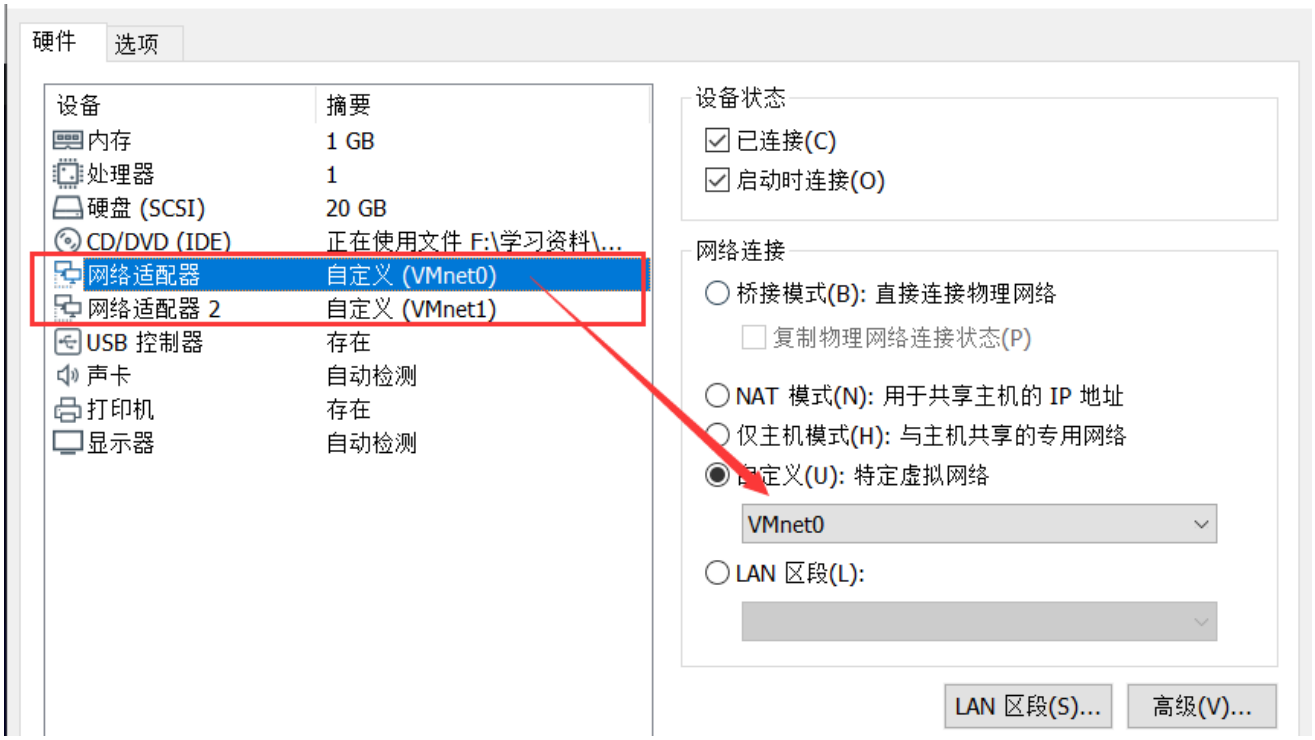
SpringBoot整合多数据源

一、MySQL主从复制搭建

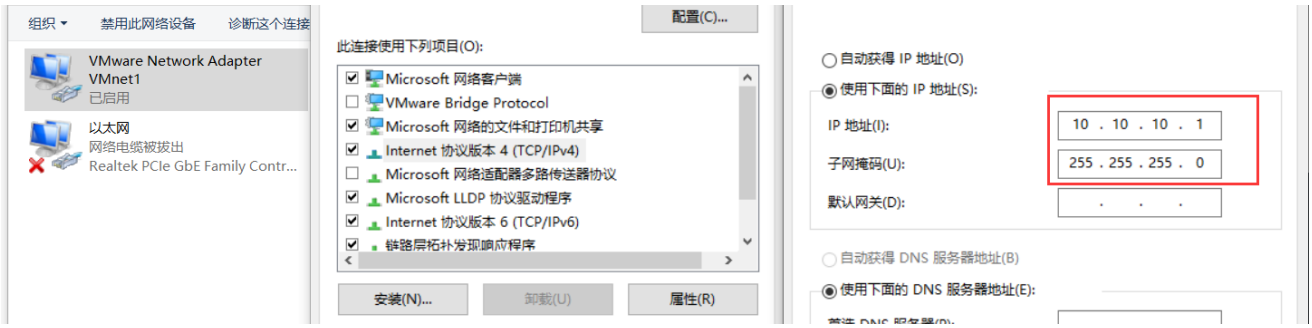
1.1 机器与环境准备

准备两台centos机器，配置两张网卡，分别配置网络，ip为10.10.10.10、10.10.10.20

如下，将网卡连接至VMnet0，虚拟机即可上网

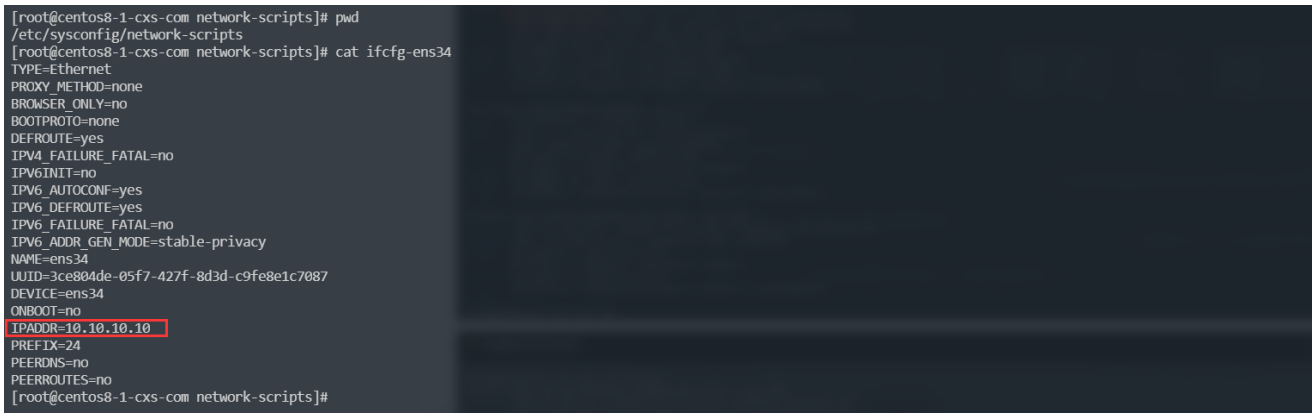


真实机vmnet1配置

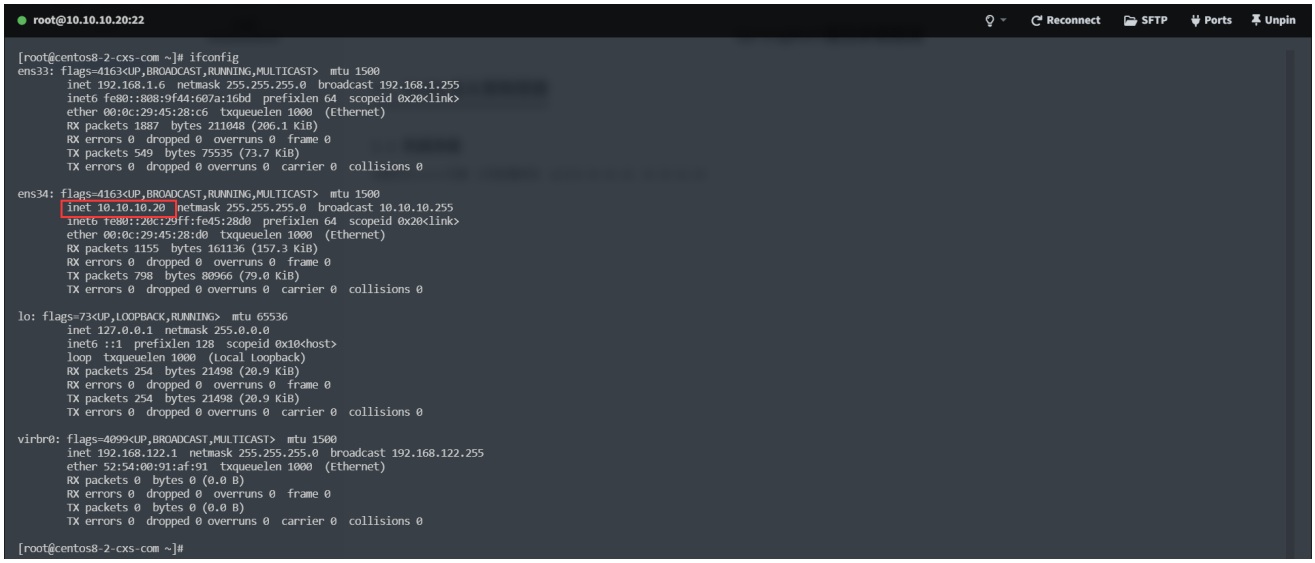


虚拟网络编辑器配置

虚拟机网卡配置（以10.10.10.10为例，20类似）



两台机器



设置Vmnet1网卡，让其真实机能与虚拟机通信

```
C:\Users\DELL>ping 10.10.10.10

正在 Ping 10.10.10.10 具有 32 字节的数据:
来自 10.10.10.10 的回复: 字节=32 时间<1ms TTL=64
来自 10.10.10.10 的回复: 字节=32 时间<1ms TTL=64
来自 10.10.10.10 的回复: 字节=32 时间<1ms TTL=64
来自 10.10.10.10 的回复: 字节=32 时间<1ms TTL=64

10.10.10.10 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
    最短 = 0ms, 最长 = 0ms, 平均 = 0ms

C:\Users\DELL>ping 10.10.10.20

正在 Ping 10.10.10.20 具有 32 字节的数据:
来自 10.10.10.20 的回复: 字节=32 时间<1ms TTL=64
来自 10.10.10.20 的回复: 字节=32 时间<1ms TTL=64
来自 10.10.10.20 的回复: 字节=32 时间<1ms TTL=64
来自 10.10.10.20 的回复: 字节=32 时间=1ms TTL=64

10.10.10.20 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
    最短 = 0ms, 最长 = 1ms, 平均 = 0ms
```

注：整合数据源在真实机操作，所以只需要真实机能与虚拟机ping通即可，如果让虚拟机与真实机ping通，需要开放入栈防火墙

1.2 安装MySQL

采用yum的方式安装（需要网）

1. 下载MySQL的yum源，安装yum源的rpm包

```
wget https://dev.mysql.com/get/mysql80-community-release-el7-3.noarch.rpm
# 下载路径：https://dev.mysql.com/get/mysql80-community-release-el7-3.noarch.rpm
# 安装yum源
rpm -Uvh mysql80-community-release-el7-3.noarch.rpm
```

2. yum源默认安装8.0的MySQL，修改为5.7的版本

```
# 编辑文件：
vim /etc/yum.repos.d/mysql-community.repo
# 将80的MySQL得enable设置为0，将57的MySQL得enable设置为1
# 搞定后通过如下命令，查看MySQL的默认安装版本
yum repolist enabled | grep mysql
```

3. 开始安装

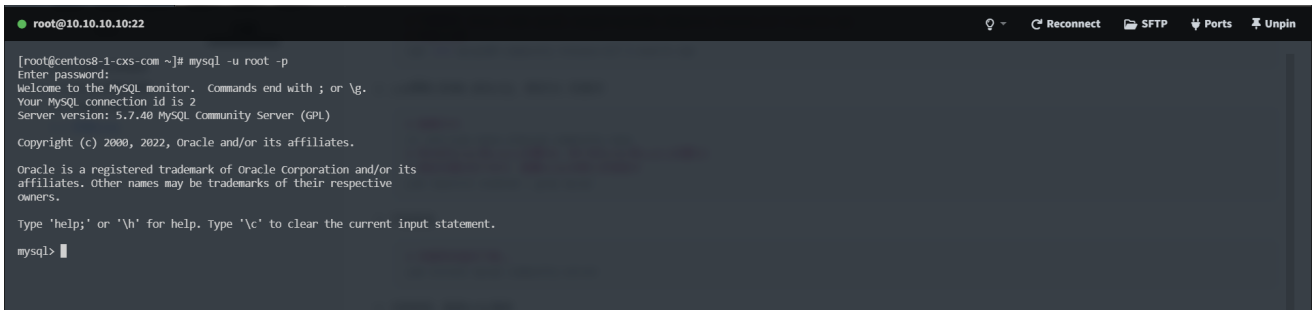
```
# 根据网络情况下载……
yum install mysql-community-server
```

4. 找到密码，登录MySQL服务

```
# 启动MySQL服务，找到密码
systemctl start mysqld
grep 'temporary password' /var/log/mysqld.log
# 正常登陆即可
mysql -u root -p
```

5. 重置密码，开启远程连接用户

```
# 登陆MySQL后，重置密码
ALTER USER 'root'@'localhost' IDENTIFIED BY 'P@ssw0rd';
# 开启远程连接用户，并刷新
alter user set host = '%' where user = 'root';
flush privileges;
```



1.3 搭建主从复制

1.3.1 master

修改mysql配置文件

```
vim /etc/my.cnf
```

```
#mysql 服务ID,保证整个集群环境中唯一
server-id=1

#mysql binlog 日志的存储路径和文件名
log-bin=/var/lib/mysql/mysqlbin

#是否只读,1 代表只读, 0 代表读写
read-only=0

#忽略的数据, 指不需要同步的数据库
binlog-ignore-db=mysql
```

重启服务

```
systemctl restart mysqld
```

连接MySQL，查看master状态

```
[root@centos8-1-cxs-com etc]# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.40-log MySQL Community Server (GPL)

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show master status;
+-----------------+----------+--------------+------------------+-------------------+
| File            | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+-----------------+----------+--------------+------------------+-------------------+
| mysqlbin.000001 |      154 |              | mysql            |                   |
+-----------------+----------+--------------+------------------+-------------------+
1 row in set (0.02 sec)
```

创建master用户

```
grant replication slave on *.* to 'master'@'10.10.10.20' identified by '@Aa1234567890';
flush privileges;
```

1.3.2 slave

修改mysql配置文件

```
#mysql服务端ID,唯一
server-id=2

#指定binlog日志
log-bin=/var/lib/mysql/mysqlbin
# 除超级用户root、及线程用户外，其余用户均只读
read-only
```

重启服务

```
systemctl restart mysqld
```

连接MySQL，指定当前从库对应的主库的IP地址，用户名，密码，从哪个日志文件开始的那个位置开始同步推送日志。

```
change master to master_host='10.10.10.10', master_user='master',
master_password='@Aa1234567890', master_log_file='mysqlbin.000001', master_log_pos=154;
```

开始同步

```
start slave;

show slave status\G;
```

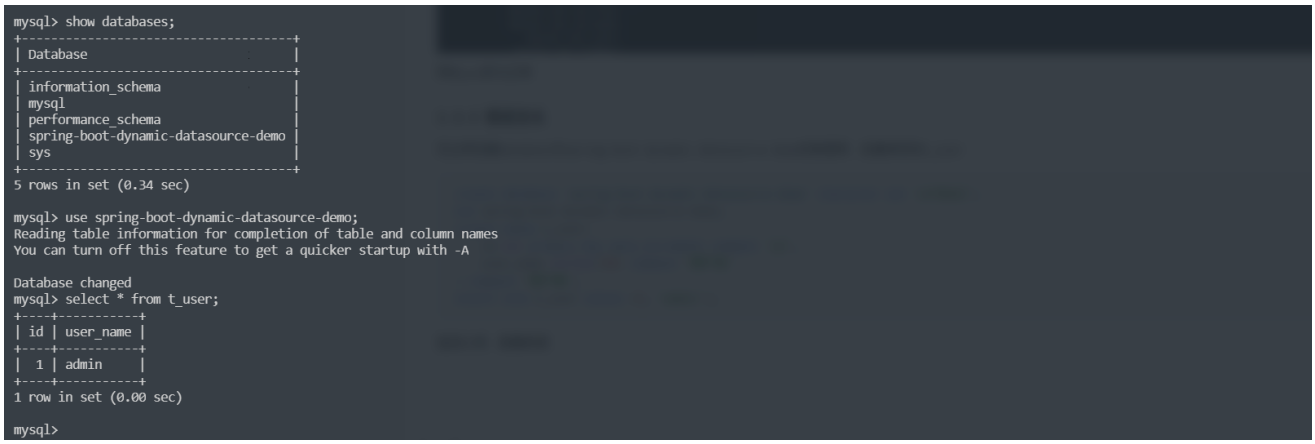
两处yes即为正常

1.3.3 验证主从

在主库创建database为spring-boot-dynamic-datasource-demo的数据库，创建表结构t_user

```
create database `spring-boot-dynamic-datasource-demo` character set 'utf8mb4';
use spring-boot-dynamic-datasource-demo;
create table t_user(
    id int primary key auto_increment comment 'id',
    user_name varchar(50) comment '用户名'
) comment '用户表';
insert into t_user values (1, 'admin');
```

登录从库，查看数据，数据已经同步至从库



二、Springboot整合多数据源

2.1 环境准备

登录主从，分别创建开发账号

```
grant all on `spring-boot-dynamic-datasource-demo`.* to 'dynamic-datasource-user'@'%'
identified by '@Aa1234567890';
flush privileges;
```

2.2 原理说明

Mybatis在进行数据库操作的时候，最终获取的连接是由DataSource接口的getConnection方法获取连接，整合多数据源的目的就是根据业务场景的不同，使用不同的数据源进行操作，举个例子，

比如MySQL的读写分离，在进行查询的时候，我们需要让MyBatis使用从库，进行增删改的时候，需要让MyBatis使用主库，那我们可不可以从这个方法下手呢？

这个方法也贴一下

```
public interface DataSource  extends CommonDataSource, Wrapper {

  /**
   * <p>Attempts to establish a connection with the data source that
   * this {@code DataSource} object represents.
   *
   * @return  a connection to the data source
   * @exception SQLException if a database access error occurs
   * @throws java.sql.SQLTimeoutException  when the driver has determined that the
   * timeout value specified by the {@code setLoginTimeout} method
   * has been exceeded and has at least tried to cancel the
   * current database connection attempt
   */
  Connection getConnection() throws SQLException;

  /**
   * <p>Attempts to establish a connection with the data source that
   * this {@code DataSource} object represents.
   *
```

```
     * @param username the database user on whose behalf the connection is
     *  being made
     * @param password the user's password
     * @return  a connection to the data source
     * @exception SQLException if a database access error occurs
     * @throws java.sql.SQLTimeoutException  when the driver has determined that the
     * timeout value specified by the {@code setLoginTimeout} method
     * has been exceeded and has at least tried to cancel the
     * current database connection attempt
     * @since 1.4
     */
    Connection getConnection(String username, String password)
      throws SQLException;
}
```

这个肯定是不行的，虽然可以达到读写分离的效果，（亲测可行），但是实现这个接口需要重写很多方法，我们只能处理getConnection这个方法，其他的方法我们无法处理，如果Spring调用其他方法的话，那后果就可想而知了，其实Spring已经考虑到开发者会整合多数据源，帮我们处理好了

AbstractRoutingDataSource这个类是Spring帮我们提供的多数据源整合的抽象类，下面我们简单说说这个类，我截取了主要的一部分代码，

主要的是3个成员变量，一个抽象方法

targetDataSources： 所有数据源，即我们的所有主从节点，需要我们手动设置

defaultTargetDataSource：默认数据源，当根据determineCurrentLookupKey()返回的key找不到就是默认的，看下图，很清楚了

```
107     protected DataSource determineTargetDataSource() {
108         Assert.notNull(this.resolvedDataSources, message: "DataSource router not initialized");
109         Object lookupKey = this.determineCurrentLookupKey();
110         DataSource dataSource = (DataSource)this.resolvedDataSources.get(lookupKey);
111         if (dataSource == null && (this.lenientFallback || lookupKey == null)) {
112             dataSource = this.resolvedDefaultDataSource;
113         }
114
115         if (dataSource == null) {
116             throw new IllegalStateException("Cannot determine target DataSource for lookup key [" + lookupKey + "]");
117         } else {
118             return dataSource;
119         }
120     }
```

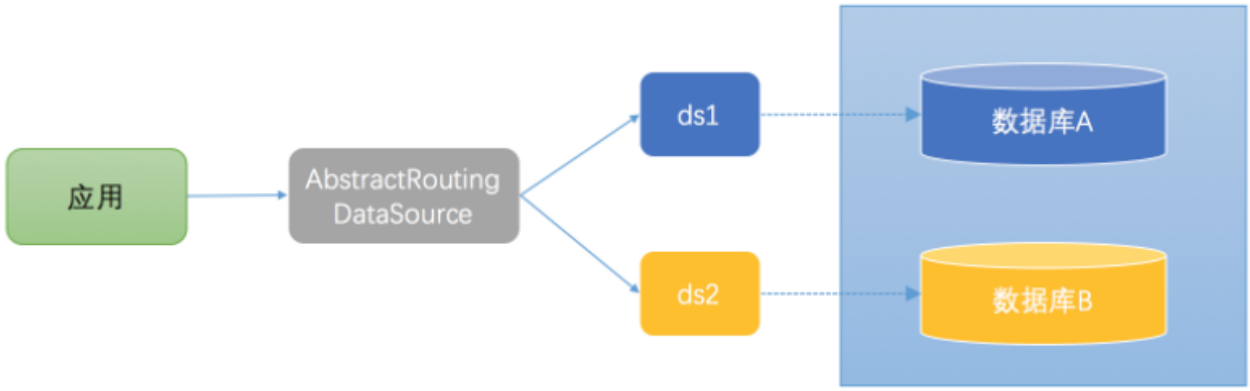
resolvedDataSources：这个变量不需要我们手动配置，在afterPropertiesSet()方法中将targetDataSources复制了一份给resolvedDataSources，至于为什么，不需要我们操心

determineCurrentLookupKey()： 这个方法需要我们重写，主要是给一个数据源的标识，比如W--写， R--读

```
public abstract class AbstractRoutingDataSource extends AbstractDataSource implements
InitializingBean {
    @Nullable
    private Map<Object, Object> targetDataSources;

    @Nullable
    private Object defaultTargetDataSource;

    @Nullable
```

```
    private Map<Object, DataSource> resolvedDataSources;

    @Nullable
    protected abstract Object determineCurrentLookupKey();

    public void afterPropertiesSet() {
        if (this.targetDataSources == null) {
            throw new IllegalArgumentException("Property 'targetDataSources' is required");
        } else {
            this.resolvedDataSources =
CollectionUtils.newHashMap(this.targetDataSources.size());
            this.targetDataSources.forEach((key, value) -> {
                Object lookupKey = this.resolveSpecifiedLookupKey(key);
                DataSource dataSource = this.resolveSpecifiedDataSource(value);
                this.resolvedDataSources.put(lookupKey, dataSource);
            });
            if (this.defaultTargetDataSource != null) {
                this.resolvedDefaultDataSource =
this.resolveSpecifiedDataSource(this.defaultTargetDataSource);
            }

        }
    }

}
```

原理图



2.3 SpringBoot整合多数据源-MyBatis插件方式

2.3.1 前置知识说明

这种方式需要对于Mybatis及插件有一定了解程度

上面说了AbstractRoutingDataSource类，中有个determineCurrentLookupKey()方法，需要我们返回一个数据源标识，那如何去实现呢，进行Select的时候设置为R，增删改的时候设置为W，这里使用Mybatis插件的方式实现

2.3.2 创建项目&导入依赖

```
<dependencies>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>2.1.4</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.47</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid-spring-boot-starter</artifactId>
            <version>1.2.4</version>
        </dependency>
        <dependency>
            <groupId>javax.persistence</groupId>
            <artifactId>persistence-api</artifactId>
            <version>1.0</version>
        </dependency>
    </dependencies>
```

2.3.3 application.yml

```
server:
  port: 2022
spring:
  datasource-w:
    type: com.alibaba.druid.pool.DruidDataSource
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql://10.10.10.10/spring-boot-dynamic-datasource-demo?
characterEncoding=utf8&useSSL=false
    username: dynamic-datasource-user
    password: "@Aa1234567890"
  datasource-r:
    type: com.alibaba.druid.pool.DruidDataSource
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql://10.10.10.20/spring-boot-dynamic-datasource-demo?
characterEncoding=utf8&useSSL=false
    username: dynamic-datasource-user
    password: "@Aa1234567890"
mybatis:
  configuration:
```

```
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
    map-underscore-to-camel-case: true
  type-aliases-package: com.cxs.model
  mapper-locations: classpath:mapper/*.xml
```

2.3.4 DynamicDataSource

AbstractRoutingDataSourcede实现类，主要做两件事，

1. 设置所有数据源及默认数据源
2. 重写determineCurrentLookupKey()，返回数据源标识

```
/*
 * @Project:spring-boot-dynamic-datasource-demo
 * @Author:cxs
 * @Motto:放下杂念,只为迎接明天更好的自己
 * */
@Primary
@Component
public class DynamicDataSource extends AbstractRoutingDataSource {

    @Autowired
    private DataSource master;

    @Autowired
    private DataSource slave;

    @Override
    protected Object determineCurrentLookupKey() {
        return DynamicDatasourceLocalUtil.getLocalCache();
    }

    @Override
    public void afterPropertiesSet() {
        super.setDefaultTargetDataSource(master);
        Map<Object, Object> map = new HashMap<>();
        map.put(TypeEnum.W.name(), master);
        map.put(TypeEnum.R.name(), slave);
        super.setTargetDataSources(map);
        super.afterPropertiesSet();
    }
}
```

2.3.5 DynamicDatasourceLocalUtil

需要记录全局的数据源标识，这里使用ThreadLocal

```
/*
 * @Project:spring-boot-dynamic-datasource-demo
 * @Author:cxs
 * @Motto:放下杂念,只为迎接明天更好的自己
 * */
public class DynamicDatasourceLocalUtil {
    private static final ThreadLocal<String> localCache = new ThreadLocal<>();
```

```
    public static void setLocalCache(TypeEnum typeEnum) {
        localCache.set(typeEnum.name());
    }

    public static String getLocalCache() {
        return localCache.get();
    }

    public static void removeLocalCache() {
        localCache.remove();
    }
}
```

2.3.6 DynamicDataSourcePlugin

Mybatis插件，帮助我们设置数据源标识，插件的使用方式就不多说了，这里将数据源标识封装成一个枚举中，规范代码，防止硬编码

```
/*
 * @Project:spring-boot-dynamic-datasource-demo
 * @Author:cxs
 * @Motto:放下杂念,只为迎接明天更好的自己
 * */
@Intercepts({
        @Signature(type = Executor.class, method = "query", args = {MappedStatement.class,
Object.class, RowBounds.class, ResultHandler.class}),
        @Signature(type = Executor.class, method = "query", args = {MappedStatement.class,
Object.class, RowBounds.class, ResultHandler.class, CacheKey.class, BoundSql.class}),
        @Signature(type = Executor.class, method = "update", args = {MappedStatement.class,
Object.class})
})
public class DynamicDataSourcePlugin implements Interceptor {
    @Override
    public Object intercept(Invocation invocation) throws Throwable {
        Object[] args = invocation.getArgs();
        MappedStatement statement = (MappedStatement) args[0];
        // SqlCommandType就是对数据库操作的类型
        SqlCommandType sqlCommandType = statement.getSqlCommandType();
        if (sqlCommandType.equals(SqlCommandType.SELECT)) {
            DynamicDatasourceLocalUtil.setLocalCache(TypeEnum.R);
        } else {
            DynamicDatasourceLocalUtil.setLocalCache(TypeEnum.W);
        }
        return invocation.proceed();
    }

    @Override
    public Object plugin(Object target) {
        if (target instanceof Executor) {
            return Plugin.wrap(target, this);
        } else {
            return target;
        }
    }
```

```
}
```

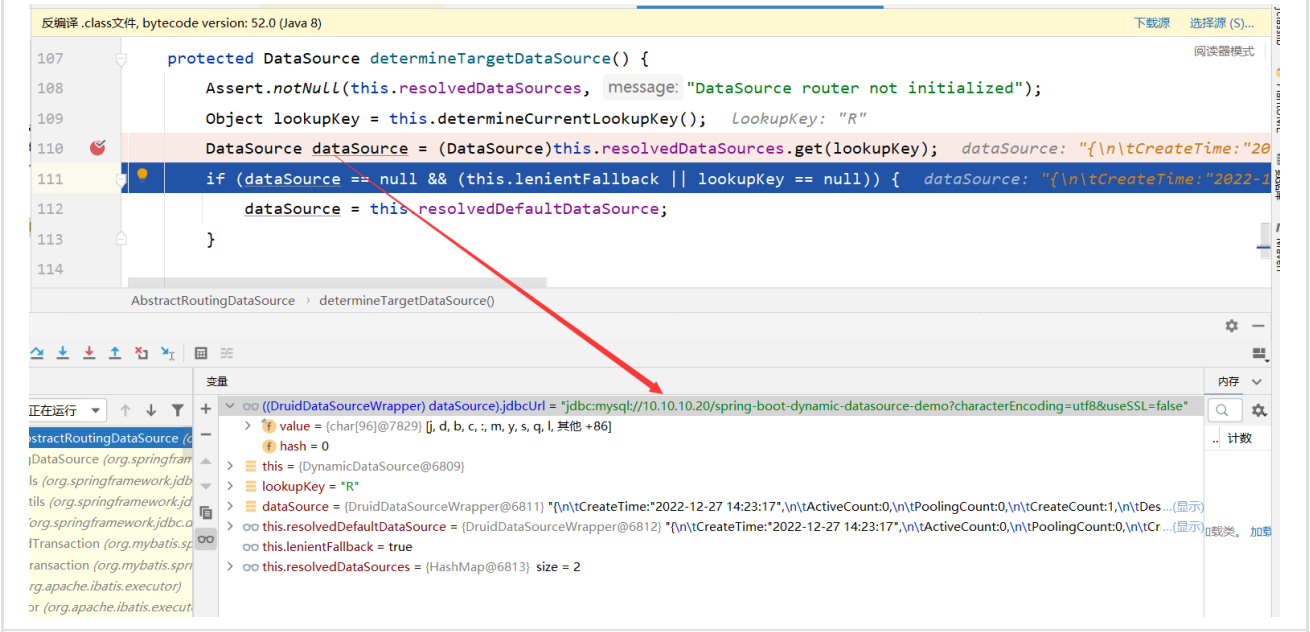
2.3.7 测试接口

/user/list: 查询用户列表，查询操作，预期走从库,(10.10.10.20)

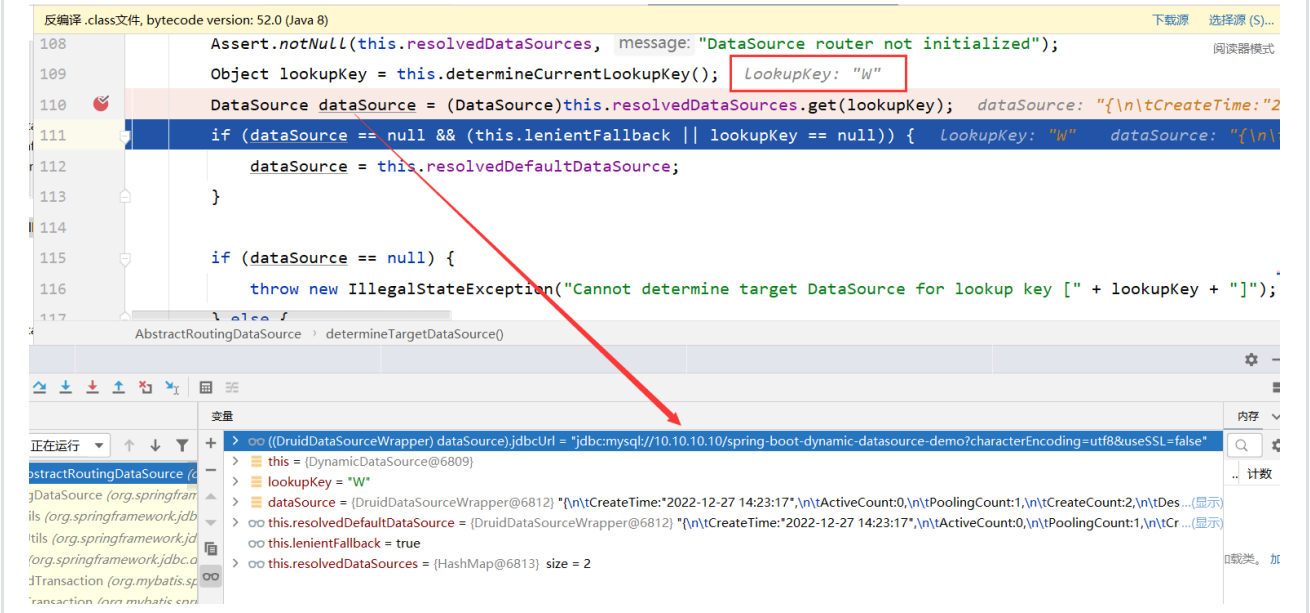
/user/insert: 插入一条数据,DML操作，预期走主库,(10.10.10.10)

```
/*
 * @Project:spring-boot-dynamic-datasource-demo
 * @Author:cxs
 * @Motto:放下杂念,只为迎接明天更好的自己
 * */
@RestController
@RequestMapping("/user")
public class UserController {

    @Autowired
    private UserService userService;

    @GetMapping("/list")
    public Object list(){
        return userService.list();
    }

    @GetMapping("/insert")
    public Object insert(){
        User user = new User();
        user.setUserName("admin");
        return userService.insert(user);
    }
}
```

/user/list



/user/insert



验证通过

2.3.8 补充

上述功能已经实现，但是存在一个问题，我们使用了ThreadLocal，弊端就不多说了，解决方式，新建一个拦截器

```
/*
 * @Project:spring-boot-dynamic-datasource-demo
 * @Author:cxs
 * @Motto:放下杂念,只为迎接明天更好的自己
 * */
public class ClearDynamicKeyInterCeptor implements HandlerInterceptor {
    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse response,
Object handler, Exception ex) throws Exception {
        // 清理DynamicLocal
        DynamicDatasourceLocalUtil.removeLocalCache();
    }
}
```

注册为/**拦截就可以了

2.4 SpringBoot整合多数据源-Aop方式

2.4.1 前置知识说明

这种方式需要对于Spring的切面有一定了解程度

上面说了使用mybatis插件实现数据源切换，那么有没有弊端，当然是有的，举个例子，首先表明一下，多数据源不一定都是主从

有这个场景，有两个数据源，分别是order库，shop库。订单库和商品库，查询商品的时候需要查询shop库，下订单的时候我要连order库，这种业务场景就不适用于插件方式了

总结一下哈：

插件方式对于主从结构来说，比较方便，但是多个数据源就会有瓶颈

AOP比较灵活，适用于多个数据源的业务场景

2.4.2 数据库环境搭建

aop方式就不适用主从结构了，实际场景应该是两个不同机器的主库，我这为了方便，同一台机器创建连个库，一个shop库，一个order库，想想哈，其实是一样的

在主库执行

```
create database `spring-boot-dynamic-datasource-aop-order` character set 'utf8mb4';
create database `spring-boot-dynamic-datasource-aop-shop` character set 'utf8mb4';
grant all on `spring-boot-dynamic-datasource-aop-order`.* to 'dynamic-datasource-user'@'%'
identified by '@Aa1234567890';
grant all on `spring-boot-dynamic-datasource-aop-shop`.* to 'dynamic-datasource-user'@'%'
identified by '@Aa1234567890';
flush privileges;
```

在spring-boot-dynamic-datasource-aop-shop库建立t_product表

```
create table t_product(
    id int primary key auto_increment,
    product_name varchar(50),
    price int
) comment '商品表';
insert into t_product values(1, '华为手机', 3000);
```

在spring-boot-dynamic-datasource-aop-order表创建t_order表

```
create table t_order(
    id int primary key auto_increment,
    product_id int,
    create_time datetime
) comment '订单表';
insert into t_order values(1, 1, '2022-12-27 12:30:15');
```

2.4.3 创建项目&导入依赖

依赖于上述类似，新增aop依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

2.4.4 application.yml

当前场景使用两个主库

```
server:
  port: 2022
spring:
  datasource-order:
    type: com.alibaba.druid.pool.DruidDataSource
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql://10.10.10.10/spring-boot-dynamic-datasource-aop-order?
characterEncoding=utf8&useSSL=false
    username: dynamic-datasource-user
    password: "@Aa1234567890"
  datasource-shop:
    type: com.alibaba.druid.pool.DruidDataSource
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql://10.10.10.10/spring-boot-dynamic-datasource-aop-shop?
characterEncoding=utf8&useSSL=false
    username: dynamic-datasource-user
    password: "@Aa1234567890"
mybatis:
  configuration:
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
    map-underscore-to-camel-case: true
  type-aliases-package: com.cxs.model
  mapper-locations: classpath:mapper/*.xml
```

2.4.5 MybatisConfig

多个数据源配置，配置两个（可自由配置）

```
/*
 * @Project:spring-boot-dynamic-datasource-demo
 * @Author:cxs
 * @Motto:放下杂念,只为迎接明天更好的自己
 * */
@Configuration
@MapperScan(basePackages = "com.cxs.mapper")
public class MybatisConfig {

    @Bean("dataSourceOrder")
    @ConfigurationProperties(prefix = "spring.datasource-order")
    public DataSource dataSourceOrder(){
        return DruidDataSourceBuilder.create().build();
    }

    @Bean("dataSourceShop")
    @ConfigurationProperties(prefix = "spring.datasource-shop")
    public DataSource dataSourceShop(){
        return DruidDataSourceBuilder.create().build();
    }

}
```

2.4.6 DynamicDataSource

```
/*
 * @Project:spring-boot-dynamic-datasource-demo
 * @Author:cxs
 * @Motto:放下杂念,只为迎接明天更好的自己
 * */
@Primary
@Component
public class DynamicDataSource extends AbstractRoutingDataSource {

    @Autowired
    private DataSource dataSourceShop;

    @Autowired
    private DataSource dataSourceOrder;

    @Override
    protected Object determineCurrentLookupKey() {
        return DynamicDatasourceLocalUtil.getLocalCache();
    }

    @Override
    public void afterPropertiesSet() {
        Map<Object, Object> map = new HashMap<>();
        super.setDefaultTargetDataSource(dataSourceShop);
        map.put(TypeEnum.T_ORDER, dataSourceOrder);
```

```
        map.put(TypeEnum.T_SHOP, dataSourceShop);
        super.setTargetDataSources(map);
        super.afterPropertiesSet();
    }
}
```

2.4.7 自定义注解&切面实现

aop方式可以在方法执行完毕后，清除ThreadLocal，无需建立拦截器处理

```
/*
 * @Project:spring-boot-dynamic-datasource-demo
 * @Author:cxs
 * @Motto:放下杂念,只为迎接明天更好的自己
 * */
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface DataSourceType {
    TypeEnum value();
}
```

```
/*
 * @Project:spring-boot-dynamic-datasource-demo
 * @Author:cxs
 * @Motto:放下杂念,只为迎接明天更好的自己
 * */
@Aspect
@Component
public class DynamicDataSourceAspect {

    @Pointcut(value = "within(com.cxs.service.impl.*)")
    public void pointCut(){}

    @Around(value = "pointCut() && @annotation(dataSourceType)")
    public Object around(ProceedingJoinPoint proceedingJoinPoint, DataSourceType
dataSourceType){
        Object result = null;
        try {
            // 设置当前处理线程的数据源标识
            DynamicDatasourceLocalUtil.setLocalCache(dataSourceType.value());
            result = proceedingJoinPoint.proceed();
        } catch (Throwable e) {
            e.printStackTrace();
        } finally {
            // 清除ThreadLocal，防止内存泄漏
            DynamicDatasourceLocalUtil.removeLocalCache();
        }
        return result;
    }
}
```

2.4.8 业务实现

以ProductServiceImpl为例

@DataSourceType(TypeEnum.T_SHOP) 指定当前业务的数据源

```
/*
 * @Project:spring-boot-dynamic-datasource-demo
 * @Author:cxs
 * @Motto:放下杂念,只为迎接明天更好的自己
 * */
@Service
public class ProductServiceImpl implements ProductService {

    @Autowired
    private ProductMapper productMapper;

    @Override
    @DataSourceType(TypeEnum.T_SHOP)
    public List<Product> productList() {
        return productMapper.selectList();
    }
}
```

2.4.9 GlobalController

两个接口：

/shop/list：使用spring-boot-dynamic-datasource-aop-shop库

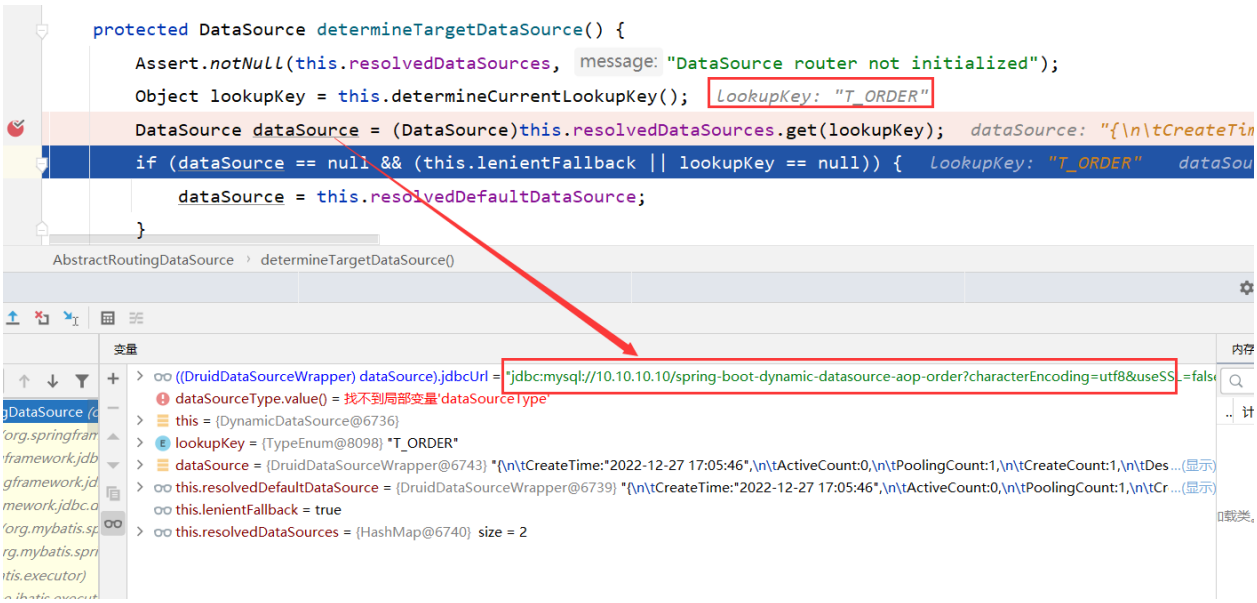
/order/add: 使用spring-boot-dynamic-datasource-aop-order库

```
/*
 * @Project:spring-boot-dynamic-datasource-demo
 * @Author:cxs
 * @Motto:放下杂念,只为迎接明天更好的自己
 * */
@RestController
public class GlobalController {

    @Autowired
    private OrderService orderService;

    @Autowired
    private ProductService productService;

    @GetMapping("/shop/list")
    public Object list(){
        return productService.productList();
    }

    @GetMapping("/order/add")
    public Object add(){
        Order order = new Order();
        order.setProductId(1);
```

```
        order.setCreateTime(LocalDateTime.now());
        return orderService.addOrder(order);
    }
}
```

/shop/list

/order/add



测试通过，与预期结果一致

2.5 SpringBoot整合多数据源-多个Mybatis方式

2.5.1 数据库环境搭建

使用Mybatis插件方式数据库

2.5.2 创建项目&导入依赖

与Mybatis插件方式完全一致

2.5.3 application.yml

```yaml
server:
  port: 2022
spring:
  datasource-master:
    type: com.alibaba.druid.pool.DruidDataSource
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql://10.10.10.10/spring-boot-dynamic-datasource-demo?
characterEncoding=utf8&useSSL=false
    username: dynamic-datasource-user
    password: "@Aa1234567890"
  datasource-slave:
    type: com.alibaba.druid.pool.DruidDataSource
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql://10.10.10.20/spring-boot-dynamic-datasource-demo?
characterEncoding=utf8&useSSL=false
    username: dynamic-datasource-user
    password: "@Aa1234567890"
```

2.5.4 主库配置类MasterMybatisConfig

由于要配置多个Mybatis环境，我们要自己创建多份SqlSessionFactory

```java
/*
 * @Project:spring-boot-dynamic-datasource-demo
 * @Author:cxs
 * @Motto:放下杂念,只为迎接明天更好的自己
 * */
@Configuration
// 我们要通过sqlSessionFactoryRef指定mapper使用哪个SqlSessionFactory
@MapperScan(basePackages = "com.cxs.mapper.master", sqlSessionFactoryRef =
"masterSqlSessionFactory")
public class MasterMybatisConfig {

    @Bean
    @ConfigurationProperties(prefix = "spring.datasource-master")
    public DataSource dataSourceMaster(){
        return DruidDataSourceBuilder.create().build();
    }

    @Bean
    public SqlSessionFactory masterSqlSessionFactory() throws Exception {
        SqlSessionFactoryBean sqlSessionFactoryBean = new SqlSessionFactoryBean();
        // 指定数据源
        sqlSessionFactoryBean.setDataSource(dataSourceMaster());
        // 指定mapper文件
        sqlSessionFactoryBean.setMapperLocations(new
PathMatchingResourcePatternResolver().getResources("classpath:mapper/master/*.xml"));
        return sqlSessionFactoryBean.getObject();
```

```
    }
}
```

2.5.5 从库配置类SlaveMybatisConfig

```
/*
 * @Project:spring-boot-dynamic-datasource-demo
 * @Author:cxs
 * @Motto:放下杂念,只为迎接明天更好的自己
 * */
@Configuration
@MapperScan(basePackages = "com.cxs.mapper.slave", sqlSessionFactoryRef =
"slaveSqlSessionFactory")
public class SlaveMybatisConfig {

    @Bean("slave")
    @ConfigurationProperties(prefix = "spring.datasource-slave")
    public DataSource dataSourceSlave(){
        return DruidDataSourceBuilder.create().build();
    }

    @Bean
    public SqlSessionFactory slaveSqlSessionFactory() throws Exception {
        SqlSessionFactoryBean sqlSessionFactoryBean = new SqlSessionFactoryBean();
        sqlSessionFactoryBean.setDataSource(dataSourceSlave());
        sqlSessionFactoryBean.setMapperLocations(new
PathMatchingResourcePatternResolver().getResources("classpath:mapper/slave/*.xml"));
        return sqlSessionFactoryBean.getObject();
    }
}
```

2.5.6 业务类

使用不同的mapper进行操作数据库

```
/*
 * @Project:spring-boot-dynamic-datasource-demo
 * @Author:cxs
 * @Motto:放下杂念,只为迎接明天更好的自己
 * */
@Service
public class UserServiceImpl implements UserService {

    @Autowired
    private MasterUserMapper masterUserMapper;

    @Autowired
    private SlaveUserMapper slaveUserMapper;

    @Override
    public List<User> list() {
        return slaveUserMapper.selectList();
    }
```

```
    @Override
    public Integer insert(User user) {
        return masterUserMapper.insert(user);
    }
}
```

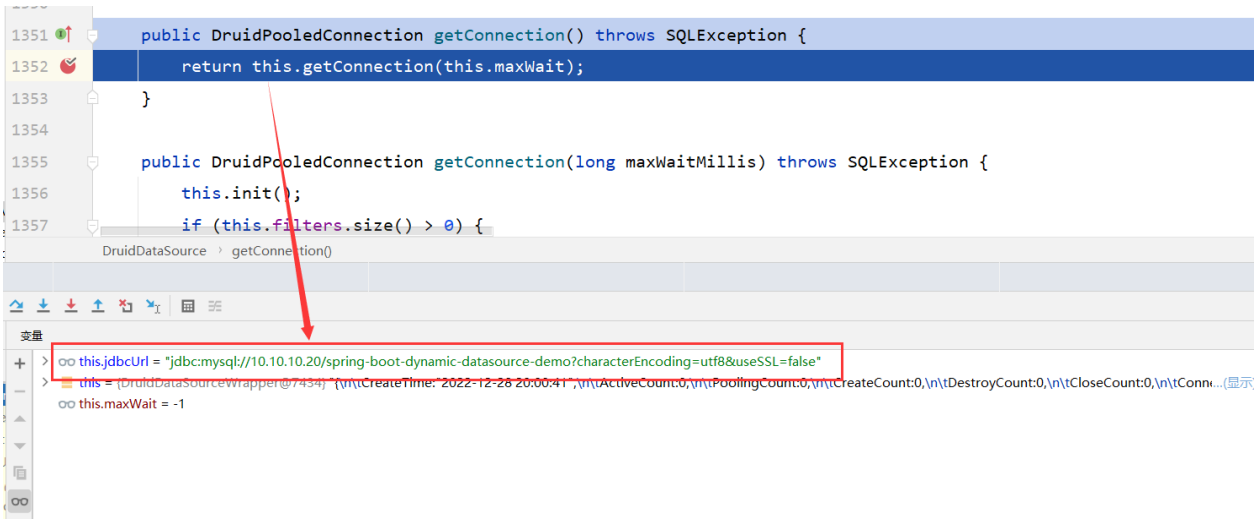
2.5.7 功能测试

两个接口

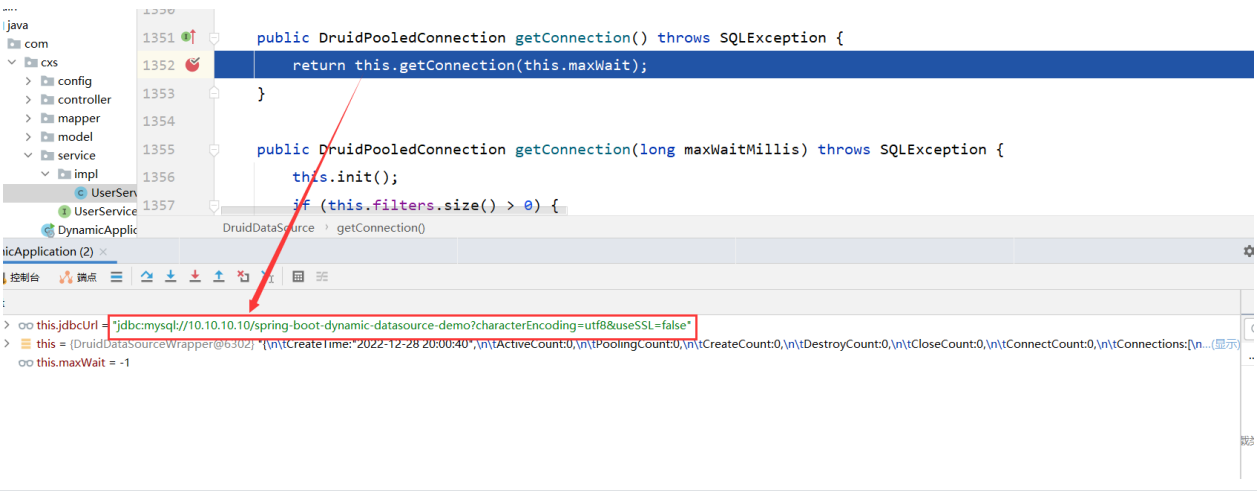
/user/list: Select操作，走slave库，（10.10.10.20）

/user/insert: Insert操作，走mater库，（10.10.10.10）

/user/list



/user/insert



测试通过，与预期结果一致

2.6 SpringBoot整合多数据源-dynamic-datasource方式

2.6.1 组件说明

第三方组件，详细文档地址： https://www.kancloud.cn/tracy5546/dynamic-datasource/2264611

原理跟我们使用Aop整合很类似，同样是使用切面+自定义注解实现

特性

- 支持 **数据源分组** ，适用于多种场景 纯粹多库 读写分离 一主多从 混合模式。
- 支持数据库敏感配置信息 **加密** ENC()。
- 支持每个数据库独立初始化表结构schema和数据库database。
- 支持无数据源启动，支持懒加载数据源（需要的时候再创建连接）。
- 支持 **自定义注解** ，需继承DS(3.2.0+)。
- 提供并简化对Druid，HikariCp，BeeCp，Dbcp2的快速集成。
- 提供对Mybatis-Plus，Quartz，ShardingJdbc，P6sy，Jndi等组件的集成方案。
- 提供 **自定义数据源来源** 方案（如全从数据库加载）。
- 提供项目启动后 **动态增加移除数据源** 方案。
- 提供Mybatis环境下的 **纯读写分离** 方案。
- 提供使用 **spel动态参数** 解析数据源方案。内置spel，session，header，支持自定义。
- 支持 **多层数据源嵌套切换** 。（ServiceA >>> ServiceB >>> ServiceC）。
- 提供 **基于seata的分布式事务方案。**
- 提供 **本地多数据源事务方案。** 附：不能和原生spring事务混用。

约定

1. 本框架只做 **切换数据源** 这件核心的事情，并**不限制你的具体操作**，切换了数据源可以做任何CRUD。
2. 配置文件所有以下划线 `_` 分割的数据源 **首部** 即为组的名称，相同组名称的数据源会放在一个组下。
3. 切换数据源可以是组名，也可以是具体数据源名称。组名则切换时采用负载均衡算法切换。
4. 默认的数据源名称为 **master** ，你可以通过 `spring.datasource.dynamic.primary` 修改。
5. 方法上的注解优先于类上注解。
6. DS支持继承抽象类上的DS，暂不支持继承接口上的DS。

2.6.2 创建项目&导入依赖

在Mybatis插件基础上新增依赖

```
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>dynamic-datasource-spring-boot-starter</artifactId>
    <version>3.5.0</version>
</dependency>
```

2.6.3 application.yml

```
server:
  port: 2022

spring:
  datasource:
    dynamic:
      primary: master
      strict: false
      datasource:
        master:
          type: com.alibaba.druid.pool.DruidDataSource
          driver-class-name: com.mysql.jdbc.Driver
          url: jdbc:mysql://10.10.10.10/spring-boot-dynamic-datasource-demo?
characterEncoding=utf8&useSSL=false
          username: dynamic-datasource-user
          password: "@Aa1234567890"
        slave_1:
          type: com.alibaba.druid.pool.DruidDataSource
          driver-class-name: com.mysql.jdbc.Driver
          url: jdbc:mysql://10.10.10.20/spring-boot-dynamic-datasource-demo?
characterEncoding=utf8&useSSL=false
          username: dynamic-datasource-user
          password: "@Aa1234567890"
mybatis:
  configuration:
    map-underscore-to-camel-case: true
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
  type-aliases-package: com.cxs.model
  mapper-locations: classpath:mapper/*.xml
```

2.6.4 业务类使用

业务类使用@DS注解标注数据源标识，该框架带有负载均衡等策略，可以研究文档(付费)

```
/*
 * @Project:spring-boot-dynamic-datasource-demo
 * @Author:cxs
 * @Motto:放下杂念,只为迎接明天更好的自己
 * */
@Service
public class UserServiceImpl implements UserService {
```

```
    @Autowired
    private UserMapper userMapper;

    @Override
    @DS(value = "slave_1")
    public List<User> list() {
        return userMapper.selectList();
    }

    @Override
    @DS(value = "master")
    public Integer insert(User user) {
        return userMapper.insert(user);
    }
}
```

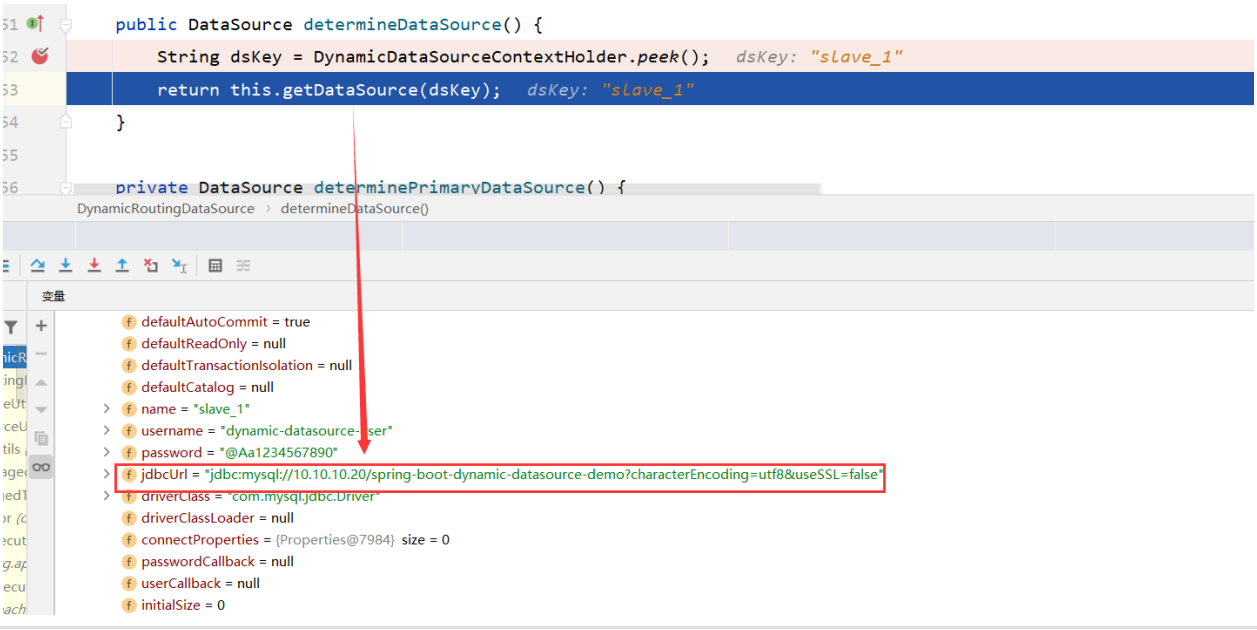
2.6.5 功能测试

两个接口

/user/list: Select操作，走slave库，（10.10.10.20）

/user/insert: Insert操作，走mater库，（10.10.10.10）

/user/list



/user/insert