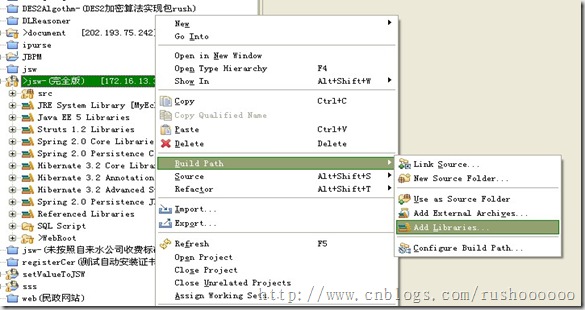
Spring事务配置(摘自网络)

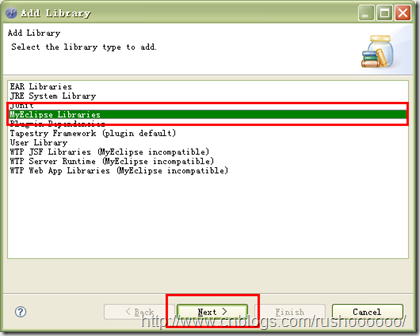
**环境配置**

项目使用SSH架构，现在要添加Spring事务管理功能，针对当前环境，只需要添加Spring 2.0 AOP类库即可。添加方法：

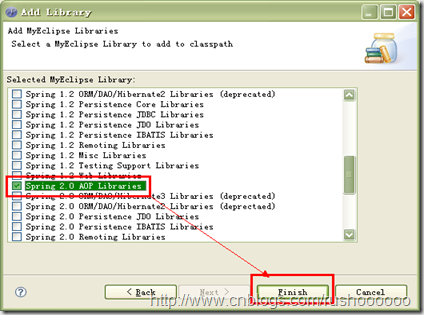
* 点击项目右键->Build Path->Add librarys:

[[](http://images.cnblogs.com/cnblogs_com/rushoooooo/201108/201108281004033211.jpg)](http://images.cnblogs.com/cnblogs_com/rushoooooo/201108/201108281004033211.jpg)

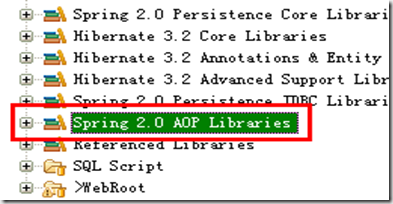
* 打开Add Libraries对话框，然后选定 MyEclipse Libraries：

[[](http://images.cnblogs.com/cnblogs_com/rushoooooo/201108/201108281004048129.png)](http://images.cnblogs.com/cnblogs_com/rushoooooo/201108/201108281004048129.png)

* 点击Next，找到Spring 2.0 aop Libraries并勾选上，点击finsh即可。

[](http://images.cnblogs.com/cnblogs_com/rushoooooo/201108/201108281004059458.png)

* 如果在项目里面能看到下面的库文件，说明已经安装成功。

[](http://images.cnblogs.com/cnblogs_com/rushoooooo/201108/201108281004051028.png)

**事务配置**

* 首先在/WEB-INF/applicationContext.xml添加以下内容：

<!-- 配置事务管理器 -->

<bean id="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">

<property name="sessionFactory">

<ref bean="mySessionFactory"/>

</property>

</bean>

注：这是作为公共使用的事务管理器Bean。这个会是事先配置好的，不需各个模块各自去配。

* 下面就开始配置各个模块所必须的部分，在各自的applicationContext-XXX-beans.xml配置的对于事务管理的详细信息。

首先就是配置事务的传播特性，如下：

<!-- 配置事务传播特性 -->

<tx:advice id="TestAdvice" transaction-manager="transactionManager">

<tx:attributes>

<tx:method name="save\*" propagation="REQUIRED"/>

<tx:method name="del\*" propagation="REQUIRED"/>

<tx:method name="update\*" propagation="REQUIRED"/>

<tx:method name="add\*" propagation="REQUIRED"/>

<tx:method name="find\*" propagation="REQUIRED"/>

<tx:method name="get\*" propagation="REQUIRED"/>

<tx:method name="apply\*" propagation="REQUIRED"/>

</tx:attributes>

</tx:advice>

<!-- 配置参与事务的类 -->

<aop:config>

<aop:pointcut id="allTestServiceMethod" expression="execution(\* com.test.testAda.test.model.service.\*.\*(..))"/>

<aop:advisor pointcut-ref="allTestServiceMethod" advice-ref="TestAdvice" />

</aop:config>

需要注意的地方:

（1） advice（建议）的命名：由于每个模块都会有自己的Advice，所以在命名上需要作出规范，初步的构想就是模块名+Advice（只是一种命名规范）。

（2） **tx:attribute标签所配置的是作为事务的方法的命名类型**。

         如<tx:method **name**="save\*" **propagation**="REQUIRED"/>

        其中\*为通配符，即代表以save为开头的所有方法，即表示符合此命名规则的方法作为一个事务。

**propagation**="REQUIRED"代表支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择。

（3） aop:pointcut标签配置参与事务的类，由于是在Service中进行数据库业务操作，配的应该是包含那些作为事务的方法的Service类。

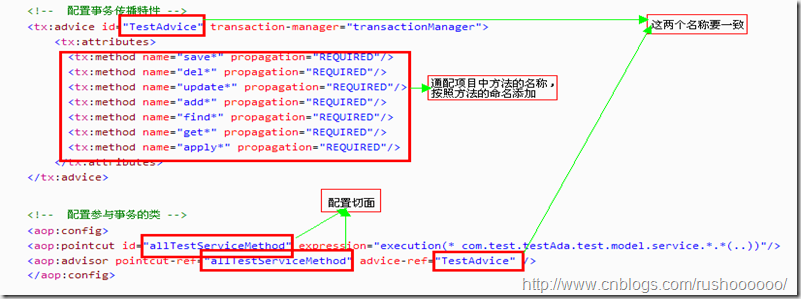
       首先应该特别注意的是id的命名，同样由于每个模块都有自己事务切面，所以我觉得初步的命名规则因为 all+模块名+ServiceMethod。而且每个模块之间不同之处还在于以下一句：

**expression**="execution(\* com.test.testAda.test.model.service.\*.\*(..))"

       其中第一个\*代表返回值，第二\*代表service下子包，第三个\*代表方法名，“（..）”代表方法参数。

（4） aop:advisor标签就是把上面我们所配置的事务管理两部分属性整合起来作为整个事务管理。

图解：

[](http://images.cnblogs.com/cnblogs_com/rushoooooo/201108/201108281004069849.png)

**下面附上配置声明式事务的一些相关的资料，以下资料均来源于互联网：**

**附一、Spring事务类型详解**

**附二、对spring事务类型详解的一点补充（关于嵌套事务）**

**附三、Transaction后缀给声明式事务管理带来的好处**

**附四、Spring中的四种声明式事务的配置**

**附一、Spring事务类型详解**

<prop key="load\*">PROPAGATION\_REQUIRED,readOnly</prop><prop key="store\*">PROPAGATION\_REQUIRED</prop>

估计有好多朋友还没有弄清楚里面的值的意思，仔细看完下面应该知道自己什么情况下面应该使用什么样的声明。^\_^

Spring中常用事务类型：

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。

**附二、对spring事务类型详解的一点补充（关于嵌套事务）**

· 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。

       可能大家对PROPAGATION\_NESTED还不怎么了解，觉得有必要再补充一下^\_^！  
PROPAGATION\_NESTED: 嵌套事务类型，是相对上面提到的六种情况（上面的六种应该称为平面事务类型），打个比方我现在有一个事务主要有一下几部分：  
      1，从A用户帐户里面减去100元钱  
      2，往B用户帐户里面添加100元钱  
       这样看和以前不同的事务可能没有什么区别，那我现在有点特殊的要求就是，A用户有3个帐户，B用户有2个帐户，现在我的要求就是只要再A用户的3个帐户里面任意一个减去100元，往B用户的两个帐户中任意一个里面增加100元就可以了！  
       一旦你有这样的要求那嵌套事务类型就非常适合你！我们可以这样理解，  
       一：将“从A用户帐户里面减去100元钱” 和 “往B用户帐户里面增加100元钱”我们暂时认为是一级事务操作  
       二：将从A用户的3个帐户的任意一个帐户里面减钱看做是“从A用户帐户里面减去100元钱”这个一级事务的子事务（二级事务），同样把后面存钱的看成是另一个的二级事务。  
      问题一：当二级事务被rollback一级事务会不会被rollback？  
      答案是不会的，二级事务的rollback只针对自己。  
      问题二：什么时候这个一级事务会commit，什么时候会被rollback呢？  
      我们主要看二级里面出现的情况，当所有的二级事务被commit了并且一级事务没有失败的操作，那整个事务就算是一个成功的事务，这种情况整个事务会被commit。  
当任意一个二级事务没有被commit那整个事务就是失败的，整个事务会被roolback。  
还是拿上面的例子来说明吧！如果我在a的三个帐户里面减钱的操作都被二级事务给rollback了，也就是3个帐户里面都没有减钱成功，整个事务就失败了就会被rollback。如果A用户帐户三个帐户里面有一个可以扣钱而且B用户的两个帐户里面也有一个帐户可以增加钱，那整个事务就算成功的，会被 commit。  
看了一下觉得上面的例子好像不是很深刻，看这个情况（A用户的3个帐户都是有信用额度的，也就是说可以超支，但是超支有金额限制）。不过原理是一样的，简单点也好说明一点，祝你好运！^\_^

**附三、Transaction后缀给声明式事务管理带来的好处**

        良好的面向对象的程序，一般都使用接口和实现分离的模式。我在《事务管理最佳实践全面解析》一文中提出，用\*Transaction和\*Dao后缀这样的形式，区分方法的不同用途。

这样，可以提醒接口的实现者和方法的使用者注意到它们对于数据库连接和事务的依赖。

实际上，使用\*Transaction后缀这样的命名方式，对于声明式事务管理也是很有用处的。如，Spring的事务管理中，我们一般使用方法名的匹配来应用声明式事务。

一、请看下面的Spring配置：

<bean id="txProxyTemplate" abstract="true" class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">

<property name="transactionManager" ref="transactionManager"/>

<property name="transactionAttributes">

<props>

<prop key="\*">readOnly</prop>

<prop key="add\*">PROPAGATION\_REQUIRED,-Exception</prop>

<prop key="save\*">PROPAGATION\_REQUIRED,-Exception</prop>

<prop key="modify\*">PROPAGATION\_REQUIRED,-Exception</prop>

<prop key="update\*">PROPAGATION\_REQUIRED,-Exception</prop>

<prop key="delete\*">PROPAGATION\_REQUIRED,-Exception</prop>

<prop key="remove\*">PROPAGATION\_REQUIRED,-Exception</prop>

<prop key="query\*">PROPAGATION\_REQUIRED, readOnly,-Exception</prop>

<prop key="load\*">PROPAGATION\_REQUIRED, -Exception</prop>

</props>

</property>

</bean>

这是来自于真实项目中的Spring声明式事务配置。我们对每一个业务层的实现类都应用了这样的事务配置。

我们对所有业务服务Service方法使用了只读事务。对以add,save,modify,update,delete,remove,load开头的方法都使用了事务。

但是，实际上，虽然我们开发的软件一个“信息管理系统”，是围绕数据库开发的。但是，在Service层，我们还是有很多不操作数据库的方法。

如，单纯根据业务逻辑进行计算的，适用缓存进行计算的，执行email发送，文件上传等等任务的方法，在这种配置下都不分青红皂白的应用了事务。

SpringAOP生成的代理对象代理了我们的服务实现类，所有的方法执行前后都被拦截，用来得到和关闭数据库连接，设置、提交和回滚事务。而不管这个方法是否用到了这个数据库。

如果遵照我提出的这个方法，使用\*Transaction后缀来标识需要处理事务的方法，那么我们使用Spring声明式事务时，就可以非常精确、有效的应用事务了！

二、请看下面的Spring事务配置：

<!-- UninstallWcmsJbpmProcessDefinition -->

<bean id="uninstallWcmsJbpmProcessDefinition" parent="txProxyTemplate">

<property name="target">

<ref bean="uninstallWcmsJbpmProcessDefinitionTarget"/>

</property>

<property name="transactionAttributes">

<props>

<prop key="uninstall\*Wcms\*Transaction">PROPAGATION\_REQUIRED,-Exception</prop>

</props>

</property>

</bean>

我们对这个类中以uninstall开头，中间包含Wcms，最后以Transaction结尾，这样的规则命名的方法，应用了事务。

三、部分源代码：

（一）2个应用了Spring声明式事务的方法：

|  |
| --- |
| /\*\*  \*使用SPring的ibatis，主要要配置iBatis的Spring声明式事务。  \*@throwsException  \*<prop key="uninstall\*Wcms\*Transaction">PROPAGATION\_REQUIRED,-Exception</prop>  \*1，还要删除所有 频道---新闻--工作流表中标记不为1的记录。  \*/  publicvoid uninstallAllWcmsProcessDefinitionsTransaction() throws Exception{  /\*\*  \*  \*  \*/  this.getWcmsSystemChannelProcessdefinitionDao().deleteAll();  this.getWcmsSystemChannelNewsinfoDao().deleteAllProcessingWcmsSystemChannelNewsinfoModule();      }  /\*\*  \*<prop key="uninstall\*Wcms\*Transaction">PROPAGATION\_REQUIRED,-Exception</prop>  \*@paramname  \*@throwsException  \*/  publicvoid uninstallWcmsSystemChannelProcessdefinitionTransaction(String name) throws Exception{  this.getWcmsSystemChannelProcessdefinitionDao().deleteByProcessdefinitionName(name);  this.getWcmsSystemChannelNewsinfoDao().deleteAllProcessingWcmsSystemChannelNewsinfoModuleByProcessdefinitionName(name);      }  （二）用到的Dao类，用来实际访问数据库的2个DAO对象。    /\*\*    \*SPring管理的ibatis功能    \*/    private IWcmsSystemChannelProcessdefinitionDao wcmsSystemChannelProcessdefinitionDao;    private IWcmsSystemChannelNewsinfoDao wcmsSystemChannelNewsinfoDao; |

**附四、Spring中的四种声明式事务的配置**

让我们言归正传吧。

以下两个bean的配置是下面要用到的。

<!-- 定义事务管理器（声明式的事务） -->

<bean id="transactionManager"

class="org.springframework.orm.hibernate3.HibernateTransactionManager">

<property name="sessionFactory">

<ref local="sessionFactory" />

</property>

</bean>

<!-- \*\*\*\*\*\*\*业务逻辑层(是对各个DAO层的正面封装)主要用到<<门面模式>>\*\*\*\*\*\* -->

<bean id="fundService"

class="com.jack.fund.service.serviceimpl.FundService">

<property name="operdao">

<ref bean="operatorDAO" />

</property>

<property name="producedao">

<ref bean="fundProduceDAO" />

</property>

<property name="customerdao">

<ref bean="customerDAO" />

</property>

<property name="accountdao">

<ref bean="accountDAO" />

</property>

<property name="fundaccountdao">

<ref bean="fundAccountDAO" />

</property>

<property name="fundtransdao">

<ref bean="fundTransDAO" />

</property>

</bean>

可能还有其他很多模块。<bean id="fundService"/>可能只是其中的模块。

第一种：配置声明式事务的方法如下。也是我们最常用的方法了，它适用于你的库表比较少的情况下。

<bean id="fundServiceDAOProxy"

class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">

<!-- 配置事务管理器 -->

<property name="transactionManager">

<ref bean="transactionManager" />

</property>

<!-- 此属性指定目标类本省是否是代理的对象，如果目标类没有实现任何类，就设为true代表自己 -->

<property name="proxyTargetClass">

<value>false</value>

</property>

<property name="proxyInterfaces">

<value>com.jack.fund.service.IFundService</value>

</property>

<!-- 目标bean -->

<property name="target">

<ref bean="fundService" />

</property>

<!-- 配置事务属性 -->

<property name="transactionAttributes">

<props>

<prop key="delete\*">PROPAGATION\_REQUIRED</prop>

<prop key="add\*">PROPAGATION\_REQUIRED</prop>

<prop key="update\*">PROPAGATION\_REQUIRED</prop>

<prop key="save\*">PROPAGATION\_REQUIRED</prop>

<prop key="find\*">PROPAGATION\_REQUIRED,readOnly</prop>

</props>

</property>

</bean>

以下可能还有其他的xxxServiceDAOProxy.大家可以看出针对每一个功能模块配置一个业务代理服务。如果模块多大话，就显得代码有点多了，发现他们只是稍微一点不一样。这时我们就应该想到继承的思想。用第二种方法。

第二种：配置声明式事务的方法如下。这种情况适合相对比较多的模块时使用。

<!-- 利用继承的思想简化配置，要把abstract="true" -->

<bean id="transactionBase"

class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean"

lazy-init="true" abstract="true">

<!-- 配置事务管理器 -->

<property name="transactionManager">

<ref bean="transactionManager" />

</property>

<!-- 配置事务属性 -->

<property name="transactionAttributes">

<props>

<prop key="delete\*">PROPAGATION\_REQUIRED</prop>

<prop key="add\*">PROPAGATION\_REQUIRED</prop>

<prop key="update\*">PROPAGATION\_REQUIRED</prop>

<prop key="save\*">PROPAGATION\_REQUIRED</prop>

<prop key="find\*">PROPAGATION\_REQUIRED,readOnly</prop>

</props>

</property>

</bean>

而具体的模块可以简单的这样配置。只要指明它的parent（父类）就可以了。父类一般把abstract="true"，因为在容器加载的时候不需要初始化，等到用的时候再有它的子类调用的时候，再去初始化。

<bean id="fundServiceDAOProxy" parent="transactionBase" >

<property name="target">

<ref bean="fundService" />

</property>

</bean>

这样配置的话，如果有多个像fundService这样模块时，可以少些很多重复的代码。

第三种：配置声明式事务的方法如下。主要利用BeanNameAutoProxyCreator自动创建事务代理

<bean id="transactionInterceptor"

class="org.springframework.transaction.interceptor.TransactionInterceptor">

<property name="transactionManager">

<ref bean="transactionManager" />

</property>

<!-- 配置事务属性 -->

<property name="transactionAttributes">

<props>

<prop key="delete\*">PROPAGATION\_REQUIRED</prop>

<prop key="add\*">PROPAGATION\_REQUIRED</prop>

<prop key="update\*">PROPAGATION\_REQUIRED</prop>

<prop key="save\*">PROPAGATION\_REQUIRED</prop>

<prop key="find\*">PROPAGATION\_REQUIRED,readOnly</prop>

</props>

</property>

</bean>

<bean

class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">

<property name="beanNames">

<list>

<value>fundService</value>

</list>

</property>

<property name="interceptorNames">

<list>

<value>transactionInterceptor</value>

</list>

</property>

</bean>

这种方法主要利用了拦截器的原理。

前三种方法一般都必需指定具体的模块bean.如果模块过多话，比如一个大型的网站一般有几十个模块。我们就得考虑用第四种的配置方式了。自动创建事务代理的方式了。

第四种：配置声明式事务的方法如下。

<bean id="transactionInterceptor"

class="org.springframework.transaction.interceptor.TransactionInterceptor">

<property name="transactionManager">

<ref bean="transactionManager" />

</property>

<!-- 自动代理 -->

<bean id="autoproxy"

class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">

<!-- 可以是Service或DAO层（最好是针对业务层\*Service） -->

<property name="beanNames">

<list>

<value>\*Service</value>

</list>

</property>

<property name="interceptorNames">

<list>

<value>transactionInterceptor</value>

</list>

</property>

</bean>

自动代理还有一种用法就是结合正规表达式和advice使用。

<bean id="transactionInterceptor"

class="org.springframework.transaction.interceptor.TransactionInterceptor">

<property name="transactionManager">

<ref bean="transactionManager" />

</property>

<bean id="autoProxyCreator"

class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator" />

<bean id="regexpMethodPointcutAdvisor"

class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">

<property name="advice">

<ref bean="transactionInterceptor" />

</property>

<property name="pattern">

<value>.\*</value>

</property>

</bean>

这个方法可以针对具体的模块进行拦截并进行事务处理。