背景介绍

        目前系统正在进行代码重构前期预研工作，目标采用spring控制事务以减少开发代码量，提高开发效率。同时避免开发人员编码控制事务所带来的链接没有释放，事务没有提交，出现异常事务没有回滚的Bug。

     为保证系统能正确使用Spring控制事务，必须很好的理解其传播特性。在沟通中发现，好多人知道这个概念但是对于事务的实际传播行为往往模棱两可。

     基于上述原因，本文采用Demo实例的方式对事务的7大传播特性给出了解析。希望能明确大家对事务传播特性的认识，在以后的工作中成功使用。

Demo说明

     采用Junit4.10.0+Spring3.2.1+Spring JDBCTemplate，通过注解方式配置事务，代码层次包括主测试类，两个Service对象，事务在Service开启。

概念

本地事务

       数据库事务，默认事务为自动提交，因此如果一个业务逻辑类中有多次数据库操作将无法保证事务的一致性。

Spring事务

       对本地事务操作的一次封装，相当于把使用JDBC代码开启、提交、回滚事务进行了封装。

上述两个概念会在demo中用到，以方便大家理解代码。

传播特性

       该特性是保证事务是否开启，业务逻辑是否使用同一个事务的保证。当事务在传播过程中会受其影响。其传播特性包括：

  1、Propagation.REQUIRED

方法被调用时自动开启事务，在事务范围内使用则使用同一个事务，否则开启新事务。

  2、Propagation.REQUIRES\_NEW

无论何时自身都会开启事务

  3、Propagation.SUPPORTS

自身不会开启事务，在事务范围内则使用相同事务，否则不使用事务

  4、Propagation.NOT\_SUPPORTED

自身不会开启事务，在事务范围内使用挂起事务，运行完毕恢复事务

  5、Propagation.MANDATORY

自身不开启事务，必须在事务环境使用否则报错

  6、Propagation.NEVER

自身不会开启事务，在事务范围使用抛出异常

  7、Propagation.NESTED

如果一个活动的事务存在，则运行在一个嵌套的事务中. 如果没有活动事务, 则按TransactionDefinition.PROPAGATION\_REQUIRED 属性执行。需要JDBC3.0以上支持。

实例Demo

Propagation.REQUIRED

测试入口代码

//会开启事务，在事务范围内使用则使用同一个事务，否则开启新事务

@Test

public void testRequires(){

sService.addStudent();

}

Service代码

@Transactional(propagation = Propagation.REQUIRED)

public void addStudent(){

String sql = "insert into student(name) values('st0')";

jdbcTemplate.execute(sql);

tService.addTeacher();

throw new RuntimeException();

}

@Transactional(propagation = Propagation.REQUIRES)

public void addTeacher(){

String sql = "insert into teacher(name) values ('t5')";

jdbcTemplate.execute(sql);

}

经测试无论在tService还是sService如果不抛出异常，那么数据提交成功，如果抛出异常，数据提交失败。这说明tService和sService使用的是同一个事务，并且只要方法被调用就开启事务。

Propagation.REQUIRES\_NEW

测试入口代码

//无论何时自身都会开启事务

@Test

public void testRequiresNew(){

sService.addStudent5();

}

Service代码

@Transactional(propagation = Propagation.REQUIRES\_NEW)

public void addStudent5(){

String sql = "insert into student(name) values('st5')";

jdbcTemplate.execute(sql);

tService.addTeacher5();

throw new RuntimeException();

}

@Transactional(propagation = Propagation.REQUIRES\_NEW)

public void addTeacher5(){

String sql = "insert into teacher(name) values ('t5')";

jdbcTemplate.execute(sql);

}

经测试如果在addStudent5中抛出异常，学生数据不能正确提交，教师信息被正确提交。说明sService和tService是在两个独立的事务中运行，并且只要方法被调用就开启事务。

Propagation.SUPPORTS

测试入口代码

//自身不会开启事务，在事务范围内则使用相同事务，否则不使用事务

@Test

public void testSupport(){

sService.addStudent6();

}

Service代码

@Transactional(propagation = Propagation.SUPPORTS)

public void addStudent6(){

String sql = "insert into student(name) values('st6')";

jdbcTemplate.execute(sql);

tService.addTeacher6();

throw new RuntimeException();

}

@Transactional(propagation = Propagation.SUPPORTS)

public void addTeacher6(){

String sql = "insert into teacher(name) values ('t6')";

jdbcTemplate.execute(sql);

}

经测试如果在addStudent6中抛出异常，学生数据和教师数据都被正确提交。说明sService和tService没有被spring管理和开启事务，而是使用了本地事务，由于本地事务默认自动提交因此数据都提交成功，但它们使用的却不是同一个事务，一旦出现异常将导致数据的不一致。

Propagation.NOT\_SUPPORTED

测试入口代码

//自身不会开启事务，在事务范围内使用挂起事务，运行完毕恢复事务

@Test

public void testNotSupport(){

sService.addStudent4();

}

Service代码

@Transactional(propagation = Propagation.NOT\_SUPPORTED)

public void addStudent4(){

String sql = "insert into student(name) values('st4')";

jdbcTemplate.execute(sql);

throw new RuntimeException();

}

经测试如果在addStudent4中抛出异常，学生数据正确提交。说明sService没有被spring管理和开启事务，而是使用了本地事务，由于本地事务默认自动提交因此数据都提交成功。

测试入口代码

//自身不会开启事务，在事务范围内使用挂起事务，运行完毕恢复事务

@Test

public void testNotSupport1(){

sService.addStudent();

}

Service代码

@Transactional(propagation = Propagation.REQUIRED)

public void addStudent(){

String sql = "insert into student(name) values('st0')";

jdbcTemplate.execute(sql);

tService.addTeacher4();

}

@Transactional(propagation = Propagation.NOT\_SUPPORTED)

public void addTeacher4(){

String sql = "insert into teacher(name) values ('t4')";

jdbcTemplate.execute(sql);

throw new RuntimeException();

}

经测试如果在addTeacher4中抛出异常，学生数据提交失败，教师数据提交成功。说明sService开启了事务，tService没有开启事务，而是使用了本地事务。

Propagation.MANDATORY

测试入口代码

//自身不开启事务，必须在事务环境使用否则报错

@Test

public void testMandatory(){

sService.addStudent1();

}

Service代码

@Transactional(propagation = Propagation.MANDATORY)

public void addStudent1(){

String sql = "insert into student(name) values('st1')";

jdbcTemplate.execute(sql);

}

经测试代码报错。

org.springframework.transaction.IllegalTransactionStateException: No existing transaction found for transaction marked with propagation 'mandatory'，没有找到事务环境。

Propagation.NEVER

测试入口代码

//自身不会开启事务，在事务范围使用抛出异常

@Test

public void testNever(){

sService.addStudent();

}

Service代码

@Transactional(propagation = Propagation.REQUIRED)

public void addStudent(){

String sql = "insert into student(name) values('st0')";

jdbcTemplate.execute(sql);

tService.addTeacher3();

}

@Transactional(propagation = Propagation.NEVER)

public void addTeacher3(){

String sql = "insert into teacher(name) values ('t3')";

jdbcTemplate.execute(sql);

}

经测试代码报错，由于sService开启了事务，当调用sService方法时由于其传播特性为never，因此报存在事务错误。

org.springframework.transaction.IllegalTransactionStateException: Existing transaction found for transaction marked with propagation 'never'

Propagation.NESTED

测试入口代码

//如果没有事务环境其特性同Propagation.REQUIRED,否则嵌套运行事务

@Test

public void testNested(){

sService.addStudent2();

}

Service代码

@Transactional(propagation = Propagation.NESTED)

public void addStudent2(){

String sql = "insert into student(name) values('st2')";

jdbcTemplate.execute(sql);

tService.addTeacher2();

throw new RuntimeException();

}

@Transactional(propagation = Propagation.NESTED)

public void addTeacher2(){

String sql = "insert into teacher(name) values ('t2')";

jdbcTemplate.execute(sql);

}

经测试代码报错，教师数据和学生数据都没有提交成功。说明其按照REQUIRED特性运行。对于嵌套事务，大家可以模拟两个数据源，一方的失败不会影响另一方。

以上是所有demo解析。完整的测试代码请在：Spring事务传播特性下载。

另：

大家感兴趣SpringAOP入门和原理，可以在SpringAOP下载。