06.Spring 声明式事务

06.Spring 声明式事务

1、Spring JdbcTemplate

2、声明式事务

1、设置隔离级别（isolation）

2、事务的传播特性

3、基于xml的事务配置

1.什么是事务

> 把一组业务当成一个业务来做；要么都成功，要么都失败，保证业务操作完整性的一种数据库机制。

1、Spring JdbcTemplate

在spring中为了**更加方便的操作JDBC**，在JDBC的基础之上定义了一个抽象层，此设计的目的是为不同类型的**JDBC操作**提供**模板方法**，每个模板方法都能控制整个过程，并允许覆盖过程中的特定任务，通过这种方式，可以尽可能保留灵活性，将数据库存取的工作量降到最低。

1、配置并测试数据源

pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.2.6.RELEASE</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>druid</artifactId>
```

```
        <version>1.1.21</version>
    </dependency>

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.47</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aop</artifactId>
        <version>5.2.6.RELEASE</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjweaver</artifactId>
        <version>1.9.5</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aspects</artifactId>
        <version>5.2.6.RELEASE</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-orm</artifactId>
        <version>5.2.6.RELEASE</version>
        <scope>compile</scope>
    </dependency>

</dependencies>
```

dbconfig.properties

```
jdbc.username=root123
password=123456
url=jdbc:mysql://localhost:3306/demo
driverClassName=com.mysql.jdbc.Driver
```

applicationContext.xml

```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:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/aop
       https://www.springframework.org/schema/aop/spring-aop.xsd
">
    <context:property-placeholder location="classpath:dbconfig.properties"></context:prop
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <property name="username" value="${jdbc.username}"></property>
        <property name="password" value="${jdbc.password}"></property>
        <property name="url" value="${jdbc.url}"></property>
        <property name="driverClassName" value="${jdbc.driverClassName}"></property>
    </bean>
</beans>
```

MyTest.java

```java
import com.alibaba.druid.pool.DruidDataSource;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import java.sql.SQLException;

public class MyTest {
    public static void main(String[] args) throws SQLException {
        ApplicationContext context = new ClassPathXmlApplicationContext("jdbcTemplate.xml
        DruidDataSource dataSource = context.getBean("dataSource", DruidDataSource.class)
        System.out.println(dataSource);
        System.out.println(dataSource.getConnection());
    }
}
```

2、给spring容器添加JdbcTemplate

spring容器提供了一个JdbcTemplate类，用来方便操作数据库。

jdbcTemplate.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:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/aop
       https://www.springframework.org/schema/aop/spring-aop.xsd
">
    <context:property-placeholder location="classpath:dbconfig.properties"></context:prope
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <property name="username" value="${jdbc.username}"></property>
        <property name="password" value="${jdbc.password}"></property>
        <property name="url" value="${jdbc.url}"></property>
        <property name="driverClassName" value="${jdbc.driverClassName}"></property>
    </bean>
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <constructor-arg name="dataSource" ref="dataSource"></constructor-arg>
    </bean>
</beans>
```

MyTest.java

```
import com.alibaba.druid.pool.DruidDataSource;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;

import java.sql.SQLException;

public class MyTest {
    public static void main(String[] args) throws SQLException {

        ApplicationContext context = new ClassPathXmlApplicationContext("jdbcTemplate.xml
```

```
        JdbcTemplate jdbcTemplate = context.getBean("jdbcTemplate", JdbcTemplate.class);
        System.out.println(jdbcTemplate);
    }
}
```

3、插入数据

MyTest.java

```
import com.alibaba.druid.pool.DruidDataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;

import java.sql.SQLException;

public class MyTest {
    public static void main(String[] args) throws SQLException {
        ApplicationContext context = new ClassPathXmlApplicationContext("jdbcTemplate.xml
        JdbcTemplate jdbcTemplate = context.getBean("jdbcTemplate", JdbcTemplate.class);
        String sql = "insert into emp(empno,ename) values(?,?)";
        int result = jdbcTemplate.update(sql, 1111, "zhangsan");
        System.out.println(result);
    }
}
```

4、批量插入数据

MyTest.java

```
import com.alibaba.druid.pool.DruidDataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;

import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class MyTest {
```

```
    public static void main(String[] args) throws SQLException {
        ApplicationContext context = new ClassPathXmlApplicationContext("jdbcTemplate.xml
        JdbcTemplate jdbcTemplate = context.getBean("jdbcTemplate", JdbcTemplate.class);
        String sql = "insert into emp(empno,ename) values(?,?)";
        List<Object[]> list = new ArrayList<Object[]>();
        list.add(new Object[]{1,"zhangsan1"});
        list.add(new Object[]{2,"zhangsan2"});
        list.add(new Object[]{3,"zhangsan3"});
        int[] result = jdbcTemplate.batchUpdate(sql, list);
        for (int i : result) {
            System.out.println(i);
        }
    }
}
```

5、查询某个值，并以对象的方式返回

MyTest.java

```
import cn.tulingxueyuan.bean.Emp;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import java.sql.SQLException;

public class MyTest {
    public static void main(String[] args) throws SQLException {
        ApplicationContext context = new ClassPathXmlApplicationContext("jdbcTemplate.xml
        JdbcTemplate jdbcTemplate = context.getBean("jdbcTemplate", JdbcTemplate.class);
        String sql = "select * from emp where empno = ?";
        Emp emp = jdbcTemplate.queryForObject(sql, new BeanPropertyRowMapper<>(Emp.class)
        System.out.println(emp);
    }
}
```

6、查询返回集合对象

MyTest.java

```
import cn.tulingxueyuan.bean.Emp;
import org.springframework.context.ApplicationContext;
```

```
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import java.sql.SQLException;
import java.util.List;

public class MyTest {
    public static void main(String[] args) throws SQLException {
        ApplicationContext context = new ClassPathXmlApplicationContext("jdbcTemplate.xml
        JdbcTemplate jdbcTemplate = context.getBean("jdbcTemplate", JdbcTemplate.class);
        String sql = "select * from emp where sal > ?";
        List<Emp> query = jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(Emp.class)
        for (Emp emp : query) {
            System.out.println(emp);
        }
    }
}
```

7、返回组合函数的值

MyTest.java

```
import cn.tulingxueyuan.bean.Emp;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import java.sql.SQLException;
import java.util.List;

public class MyTest {
    public static void main(String[] args) throws SQLException {
        ApplicationContext context = new ClassPathXmlApplicationContext("jdbcTemplate.xml
        JdbcTemplate jdbcTemplate = context.getBean("jdbcTemplate", JdbcTemplate.class);
        String sql = "select max(sal) from emp";
        Double aDouble = jdbcTemplate.queryForObject(sql, Double.class);
        System.out.println(aDouble);
    }
}
```

8、使用具备具名函数的JdbcTemplate

jdbcTemplate.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:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/aop
       https://www.springframework.org/schema/aop/spring-aop.xsd
">
    <context:property-placeholder location="classpath:dbconfig.properties"></context:prope
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <property name="username" value="${jdbc.username}"></property>
        <property name="password" value="${jdbc.password}"></property>
        <property name="url" value="${jdbc.url}"></property>
        <property name="driverClassName" value="${jdbc.driverClassName}"></property>
    </bean>
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <constructor-arg name="dataSource" ref="dataSource"></constructor-arg>
    </bean>
    <bean id="namedParameterJdbcTemplate" class="org.springframework.jdbc.core.namedparam
        <constructor-arg name="dataSource" ref="dataSource"></constructor-arg>
    </bean>
</beans>
```

MyTest.java

```
import cn.tulingxueyuan.bean.Emp;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

import java.sql.SQLException;
```

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class MyTest {
    public static void main(String[] args) throws SQLException {
        ApplicationContext context = new ClassPathXmlApplicationContext("jdbcTemplate.xml
        NamedParameterJdbcTemplate jdbcTemplate = context.getBean("namedParameterJdbcTemp
        String sql = "insert into emp(empno,ename) values(:empno,:ename)";
        Map<String,Object> map = new HashMap<>();
        map.put("empno",2222);
        map.put("ename","sili");
        int update = jdbcTemplate.update(sql, map);
        System.out.println(update);
    }
}
```

9、整合EmpDao

jdbcTemplate.xml

```
    <context:component-scan base-package="cn.tulingxueyuan"></context:component-scan>
```

```
package cn.tulingxueyuan.dao;

import cn.tulingxueyuan.bean.Emp;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;

public class EmpDao {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    public void save(Emp emp){
        String sql = "insert into emp(empno,ename) values(?,?)";
        int update = jdbcTemplate.update(sql, emp.getEmpno(), emp.getEname());
        System.out.println(update);
    }
}
```

MyTest.java

```java
import cn.tulingxueyuan.bean.Emp;
import cn.tulingxueyuan.dao.EmpDao;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

import java.sql.SQLException;
import java.util.HashMap;
import java.util.Map;

public class MyTest {
    public static void main(String[] args) throws SQLException {
        ApplicationContext context = new ClassPathXmlApplicationContext("jdbcTemplate.xml
        EmpDao empDao = context.getBean("empDao", EmpDao.class);
        empDao.save(new Emp(3333,"wangwu"));
    }
}
```

2、声明式事务

ACID 四大特性

A 原子性：原子性指的是 在一组业务操作下 要么都成功 要么都失败

在一组增删改查的业务下 要么都提交 要么都回滚

C 一致性：事务前后的数据要保证数据的一致性

在一组的查询业务下 必须要保证前后关联数据的一致性

I 隔离性：在并发情况下 事物之间要相互隔离。

D 持久性：数据一旦保存就是持久性的。

总结：在事务控制方面，主要有两个分类：

编程式事务：在代码中直接加入处理事务的逻辑，可能需要在代码中显式调用beginTransaction()、commit()、rollback()等事务管理相关的方法

connetion.autoCommit(false);

---

connction.commint()

catch(){

connction.rollback();

}

声明式事务：在方法的外部添加注解或者直接在配置文件中定义，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。spring的AOP恰好可以完成此功能：事务管理代码的固定模式作为一种横切关注点，通过AOP方法模块化，进而实现声明式事务。

2、声明式事务的简单配置

Spring从不同的事务管理API中抽象出了一整套事务管理机制，让事务管理代码从特定的事务技术中独立出来。开发人员通过配置的方式进行事务管理，而不必了解其底层是如何实现的。

Spring的核心事务管理抽象是PlatformTransactionManager。它为事务管理封装了一组独立于技术的方法。无论使用Spring的哪种事务管理策略(编程式或声明式)，事务管理器都是必须的。

事务管理器可以以普通的bean的形式声明在Spring IOC容器中。下图是spring提供的事务管理器

1、在配置文件中添加事务管理器

jdbcTemplate.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:context="http://www.springframework.org/schema/context"
       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/context
       http://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/aop
       https://www.springframework.org/schema/aop/spring-aop.xsd
       http://www.springframework.org/schema/tx
       https://www.springframework.org/schema/tx/spring-tx.xsd
">
    <context:component-scan base-package="cn.tulingxueyuan"></context:component-scan>
    <context:property-placeholder location="classpath:dbconfig.properties"></context:pro
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <property name="username" value="${jdbc.username}"></property>
        <property name="password" value="${jdbc.password}"></property>
        <property name="url" value="${jdbc.url}"></property>
        <property name="driverClassName" value="${jdbc.driverClassName}"></property>
    </bean>
```

```
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
```

```
        <constructor-arg name="dataSource" ref="dataSource"></constructor-arg>
    </bean>
    <!--事务控制-->
    <!--配置事务管理器的bean-->
    <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTr
        <property name="dataSource" ref="dataSource"></property>
    </bean>
    <!--开启基于注解的事务控制模式，依赖tx名称空间-->
    <tx:annotation-driven transaction-manager="transactionManager"></tx:annotation-driven
</beans>
```

@Transactional注解应该写在哪：

@Transactional 可以标记在类上面（当前类所有的方法都运用上了事务）

@Transactional 标记在方法则只是当前方法运用事务

也可以类和方法上面同时都存在， 如果类和方法都存在@Transactional会以方法的为准。如果方法上面没有@Transactional会以类上面的为准

建议：@Transactional写在方法上面，控制粒度更细， 建议@Transactional写在业务逻辑层上，因为只有业务逻辑层才会有嵌套调用的情况。

3、事务配置的属性

isolation：设置事务的隔离级别

propagation：事务的传播行为

noRollbackFor：那些异常事务可以不回滚

noRollbackForClassName：填写的参数是全类名

rollbackFor：哪些异常事务需要回滚

rollbackForClassName：填写的参数是全类名

readOnly：设置事务是否为只读事务

timeout：事务超出指定执行时长后自动终止并回滚,单位是秒

1、设置隔离级别（isolation）

用来解决并发事务所产生一些问题：

并发： 同一个时间，多个线程同时进行请求。

什么时候会生成并发问题：在并发情况下，对同一个数据（变量、对象）进行读写操作才会产生并发问题

并发会产生什么问题?

1.脏读

2.不可重复度

3.幻影读

概念： 通过设置隔离级别可解决在并发过程中产生的那些问题：

1.脏读

事务1 begin	**事务2** begin
	update t_user set balance=800 where id=1; #balance=800
select * from t_user where id=1 commit; #balance=800	
	rollback; #回滚 #balance=1000

1. 一个事务，读取了另一个事务中没有提交的数据，会在本事务中产生的数据不一致的问题

解决方式：@Transactional(isolation = Isolation.READ_COMMITTED)

读已提交：READ COMMITTED

要求Transaction01只能读取Transaction02已提交的修改。

2.不可重复度

事务1 begin	**事务2** begin
select * from t_user where id=1 #balance=1000	
	update t_user set balance=800 where id=1; commit; #balance=800
select * from t_user where id=1 #balance=800	
commit;	

一个事务中，多次读取相同的数据， 但是读取的结果不一样， 会在本事务中产生数据不一致的问题。

解决方式：@Transactional(isolation = Isolation.REPEATABLE_READ)

可重复读：REPEATABLE READ

确保Transaction01可以多次从一个字段中读取到相同的值，即Transaction01执行期间禁止其它事务对这个字段进行更新。（行锁）

3.幻影读

事务1 begin	**事务2** begin
select sum(balance) from t_user	

where id=1 #balance=3000	
	INSERT INTO t_user VALUES ('4', '赵六', '123456784', '1000'); commit;
select sum(balance) from t_user where id=1 #balance=4000	
commit;	

一个事务中，多次对数据进行整表数据读取（统计），但是结果不一样， 会在本事务中产生数据不一致的问题。

解决方式：@Transactional(isolation = Isolation.SERIALIZABLE)

串行化：SERIALIZABLE

确保Transaction01可以多次从一个表中读取到相同的行，在Transaction01执行期间，禁止其它事务对这个表进行添加、更新、删除操作。可以避免任何并发问题，但性能十分低下。（表锁）

很多人容易搞混不可重复读和幻读，确实这两者有些相似：

对于前者， 只需要锁行

对于后者， 需要锁表

√: 可能出现 ×: 不会出现

	脏读	不可重复读	幻读
Read uncommitted	√	√	√
Read committed	×	√	√
Repeatable read	×	×	√
Serializable	×	×	×

```
1 并发安全：SERIALIZABLE>REPEATABLE_READ>READ_COMMITTED
2 运行效率：READ_COMMITTED>REPEATABLE_READ>SERIALIZABLE
```

当不设置事务隔离级别将使用数据库的默认事务隔离级别：

```sql
#MYSQL：REPEATABLE-READ
SELECT @@tx_isolation;
#ORACLE: READ_COMMITTED
SELECT s.sid, s.serial#,
    CASE BITAND(t.flag, POWER(2, 28))
        WHEN 0 THEN 'READ COMMITTED'
        ELSE 'SERIALIZABLE'
    END AS isolation_level
FROM v$transaction t
JOIN v$session s ON t.addr = s.taddr AND s.sid = sys_context('USERENV', 'SID');
```

2、事务的传播特性

事务的传播特性指的是当一个事务方法被另一个事务方法调用时，这个事务方法应该如何进行?

希望如果外部存在事务就用外部的， 外部不存在就自己开启事务

@Transactional
trans(){
    sub();
    log(); // 记录流水 数据库操作
    add();
}
@Transactional REQUIRES_NEW
log();

spring的事务传播行为：

事务传播行为类型	外部不存在事务	外部存在事务	使用方式
REQUIRED（默认）	开启新的事务	融合到外部事务中	@Transactional(propagation = Pro 适用增删改查
SUPPORTS	不开启新的事务	融合到外部事务中	@Transactional(propagation = Pro 适用查询
REQUIRES_NEW	开启新的事务	不用外部事务，创建新的事务	@Transactional(propagation = Propagation.REQUIRES_NEW) 适用内部事务和外部事务不存在业务
NOT_SUPPORTED	不开启新的事务	不用外部事务	@Transactional(propagation = Propagation.NOT_SUPPORTED) 不常用
NEVER	不开启新的事务	抛出异常	@Transactional(propagation = Pro 不常用
MANDATORY	抛出异常	融合到外部事务中	@Transactional(propagation = Pro

			不常用
NESTED	开启新的事务	融合到外部事务中,SavePoint机制，外层影响内层， 内层不会影响外层	@Transactional(propagation = Pro 不常用

变更：2021-08-04 用挂起这个词容易造成误解，直接说不用外部事务 更适当

3、超时属性(timeout)

指定事务等待的最长时间（秒）

当前事务访问数据时，有可能访问的数据被别的数据进行加锁的处理，那么此时事务就必须等待，如果等待时间过长给用户造成的体验感差。

4、设置事务只读(readOnly)

readonly:只会设置在查询的业务方法中

connection.setReadOnly(true) 通知数据库，当前数据库操作是只读，数据库就会对当前只读做相应优化

当将事务设置只读 就必须要你的业务方法里面有增删改。

如果你一次执行单条查询语句，则没有必要启用事务支持，数据库默认支持SQL执行期间的读一致性；

如果你一次执行多条查询语句，例如统计查询，报表查询，在这种场景下，多条查询SQL必须保证整体的读一致性，否则，在前条SQL查询之后，后条SQL查询之前，数据被其他用户改变，则该次整体的统计查询将会出现读数据不一致的状态，此时，应该启用事务支持（如：设置不可重复度、幻影读级别）。

对于只读事务，可以指定事务类型为readonly，即只读事务。由于只读事务不存在数据的修改，因此数据库将会为只读事务提供一些优化手段

5、异常属性

设置 当前事务出现的那些异常就进行回滚或者提交。

默认对于RuntimeException 及其子类 采用的是回滚的策略。

默认对于Exception 及其子类 采用的是提交的策略。

1、**设置哪些异常不回滚(noRollbackFor)**

2、设置哪些异常回滚（rollbackFor ）

@Transactional(timeout = 3,rollbackFor = {FileNotFoundException.class})

6、在实战中事务的使用方式

如果当前业务方法是一组 增、改、删 可以这样设置事务

@Transactional

如果当前业务方法是一组 查询 可以这样设置事务

@Transactionl(readOnly=true)
如果当前业务方法是单个 查询 可以这样设置事务
@Transactionl(propagation=propagation.SUPPORTS ,readOnly=true)

3、基于xml的事务配置

jdbcTemplate.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:context="http://www.springframework.org/schema/context"
       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/context
       http://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/aop
       https://www.springframework.org/schema/aop/spring-aop.xsd
       http://www.springframework.org/schema/tx
       https://www.springframework.org/schema/tx/spring-tx.xsd
">
    <context:component-scan base-package="cn.tulingxueyuan"></context:component-scan>
    <context:property-placeholder location="classpath:dbconfig.properties"></context:prope
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <property name="username" value="${jdbc.username}"></property>
        <property name="password" value="${jdbc.password}"></property>
        <property name="url" value="${jdbc.url}"></property>
        <property name="driverClassName" value="${jdbc.driverClassName}"></property>
    </bean>
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <constructor-arg name="dataSource" ref="dataSource"></constructor-arg>
    </bean>
    <bean id="namedParameterJdbcTemplate" class="org.springframework.jdbc.core.namedparam
        <constructor-arg name="dataSource" ref="dataSource"></constructor-arg>
    </bean>
    <!--事务控制-->
    <!--配置事务管理器的bean-->
    <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTra
```

```
33          <property name="dataSource" ref="dataSource"></property>
34      </bean>
35      <!--
36      基于xml配置的事务：依赖tx名称空间和aop名称空间
37          1、spring中提供事务管理器（切面），配置这个事务管理器
38          2、配置出事务方法
39          3、告诉spring哪些方法是事务方法（事务切面按照我们的切入点表达式去切入事务方法）
40      -->
41      <bean id="bookService" class="cn.tulingxueyuan.service.BookService"></bean>
42      <aop:config>
43          <aop:pointcut id="txPoint" expression="execution(* cn.tulingxueyuan.service.*.*(.
44          <!--事务建议：advice-ref:指向事务管理器的配置-->
45          <aop:advisor advice-ref="myAdvice" pointcut-ref="txPoint"></aop:advisor>
46      </aop:config>
47      <tx:advice id="myAdvice" transaction-manager="transactionManager">
48          <!--事务属性-->
49          <tx:attributes>
50              <!--指明哪些方法是事务方法-->
51              <tx:method name="*"/>
52              <tx:method name="checkout" propagation="REQUIRED"/>
53              <tx:method name="get*" read-only="true"></tx:method>
54          </tx:attributes>
55      </tx:advice>
56  </beans>
```

面试题：

- Spring事务的实现方式和实现原理
- 说一下Spring的事务传播行为
- 说一下 spring 的事务隔离?
- Spring框架的事务管理有哪些优点?
- 操作！！