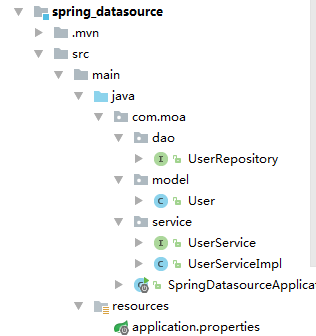
# 1 JPA数据源（单）

## 1.1 项目结构：



## 1.2 POM.XML配置

|  |
| --- |
| <**parent**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-parent</**artifactId**>  <**version**>1.5.7.RELEASE</**version**>  <**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**> </**properties**>  <**dependencies**>  <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter</**artifactId**>  </**dependency**>   <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-test</**artifactId**>  <**scope**>test</**scope**>  </**dependency**>   <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-data-jpa</**artifactId**>  </**dependency**>   *<!--MySQL驱动 ，-->* <**dependency**>  <**groupId**>mysql</**groupId**>  <**artifactId**>mysql-connector-java</**artifactId**>  <**version**>5.1.21</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.springframework.data</**groupId**>  <**artifactId**>spring-data-jpa</**artifactId**>  <**version**>1.11.1.RELEASE</**version**>  </**dependency**> </**dependencies**> |

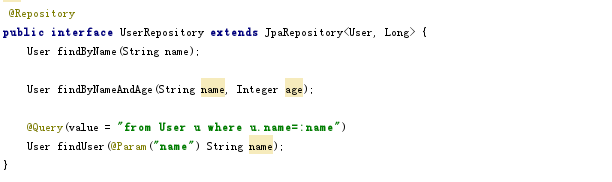
## 1.3 application.properties

|  |
| --- |
| **spring.datasource.url**=**jdbc:mysql://localhost:3306/test spring.datasource.username**=**root spring.datasource.password**=**123456 spring.datasource.driver-class-name**=**com.mysql.jdbc.Driver spring.jpa.properties.hibernate.hbm2ddl.auto**=**create** |

## 1.4 实体

|  |
| --- |
| **package** com.moa.model;  **import** javax.persistence.\*;  */\*\*  \* Created by Administrator on 2017/10/17.  \*/* @Entity @Table(name = **"t\_user"**) **public class** User {  @Id  @GeneratedValue  **private** Long **id**;  @Column(nullable = **false**)  **private** String **name**;  @Column(nullable = **false**)  **private** Integer **age**;   **public** User() {  }   **public** User(String name, Integer age) {  **this**.**name** = name;  **this**.**age** = age;  }   **public** Long getId() {  **return id**;  }   **public** User setId(Long id) {  **this**.**id** = id;  **return this**;  }   **public** String getName() {  **return name**;  }   **public** User setName(String name) {  **this**.**name** = name;  **return this**;  }   **public** Integer getAge() {  **return age**;  }   **public** User setAge(Integer age) {  **this**.**age** = age;  **return this**;  } } |

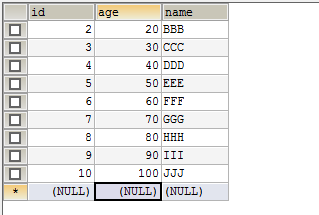
## 1.5 dao



## 1.6 测试

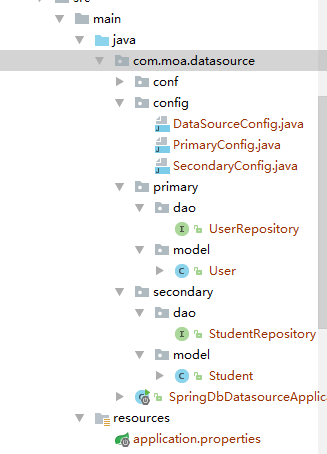
|  |
| --- |
| @Test **public void** testSave() **throws** Exception {  *// 创建10条记录* **userRepository**.save(**new** User(**"AAA"**, 10));  **userRepository**.save(**new** User(**"BBB"**, 20));  **userRepository**.save(**new** User(**"CCC"**, 30));  **userRepository**.save(**new** User(**"DDD"**, 40));  **userRepository**.save(**new** User(**"EEE"**, 50));  **userRepository**.save(**new** User(**"FFF"**, 60));  **userRepository**.save(**new** User(**"GGG"**, 70));  **userRepository**.save(**new** User(**"HHH"**, 80));  **userRepository**.save(**new** User(**"III"**, 90));  **userRepository**.save(**new** User(**"JJJ"**, 100));  *// 测试findAll, 查询所有记录* Assert.*assertEquals*(10, **userRepository**.findAll().size());  *// 测试findByName, 查询姓名为FFF的User* Assert.*assertEquals*(60, **userRepository**.findByName(**"FFF"**).getAge().longValue());  *// 测试findUser, 查询姓名为FFF的User* Assert.*assertEquals*(60, **userRepository**.findUser(**"FFF"**).getAge().longValue());  *// 测试findByNameAndAge, 查询姓名为FFF并且年龄为60的User* Assert.*assertEquals*(**"FFF"**, **userRepository**.findByNameAndAge(**"FFF"**, 60).getName());  *// 测试删除姓名为AAA的User* **userRepository**.delete(**userRepository**.findByName(**"AAA"**));  System.***out***.println(**"Hello"**);  *// 测试findAll, 查询所有记录, 验证上面的删除是否成功* Assert.*assertEquals*(9, **userRepository**.findAll().size()); } |

## 1.7 结果



# JPA多数据源（不同步）

## 2.1项目结构



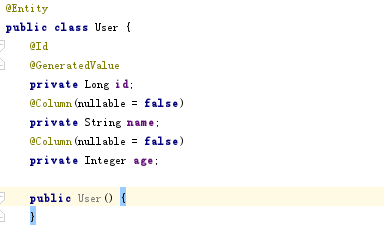
## 2.2 POM.XML

|  |
| --- |
| *<?***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.moa</**groupId**>  <**artifactId**>spring\_db\_datasource</**artifactId**>  <**version**>0.0.1-SNAPSHOT</**version**>  <**packaging**>jar</**packaging**>   <**name**>spring\_db\_datasource</**name**>  <**description**> </**description**>   <**parent**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-parent</**artifactId**>  <**version**>1.5.7.RELEASE</**version**>  <**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**>  </**properties**>   <**dependencies**>  <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter</**artifactId**>  </**dependency**>   <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-test</**artifactId**>  <**scope**>test</**scope**>  </**dependency**>   <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-data-jpa</**artifactId**>  </**dependency**>   *<!--MySQL驱动 ，-->* <**dependency**>  <**groupId**>mysql</**groupId**>  <**artifactId**>mysql-connector-java</**artifactId**>  <**version**>5.1.21</**version**>  </**dependency**>    *<!--oracle 驱动-->* <**dependency**>  <**groupId**>cn.osworks.lib</**groupId**>  <**artifactId**>ojdbc6</**artifactId**>  <**version**>6</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.springframework.data</**groupId**>  <**artifactId**>spring-data-jpa</**artifactId**>  <**version**>1.11.1.RELEASE</**version**>  </**dependency**>   *<!-- postgresql驱动 -->* <**dependency**>  <**groupId**>postgresql</**groupId**>  <**artifactId**>postgresql</**artifactId**>  <**version**>9.1-901.jdbc4</**version**>  </**dependency**>   </**dependencies**>   <**build**>  <**plugins**>  <**plugin**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-maven-plugin</**artifactId**>  </**plugin**>  </**plugins**>  </**build**>  </**project**> |

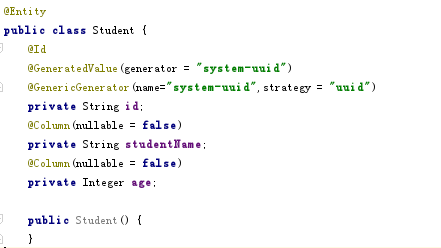
## 2.3 application.properties

|  |
| --- |
| *!-- 主数据源url* **spring.datasource.primary.url**=**jdbc:mysql://localhost:3306/test** *!-- 用户名* **spring.datasource.primary.username**=**root** *!-- 密码* **spring.datasource.primary.password**=**123456** *!-- 驱动* **spring.datasource.primary.driver-class-name**=**com.mysql.jdbc.Driver** *!-- dao操作层* **spring.datasource.primary.basePackages**=**com.moa.datasource.primary** *!-- 第二个数据源配置url* **spring.datasource.secondary.url**=**jdbc:oracle:thin:@192.168.6.174:1521:ORCL** *!-- 用户名* **spring.datasource.secondary.username**=**FP** *!-- 密码* **spring.datasource.secondary.password**=**123456** *!-- 驱动* **spring.datasource.secondary.driver-class-name**=**oracle.jdbc.driver.OracleDriver** *!-- dao操作层* **spring.datasource.secondary.basePackages**=**com.moa.datasource.secondary** *!-- 第三个数据源配置url !-- 此配置只对主数据源有效* **spring.jpa.properties.hibernate.hbm2ddl.auto**=**create  spring.jta.log-dir**=**./atomikos.log** |

## 2.4 实体

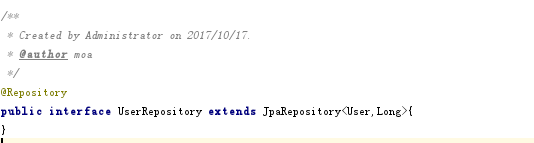


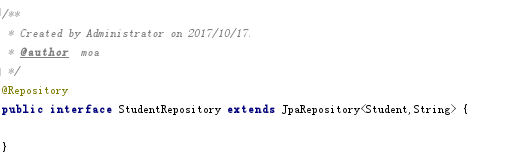
省略getter/setter



省略getter/setter

## 2.5 dao





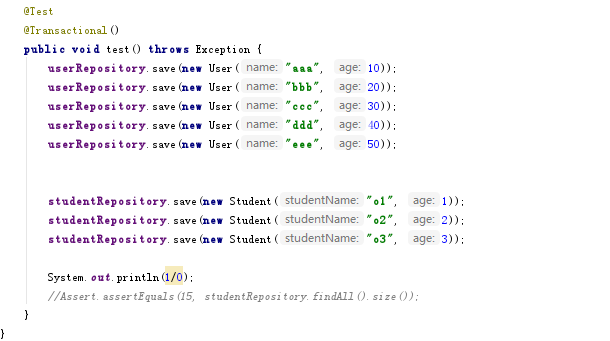
## 2.6 配置类

|  |
| --- |
| **package** com.moa.datasource.config;  **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.beans.factory.annotation.Qualifier; **import** org.springframework.beans.factory.annotation.Value; **import** org.springframework.boot.autoconfigure.orm.jpa.JpaProperties; **import** org.springframework.boot.orm.jpa.EntityManagerFactoryBuilder; **import** org.springframework.context.annotation.Bean; **import** org.springframework.context.annotation.Configuration; **import** org.springframework.data.jpa.repository.config.EnableJpaRepositories; **import** org.springframework.orm.jpa.JpaTransactionManager; **import** org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean; **import** org.springframework.transaction.PlatformTransactionManager; **import** org.springframework.transaction.annotation.EnableTransactionManagement;  **import** javax.persistence.EntityManager; **import** javax.sql.DataSource; **import** java.util.Map;  */\*\*  \* Created by Administrator on 2017/10/17.  \*  \** ***@author*** *moa  \*/* @Configuration @EnableTransactionManagement @EnableJpaRepositories(  entityManagerFactoryRef = **"entityManagerFactorySecondary"**,  transactionManagerRef = **"transactionManagerSecondary"**,  basePackages = {**"${spring.datasource.secondary.basePackages}"**}) *//设置Repository所在位置* **public class** SecondaryConfig {   *// 注入数据源* @Autowired  @Qualifier(**"secondaryDataSource"**)  **private** DataSource **secondaryDataSource**;   *// 扫描包 在application.properties中配置* @Value(**"${spring.datasource.secondary.basePackages}"**)  **private** String **basePackages**;   *// 创建事物管理器* @Bean(name = **"entityManagerSecondary"**)  **public** EntityManager entityManager(EntityManagerFactoryBuilder builder) {  **return** entityManagerFactorySecondary(builder).getObject().createEntityManager();  }    *// 创建实体工厂* @Bean(name = **"entityManagerFactorySecondary"**)  **public** LocalContainerEntityManagerFactoryBean entityManagerFactorySecondary(EntityManagerFactoryBuilder builder) {  **return** builder  .dataSource(**secondaryDataSource**)  .properties(getVendorProperties(**secondaryDataSource**))  .packages(**basePackages**) *//设置实体类所在位置* .persistenceUnit(**"secondaryPersistenceUnit"**)  .build();  }   @Autowired  **private** JpaProperties **jpaProperties**;   **private** Map<String, String> getVendorProperties(DataSource dataSource) {  **return jpaProperties**.getHibernateProperties(dataSource);  }    *// 第二个数据源的事务管理器* @Bean(name = **"transactionManagerSecondary"**)  PlatformTransactionManager transactionManagerSecondary(EntityManagerFactoryBuilder builder) {  **return new** JpaTransactionManager(entityManagerFactorySecondary(builder).getObject());  } } |

|  |
| --- |
| **package** com.moa.datasource.config;  **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.beans.factory.annotation.Qualifier; **import** org.springframework.beans.factory.annotation.Value; **import** org.springframework.boot.autoconfigure.orm.jpa.JpaProperties; **import** org.springframework.boot.orm.jpa.EntityManagerFactoryBuilder; **import** org.springframework.context.annotation.Bean; **import** org.springframework.context.annotation.Configuration; **import** org.springframework.context.annotation.Primary; **import** org.springframework.data.jpa.repository.config.EnableJpaRepositories; **import** org.springframework.orm.jpa.JpaTransactionManager; **import** org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean; **import** org.springframework.transaction.PlatformTransactionManager; **import** org.springframework.transaction.annotation.EnableTransactionManagement;  **import** javax.persistence.EntityManager; **import** javax.sql.DataSource; **import** java.util.Map;  */\*\*  \* Created by Administrator on 2017/10/17.  \** ***@author*** *moa  \*/* @Configuration @EnableTransactionManagement @EnableJpaRepositories(  entityManagerFactoryRef=**"entityManagerFactoryPrimary"**,  transactionManagerRef=**"transactionManagerPrimary"**,  basePackages= { **"${spring.datasource.primary.basePackages}"** }) *//设置Repository所在位置* **public class** PrimaryConfig {  @Autowired  @Qualifier(**"primaryDataSource"**)  **private** DataSource **primaryDataSource**;    *// 设置包* @Value(**"${spring.datasource.primary.basePackages}"**)  **private** String **basePackages**;   @Primary  @Bean(name = **"entityManagerPrimary"**)  **public** EntityManager entityManager(EntityManagerFactoryBuilder builder) {  **return** entityManagerFactoryPrimary(builder).getObject().createEntityManager();  }    @Primary  @Bean(name = **"entityManagerFactoryPrimary"**)  **public** LocalContainerEntityManagerFactoryBean entityManagerFactoryPrimary (EntityManagerFactoryBuilder builder) {  **return** builder  .dataSource(**primaryDataSource**)  .properties(getVendorProperties(**primaryDataSource**))  .packages(**basePackages**) *//设置实体类所在位置* .persistenceUnit(**"primaryPersistenceUnit"**)  .build();  }    @Autowired  **private** JpaProperties **jpaProperties**;   **private** Map<String, String> getVendorProperties(DataSource dataSource) {  **return jpaProperties**.getHibernateProperties(dataSource);  }    *// 获取主数据源的事务管理* @Primary  @Bean(name = **"transactionManagerPrimary"**)  **public** PlatformTransactionManager transactionManagerPrimary(EntityManagerFactoryBuilder builder) {  **return new** JpaTransactionManager(entityManagerFactoryPrimary(builder).getObject());  } } |

|  |
| --- |
| **package** com.moa.datasource.config;  **import** org.springframework.beans.factory.annotation.Qualifier; **import** org.springframework.boot.autoconfigure.jdbc.DataSourceBuilder; **import** org.springframework.boot.context.properties.ConfigurationProperties; **import** org.springframework.context.annotation.Bean; **import** org.springframework.context.annotation.Configuration; **import** org.springframework.context.annotation.Primary;  **import** javax.sql.DataSource;  */\*\*  \* Created by Administrator on 2017/10/17.  \*  \** ***@author*** *moa  \*/* @Configuration **public class** DataSourceConfig {   *// 主数据源* @Bean(name = **"primaryDataSource"**)  @Qualifier(**"primaryDataSource"**)  @Primary *// 设置mysql为默认数据源* @ConfigurationProperties(prefix = **"spring.datasource.primary"**)  **public** DataSource primaryDataSource() {  **return** DataSourceBuilder.*create*().build();  }   *// 第二个数据源* @Bean(name = **"secondaryDataSource"**)  @Qualifier(**"secondaryDataSource"**)  @ConfigurationProperties(prefix = **"spring.datasource.secondary"**)  **public** DataSource secondaryDataSource() {  **return** DataSourceBuilder.*create*().build();  } } |

## 2.7 测试



最后测试结果为： 因为设置了主数据源，在一个方法中对两个数据源操作，事物管理器并没有进行统一管理，默认只有主数据源的事务管理。导致最后只有主数据源user表中的数据进行回滚，所以这个需要修改，通过atomikos来实现JTA.

# JPA多数据源（同步）

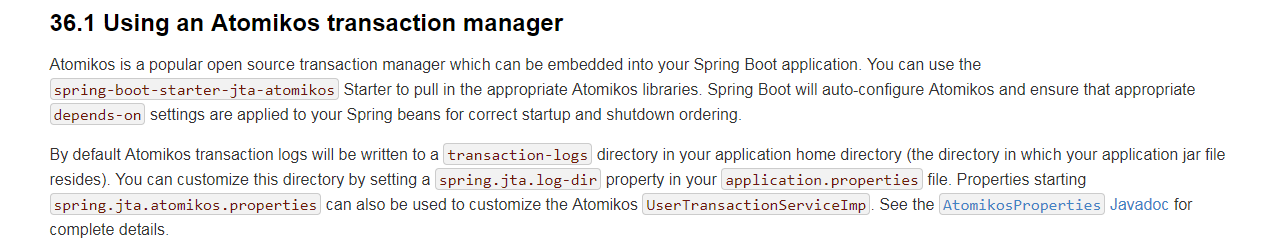
## 3.1 Atomikos

一个为Java平台提供事务服务并且开源类事务管理器。

在pom.xml中加入atomikos依赖。

|  |
| --- |
| *<!-- 添加atomikos 的事物管理-->* <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-jta-atomikos</**artifactId**> </**dependency**> |

spring boot 官网说明

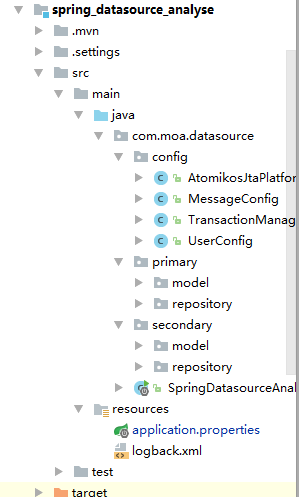


参考：

<https://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/reference/htmlsingle/#boot-features-jta-atomikos>

<http://spring.io/blog/2011/08/15/configuring-spring-and-jta-without-full-java-ee/>

## 3.2 项目结构



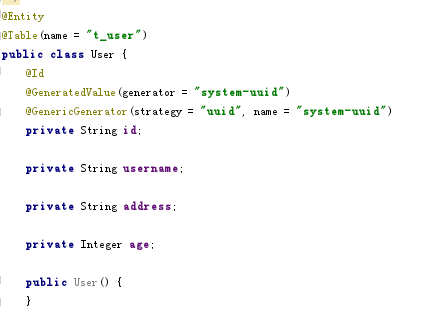
## 3.3 POM.XML

|  |
| --- |
| *<?***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.moa</**groupId**>  <**artifactId**>spring\_datasource\_analyse</**artifactId**>  <**version**>0.0.1-SNAPSHOT</**version**>  <**packaging**>jar</**packaging**>   <**name**>spring\_datasource\_analyse</**name**>  <**description**></**description**>   <**parent**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-parent</**artifactId**>  <**version**>1.5.4.RELEASE</**version**>  <**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**>  </**properties**>   <**dependencies**>  <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter</**artifactId**>  </**dependency**>   <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-test</**artifactId**>  <**scope**>test</**scope**>  </**dependency**>   <**dependency**>  <**groupId**>org.springframework.data</**groupId**>  <**artifactId**>spring-data-jpa</**artifactId**>  </**dependency**>   <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-data-jpa</**artifactId**>  </**dependency**>  *<!--MySQL驱动 ，-->* <**dependency**>  <**groupId**>mysql</**groupId**>  <**artifactId**>mysql-connector-java</**artifactId**>  <**version**>5.1.36</**version**>  </**dependency**>    *<!--oracle 驱动-->* <**dependency**>  <**groupId**>cn.osworks.lib</**groupId**>  <**artifactId**>ojdbc6</**artifactId**>  <**version**>6</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.springframework.data</**groupId**>  <**artifactId**>spring-data-jpa</**artifactId**>  <**version**>1.11.1.RELEASE</**version**>  </**dependency**>   *<!-- postgresql驱动 -->* <**dependency**>  <**groupId**>postgresql</**groupId**>  <**artifactId**>postgresql</**artifactId**>  <**version**>9.1-901.jdbc4</**version**>  </**dependency**>     *<!-- 加入atomokios -->* <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-jta-atomikos</**artifactId**>  </**dependency**>    </**dependencies**>   <**build**>  <**plugins**>  <**plugin**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-maven-plugin</**artifactId**>  </**plugin**>  </**plugins**>  </**build**>   </**project**> |

## 3.4 application.properties

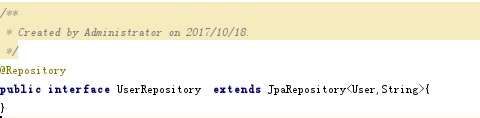
|  |
| --- |
| **spring.jpa.hibernate.ddl-auto**=**update spring.datasource.druid.max-active**=**10 spring.datasource.druid.initial-size**=**5 spring.datasource.druid.filters**=**stat,log4j** *!spring.datasource.druid.poolSize=5* **spring.datasource.druid.primary.url**=**jdbc:mysql://localhost:3306/test2 spring.datasource.druid.primary.username**=**root spring.datasource.druid.primary.password**=**123456****spring.datasource.druid.primary.basePackages**=**com.moa.druid.primary**  *!-- 连接2 mysql* **spring.datasource.druid.secondary.url**=**jdbc:mysql://localhost:3306/test spring.datasource.druid.secondary.username**=**root spring.datasource.druid.secondary.password**=**123456****spring.datasource.druid.secondary.basePackages**=**com.moa.druid.primary** |

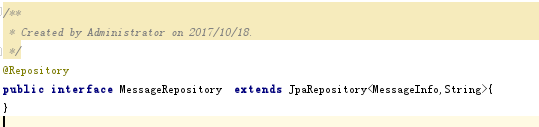
## 3.5 实体



省略getter/setter方法

## 3.6 dao





## 3.7 配置

主数据源：

|  |
| --- |
| package com.moa.druid.config;  /\*\*  \* Created by Administrator on 2017/10/18.  \*/  import com.alibaba.druid.spring.boot.autoconfigure.DruidDataSourceProperties;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.beans.factory.annotation.Qualifier;  import org.springframework.beans.factory.annotation.Value;  import org.springframework.boot.autoconfigure.jdbc.DataSourceProperties;  import org.springframework.boot.context.properties.ConfigurationProperties;  import org.springframework.boot.jta.atomikos.AtomikosDataSourceBean;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.context.annotation.DependsOn;  import org.springframework.context.annotation.Primary;  import org.springframework.core.env.Environment;  import org.springframework.data.jpa.repository.config.EnableJpaRepositories;  import org.springframework.orm.jpa.JpaVendorAdapter;  import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;  import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;  import javax.persistence.PostRemove;  import javax.sql.DataSource;  import java.util.HashMap;  import java.util.Properties;  /\*\*  \* 配置user的 oracle 数据源  \*/  @Configuration  //@DependsOn("transactionManager")  @EnableJpaRepositories(basePackages = "${spring.datasource.druid.primary.basePackages}",  entityManagerFactoryRef = "userEntityManager",  transactionManagerRef = "transactionManager")  public class UserConfig {  @Autowired  private JpaVendorAdapter jpaVendorAdapter;  // @Value("${spring.datasource.druid.primary.platform}")  // private String platform;  // 设置包  @Value("${spring.datasource.druid.primary.basePackages}")  private String basePackages;  @ConfigurationProperties("spring.datasource.druid.primary")  @Bean  @Primary  public DruidDataSourceProperties myPrimaryDruidDatasourceProperties(){  return new MyDruidDatasourceProperties();  }  @Bean(name = "primaryDataSource")  @Primary  @Autowired  public DataSource primaryDataSource(Environment env) {  AtomikosDataSourceBean ds = new AtomikosDataSourceBean();  MyDruidDatasourceProperties druidDataSourceProperties = (MyDruidDatasourceProperties)myPrimaryDruidDatasourceProperties();  // 设置驱动数据源  ds.setXaDataSourceClassName("com.alibaba.druid.pool.xa.DruidXADataSource");  // 设置名  ds.setUniqueResourceName("primaryDataSource");  ds.setPoolSize(5);  ds.setXaProperties(druidDataSourceProperties.getProperties());  return ds;  }  @Bean  @ConfigurationProperties("spring.datasource.druid")  public JpaVendorAdapter primaryJpaVendorAdapter(){  return new HibernateJpaVendorAdapter();  }  @Bean(name = "userEntityManager")  @DependsOn("transactionManager")  @Primary  @Autowired  public LocalContainerEntityManagerFactoryBean userEntityManager(Environment env) throws Throwable {  HashMap<String, Object> properties = new HashMap<String, Object>();  properties.put("hibernate.transaction.jta.platform", AtomikosJtaPlatform.class.getName());  properties.put("hibernate.format\_sql", "true");  properties.put("hibernate.hbm2ddl.auto", "update");  LocalContainerEntityManagerFactoryBean entityManager = new LocalContainerEntityManagerFactoryBean();  //设置数据源  entityManager.setJtaDataSource(primaryDataSource(env));  // HibernateJpaVendorAdapter hibernateJpaVendorAdapter = (HibernateJpaVendorAdapter) jpaVendorAdapter;  // hibernateJpaVendorAdapter.setDatabasePlatform(platform);  // 默认方言为mysql  entityManager.setJpaVendorAdapter(jpaVendorAdapter);  entityManager.setPackagesToScan(basePackages);  // 创建持久单元名称， 唯一  entityManager.setPersistenceUnitName("primaryPersistenceUnit");    entityManager.setJpaPropertyMap(properties);  return entityManager;  }  } |

第二个数据源：

|  |
| --- |
| package com.moa.druid.config;  /\*\*  \* Created by Administrator on 2017/10/18.  \*/  import com.alibaba.druid.spring.boot.autoconfigure.DruidDataSourceProperties;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.beans.factory.annotation.Value;  import org.springframework.boot.context.properties.ConfigurationProperties;  import org.springframework.boot.jta.atomikos.AtomikosDataSourceBean;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.context.annotation.DependsOn;  import org.springframework.context.annotation.Primary;  import org.springframework.core.env.Environment;  import org.springframework.data.jpa.repository.config.EnableJpaRepositories;  import org.springframework.orm.jpa.JpaVendorAdapter;  import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;  import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;  import javax.sql.DataSource;  import java.util.HashMap;  import java.util.Properties;  /\*\*  \* 配置mssage的mysql数据源  \*/  @Configuration  //@DependsOn("transactionManager")  @EnableJpaRepositories(basePackages = "com.moa.druid.secondary",  entityManagerFactoryRef = "messageEntityManager",  transactionManagerRef = "transactionManager")  public class MessageConfig {  // 设置包  @Value("${spring.datasource.druid.secondary.basePackages}")  private String basePackages;  // 设置方言  // @Value("${spring.datasource.druid.secondary.platform}")  // private String platform;  @Bean  @ConfigurationProperties("spring.datasource.druid.secondary")  public DruidDataSourceProperties myDruidDatasourceProperties() {  return new MyDruidDatasourceProperties();  }  @Autowired  private JpaVendorAdapter jpaVendorAdapter;  @Bean(name = "secondaryDataSource")  @Autowired  public DataSource secondaryDataSource(Environment env) {  AtomikosDataSourceBean ds = new AtomikosDataSourceBean();  // 获取设置  MyDruidDatasourceProperties druidDataSourceProperties = (MyDruidDatasourceProperties) myDruidDatasourceProperties();  // 设置驱动数据源  ds.setXaDataSourceClassName("com.alibaba.druid.pool.xa.DruidXADataSource");  // 设置唯一名  ds.setUniqueResourceName("secondaryDataSource");  //　设置池大小　－－－　加入设置  ds.setPoolSize(5);  ds.setXaProperties(druidDataSourceProperties.getProperties());  return ds;  }  @Bean(name = "messageEntityManager")  @DependsOn("transactionManager")  @Autowired  public LocalContainerEntityManagerFactoryBean messageEntityManager(Environment env) throws Throwable {  HashMap<String, Object> properties = new HashMap<String, Object>();  // 这个一定需要，否则事务管理不起作用  properties.put("hibernate.transaction.jta.platform", AtomikosJtaPlatform.class.getName());  // 格式化  // properties.put("hibernate.format\_sql", "true");  // properties.put("hibernate.hbm2ddl.auto", "update");  LocalContainerEntityManagerFactoryBean entityManager = new LocalContainerEntityManagerFactoryBean();  //设置数据源  entityManager.setJtaDataSource(secondaryDataSource(env));  // HibernateJpaVendorAdapter hibernateJpaVendorAdapter = (HibernateJpaVendorAdapter) jpaVendorAdapter;  // 设置方言  //hibernateJpaVendorAdapter.setDatabasePlatform(platform);  entityManager.setJpaVendorAdapter(jpaVendorAdapter);  // 设置entity的包位置  entityManager.setPackagesToScan(basePackages);  // 创建持久单元名称， 唯一  entityManager.setPersistenceUnitName("secondarrPersistenceUnit");    entityManager.setJpaPropertyMap(properties);  return entityManager;  }  } |

事物管理器类

|  |
| --- |
| **package** com.moa.datasource.config;  **import** com.atomikos.icatch.jta.UserTransactionImp; **import** com.atomikos.icatch.jta.UserTransactionManager; **import** org.springframework.context.annotation.Bean; **import** org.springframework.context.annotation.ComponentScan; **import** org.springframework.context.annotation.Configuration; **import** org.springframework.context.annotation.DependsOn; **import** org.springframework.context.support.PropertySourcesPlaceholderConfigurer; **import** org.springframework.orm.jpa.JpaVendorAdapter; **import** org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter; **import** org.springframework.transaction.PlatformTransactionManager; **import** org.springframework.transaction.annotation.EnableTransactionManagement; **import** org.springframework.transaction.jta.JtaTransactionManager;  **import** javax.transaction.TransactionManager; **import** javax.transaction.UserTransaction;  */\*\*  \* Created by Administrator on 2017/10/17.  \*/* @Configuration @ComponentScan @EnableTransactionManagement **public class** TransactionManagerConfig {   */\*\*  \* 需要测试这个是做什么  \** ***@return*** *\*/  /\* @Bean  public PropertySourcesPlaceholderConfigurer propertySourcesPlaceholderConfigurer() {  return new PropertySourcesPlaceholderConfigurer();  }\*/    /\*\*  \* 创建atomkiosbean  \** ***@return*** *\** ***@throws*** *Throwable  \*/* @Bean(name = **"userTransaction"**)  **public** UserTransaction userTransaction() **throws** Throwable {  UserTransactionImp userTransactionImp = **new** UserTransactionImp();  userTransactionImp.setTransactionTimeout(10000);  **return** userTransactionImp;  }    @Bean(name = **"atomikosTransactionManager"**, initMethod = **"init"**, destroyMethod = **"close"**)  **public** TransactionManager atomikosTransactionManager() **throws** Throwable {  UserTransactionManager userTransactionManager = **new** UserTransactionManager();  userTransactionManager.setForceShutdown(**false**);   AtomikosJtaPlatform.*transactionManager* = userTransactionManager;   **return** userTransactionManager;  }   @Bean(name = **"transactionManager"**)  @DependsOn({ **"userTransaction"**, **"atomikosTransactionManager"** })  **public** PlatformTransactionManager transactionManager() **throws** Throwable {  UserTransaction userTransaction = userTransaction();  AtomikosJtaPlatform.*transaction* = userTransaction;  TransactionManager atomikosTransactionManager = atomikosTransactionManager();  JtaTransactionManager jtaTransactionManager = **new** JtaTransactionManager(userTransaction, atomikosTransactionManager);  jtaTransactionManager.setAllowCustomIsolationLevels(**true**);  **return** jtaTransactionManager;  }   */\*\*  \* 在这里创建jpaVendor 后者用调用只需要更新方言  \* 默认创建mysql方言  \*/* @Bean  **public** JpaVendorAdapter jpaVendorAdapter(){  HibernateJpaVendorAdapter hibernateJpaVendorAdapter = **new** HibernateJpaVendorAdapter();  *// 默认开启打印SQL* hibernateJpaVendorAdapter.setShowSql(**true**);  hibernateJpaVendorAdapter.setGenerateDdl(**true**);  hibernateJpaVendorAdapter.setDatabasePlatform(**"org.hibernate.dialect.MySQL5Dialect"**);  **return** hibernateJpaVendorAdapter;   } } |

|  |
| --- |
| **package** com.moa.datasource.config;  **import** org.hibernate.engine.transaction.jta.platform.internal.AbstractJtaPlatform;  **import** javax.transaction.TransactionManager; **import** javax.transaction.UserTransaction;   *// 管理事务* **public class** AtomikosJtaPlatform **extends** AbstractJtaPlatform {   **private static final long *serialVersionUID*** = 1L;   **static** TransactionManager transactionManager;  **static** UserTransaction *transaction*;   @Override  **protected** TransactionManager locateTransactionManager() {  **return** transactionManager;  }   @Override  **protected** UserTransaction locateUserTransaction() {  **return** *transaction*;  } } |

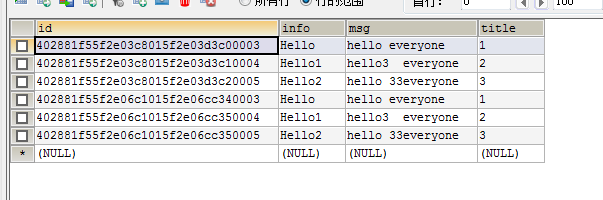
## 3.8 测试

|  |
| --- |
| @RunWith(SpringRunner.**class**) @SpringBootTest **public class** SpringDatasourceAnalyseApplicationTests {   @Autowired  **private** MessageRepository **messageRepository**;   @Autowired  **private** UserRepository **userRepository**;   @Test  @Transactional  @Rollback(**false**)  **public void** contextLoads() {  **userRepository**.save(**new** User(**"张三"**,**"Hello atomikos"**,1));  **userRepository**.save(**new** User(**"李四"**,**"Hello atomikos"**,6));  **userRepository**.save(**new** User(**"王武"**,**"Hello atomikos"**,4));    **messageRepository**.save(**new** MessageInfo(**"1"**,**"Hello"**,**"hello everyone"**));  **messageRepository**.save(**new** MessageInfo(**"2"**,**"Hello1"**,**"hello3 everyone"**));  **messageRepository**.save(**new** MessageInfo(**"3"**,**"Hello2"**,**"hello 33everyone"**));  }  } |

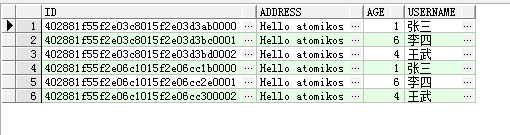
测试三次：两次正常，一次失败

结果：

mysql:



oracle:



UserTrsaction:

TransactionManager:

# 4 DRUID+ATOMIKOS

## 4.1 加入druid依赖

|  |
| --- |
| *<!-- 配置 alibaba 的druid -->* <**dependency**>  <**groupId**>com.alibaba</**groupId**>  <**artifactId**>druid</**artifactId**>  <**version**>1.1.4</**version**> </**dependency**>   <**dependency**>  <**groupId**>com.alibaba</**groupId**>  <**artifactId**>druid-spring-boot-starter</**artifactId**>  <**version**>1.1.1</**version**> </**dependency**>  *<!-- 配置 alibaba 的druid结束-->* |

## 4.2 修改代码

新增类：

用于获取连接池设置

|  |
| --- |
| **package** com.moa.druid.config;  **import** com.alibaba.druid.pool.DruidDataSource; **import** com.alibaba.druid.spring.boot.autoconfigure.DruidDataSourceProperties;  **import** java.sql.SQLException; **import** java.util.Properties;  */\*\*  \* Created by Administrator on 2017/10/18.  \*/* **public class** MyDruidDatasourceProperties **extends** DruidDataSourceProperties {   */\*\*  \* 获取所有属性的properties  \** ***@return*** *\*/* **public** Properties getProperties(){  Properties prop = **new** Properties();  **if** (getUrl() != **null**) {  prop.put(**"url"**,getUrl() );  }  **if** (getUsername() != **null**) {  prop.put(**"username"**,getUsername());  }  **if** (getPassword() != **null**) {  prop.put(**"password"**,getPassword());  }  **if** (getDriverClassName() != **null**) {  prop.put(**"driverClassName"**,getDriverClassName());  }  **if** (getInitialSize() != **null**) {  prop.put(**"initialSize"**, getInitialSize());  }  **if** (getMaxActive() != **null**) {  prop.put(**"maxActive"**, getMaxActive());  }  **if** (getMinIdle() != **null**) {  prop.put(**"minIdle"**,getMinIdle());  }  **if** (getMaxWait() != **null**) {  prop.put(**"maxWait"**,getMaxWait());  }  **if** (getPoolPreparedStatements() != **null**) {  prop.put(**"poolPreparedStatements"**, getPoolPreparedStatements());  }  **if** (getMaxOpenPreparedStatements() != **null**) {  prop.put(**"maxPoolPreparedStatementPerConnectionSize"**,getMaxOpenPreparedStatements());  }  **if** (getMaxPoolPreparedStatementPerConnectionSize() != **null**) {  prop.put(**"maxPoolPreparedStatementPerConnectionSize"**,getMaxPoolPreparedStatementPerConnectionSize());  }  **if** (getValidationQuery() != **null**) {  prop.put(**"validationQuery"**,getValidationQuery());  }  **if** (getValidationQueryTimeout() != **null**) {  prop.put(**"validationQueryTimeout"**,getValidationQueryTimeout());  }  **if** (getTestWhileIdle() != **null**) {  prop.put(**"testWhileIdle"**,getTestWhileIdle());  }  **if** (getTestOnBorrow() != **null**) {  prop.put(**"testOnBorrow"**, getTestOnBorrow());  }  **if** (getTestOnReturn() != **null**) {  prop.put(**"testOnReturn"**,getTestOnReturn());  }  **if** (getTimeBetweenEvictionRunsMillis() != **null**) {  prop.put(**"timeBetweenEvictionRunsMillis"**,getTimeBetweenEvictionRunsMillis());  }  **if** (getMinEvictableIdleTimeMillis() != **null**) {  prop.put(**"MinEvictableIdleTimeMillis"**,getMinEvictableIdleTimeMillis());  }  **if** (getMaxEvictableIdleTimeMillis() != **null**) {  prop.put(**"maxEvictableIdleTimeMillis"**,getMaxEvictableIdleTimeMillis());  }  **try** {  **if** (getFilters() != **null**) {  prop.put(**"filters"**,getFilters());  }  } **catch** (Exception e) {  e.printStackTrace();  }  **return** prop;  } } |

修改获取数据源方式

|  |
| --- |
| **package** com.moa.druid.config;  */\*\*  \* Created by Administrator on 2017/10/18.  \*/* **import** com.alibaba.druid.spring.boot.autoconfigure.DruidDataSourceProperties; **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.beans.factory.annotation.Value; **import** org.springframework.boot.context.properties.ConfigurationProperties; **import** org.springframework.boot.jta.atomikos.AtomikosDataSourceBean; **import** org.springframework.context.annotation.Bean; **import** org.springframework.context.annotation.Configuration; **import** org.springframework.context.annotation.DependsOn; **import** org.springframework.context.annotation.Primary; **import** org.springframework.core.env.Environment; **import** org.springframework.data.jpa.repository.config.EnableJpaRepositories; **import** org.springframework.orm.jpa.JpaVendorAdapter; **import** org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean; **import** org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;  **import** javax.sql.DataSource; **import** java.util.HashMap; **import** java.util.Properties;  */\*\*  \* 配置mssage的mysql数据源  \*/* @Configuration @DependsOn(**"transactionManager"**) @EnableJpaRepositories(basePackages = **"${spring.datasource.druid.secondary.basePackages}"**,  entityManagerFactoryRef = **"messageEntityManager"**,  transactionManagerRef = **"transactionManager"**) **public class** MessageConfig {   *// 设置包* @Value(**"${spring.datasource.druid.secondary.basePackages}"**)  **private** String **basePackages**;    @Value(**"${spring.datasource.druid.secondary.platform}"**)  **private** String **platform**;   @Bean  @ConfigurationProperties(**"spring.datasource.druid.secondary"**)  **public** DruidDataSourceProperties myDruidDatasourceProperties() {  **return new** MyDruidDatasourceProperties();  }   @Autowired  **private** JpaVendorAdapter **jpaVendorAdapter**;   @Bean(name = **"secondaryDataSource"**)  @Autowired  **public** DataSource secondaryDataSource(Environment env) {  AtomikosDataSourceBean ds = **new** AtomikosDataSourceBean();  *// 获取设置* MyDruidDatasourceProperties druidDataSourceProperties = (MyDruidDatasourceProperties) myDruidDatasourceProperties();  *// 设置驱动数据源* ds.setXaDataSourceClassName(**"com.alibaba.druid.pool.xa.DruidXADataSource"**);  *// 设置唯一名* ds.setUniqueResourceName(**"secondaryDataSource"**);  *//　设置池大小　－－－　加入设置* ds.setPoolSize(5);  ds.setXaProperties(druidDataSourceProperties.getProperties());  **return** ds;   }    @Bean(name = **"messageEntityManager"**)  @DependsOn(**"transactionManager"**)  @Autowired  **public** LocalContainerEntityManagerFactoryBean messageEntityManager(Environment env) **throws** Throwable {  HashMap<String, Object> properties = **new** HashMap<String, Object>();  *// 这个一定需要，否则事务管理不起作用* properties.put(**"hibernate.transaction.jta.platform"**, AtomikosJtaPlatform.**class**.getName());  *// 格式化* properties.put(**"hibernate.format\_sql"**, **"true"**);  LocalContainerEntityManagerFactoryBean entityManager = **new** LocalContainerEntityManagerFactoryBean();   *//设置数据源* entityManager.setJtaDataSource(secondaryDataSource(env));  HibernateJpaVendorAdapter hibernateJpaVendorAdapter = (HibernateJpaVendorAdapter) **jpaVendorAdapter**;  hibernateJpaVendorAdapter.setDatabasePlatform(**platform**);  entityManager.setJpaVendorAdapter(**jpaVendorAdapter**);  entityManager.setPackagesToScan(**basePackages**);   *// 创建持久单元名称， 唯一* entityManager.setPersistenceUnitName(**"secondarrPersistenceUnit"**);  *// 这个设置* entityManager.setJpaPropertyMap(properties);   **return** entityManager;  }  } |

|  |
| --- |
| **package** com.moa.druid.config;  */\*\*  \* Created by Administrator on 2017/10/18.  \*/* **import** com.alibaba.druid.spring.boot.autoconfigure.DruidDataSourceProperties; **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.beans.factory.annotation.Qualifier; **import** org.springframework.beans.factory.annotation.Value; **import** org.springframework.boot.autoconfigure.jdbc.DataSourceProperties; **import** org.springframework.boot.context.properties.ConfigurationProperties; **import** org.springframework.boot.jta.atomikos.AtomikosDataSourceBean; **import** org.springframework.context.annotation.Bean; **import** org.springframework.context.annotation.Configuration; **import** org.springframework.context.annotation.DependsOn; **import** org.springframework.context.annotation.Primary; **import** org.springframework.core.env.Environment; **import** org.springframework.data.jpa.repository.config.EnableJpaRepositories; **import** org.springframework.orm.jpa.JpaVendorAdapter; **import** org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean; **import** org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;  **import** javax.persistence.PostRemove; **import** javax.sql.DataSource; **import** java.util.HashMap; **import** java.util.Properties;  */\*\*  \* 配置user的 oracle 数据源  \*/* @Configuration @DependsOn(**"transactionManager"**) @EnableJpaRepositories(basePackages = **"${spring.datasource.druid.primary.basePackages}"**,  entityManagerFactoryRef = **"userEntityManager"**,  transactionManagerRef = **"transactionManager"**) **public class** UserConfig {     @Autowired  **private** JpaVendorAdapter **jpaVendorAdapter**;    @Value(**"${spring.datasource.druid.primary.platform}"**)  **private** String **platform**;    *// 设置包* @Value(**"${spring.datasource.druid.primary.basePackages}"**)  **private** String **basePackages**;   @ConfigurationProperties(**"spring.datasource.druid.primary"**)  @Bean  @Primary  **public** DruidDataSourceProperties myPrimaryDruidDatasourceProperties(){  **return new** MyDruidDatasourceProperties();  }   @Bean(name = **"primaryDataSource"**)  @Primary  @Autowired  **public** DataSource primaryDataSource(Environment env) {  AtomikosDataSourceBean ds = **new** AtomikosDataSourceBean();  MyDruidDatasourceProperties druidDataSourceProperties = (MyDruidDatasourceProperties)myPrimaryDruidDatasourceProperties();  *// 设置驱动数据源* ds.setXaDataSourceClassName(**"com.alibaba.druid.pool.xa.DruidXADataSource"**);  *// 设置名* ds.setUniqueResourceName(**"primaryDataSource"**);  ds.setPoolSize(5);  ds.setXaProperties(druidDataSourceProperties.getProperties());  **return** ds;   }   @Bean   @ConfigurationProperties(**"spring.datasource.druid"**)  **public** JpaVendorAdapter primaryJpaVendorAdapter(){  **return new** HibernateJpaVendorAdapter();   }    @Bean(name = **"userEntityManager"**)  @DependsOn(**"transactionManager"**)  @Primary  @Autowired  **public** LocalContainerEntityManagerFactoryBean userEntityManager(Environment env) **throws** Throwable {   HashMap<String, Object> properties = **new** HashMap<String, Object>();  properties.put(**"hibernate.transaction.jta.platform"**, AtomikosJtaPlatform.**class**.getName());  properties.put(**"hibernate.format\_sql"**, **"true"**);   LocalContainerEntityManagerFactoryBean entityManager = **new** LocalContainerEntityManagerFactoryBean();  *//设置数据源* entityManager.setJtaDataSource(primaryDataSource(env));  HibernateJpaVendorAdapter hibernateJpaVendorAdapter = (HibernateJpaVendorAdapter) **jpaVendorAdapter**;  hibernateJpaVendorAdapter.setDatabasePlatform(**platform**);   *// 默认方言为mysql* entityManager.setJpaVendorAdapter(**jpaVendorAdapter**);  entityManager.setPackagesToScan(**basePackages**);  *// 创建持久单元名称， 唯一* entityManager.setPersistenceUnitName(**"primaryPersistenceUnit"**);  *// 这个设置一定需要 ，为什么* entityManager.setJpaPropertyMap(properties);   **return** entityManager;  } } |

其余保持不变，然后运行测试单元，结果与3一致。

4.3 日志记录

配置项：

spring.datasource.druid.filters=stat,log4j

在使用druid数据连接池的时候，希望能清楚的看到一些打印日志信息





druid的一些常见问题

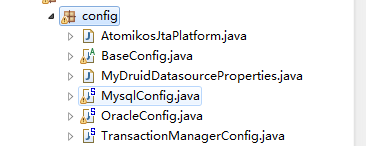
<https://github.com/alibaba/druid/wiki/%E5%B8%B8%E8%A7%81%E9%97%AE%E9%A2%98>

druid git地址：

<https://github.com/alibaba/druid>

# 5 多数据源配置（事务同步）

## 5.1 主要配置类截图



说明：

AtomikosJtaPlatform.java 实现spring事务管理器与hibernate的事务管理器一致

BaseConfig.java 基本配置类，所有的数据源配置类需继承此类

MyDruidDatasourceProperties.java 连接池的属性接收类

TransactionManagerConfig.java 事物管理器配置类

## 5.2 application.properties

|  |
| --- |
| spring.jpa.hibernate.ddl-auto=update  spring.datasource.druid.max-active=10  spring.datasource.druid.initialSize=5  spring.datasource.druid.filters=stat,log4j    #配置连接池的class路径  spring.datasource.druid.XaDataSourceClassName=com.alibaba.druid.pool.xa.DruidXADataSource    #配置数据源  spring.datasource.druid.primary.url=jdbc:mysql://localhost:3306/test2  spring.datasource.druid.primary.username=root  spring.datasource.druid.primary.password=123456  !—实体包，不配置默认从com.xk开始  spring.datasource.druid.primary.entityBasePackages=com.moa.druid.primary.model  !—dao包， spring.datasource.druid.primary.jpaBasePackages=com.moa.druid.primary.repository  !—唯一的名称，用来区分各个数据源，必须配置  spring.datasource.druid.primary.uniqueName=primaryUnite    !—第二个数据源  spring.datasource.druid.secondary.url=jdbc:mysql://localhost:3306/test  spring.datasource.druid.secondary.username=root  spring.datasource.druid.secondary.password=123456  !—实体包，不配置默认从com.xk开始  spring.datasource.druid.secondary.entityBasePackages=com.moa.druid.secondary  !—dao包  spring.datasource.druid.secondary.jpaBasePackages=com.moa.druid.secondary.repository  !—唯一的名称，用来区分各个数据源，必须配置  spring.datasource.druid.secondary.uniqueName=secondaryUnite |

## 5.3 POM.xml

|  |
| --- |
| <dependency>  <groupId>org.springframework.data</groupId>  <artifactId>spring-data-jpa</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency>  <!--MySQL驱动 ，-->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  <!--oracle 驱动-->  <dependency>  <groupId>cn.osworks.lib</groupId>  <artifactId>ojdbc6</artifactId>  <version>6</version>  </dependency>  <!-- 加入atomokios -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-jta-atomikos</artifactId>  </dependency>  <!-- 配置 alibaba 的druid -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.1.4</version>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid-spring-boot-starter</artifactId>  <version>1.1.1</version>  </dependency>  <!-- 配置 alibaba 的druid结束--> |

## 5.4 新增数据源

第一：在application.properties中加入数据源配置

第二：新增配置类，该配置类继承BaseConfig,实现并重写里面的方法

createDataSource(uniqueResourceName);

createEntityManager(env, uniqueResouceName);

这两个方法必须重写,在方法内部调用父类方法，然后对返回结果操作。

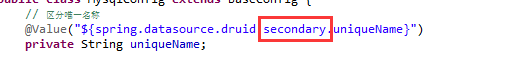
例如：

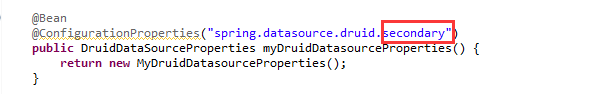
|  |
| --- |
| /\*\*  \* 配置mysql数据源  \*/  @Configuration  @EnableJpaRepositories(basePackages = "${spring.datasource.druid.secondary.jpaBasePackages}", //  entityManagerFactoryRef = "messageEntityManager")  **public** **class** MysqlConfig **extends** BaseConfig {  //  @Value("${spring.datasource.druid.secondary.uniqueName}")  **private** String uniqueName;  @Bean  @ConfigurationProperties("spring.datasource.druid.secondary")  **public** DruidDataSourceProperties myDruidDatasourceProperties() {  **return** **new** MyDruidDatasourceProperties();  }  @Bean(name = "secondaryDataSource")  **public** DataSource createDataSource() {  **return** **super**.createDataSource(uniqueName);  }  @Bean(name = "messageEntityManager")  @DependsOn("transactionManager")  @Autowired  **public** LocalContainerEntityManagerFactoryBean messageEntityManager(Environment env) **throws** Throwable {  LocalContainerEntityManagerFactoryBean entityManager = **super**.createEntityManager(env, uniqueName);  **return** entityManager;  }  } |

修改地方：

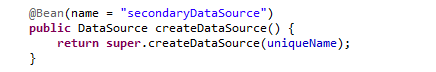
1. 

@ EnableJpaRepositories注解中的basePackages 里面的表达式替换成对应新增的数据源的jpaBasePackages属性。

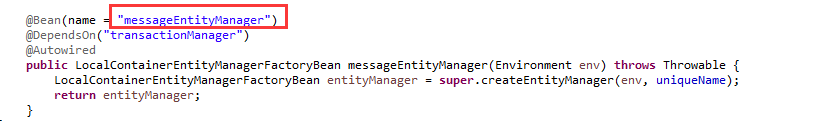
@Value红色部分替换成对应的数据源配置

1. 

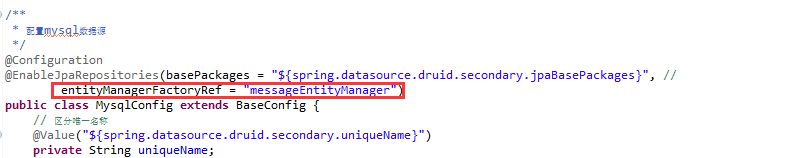
@ConfigurationProperties红色部分替换成对应的数据源配置

1. 

创建数据源bean的名称必须不一致，默认取名规则为当前是第几个数据源。例如：第二个数据源（secondaryDataSource）

1. 

1 创建实体管理工厂bean的名称必须不一致，默认取名规则为当前是第几个实体管理工厂，例如第二个实体管理工厂（secondaryEntityManager）

2 

该名称必须与类名称注解上@EnableJpaRepositories里面的entityManagerFactoryRef里面的值一致。