Atitit spring注解事务的demo与代码说明

目录

[1.1. Spring框架中，要如何实现事务？有一个注解，@EnableTransactionManagement 1](#_Toc326)

[1.2. 事务管理 99.99999%都是使用了xml来配置的 1](#_Toc31985)

[1.3. 开启事务注解驱动@EnableTransactionManagement 2](#_Toc8318)

[1.4. 开启aop @EnableAspectJAutoProxy------- 3](#_Toc12826)

[1.5. 配置事务管理器 AppConfig 3](#_Toc2223)

[1.6. 配置数据源/AtiDateSource.java 4](#_Toc8742)

[1.7. Userservice 和uesrdao 4](#_Toc32723)

[1.8. 调用spring4demoApplication 5](#_Toc9094)

[2. Other 6](#_Toc4462)

[2.1. @Transactional所需要的jar包 6](#_Toc20256)

[2.2. 二、使用@Transactional 7](#_Toc2171)

[3. Ref 8](#_Toc11017)

## Spring框架中，要如何实现事务？有一个注解，@EnableTransactionManagement

传统Spring框架的事务配置都是在XML配置文件中配置的，指定数据源，事务管理器，切入点等等。

那在零配置下的Spring框架中，要如何实现事务？有一个注解，@EnableTransactionManagement，这个注解能实现对标识了@Transactional注解的类或者方法环绕事务。

---------------------

## 事务管理 99.99999%都是使用了xml来配置的

但由于Spring4实战里并没有讲有关事务管理这方面的内容，而网上的教程99.99999%都是使用了xml来配置的，但由于个人更倾向于完全基于Java的配置，所以只能自己想办法解决。

<!-- 事务管理器，对mybatis操作数据库进行事务控制，spring使用jdbc的事务控制类  -->

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dataSourceSuport"></property>

</bean>

<!—开启spring 事务注解支持  mode="aspectj"表示采用切面    mode="proxy"表示代理模式(默认) -->

     <tx:annotation-driven transaction-manager="transactionManager"  />

MyBatis自动参与到spring事务管理中，无需额外配置，只要org.mybatis.spring.SqlSessionFactoryBean引用的数据源与DataSourceTransactionManager引用的数据源一致即可，否则事务管理会不起作用。

@Transactional可以作用于接口、接口方法、类以及类方法上。当作用于类上时，该类的所有 public 方法将都具有该类型的事务属性，同时，我们也可以在方法级别使用该标注来覆盖类级别的定义。 虽然@Transactional 注解可以作用于接口、接口方法、类以及类方法上，但是 Spring 建议不要在接口或者接口方法上使用该注解，因为这只有在使用基于接口的代理时它才会生效。另外，@Transactional 注解应该只被应用到public 方法上，这是由 Spring AOP 的本质决定的。如果你在 protected、private 或者默认可见性的方法上使用@Transactional 注解，这将被忽略，也不会抛出任何异常。默认情况下，只有来自外部的方法调用才会被AOP代理捕获，也就是，类内部方法调用本类内部的其他方法并不会引起事务行为，即使被调用方法使用@Transactional注解进行修饰。

---------------------

零配置下的Spring框架中，要如何实现事务？有一个注解，@EnableTransactionManagement，这个注解能实现对标识了@Transactional注解的类或者方法环绕事务。但是我们使用事务配置的最好的方式是不希望对业务代码上添加额外的东西，或者说事务的相关代码不要和业务的代码有所关联，所以使用@EnableTransactionManagement注解的形式也不可取。

## 开启事务注解驱动@EnableTransactionManagement

<!-- 事务注解驱动，标注@Transactional的类和方法将具有事务性 -->

<tx:annotation-driven transaction-manager="txManager" />

1

2

但有没有方法可以代替这句话，我找了很久，终于发现在知乎中也有相关问题tx:annotation-driven用什么注解代替

---------------------

用@EnableTransactionManagement来 启用注解式事务管理，其效果等同于上面的xml配置中的< tx:annotation-driven/>，但此时还不能起效果，因为

还要找个东西来代替transaction-manager属性，官方文档也给出了解决方法，就是让配置类实现TransactionManagementConfigurer接口，它会覆盖一个方法，（return那句是我自己加的）

@Override

public PlatformTransactionManager annotationDrivenTransactionManager() {

return new DataSourceTransactionManager(dataSource());

}

## 开启aop @EnableAspectJAutoProxy-------

## **配置事务管理器 AppConfig**

**package** iocSpring5demo;

**import** javax.sql.DataSource;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.ComponentScan;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.context.annotation.EnableAspectJAutoProxy;

**import** org.springframework.jdbc.core.JdbcTemplate;

**import** org.springframework.jdbc.datasource.DataSourceTransactionManager;

**import** org.springframework.transaction.PlatformTransactionManager;

**import** org.springframework.transaction.annotation.TransactionManagementConfigurer;

***@EnableAspectJAutoProxy***

***@Configuration*** /\*\* 该注解表示这个类是一个Spring的配置类 \*\*/

***@ComponentScan***(basePackages = {

"iocSpring5demo" }) /\*\*\*

\* 该注解表示启用spring的组件扫描功能，并且配置了扫描包net.xqlee.project.

\* demo下的所有类

\*\*/

**public** **class** **AppConfig** **implements** TransactionManagementConfigurer {

***@Bean***

**public** **JdbcTemplate** **jdbcTemplate**() {

**return** **new** JdbcTemplate(dataSource());

}

***@Bean***

**public** DataSource **dataSource**() {

**AtiDateSource** **instance** = **new** AtiDateSource();

**return** instance;

// configure and return the necessary JDBC DataSource

}

// @Bean

// public PlatformTransactionManager txManager() {

// return new DataSourceTransactionManager(dataSource());

// }

***@Override***

**public** PlatformTransactionManager **annotationDrivenTransactionManager**() {

**return** **new** DataSourceTransactionManager(dataSource());

}

## 配置数据源/AtiDateSource.java

/iocSpring52Prj/src/iocSpring5demo/AtiDateSource.java

## Userservice 和uesrdao

**package** iocSpring5demo;

**import** javax.annotation.Resource;

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

**import** org.springframework.jdbc.core.JdbcTemplate;

**import** org.springframework.stereotype.Component;

**import** org.springframework.transaction.annotation.EnableTransactionManagement;

**import** org.springframework.transaction.annotation.Transactional;

**import** org.springframework.transaction.support.TransactionTemplate;

***@Component*** /\*\* 注册为spring的组件bean \*\*/

***@EnableTransactionManagement***

***@Transactional***

**public** **class** **UserService** {

***@Autowired***

**public** **JdbcTemplate** template;

***@Autowired***

**public** **userDao** userDao1;

// @Autowired

// TransactionTemplate transactionTemplate;

***@Transactional***

**public** **String** **addUser**(**String** name) {

userDao1.addUser1(name+"\_bydataService1\_adduser1" );

userDao1. addUser2(name+"\_bydataService1\_adduse2" );

**return** **null**;

}

// public String addUser1(String string) {

// String sql="insert user(name)values('@u@') ";

// sql=sql.replaceAll("@u@", string);

// int r= template.update(sql);

// return "hhh";

// }

//

// public void addUser2(String string) {

// String sql="insert user(name)values('@u@') a";

// sql=sql.replaceAll("@u@", string);

// int r= template.update(sql);

//

// }

}

## 调用spring4demoApplication

**import** org.springframework.context.annotation.Bean;

/\*\*

\* spring 5程序启动类

\*

\* **@author** xqlee

\*

\*/

**public** **class** **spring4demoApplication** {

/\*\* spring 依赖注入用户测试类 \*\*/

**public** **static** **void** **main**(**String**[] args) {

// com.mysql.jdbc.Driver

// org.springframework.transaction.PlatformTransactionManager

// 创建spring 基于注解配置的容器

**AnnotationConfigApplicationContext** **context** = **new** AnnotationConfigApplicationContext(**AppConfig**.**class**);

// 获取通过注解注入容器的UserService

**UserService** **userService** = context.getBean(**UserService**.**class**);

// 调用userService的方法执行

**String** **message** = userService.addUser("Leftso");

// userService.addUser("user2");

// 输出结果

**System**.***out***.println(message);

// 关闭容器,释放JVM资源

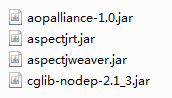
context.close();

}

}

# Other

## **@Transactional所需要的jar包**



1、aopalliance.jar  这个包是[AOP](https://www.baidu.com/s?wd=AOP&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9rADznyNBryD1nW64nyPh0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPH03PHT4nHfs" \t "https://www.cnblogs.com/guoqixing/p/_blank)联盟的[API](https://www.baidu.com/s?wd=API&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9rADznyNBryD1nW64nyPh0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPH03PHT4nHfs" \t "https://www.cnblogs.com/guoqixing/p/_blank)包，里面包含了针对面向切面的接口。(通常Spring等其它具备动态织入功能的框架依赖此包)

2、aspectjrt.jar　　　　     处理事务和AOP所需的包

3、aspectjweaver.jar        处理事务和AOP所需的包

4、cglib-nodep.jar　　　    [spring](http://lib.csdn.net/base/17" \o "Java EE知识库" \t "https://www.cnblogs.com/guoqixing/p/_blank)中自动代理所需jar包

## **二、使用@Transactional**

1、@Transactional可以在service类或方法前加上@Transactional，在service类上声明@Transactional，service所有方法需要事务管理。每一个业务方法开始时都会打开一个事务。

问题

No qualifying bean of type 'org.springframework.transaction.PlatformTransactionManager

You can use:

@Transactional("transactionManagerName")

# Ref

Spring零配置下的事务实现 - JJB的博客 - CSDN博客.html

零xml配置Spring事务管理 - dgeek的博客 - CSDN博客.html

Spring详解（八）------事务管理 - YSOcean - 博客园.html