# Spring注解驱动开发第34讲——你了解基于注解版的声明式事务吗?

文章目录



## 自我批评一下

不知不觉，时间已快过一个月了，而这个 Spring注解　驱动开发系列教程才写到第34讲，实在是说不过去，我向大家宣布一件事情，那就是：

> 我是一个傻逼!
> 我是一个傻逼!
> 我是一个傻逼!

我得承认这样一个事实，但无论如何抱怨，路还是要继续走下去，这可真是一件没有道理的事情。

好吧！回到正文，在这一讲中，我来向大家讲解一下声明式事务，也就是看看Spring用注解是如何来简化对数据库事务的开发的。

## 搭建声明式事务的环境

是个人都知道，在对数据库进行增删改操作时，必然是要使用到事务的。因此，接下来，我们就来搭建好声明式事务的基本环境。

### 导入相关依赖

首先，在项目的pom.xml文件中添加c3p0数据源的依赖，如下所示。

```xml
<dependency>
    <groupId>c3p0</groupId>
    <artifactId>c3p0</artifactId>
    <version>0.9.1.2</version>
</dependency>
```

AI写代码xml

然后，在项目的pom.xml文件中添加MySQL数据库　驱动的依赖，如下所示。

```xml
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.44</version>
</dependency>
```

AI写代码xml

最后，在项目的pom.xml文件中添加spring-jdbc模块的依赖，如下所示。

```xml
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>4.3.12.RELEASE</version>
</dependency>
```

AI写代码xml

Spring简化了对数据库的操作，只要我们在项目中导入了以上spring-jdbc模块的依赖，那么它就可以简化对数据库的操作以及事务控制。我们在后面就可以用Spring提供的JDBC模板（即JdbcTemplate）来操作数据库。

### 配置数据源以及JdbcTemplate

首先，我们得向IOC容器中注册一个c3p0数据源，那么如何做到这一点呢？很简单，先新建一个配置类，例如TxConfig，再使用@Bean注解向IOC容器中注册一个c3p0数据源，如下所示。

```java
package com.meimeixia.tx;

import javax.sql.DataSource;

```

```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import com.mchange.v2.c3p0.ComboPooledDataSource;

/**
 *
 * @author liayun
 *
 */
@Configuration
public class TxConfig {

    // 注册c3p0数据源
    @Bean
    public DataSource dataSource() throws Exception {
        ComboPooledDataSource dataSource = new ComboPooledDataSource();
        dataSource.setUser("root");
        dataSource.setPassword("liayun");
        dataSource.setDriverClass("com.mysql.jdbc.Driver");
        dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/test");
        return dataSource;
    }

}
```

AI写代码java运行

然后，再向IOC容器中注册一个JdbcTemplate组件，它是Spring提供的一个简化数据库操作的工具，它能简化对数据库的增删改查操作。

```java
@Bean
public JdbcTemplate jdbcTemplate() throws Exception {
    JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource());
    return jdbcTemplate;
}
```

AI写代码java运行

注意，在创建JdbcTemplate对象的时候，得把数据源传入JdbcTemplate类的有参构造器中，因为需要从数据源里面获取数据库连接。

其实，将数据源传入JdbcTemplate类的有参构造器中，一共有两种方式。第一种方式是将数据源作为一个参数传递到TxConfig配置类的jdbcTemplate()方法中。这样，JdbcTemplate类的有参构造器就可以使用到这个数据源了。

```java
@Bean
public JdbcTemplate jdbcTemplate(DataSource dataSource) throws Exception {
    JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
    return jdbcTemplate;
}
```

AI写代码java运行

为什么可以这样做呢？因为@Bean注解标注的方法在创建对象的时候，方法参数的值是从IOC容器中获取的，并且标注在这个方法的参数上的@Autowired注解可以省略。

第二种方式就不用那么麻烦了，在JdbcTemplate类的有参构造器中调用一次dataSource()方法即可。可以看到，向IOC容器中注册一个JdbcTemplate组件时，使用的就是这种方式。

有些同学可能会有一些疑问，TxConfig配置类的dataSource()方法是向IOC容器中注册一个c3p0数据源的，该方法的逻辑也很简单，就是创建一个c3p0数据源并将其返回出去，而在向IOC容器中注册一个JdbcTemplate组件时，会在其有参构造器中调用一次dataSource()方法，那岂不是又会创建一个c3p0数据源呢？不知你会不会有这样一个疑问，反正我是有的。其实，并不会再创建一个c3p0数据源，因为对于Spring的配置类而言，只要某个方法是给IOC容器中注册组件的，那么我们第二次调用该方法，就相当于是从IOC容器中找组件，而不是说把该方法再运行一遍。

总结一下，**Spring对@Configuration注解标注的类会做特殊处理，多次调用给IOC容器中添加组件的方法，都只是从IOC容器中找组件而已。**

### 码起来

首先在test数据库中临时创建一张表，例如tbl_user，建表语句如下：

```sql
CREATE TABLE `tbl_user` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(50) DEFAULT NULL,
  `age` int(2) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

AI写代码sql

表建好之后，我们就来说说开发需求，其实很简单，就是向tbl_user表中插入一条记录。接下来，我们就来编码实现这个需求。

### 开发dao层

新建一个UserDao类，代码如下所示：

```java
package com.meimeixia.tx;

import java.util.UUID;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

@Repository
public class UserDao {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    public void insert() {
        String sql = "insert into `tbl_user`(username, age) values(?, ?)";
        String username = UUID.randomUUID().toString().substring(0, 5);
        jdbcTemplate.update(sql, username, 19); // 增删改都来调用这个方法
    }

}
```

AI写代码java运行

注意，该类上标注了一个@Repository注解，因为待会我们要用到@ComponentScan注解来配置包扫描。

### 开发service层

新建一个UserService类，代码如下所示：

```java
package com.meimeixia.tx;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class UserService {

    @Autowired
    private UserDao userDao;

    public void insertUser() {
        userDao.insert();
        System.out.println("插入完成...");
    }

}
```

AI写代码java运行

可以看到，现在默认insertUser()方法是没有任何事务特性的。如果这个方法上有事务，那么只要这个方法里面有任何一句代码出现了问题，该行代码之前执行的所有操作就都应该回滚。

此外，还应注意到，该类上标注了一个@Service注解。

接下来，我们就要在TxConfig配置类上添加@ComponentScan注解来配置包扫描了，如下所示。

```java
package com.meimeixia.tx;

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.jdbc.core.JdbcTemplate;

```

```java
import com.mchange.v2.c3p0.ComboPooledDataSource;

/**
 *
 * @author liayun
 *
 */
@ComponentScan("com.meimeixia.tx")
@Configuration
public class TxConfig {

    // 注册c3p0数据源
    @Bean
    public DataSource dataSource() throws Exception {
        ComboPooledDataSource dataSource = new ComboPooledDataSource();
        dataSource.setUser("root");
        dataSource.setPassword("liayun");
        dataSource.setDriverClass("com.mysql.jdbc.Driver");
        dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/test");
        return dataSource;
    }

    @Bean
    public JdbcTemplate jdbcTemplate() throws Exception {
        JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource());
        return jdbcTemplate;
    }

}
```

AI写代码java运行

至此，声明式事务的基本环境，我们就搭建好了，接下来就是来进行测试了。

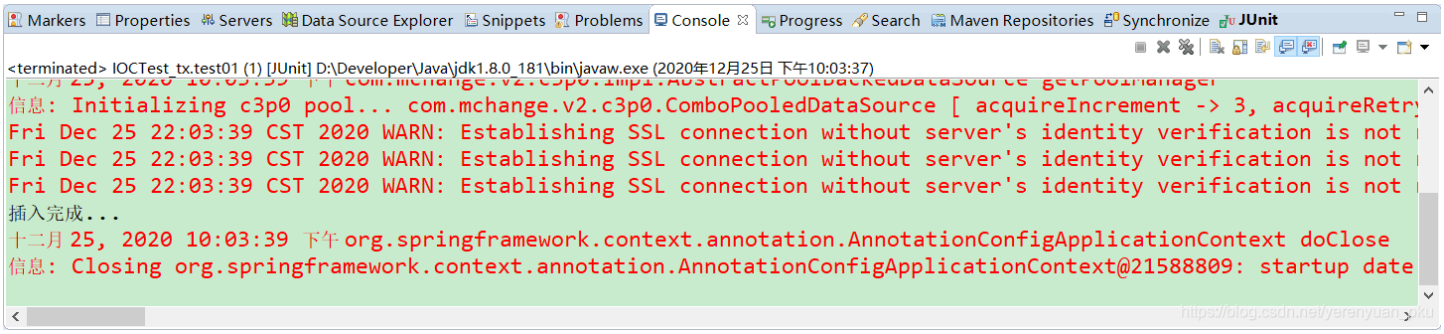
## 测试

新建一个单元测试类，例如IOCTest_tx，代码如下所示：

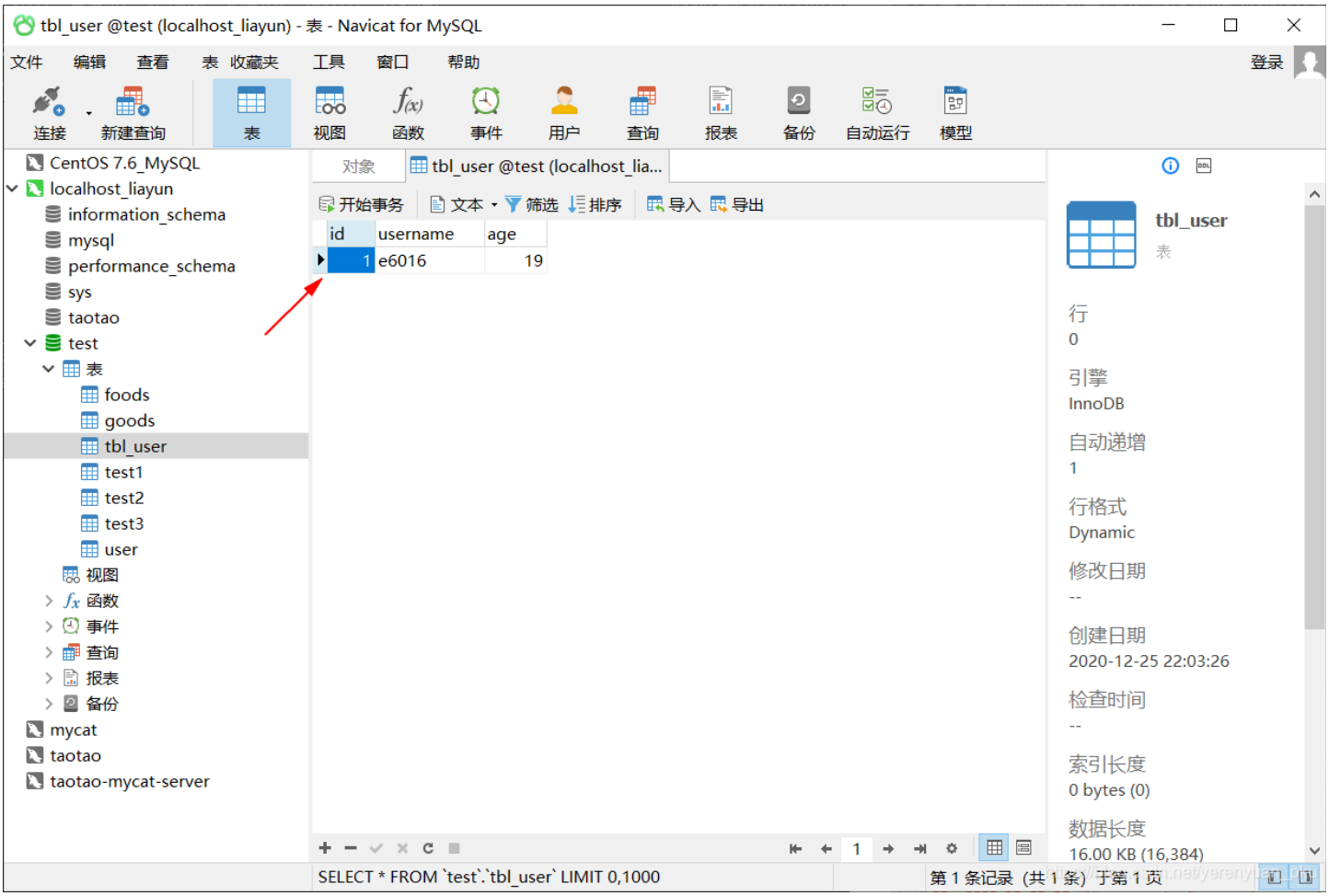
```java
package com.meimeixia.test;

import org.junit.Test;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.meimeixia.tx.TxConfig;
import com.meimeixia.tx.UserService;

public class IOCTest_tx {

    @Test
    public void test01() {
        AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(TxConfig.class);

        UserService userService = applicationContext.getBean(UserService.class);

        userService.insertUser();

        applicationContext.close();
    }

}
```

AI写代码java运行

运行以上test01()方法，发现Eclipse　控制台打印出了如下一条信息。




并且，刷新一下tbl_user表，可以看到确实是向该表中插入了一条记录，如下图所示。




接下来，我们就为UserService类中的insertUser()方法添加上事务，添加上事务以后，只要这个方法里面有任何一句代码出现了问题，那么该行代码之前执行的所有操作就都应该回滚。

如果要想为该方法添加上事务，那么就得使用@Transactional注解了。我们在该方法上标注这么一个注解，就是为了告诉Spring这个方法它是一个事务方法，这样，Spring在执行这个方法的时候，就会自动地进行事务控制。如果该方法正常执行，没出现任何问题，那么该方法中的所有操作都会生效，最终就会提交；如果该方法运行期间出现异常，那么该方法中的所有操作都会回滚。

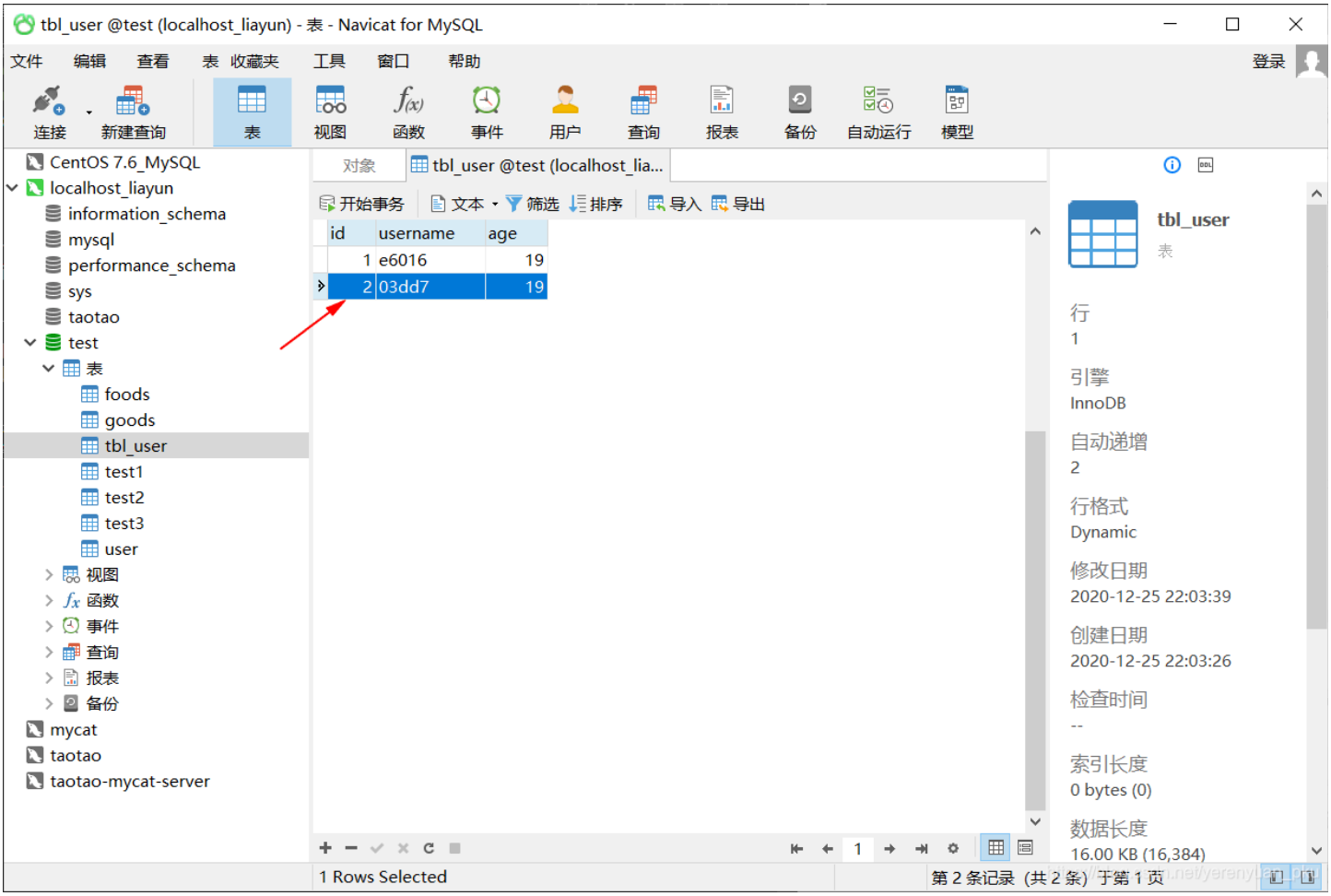
```java
@Transactional
public void insertUser() {
    userDao.insert();
    // otherDao.other(); // 该方法中的业务逻辑势必不会像现在这么简单，肯定还会调用其他dao的方法
    System.out.println("插入完成...");

    int i = 10 / 0;
}
```

AI写代码java运行

为insertUser()方法标注了@Transactional注解之后，那么它是不是就真的变成了一个事务方法呢？为了验证这一点，我们特地在该方法中故意抛出了一个算术异常。

目前tbl_user表中是只有一条记录的，如果insertUser()方法真的变成了一个事务方法，那么执行该方法再向tbl_user表中插入一条记录时，肯定是会出现问题的，既然出现了问题，插入操作势必就会回滚，最终tbl_user表中是不会再插入一条新记录的。

那么到底是不是这样的呢？我们拭目以待，运行完IOCTest_tx类中的test01()方法之后，虽说Eclipse控制台是打印出了 插入完成... 这样的消息，而且也给我们看到了除零的算术异常，但是刷新tbl_user表之后，你会发现仍然会向tbl_user表中插入一条新的记录，如下图所示。




这说明，虽然insertUser()方法是标注了@Transactional注解，但是它并不是一个真正的事务方法。

也就是说，光为insertUser()方法加一个@Transactional注解是不行的，那我们还得做什么呢？还得在TxConfig配置类上标注一个@EnableTransactionManagement注解，来开启基于注解的事务管理功能。

```
package com.meimeixia.tx;

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.jdbc.core.JdbcTemplate;
import org.springframework.transaction.annotation.EnableTransactionManagement;

import com.mchange.v2.c3p0.ComboPooledDataSource;

/**
 *
 * @author liayun
 *
 */
@EnableTransactionManagement // 它是来开启基于注解的事务管理功能的
@ComponentScan("com.meimeixia.tx")
@Configuration
public class TxConfig {

    // 注册c3p0数据源
    @Bean
    public DataSource dataSource() throws Exception {
        ComboPooledDataSource dataSource = new ComboPooledDataSource();
        dataSource.setUser("root");
        dataSource.setPassword("liayun");
        dataSource.setDriverClass("com.mysql.jdbc.Driver");
        dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/test");
        return dataSource;
    }

    @Bean
    public JdbcTemplate jdbcTemplate() throws Exception {
        JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource());

```

```
        return jdbcTemplate;
    }

}
```

AI写代码java运行

如果是像以前一样基于配置文件来开发，那么就得在配置文件中添加如下这样一行配置，来开启基于注解的事务管理功能。

```
<tx:annotation-driven/>
```

AI写代码xml

好，现在我们再来运行一下IOCTest_tx类中的test01()方法，发现Eclipse控制台并没有打印出 插入完成... 这样的消息，而是抛了一个如下所示的异常，即没有这样一个bean定义的异常。

```
org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type 'org.springframework.transaction.PlatformT
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.getBean(DefaultListableBeanFactory.java:353)
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.getBean(DefaultListableBeanFactory.java:340)
    at org.springframework.transaction.interceptor.TransactionAspectSupport.determineTransactionManager(TransactionAspectSupport.java:
    at org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(TransactionAspectSupport.java:272)
    at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:96)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:179)
    at org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedInterceptor.intercept(CglibAopProxy.java:673)
    at com.meimeixia.tx.UserService$$EnhancerBySpringCGLIB$$5cd1daa0.insertUser(<generated>)
    at com.meimeixia.test.IOCTest_tx.test01(IOCTest_tx.java:17)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
    at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
    at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:47)
    at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
    at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57)
    at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
    at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
    at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
    at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
    at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
    at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
    at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUnit4TestReference.java:86)
    at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)
    at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:459)
    at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:675)
    at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:382)
    at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:192)
```

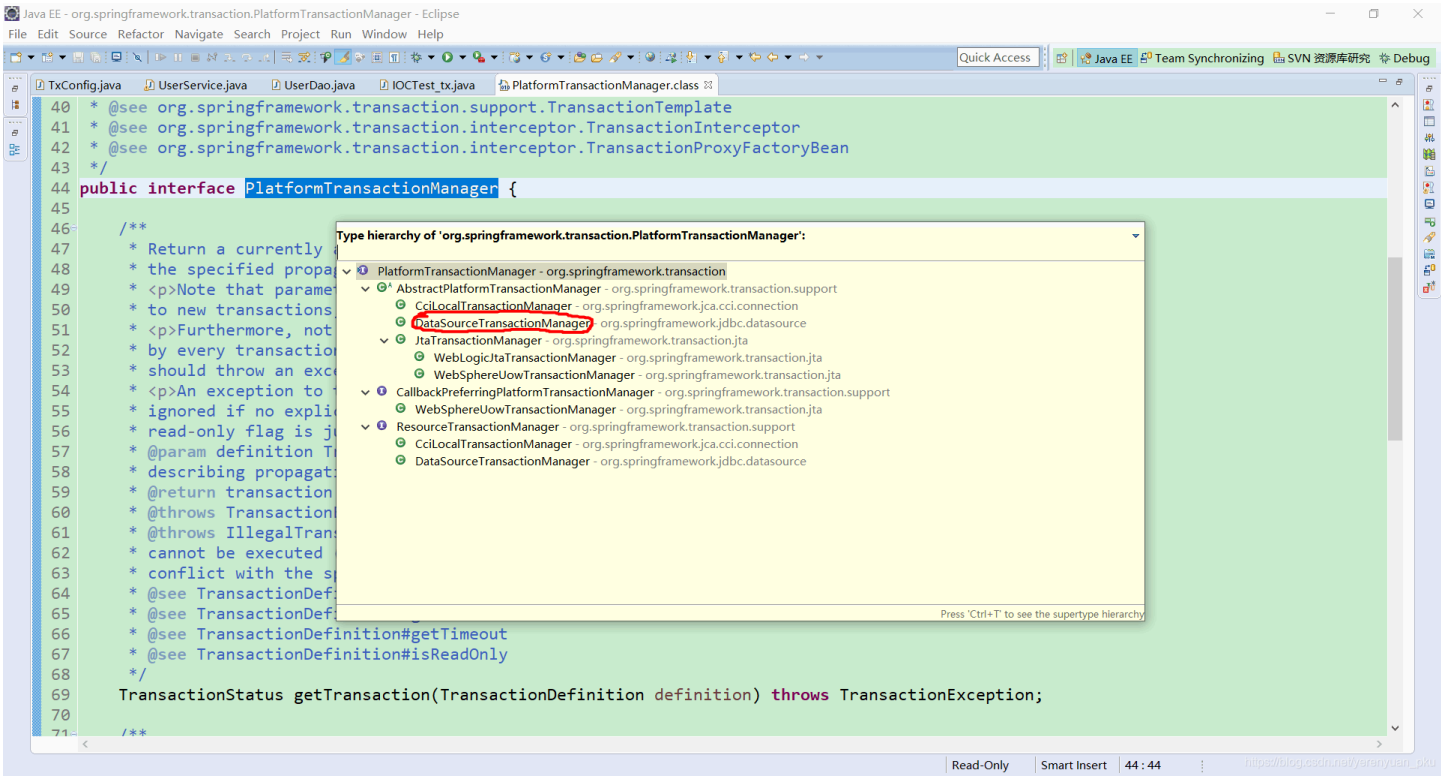
AI写代码java运行

抛了这样一个异常，自然是不会向tbl_user表中插入一条新的记录的。而且从NoSuchBeanDefinitionException异常的描述信息中我们可以知道，现在是没有 `org.springframework.transaction.PlatformTransactionManager` 这种类型的bean的定义的，也就是说我们还没有配置基于平台的事务管理器。

因此，最关键的一步就是配置事务管理器来控制事务。在这之前，我们可以查阅一下PlatformTransactionManager这个东东的源码，如下图所示。

```
TxConfig.java  UserService.java  UserDao.java  IOCTest_tx.java  PlatformTransactionManager.class

 2+ * Copyright 2002-2012 the original author or authors.
16
17 package org.springframework.transaction;
18
20+ * This is the central interface in Spring's transaction infrastructure.
44 public interface PlatformTransactionManager {
45
47+     * Return a currently active transaction or create a new one, according to
69     TransactionStatus getTransaction(TransactionDefinition definition) throws TransactionException;
70
72+     * Commit the given transaction, with regard to its status. If the transaction
98     void commit(TransactionStatus status) throws TransactionException;
99
101+    * Perform a rollback of the given transaction.
116    void rollback(TransactionStatus status) throws TransactionException;
117
118 }
119
```

发现它是一个接口，然后我们再来看看该接口有些什么实现类，如下图所示，发现该接口有很多实现类，其中有一个实现类是DataSourceTransactionManager，它使用频率很高，在这儿我们也是用它。




像Spring的spring-jdbc模块，以及MyBatis框架等等这些想要进行事务控制，都需要用到这个DataSourceTransactionManager实现类。

接下来，我们就向IOC容器中注册事务管理器，即需要向TxConfig配置类中添加一个如下方法。

```java
// 注册事务管理器在容器中
@Bean
public PlatformTransactionManager platformTransactionManager() throws Exception {
    return new DataSourceTransactionManager(dataSource());
}
```

AI写代码java运行

注意，这个事务管理器有一个特别重要的地方，就是它要管理数据源，也就是说事务管理器一定要把数据源控制住。这样的话，它才会控制住数据源里面的每一个连接，这时该连接上的回滚以及事务的开启等操作，都将会由这个事务管理器来做。

好了，现在我们就要来测试一把了。运行IOCTest_tx类中的test01()方法，你会发现Eclipse控制台不仅打印出了 插入完成... 这样的消息，而且还抛出了一个除零的算术异常，最重要的是没有向tbl_user表中插入一条新的记录，这说明insertUser()方法现在可真的成了一个事务方法。