Spring的事务管理机制

# Spring中的事务管理

作为企业级应用程序框架，Spring在不同的事务管理API之上定义了一个抽象层。而应用程序开发人员不必了解底层的事务管理API，就可以使用Spring的事务管理机制。

Spring既支持**编程式事务管理**(也称编码式事务)，也支持**声明式的事务管理。**

Spring的事务管理：**编程式**和**声明式(AOP)**。

**编程式事务管理**：将事务管理代码嵌入到业务方法中来控制事务的提交和回滚，在编程式事务中，必须在每个业务操作中包含**额外的事务管理代码**。

**声明式事务管理**：大多数情况下比编程式事务管理更好用。它将事务管理代码从业务方法中分离出来，**以声明的方式来实现事务管理**。**事务管理作为一种横切关注点，可以通过AOP方法模块化。Spring通过Spring AOP框架支持声明式事务管理。**

# Spring的事务管理器

**Spring并不直接管理事务，而是提供了多种事务管理器**，它们将事务管理的职责委托给JTA或其他持久化机制所提供的平台相关的事务实现。每个事务管理器都会充当某一特定平台的事务实现的门面，这使得**用户在Spring中使用事务时，几乎不用关注实际的事务实现是什么**。

Spring提供了许多内置事务管理器实现：

### **DataSourceTransactionManager**：位于org.springframework.jdbc.datasource包中，**数据源事务管理器**，提供对**单个javax.sql.DataSource**事务管理，用于Spring JDBC抽象框架、iBATIS或MyBatis框架的事务管理；

### **JdoTransactionManager**：位于org.springframework.**orm.jdo**包中，提供对**单个javax.jdo.PersistenceManagerFactory事务管理**，用于集成JDO框架时的事务管理；

### **JpaTransactionManager**：位于org.springframework.orm.jpa包中，提供对**单个javax.persistence.EntityManagerFactory**事务支持，用于**集成JPA**实现框架时的事务管理；

### **HibernateTransactionManager**：位于org.springframework.orm.hibernate3包中，提供对单个org.hibernate.SessionFactory事务支持，用于集成Hibernate框架时的事务管理；该事务管理器只支持Hibernate3+版本，且Spring3.0+版本只支持Hibernate 3.2+版本；

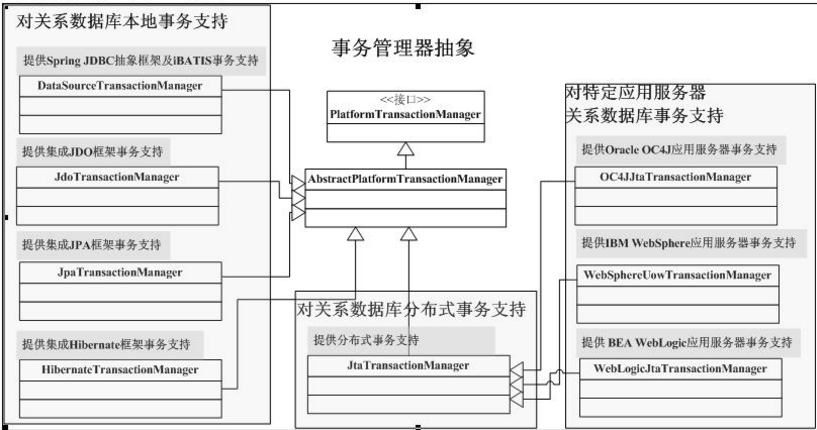
### **JtaTransactionManager**：位于org.springframework.transaction.jta包中，提供**对分布式事务管理的支持**，并将事务管理委托给**Java EE应用服务器事务管理器**；

### **OC4JjtaTransactionManager**：位于org.springframework.transaction.jta包中，Spring提供的对OC4J10.1.3+应用服务器事务管理器的适配器，此适配器用于**对应用服务器提供的高级事务的支持**；

### **WebSphereUowTransactionManager**：位于org.springframework.**transaction.jta**包中，Spring提供的对WebSphere 6.0+应用服务器事务管理器的适配器，此适配器用于**对应用服务器提供的高级事务的支持**；

### WebLogicJtaTransactionManager：位于org.springframework.**transaction.jta**包中，Spring提供的对**WebLogic 8.1+应用服务器事务管理器的适配器**，此适配器用于对应用服务器提供的高级事务的支持。

Spring不仅提供这些事务管理器，还提供对如JMS事务管理的管理器等，Spring提供一致的事务抽象如图9-1所示。



Spring的事务管理器

# 如何在Spring配置文件中定义事务管理器：

声明对本地事务的支持：

### JDBC及iBATIS、MyBatis框架事务管理器

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

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

</bean>

通过dataSource属性指定需要事务管理的**单个javax.sql.DataSource对象**。在幕后DataSourceTransactionManager通过调用java.sql.Connection来管理事务,而后者是通过DataSource获取到的。通过调用连接的commit()方法来提交事务。同样，事务失败时通过调用rollback()方法进行回滚。

### Jdo事务管理器

<bean id="txManager" class="org.springframework.orm.jdo.JdoTransactionManager">

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

</bean>

通过persistenceManagerFactory属性指定需要事务管理的javax.jdo.PersistenceManagerFactory对象。

### Jpa事务管理器

<bean id="txManager" class="org.springframework.orm.jpa.JpaTransactionManager">

<property name="**entityManagerFactory**" ref="entityManagerFactory"/>

</bean>

通过**entityManagerFactory**属性指定需要事务管理的javax.persistence.EntityManagerFactory对象。

还需要为entityManagerFactory对象指定**jpaDialect**属性，该属性所对应的对象指定了如何获取连接对象、开启事务、关闭事务等事务管理相关的行为。

<bean id="entityManagerFactory" class="org.springframework.orm.jpa.**LocalContainerEntityManagerFactoryBean**">

……

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

</bean>

<bean id="jpaDialect" class="org.springframework.orm.jpa.vendor.**HibernateJpaDialect**"/>

### Hibernate事务管理器

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

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

</bean>

在幕后HibernateTransactionManager将事务管理的职责委托给org.hibernate.Transaction对象，而后者是从Hibernate Session中获取到的。当事务成功完成时，HibernateTransactionManager将会调用Transaction对象的commit()方法来提交事务。同样，**事务失败时通过调用Transaction的rollback()方法进行回滚**。

# Spring对全局事务的支持

## JTA事务管理器

<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-3.0.xsd

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

http://www.springframework.org/schema/jee/spring-jee-3.0.xsd">

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

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

<property name="transactionManagerName" value=" java:comp/TransactionManager"/>

</bean>

</beans>

“dataSource”Bean表示从JNDI中获取的数据源，而txManager是JTA事务管理器，其中属性transactionManagerName指定了JTA事务管理器的JNDI名字，从而将事务管理委托给该事务管理器。

# 定义事务属性

在Spring中，**声明式事务**是通过**事务属性**来定义的，事务属性描述了**事务策略如何应用到方法上**。事务属性包含了5个方面，尽管Spring提供了多种声明式事务的机制，但是所有的方式都依赖这**五个参数**来控制**如何管理事务策略**。声明式事务通过**传播行为，隔离级别，只读提示，事务超时及回滚规则**来进行定义。

## Spring事务的传播行为：

**当事务方法被另一个事务方法调用时，必须指定事务应该如何传播**。例如：方法可能继续在现有事务中运行，也可能开启一个新事务，并在自己的事务中运行。

事务的传播行为可以由传播属性指定。Spring定义了7种传播行为：

|  |  |
| --- | --- |
| Spring支持的事务传播行为 | |
| 传播行为 | 含义 |
| PROPAGATION\_MANDATORY | 表示该方法必须在事务中运行，如果当前事务不存在，则会抛出一个异常 |
| PROPAGATION\_NESTED | 表示如果当前已经存在一个事务，那么该方法将会在嵌套事务中运行。嵌套的事务可以独立于当前事务进行单独地提交或回滚。如果当前事务不存在，那么其行为与PROPAGATION\_REQUIRED一样。注意各厂商对这种传播行为的支持是有所差异的。可以参考资源管理器的文档来确认它们是否支持嵌套事务 |
| PROPAGATION\_NEVER | 表示当前方法不应该运行在事务上下文中。如果当前正有一个事务在运行，则会抛出异常 |
| PROPAGATION\_NOT\_SUPPORTED | 表示该方法不应该运行在事务中。如果存在当前事务，在该方法运行期间，当前事务将被挂起。如果使用JTATransactionManager的话，则需要访问TransactionManager |
| **PROPAGATION\_REQUIRED** | **表示当前方法必须运行在事务中**。如果当前事务存在，方法将会在该事务中运行。否则，会启动一个新的事务 |
| PROPAGATION\_REQUIRED\_NEW | 表示当前方法必须运行在它自己的事务中。一个新的事务将被启动。如果存在当前事务，在该方法执行期间，当前事务会被挂起。如果使用JTATransactionManager的话，则需要访问TransactionManager |
| PROPAGATION\_SUPPORTS | 表示当前方法不需要事务上下文，但是如果存在当前事务的话，那么该方法会在这个事务中运行 |

其中PROPAGATION\_REQUIRED为默认的传播属性

## Spring事务的隔离级别

隔离级别定义了一个事务可能受其他并发事务影响的程度。在典型的应用程序中，多个事务并发运行，经常会操作相同的数据来完成各自的任务。并发，虽然是必须的，可是会导致下面的问题。

并发事务所导致的问题可以分为以下三类：

① **脏读(Dirty reads)**：脏读发生在一个事务读取了另一个事务改写但尚未提交的数据时。如果改写在稍后被回滚了，那么第一个事务获取的数据就是无效的。

② **不可重复读(Nonrepeatable read)**：不可重复读发生在一个事务执行相同的查询两次或两次以上，但是每次都得到不同的数据时。这通常是因为**另一个并发事务在两次查询期间更新了数据。**

③ **幻读(Phantom read)**：幻读与不可重复读类似。它发生在一个事务(T1)读取了几行数据，接着另一个并发事务(T2)插入了一些数据时。在随后的查询中，第一个事务(T1)就会发现多了一些原本不存在的记录。

**Spring事务的隔离级别**

1. ISOLATION\_DEFAULT： 这是一个PlatfromTransactionManager默认的隔离级别，使用数据库默认的事务隔离级别.

**另外四个与JDBC的隔离级别相对应**

2. **ISOLATION\_READ\_UNCOMMITTED**： 这是事务最低的隔离级别，它充许令外一个事务可以看到这个事务未提交的数据。

这种隔离级别会产生脏读，不可重复读和幻读。

3. **ISOLATION\_READ\_COMMITTED**： 保证一个事务修改的数据提交后才能被另外一个事务读取。另外一个事务不能读取该事务未提交的数据

4. **ISOLATION\_REPEATABLE\_READ**： 这种事务隔离级别可以防止脏读，不可重复读。但是可能出现幻像读。

它除了保证一个事务不能读取另一个事务未提交的数据外，还保证了避免下面的情况产生(不可重复读)。

5. **ISOLATION\_SERIALIZABLE** 这是花费最高代价但是最可靠的**事务隔离级别**。事务被处理为顺序执行。

除了防止脏读，不可重复读外，还避免了幻像读。

Spring事务的只读

“只读事务”并不是一个强制选项，它只是一个“暗示”，提示数据库驱动程序和数据库系统，**这个事务并不包含更改数据的操作**，那么JDBC驱动程序和数据库就有可能根据这种情况对该事务进行一些特定的优化，比方说**不安排相应的数据库锁，以减轻事务对数据库的压力，毕竟事务也是要消耗数据库的资源的**。“只读事务”仅仅是一个性能优化的推荐配置而已，并非强制你要这样做不可。

## Spring事务的事务超时

为了使应用程序更好的运行，事务不能运行太长的时间。因此，**声明式事务的第四个特性就是超时**。

## Spring事务的回滚规则

默认情况下，事务只有在遇到运行期异常时才会回滚，而在遇到**检查型异常**时不会回滚，但是也可以声明事务在遇到特定的检查型异常时像遇到运行期异常那样回滚。同样，**你还可以声明事务遇到特定的异常不回滚，即使这些异常是运行期异常**。