自定义多数据源动态切换（SpringBoot+Mybatis）实现数据库读写分离

1，自定义多数据源动态切换原理

2，自定义数据源动态实现（基于Spring AOP+SpringBoot+Mybatis）

1，准备工作

2，核心代码

3，结果展示

3，多数据源事物管理

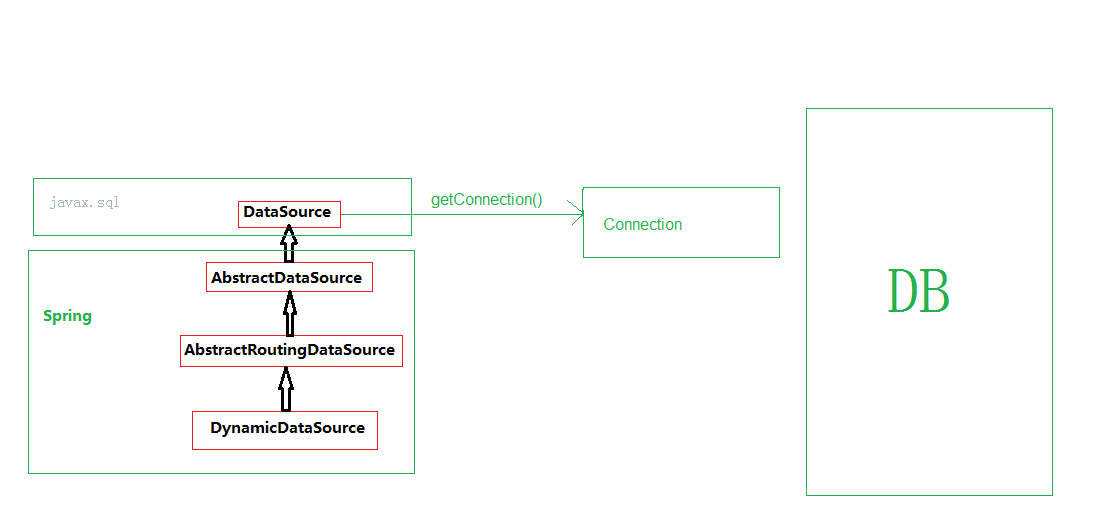
1，切换数据源会不会造成事务切换错误？

总结

本文是使用Spring2.0以后新增的AbstractRoutingDataSource类来实现多数据源动态切换，文章末尾附上可执行的源码（注意修改数据库配置）。

1，自定义多数据源动态切换原理

AbstractRoutingDataSource类是spring提供用来控制当前线程最终选择某个数据源的路由器。我们自定义一个动态数据源DynamicDataSource类来继承AbstractRoutingDataSource。AbstractRoutingDataSource继承于 AbstractDataSource，AbstractDataSource继承于javax.sql.DataSource，DataSource通过getDataSource()方法得到连接从而操作数据库。如下图：AbstractDataSource，AbstractDataSource继承于javax.sql.DataSource，DataSource通过getDataSource()方法得到连接从而操作数据库。如下图：



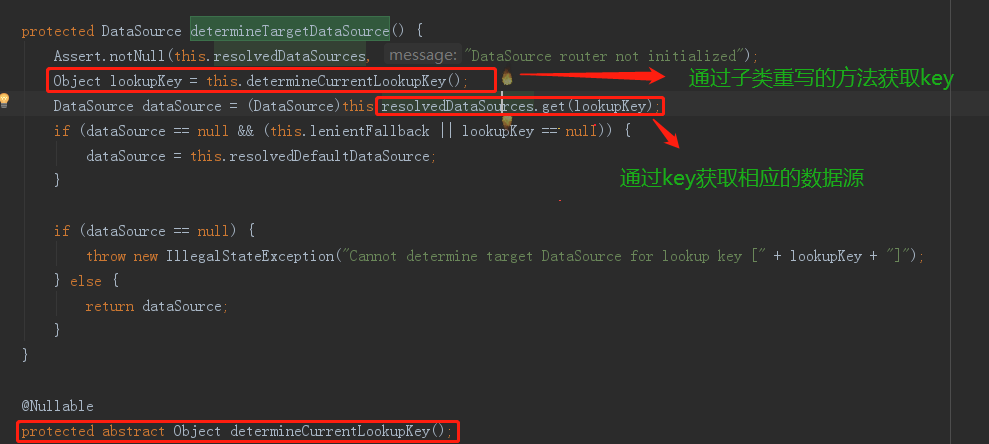
2.DynamicDataSource实现determineCurrentLookupKey()方法，该方法返回最终要选择的数据源名称key，然后通过key路由器将会路由到正确的数据源上面，得到预期数据源。

具体如下：

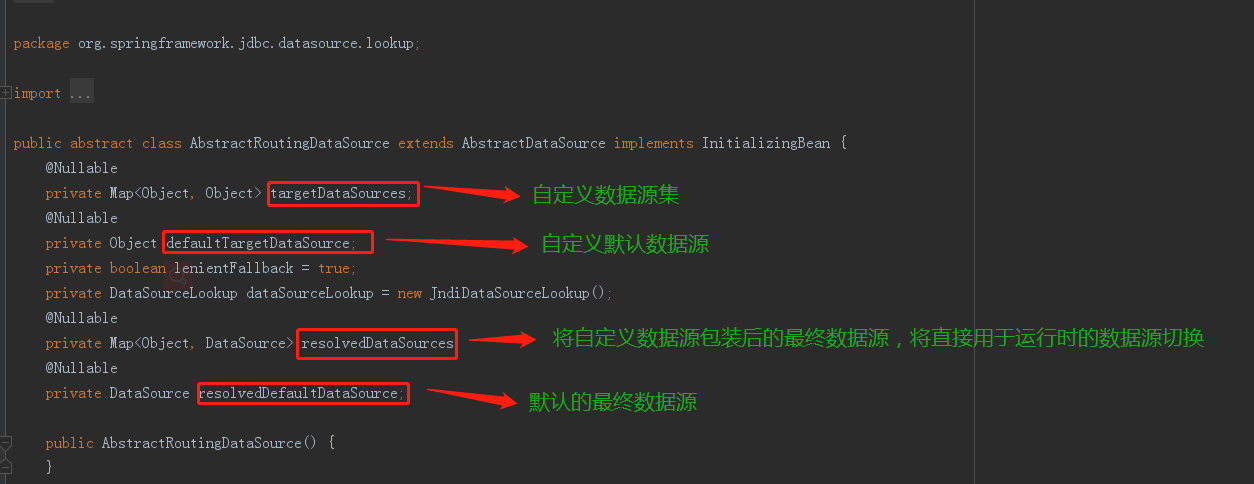
1. DynamicDataSource先继承DynamicDataSource实现determineCurrentLookupKey()方法；



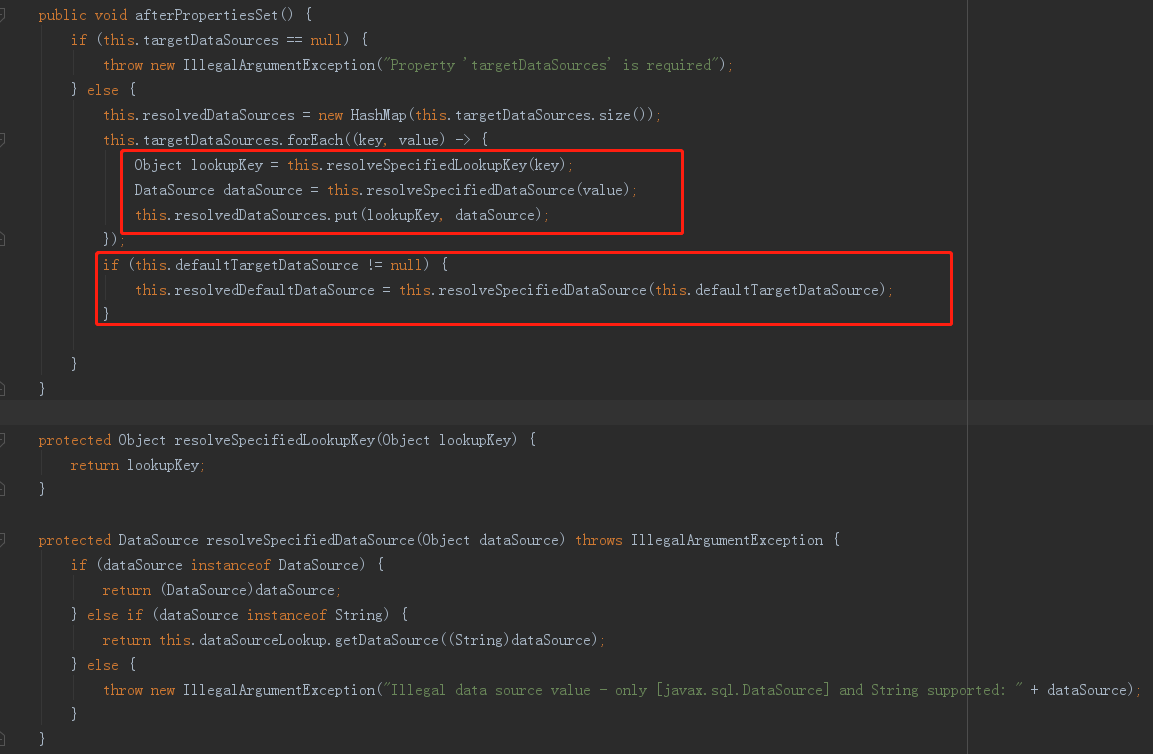
2.AbstractRoutingDataSource 调用自己的determineTargetDataSource()方法，determineTargetDataSource()方法中就调用了我们实现的determineCurrentLookupKey()方法得到key然后从resolvedDataSources中按照key取出指定的数据源



3. 来看看AbstractRoutingDataSource 的一些属性吧。targetDataSources是我们设置的自定义数据源集(这个是需要我们手动设置的)，defaultTargetDataSource这个使我们设置的默认的数据源(也是需要手动设置的)，resolvedDataSources这个是将我们传入进来的数据源集进过包装后的最终在程序运行时切换的数据源集(这个是自动设置的，不需要我们手动设置)，resolvedDefaultDataSource这个是默认的系统在运行时选择的数据源



那么resolvedDataSources是怎么样被targetDataSources填充的呢？答案也在源码里，它在程序加载完配置文件后，填充到resolvedDataSources里面的，调用的方法是afterPropertiesSet(),在afterPropertiesSet()方法中调用resolveSpecifiedDataSource()方法将不同厂商的数据源转换成DataSource存储，什么时候设置targetDataSources我们将在下面的具体实现的核心代码部分看见

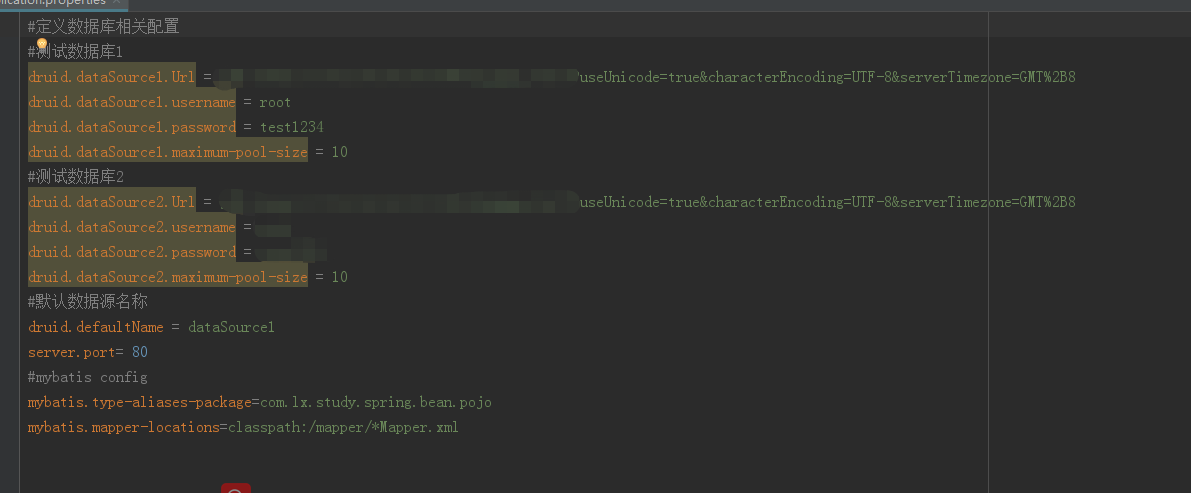


最后便是调用从DataSource一层一层继承下来的getConnection()方法来获取连接，从而操作数据库。

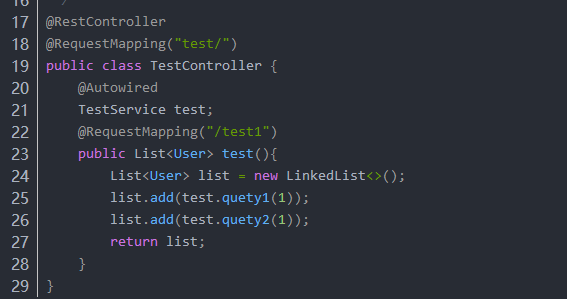
总结一下： 通过自定义动态数据DynamicDataSource重写的determineCurrentLookupKey()方法程序动态的获得不同的数据源名称，从而取出不同的数据源获得连接，实现动态数据源切换的功能

# 2，自定义数据源动态实现（基于Spring AOP+SpringBoot+Mybatis）

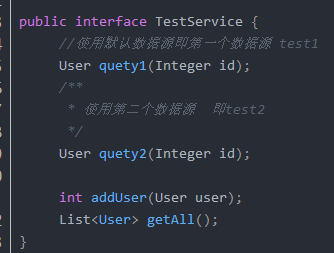
准备web项目常见的controller(可以不要，我们可以用单元测试),service,mapper层以及实体类User和数据库配置



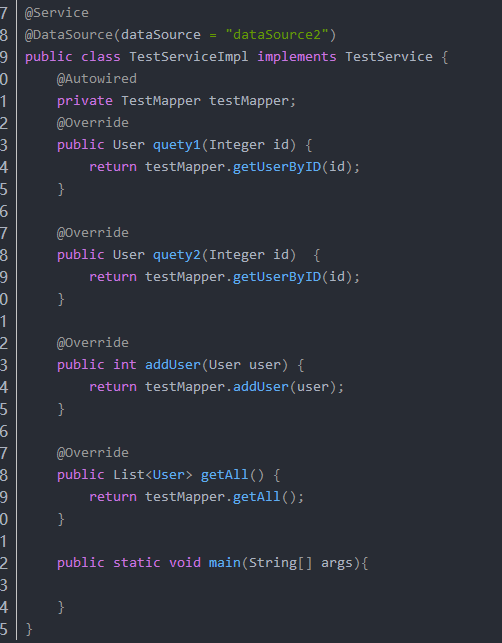
Controlller类



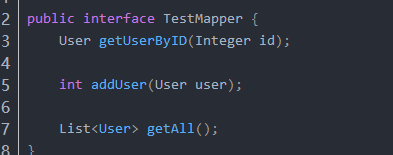
接口



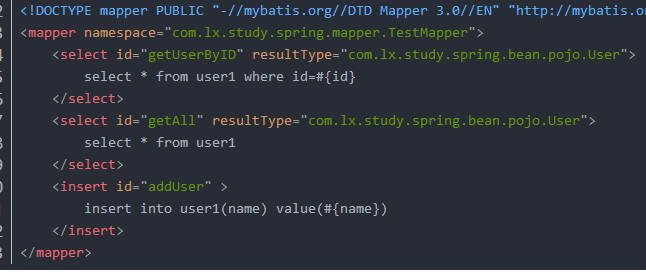
实现类



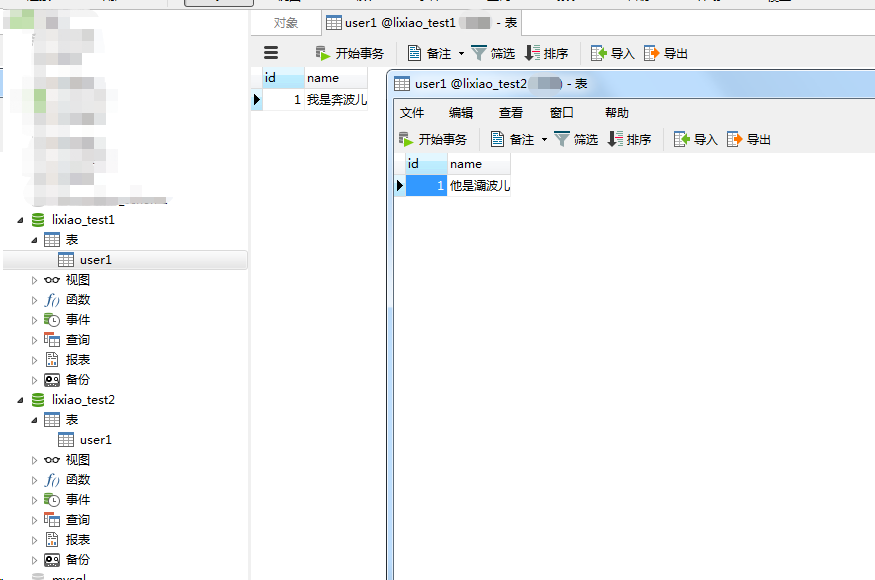
Mapper



Mapper.xml



建两个数据库，并建立相同的表user1，并添加一条记录



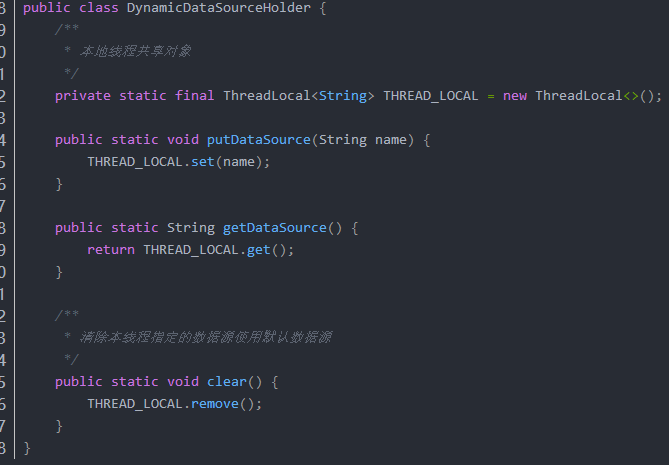
## 2，核心代码

类DBProperties 用来读取所有配置的数据源，这里利用的是其中

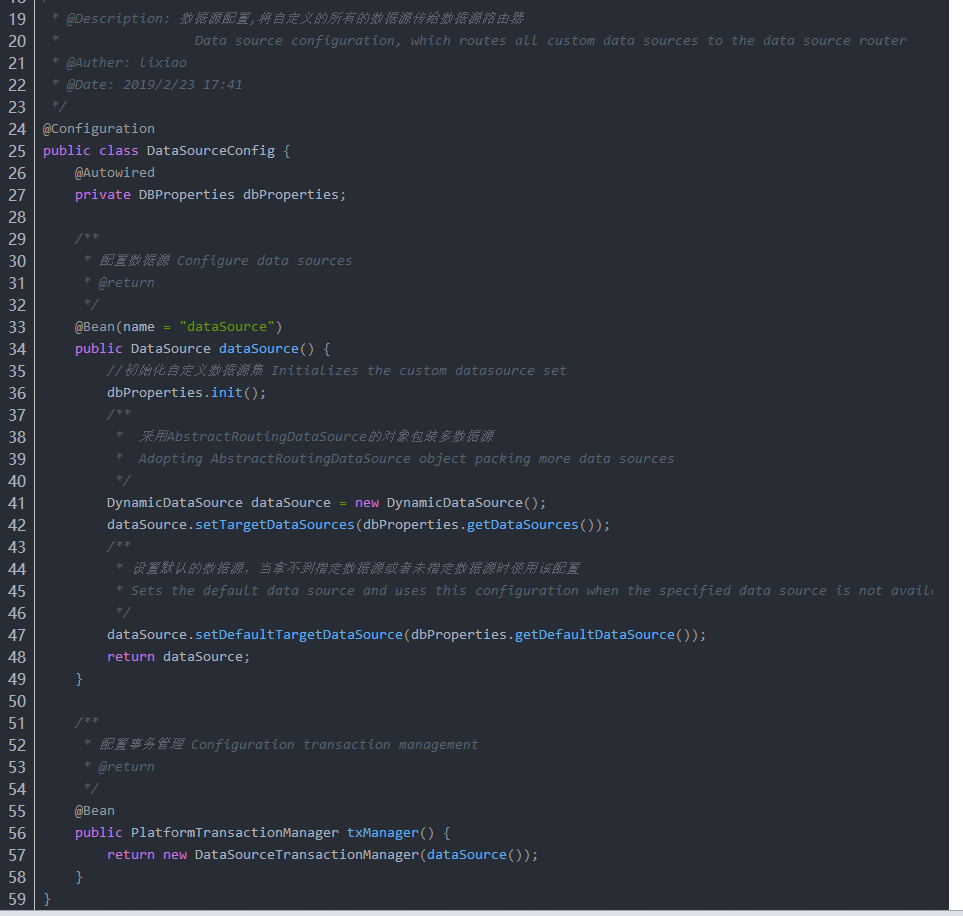
@ConfigurationProperties的大致作用就是通过它可以把properties或者yml配置直接转成对象(并获取properties的值)  
prefix = "druid"是properties里的前缀



类DynamicDataSourceHolder 这个类是实现动态数据源切换的关键，因为DynamicDataSource是单例的所以我们采用ThreadLocal一来保证线程的安全性，而来保证同一个线程获取数据源的唯一性。putDataSource(name)方法用于切换数据源，get()方法用于获取数据源，clear()方法用于清空本地线程的所有数据源，让其使用默认的数据源

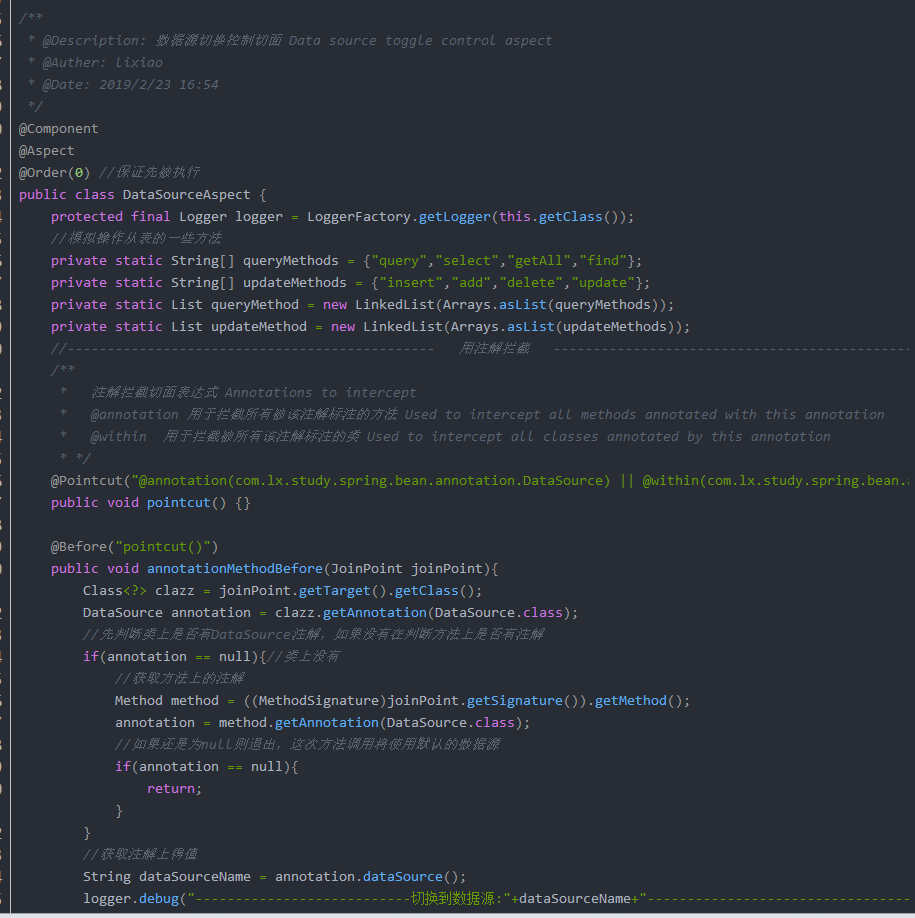


类DataSourceConfig，这个类使用将我们定义好的数据源集装载到数据源路由器中(AbstractRoutingDataSource)。也是在这个类设置我们上面提到的targetDataSources(自定义数据源集)和DefaultTargetDataSource(默认的自定义数据源)，换需要配置事物管理对象，关于事物管理在后面会详细讲！到





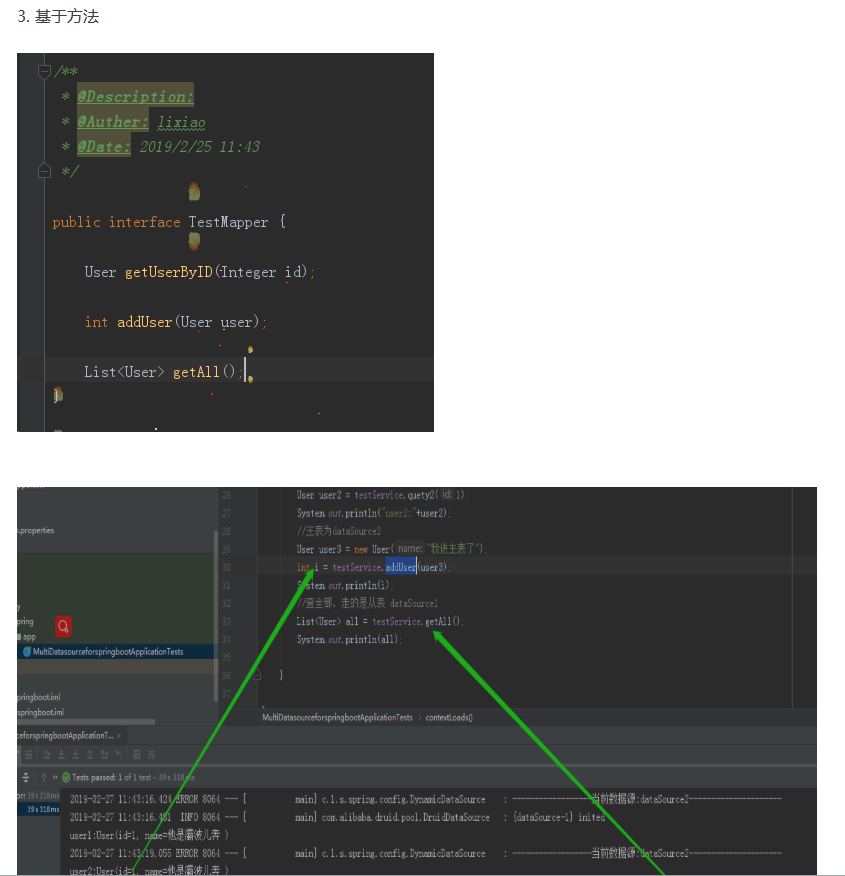
最后一个，也是核心的一个类DataSourceAspect，它是一个切面，来实现动态切换数据库，本文一共演示了3种拦截逻辑。1是基于注解拦截。2是基于类拦截。3是基于方法拦截，具体怎么选择和拓展可更具业务选择。需要注意的是，每个被拦截的方法或类，在执行完方法后，必须清空本地线程中的数据源，也就是调用DynamicDataSourceHolder的clear()方法，











这里一般不需要在方法获类上面用@DataSource注解了（而是通过方法名称的约定来判断用读库还是用写库）

原博客

https://blog.csdn.net/Mr\_lifengzi/article/details/87981864