
回到首页

芋道源码 —— 知识星球

我是一段不羁的公告！
记得给芳芳这 3 个项目加油，添加一个 STAR 噢。
https://github.com/YunaiV/SpringBoot-Labs
https://github.com/YunaiV/onemall
https://github.com/YunaiV/ruoyi-vue-pro
2020-07-01
Spring

精尽 Spring 源码分析 —— Transaction 源码简单导读

> 对应 Maven 模块为 `spring-tx` 。
>
> 为什么是 tx 缩写呢？因为 Transaction 的发音，和 tx 比较接近。

1. 前置内容

Spring Transaction 主要基于 Spring AOP 机制来实现的，所以建议对 AOP 的源码有一定的了解。

当然，一般情况下，也是先看完 AOP 相关的源码，在来看 Transaction 的源码。

> 如果胖友头铁，直接开始看 Transaction 的源码，也未尝不可。

2. 如何调试

① 调试 `<tx:advice />` 标签的解析的流程

可调试 `org.springframework.transaction.TxNamespaceHandlerTests` 这个单元测试里的方法。

另外，如果你想调试事务的提交和回滚，可以调试如下两个单元测试方法：

> `#invokeTransactional()` 方法，提交事务。
> `#rollbackRules()` 方法，回滚事务。

当然，这两个方法，是不包含 DB 相关的，只能了解大体的事务执行过程。

② 调试 `@Transactional` 注解的解析的流程

使用的还是 `org.springframework.transaction.TxNamespaceHandlerTests` 这个单元测试里的方法。下面，芳芳列下做出调整的地方。

1、增加 TestBean2 类。如下：

```
package org.springframework.transaction.yunai; // <1>

import org.springframework.tests.sample.beans.TestBean;
import org.springframework.transaction.annotation.Transactional;

public class TestBean2 extends TestBean {

    @Override
    @Transactional // <2>
    public Object returnsThis() {
        return super.returnsThis();
    }

}
```

<1> 处，因为 TestBean 在 spring-beans 项目里，不好加上 @Transactional 注解。
<2> 处，增加 @Transactional 注解。

2、修改 txNamespaceHandlerTests.xml 配置文件。如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:aop="http://www.springframework.org/schema/aop"
     xmlns:tx="http://www.springframework.org/schema/tx"
     xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spri
                http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.5.xs
                http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

    <!-- add by 芋艿：用于触发 AnnotationDrivenBeanDefinitionParser 的调试 -->
    <tx:annotation-driven proxy-target-class="true" order="0" />

    <aop:config>
        <!-- modify by 芋艿：增加 order="1" -->
     <aop:advisor pointcut="execution(* *..ITestBean.*(..))" advice-ref="txAdvice" order="1" />
 </aop:config>

 <tx:advice id="txAdvice"> <!-- 创建 TransactionInterceptor -->
     <tx:attributes>
         <tx:method name="get*" read-only="true"/>
         <tx:method name="set*"/>
         <tx:method name="exceptional"/>
     </tx:attributes>
 </tx:advice>

 <tx:advice id="txRollbackAdvice"> <!-- 创建 TransactionInterceptor -->
     <tx:attributes>
         <tx:method name="get*" rollback-for="java.lang.Exception"/>
         <tx:method name="set*" no-rollback-for="java.lang.RuntimeException"/>
     </tx:attributes>
 </tx:advice>

 <bean id="transactionManager" class="org.springframework.tests.transaction.CallCountingTransactionManager"/>

 <bean id="testBean" class="org.springframework.tests.sample.beans.TestBean"/>

    <bean id="testBean2" class="org.springframework.transaction.yunai.TestBean2" /> <!-- add by 芋艿：增加 TestBean2
```

```
</beans>
```

有 <!-- add by 芋艿 --> 和 <!-- modify by 芋艿 --> 的部分，就是艿艿修改过的地方。

3、在 TxNamespaceHandlerTests 类中，增加新的单元测试方法。代码如下：

```
// TxNamespaceHandlerTests.java

// add by 芋艿
@Test
public void invokeTransactional2() {
    ITestBean bean = getTestBean2();
    bean.returnsThis(); // 测试，我们对 @Transactional 注解的效果
//        bean.getAge(); // 测试，原有 xml 配置的事务的效果
}

// add by 芋艿
private ITestBean getTestBean2() {
    return (ITestBean) context.getBean("testBean2");
}
```

愉快的运行吧。

③ 基于 DB 的事务管理器 DataSourceTransactionManager 的事务的流程

可调试 org.springframework.transaction.TxNamespaceHandlerTests 这个单元测试里的方法。里面的测试用例非常多，胖友可以挑选自己需要的。当然，如果胖友想测试不同事务传播级别的组合，也可以自己写单元测试，示例如下：

```
@Test
public void test01() throws SQLException {
    given(con.getAutoCommit()).willReturn(true);

    final TransactionTemplate tt01 = new TransactionTemplate(tm);
    tt01.setPropagationBehavior(TransactionTemplate.PROPAGATION_REQUIRES_NEW);

    final TransactionTemplate tt02 = new TransactionTemplate(tm);
    tt02.setPropagationBehavior(TransactionTemplate.PROPAGATION_NOT_SUPPORTED);

    // 首先，PROPAGATION_REQUIRES_NEW
    tt01.execute(new TransactionCallbackWithoutResult() {

        @Override
        protected void doInTransactionWithoutResult(TransactionStatus status) {

            // 然后，PROPAGATION_NOT_SUPPORTED
            tt02.execute(new TransactionCallbackWithoutResult() {

                @Override
                protected void doInTransactionWithoutResult(TransactionStatus status) {
                    System.out.println("滴滴滴");

                    // 最后，PROPAGATION_REQUIRES_NEW
                    tt01.execute(new TransactionCallbackWithoutResult() {
                        @Override
```

```
                        protected void doInTransactionWithoutResult(TransactionStatus status) {
                            System.out.println("滴滴滴");
                        }
                    });

                }

            });
        }

    });
}
```

使用 org.springframework.transaction.support.TransactionTemplate 类，很方便的，胖友直接上手写两个。

3. 推荐资料

【必读】首先，推荐的是《Spring 源码深度解析》的 「第10章 事务」 章节。因为事务内嵌 PROPAGATION_NESTED 的传播级别是 Spring 独有，且实际场景下非常少（ 和一群基佬沟通了下，都没用到 ），所以艿艿自己也没研究。也就说，想要偷懒的胖友，请放心偷懒，哈哈哈哈。

然后，推荐的源码解析文章是

张强的 《原创 Spring 源码解析之事务篇》
Bww 《Spring-事务的源码分析（七）》

最后，推荐一些和 Transaction 相关的有趣的文章：

《可能是最漂亮的 Spring 事务管理详解》

4. 后置内容

看完 Spring AOP 之后，可以考虑看看 Spring MVC 啦。

虽然说，MVC 和 AOP 没什么必然的联系，但是距离我们日常开发特别接近，并且拓展的空间也会更多。

5. 重要的类

老艿艿：本小节，就是艿艿简单的笔记，可以忽略。嘿嘿。后面，在找时间完善下。

事务定义

- org.springframework.transaction.TransactionDefinition 接口，事务定义接口。
  - org.springframework.transaction.interceptor.TransactionAttribute 接口，支持定义返回异常回滚的事务定义接口。
    - org.springframework.transaction.interceptor.RuleBasedTransactionAttribute 类，基于 {@link RollbackRuleAttribute} 的事务定义实现类。
  - ps：因为我们看到，即有 Definition（ 定义 ），又有 Attribute（ 属性 ）的两种

结尾，考虑到好理解，统一下面叫属性，即事务属性。
    - 每个 `@Transactional` 注解的方法，都会被解析成一个 RuleBasedTransactionAttribute 对象。
    - 每个 `<tx:method />` XML 的配置，也会被解析成一个 RuleBasedTransactionAttribute 对象。

事务属性源

- org.springframework.transaction.interceptor.TransactionAttributeSource 接口，事务属性源，可以理解成 TransactionAttribute 的 Map 。
    - org.springframework.transaction.interceptor.AbstractFallbackTransactionAttributeSource 抽象类，支持按照【实现方法的事务属性 > 实现类的事务属性 > 接口的事务属性 > 接口的事务属性】的优先级，获取 TransactionAttribute 的 TransactionAttributeSource 抽象类。重点在 `#computeTransactionAttribute(Method method, Class<?> targetClass)` 方法的实现。
        - 【用于注解配置】org.springframework.transaction.annotation.AnnotationTransactionAttributeSource 类，基于 {@link Transactional} 注解的事务的 TransactionAttributeSource 实现类。
    - 【用于 XML 配置】org.springframework.transaction.interceptor.NameMatchTransactionAttributeSource ，基于方法名模式匹配的 TransactionAttributeSource 实现类。

解析器相关

- org.springframework.transaction.config.TxNamespaceHandler 类，Transaction 自定义标签的 NamespaceHandler 实现类。
- org.springframework.transaction.annotation.SpringTransactionAnnotationParser 类，解析 `@Transactional` 注解的解析器。
- org.springframework.transaction.config.TxAdviceBeanDefinitionParser 类，解析 `<tx:method />` XML 配置的解析器。

AOP 相关

- org.springframework.transaction.interceptor.TransactionInterceptor 类，AOP 事务拦截器。重点在其父类 TransactionAspectSupport 的 `#invokeWithinTransaction(Method method, Class<?> targetClass, final InvocationCallback invocation)` 方法。
- org.springframework.transaction.interceptor.TransactionAttributeSourceAdvisor ，事务属性源 Advisor 实现类。
- org.springframework.transaction.interceptor.TransactionAttributeSourcePointcut ，事务属性源 Pointcut 实现类。

事务管理器

- org.springframework.transaction.TransactionStatus 接口，事务状态接口。
    - org.springframework.transaction.support.AbstractTransactionStatus 抽象类，事务状态抽象类。
        - org.springframework.transaction.support.DefaultTransactionStatus 类，默认事务状态实现类。
- org.springframework.transaction.PlatformTransactionManager 接口，事务管理器接口。定义了 `#getTransaction(TransactionDefinition definition)`、`#commit(TransactionStatus status)`、`#rollback(TransactionStatus status)` 事务操作的三个关键方法。
    - 【重要】org.springframework.transaction.support.AbstractPlatformTransactionManager 抽象类，事务管理器抽象类，实现事务管理的核心流程。

        - 【重要】org.springframework.jdbc.datasource.DataSourceTransactionManager 类，JDBC DataSource 事务管理器。另外，该类在 spring-jdbc 模块中实现。
        - org.springframework.orm.hibernate5.HibernateTransactionManager 类，Hibernate 事务管理器。另外，该类在 spring-hibernate 模块中实现。 感兴趣的胖友，可以自己研究。我…没有研究，哈哈哈哈。
    - org.springframework.transaction.support.TransactionSynchronizationManager ，基于 ThreadLocal ，记录当前线程的事务的一些信息。例如：1）事务的隔离级别、；2）在 DataSourceTransactionManager 事物管理器中，会存储当前的数据库连接信息。

文章目录

回到首页