Spring事务管理

Spring是SSH中的管理员，负责管理其它框架，协调各个部分的工作。今天一起学习一下Spring的事务管理。Spring的事务管理分为声明式跟编程式。声明式就是在Spring的配置文件中进行相关配置；编程式就是用注解的方式写到代码里。下面先说声明式：

Spring配置文件中关于事务配置总是由三个组成部分，分别是DataSource、TransactionManager和代理机制这三部分，无论哪种配置方式，一般变化的只是代理机制这部分。 DataSource、 TransactionManager这两部分只是会根据数据访问方式有所变化，比如使用Hibernate进行数据访问时，DataSource实际为SessionFactory，TransactionManager的实现为HibernateTransactionManager。下面一起看看三种声明式事务的具体配置：

声明式事务

公共配置

复制代码

```
<!-- 配置sessionFactory -->
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">

    <property name="configLocation">
        <value>classpath:config/hibernate.cfg.xml</value>
    </property>
    <property name="packagesToScan">
        <list>
            <value>com.entity</value>
        </list>
    </property>
</bean>

<!-- 配置事务管理器（声明式的事务） -->
<bean id="transactionManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
```

```
    <property name="sessionFactory" ref="sessionFactory"></property>
</bean>

<!-- 配置DAO -->
<bean id="userDao" class="com.dao.UserDaoImpl">
    <property name="sessionFactory" ref="sessionFactory"></property>
</bean>
```

复制代码

第一种，使用tx标签方式

复制代码

```
<!-- 第一种配置事务的方式 ，tx-->
<tx:advice id="txadvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="add*" propagation="REQUIRED" rollback-for="Exception" />
        <tx:method name="modify*" propagation="REQUIRED" rollback-for="Exception"
/>
        <tx:method name="del*" propagation="REQUIRED" rollback-for="Exception"/>
        <tx:method name="*" propagation="REQUIRED" read-only="true"/>
    </tx:attributes>
</tx:advice>

<aop:config>
    <aop:pointcut id="daoMethod" expression="execution(* com.dao.*.*(..))"/>
    <aop:advisor pointcut-ref="daoMethod" advice-ref="txadvice"/>
</aop:config>
```

复制代码

expression="execution(* com.dao.*.*(..))"
其中第一个*代表返回值，第二*代表dao下子包，第三个*代表方法名，“（..）”代表方法参数。

第二种，使用代理方式

复制代码

```
<!-- 第二种配置事务的方式 ，代理-->
<bean id="transactionProxy"

class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean"
abstract="true">
    <property name="transactionManager" ref="transactionManager"></property>
    <property name="transactionAttributes">
        <props>
            <prop key="add*">PROPAGATION_REQUIRED, -Exception</prop>
            <prop key="modify*">PROPAGATION_REQUIRED, -Exception</prop>
            <prop key="del*">PROPAGATION_REQUIRED, -Exception</prop>
            <prop key="*">PROPAGATION_REQUIRED, readOnly</prop>
        </props>
    </property>
</bean>
<bean id="userDao" parent="transactionProxy">
    <property name="target">
        <!-- 用bean代替ref的方式-->
        <bean class="com.dao.UserDaoImpl">
            <property name="sessionFactory" ref="sessionFactory"></property>
        </bean>
    </property>
</bean>
```

复制代码

将transactionProxy的abstract属性设置为"true"，然后将具体的Dao的parent属性设置为"transactionProxy"，可以精简代码。

第三种，使用拦截器

复制代码

```
<!-- 第三种配置事务的方式，拦截器 （不常用）-->
<bean id="transactionInterceptor"
class="org.springframework.transaction.interceptor.TransactionInterceptor">
    <property name="transactionManager" ref="transactionManager"></property>
    <property name="transactionAttributes">
        <props>
            <prop key="add*">PROPAGATION_REQUIRED, -Exception</prop>
            <prop key="modify*">PROPAGATION_REQUIRED, -Exception</prop>
            <prop key="del*">PROPAGATION_REQUIRED, -Exception</prop>
            <prop key="*">PROPAGATION_REQUIRED, readOnly</prop>
        </props>
    </property>
</bean>
<bean id="proxyFactory"
class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">
    <property name="interceptorNames">
        <list>
            <value>transactionInterceptor</value>
        </list>
    </property>
    <property name="beanNames">
        <list>
            <value>*Dao</value>
        </list>
    </property>
</bean>
```

复制代码

Spring事务类型详解：

PROPAGATION_REQUIRED--支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择。

PROPAGATION_SUPPORTS--支持当前事务，如果当前没有事务，就以非事务方式执行。

PROPAGATION_MANDATORY--支持当前事务，如果当前没有事务，就抛出异常。

PROPAGATION_REQUIRES_NEW--新建事务，如果当前存在事务，把当前事务挂起。
PROPAGATION_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。
PROPAGATION_NEVER--以非事务方式执行，如果当前存在事务，则抛出异常。
PROPAGATION_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，则进行与PROPAGATION_REQUIRED类似的操作。

编程式事务

编程式即采用注解的方式，需要注意的是，使用注解的方式需要在Spring的配置文件中加入一句话：<context:annotation-config />，其作用是开启注解的方式。具体配置如下：


复制代码

```
<!--开启注解方式-->
<context:annotation-config />

<!-- 配置sessionFactory -->
<bean id="sessionFactory" 
class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">

    <property name="configLocation">
        <value>classpath:config/hibernate.cfg.xml</value>
    </property>
    <property name="packagesToScan">
        <list>
            <value>com.entity</value>
        </list>
    </property>
</bean>

<!-- 配置事务管理器 -->
<bean id="transactionManager" 
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory"></property>
```

```
</bean>

<!-- 第四种配置事务的方式，注解 -->
<tx:annotation-driven transaction-manager="transactionManager"/>
```

复制代码

注解文件：

复制代码

```
package com.dao;

import org.springframework.orm.hibernate3.HibernateTemplate;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;

import com.entity.User;

@Transactional
public class UserDaoImpl_BAK extends HibernateTemplate {

    @Transactional(propagation=Propagation.REQUIRED,rollbackForClassName="Exception")
    public void addUser(User user) throws Exception {
        this.save(user);
    }

    @Transactional(propagation=Propagation.REQUIRED,rollbackForClassName="Exception")
    public void modifyUser(User user) {
        this.update(user);
    }

    @Transactional(propagation=Propagation.REQUIRED,rollbackForClassName="Exception")
    public void delUser(String username) {
        this.delete(this.load(User.class, username));
    }
```

```
    @Transactional(readOnly=true)
    public void selectUser() {

    }

}
```

复制代码

类头的@Transactional为默认事务配置，如方法没有自己的事务类型，则按默认事务，如有自己的配置，则按自己的配置。

以上四种配置方式最常用的还是第一、二种，第三种是比较老旧的方式，而注解的方式不太适合比较大的项目，用于简单的小项目还是很好的，其特点就是简单明了。每种方法都有每种方法的特点跟适用的环境，没有绝对的好与坏，只不过前两种在实际的工作当中用的更多一些。