **private** String **cityCode**;  
  
 @ApiModelProperty(value = **"区code"**)  
 **private** String **districtCode**;  
  
 @ApiModelProperty(value = **"详情地址"**)  
 **private** String **address**;  
  
 @ApiModelProperty(value = **"医院logo"**)  
 **private** String **logoData**;  
  
 @ApiModelProperty(value = **"医院简介"**)  
 **private** String **intro**;  
  
 @ApiModelProperty(value = **"坐车路线"**)  
 **private** String **route**;  
  
 @ApiModelProperty(value = **"状态 0：未上线 1：已上线"**)  
 **private** Integer **status**;  
  
 *//预约规则* @ApiModelProperty(value = **"预约规则"**)  
 **private** BookingRule **bookingRule**;  
  
 **public void** setBookingRule(String bookingRule) {  
 **this**.**bookingRule** = JSONObject.*parseObject*(bookingRule, BookingRule.**class**);  
 }  
  
}

#### 2.2 添加Repository

**package** com.atguigu.yygh.hosp.repository;  
  
@Repository *//交给spring管理***public interface** HospitalRepository **extends** MongoRepository<Hospital,String> {

*//判断是否存在数据* Hospital getHospitalByHoscode(String hoscode);  
  
 *//根据医院名称查询* List<Hospital> findHospitalByHosnameLike(String hosname);  
}

#### 2.3 添加service接口及实现类

2.3.1 添加com.atguigu.yygh.hosp.service.HospitalService接口

|  |
| --- |
| **public interface** HospitalService {   } |

2.3.2 添加com.atguigu.yygh.hosp.service.impl.HospitalServiceImpl接口实现

|  |
| --- |
| **package** com.atguigu.yygh.hosp.service.impl;  @Service **public class** HospitalServiceImpl **implements** HospitalService {   @Autowired  **private** HospitalRepository **hospitalRepository**; } |

#### 2.4 添加controller

添加com.atguigu.yygh.hosp.api.ApiController

|  |
| --- |
| **package** com.atguigu.yygh.hosp.api; @Api(tags = **"医院管理API接口"**) @RestController @RequestMapping(**"/api/hosp"**) **public class** ApiController {   @Autowired  **private** HospitalService **hospitalService**;} |

说明：平台对外开发的接口都写在该Controller类

### 上传医院

参考《尚医通API接口文档.doc》业务接口4.1上传医院

医院编号是平台分配的，全局唯一，上传医院接口可以多次调用，如果存在相同编号的为更新操作

#### 3.1 接口数据分析

|  |
| --- |
| { **"hoscode"**: **"1000\_0"**, **"hosname"**: **"北京协和医院"**, **"hostype"**: **"1"**, **"provinceCode"**: **"110000"**, **"cityCode"**: **"110100"**, **"districtCode"**: **"110102"**, **"address"**: **"大望路"**, **"intro"**: **"北京协和医院是集医疗、教学、科研于一体的大型三级甲等综合医院，是国家卫生计生委...目标而继续努力。"**, **"route"**: **"东院区乘车路线：106、...更多乘车路线详见须知。"**, **"logoData"**: **"iVBORw0KGgoAAAA...NSUhEUg=="**, **"bookingRule"**: {  **"cycle"**: **"1"**,  **"releaseTime"**: **"08:30"**,  **"stopTime"**: **"11:30"**,  **"quitDay"**: **"-1"**,  **"quitTime"**: **"15:30"**,  **"rule"**: [ **"西院区预约号取号地点：西院区门诊楼一层大厅挂号窗口取号"**, **"东院区预约号取号地点：东院区老门诊楼一层大厅挂号窗口或新门诊楼各楼层挂号/收费窗口取号"** ]  } } |

说明：

1. 数据分为医院基本信息与预约规则信息
2. 医院logo转换为base64字符串
3. 预约规则信息属于医院基本信息的一个属性
4. 预约规则rule，以数组形式传递
5. 数据传递过来我们还要验证签名，只允许平台开通的医院可以上传数据，保证数据安全性

#### 3.2 添加service接口

1、在HospitalService 类添加接口

|  |
| --- |
| */\*\*  \* 上传医院信息  \** ***@param paramMap*** *\*/* **void** save(Map<String, Object> paramMap); |

说明：参数使用Map，减少对象封装，有利于签名校验，后续会体验到

2、在HospitalServiceImpl类添加实现

**package** com.atguigu.yygh.hosp.service.impl;  
  
@Service  
**public class** HospitalServiceImpl **implements** HospitalService {  
  
 @Autowired  
 **private** HospitalRepository **hospitalRepository**;  
  
 @Autowired  
 **private** DictFeignClient **dictFeignClient**;  
  
 @Override  
 **public void** save(Map<String, Object> paramMap) {  
 *//把参数map集合转换对象 Hospital* String mapString = JSONObject.*toJSONString*(paramMap);  
 Hospital hospital = JSONObject.*parseObject*(mapString, Hospital.**class**);  
  
 *//判断是否存在数据* String hoscode = hospital.getHoscode();  
 Hospital hospitalExist = **hospitalRepository**.getHospitalByHoscode(hoscode);  
  
 *//如果存在，进行修改* **if**(hospitalExist != **null**) {  
 hospital.setStatus(hospitalExist.getStatus());  
 hospital.setCreateTime(hospitalExist.getCreateTime());  
 hospital.setUpdateTime(**new** Date());  
 hospital.setIsDeleted(0);  
 **hospitalRepository**.save(hospital);  
 } **else** {*//如果不存在，进行添加* hospital.setStatus(0);  
 hospital.setCreateTime(**new** Date());  
 hospital.setUpdateTime(**new** Date());  
 hospital.setIsDeleted(0);  
 **hospitalRepository**.save(hospital);  
 }  
 }  
}

**说明：**

*//把参数map集合转换对象 Hospital*String mapString = JSONObject.*toJSONString*(paramMap);  
Hospital hospital = JSONObject.*parseObject*(mapString, Hospital.**class**);

Map转换为Hospital对象时，预约规则bookingRule为一个对象属性，rule为一个数组属性，因此在转换时我们要重新定义对应的set方法，不然转换不会成功

**package** com.atguigu.yygh.model.hosp;  
@Data  
@ApiModel(description = **"Hospital"**)  
@Document(**"Hospital"**)  
**public class** Hospital **extends** BaseMongoEntity {  
   
 **private static final long *serialVersionUID*** = 1L;  
   
 @ApiModelProperty(value = **"医院编号"**)  
 @Indexed(unique = **true**) *//唯一索引* **private** String **hoscode**;  
  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
 *//预约规则* @ApiModelProperty(value = **"预约规则"**)  
 **private** BookingRule **bookingRule**;  
  
 **public void** setBookingRule(String bookingRule) {  
 **this**.**bookingRule** = JSONObject.*parseObject*(bookingRule, BookingRule.**class**);  
 }  
}

**package** com.atguigu.yygh.model.hosp;  
@Data  
@ApiModel(description = **"预约规则"**)  
@Document(**"BookingRule"**)  
**public class** BookingRule {  
   
 @ApiModelProperty(value = **"预约周期"**)  
 **private** Integer **cycle**;  
  
 @ApiModelProperty(value = **"放号时间"**)  
 **private** String **releaseTime**;  
  
 @ApiModelProperty(value = **"停挂时间"**)  
 **private** String **stopTime**;  
  
 @ApiModelProperty(value = **"退号截止天数（如：就诊前一天为-1，当天为0）"**)  
 **private** Integer **quitDay**;  
  
 @ApiModelProperty(value = **"退号时间"**)  
 **private** String **quitTime**;  
  
 @ApiModelProperty(value = **"预约规则"**)  
 **private** List<String> **rule**;  
**public void** setRule(String rule) {  
 **if**(!StringUtils.*isEmpty*(rule)) {  
 **this**.**rule** = JSONArray.*parseArray*(rule, String.**class**);  
 }  
 }  
}

#### 3.3 添加repository接口

在HospitalRepository类添加接口

**package** com.atguigu.yygh.hosp.repository;  
  
@Repository *//交给spring管理***public interface** HospitalRepository **extends** MongoRepository<Hospital,String> {

*//判断是否存在数据* Hospital getHospitalByHoscode(String hoscode);  
}

#### 3.4 添加controller接口

在ApiController类添加接口

**package** com.atguigu.yygh.hosp.controller.api;  
  
@RestController  
@RequestMapping(**"/api/hosp"**)  
**public class** ApiController {  
  
 @Autowired  
 **private** HospitalService **hospitalService**;  
  
 *//上传医院接口* @PostMapping(**"saveHospital"**)  
 **public** Result saveHosp(HttpServletRequest request) {

*//获取传递过来医院信息* Map<String, String[]> requestMap = request.getParameterMap();  
 Map<String, Object> paramMap = HttpRequestHelper.*switchMap*(requestMap);  
  
 *//调用service的方法* **hospitalService**.save(paramMap);  
 **return** Result.*ok*();  
 }  
}

#### 3.5 添加帮助类

在service-util模块添加HttpRequestHelper帮助类

**package** com.atguigu.yygh.common.helper;  
  
@Slf4j  
**public class** HttpRequestHelper {  
  
 */\*\*  
 \** ***@param paramMap*** *\** ***@return*** *\*/* **public static** Map<String, Object> **switchMap**(Map<String, String[]> paramMap) {  
  
 Map<String, Object> resultMap = **new** HashMap<>();  
  
 **for** (Map.Entry<String, String[]> param : paramMap.entrySet()) {  
  
 resultMap.put(param.getKey(), param.getValue()[0]);  
 }  
 **return** resultMap;  
 }

}

#### 3.6 使用Swagger2测试上传接口

略

### 参数签名

参考《尚医通API接口文档.doc》业务接口3.1传参说明

#### 4.1 封装签名方法

在service-util模块HttpRequestHelper类添加方法

**package** com.atguigu.yygh.common.helper;  
  
@Slf4j  
**public class** HttpRequestHelper {  
  
 **public static void** main(String[] args) {  
 Map<String, Object> paramMap = **new** HashMap<>();  
 paramMap.put(**"d"**, **"4"**);  
 paramMap.put(**"b"**, **"2"**);  
 paramMap.put(**"c"**, **"3"**);  
 paramMap.put(**"a"**, **"1"**);  
 paramMap.put(**"timestamp"**, *getTimestamp*());  
 ***log***.info(*getSign*(paramMap, **"111111111"**));  
 }  
 */\*\*  
 \* 请求数据获取签名  
 \** ***@param paramMap*** *\** ***@param signKey*** *\** ***@return*** *\*/* **public static** String getSign(Map<String, Object> paramMap, String signKey) {  
 **if**(paramMap.containsKey(**"sign"**)) {  
   
 paramMap.remove(**"sign"**);  
 }  
 TreeMap<String, Object> sorted = **new** TreeMap<>(paramMap);

StringBuilder str = **new** StringBuilder();  
 **for** (Map.Entry<String, Object> param : sorted.entrySet()) {  
   
 str.append(param.getValue()).append(**"|"**);  
 }  
 str.append(signKey);  
 ***log***.info(**"加密前："** + str.toString());  
 String md5Str = MD5.*encrypt*(str.toString());  
 ***log***.info(**"加密后："** + md5Str);  
 **return** md5Str;  
 }  
 */\*\*  
 \* 签名校验  
 \** ***@param paramMap*** *\** ***@param signKey*** *\** ***@return*** *\*/* **public static boolean** isSignEquals(Map<String, Object> paramMap, String signKey) {  
   
 String sign = (String)paramMap.get(**"sign"**);  
 String md5Str = *getSign*(paramMap, signKey);  
 **if**(!sign.equals(md5Str)) {  
 **return false**;  
 }  
 **return true**;  
 }  
 */\*\*  
 \* 获取时间戳  
 \** ***@return*** *\*/* **public static long** getTimestamp() {  
   
 **return new** Date().getTime();  
 }

}

#### 4.2 上传医院添加签名校验

我们在医院设置的时候，为每个医院生成了医院编码与签名key，因此我在验证签名时要根据医院编码去动态获取签名key，然后再做签名校验

##### 4.2.1 添加获取签名key接口

1、在HospitalSetService类添加接口

|  |
| --- |
| */\*\*  \* 获取签名key  \** ***@param hoscode*** *\** ***@return*** *\*/* String getSignKey(String hoscode); |

2、在HospitalSetServiceImpl类实现接口

**package** com.atguigu.yygh.hosp.service.impl;  
  
@Service  
**public class** HospitalSetServiceImpl **extends** ServiceImpl<HospitalSetMapper, HospitalSet> **implements** HospitalSetService {  
  
 *// 根据传递过来医院编码，查询数据库，查询签名* @Override  
 **public** String getSignKey(String hoscode) {  
  
 QueryWrapper<HospitalSet> wrapper = **new** QueryWrapper<>();  
 wrapper.eq(**"hoscode"**,hoscode);  
 HospitalSet hospitalSet = **baseMapper**.selectOne(wrapper);  
  
 **if**(**null** == hospitalSet) {  
  
 **throw new** YyghException(ResultCodeEnum.***HOSPITAL\_OPEN***);  
 }  
 **if**(hospitalSet.getStatus().intValue() == 0) {  
  
 **throw new** YyghException(ResultCodeEnum.***HOSPITAL\_LOCK***);  
 }  
 **return** hospitalSet.getSignKey();  
 }

}

##### 4.2.2 修改ApiController类上传医院接口

修改ApiController类上传医院接口

**package** com.atguigu.yygh.hosp.controller.api;  
  
@RestController  
@RequestMapping(**"/api/hosp"**)  
**public class** ApiController {  
  
 @Autowired  
 **private** HospitalService **hospitalService**;  
  
 @Autowired  
 **private** HospitalSetService **hospitalSetService**;  
  
 *//上传医院接口* @PostMapping(**"saveHospital"**)  
 **public** Result saveHosp(HttpServletRequest request) {

*//获取传递过来医院信息* Map<String, String[]> requestMap = request.getParameterMap();  
 Map<String, Object> paramMap = HttpRequestHelper.*switchMap*(requestMap);  
  
 *//1 获取医院系统传递过来的签名,签名进行MD5加密* String hospSign = (String)paramMap.get(**"sign"**);  
  
 *//2 根据传递过来医院编码，查询数据库，查询签名* String hoscode = (String)paramMap.get(**"hoscode"**);  
 String signKey = **hospitalSetService**.getSignKey(hoscode);  
  
 *//3 把数据库查询签名进行MD5加密* String signKeyMd5 = MD5.*encrypt*(signKey);  
  
 *//4 判断签名是否一致* **if**(!hospSign.equals(signKeyMd5)) {  
 **throw new** YyghException(ResultCodeEnum.***SIGN\_ERROR***);  
 }  
  
 *//调用service的方法* **hospitalService**.save(paramMap);  
 **return** Result.*ok*();  
 }  
}

### 图片base64编码

#### 5.1 图片base64说明

图片的base64编码就是可以将一张图片数据编码成一串字符串，使用该字符串代替图像地址url

在前端页面中常见的base64图片的引入方式：

<img src="data:image/png;base64,iVBORw0..>

1. 优点
2. base64格式的图片是文本格式，占用内存小，转换后的大小比例大概为1/3，降低了资源服务器的消耗；

（2）网页中使用base64格式的图片时，不用再请求服务器调用图片资源，减少了服务器访问次数。

2. 缺点

（1）base64格式的文本内容较多，存储在数据库中增大了数据库服务器的压力；

（2）网页加载图片虽然不用访问服务器了，但因为base64格式的内容太多，所以加载网页的速度会降低，可能会影响用户的体验。

说明：医院logo图片小，因此上传医院logo是可以使用base64格式保存

#### 5.2 图片base64工具类

在common-util模块添加工具类

添加com.atguigu.yygh.common.util.ImageBase64Util类

|  |
| --- |
| **package** com.atguigu.yygh.common.util;  **import** org.apache.commons.codec.binary.Base64;  **import** java.io.File; **import** java.io.FileInputStream; **import** java.io.InputStream;  **public class** ImageBase64Util {  **public static void** main(String[] args) {  String imageFile= **"D:\\yygh\_work\\xh.png"**;*// 待处理的图片* System.***out***.println(*getImageString*(imageFile));  }  **public static** String getImageString(String imageFile){  InputStream is = **null**; **try** { **byte**[] data = **null**;  is = **new** FileInputStream(**new** File(imageFile));  data = **new byte**[is.available()];  is.read(data); **return new** String(Base64.*encodeBase64*(data));  } **catch** (Exception e) {  e.printStackTrace();  } **finally** { **if** (**null** != is) { **try** {  is.close();  is = **null**;  } **catch** (Exception e) {  e.printStackTrace();  }  }  } **return ""**;  } } |

#### 5.3 上传医院接口修正

图片转换为base64字符串时，该字符串中包含大量的加号“+”，服务器在解析数据时会把加号当成连接符，转换为空格，因此我们要做一下特殊处理

修改ApiController类上传接口

**package** com.atguigu.yygh.hosp.controller.api;  
  
@RestController  
@RequestMapping(**"/api/hosp"**)  
**public class** ApiController {  
  
 @Autowired  
 **private** HospitalService **hospitalService**;  
  
 @Autowired  
 **private** HospitalSetService **hospitalSetService**;  
  
 *//上传医院接口* @PostMapping(**"saveHospital"**)  
 **public** Result saveHosp(HttpServletRequest request) {  
 *//获取传递过来医院信息* Map<String, String[]> requestMap = request.getParameterMap();  
 Map<String, Object> paramMap = HttpRequestHelper.*switchMap*(requestMap);  
  
 *//1 获取医院系统传递过来的签名,签名进行MD5加密* String hospSign = (String)paramMap.get(**"sign"**);  
  
 *//2 根据传递过来医院编码，查询数据库，查询签名* String hoscode = (String)paramMap.get(**"hoscode"**);  
 String signKey = **hospitalSetService**.getSignKey(hoscode);  
  
 *//3 把数据库查询签名进行MD5加密* String signKeyMd5 = MD5.*encrypt*(signKey);  
  
 *//4 判断签名是否一致* **if**(!hospSign.equals(signKeyMd5)) {  
 **throw new** YyghException(ResultCodeEnum.***SIGN\_ERROR***);  
 }  
  
 *//传输过程中“+”转换为了“ ”，因此我们要转换回来* String logoData = (String)paramMap.get(**"logoData"**);  
 logoData = logoData.replaceAll(**" "**,**"+"**);  
 paramMap.put(**"logoData"**,logoData);  
  
 *//调用service的方法* **hospitalService**.save(paramMap);  
 **return** Result.*ok*();  
 }  
}

### 6、集成测试

参考《医院接口模拟系统.docx》进行接口测试与数据上传，后续不做说明，需要测试时即可使用

## 查询医院接口

参考《尚医通API接口文档.docx》业务接口4.4查询医院

### 1、添加service接口

1.1 在HospitalService 类添加接口

**package** com.atguigu.yygh.hosp.service;  
  
**public interface** HospitalService {  
  
 *//实现根据医院编号查询* Hospital getByHoscode(String hoscode);

}

1.2 在HospitalServiceImpl类添加实现

**package** com.atguigu.yygh.hosp.service.impl;  
  
@Service  
**public class** HospitalServiceImpl **implements** HospitalService {  
  
 @Autowired  
 **private** HospitalRepository **hospitalRepository**;

*//查询医院* @Override  
 **public** Hospital getByHoscode(String hoscode) {  
   
 Hospital hospital = **hospitalRepository**.getHospitalByHoscode(hoscode);  
 **return** hospital;  
 }

}

### 添加controller接口

在ApiController类添加接口

**package** com.atguigu.yygh.hosp.controller.api;  
  
@RestController  
@RequestMapping(**"/api/hosp"**)  
**public class** ApiController {  
  
 @Autowired  
 **private** HospitalService **hospitalService**;  
  
 @Autowired  
 **private** HospitalSetService **hospitalSetService**;  
  
 *//查询医院* @PostMapping(**"hospital/show"**)  
 **public** Result getHospital(HttpServletRequest request) {

*//获取传递过来医院信息* Map<String, String[]> requestMap = request.getParameterMap();  
 Map<String, Object> paramMap = HttpRequestHelper.*switchMap*(requestMap);

*//获取医院编号* String hoscode = (String)paramMap.get(**"hoscode"**);  
 *//1 获取医院系统传递过来的签名,签名进行MD5加密* String hospSign = (String)paramMap.get(**"sign"**);  
  
 *//2 根据传递过来医院编码，查询数据库，查询签名* String signKey = **hospitalSetService**.getSignKey(hoscode);  
  
 *//3 把数据库查询签名进行MD5加密* String signKeyMd5 = MD5.*encrypt*(signKey);  
  
 *//4 判断签名是否一致* **if**(!hospSign.equals(signKeyMd5)) {  
 **throw new** YyghException(ResultCodeEnum.***SIGN\_ERROR***);  
 }  
  
 *//调用service方法实现根据医院编号查询* Hospital hospital = **hospitalService**.getByHoscode(hoscode);  
 **return** Result.*ok*(hospital);  
 }  
}

## 上传科室接口

参考《尚医通API接口文档.docx》业务接口4.2上传科室

### 1、添加科室基础类

#### 1.1 添加model

说明：由于实体对象没有逻辑，我们已经统一导入

com.atguigu.yygh.model.hosp.Department

**package** com.atguigu.yygh.model.hosp;  
@Data  
@ApiModel(description = **"Department"**)  
@Document(**"Department"**)  
**public class** Department **extends** BaseMongoEntity {  
   
 **private static final long *serialVersionUID*** = 1L;  
  
 @ApiModelProperty(value = **"医院编号"**)  
 @Indexed *//普通索引* **private** String **hoscode**;  
  
 @ApiModelProperty(value = **"科室编号"**)  
 @Indexed(unique = **true**) *//唯一索引* **private** String **depcode**;  
  
 @ApiModelProperty(value = **"科室名称"**)  
 **private** String **depname**;  
  
 @ApiModelProperty(value = **"科室描述"**)  
 **private** String **intro**;  
  
 @ApiModelProperty(value = **"大科室编号"**)  
 **private** String **bigcode**;  
  
 @ApiModelProperty(value = **"大科室名称"**)  
 **private** String **bigname**;  
}

#### 1.2 添加repository

添加com.atguigu.yygh.hosp.repository.DepartmentRepository

**package** com.atguigu.yygh.hosp.repository;  
  
@Repository  
**public interface** DepartmentRepository **extends** MongoRepository<Department,String> {  
   
 *//上传科室接口* Department getDepartmentByHoscodeAndDepcode(String hoscode, String depcode);  
}

#### 1.3 添加service接口及实现类

1，添加com.atguigu.yygh.hosp.service.DepartmentService接口

|  |
| --- |
| **package** com.atguigu.yygh.hosp.service;  **public interface** DepartmentService { } |

2，添加com.atguigu.yygh.hosp.service.impl.DepartmentServiceImpl接口实现

|  |
| --- |
| **package** com.atguigu.yygh.hosp.service.impl;  @Service @Slf4j **public class** DepartmentServiceImpl **implements** DepartmentService {   @Autowired  **private** DepartmentRepository department**Repository**; } |

### 2、上传科室

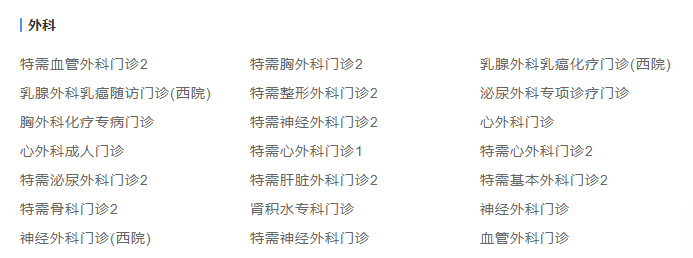
参考《尚医通API接口文档.doc》业务接口4.2上传科室

医院编号是平台分配的，全局唯一，科室编号为医院自己的编号，相对医院唯一，上传科室接口可以多次调用，如果医院编号与科室编号组合唯一为更新操作

#### 2.1 接口数据分析

|  |
| --- |
| { **"hoscode"**: **"1000\_0"**, **"depcode"**: **"200050923"**, **"depname"**: **"门诊部核酸检测门诊(东院)"**, **"intro"**: **"门诊部核酸检测门诊(东院)"**, **"bigcode"**: **"44f162029abb45f9ff0a5f743da0650d"**, **"bigname"**: **"体检科"**  } |

说明：一个大科室下可以有多个小科室，如图：



#### 2.2 添加service接口

2.2.1 在DepartmentService 类添加接口

|  |
| --- |
| */\*\*  \* 上传科室信息  \** ***@param paramMap*** *\*/* **void** save(Map<String, Object> paramMap); |

说明：参数使用Map，减少对象封装，有利于签名校验，后续会体验到

2.2.2 在DepartmentServiceImpl类添加实现

**package** com.atguigu.yygh.hosp.service.impl;  
  
@Service  
**public class** DepartmentServiceImpl **implements** DepartmentService {

@Autowired  
 **private** DepartmentRepository **departmentRepository**;  
  
 *//上传科室接口* @Override  
 **public void** save(Map<String, Object> paramMap) {  
   
 *//paramMap 转换department对象* String paramMapString = JSONObject.*toJSONString*(paramMap);  
 Department department = JSONObject.*parseObject*(paramMapString,Department.**class**);  
  
 *//根据医院编号 和 科室编号查询* Department departmentExist = **departmentRepository**.  
getDepartmentByHoscodeAndDepcode(department.getHoscode(),department.getDepcode());

*//判断* **if**(departmentExist != **null**) {  
 departmentExist.setUpdateTime(**new** Date());  
 departmentExist.setIsDeleted(0);  
 **departmentRepository**.save(departmentExist);  
 } **else** {  
 department.setCreateTime(**new** Date());  
 department.setUpdateTime(**new** Date());  
 department.setIsDeleted(0);  
 **departmentRepository**.save(department);  
 }  
 }  
}

#### 2.3 添加repository接口

在DepartmentRepository添加方法

**package** com.atguigu.yygh.hosp.repository;  
  
@Repository  
**public interface** DepartmentRepository **extends** MongoRepository<Department,String> {  
   
 *//上传科室接口* Department getDepartmentByHoscodeAndDepcode(String hoscode, String depcode);  
}

#### 2.4 添加controller接口

在ApiController类添加接口

**package** com.atguigu.yygh.hosp.controller.api;  
  
@RestController  
@RequestMapping(**"/api/hosp"**)  
**public class** ApiController {  
  
 @Autowired  
 **private** HospitalSetService **hospitalSetService**;  
  
 @Autowired  
 **private** DepartmentService **departmentService**;  
  
 *//上传科室接口* @PostMapping(**"saveDepartment"**)  
 **public** Result saveDepartment(HttpServletRequest request) {

*//获取传递过来科室信息* Map<String, String[]> requestMap = request.getParameterMap();  
 Map<String, Object> paramMap = HttpRequestHelper.*switchMap*(requestMap);  
  
 *//获取医院编号* String hoscode = (String)paramMap.get(**"hoscode"**);  
 *//1 获取医院系统传递过来的签名,签名进行MD5加密* String hospSign = (String)paramMap.get(**"sign"**);  
  
 *//2 根据传递过来医院编码，查询数据库，查询签名* String signKey = **hospitalSetService**.getSignKey(hoscode);  
  
 *//3 把数据库查询签名进行MD5加密* String signKeyMd5 = MD5.*encrypt*(signKey);  
  
 *//4 判断签名是否一致* **if**(!hospSign.equals(signKeyMd5)) {  
 **throw new** YyghException(ResultCodeEnum.***SIGN\_ERROR***);  
 }  
  
 *//调用service的方法* **departmentService**.save(paramMap);  
 **return** Result.*ok*();  
 }  
}

## 查询科室接口

参考《尚医通API接口文档.docx》业务接口4.5查询医院

一个医院有多个科室，因此我们采取分页查询方式

### 1、添加service接口

1.1 在DepartmentService 类添加接口

|  |
| --- |
| */\*\*  \* 分页查询  \** ***@param page*** *当前页码  \** ***@param limit*** *每页记录数  \** ***@param departmentQueryVo*** *查询条件  \** ***@return*** *\*/* Page<Department> selectPage(Integer page, Integer limit, DepartmentQueryVo departmentQueryVo); |

1.2 在DepartmentServiceImpl类添加实现

**package** com.atguigu.yygh.hosp.service.impl;  
  
@Service  
**public class** DepartmentServiceImpl **implements** DepartmentService {

@Autowired  
 **private** DepartmentRepository **departmentRepository**;  
  
 *//科室分页查询*  
 @Override  
 **public** Page<Department> findPageDepartment(**int** page, **int** limit, DepartmentQueryVo departmentQueryVo) {

Sort sort = Sort.*by*(Sort.Direction.***DESC***, **"createTime"**);

*//创建Pageable对象，设置当前页和每页记录数  
 //0是第一页* Pageable pageable = PageRequest.*of*(page-1, limit, sort);

*// 创建Example对象* Department department = **new** Department();  
 BeanUtils.*copyProperties*(departmentQueryVo,department);  
 department.setIsDeleted(0);

*//创建匹配器，即如何使用查询条件（MongoDB的方法）*  
 ExampleMatcher matcher = ExampleMatcher.*matching*() *//构建对象*  
 .withStringMatcher(ExampleMatcher.StringMatcher.***CONTAINING***) *//改变默认字符串匹配方式：模糊查询*  
 .withIgnoreCase(**true**); *//改变默认大小写忽略方式：忽略大小写*

*//创建实例*  
 Example<Department> example = Example.*of*(department,matcher);  
  
 Page<Department> all = **departmentRepository**.findAll(example, pageable);  
 **return** all;  
 }  
}

### 2、添加controller接口

在ApiController类添加接口

**package** com.atguigu.yygh.hosp.controller.api;  
  
@RestController  
@RequestMapping(**"/api/hosp"**)  
**public class** ApiController {  
  
 @Autowired  
 **private** DepartmentService **departmentService**;  
  
 *//查询科室接口* @PostMapping(**"department/list"**)  
 **public** Result findDepartment(HttpServletRequest request) {  
 *//获取传递过来科室信息* Map<String, String[]> requestMap = request.getParameterMap();  
 Map<String, Object> paramMap = HttpRequestHelper.*switchMap*(requestMap);  
  
 *//医院编号* String hoscode = (String)paramMap.get(**"hoscode"**);

*//当前页 和 每页记录数* **int** page = StringUtils.*isEmpty*(paramMap.get(**"page"**)) ? 1 : Integer.*parseInt*((String)paramMap.get(**"page"**));  
 **int** limit = StringUtils.*isEmpty*(paramMap.get(**"limit"**)) ? 1 : Integer.*parseInt*((String)paramMap.get(**"limit"**));  
 *//****TODO 签名校验*** DepartmentQueryVo departmentQueryVo = **new** DepartmentQueryVo();  
 departmentQueryVo.setHoscode(hoscode);  
 *//调用service方法* Page<Department> pageModel = **departmentService**.findPageDepartment(page,limit,departmentQueryVo);  
 **return** Result.*ok*(pageModel);  
 }  
}

**package** com.atguigu.yygh.vo.hosp;  
  
@Data  
@ApiModel(description = **"Department"**)  
**public class** DepartmentQueryVo {  
   
 @ApiModelProperty(value = **"医院编号"**)  
 **private** String **hoscode**;  
  
 @ApiModelProperty(value = **"科室编号"**)  
 **private** String **depcode**;  
  
 @ApiModelProperty(value = **"科室名称"**)  
 **private** String **depname**;  
  
 @ApiModelProperty(value = **"大科室编号"**)  
 **private** String **bigcode**;  
  
 @ApiModelProperty(value = **"大科室名称"**)  
 **private** String **bigname**;  
}

## 删除科室接口

参考《尚医通API接口文档.docx》业务接口4.7删除科室

根据医院编号与科室编号删除科室

### 1、添加service接口

1.1 在DepartmentService 类添加接口

|  |
| --- |
| */\*\*  \* 删除科室  \** ***@param hoscode*** *\** ***@param depcode*** *\*/* **void** remove(String hoscode, String depcode); |

1.2 在DepartmentServiceImpl类添加实现

**package** com.atguigu.yygh.hosp.service.impl;  
  
@Service  
**public class** DepartmentServiceImpl **implements** DepartmentService {

@Autowired  
 **private** DepartmentRepository **departmentRepository**;  
  
 *//删除科室接口* @Override  
 **public void** remove(String hoscode, String depcode) {

*//根据医院编号 和 科室编号查询* Department department = **departmentRepository**.getDepartmentByHoscodeAndDepcode(hoscode, depcode);

**if**(department != **null**) {  
 *//调用方法删除* **departmentRepository**.deleteById(department.getId());  
 }  
 }  
}

### 2、添加controller接口

在ApiController类添加接口

**package** com.atguigu.yygh.hosp.controller.api;  
  
@RestController  
@RequestMapping(**"/api/hosp"**)  
**public class** ApiController {  
  
 @Autowired  
 **private** DepartmentService **departmentService**;  
  
 *//删除科室接口* @PostMapping(**"department/remove"**)  
 **public** Result removeDepartment(HttpServletRequest request) {

*//获取传递过来科室信息* Map<String, String[]> requestMap = request.getParameterMap();  
 Map<String, Object> paramMap = HttpRequestHelper.*switchMap*(requestMap);  
 *//医院编号 和 科室编号* String hoscode = (String)paramMap.get(**"hoscode"**);  
 String depcode = (String)paramMap.get(**"depcode"**);

*//****TODO 签名校验* departmentService**.remove(hoscode,depcode);  
 **return** Result.*ok*();  
 }  
}

## 上传排班接口

参考《尚医通API接口文档.docx》业务接口4.3上传排班

### 1、添加排班基础类

#### 1.1 添加model

说明：由于实体对象没有逻辑，我们已经统一导入

com.atguigu.yygh.model.hosp.Schedule

**package** com.atguigu.yygh.model.hosp;  
@Data  
@ApiModel(description = **"Schedule"**)  
@Document(**"Schedule"**)  
**public class** Schedule **extends** BaseMongoEntity {  
   
 **private static final long *serialVersionUID*** = 1L;  
  
 @ApiModelProperty(value = **"医院编号"**)  
 @Indexed *//普通索引* **private** String **hoscode**;  
  
 @ApiModelProperty(value = **"科室编号"**)  
 @Indexed *//普通索引* **private** String **depcode**;  
  
 @ApiModelProperty(value = **"职称"**)  
 **private** String **title**;  
  
 @ApiModelProperty(value = **"医生名称"**)  
 **private** String **docname**;  
  
 @ApiModelProperty(value = **"擅长技能"**)  
 **private** String **skill**;  
  
 @ApiModelProperty(value = **"排班日期"**)  
 @JsonFormat(pattern = **"yyyy-MM-dd"**)  
 **private** Date **workDate**;  
  
 @ApiModelProperty(value = **"排班时间（0：上午 1：下午）"**)  
 **private** Integer **workTime**;  
  
 @ApiModelProperty(value = **"可预约数"**)  
 **private** Integer **reservedNumber**;  
  
 @ApiModelProperty(value = **"剩余预约数"**)  
 **private** Integer **availableNumber**;  
  
 @ApiModelProperty(value = **"挂号费"**)  
 **private** BigDecimal **amount**;  
  
 @ApiModelProperty(value = **"排班状态（-1：停诊 0：停约 1：可约）"**)  
 **private** Integer **status**;  
  
 @ApiModelProperty(value = **"排班编号（医院自己的排班主键）"**)  
 @Indexed *//普通索引* **private** String **hosScheduleId**;  
}

#### 1.2 添加repository

添加com.atguigu.yygh.hosp.repository.ScheduleRepository

**package** com.atguigu.yygh.hosp.repository;  
  
@Repository  
**public interface** ScheduleRepository **extends** MongoRepository<Schedule,String> {

*//根据医院编号 和 排班编号查询* Schedule getScheduleByHoscodeAndHosScheduleId(String hoscode, String hosScheduleId);  
  
 *//根据医院编号 、科室编号和工作日期，查询排班详细信息* List<Schedule> findScheduleByHoscodeAndDepcodeAndWorkDate(String hoscode, String depcode, Date toDate);  
}

#### 1.3 添加service接口及实现类

1，添加com.atguigu.yygh.hosp.service.ScheduleService接口

|  |
| --- |
| **package** com.atguigu.yygh.hosp.service;  **public interface** ScheduleService { } |

2，添加com.atguigu.yygh.hosp.service.impl.ScheduleServiceImpl接口实现

|  |
| --- |
| **package** com.atguigu.yygh.hosp.service.impl;  @Service @Slf4j **public class** ScheduleServiceImpl **implements** ScheduleService {   @Autowired  **private** ScheduleRepository schedule**Repository**; } |

### 2、上传排班

参考《尚医通API接口文档.doc》业务接口4.3上传排班

医院编号是平台分配的，全局唯一，排班编号为医院自己的编号，相对医院唯一，上传排班接口可以多次调用，如果医院编号与排班编号组合唯一为更新操作

#### 2.1 接口数据分析

|  |
| --- |
| { **"hoscode"**: **"1000\_0"**, **"depcode"**: **"200040878"**, **"title"**: **"医师"**, **"docname"**: **""**, **"skill"**: **"内分泌科常见病。"**, **"workDate"**: **"2020-06-22"**, **"workTime"**: 0, **"reservedNumber"**: 33, **"availableNumber"**: 22, **"amount"**: **"100"**, **"status"**: 1, **"hosScheduleId"**: **"1"**  } |

#### 2.2 添加service接口

2.2.1 在ScheduleService 类添加接口

|  |
| --- |
| */\*\*  \* 上传排班信息  \** ***@param paramMap*** *\*/* **void** save(Map<String, Object> paramMap); |

说明：参数使用Map，减少对象封装，有利于签名校验，后续会体验到

2.2.2 在ScheduleServiceImpl类添加实现

**package** com.atguigu.yygh.hosp.service.impl;  
  
@Service  
**public class** ScheduleServiceImpl **extends** ServiceImpl<ScheduleMapper, Schedule> **implements** ScheduleService {  
  
 @Autowired  
 **private** ScheduleRepository **scheduleRepository**;  
  
 @Autowired  
 **private** MongoTemplate **mongoTemplate**;  
  
 *//上传排班接口* @Override  
 **public void** save(Map<String, Object> paramMap) {

*//paramMap 转换department对象* String paramMapString = JSONObject.*toJSONString*(paramMap);  
 Schedule schedule = JSONObject.*parseObject*(paramMapString,Schedule.**class**);  
  
 *//根据医院编号 和 排班编号查询* Schedule scheduleExist = **scheduleRepository**.  
getScheduleByHoscodeAndHosScheduleId(schedule.getHoscode(),schedule.getHosScheduleId());  
 *//判断* **if**(scheduleExist!=**null**) {  
 scheduleExist.setUpdateTime(**new** Date());  
 scheduleExist.setIsDeleted(0);  
 scheduleExist.setStatus(1);  
 **scheduleRepository**.save(scheduleExist);  
 } **else** {  
 schedule.setCreateTime(**new** Date());  
 schedule.setUpdateTime(**new** Date());  
 schedule.setIsDeleted(0);  
 schedule.setStatus(1);  
 **scheduleRepository**.save(schedule);  
 }  
 }  
}

#### 2.3 添加repository接口

在ScheduleRepository添加方法

**package** com.atguigu.yygh.hosp.repository;  
  
@Repository  
**public interface** ScheduleRepository **extends** MongoRepository<Schedule,String> {

*//根据医院编号 和 排班编号查询* Schedule getScheduleByHoscodeAndHosScheduleId(String hoscode, String hosScheduleId);  
  
 *//根据医院编号 、科室编号和工作日期，查询排班详细信息* List<Schedule> findScheduleByHoscodeAndDepcodeAndWorkDate(String hoscode, String depcode, Date toDate);  
}

#### 2.4 添加controller接口

在ApiController类添加接口

**package** com.atguigu.yygh.hosp.controller.api;  
  
@RestController  
@RequestMapping(**"/api/hosp"**)  
**public class** ApiController {  
  
 @Autowired  
 **private** ScheduleService **scheduleService**;  
  
 *//上传排班接口* @PostMapping(**"saveSchedule"**)  
 **public** Result saveSchedule(HttpServletRequest request) {

*//获取传递过来科室信息* Map<String, String[]> requestMap = request.getParameterMap();  
 Map<String, Object> paramMap = HttpRequestHelper.*switchMap*(requestMap);  
  
 *//****TODO 签名校验* scheduleService**.save(paramMap);  
 **return** Result.*ok*();  
 }  
}

## 查询排班接口

参考《尚医通API接口文档.docx》业务接口4.6查询医院

一个科室有多个科室，因此我们采取分页查询方式

### 1、添加service接口

1，在ScheduleService 类添加接口

|  |
| --- |
| */\*\*  \* 分页查询  \** ***@param page*** *当前页码  \** ***@param limit*** *每页记录数  \** ***@param scheduleQueryVo*** *查询条件  \** ***@return*** *\*/* Page<Schedule> selectPage(Integer page, Integer limit, ScheduleQueryVo scheduleQueryVo); |

2，在ScheduleServiceImpl类添加实现

**package** com.atguigu.yygh.hosp.service.impl;  
  
@Service  
**public class** ScheduleServiceImpl **extends** ServiceImpl<ScheduleMapper, Schedule> **implements** ScheduleService {  
  
 @Autowired  
 **private** ScheduleRepository **scheduleRepository**;  
  
 @Autowired  
 **private** MongoTemplate **mongoTemplate**;  
  
 *//查询排班接口* @Override  
 **public** Page<Schedule> findPageSchedule(**int** page, **int** limit, ScheduleQueryVo scheduleQueryVo) {

Sort sort = Sort.*by*(Sort.Direction.***DESC***, **"createTime"**);  
 *// 创建Pageable对象，设置当前页和每页记录数  
 //0是第一页* Pageable pageable = PageRequest.*of*(page-1, limit, sort);  
 *// 创建Example对象* Schedule schedule = **new** Schedule();  
 BeanUtils.*copyProperties*(scheduleQueryVo,schedule);  
 schedule.setIsDeleted(0);  
 schedule.setStatus(1);

*//创建匹配器，即如何使用查询条件*  
 ExampleMatcher matcher = ExampleMatcher.*matching*() *//构建对象*

*//改变默认字符串匹配方式：模糊查询*  
 .withStringMatcher(ExampleMatcher.StringMatcher.***CONTAINING***)  
 .withIgnoreCase(**true**); *//改变默认大小写忽略方式：忽略大小写*

*//创建实例*  
 Example<Schedule> example = Example.*of*(schedule,matcher);  
  
 Page<Schedule> all = **scheduleRepository**.findAll(example, pageable);  
 **return** all;  
 }  
}

### 2、添加controller接口

在ApiController类添加接口

**package** com.atguigu.yygh.hosp.controller.api;  
  
@RestController  
@RequestMapping(**"/api/hosp"**)  
**public class** ApiController {  
  
 @Autowired  
 **private** ScheduleService **scheduleService**;  
  
 *//查询排班接口* @PostMapping(**"schedule/list"**)  
 **public** Result findSchedule(HttpServletRequest request) {  
 *//获取传递过来科室信息* Map<String, String[]> requestMap = request.getParameterMap();  
 Map<String, Object> paramMap = HttpRequestHelper.*switchMap*(requestMap);  
  
 *//医院编号* String hoscode = (String)paramMap.get(**"hoscode"**);  
  
 *//科室编号* String depcode = (String)paramMap.get(**"depcode"**);

*//当前页 和 每页记录数* **int** page = StringUtils.*isEmpty*(paramMap.get(**"page"**)) ? 1 : Integer.*parseInt*((String)paramMap.get(**"page"**));  
 **int** limit = StringUtils.*isEmpty*(paramMap.get(**"limit"**)) ? 1 : Integer.*parseInt*((String)paramMap.get(**"limit"**));  
 *//****TODO 签名校验*** ScheduleQueryVo scheduleQueryVo = **new** ScheduleQueryVo();  
 scheduleQueryVo.setHoscode(hoscode);  
 scheduleQueryVo.setDepcode(depcode);  
 *//调用service方法* Page<Schedule> pageModel = **scheduleService**.findPageSchedule(page,limit,scheduleQueryVo);  
 **return** Result.*ok*(pageModel);  
 }  
}

## 删除排班接口

参考《尚医通API接口文档.docx》业务接口4.8删除科室

根据医院编号与排班编号删除科室

### 1、添加service接口

1.1 在ScheduleService 类添加接口

|  |
| --- |
| */\*\*  \* 删除科室  \** ***@param hoscode*** *\** ***@param*** hosScheduleId*\*/* **void** remove(String hoscode, String hosScheduleId); |

1.2 在ScheduleServiceImpl类添加实现

**package** com.atguigu.yygh.hosp.service.impl;  
  
@Service  
**public class** ScheduleServiceImpl **extends** ServiceImpl<ScheduleMapper, Schedule> **implements** ScheduleService {  
  
 @Autowired  
 **private** ScheduleRepository **scheduleRepository**;  
  
 @Autowired  
 **private** MongoTemplate **mongoTemplate**;  
  
 *//删除排班* @Override  
 **public void** remove(String hoscode, String hosScheduleId) {  
 *//根据医院编号和排班编号查询信息* Schedule schedule = **scheduleRepository**.getScheduleByHoscodeAndHosScheduleId(hoscode, hosScheduleId);

**if**(schedule != **null**) {  
 **scheduleRepository**.deleteById(schedule.getId());  
 }  
 }  
}

### 2、添加controller接口

在ApiController类添加接口

**package** com.atguigu.yygh.hosp.controller.api;  
  
@RestController  
@RequestMapping(**"/api/hosp"**)  
**public class** ApiController {  
  
 @Autowired  
 **private** ScheduleService **scheduleService**;  
  
 *//删除排班* @PostMapping(**"schedule/remove"**)  
 **public** Result remove(HttpServletRequest request) {

*//获取传递过来科室信息* Map<String, String[]> requestMap = request.getParameterMap();  
 Map<String, Object> paramMap = HttpRequestHelper.*switchMap*(requestMap);

*//获取医院编号和排班编号* String hoscode = (String)paramMap.get(**"hoscode"**);  
 String hosScheduleId = (String)paramMap.get(**"hosScheduleId"**);  
  
 *//****TODO 签名校验* scheduleService**.remove(hoscode,hosScheduleId);  
 **return** Result.*ok*();  
 }  
}