自：https://blog.csdn.net/Winter\_chen001/article/details/80513993

**springboot mybatis 使用多数据源**

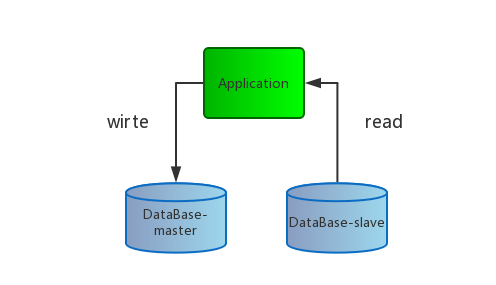
# springboot mybatis 使用多数据源

2018年05月30日 18:15:28 [LuisChen的博客](https://me.csdn.net/Winter_chen001) 阅读数：3634 标签： [springboot](http://so.csdn.net/so/search/s.do?q=springboot&t=blog)[mybatis](http://so.csdn.net/so/search/s.do?q=mybatis&t=blog)[多数据源](http://so.csdn.net/so/search/s.do?q=%E5%A4%9A%E6%95%B0%E6%8D%AE%E6%BA%90&t=blog) 更多

个人分类： [SpringBoot](https://blog.csdn.net/Winter_chen001/article/category/7005441)[mybatis](https://blog.csdn.net/Winter_chen001/article/category/6873749)

所属专栏： [springboot](https://blog.csdn.net/column/details/22005.html)

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/Winter\_chen001/article/details/80513993



[SpringBoot系列博客目录，含1.5.X版本和2.X版本](https://blog.csdn.net/Winter_chen001/article/details/80537829)

springboot2.0正式版发布之后，很多的组件集成需要变更了，这次将多数据源的使用踩的坑给大家填一填。当前多数据源的主要为主从库，读写分离，动态切换数据源。使用的技术就是AOP进行dao方法的切面，所以大家的方法名开头都需要按照规范进行编写，如：get\*\*\*、add\*\*\* 等等，

## 起步基础

本次的教程需要有springboot2.0集成mybatis 作为基础：

* 博客地址：[springboot2.0 Mybatis 整合 (springboot2.0版本)](https://blog.csdn.net/Winter_chen001/article/details/80010967)
* 基础项目源码：<https://github.com/WinterChenS/springboot2-mybatis-demo>

需要以上的步骤作为基础，运行成功之后可就可以开始配置多数据源了

## 开始动手

### 添加依赖

<dependency>

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

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

</dependency>

* 1
* 2
* 3
* 4

### 修改启动类

修改之前：

@SpringBootApplication

@MapperScan("com.winterchen.dao")

public class SpringBootMybatisMutilDatabaseApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBootMybatisMutilDatabaseApplication.class, args);

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8

修改之后：

@SpringBootApplication

public class SpringBootMybatisMutilDatabaseApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBootMybatisMutilDatabaseApplication.class, args);

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7

因为改用多数据源，所以dao接口的扫描我们放在配置类中进行

### 修改项目配置

首先我们需要在配置文件中配置多数据源，看一下原本项目的配置：

spring:

datasource:

name: mysql\_test

#-----------------start-----------------# （1）

type: com.alibaba.druid.pool.DruidDataSource

#-----------------end-----------------#

#druid相关配置

druid:

#监控统计拦截的filters

filters: stat

#-----------------start-----------------# （2）

driver-class-name: com.mysql.jdbc.Driver

#基本属性

url: jdbc:mysql://127.0.0.1:3306/mytest?useUnicode=true&characterEncoding=UTF-8&allowMultiQueries=true

username: root

password: root

#-----------------end-----------------#

#配置初始化大小/最小/最大

initial-size: 1

min-idle: 1

max-active: 20

#获取连接等待超时时间

max-wait: 60000

#间隔多久进行一次检测，检测需要关闭的空闲连接

time-between-eviction-runs-millis: 60000

#一个连接在池中最小生存的时间

min-evictable-idle-time-millis: 300000

validation-query: SELECT 'x'

test-while-idle: true

test-on-borrow: false

test-on-return: false

#打开PSCache，并指定每个连接上PSCache的大小。oracle设为true，mysql设为false。分库分表较多推荐设置为false

pool-prepared-statements: false

max-pool-prepared-statement-per-connection-size: 20

* 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

**需要修改的地方:**

* (1) 需要将 type: com.alibaba.druid.pool.DruidDataSource去除；
* (2) 将关于数据库的连接信息: driver-class-name、url、username 、password 去除；

**修改后：**

spring:

datasource:

name: mysql\_test

#-------------- start ----------------# (1)

master:

#基本属性--注意，这里的为【jdbcurl】-- 默认使用HikariPool作为数据库连接池

jdbcurl: jdbc:mysql://127.0.0.1:3306/mytest?useUnicode=true&characterEncoding=UTF-8&allowMultiQueries=true

username: root

password: root

driver-class-name: com.mysql.jdbc.Driver

slave:

#基本属性--注意，这里为 【url】-- 使用 druid 作为数据库连接池

url: jdbc:mysql://127.0.0.1:3306/mytest?useUnicode=true&characterEncoding=UTF-8&allowMultiQueries=true

username: root

password: root

driver-class-name: com.mysql.jdbc.Driver

read: get,select,count,list,query,find

write: add,create,update,delete,remove,insert

#-------------- end ----------------#

#druid相关配置

druid:

#监控统计拦截的filters

filters: stat,wall

#配置初始化大小/最小/最大

initial-size: 1

min-idle: 1

max-active: 20

#获取连接等待超时时间

max-wait: 60000

#间隔多久进行一次检测，检测需要关闭的空闲连接

time-between-eviction-runs-millis: 60000

#一个连接在池中最小生存的时间

min-evictable-idle-time-millis: 300000

validation-query: SELECT 'x'

test-while-idle: true

test-on-borrow: false

test-on-return: false

#打开PSCache，并指定每个连接上PSCache的大小。oracle设为true，mysql设为false。分库分表较多推荐设置为false

pool-prepared-statements: false

max-pool-prepared-statement-per-connection-size: 20

* 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

需要修改地方：

* (1) 在如上的配置中添加master、slave两个数据源;

**注意！！**两中数据源中有一处是不一样的，原因是因为master数据源使用 Hikari连接池，slave使用的是druid作为数据库连接池，所以两处的配置分别为：

master:

jdbcurl: jdbc:mysql://127.0.0.1:3306/mytest?useUnicode=true&characterEncoding=UTF-8&allowMultiQueries=true

* 1
* 2

slave:

url: jdbc:mysql://127.0.0.1:3306/mytest?useUnicode=true&characterEncoding=UTF-8&allowMultiQueries=true

* 1
* 2

数据库的连接不一样的，如果配置成一样的会在启动的时候报错。

**注意！！**

dao接口方法的方法名规则配置在这里了，当然可以自行更改：

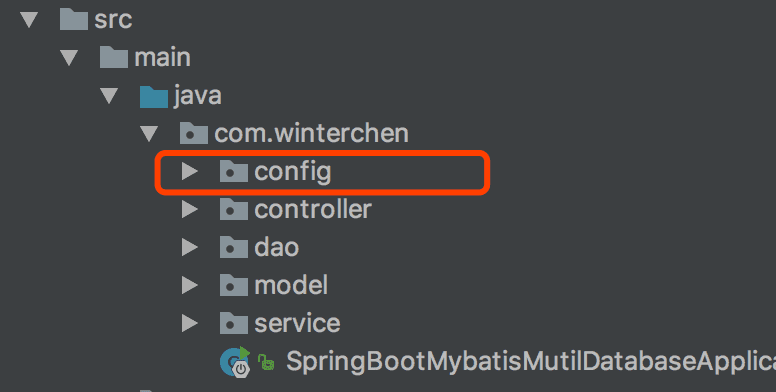
read: get,select,count,list,query,find

write: add,create,update,delete,remove,insert

* 1
* 2

### 创建配置包

首先在项目的/src/main/java/com/winterchen/包下创建config包



### 创建数据源类型的枚举DatabaseType

该枚举类主要用来区分读写

package com.winterchen.config;

/\*\*

\* 列出数据源类型

\* Created by Donghua.Chen on 2018/5/29.

\*/

public enum DatabaseType {

master("write"), slave("read");

DatabaseType(String name) {

this.name = name;

}

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return "DatabaseType{" +

"name='" + name + '\'' +

'}';

}

}

* 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

### 创建线程安全的DatabaseType容器

多数据源必须要保证数据源的线程安全的

package com.winterchen.config;

/\*\*

\* 保存一个线程安全的DatabaseType容器

\* Created by Donghua.Chen on 2018/5/29.

\*/

public class DatabaseContextHolder {

//用于存放多线程环境下的成员变量

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

public static void setDatabaseType(DatabaseType type) {

contextHolder.set(type);

}

public static DatabaseType getDatabaseType() {

return contextHolder.get();

}

}

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

### 创建动态数据源

实现数据源切换的功能就是自定义一个类扩展AbstractRoutingDataSource抽象类，其实该相当于数据源DataSource的路由中介，可以实现在项目运行时根据相应key值切换到对应的数据源DataSource上，有兴趣的同学可以看看它的源码。

public class DynamicDataSource extends AbstractRoutingDataSource {

static final Map<DatabaseType, List<String>> METHOD\_TYPE\_MAP = new HashMap<>();

@Nullable

@Override

protected Object determineCurrentLookupKey() {

DatabaseType type = DatabaseContextHolder.getDatabaseType();

logger.info("====================dataSource ==========" + type);

return type;

}

void setMethodType(DatabaseType type, String content) {

List<String> list = Arrays.asList(content.split(","));

METHOD\_TYPE\_MAP.put(type, list);

}

}

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

### 创建数据源配置类DataSourceConfig

@Configuration

@MapperScan("com.winterchen.dao")

@EnableTransactionManagement

public class DataSourceConfig {

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

@Autowired

private Environment env; // (1)

@Autowired

private DataSourceProperties properties; // (2)

@Value("${spring.datasource.druid.filters}") // (3)

private String filters;

@Value("${spring.datasource.druid.initial-size}")

private Integer initialSize;

@Value("${spring.datasource.druid.min-idle}")

private Integer minIdle;

@Value("${spring.datasource.druid.max-active}")

private Integer maxActive;

@Value("${spring.datasource.druid.max-wait}")

private Integer maxWait;

@Value("${spring.datasource.druid.time-between-eviction-runs-millis}")

private Long timeBetweenEvictionRunsMillis;

@Value("${spring.datasource.druid.min-evictable-idle-time-millis}")

private Long minEvictableIdleTimeMillis;

@Value("${spring.datasource.druid.validation-query}")

private String validationQuery;

@Value("${spring.datasource.druid.test-while-idle}")

private Boolean testWhileIdle;

@Value("${spring.datasource.druid.test-on-borrow}")

private boolean testOnBorrow;

@Value("${spring.datasource.druid.test-on-return}")

private boolean testOnReturn;

@Value("${spring.datasource.druid.pool-prepared-statements}")

private boolean poolPreparedStatements;

@Value("${spring.datasource.druid.max-pool-prepared-statement-per-connection-size}")

private Integer maxPoolPreparedStatementPerConnectionSize;

/\*\*

\* 通过Spring JDBC 快速创建 DataSource

\* @return

\*/

@Bean(name = "masterDataSource")

@Qualifier("masterDataSource")

@ConfigurationProperties(prefix = "spring.datasource.master") // (4)

public DataSource masterDataSource() {

return DataSourceBuilder.create().build();

}

/\*\*

\* 手动创建DruidDataSource,通过DataSourceProperties 读取配置

\* @return

\* @throws SQLException

\*/

@Bean(name = "slaveDataSource")

@Qualifier("slaveDataSource")

@ConfigurationProperties(prefix = "spring.datasource.slave")

public DataSource slaveDataSource() throws SQLException {

DruidDataSource dataSource = new DruidDataSource();

dataSource.setFilters(filters);

dataSource.setUrl(properties.getUrl());

dataSource.setDriverClassName(properties.getDriverClassName());

dataSource.setUsername(properties.getUsername());

dataSource.setPassword(properties.getPassword());

dataSource.setInitialSize(initialSize);

dataSource.setMinIdle(minIdle);

dataSource.setMaxActive(maxActive);

dataSource.setMaxWait(maxWait);

dataSource.setTimeBetweenEvictionRunsMillis(timeBetweenEvictionRunsMillis);

dataSource.setMinEvictableIdleTimeMillis(minEvictableIdleTimeMillis);

dataSource.setValidationQuery(validationQuery);

dataSource.setTestWhileIdle(testWhileIdle);

dataSource.setTestOnBorrow(testOnBorrow);

dataSource.setTestOnReturn(testOnReturn);

dataSource.setPoolPreparedStatements(poolPreparedStatements);

dataSource.setMaxPoolPreparedStatementPerConnectionSize(maxPoolPreparedStatementPerConnectionSize);

return dataSource;

}

/\*\*

\* 构造多数据源连接池

\* Master 数据源连接池采用 HikariDataSource

\* Slave 数据源连接池采用 DruidDataSource

\* @param master

\* @param slave

\* @return

\*/

@Bean

@Primary

public DynamicDataSource dataSource(@Qualifier("masterDataSource") DataSource master,

@Qualifier("slaveDataSource") DataSource slave) {

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

targetDataSources.put(DatabaseType.master, master);

targetDataSources.put(DatabaseType.slave, slave);

DynamicDataSource dataSource = new DynamicDataSource();

dataSource.setTargetDataSources(targetDataSources);// 该方法是AbstractRoutingDataSource的方法

dataSource.setDefaultTargetDataSource(slave);// 默认的datasource设置为myTestDbDataSource

String read = env.getProperty("spring.datasource.read");

dataSource.setMethodType(DatabaseType.slave, read);

String write = env.getProperty("spring.datasource.write");

dataSource.setMethodType(DatabaseType.master, write);

return dataSource;

}

@Bean

public SqlSessionFactory sqlSessionFactory(@Qualifier("masterDataSource") DataSource myTestDbDataSource,

@Qualifier("slaveDataSource") DataSource myTestDb2DataSource) throws Exception {

SqlSessionFactoryBean fb = new SqlSessionFactoryBean();

fb.setDataSource(this.dataSource(myTestDbDataSource, myTestDb2DataSource));

fb.setTypeAliasesPackage(env.getProperty("mybatis.type-aliases-package"));

fb.setMapperLocations(new PathMatchingResourcePatternResolver().getResources(env.getProperty("mybatis.mapper-locations")));

return fb.getObject();

}

@Bean

public DataSourceTransactionManager transactionManager(DynamicDataSource dataSource) throws Exception {

return new DataSourceTransactionManager(dataSource);

}

}

* 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
* 119
* 120
* 121
* 122
* 123
* 124
* 125
* 126
* 127
* 128
* 129
* 130
* 131
* 132
* 133
* 134
* 135
* 136
* 137
* 138
* 139
* 140

以上的代码中：

* (1) 注入类 Environment 可以很方便的获取配置文件中的参数
* (2) DataSourceProperties和（4）中的 @ConfigurationProperties(prefix = "spring.datasource.master")配合使用，将配置文件中的配置数据自动封装到实体类DataSourceProperties中
* (3) @Value注解同样是指定获取配置文件中的配置;

更详细的配置大家可以参考官方文档。

### 配置AOP

本章的开头已经说过，多数据源动态切换的原理是利用AOP切面进行动态的切换的，当调用dao接口方法时，根据接口方法的方法名开头进行区分读写。

/\*\*

\*

\* 动态处理数据源，根据命名区分

\* Created by Donghua.Chen on 2018/5/29.

\*/

@Aspect

@Component

@EnableAspectJAutoProxy(proxyTargetClass = true)

public class DataSourceAspect {

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

@Pointcut("execution(\* com.winterchen.dao.\*.\*(..))")//切点

public void aspect() {

}

@Before("aspect()")

public void before(JoinPoint point) { //在指定切点的方法之前执行

String className = point.getTarget().getClass().getName();

String method = point.getSignature().getName();

String args = StringUtils.join(point.getArgs(), ",");

logger.info("className:{}, method:{}, args:{} ", className, method, args);

try {

for (DatabaseType type : DatabaseType.values()) {

List<String> values = DynamicDataSource.METHOD\_TYPE\_MAP.get(type);

for (String key : values) {

if (method.startsWith(key)) {

logger.info(">>{} 方法使用的数据源为:{}<<", method, key);

DatabaseContextHolder.setDatabaseType(type);

DatabaseType types = DatabaseContextHolder.getDatabaseType();

logger.info(">>{}方法使用的数据源为:{}<<", method, types);

}

}

}

} catch (Exception e) {

logger.error(e.getMessage(), e);

}

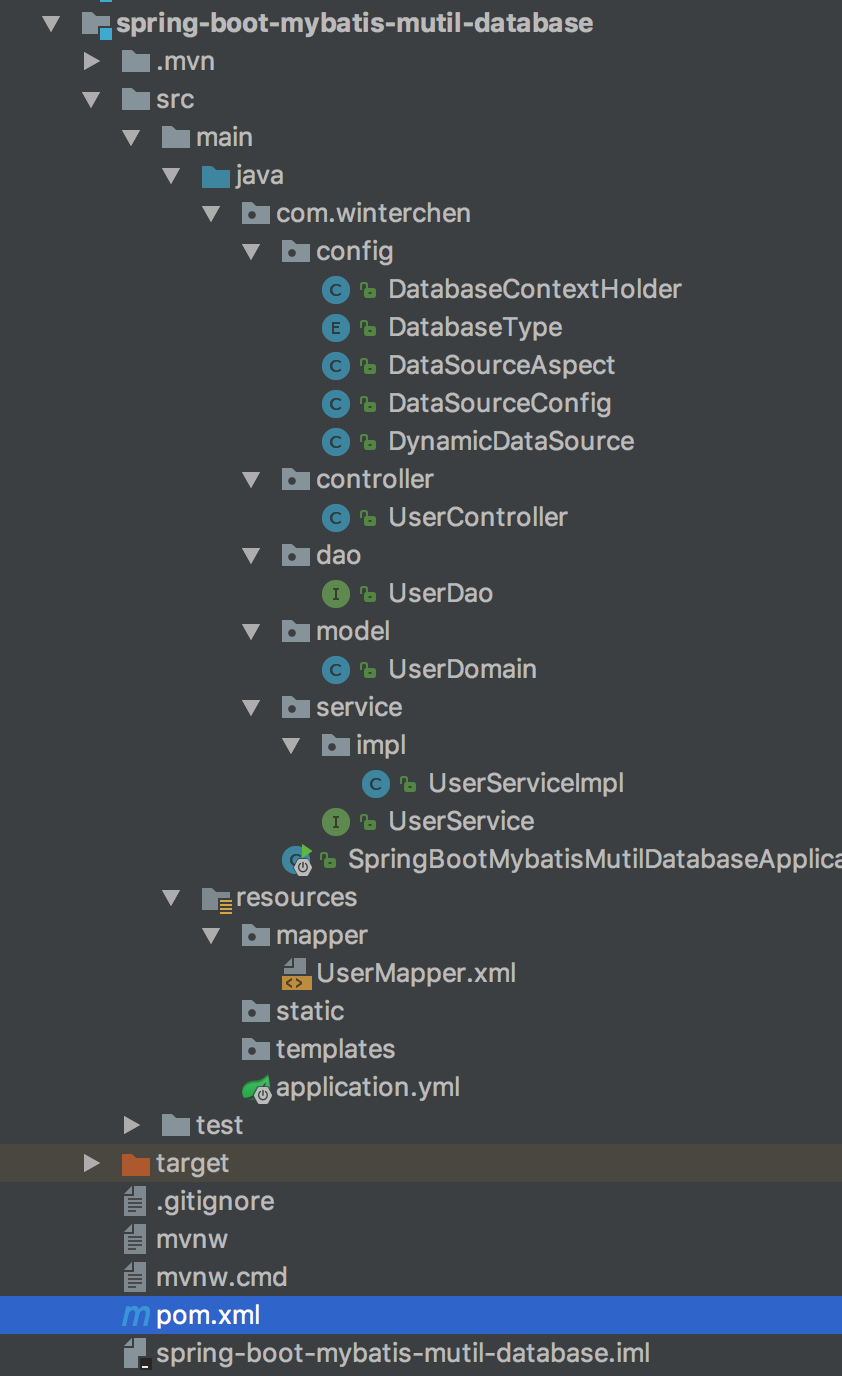
}

}

* 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

如上可以看到，切点切在dao的接口方法中，根据接口方法的方法名进行匹配数据源，然后将数据源set到用于存放数据源线程安全的容器中；

完整的项目结构了解一下：



## 项目启动

启动成功：

2018-05-30 17:27:16.492 INFO 35406 --- [ main] o.s.j.e.a.AnnotationMBeanExporter : Located MBean 'masterDataSource': registering with JMX server as MBean [com.zaxxer.hikari:name=masterDataSource,type=HikariDataSource]

2018-05-30 17:27:16.496 INFO 35406 --- [ main] o.s.j.e.a.AnnotationMBeanExporter : Located MBean 'slaveDataSource': registering with JMX server as MBean [com.alibaba.druid.pool:name=slaveDataSource,type=DruidDataSource]

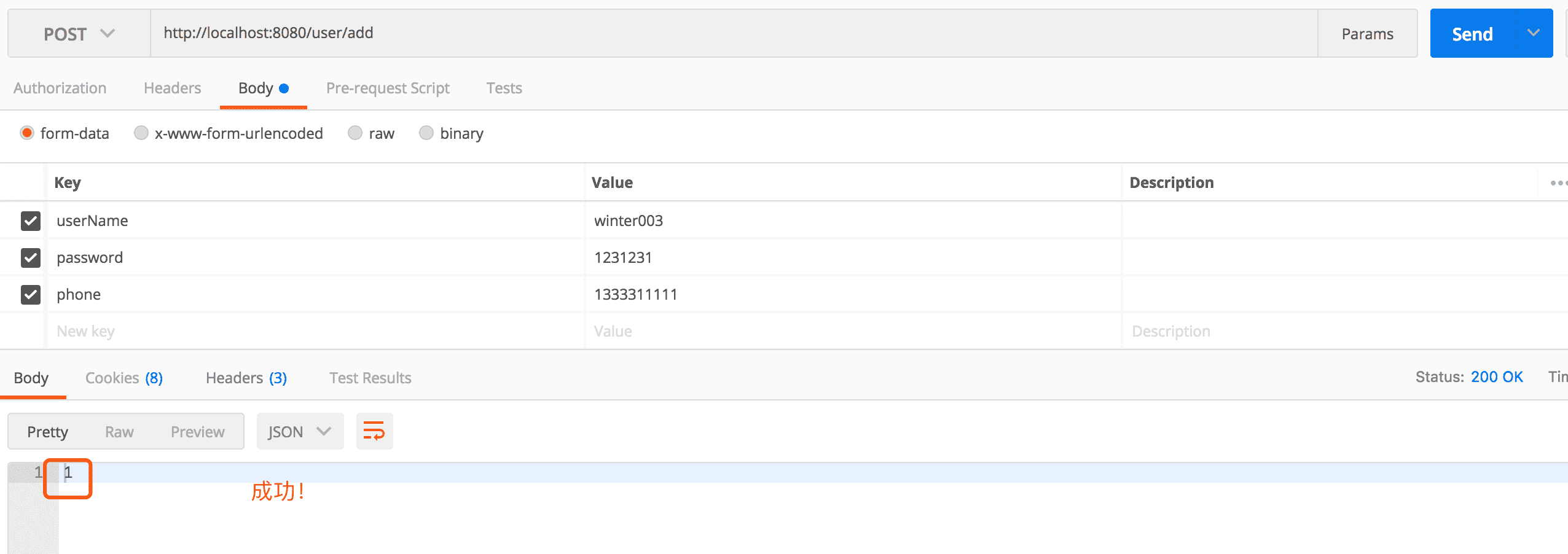
2018-05-30 17:27:16.498 INFO 35406 --- [ main] o.s.j.e.a.AnnotationMBeanExporter : Located MBean 'statFilter': registering with JMX server as MBean [com.alibaba.druid.filter.stat:name=statFilter,type=StatFilter]

2018-05-30 17:27:16.590 INFO 35406 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''

2018-05-30 17:27:16.598 INFO 35406 --- [ main] pringBootMybatisMutilDatabaseApplication : Started SpringBootMybatisMutilDatabaseApplication in 11.523 seconds (JVM running for 13.406)

* 1
* 2
* 3
* 4
* 5

添加用户（write）：



日志：

2018-05-30 17:29:07.347 INFO 35406 --- [nio-8080-exec-1] com.winterchen.config.DataSourceAspect : className:com.sun.proxy.$Proxy73, method:insert, args:com.winterchen.model.UserDomain@4b5b52dc

2018-05-30 17:29:07.350 INFO 35406 --- [nio-8080-exec-1] com.winterchen.config.DataSourceAspect : >>insert 方法使用的数据源为:insert<<

2018-05-30 17:29:07.351 INFO 35406 --- [nio-8080-exec-1] com.winterchen.config.DataSourceAspect : >>insert方法使用的数据源为:DatabaseType{name='write'}<<

2018-05-30 17:29:07.461 INFO 35406 --- [nio-8080-exec-1] com.winterchen.config.DynamicDataSource : ====================dataSource ==========DatabaseType{name='write'}

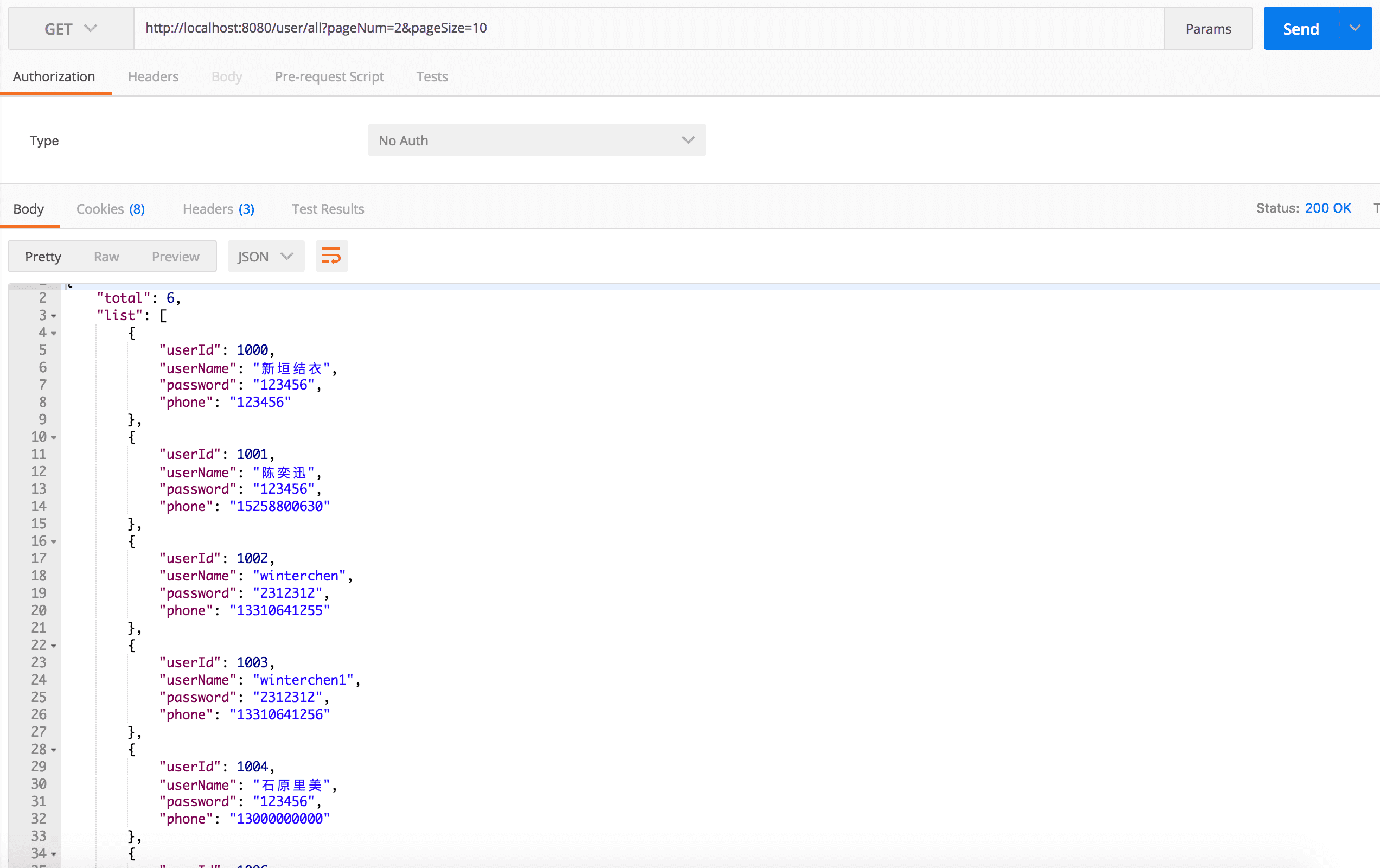
2018-05-30 17:29:07.462 INFO 35406 --- [nio-8080-exec-1] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...

2018-05-30 17:29:07.952 INFO 35406 --- [nio-8080-exec-1] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.

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

可以看出使用的就是write数据源，并且该数据源是使用HikariPool作为数据库连接池的

查询用户（read）：



日志：

2018-05-30 17:29:41.616 INFO 35406 --- [nio-8080-exec-2] com.winterchen.config.DataSourceAspect : className:com.sun.proxy.$Proxy73, method:selectUsers, args:

2018-05-30 17:29:41.618 INFO 35406 --- [nio-8080-exec-2] com.winterchen.config.DataSourceAspect : >>selectUsers 方法使用的数据源为:select<<

2018-05-30 17:29:41.618 INFO 35406 --- [nio-8080-exec-2] com.winterchen.config.DataSourceAspect : >>selectUsers方法使用的数据源为:DatabaseType{name='read'}<<

2018-05-30 17:29:41.693 INFO 35406 --- [nio-8080-exec-2] com.winterchen.config.DynamicDataSource : ====================dataSource ==========DatabaseType{name='read'}

2018-05-30 17:29:41.982 INFO 35406 --- [nio-8080-exec-2] com.alibaba.druid.pool.DruidDataSource : {dataSource-1} inited

* 1
* 2
* 3
* 4
* 5

可以看出使用的是read数据源。

源码地址：[戳这里](https://github.com/WinterChenS/springboot-learning-experience/tree/master/spring-boot-mybatis-mutil-database) https://github.com/WinterChenS/springboot-learning-experience/tree/master/spring-boot-mybatis-mutil-database

springboot技术交流群：681513531