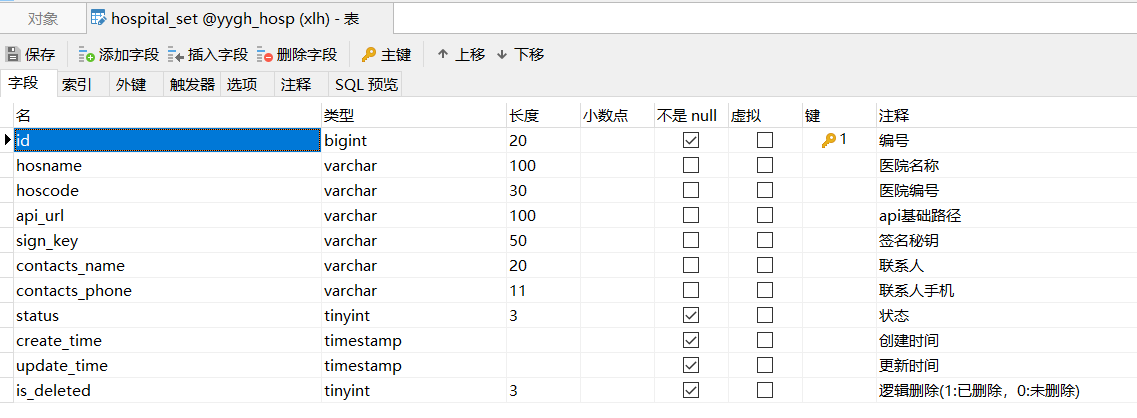
**医院设置**

## 一、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。

## 二、数据表结构



**说明：**

1. hosname：医院名称
2. hoscode：医院编号（平台分配，全局唯一，api接口必填信息）
3. 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）
4. sign\_key：双方api接口调用的签名key，有平台生成
5. contacts\_name：医院联系人姓名
6. contacts\_phone：医院联系人手机
7. status：状态（锁定/解锁）

## 三、医院模块开发

### 1、搭建医院模块service-hosp

在service父模块下构建service-hosp子模块，pom文件如下：

1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5. <parent>
6. <artifactId>service</artifactId>
7. <groupId>com.xlh</groupId>
8. <version>0.0.1-SNAPSHOT</version>
9. </parent>
10. <modelVersion>4.0.0</modelVersion>
12. <artifactId>service\_hosp</artifactId>
13. <dependencies>
14. <dependency>
15. <groupId>com.xlh</groupId>
16. <artifactId>model</artifactId>
17. <version>0.0.1-SNAPSHOT</version>
18. </dependency>
19. <dependency>
20. <groupId>org.springframework.boot</groupId>
21. <artifactId>spring-boot-starter-data-mongodb</artifactId>
22. </dependency>
23. </dependencies>

26. </project>

**配置application.properties文件：**

1. # 服务端口
2. server.port=8201
3. # 服务名
4. spring.application.name=service-hosp
6. # 环境设置：dev、test、prod
7. spring.profiles.active=dev
9. # mysql数据库连接
10. spring.datasource.driver-**class**-name=com.mysql.jdbc.Driver
11. spring.datasource.url=jdbc:mysql://localhost:3306/yygh\_hosp?characterEncoding=utf-8&useSSL=false
12. spring.datasource.username=root
13. spring.datasource.password=root
15. #返回json的全局时间格式
16. spring.jackson.date-format=yyyy-MM-dd HH:mm:ss
17. spring.jackson.time-zone=GMT+8
19. #配置mapper xml文件的路径
20. #mybatis-plus.mapper-locations=classpath:com/xlh/yygh/mapper/xml/\*.xml
21. #mybatis-plus.mapper-locations=classpath:com/xlh/yygh/mapper/xml/\*.xml
22. # nacos服务地址
23. #spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
25. #开启sentinel
26. #feign.sentinel.enabled=true
27. #设置sentinel地址
28. #spring.cloud.sentinel.transport.dashboard=http://127.0.0.1:8858
30. #mongodb地址
31. spring.data.mongodb.host=192.168.243.130
32. spring.data.mongodb.port=27017
33. spring.data.mongodb.database=yygh\_hosp
35. #rabbitmq地址
36. #spring.rabbitmq.host=127.0.0.1
37. #spring.rabbitmq.port=5672
38. #spring.rabbitmq.username=guest
39. #spring.rabbitmq.password=guest

**添加启动类：**

1. package com.xlh.yygh.hosp;
3. import org.springframework.boot.SpringApplication;
4. import org.springframework.boot.autoconfigure.SpringBootApplication;
5. import org.springframework.context.annotation.ComponentScan;
7. @SpringBootApplication
8. @ComponentScan(basePackages = "com.xlh")
9. **public** **class** ServiceHospApplication {
10. **public** **static** void main(String[] args) {
11. SpringApplication.run(ServiceHospApplication.**class**, args);
12. }
14. }

### 2、添加医院设置的CURD

步骤：

1. 添加model并引入

2. 使用mybatis plus并添加mapper

3. 添加service接口及其实现类

4. 添加controller

### 3、使用swagger2调试

### 4、发送签名ke功能

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api文档对接接口了。

## 四、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。

### 1、自定义异常类

在common-util模块已经添加YyghException类。

### 2、添加全局异常处理类

1. package com.xlh.yygh.common.exception;
3. import com.xlh.yygh.common.result.Result;
4. import org.springframework.web.bind.annotation.ControllerAdvice;
5. import org.springframework.web.bind.annotation.ExceptionHandler;
6. import org.springframework.web.bind.annotation.ResponseBody;
8. //用于异常全局处理
9. @ControllerAdvice
10. **public** **class** GlobalExceptionHandler {
12. @ExceptionHandler(Exception.**class**)
13. @ResponseBody
14. **public** Result error(Exception e){
15. e.printStackTrace();
16. **return** Result.fail();
17. }
19. @ExceptionHandler(YyghException.**class**)
20. @ResponseBody
21. **public** Result error(YyghException e){
22. e.printStackTrace();
23. **return** Result.fail();
24. }
26. }

## 六、日志

### 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：

**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

注：这种方式只能将日志打印在控制台上

### 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。

### 3、配置日志

resources/ logback-spring.xml文件：

1. <?xml version="1.0" encoding="UTF-8"?>
2. <configuration  scan="true" scanPeriod="10 seconds">
3. <!-- 日志级别从低到高分为TRACE < DEBUG < INFO < WARN < ERROR < FATAL，如果设置为WARN，则低于WARN的信息都不会输出 -->
4. <!-- scan:当此属性设置为true时，配置文件如果发生改变，将会被重新加载，默认值为true -->
5. <!-- scanPeriod:设置监测配置文件是否有修改的时间间隔，如果没有给出时间单位，默认单位是毫秒。当scan为true时，此属性生效。默认的时间间隔为1分钟。 -->
6. <!-- debug:当此属性设置为true时，将打印出logback内部日志信息，实时查看logback运行状态。默认值为false。 -->
8. <contextName>logback</contextName>
9. <!-- name的值是变量的名称，value的值时变量定义的值。通过定义的值会被插入到logger上下文中。定义变量后，可以使“${}”来使用变量。 -->
10. <property name="log.path" value="E:/yygh\_log/edu" />
12. <!-- 彩色日志 -->
13. <!-- 配置格式变量：CONSOLE\_LOG\_PATTERN 彩色日志格式 -->
14. <!-- magenta:洋红 -->
15. <!-- boldMagenta:粗红-->
16. <!-- cyan:青色 -->
17. <!-- white:白色 -->
18. <!-- magenta:洋红 -->
19. <property name="CONSOLE\_LOG\_PATTERN"
20. value="%yellow(%date{yyyy-MM-dd HH:mm:ss}) |%highlight(%-5level) |%blue(%thread) |%blue(%file:%line) |%green(%logger) |%cyan(%msg%n)"/>

23. <!--输出到控制台-->
24. <appender name="CONSOLE" **class**="ch.qos.logback.core.ConsoleAppender">
25. <!--此日志appender是为开发使用，只配置最底级别，控制台输出的日志级别是大于或等于此级别的日志信息-->
26. <!-- 例如：如果此处配置了INFO级别，则后面其他位置即使配置了DEBUG级别的日志，也不会被输出 -->
27. <filter **class**="ch.qos.logback.classic.filter.ThresholdFilter">
28. <level>INFO</level>
29. </filter>
30. <encoder>
31. <Pattern>${CONSOLE\_LOG\_PATTERN}</Pattern>
32. <!-- 设置字符集 -->
33. <charset>UTF-8</charset>
34. </encoder>
35. </appender>

38. <!--输出到文件-->
40. <!-- 时间滚动输出 level为 INFO 日志 -->
41. <appender name="INFO\_FILE" **class**="ch.qos.logback.core.rolling.RollingFileAppender">
42. <!-- 正在记录的日志文件的路径及文件名 -->
43. <file>${log.path}/log\_info.log</file>
44. <!--日志文件输出格式-->
45. <encoder>
46. <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>
47. <charset>UTF-8</charset>
48. </encoder>
49. <!-- 日志记录器的滚动策略，按日期，按大小记录 -->
50. <rollingPolicy **class**="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
51. <!-- 每天日志归档路径以及格式 -->
52. <fileNamePattern>${log.path}/info/log-info-%d{yyyy-MM-dd}.%i.log</fileNamePattern>
53. <timeBasedFileNamingAndTriggeringPolicy **class**="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
54. <maxFileSize>100MB</maxFileSize>
55. </timeBasedFileNamingAndTriggeringPolicy>
56. <!--日志文件保留天数-->
57. <maxHistory>15</maxHistory>
58. </rollingPolicy>
59. <!-- 此日志文件只记录info级别的 -->
60. <filter **class**="ch.qos.logback.classic.filter.LevelFilter">
61. <level>INFO</level>
62. <onMatch>ACCEPT</onMatch>
63. <onMismatch>DENY</onMismatch>
64. </filter>
65. </appender>
67. <!-- 时间滚动输出 level为 WARN 日志 -->
68. <appender name="WARN\_FILE" **class**="ch.qos.logback.core.rolling.RollingFileAppender">
69. <!-- 正在记录的日志文件的路径及文件名 -->
70. <file>${log.path}/log\_warn.log</file>
71. <!--日志文件输出格式-->
72. <encoder>
73. <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>
74. <charset>UTF-8</charset> <!-- 此处设置字符集 -->
75. </encoder>
76. <!-- 日志记录器的滚动策略，按日期，按大小记录 -->
77. <rollingPolicy **class**="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
78. <fileNamePattern>${log.path}/warn/log-warn-%d{yyyy-MM-dd}.%i.log</fileNamePattern>
79. <timeBasedFileNamingAndTriggeringPolicy **class**="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
80. <maxFileSize>100MB</maxFileSize>
81. </timeBasedFileNamingAndTriggeringPolicy>
82. <!--日志文件保留天数-->
83. <maxHistory>15</maxHistory>
84. </rollingPolicy>
85. <!-- 此日志文件只记录warn级别的 -->
86. <filter **class**="ch.qos.logback.classic.filter.LevelFilter">
87. <level>warn</level>
88. <onMatch>ACCEPT</onMatch>
89. <onMismatch>DENY</onMismatch>
90. </filter>
91. </appender>

94. <!-- 时间滚动输出 level为 ERROR 日志 -->
95. <appender name="ERROR\_FILE" **class**="ch.qos.logback.core.rolling.RollingFileAppender">
96. <!-- 正在记录的日志文件的路径及文件名 -->
97. <file>${log.path}/log\_error.log</file>
98. <!--日志文件输出格式-->
99. <encoder>
100. <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>
101. <charset>UTF-8</charset> <!-- 此处设置字符集 -->
102. </encoder>
103. <!-- 日志记录器的滚动策略，按日期，按大小记录 -->
104. <rollingPolicy **class**="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
105. <fileNamePattern>${log.path}/error/log-error-%d{yyyy-MM-dd}.%i.log</fileNamePattern>
106. <timeBasedFileNamingAndTriggeringPolicy **class**="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
107. <maxFileSize>100MB</maxFileSize>
108. </timeBasedFileNamingAndTriggeringPolicy>
109. <!--日志文件保留天数-->
110. <maxHistory>15</maxHistory>
111. </rollingPolicy>
112. <!-- 此日志文件只记录ERROR级别的 -->
113. <filter **class**="ch.qos.logback.classic.filter.LevelFilter">
114. <level>ERROR</level>
115. <onMatch>ACCEPT</onMatch>
116. <onMismatch>DENY</onMismatch>
117. </filter>
118. </appender>
120. <!--
121. <logger>用来设置某一个包或者具体的某一个类的日志打印级别、以及指定<appender>。
122. <logger>仅有一个name属性，
123. 一个可选的level和一个可选的addtivity属性。
124. name:用来指定受此logger约束的某一个包或者具体的某一个类。
125. level:用来设置打印级别，大小写无关：TRACE, DEBUG, INFO, WARN, ERROR, ALL 和 OFF，
126. 如果未设置此属性，那么当前logger将会继承上级的级别。
127. -->
128. <!--
129. 使用mybatis的时候，sql语句是debug下才会打印，而这里我们只配置了info，所以想要查看sql语句的话，有以下两种操作：
130. 第一种把<root level="INFO">改成<root level="DEBUG">这样就会打印sql，不过这样日志那边会出现很多其他消息
131. 第二种就是单独给mapper下目录配置DEBUG模式，代码如下，这样配置sql语句会打印，其他还是正常DEBUG级别：
132. -->
133. <!--开发环境:打印控制台-->
134. <springProfile name="dev">
135. <!--可以输出项目中的debug日志，包括mybatis的sql日志-->
136. <logger name="com.guli" level="INFO" />
138. <!--
139. root节点是必选节点，用来指定最基础的日志输出级别，只有一个level属性
140. level:用来设置打印级别，大小写无关：TRACE, DEBUG, INFO, WARN, ERROR, ALL 和 OFF，默认是DEBUG
141. 可以包含零个或多个appender元素。
142. -->
143. <root level="INFO">
144. <appender-ref ref="CONSOLE" />
145. <appender-ref ref="INFO\_FILE" />
146. <appender-ref ref="WARN\_FILE" />
147. <appender-ref ref="ERROR\_FILE" />
148. </root>
149. </springProfile>

152. <!--生产环境:输出到文件-->
153. <springProfile name="pro">
155. <root level="INFO">
156. <appender-ref ref="CONSOLE" />
157. <appender-ref ref="DEBUG\_FILE" />
158. <appender-ref ref="INFO\_FILE" />
159. <appender-ref ref="ERROR\_FILE" />
160. <appender-ref ref="WARN\_FILE" />
161. </root>
162. </springProfile>
164. </configuration>