68_事务的基础知识筑基（二）：Spring 的事务支持以及传播特性

1、开篇

上节课通过 Spring 自定义验证的方法，将 Order 中 statrtDate 和 endDate 的验证方式进行类改造。分别进行了 IsDate 注释接口定义，DateValidator 类定义，并将 IsDate 注释到 Order 类的 statrtDate 和 endDate 字段上。本节课会思考，如何解决订单、优惠券、积分的数据一致性问题。今天课程的内容包括以下几个部分：

- Spring 对事务的支持
- Spring 事务的声明方式
- Spring 事务的传播特性

2、Spring 对事务的支持

Spring 提供了三个接口对事务进行管理和支持，分别是 TransactionDefinition:定义了事务的属性；TransactionStatus:描述事务具体的运行状态；PlatFormTransactionManager: 事务管理器。下面就具体看看它们做了哪些具体工作：

TransactionDefinition:定义了事务的属性，如图 1 所示，TransactionDefinition 接口包括：事务传播行为；事务隔离级别；事务超时；是否是只读等信息。

```java
public interface TransactionDefinition {
    int PROPAGATION_REQUIRED = 0;
    int PROPAGATION_SUPPORTS = 1;
    int PROPAGATION_MANDATORY = 2;
    int PROPAGATION_REQUIRES_NEW = 3;
    int PROPAGATION_NOT_SUPPORTED = 4;
    int PROPAGATION_NEVER = 5;
    int PROPAGATION_NESTED = 6;
    int ISOLATION_DEFAULT = -1;
    int ISOLATION_READ_UNCOMMITTED = 1;
    int ISOLATION_READ_COMMITTED = 2;
    int ISOLATION_REPEATABLE_READ = 4;
    int ISOLATION_SERIALIZABLE = 8;
    int TIMEOUT_DEFAULT = -1;
    int getPropagationBehavior();
    int getIsolationLevel();
    int getTimeout();
    boolean isReadOnly();
    String getName();
}
```

图 1

TransactionStatus:描述事务具体的运行状态，如图 2 所示，它继承了 SavepointManager 接口。接口中定义的方法解释如下：

- boolean isNewTransaction()：判断当前事务是否是一个新事务，如果返回 false，说明当前事务已存在，或者该操作没在事务环境中。
- boolean hasSavepoint()：判断当前事务是否有保存点。
- void setRollbackOnly()：设置当前事务为 rollback-only 模式，事务管理器接收到这个指令之后会回滚事务。
- boolean isRollbackOnly()：判断当前事务是否是 rollback-only。
- void flush()：用于刷新底层会话的修改到数据库。
- boolean isCompleted()：判断当前事务是否已经结束。
- Object createSavepoint()：创建保存点。
- void rollbackToSavepoint(Object var1) ：使事务回滚到保存点上，回滚之后保存的自动释放。

- void releaseSavepoint(Object var1) ：释放指定保存点。

```
public interface TransactionStatus extends SavepointManager, Flushable {
    boolean isNewTransaction();
    boolean hasSavepoint();
    void setRollbackOnly();
    boolean isRollbackOnly();
    void flush();
    boolean isCompleted();
}
public interface SavepointManager {
    Object createSavepoint() throws TransactionException;
    void rollbackToSavepoint(Object var1) throws TransactionException;
    void releaseSavepoint(Object var1) throws TransactionException;
}
```

图 2

PlatFormTransactionManager: 事务管理器。如图 3 所示，

getTransaction 方法传入 TransactionDefinition 去创建一个事务，并返回一个 TransactionStatus 用来具体描述事务的运行状态。

Commit 方法：事务管理器根据事务的运行状态对事务进行提交操作，如果该事物被设置为 rollback-only，则将事务回滚

Rollback 方法： 事务管理器对事务进行回滚操作。

```
public interface PlatformTransactionManager {
    TransactionStatus getTransaction(TransactionDefinition var1) throws TransactionException;

    void commit(TransactionStatus var1) throws TransactionException;

    void rollback(TransactionStatus var1) throws TransactionException;
}
```

图 3

3、Spring 事务的声明方式

Spring 事务的声明方式可以通过 AOP 进行，针对 Class 和 Method 打上 @Transactional 的标签。如图 4 所示，在 BttForum 类上面打上@Transactional 的注释。

```
@Service
@Transactional
public class BbtForum{
    public ForumDao forumDao;
    public TopicDao topicDao;
    public PostDao  postDao;
    public void addTopic(Topic topic){
        topic.addTopic(topic);
        postDao.addPost(topic.getPost());
    }
    public Forum getForum(int forumId){
        return forumDao.getForum(forumId);
    }
    public void updateForum(Forum forum){
        forumDao.updateForum(forum);
    }
    public int getForumNum(){
        return forumDao.getForumNum();
    }
}
```

图 4

如图 5 所示，Spring 通过 aop:pointcut 来指定需要织入增强的切点，此处定义的是该包下所有类的所有方法，通过 tx:advice 声明增强，此处需要事务管理器，通过 tx:method 为加入事务的方法指定属性，tx:method 的属性有 name，propagation，isolation，timeout，read-only，rollback-for，non-rollback-for。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mvc="http://www.springframework.org/schem
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc-4.3.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.3.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-4.3.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-4.3.xsd ">

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>

<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="get*" propagation="REQUIRED" read-only="true"/>
        <tx:method name="update*" rollback-for="Exception"/>
        <tx:method name="update*"/>
    </tx:attributes>
</tx:advice>

<aop:config>
    <aop:pointcut expression="execution(* cn.itcast.ssm.service.impl.*.*(..))" id="serviceMethod"/>
    <aop:advisor advice-ref="txAdvice" pointcut-ref="serviceMethod"/>
</aop:config>
```


图 5

注解驱动配置后，就可以用@Transacational 注解来配置事务，该注解有以下属性可以配置：propagation，isolation，read-only，timeout，rollbackFor，noRollbackFor，rollbackForClassName,noRollBackForClassName。

@Transacational 注解可以用在类上，代表该类所有的方法都运行在事务环境中，也可以单独配置在方法上，给方法设置特定的事务环境。但是一般不标注在接口或抽象类上，因为@Transacational 不能被继承。

4、Spring 事务的传播特性

事务传播特性就是，事务中嵌套其他的事务，事务与被嵌套事务之间如何相互影响，如何执行的，这就是事务传播性。来看一个例子如图 6 所示，ServiceA 中定义了 methodA 方法，在该方法打上了@Transactional 的注释表明支持事务，同时在 methodA 方法体内部调用 methodB。methodB 方法在 ServiceB 中定义了，并

且也是支持事务的。也就是说 methodA 方法本身支持事务，嵌套在其中的 methodB 也支持事务。

```
public class ServiceA {
    @Autowired
    private ServiceB b;
    @Transactional
    public void methodA() {
        // 数据库操作
        for(int i = 0; i < 51; i++) {
        try {
                b.methodB();
            } catch(Exception e) {
            // 打印异常日志
            }
        }

    }
}

public class ServiceB {
    @Transactional(propagation = PROPAGATION_REQUIRES_NEW)
    public void methodB() throws Exception {
    // 数据库操作
    }
}
```

图 6

存在上述嵌套事务的情况那么如何描述两个方法的事务行为的互相作用和互相影响呢，这里将事务传播行为分为了 7 种，主要集中在被嵌套的方法 methodB 这端的定义，也就是@Transcational(propagation = PROPAGATION_REQUIRES_NEW) ,这句定义中 propagation 的赋值：

1. PROPAGATION_REQUIRED：这个是最常见的，就是说，如果 ServiceA.methodA 调用了 ServiceB.methodB，如果 ServiceA.methodA 开启了事务，然后 ServiceB.methodB 也声明了事务，那么 ServiceB.methodB 不会开启独立事务，而是将自己的操作放在 ServiceA.methodA 的事务中来执行，ServiceA.methodA 和 ServiceB.methodB 任何一个报错都会导致整个事务回滚。

1. PROPAGATION_SUPPORTS：如果 ServiceA.methodA 开了事务，那么 ServiceB.methodB 就将自己加入 ServiceA.methodA 中来运行，如果 ServiceA.methodA 没有开事务，那么 ServiceB.methodB 自己也不开事务
2. PROPAGATION_MANDATORY：必须被一个开启了事务的方法来调用自己，否则报错。
3. PROPAGATION_REQUIRES_NEW：ServiceB.method 强制性自己开启一个新的事务，然后 ServiceA.methodA 的事务会等待 ServiceB.methodB 事务完了再继续执行。这就是影响的回滚了，如果 ServiceA.methodA 报错了，ServiceB.methodB 是不会受到影响的，ServiceB.methodB 报错了，ServiceA.methodA 也可以选择性的回滚或者是提交。
4. PROPAGATION_NOT_SUPPORTED：就是 ServiceB.methodB 不支持事务，ServiceA.methodA 的事务执行到 ServiceB.methodB 那儿，就挂起来了，ServiceB.methodB 用非事务方式运行结束，ServiceA.methodA 事务再继续运行。这个好处就是 ServiceB.methodB 代码报错不会让 ServiceA.methodA 回滚。
5. PROPAGATION_NEVER：不能被一个事务来调用，ServiceA.methodA 开事务了，但是调用了 ServiceB.methodB 会报错。
6. PROPAGATION_NESTED：开启嵌套事务，ServiceB.methodB 开启一个子事务，如果回滚的话，那么 ServiceB.methodB 就回滚到开启子事务的这个 save point。

5、总结

本节课主要介绍 Spring 的事务支持以及传播特性，包括：Spring 对事务的支持，Spring 事务的声明方式，Spring 事务的传播特性三个部分。Spring 对事务的支持中介绍了三个重要的接口，分别负责事务属性、运行状态和事务管理。Spring 事务的声明方式以@Transactional 的注释为主，可以针对类和方法。事务的传播特性在事务嵌套调用的场景会用到，这里列举了 7 个类型供大家参考。

下节课带大家看看 Spring 事务框架源码，了解事务的开启和开启事务之后执行的业务逻辑。下期见，拜拜。