## 12. Transaction Management

### 12.1 Introduction to Spring Framework transaction management

### 12.2 Advantages of the Spring Framework’s transaction support model

### 12.3 Understanding the Spring Framework transaction abstraction

Spring事务原理，事务策略是通过org.springframework.transaction.PlatformTransactionManager接口。

**public** **interface** PlatformTransactionManager {

TransactionStatus getTransaction(

TransactionDefinition definition) **throws** TransactionException;

**void** commit(TransactionStatus status) **throws** TransactionException;

**void** rollback(TransactionStatus status) **throws** TransactionException;

}

TransactionException会被任何PlatformTransactionManager接口方法在运行状态下丢出异常，

getTranscation(…)方法会返回一个TransactionStatus依托于TransactionDefintion。

TransactionDefintion接口定了以下内容：

* Isolation（独立性）：
* Propageation（传播性）：所有的代码都可以在一个事务的局域内运行
* Timeout（逾期性）：在事务逾期时间范围前，事务能够常态地运行
* Read\_only status（只读）：一个只读事务状态，只读状态是一种优化措施，详见Hibernate。

通过TransactionStatus接口提供极其简单的方式帮我们控制事务的执行以及查询事务的状态，这种场景似曾相识，让我们查看下常见的事务的API

**public** **interface** TransactionStatus **extends** SavepointManager {

**boolean** isNewTransaction();

**boolean** hasSavepoint();

**void** setRollbackOnly();

**boolean** isRollbackOnly();

**void** flush();

**boolean** isCompleted();

}

在JDBC, JTA, Hibernate等上面，PlatformTransactionManager的实现是最为普遍的，

假设你定义JDBC DataSource

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">

<property name="driverClassName" value="${jdbc.driverClassName}" />

<property name="url" value="${jdbc.url}" />

<property name="username" value="${jdbc.username}" />

<property name="password" value="${jdbc.password}" />

</bean>

关联到PlatformTransactionManager上通过成员变量DataSource

<bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

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

</bean>

假设你采用JTA方式获取DataSource

通过JNDI配合JtaTransactionManager结合

<?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:jee="http://www.springframework.org/schema/jee"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/jee

http://www.springframework.org/schema/jee/spring-jee.xsd">

<jee:jndi-lookup id="dataSource" jndi-name="jdbc/jpetstore"/>

<bean id="txManager" class="org.springframework.transaction.jta.JtaTransactionManager" />

*<!-- other <bean/> definitions here -->*

</beans>

JtaTransactionManager不需要知道关于DataSource，因为它用了全局事务管理架构

假设你采用的是Hibernate方式

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

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

<property name="mappingResources">

<list>

<value>org/springframework/samples/petclinic/hibernate/petclinic.hbm.xml</value>

</list>

</property>

<property name="hibernateProperties">

<value>

hibernate.dialect=${hibernate.dialect}

</value>

</property>

</bean>

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

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

</bean>

### 12.4 Synchronizing resources with transactions

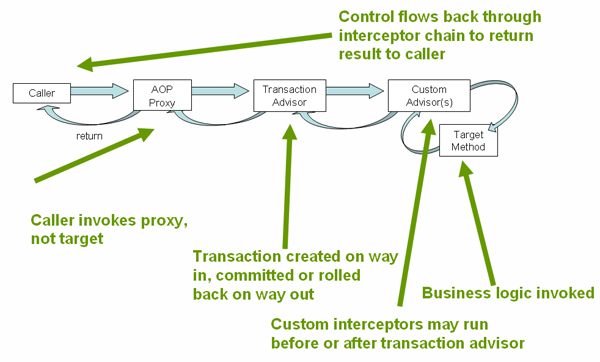
#### 12.4.1 High-level synchronization approach

Spring提倡使用ORM api或者采用JDBCTemplate

#### 12.4.2 Low-level synchronization approach

### 12.5 Declarative transaction management（声明式事务管理）

#### 12.5.1 Understanding the Spring Framework’s declarative transaction implementation



#### 12.5.2 Example of declarative transaction implementation

**package** x.y.service;

**public** **interface** FooService {

Foo getFoo(String fooName);

Foo getFoo(String fooName, String barName);

**void** insertFoo(Foo foo);

**void** updateFoo(Foo foo);

}

**public** **class** DefaultFooService **implements** FooService {

**public** Foo getFoo(String fooName) {

**throw** **new** UnsupportedOperationException();

}

**public** Foo getFoo(String fooName, String barName) {

**throw** **new** UnsupportedOperationException();

}

**public** **void** insertFoo(Foo foo) {

**throw** **new** UnsupportedOperationException();

}

**public** **void** updateFoo(Foo foo) {

**throw** **new** UnsupportedOperationException();

}

}

假设FooService接口，在事务的情况下，get方法在只读的场景中，其它的方法是读写的场景下。

*<!-- from the file 'context.xml' -->*

<?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/spring-beans.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop.xsd">

*<!-- this is the service object that we want to make transactional -->*

<bean id="fooService" class="x.y.service.DefaultFooService"/>

*<!-- the transactional advice (what 'happens'; see the <aop:advisor/> bean below) -->*

<tx:advice id="txAdvice" transaction-manager="txManager">

*<!-- the transactional semantics... -->*

<tx:attributes>

*<!-- all methods starting with 'get' are read-only -->*

<tx:method name="get\*" read-only="true"/>

*<!-- other methods use the default transaction settings (see below) -->*

<tx:method name="\*"/>

</tx:attributes>

</tx:advice>

*<!-- ensure that the above transactional advice runs for any execution*

*of an operation defined by the FooService interface -->*

<aop:config>

<aop:pointcut id="fooServiceOperation" expression="execution(\* x.y.service.FooService.\*(..))"/>

<aop:advisor advice-ref="txAdvice" pointcut-ref="fooServiceOperation"/>

</aop:config>

*<!-- don't forget the DataSource -->*

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">

<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>

<property name="url" value="jdbc:oracle:thin:@rj-t42:1521:elvis"/>

<property name="username" value="scott"/>

<property name="password" value="tiger"/>

</bean>

*<!-- similarly, don't forget the PlatformTransactionManager -->*

<bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

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

</bean>

*<!-- other <bean/> definitions here -->*

</beans>