**第一步：因为在application.properties中自己配置了多个数据源，**

**所以就不能让springboot自动加载dataSource**

**所以要移除**DataSourceAutoConfiguration这个自动配置类

@SpringBootApplication(exclude = {DataSourceAutoConfiguration.**class**})

**public class** SbtemfApplication {

**application.properties**

**2.0之前要这样写**

**spring.datasource.bian.driver-class-name**=**com.mysql.jdbc.Driver  
spring.datasource.bian.url**=**jdbc:mysql://localhost:3306/bian  
spring.datasource.bian.username**=**root  
spring.datasource.bian.password**=**ls0611**

**spring.datasource.boke.driver-class-name**=**com.mysql.jdbc.Driver  
spring.datasource.boke.url**=**jdbc:mysql://localhost:3306/boke  
spring.datasource.boke.username**=**root  
spring.datasource.boke.password**=**ls0611**

**2.0之后要这样写，多了个hikari**

**spring.datasource.hikari.bian.driver-class-name**=**com.mysql.jdbc.Driver  
spring.datasource.hikari.bian.jdbc-url**=**jdbc:mysql://localhost:3306/bian  
spring.datasource.hikari.bian.username**=**root  
spring.datasource.hikari.bian.password**=**ls0611**

**spring.datasource.hikari.boke.driver-class-name**=**com.mysql.jdbc.Driver  
spring.datasource.hikari.boke.jdbc-url**=**jdbc:mysql://localhost:3306/bian  
spring.datasource.hikari.boke.username**=**root  
spring.datasource.hikari.boke.password**=**ls0611**

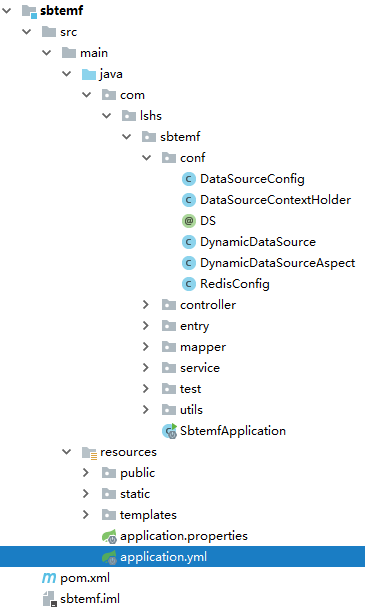
**application.yml**

**spring:  
 datasource:** *###################以下为druid增加的配置###########################* **type:** com.alibaba.druid.pool.DruidDataSource  
 *# 下面为连接池的补充设置，应用到上面所有数据源中  
 # 初始化大小，最小，最大* **initialSize:** 5  
 **minIdle:** 5  
 **maxActive:** 20  
 *# 配置获取连接等待超时的时间* **maxWait:** 60000  
 *# 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒* **timeBetweenEvictionRunsMillis:** 60000  
 *# 配置一个连接在池中最小生存的时间，单位是毫秒* **minEvictableIdleTimeMillis:** 300000  
 **validationQuery:** SELECT 1 FROM DUAL  
 **testWhileIdle:** true  
 **testOnBorrow:** false  
 **testOnReturn:** false  
 *# 打开PSCache，并且指定每个连接上PSCache的大小* **poolPreparedStatements:** true  
 **maxPoolPreparedStatementPerConnectionSize:** 20  
 *#配置监控统计拦截的filters，去掉后监控界面sql无法统计，'wall'用于防火墙，此处是filter修改的地方* **filters:  
 commons-log.connection-logger-name:** stat,wall,log4j  
 *# 通过connectProperties属性来打开mergeSql功能；慢SQL记录* **connectionProperties:** druid.stat.mergeSql=true;druid.stat.slowSqlMillis=5000  
 *# 合并多个DruidDataSource的监控数据* **useGlobalDataSourceStat:** true

**可以分成两个配置文件写，都会被识别到**

**第二步：在application启动类的这层新建一个package名字为conf，springboot自动扫描这个包下面的**

**@Configuration注解**



**这是整个项目的结构**

**DataSourceContextHolder.java**

**package** com.lshs.sbtemf.conf;  
  
**import** org.slf4j.Logger;  
**import** org.slf4j.LoggerFactory;  
  
*/\*\*  
 \** ***@Description: 多数据源动态切换配置*** *\** ***@author:*** *LuShao  
 \** ***@create:*** *2018-09-03 16:59  
 \*\*/***public class** DataSourceContextHolder {  
  
 **public static final** Logger ***log*** = LoggerFactory.*getLogger*(DataSourceContextHolder.**class**);  
  
 */\*\*  
 \* 默认数据源  
 \*/* **public static final** String ***DEFAULT\_DS*** = **"bian"**;  
 /\*\*

\* 当使用ThreadLocal维护变量时，ThreadLocal为每个使用该变量的线程提供独立的变量副本

\* 所以每一个线程都可以独立改变自己的副本，而不会影响其他变量所使用的副本

\*/  
 **private static final** ThreadLocal<String> ***contextHolder*** = **new** ThreadLocal<>();  
  
 *// 设置数据源名* **public static void** setDB(String dbType) {  
 ***log***.debug(**"切换到{}数据源"**, dbType);  
 ***contextHolder***.set(dbType);  
 }  
  
 *// 获取数据源名* **public static** String getDB() {  
 **return** (***contextHolder***.get());  
 }  
  
 *// 清除数据源名* **public static void** clearDB() {  
 ***contextHolder***.remove();  
 }  
}

**DynamicDataSource.java**

**package** com.lshs.sbtemf.conf;  
  
**import** org.slf4j.Logger;  
**import** org.slf4j.LoggerFactory;  
**import** org.springframework.jdbc.datasource.lookup.AbstractRoutingDataSource;  
  
*/\*\*  
 \** ***@Description:*** *\** ***@author:*** *LuShao  
 \** ***@create:*** *2018-09-03 16:59  
 \*\*/***public class** DynamicDataSource **extends** AbstractRoutingDataSource {  
 **private static final** Logger ***log*** = LoggerFactory.*getLogger*(DynamicDataSource.**class**);  
 /\*\*

\* extends AbstractRoutingDataSource 必须实现的一个方法个，取得一个字符串

\* 该字符串将于配置文件中的响应字符串进行匹配以定位数据源

\* 配置文件，即applicationContext.xml文件中需要以下代码

\* @see org.springframework.jdbc.datasource.lookup.AbstractRoutingDataSource#determineCurrentLookupKey()

\* @return

\*/  
 @Override  
 **protected** Object determineCurrentLookupKey() {  
 ***log***.debug(**"数据源为{}"**, DataSourceContextHolder.*getDB*());  
 /\*\*

\* DynamicDataSourceContextHolder代码中设置当前数据源，

\* 此时由这个方法获取，交给AbstractRoutingDataSource进行注入使用

\*/  
 **return** DataSourceContextHolder.*getDB*();  
 }  
  
}

**DataSourceConfig.java**

**package** com.lshs.sbtemf.conf;  
  
**import** org.springframework.boot.context.properties.ConfigurationProperties;  
**import** org.springframework.boot.jdbc.DataSourceBuilder;  
**import** org.springframework.context.annotation.Bean;  
**import** org.springframework.context.annotation.Configuration;  
**import** org.springframework.context.annotation.Primary;  
**import** org.springframework.jdbc.datasource.DataSourceTransactionManager;  
**import** org.springframework.transaction.PlatformTransactionManager;  
  
**import** javax.sql.DataSource;  
**import** java.util.HashMap;  
**import** java.util.Map;  
  
*/\*\*  
 \** ***@Description:*** *\** ***@author:*** *LuShao  
 \** ***@create:*** *2018-09-03 17:04  
 \*\*/*@Configuration  
**public class** DataSourceConfig {  
  
 @Bean(name = **"bianDS"**)  
 @ConfigurationProperties(prefix = **"spring.datasource.hikari.bian"**) *// application.properteis中对应属性的前缀* **public** DataSource dataSource1() {  
 **return** DataSourceBuilder.*create*().build();  
 }  
  
 @Bean(name = **"bokeDS"**)  
 @ConfigurationProperties(prefix = **"spring.datasource.hikari.boke"**) *// application.properteis中对应属性的前缀* **public** DataSource dataSource2() {  
 **return** DataSourceBuilder.*create*().build();  
 }  
  
 */\*\*  
 \* 动态数据源: 通过AOP在不同数据源之间动态切换  
 \** ***@return*** *\*/  
 //自动装配时当出现多个Bean候选者时，被注解为@Primary的Bean将作为首选者，否则将抛出异常，因为有多个dataSourcespringboot都会识别到因为都注解了bean，所以她不知道最后该用谁* @Primary  
 @Bean(name = **"dynamicDataSource"**)  
 **public** DataSource dynamicDataSource() {  
 DynamicDataSource dynamicDataSource = **new** DynamicDataSource();  
 *// 默认数据源* dynamicDataSource.setDefaultTargetDataSource(dataSource1());  
  
 *// 配置多数据源* Map<Object, Object> dsMap = **new** HashMap(5);  
 dsMap.put(**"bian"**, dataSource1());  
 dsMap.put(**"boke"**, dataSource2());  
  
 dynamicDataSource.setTargetDataSources(dsMap);  
  
 **return** dynamicDataSource;  
 }  
  
 */\*\*  
 \* 配置@Transactional注解事物  
 \** ***@return*** *\*/* @Bean  
 **public** PlatformTransactionManager transactionManager() {  
 **return new** DataSourceTransactionManager(dynamicDataSource());  
 }  
}

**DS.java**

**package** com.lshs.sbtemf.conf;  
  
**import** java.lang.annotation.ElementType;  
**import** java.lang.annotation.Retention;  
**import** java.lang.annotation.RetentionPolicy;  
**import** java.lang.annotation.Target;  
  
@Retention(RetentionPolicy.***RUNTIME***)  
@Target({ElementType.***METHOD***})作用在方法上  
**public** @**interface** DS {  
 String value() **default "bian"**;  
}

**DynamicDataSourceAspect.java**

**package** com.lshs.sbtemf.conf;  
  
**import** org.aspectj.lang.JoinPoint;  
**import** org.aspectj.lang.annotation.After;  
**import** org.aspectj.lang.annotation.Aspect;  
**import** org.aspectj.lang.annotation.Before;  
**import** org.aspectj.lang.reflect.MethodSignature;  
**import** org.springframework.stereotype.Component;  
  
**import** java.lang.reflect.Method;  
  
*/\*\*  
 \** ***@Description:*** *\** ***@author:*** *LuShao  
 \** ***@create:*** *2018-09-03 17:02  
 \*\*/*@Aspect  
@Component  
**public class** DynamicDataSourceAspect {  
  
 @Before(**"@annotation(com.lshs.sbtemf.conf.DS)"**)  
 **public void** beforeSwitchDS(JoinPoint point){  
  
 *//获得当前访问的class* Class<?> className = point.getTarget().getClass();  
  
 *//获得访问的方法名* String methodName = point.getSignature().getName();  
 *//得到方法的参数的类型* Class[] argClass = ((MethodSignature)point.getSignature()).getParameterTypes();  
 String dataSource = DataSourceContextHolder.***DEFAULT\_DS***;  
 **try** {  
 *// 得到访问的方法对象* Method method = className.getMethod(methodName, argClass);  
  
 *// 判断是否存在@DS注解* **if** (method.isAnnotationPresent(DS.**class**)) {  
 DS annotation = method.getAnnotation(DS.**class**);  
 *// 取出注解中的数据源名* dataSource = annotation.value();  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
  
 *// 切换数据源* DataSourceContextHolder.*setDB*(dataSource);  
  
 }  
  
  
 @After(**"@annotation(com.lshs.sbtemf.conf.DS)"**)  
 **public void** afterSwitchDS(JoinPoint point){  
  
 DataSourceContextHolder.*clearDB*();  
  
 }  
}

**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**>  
  
 <**groupId**>com.lshs</**groupId**>  
 <**artifactId**>sbtemf</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 <**packaging**>jar</**packaging**>  
  
 <**name**>sbtemf</**name**>  
 <**description**>Demo project for Spring Boot</**description**>  
  
 <**parent**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-parent</**artifactId**>  
 <**version**>2.0.4.RELEASE</**version**>  
 *<!--spring boot 2.x版本 ： 如果你用了 spring- druid java.lang.ClassNotFoundException org.springframework.boot.bind.RelaxedDataBinder 改成1.5.X-->* <**relativePath**/> *<!-- lookup parent from repository -->* </**parent**>  
  
 <**properties**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 <**project.reporting.outputEncoding**>UTF-8</**project.reporting.outputEncoding**>  
 <**java.version**>1.8</**java.version**>  
 *<!--使用高版本的模板-->* <**thymeleaf.version**>3.0.9.RELEASE</**thymeleaf.version**>  
 *<!--布局支持 2 和thymeleaf 3 版本结合使用-->* <**thymeleaf-layout-dialect.version**>2.1.1</**thymeleaf-layout-dialect.version**>  
 </**properties**>  
  
 <**dependencies**>  
  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 <**version**>5.1.45</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>com.github.pagehelper</**groupId**>  
 <**artifactId**>pagehelper-spring-boot-starter</**artifactId**>  
 <**version**>1.2.5</**version**>  
 <**exclusions**>  
 <**exclusion**>  
 <**artifactId**>mybatis-spring-boot-starter</**artifactId**>  
 <**groupId**>org.mybatis.spring.boot</**groupId**>  
 </**exclusion**>  
 </**exclusions**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.mybatis.spring.boot</**groupId**>  
 <**artifactId**>mybatis-spring-boot-starter</**artifactId**>  
 <**version**>1.3.1</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>druid</**artifactId**>  
 <**version**>1.1.0</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.aspectj</**groupId**>  
 <**artifactId**>aspectjweaver</**artifactId**>  
 <**version**>1.8.9</**version**> *<!--可以查查最新版-->* </**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**>\*\*/\*.xml</**include**>  
 </**includes**>  
 *<!--<filtering>false</filtering>-->* </**resource**></**resources**>  
 </**build**>  
  
</**project**>

**然后你就可以在service层的方法上加上注解，注意是方法**

**Service。Java**

**package** com.lshs.sbtemf.service;  
  
**import** com.github.pagehelper.PageHelper;  
**import** com.github.pagehelper.PageInfo;  
**import** com.lshs.sbtemf.conf.DS;  
**import** com.lshs.sbtemf.entry.BiAn;  
**import** com.lshs.sbtemf.entry.Contains;  
**import** com.lshs.sbtemf.entry.EUDataGridResult;  
**import** com.lshs.sbtemf.entry.QueryEntry;  
**import** com.lshs.sbtemf.mapper.BiAnMapper;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.stereotype.Service;  
  
**import** java.util.Date;  
**import** java.util.List;  
  
*/\*\*  
 \** ***@Description:*** *\** ***@author:*** *LuShao  
 \** ***@create:*** *2018-08-24 10:23  
 \*\*/*@Service  
**public class** BianServiceI **implements** BianService {  
  
 @Autowired  
 **private** BiAnMapper **biAnMapper**;  
 @Override  
 @DS(**"bian"**)  
 **public int** saveBian(BiAn biAn) {  
 String format = Contains.***SDF2***.format(**new** Date());  
 biAn.setAddtime(format);  
 **int** i = **biAnMapper**.insertSelective(biAn);  
 **return** i;  
 }  
  
 @Override  
 @DS(**"bian"**)  
 **public int** deleteBian() {  
 **int** i = **biAnMapper**.deleteBian();  
 **return** i;  
 }@Override  
 @DS(**"bian"**)  
 **public** BiAn firstData(String spx) {  
 BiAn = **biAnMapper**.firstData(spx).get(0);  
 **return** biAn;  
 }  
  
 */\*\*  
 \*  
 \** ***@param page*** *\** ***@param rows*** *\** ***@param queryEntry*** *\** ***@return*** *\*/* @Override  
 @DS(**"bian"**)  
 **public** EUDataGridResult findImgs(QueryEntry queryEntry){  
 PageHelper.*startPage*(queryEntry.getPage(), queryEntry.getRows());  
 List<BiAn> list = **biAnMapper**.findByCase(queryEntry);  
 PageInfo<BiAn> info=**new** PageInfo<>(list);  
 EUDataGridResult result=**new** EUDataGridResult();  
 result.setRows(list);  
 result.setTotal(info.getTotal());  
 **return** result;  
 }  
}