# 1 项目介绍

## 1.1 项目背景

随着现代互联网的快速发展，人们的生活越来越离不开互联网带给我们的便捷。在这样的环境下，传统的医院医疗系统，现场挂号，就诊，取药等方式已经不能满足我们，所以线上预约挂号，看病的需求迫切要得到解决。在这样市场的要求下，我们的线上医疗系统应运而生。

## 1.2 业务模型

系统框架

前

台

系

统

患

者

模

块

医

生

模

块

后

台

系

统

# 2 项目架构

## 2.1 框架技术

后台框架：Spring Boot

持久层框架： BeetlSQL

引擎模板：Beetl

## 2.2 maven依赖

<**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-jdbc</**artifactId**>  
 </**dependency**>  
 *<!-- beetl-framework-starter -->* <**dependency**>  
 <**groupId**>com.ibeetl</**groupId**>  
 <**artifactId**>beetl-framework-starter</**artifactId**>  
 <**version**>1.2.24.RELEASE</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 <**version**>${mysql.driver.verion}</**version**>  
 <**scope**>runtime</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.projectlombok</**groupId**>  
 <**artifactId**>lombok</**artifactId**>  
 <**optional**>true</**optional**>  
 </**dependency**>  
 *<!--热部署-->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-devtools</**artifactId**>  
 <**scope**>runtime</**scope**>  
 <**optional**>true</**optional**>  
 </**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**>

## 2.4 yml文件配置

*###### WEB 路径 端口 ######***server**:  
 **servlet**:  
 **context-path**: /  
 **port**: 9000  
  
*################### beetsql配置 ########################  
# 默认是true，即向控制台输出执行时候的sql，参数，执行时间，以及执行的位置，每次修改sql文件的时候，自动检测sql文件修改***beetl-beetlsql**:  
 **dev**: true  
**beetlsql**:  
 **basePackage**: com.lb.dao  
 *# sql 文件的位置* **sqlPath**: /sql  
 **daoSuffix**: Dao  
 **dbStyle**: org.beetl.sql.core.db.MySqlStyle  
  
*################### mysql数据库连接配置 #################***spring**:  
 **datasource**:  
 **driver-class-name**: com.mysql.cj.jdbc.Driver  
 **url**: jdbc:mysql://localhost:3306/hospital?useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimezone=CTT  
 **username**: root  
 **password**: 123456

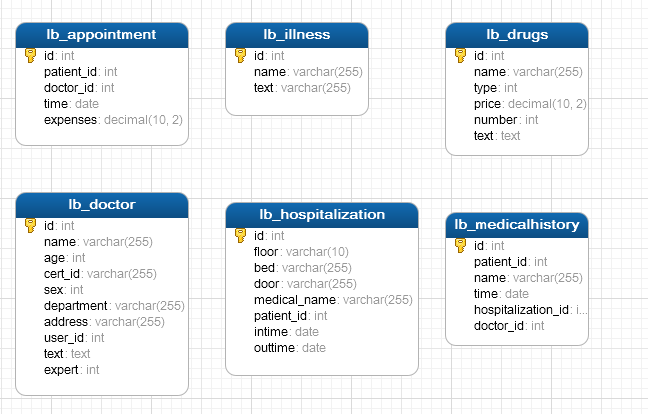
## 2.5 beetlsql 配置类

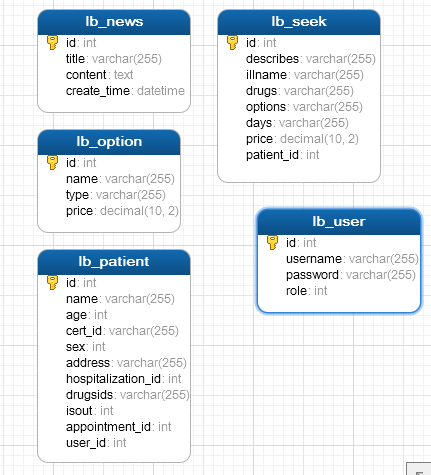
**public class** DataSourceConfig {  
 */\*\*  
 \* BeetlSQL 官方推荐配置  
 \* 使用的是 Hikari 连接池 springboot2.0 默认支持  
 \*/* @Bean(name=**"datasource"**)  
 **public** DataSource datasource(Environment env) {  
 HikariDataSource ds = **new** HikariDataSource();  
 ds.setJdbcUrl(env.getProperty(**"spring.datasource.url"**));  
 ds.setUsername(env.getProperty(**"spring.datasource.username"**));  
 ds.setPassword(env.getProperty(**"spring.datasource.password"**));  
 ds.setDriverClassName(env.getProperty(**"spring.datasource.driver-class-name"**));  
 **return** ds;  
 }  
}

## 2.6 数据库设计

数据库版本：MySQL 5.7.22

表设计：





具体的SQL脚本参考项目素材包。

## 2.7 逆向工程

使用SQLManager对象帮我们将数据表逆向生成pojo、sql文件以及接口。

在这一步操作时，已提前准备好3个模板文件。

测试工具类：

**public class** BeetlSqlTest {  
  
 *// ========数据库配置=========* **private static** String *driver* = **"com.mysql.cj.jdbc.Driver"**;  
 **private static** String *url* = **"jdbc:mysql://localhost:3306/hospital?useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimezone=CTT"**;  
 **private static** String *userName* = **"root"**;  
 **private static** String *password* = **"123456"**;  
 *// ========模板的路径, 示例是spring boot的[src/main/resources/beetlsqlTemplate 文件夹]=========* **private static** String *templatePath* = **"/beetlsqlTemplate"**;  
 *// ========md生成路径 要提前创建=========* **private static** String *mdPath* = **"/sql"**;  
 *// ========生成实体类所在的包=========* **private static** String *pojoPkg* = **"com.lb.entity"**;  
 *// ========生成mapper类所在的包=========* **private static** String *mapperPkg* = **"com.lb.dao"**;  
  
 */\*\*  
 \* 入口  
 \*/* **public static void** main(String[] args) **throws** Exception {  
 *genAll*();  
 }  
  
 **public static void** genAll() **throws** Exception {  
 *//准备工作* ConnectionSource source = ConnectionSourceHelper.*getSimple*(*driver*, *url*, *userName*, *password*);  
 DBStyle mysql = **new** MySqlStyle();  
 SQLLoader loader = **new** ClasspathLoader(*mdPath*);  
 UnderlinedNameConversion nc = **new** UnderlinedNameConversion();  
 SQLManager sqlManager = **new** SQLManager(mysql, loader, source, nc, **null**);  
  
 GenConfig config = **new** GenConfig();  
 config.setDisplay(**false**);  
 config.setPreferBigDecimal(**true**);  
  
 System.***out***.println(**"======生成代码======"**);  
 Set<String> tables = sqlManager.getMetaDataManager().allTable();  
 **for** (String table : tables) {  
 System.***out***.printf(**"%-20s %s\n"**,table , **"生成完毕"**);  
 *//默认生成实体类的实现* sqlManager.genPojoCode(table, *pojoPkg*, config);  
 *//自定义实现  
 genMd*(sqlManager, config, table);  
 *//自定义实现  
 genMapper*(sqlManager, config, table);  
 }  
 System.***out***.println(**"=====生成完毕====="**);  
 }  
  
 */\*\*  
 \* 生成md文件  
 \*/* **public static void** genMd(SQLManager sqlManager, GenConfig config, String table) **throws** IOException {  
 String fileName = StringKit.*toLowerCaseFirstOne*(sqlManager.getNc().getClassName(table));  
 **if** (config.getIgnorePrefix() != **null** && !config.getIgnorePrefix().trim().equals(**""**)) {  
 fileName = fileName.replaceFirst(StringKit.*toLowerCaseFirstOne*(config.getIgnorePrefix()), **""**);  
 fileName = StringKit.*toLowerCaseFirstOne*(fileName);  
 }  
 String target = GenKit.*getJavaResourcePath*() + **"/"** + *mdPath* + **"/"** + fileName + **".md"**;  
 TableDesc desc = sqlManager.getMetaDataManager().getTable(table);  
 FileWriter writer = **new** FileWriter(**new** File(target));  
 MDCodeGen mdCodeGen = **new** MDCodeGen();  
 mdCodeGen.setMapperTemplate(config.getTemplate(*templatePath* + **"/md.btl"**));  
 mdCodeGen.genCode(sqlManager.getBeetl(), desc, sqlManager.getNc(), **null**, writer);  
 writer.close();  
 }  
  
 */\*\*  
 \* 生成mapper  
 \*/* **public static void** genMapper(SQLManager sqlManager, GenConfig config, String table) {  
 MapperCodeGen mapperCodeGen = **new** MapperCodeGen(*mapperPkg*);  
 mapperCodeGen.setMapperTemplate(config.getTemplate(*templatePath* + **"/mapper.btl"**));  
 mapperCodeGen.genCode(*pojoPkg*, sqlManager.getNc().getClassName(table), sqlManager.getMetaDataManager().getTable(table), **null**, **false**);  
 }  
}

# 3 功能列表

## 3.1 用户登录

### 3.1.1 添加beetl配置

此处使用配置类的方式实现

**public class** BeetlConfig {  
 @Bean(name = **"beetlConfig"**)  
 **public** BeetlGroupUtilConfiguration getBeetlGroupUtilConfiguration() {  
 BeetlGroupUtilConfiguration beetlGroupUtilConfiguration = **new** BeetlGroupUtilConfiguration();  
 ClasspathResourceLoader classpathResourceLoader = **new** ClasspathResourceLoader();  
 beetlGroupUtilConfiguration.setResourceLoader(classpathResourceLoader);  
 beetlGroupUtilConfiguration.init();  
 **return** beetlGroupUtilConfiguration;  
 }  
 @Bean(name = **"beetlViewResolver"**)  
 **public** BeetlSpringViewResolver getBeetlSpringViewResolver(  
 @Qualifier(**"beetlConfig"**) BeetlGroupUtilConfiguration beetlGroupUtilConfiguration) {  
 BeetlSpringViewResolver beetlSpringViewResolver = **new** BeetlSpringViewResolver();  
 beetlSpringViewResolver.setPrefix(**"/templates/"**);  
 beetlSpringViewResolver.setSuffix(**".html"**);  
 beetlSpringViewResolver.setContentType(**"text/html;charset=UTF-8"**);  
 beetlSpringViewResolver.setOrder(0);  
 beetlSpringViewResolver.setConfig(beetlGroupUtilConfiguration);  
 **return** beetlSpringViewResolver;  
 }  
}

### 3.1.2 定义登录视图控制器

@Controller  
@RequestMapping(**"/home"**)  
**public class** LoginController {  
 */\*\*  
 \* 登录页面  
 \*/* @RequestMapping(**"/loginRegiterPage"**)  
 **public** String loginRegiterPage() {  
 **return "home/login&regist"**;  
 }  
}

### 3.1.3 开发登录控制器

@ResponseBody  
@RequestMapping(**"/login"**)  
**public** ResponseResult login(@RequestBody LbUser user, HttpSession session) {  
 ResponseResult result = **lbUserService**.checkUser(user);  
 **if** (result.getCode().equals(**"202"**)) {  
 session.setAttribute(**"user"**,user);  
 }  
 **return** result;  
}

### 3.1.3 开发service

**public** ResponseResult checkUser(LbUser user) {  
 *//从数据库中查询用户* ResponseResult result = **new** ResponseResult();  
 LbUser sysUser = **lbUserDao**.findUserByUsername(user.getUsername());  
 **if** (sysUser == **null**) {  
 result.setCode(**"201"**);*//用户不存在* result.setMessage(**"用户名或密码错误"**);  
 } **else** {  
 *//校验密码* **if** (sysUser.getPassword().equals(user.getPassword())) {  
 result.setCode(**"202"**);  
 result.setMessage(String.*valueOf*(sysUser.getRole()));*//绑定登录角色* } **else** {  
 result.setCode(**"203"**);*//密码错误* result.setMessage(**"用户名或密码错误"**);  
 }  
 }  
 **return** result;  
}