链滴

spring 编程式事务、声明式事务

作者：moloee
原文链接：https://ld246.com/article/1512442754320
来源网站：链滴

事务管理是应用系统中必不可少的一部分，它保证了用户的每一次操作都是可靠的，即便是出现了异情况，也不至于破坏后台数据的完整性。 Spring提供了丰富的事务管理功能，Spring的事务管分为编程式事务管理和声明式事务管理两种方式。编程式事务管理指通过编码的方式实现事务管理，明式事务基于AOP，将业务逻辑与事务处理解耦。声明式事务对代码侵入较少，在实际使用中使用比广泛。

一、包依赖

项目中使用的Spring和MyBatis包依赖如下：

```
...
    <properties>
        <spring-version>4.2.2.RELEASE</spring-version>
    </properties>

    <dependencies>
        <!-- ***************** -->
        <!--       spring      -->
        <!-- ***************** -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
            <version>${spring-version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring-version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>${spring-version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>${spring-version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>${spring-version}</version>
        </dependency>

        <!-- ***************** -->
        <!--       mybatis     -->
        <!-- ***************** -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis-spring</antifactId>
            <version>1.3.0</version>
```

```
        </dependency>
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.3.1</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.41</version>
        </dependency>
...
    </dependencies>
...
```

二、编程式事务

Spring编程式事务管理通过编码的方式实现事务管理，需要在代码中显示的getTransaction(), commit(), rollback()等事务管理方法，通过这些Spring提供的API可以灵活控制事务的执行，在底层，Spring将这些事务的操作委托给持久化框架执行。

Spring配置文件config.xml如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.springframework.org/schema/beans"
      xmlns:tx="http://www.springframework.org/schema/tx"
      xmlns:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
              http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
          http://www.springframework.org/schema/tx
              http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
          http://www.springframework.org/schema/context
              http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <!-- 引入属性文件 -->
    <bean id="propertyConfigurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="locations">
            <list>
                <value>classpath:config.properties</value>
            </list>
        </property>
    </bean>

    <!-- 配置数据源 -->
    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName">
            <value>${driver}</value>
        </property>
        <property name="url">
```

```
        <value>${url}</value>
    </property>
    <property name="username">
        <value>${username}</value>
    </property>
    <property name="password">
        <value>${password}</value>
    </property>
</bean>

<!-- 自动扫描了所有的mapper配置文件对应的mapper接口文件 -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.xiaofan.test" />
</bean>

<!-- 配置Mybatis的文件 -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="mapperLocations" value="classpath:user_mapper.xml"/>
    <property name="configLocation" value="classpath:mybatis_config.xml" />
</bean>

<!-- 配置JDBC事务管理器 -->
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceT
ansactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>

<bean id="userService" class="com.xiaofan.test.UserService">
</bean>

</beans>
```

根据PlatformTransactionManager、TransactionDefinition和TransactionStatus三个接口，以通过编程的方式来进行事务管理, TransactionDefinition实例用于定义一个事务，PlatformTransact onManager实例用语执行事务管理操作，TransactionStatus实例用于跟踪事务的状态。UserServic 服务中配置如下：

```
public class UserService {

    @Resource
    UserDAO userDAO;

    @Resource
    DataSource dataSource;

    @Resource
    PlatformTransactionManager transactionManager;

    public void addUser(User user) throws Exception {

        TransactionDefinition def = new DefaultTransactionDefinition();
        TransactionStatus status = transactionManager.getTransaction(def);
```

```
        try {

            // [1] 插入纪录
            userDAO.insert(user);

            // [2] 范例抛出异常
            Integer i = null;
            if (i.equals(0)) {

            }

            transactionManager.commit(status);
        } catch (Exception e) {
            transactionManager.rollback(status);
            throw e;
        }
        return;
    }

}
```

Spring测试代码如下

```
@ContextConfiguration(locations = {"classpath:config.xml"})
@RunWith(SpringJUnit4ClassRunner.class)
public class Test extends AbstractJUnit4SpringContextTests{

    @Resource
    UserService userService;

    @org.junit.Test
    public void testAdd() {
        try {
            userService.addUser(new User(null, "LiLei", 25));
        } catch (Exception e) {
        }
    }
}
```

如果［2］处抛出异常，则事务执行回滚，如果［2］没有抛出异常，则提交执行纪录插入操作。

另一种编程式事务管理

以上这种事务管理方式容易理解，但事务管理代码散落在业务代码中，破坏了原有代码的条理性且每个事务方法中都包含了启动事务、提交/回滚事务的功能，基于此，Spring提供了简化的模版回模式（TransactionTemplate）。 在config.xml配置文件中加入TransactionTemplate bean配：

```
...
    <bean id="transactionTemplate"  class="org.springframework.transaction.support.Transact
onTemplate">
        <property name="transactionManager" ref="transactionManager"/>
    # 新增
    <property name="isolationLevelName" value="ISOLATION_DEFAULT" />
```

```
    <property name="propagationBehaviorName" value="PROPAGATION_REQUIRED" />
    </bean>
...
```

TransactionTemplate 的execute()方法有一个TransactionCallback类型的参数，该接口中定义一个doInTransaction()方法，可通过匿名内部累的方式实现TransactionCallBack接口，将业务代码在doInTransaction()方法中，业务代码中不需要显示调用任何事物管理API，除了异常回滚外，也可在业务代码的任意位置通过transactionStatus.setRollbackOnly();执行回滚操作。UserService服务码变更为：

注：如果抛异常的话，也会自动rollback

```
public class UserService {

    @Resource
    UserDAO userDAO;

    @Resource
    TransactionTemplate transactionTemplate;

    public void addUser(final User user) {

        transactionTemplate.execute(new TransactionCallback() {

            public Object doInTransaction(TransactionStatus transactionStatus) {

                userDAO.insert(user);

                // transactionStatus.setRollbackOnly();

                Integer i = null;

                if (i.equals(0)) {
                  // 自动rollback
                  throw new Exception("xxx");

                }
                return null;
            }
        });
    }
}
```

三、声明式事务

Spring的声明式事务管理建立在AOP基础上，其本质是在目标方法执行前进行拦截，在方法开始创建一个事务，在执行完方法后根据执行情况提交或回滚事务。声明式事务最大的优点就是不需要通编程的方式管理事务，这样就不用侵入业务代码，只需要在配置文件中做相关的事物声明就可将业务则应用到业务逻辑中。和编程式事务相比，声明式事务唯一的不足是智能作用到方法级别，无法做到编程式事务那样到代码块级别

声明式事务有**四种方式**，a.基于TransactionInterceptor的声明式事务；b.基于TransactionProxy actoryBean的声明式事务；c.基于\命名空间的声明式事务；d.基于标注（@Transactional）的声明事务。

注：一般d用得比较多，使用和学习成本都比较低。

a.基于TransactionInterceptor的声明式事务

TransactionInterceptor主要有两个属性，一个是transactionManager，用于指定一个事务管器；另一个是transactionAttributes，通过键值对的方式指定相应方法的事物属性，其中键值可以使通配符。在config.xml配置文件中加入TransactionInterceptor配置：

```
...
  <bean id="transactionInterceptor"
      class="org.springframework.transaction.interceptor.TransactionInterceptor">
    <property name="transactionManager" ref="transactionManager"/>
    <property name="transactionAttributes">
      <props>
        <prop key="*">PROPAGATION_REQUIRED</prop>
      </props>
    </property>
  </bean>
  <bean id="userService"
      class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="target">
      <bean class="com.xiaofan.test.UserService" />
    </property>
    <property name="interceptorNames">
      <list>
        <idref bean="transactionInterceptor"/>
      </list>
    </property>
  </bean>
...
```

其中事务的传播行为边界为：

传播	功能
PROPAGATION_REQUIRED	支持当前事务，如当前没有事务，则新建一个事务
PROPAGATION_SUPPORT	支持当前事务，如当前没有事务，则以非事务执行
PROPAGATION_MANDATORY	支持当前事务如果当前没有事务，则抛出异常
PROPAGATION_REQUIRES_NEW	新建事务，果当前存在事务，则把当前事务挂起
PROPAGATION_NOT_SUPPORT	以非事务方式作，如果当前有事务，则把当前事务挂起
PROPAGATION_NEVER	以非事务方式操作，如当前有事务，则抛出异常

UserService服务代码变更为：

```
public class UserService {
```

```
    @Resource
    UserDAO userDAO;

    public void addUser3(User user) {

        userDAO.insert(user);

        Integer i = 1;
        if (i.equals(0)) {
        }
    }
}
```

b.基于TransactionProxyFactoryBean的声明式事务

以上基于TransactionInterceptor的方式每个服务bean都需要配置一个ProxyFactoryBean，这导致配置文件冗长，为了缓解这个问题，Spring提供了基于TransactionProxyFactoryBean的声明式务配置方式。在config.xml配置文件中加入TransactionProxyFactoryBean配置：

```
...
    <bean id="userService" class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
        <property name="target">
            <bean class="com.xiaofan.test.UserService" />
        </property>
        <property name="transactionManager" ref="transactionManager"/>
        <property name="transactionAttributes">
            <props>
                <prop key="insert*">PROPAGATION_REQUIRED</prop>
                <prop key="update*">PROPAGATION_REQUIRED</prop>
                <prop key="*">PROPAGATION_REQUIRED</prop>
            </props>
        </property>
    </bean>
...
```

UserService服务代码如上，不用变更。

c.基于\命名空间的声明式事务

Spring 2.x引入了\命名空间，加上\命名空间的切点表达式支持，声明式事务变的更加强大，借于切点表达式，可以不需要为每个业务类创建一个代理。为了使用动态代理，首先需要添加pom依赖：

```
...
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.7.4</version>
        </dependency>
...
```

config.xml文件添加如下配置：

```
...
```

```
  <bean id="userService" class="com.xiaofan.test.UserService">
  </bean>
  <tx:advice id="userAdvice" transaction-manager="transactionManager">
    <tx:attributes>
      <tx:method name="*" propagation="REQUIRED"/>
    </tx:attributes>
  </tx:advice>
  <aop:config>
    <aop:pointcut id="userPointcut" expression="execution (* com.xiaofan.test.*.*(..))"/>
    <aop:advisor advice-ref="userAdvice" pointcut-ref="userPointcut"/>
  </aop:config>
...
```

UserService服务代码如上，不用变更。

d.基于标注（@Transactional）的声明式事务

除了基于命名空间的事务配置方式外，Spring2.x还引入了基于注解的方式，主要涉及@ransactional注解，它可以作用于接口、接口方法、类和类的方法上，当做用于类上时，该类的所有pblic方法都有该类型的事务属性，可被方法级事务覆盖。在config.xml配置文件中加入注解识别配置：

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

@Transactional注解应该被应用到public方法上，这是由AOP的本质决定的，如果应用在protec ed、private的方法上，事务将被忽略。UserService服务代码如下：

```
public class UserService {

  @Resource
  UserDAO userDAO;

  @Transactional(propagation = Propagation.REQUIRED)
  public void addUser4(User user) {

    userDAO.insert(user);

    Integer i = 1;
    if (i.equals(0)) {

    }

  }
}
```

@Transactional注解的完整属性信息如下表[1]：

属性名	功能
name	指定选择多个事务管理器中的某个事务管器
propagation	事务传播行为，默认为REQUIRED
isolation	事务个力度，默认为DEFAULT

timeout	事务超时时间，默认为 - 1
read-only	指定事务为只读，默认为false
rollback-for	指定触发事务回滚的异常类型，多异常通过逗号分隔
no-rollback-for	抛出指定的异常类型，不回滚

基于命名空间和基于注解的事务声明各有优缺点：基于命名空间的方式一个配置可以匹配多个方，但配置较注解方式复杂，但可以更细粒度的控制事务；基于注解的方式需要在每个需要使用事务的法或类上标注，但基于注解的方法学习成本更低。

参考文章