



黑马程序员™
www.itheima.com

传智播客旗下
高端IT教育品牌

声明式事务控制

目录 Contents

- ◆ 程式化事务控制相关对象
- ◆ 基于 XML 的声明式事务控制
- ◆ 基于注解的声明式事务控制

1. 程式事务控制相关对象

1.1 PlatformTransactionManager

PlatformTransactionManager 接口是 spring 的事务管理器，它里面提供了我们常用的操作事务的方法。

方法	说明
TransactionStatus getTransaction(TransactionDefination defination)	获取事务的状态信息
void commit(TransactionStatus status)	提交事务
void rollback(TransactionStatus status)	回滚事务

注意:

PlatformTransactionManager 是接口类型，不同的 Dao 层技术则有不同的实现类，例如：Dao 层技术是jdbc 或 mybatis 时：org.springframework.jdbc.datasource.DataSourceTransactionManager
Dao 层技术是hibernate时：org.springframework.orm.hibernate5.HibernateTransactionManager

1. 程式事务控制相关对象

1.2 TransactionDefinition

TransactionDefinition 是事务的定义信息对象，里面有如下方法：

方法	说明
<code>int getIsolationLevel()</code>	获得事务的隔离级别
<code>int getPropogationBehavior()</code>	获得事务的传播行为
<code>int getTimeout()</code>	获得超时时间
<code>boolean isReadOnly()</code>	是否只读

■ 1. 程式事务控制相关对象

1.2 TransactionDefinition

1. 事务隔离级别


设置隔离级别，可以解决事务并发产生的问题，如脏读、不可重复读和虚读。

- ISOLATION_DEFAULT
- ISOLATION_READ_UNCOMMITTED
- ISOLATION_READ_COMMITTED
- ISOLATION_REPEATABLE_READ
- ISOLATION_SERIALIZABLE

■ 1. 程式事务控制相关对象

1.2 TransactionDefinition

2. 事务传播行为

 A调B，如果A有事务，B就加入，如果A没有事务，B就新建一个事务

- REQUIRED：如果当前没有事务，就新建一个事务，如果已经存在一个事务中，加入到这个事务中。一般的选择（默认值）
- SUPPORTS：支持当前事务，如果当前没有事务，就以非事务方式执行（没有事务）
- MANDATORY：使用当前的事务，如果当前没有事务，就抛出异常
- REQUIRES_NEW：新建事务，如果当前在事务中，把当前事务挂起。
- NOT_SUPPORTED：以非事务方式执行操作，如果当前存在事务，就把当前事务挂起
- NEVER：以非事务方式运行，如果当前存在事务，抛出异常
- NESTED：如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则执行 REQUIRED 类似的操作
- 超时时间：默认值是-1，没有超时限制。如果有，以秒为单位进行设置
- 是否只读：建议查询时设置为只读

1. 程式事务控制相关对象

1.3 TransactionStatus

TransactionStatus 接口提供的是事务具体的运行状态，方法介绍如下。

方法	说明
<code>boolean hasSavepoint()</code>	是否存储回滚点
<code>boolean isCompleted()</code>	事务是否完成
<code>boolean isNewTransaction()</code>	是否是新事务
<code>boolean isRollbackOnly()</code>	事务是否回滚

■ 1. 程式事务控制相关对象

1.4 知识要点

程式事务控制三大对象

- PlatformTransactionManager
- TransactionDefinition
- TransactionStatus

目录 Contents

- ◆ 编程式事务控制相关对象
- ◆ 基于 XML 的声明式事务控制
- ◆ 基于注解的声明式事务控制

■ 2. 基于 XML 的声明式事务控制

2.1 什么是声明式事务控制

Spring 的声明式事务顾名思义就是**采用声明的方式来处理事务**。这里所说的声明，就是指在配置文件中声明，用在 Spring 配置文件中声明式的处理事务来代替代码式的处理事务。

声明式事务处理的作用

- 事务管理不侵入开发的组件。具体来说，业务逻辑对象就不会意识到正在事务管理之中，事实上也应该如此，因为事务管理是属于系统层面的服务，而不是业务逻辑的一部分，如果想要改变事务管理策划的话，也只需要在定义文件中重新配置即可
- 在不需要事务管理的时候，只要在设定文件上修改一下，即可移去事务管理服务，无需改变代码重新编译，这样维护起来极其方便

注意：Spring 声明式事务控制底层就是AOP。

■ 2. 基于 XML 的声明式事务控制

2.2 声明式事务控制的实现

声明式事务控制明确事项：

- 谁是切点？
- 谁是通知？
- 配置切面？

■ 2. 基于 XML 的声明式事务控制

2.2 声明式事务控制的实现

① 引入tx命名空间

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:aop="http://www.springframework.org/schema/aop"
        xmlns:tx="http://www.springframework.org/schema/tx"
        xsi:schemaLocation="

        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">
```

2. 基于 XML 的声明式事务控制

2.2 声明式事务控制的实现

② 配置事务增强

```
<!--平台事务管理器-->
<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"></property>
</bean>

<!--事务增强配置-->
<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="*" />
    </tx:attributes>
</tx:advice>
```

■ 2. 基于 XML 的声明式事务控制

2.2 声明式事务控制的实现

③ 配置事务 AOP 织入

```
<!--事务的aop增强-->
<aop:config>
    <aop:pointcut id="myPointcut" expression="execution(*
com.itheima.service.impl.*.*(..))"/>
    <aop:advisor advice-ref="txAdvice" pointcut-ref="myPointcut"></aop:advisor>
</aop:config>
```

2. 基于 XML 的声明式事务控制

2.2 声明式事务控制的实现

④ 测试事务控制转账业务代码

```
@Override  
public void transfer(String outMan, String inMan, double money) {  
    accountDao.out(outMan,money);  
    int i = 1/0;  
    accountDao.in(inMan,money);  
}
```

2. 基于 XML 的声明式事务控制

2.3 切点方法的事务参数的配置

```
<!--事务增强配置-->
<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="*" />
    </tx:attributes>
</tx:advice>
```

其中，<tx:method> 代表切点方法的事务参数的配置，例如：

```
<tx:method name="transfer" isolation="REPEATABLE_READ" propagation="REQUIRED" timeout="-1"
read-only="false"/>
```

- name: 切点方法名称
- isolation: 事务的隔离级别
- propagation: 事务的传播行为
- timeout: 超时时间
- read-only: 是否只读

■ 2. 基于 XML 的声明式事务控制

2.4 知识要点

声明式事务控制的配置要点

- 平台事务管理器配置
- 事务通知的配置
- 事务aop织入的配置

```
<!-- 配置平台事务管理器 -->  
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  
    <property name="dataSource" ref="dataSource"/></property>  
</bean>
```

指定需要增强的方法

```
<!-- 通知 事务的增强 -->  
<tx:advice id="txAdvice" transaction-manager="transactionManager">  
    <!-- 设置事务的属性信息的 -->  
    <tx:attributes>  
        <tx:method name="transfer" isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="false"/>  
        <tx:method name="update*" isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="true"/>  
        <tx:method name="*" />  
    </tx:attributes>  
</tx:advice>
```

```
<!-- 配置事务的aop织入 -->  
<aop:config>  
    <aop:pointcut id="txPointcut" expression="execution(* com.itheima.service.impl.*(..))"/>  
    <aop:advisor advice-ref="txAdvice" pointcut-ref="txPointcut"/>  
</aop:config>
```

目录 Contents

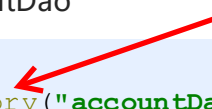
- ◆ 编程式事务控制相关对象
- ◆ 基于 XML 的声明式事务控制
- ◆ 基于注解的声明式事务控制

3. 基于注解的声明式事务控制

3.1 使用注解配置声明式事务控制

1. 编写 AccountDao

用于存储层bean



```
@Repository("accountDao")
public class AccountDaoImpl implements AccountDao {
    @Autowired
    private JdbcTemplate jdbcTemplate;

    public void out(String outMan, double money) {
        jdbcTemplate.update("update account set money=money-? where
name=?", money, outMan);
    }

    public void in(String inMan, double money) {
        jdbcTemplate.update("update account set money=money+? where
name=?", money, inMan);
    }
}
```

3. 基于注解的声明式事务控制

3.1 使用注解配置声明式事务控制

2. 编写 AccountService

用于业务层bean

```
@Service("accountService")
@Transactional
public class AccountServiceImpl implements AccountService {
    @Autowired
    private AccountDao accountDao;

    @Transactional(isolation = Isolation.READ_COMMITTED, propagation =
    Propagation.REQUIRED)
    public void transfer(String outMan, String inMan, double money) {
        accountDao.out(outMan, money);
        int i = 1/0;
        accountDao.in(inMan, money);
    }
}
```

3. 基于注解的声明式事务控制

3.1 使用注解配置声明式事务控制

3. 编写 applicationContext.xml 配置文件

```
<!--之前省略dataSource、jdbcTemplate、平台事务管理器的配置-->  
<!--组件扫描-->  
<context:component-scan base-package="com.itheima"/>  
<!--事务的注解驱动-->  
<tx:annotation-driven/>
```

3. 基于注解的声明式事务控制

3.2 注解配置声明式事务控制解析

- ① 使用 @Transactional 在需要进行事务控制的类或是方法上修饰，注解可用的属性同 xml 配置方式，例如隔离级别、传播行为等。
- ② 注解使用在类上，那么该类下的所有方法都使用同一套注解参数配置。
- ③ 使用在方法上，不同的方法可以采用不同的事务参数配置。
- ④ Xml配置文件中要开启事务的注解驱动<tx:annotation-driven />

■ 3. 基于注解的声明式事务控制

3.3 知识要点

注解声明式事务控制的配置要点

- 平台事务管理器配置 (xml方式)
- 事务通知的配置 (@Transactional注解配置)
- 事务注解驱动的配置 `<tx:annotation-driven/>`



传智播客旗下高端IT教育品牌