<http://blog.csdn.net/catoop/article/details/50575038>

版权声明：本文为博主原创文章，未经博主允许不得转载。

本文实现案例场景：   
某系统除了需要从自己的主要数据库上读取和管理数据外，还有一部分业务涉及到其他多个数据库，要求可以在任何方法上可以灵活指定具体要操作的数据库。

为了在开发中以最简单的方法使用，本文基于注解和AOP的方法实现，在spring boot框架的项目中，添加本文实现的代码类后，只需要配置好数据源就可以直接通过注解使用，简单方便。

**一配置二使用**   
**1. 启动类注册动态数据源**   
**2. 配置文件中配置多个数据源**   
**3. 在需要的方法上使用注解指定数据源**

1、在启动类添加 @Import({DynamicDataSourceRegister.class, MProxyTransactionManagementConfiguration.class})

@SpringBootApplication

@Import({DynamicDataSourceRegister.class}) // 注册动态多数据源

public class SpringBootSampleApplication {

// 省略其他代码

}

* 1
* 2
* 3
* 4
* 5
* 6

2、配置文件配置内容为：   
（不包括项目中的其他配置，这里只是数据源相关的）

# 主数据源，默认的

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

spring.datasource.url=jdbc:mysql://localhost:3306/test

spring.datasource.username=root

spring.datasource.password=123456

# 更多数据源

custom.datasource.names=ds1,ds2

custom.datasource.ds1.driver-class-name=com.mysql.jdbc.Driver

custom.datasource.ds1.url=jdbc:mysql://localhost:3306/test1

custom.datasource.ds1.username=root

custom.datasource.ds1.password=123456

custom.datasource.ds2.driver-class-name=com.mysql.jdbc.Driver

custom.datasource.ds2.url=jdbc:mysql://localhost:3306/test2

custom.datasource.ds2.username=root

custom.datasource.ds2.password=123456

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17

3、使用方法

package org.springboot.sample.service;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.List;

import org.springboot.sample.datasource.TargetDataSource;

import org.springboot.sample.entity.Student;

import org.springboot.sample.mapper.StudentMapper;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.core.RowMapper;

import org.springframework.stereotype.Service;

/\*\*

\* Student Service

\*

\* @author 单红宇(365384722)

\* @myblog http://blog.csdn.net/catoop/

\* @create 2016年1月12日

\*/

@Service

public class StudentService {

@Autowired

private JdbcTemplate jdbcTemplate;

// MyBatis的Mapper方法定义接口

@Autowired

private StudentMapper studentMapper;

@TargetDataSource(name="ds2")

public List<Student> likeName(String name){

return studentMapper.likeName(name);

}

public List<Student> likeNameByDefaultDataSource(String name){

return studentMapper.likeName(name);

}

/\*\*

\* 不指定数据源使用默认数据源

\*

\* @return

\* @author SHANHY

\* @create 2016年1月24日

\*/

public List<Student> getList(){

String sql = "SELECT ID,NAME,SCORE\_SUM,SCORE\_AVG, AGE FROM STUDENT";

return (List<Student>) jdbcTemplate.query(sql, new RowMapper<Student>(){

@Override

public Student mapRow(ResultSet rs, int rowNum) throws SQLException {

Student stu = new Student();

stu.setId(rs.getInt("ID"));

stu.setAge(rs.getInt("AGE"));

stu.setName(rs.getString("NAME"));

stu.setSumScore(rs.getString("SCORE\_SUM"));

stu.setAvgScore(rs.getString("SCORE\_AVG"));

return stu;

}

});

}

/\*\*

\* 指定数据源

\*

\* @return

\* @author SHANHY

\* @create 2016年1月24日

\*/

@TargetDataSource(name="ds1")

public List<Student> getListByDs1(){

String sql = "SELECT ID,NAME,SCORE\_SUM,SCORE\_AVG, AGE FROM STUDENT";

return (List<Student>) jdbcTemplate.query(sql, new RowMapper<Student>(){

@Override

public Student mapRow(ResultSet rs, int rowNum) throws SQLException {

Student stu = new Student();

stu.setId(rs.getInt("ID"));

stu.setAge(rs.getInt("AGE"));

stu.setName(rs.getString("NAME"));

stu.setSumScore(rs.getString("SCORE\_SUM"));

stu.setAvgScore(rs.getString("SCORE\_AVG"));

return stu;

}

});

}

/\*\*

\* 指定数据源

\*

\* @return

\* @author SHANHY

\* @create 2016年1月24日

\*/

@TargetDataSource(name="ds2")

public List<Student> getListByDs2(){

String sql = "SELECT ID,NAME,SCORE\_SUM,SCORE\_AVG, AGE FROM STUDENT";

return (List<Student>) jdbcTemplate.query(sql, new RowMapper<Student>(){

@Override

public Student mapRow(ResultSet rs, int rowNum) throws SQLException {

Student stu = new Student();

stu.setId(rs.getInt("ID"));

stu.setAge(rs.getInt("AGE"));

stu.setName(rs.getString("NAME"));

stu.setSumScore(rs.getString("SCORE\_SUM"));

stu.setAvgScore(rs.getString("SCORE\_AVG"));

return stu;

}

});

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111
* 112
* 113
* 114
* 115
* 116
* 117
* 118

要注意的是，在使用MyBatis时，注解@TargetDataSource 不能直接在接口类Mapper上使用。   
按上面的代码中StudentMapper为接口，代码如下：

package org.springboot.sample.mapper;

import java.util.List;

import org.springboot.sample.entity.Student;

/\*\*

\* StudentMapper，映射SQL语句的接口，无逻辑实现

\*

\* @author 单红宇(365384722)

\* @myblog http://blog.csdn.net/catoop/

\* @create 2016年1月20日

\*/

public interface StudentMapper {

// 注解 @TargetDataSource 不可以在这里使用

List<Student> likeName(String name);

Student getById(int id);

String getNameById(int id);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24

请将下面几个类放到Spring Boot项目中。   
DynamicDataSource.java   
DynamicDataSourceAspect.java   
DynamicDataSourceContextHolder.java   
DynamicDataSourceRegister.java   
TargetDataSource.java

package org.springboot.sample.datasource;

import org.springframework.jdbc.datasource.lookup.AbstractRoutingDataSource;

/\*\*

\* 动态数据源

\*

\* @author 单红宇(365384722)

\* @myblog http://blog.csdn.net/catoop/

\* @create 2016年1月22日

\*/

public class DynamicDataSource extends AbstractRoutingDataSource {

@Override

protected Object determineCurrentLookupKey() {

return DynamicDataSourceContextHolder.getDataSourceType();

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20

package org.springboot.sample.datasource;

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.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.stereotype.Component;

/\*\*

\* 切换数据源Advice

\*

\* @author 单红宇(365384722)

\* @myblog http://blog.csdn.net/catoop/

\* @create 2016年1月23日

\*/

@Aspect

@Order(-1)// 保证该AOP在@Transactional之前执行

@Component

public class DynamicDataSourceAspect {

private static final Logger logger = LoggerFactory.getLogger(DynamicDataSourceAspect.class);

@Before("@annotation(ds)")

public void changeDataSource(JoinPoint point, TargetDataSource ds) throws Throwable {

String dsId = ds.name();

if (!DynamicDataSourceContextHolder.containsDataSource(dsId)) {

logger.error("数据源[{}]不存在，使用默认数据源 > {}", ds.name(), point.getSignature());

} else {

logger.debug("Use DataSource : {} > {}", ds.name(), point.getSignature());

DynamicDataSourceContextHolder.setDataSourceType(ds.name());

}

}

@After("@annotation(ds)")

public void restoreDataSource(JoinPoint point, TargetDataSource ds) {

logger.debug("Revert DataSource : {} > {}", ds.name(), point.getSignature());

DynamicDataSourceContextHolder.clearDataSourceType();

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43

package org.springboot.sample.datasource;

import java.util.ArrayList;

import java.util.List;

public class DynamicDataSourceContextHolder {

private static final ThreadLocal<String> contextHolder = new ThreadLocal<String>();

public static List<String> dataSourceIds = new ArrayList<>();

public static void setDataSourceType(String dataSourceType) {

contextHolder.set(dataSourceType);

}

public static String getDataSourceType() {

return contextHolder.get();

}

public static void clearDataSourceType() {

contextHolder.remove();

}

/\*\*

\* 判断指定DataSrouce当前是否存在

\*

\* @param dataSourceId

\* @return

\* @author SHANHY

\* @create 2016年1月24日

\*/

public static boolean containsDataSource(String dataSourceId){

return dataSourceIds.contains(dataSourceId);

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35

package org.springboot.sample.datasource;

import java.util.HashMap;

import java.util.Map;

import javax.sql.DataSource;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.MutablePropertyValues;

import org.springframework.beans.PropertyValues;

import org.springframework.beans.factory.support.BeanDefinitionRegistry;

import org.springframework.beans.factory.support.GenericBeanDefinition;

import org.springframework.boot.autoconfigure.jdbc.DataSourceBuilder;

import org.springframework.boot.bind.RelaxedDataBinder;

import org.springframework.boot.bind.RelaxedPropertyResolver;

import org.springframework.context.EnvironmentAware;

import org.springframework.context.annotation.ImportBeanDefinitionRegistrar;

import org.springframework.core.convert.ConversionService;

import org.springframework.core.convert.support.DefaultConversionService;

import org.springframework.core.env.Environment;

import org.springframework.core.type.AnnotationMetadata;

/\*\*

\* 动态数据源注册<br/>

\* 启动动态数据源请在启动类中（如SpringBootSampleApplication）

\* 添加 @Import(DynamicDataSourceRegister.class)

\*

\* @author 单红宇(365384722)

\* @myblog http://blog.csdn.net/catoop/

\* @create 2016年1月24日

\*/

public class DynamicDataSourceRegister

implements ImportBeanDefinitionRegistrar, EnvironmentAware {

private static final Logger logger = LoggerFactory.getLogger(DynamicDataSourceRegister.class);

private ConversionService conversionService = new DefaultConversionService();

private PropertyValues dataSourcePropertyValues;

// 如配置文件中未指定数据源类型，使用该默认值

private static final Object DATASOURCE\_TYPE\_DEFAULT = "org.apache.tomcat.jdbc.pool.DataSource";

// private static final Object DATASOURCE\_TYPE\_DEFAULT =

// "com.zaxxer.hikari.HikariDataSource";

// 数据源

private DataSource defaultDataSource;

private Map<String, DataSource> customDataSources = new HashMap<>();

@Override

public void registerBeanDefinitions(AnnotationMetadata importingClassMetadata, BeanDefinitionRegistry registry) {

Map<Object, Object> targetDataSources = new HashMap<Object, Object>();

// 将主数据源添加到更多数据源中

targetDataSources.put("dataSource", defaultDataSource);

DynamicDataSourceContextHolder.dataSourceIds.add("dataSource");

// 添加更多数据源

targetDataSources.putAll(customDataSources);

for (String key : customDataSources.keySet()) {

DynamicDataSourceContextHolder.dataSourceIds.add(key);

}

// 创建DynamicDataSource

GenericBeanDefinition beanDefinition = new GenericBeanDefinition();

beanDefinition.setBeanClass(DynamicDataSource.class);

beanDefinition.setSynthetic(true);

MutablePropertyValues mpv = beanDefinition.getPropertyValues();

mpv.addPropertyValue("defaultTargetDataSource", defaultDataSource);

mpv.addPropertyValue("targetDataSources", targetDataSources);

registry.registerBeanDefinition("dataSource", beanDefinition);

logger.info("Dynamic DataSource Registry");

}

/\*\*

\* 创建DataSource

\*

\* @param type

\* @param driverClassName

\* @param url

\* @param username

\* @param password

\* @return

\* @author SHANHY

\* @create 2016年1月24日

\*/

@SuppressWarnings("unchecked")

public DataSource buildDataSource(Map<String, Object> dsMap) {

try {

Object type = dsMap.get("type");

if (type == null)

type = DATASOURCE\_TYPE\_DEFAULT;// 默认DataSource

Class<? extends DataSource> dataSourceType;

dataSourceType = (Class<? extends DataSource>) Class.forName((String) type);

String driverClassName = dsMap.get("driver-class-name").toString();

String url = dsMap.get("url").toString();

String username = dsMap.get("username").toString();

String password = dsMap.get("password").toString();

DataSourceBuilder factory = DataSourceBuilder.create().driverClassName(driverClassName).url(url)

.username(username).password(password).type(dataSourceType);

return factory.build();

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

return null;

}

/\*\*

\* 加载多数据源配置

\*/

@Override

public void setEnvironment(Environment env) {

initDefaultDataSource(env);

initCustomDataSources(env);

}

/\*\*

\* 初始化主数据源

\*

\* @author SHANHY

\* @create 2016年1月24日

\*/

private void initDefaultDataSource(Environment env) {

// 读取主数据源

RelaxedPropertyResolver propertyResolver = new RelaxedPropertyResolver(env, "spring.datasource.");

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

dsMap.put("type", propertyResolver.getProperty("type"));

dsMap.put("driver-class-name", propertyResolver.getProperty("driver-class-name"));

dsMap.put("url", propertyResolver.getProperty("url"));

dsMap.put("username", propertyResolver.getProperty("username"));

dsMap.put("password", propertyResolver.getProperty("password"));

defaultDataSource = buildDataSource(dsMap);

dataBinder(defaultDataSource, env);

}

/\*\*

\* 为DataSource绑定更多数据

\*

\* @param dataSource

\* @param env

\* @author SHANHY

\* @create 2016年1月25日

\*/

private void dataBinder(DataSource dataSource, Environment env){

RelaxedDataBinder dataBinder = new RelaxedDataBinder(dataSource);

//dataBinder.setValidator(new LocalValidatorFactory().run(this.applicationContext));

dataBinder.setConversionService(conversionService);

dataBinder.setIgnoreNestedProperties(false);//false

dataBinder.setIgnoreInvalidFields(false);//false

dataBinder.setIgnoreUnknownFields(true);//true

if(dataSourcePropertyValues == null){

Map<String, Object> rpr = new RelaxedPropertyResolver(env, "spring.datasource").getSubProperties(".");

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

// 排除已经设置的属性

values.remove("type");

values.remove("driver-class-name");

values.remove("url");

values.remove("username");

values.remove("password");

dataSourcePropertyValues = new MutablePropertyValues(values);

}

dataBinder.bind(dataSourcePropertyValues);

}

/\*\*

\* 初始化更多数据源

\*

\* @author SHANHY

\* @create 2016年1月24日

\*/

private void initCustomDataSources(Environment env) {

// 读取配置文件获取更多数据源，也可以通过defaultDataSource读取数据库获取更多数据源

RelaxedPropertyResolver propertyResolver = new RelaxedPropertyResolver(env, "custom.datasource.");

String dsPrefixs = propertyResolver.getProperty("names");

for (String dsPrefix : dsPrefixs.split(",")) {// 多个数据源

Map<String, Object> dsMap = propertyResolver.getSubProperties(dsPrefix + ".");

DataSource ds = buildDataSource(dsMap);

customDataSources.put(dsPrefix, ds);

dataBinder(ds, env);

}

}

}

package org.springboot.sample.datasource;

import java.lang.annotation.Documented;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

/\*\*

\* 在方法上使用，用于指定使用哪个数据源

\*

\* @author 单红宇(365384722)

\* @myblog http://blog.csdn.net/catoop/

\* @create 2016年1月23日

\*/

@Target({ ElementType.METHOD, ElementType.TYPE })

@Retention(RetentionPolicy.RUNTIME)

@Documented

public @interface TargetDataSource {

String name();

}

本文代码博主是经过测试后没有问题才发出来共享给大家的。对于连接池参数配置会应用到所有数据源上。   
比如配置一个：

spring.datasource.maximum-pool-size=80

那么我们所有的数据源都会自动应用上。

补充：   
如果你使用的是SpringMVC，并集成了Shiro，一般按网上的配置你可能是：

<bean class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator" depends-on="lifecycleBeanPostProcessor">

<property name="proxyTargetClass" value="true" />

</bean>

<bean class="org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor">

<property name="securityManager" ref="securityManager"/>

</bean>

那么你请不要这样做，请按下面方法配置：

<!-- AOP式方法级权限检查 -->

<!-- 不要使用 DefaultAdvisorAutoProxyCreator 会出现二次代理的问题，这里不详述。 mark by shanhy 2016-05-15 -->

<aop:config proxy-target-class="true"/>

<!-- 或者你使用了 <aop:aspectj-autoproxy proxy-target-class="true" /> 也可以。 -->

<bean class="org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor">

<property name="securityManager" ref="securityManager"/>

</bean>