数据层整合

本章目标

- 掌握事务分类
- 掌握Spring事务模型
- 掌握Spring事务抽象模型
- 掌握事务与资源管理
- 掌握Spring声明性事务
- 掌握Spring编程式事务
- 掌握声明性事务与编程式事务选择
- Spring事务传播性
- 数据库连接管理

事务分类

- 本地事务：local transaction，使用单一资源管理器，管理本地资源
- 全局事务：global transaction，通过事务管理器和多种资源管理器，管理多种不同类型的资源，如JDBC资源和JMS资源
- 编程式事务：通过编码方式，开启事务、提交事务、回滚事务
- 声明性事务：通过xml配置或注解，实现事务管理。Spring AOP和EJB都是声明性事务

事务分类

- JTA事务：Java Transaction API，使用javax.transaction.UserTransaction接口 ，访问多种资源管理器。JTA事

```
@Resource UserTransaction tx;

public void updateData(...) {
    ...
    // Start a transaction.
    tx.begin();
    ...
    // Perform transactional operations on data.
    ...
    // Commit the transaction.
    tx.commit();
    ...
}
```

事务分类

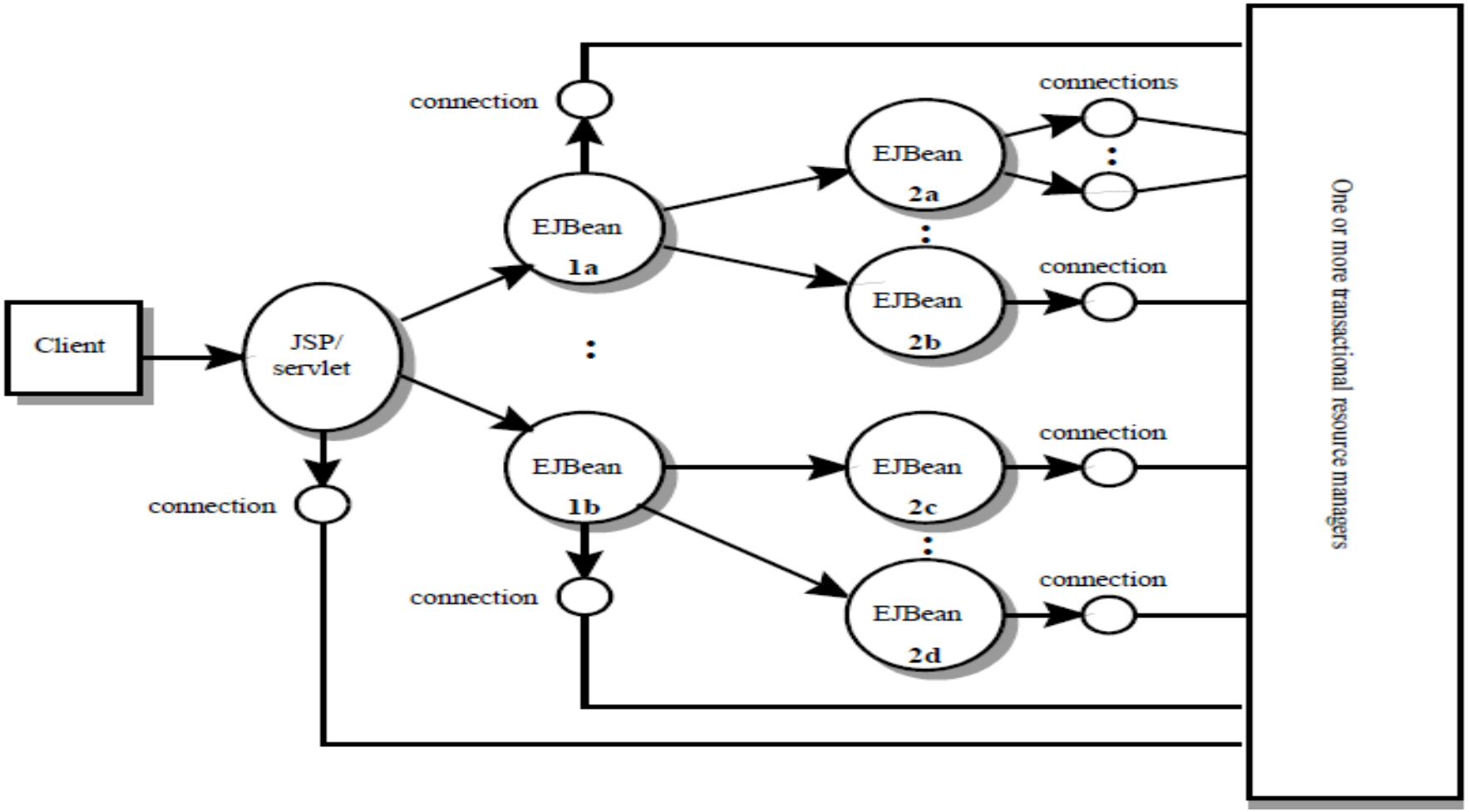
- CMT事务：Container Management transaction，通过容器自动控制事务的开启，提交和回滚。开发人员不需要手工编写代码，配置注解，由容器来控制事务的边界。

```
<enterprise-beans>

    <session>
        <ejb-name>BookBizBean</ejb-name>
        <ejb-class>com.icss.biz.BookBiz</ejb-class>
        <remote>com.icss.biz.BookRemote</remote>
        <local>com.icss.biz.BookLocal</local>
        <sesion-type>Stateless</sesion-type>
    </session>
```

```
@Stateless
public class BookBiz implements BookLocal,BookRemote{
```

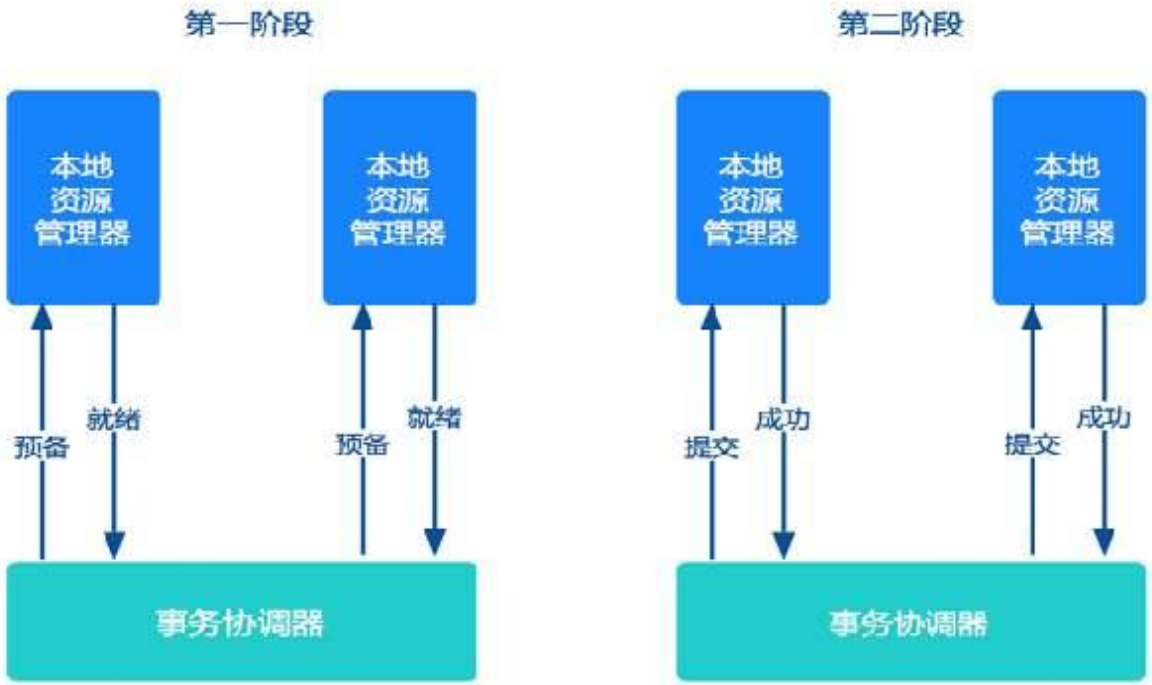
事务分类

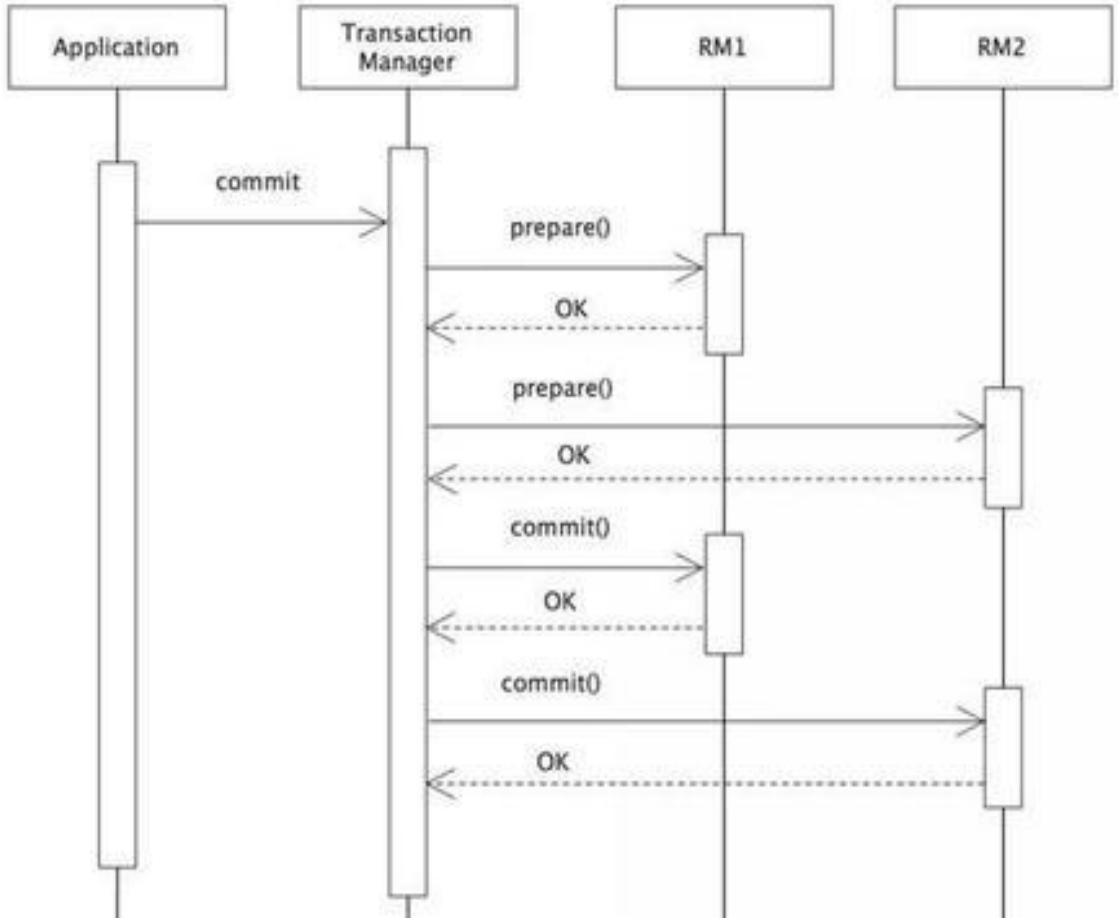
- JavaEE事务模型



事务分类

- 使用XA的两段提交协议来管理全局事务：





Spring事务模型

- Spring框架最核心的功能就是全面的事务管理功能。它提供了一致的事务管理抽象，这带来了以下好处：
  - 为复杂的事务API提供了一致的编程模型，如JTA、JDBC、Hibernate、JPA和JDO
  - 支持声明式事务管理
  - 提供比大多数复杂的事务API（诸如JTA）更简单的，更易于使用的编程式事务管理API
  - 非常好地整合各种数据访问层抽象

Spring事务模型

- 全局事务 ：
  - 有个缺陷，代码需要使用JTA， 一个笨重的API（部分是因为它的异常模型）。此外，JTA的UserTransaction通常需要从JNDI获得，这意味着我们为了JTA，需要同时使用JNDI *和* JTA。JTA需要应用服务器环境支持。基于EJB的CMT管理是声明性事务模式，EJB3之前配置过于复杂，常被人诟病。

```
public void updateData(...) {
    ...
    // Obtain the default initial JNDI context.
    Context initCtx = new InitialContext();

    // Look up the UserTransaction object.
    UserTransaction tx = (UserTransaction)initCtx.lookup(
                                "java:comp/UserTransaction");

    // Start a transaction.
    tx.begin();
    ...
    // Perform transactional operations on data.
    ...
    // Commit the transaction.
    tx.commit();
    ...
}
```

Spring事务模型

- 本地事务：
  - 本地事务容易使用，但也有明显的缺点：它们不能用于多个事务性资源。例如，使用JDBC连接事务管理的代码不能用于全局的JTA事务中。另一个缺点是局部事务为侵入式编程模型，编程繁琐。
  - Spring解决了这些问题。它使应用开发者能够在不同环境下使用一致的编程模型。你只需要写一次代码，就可以在不同的事务环境下的不同事务策略切换时享受好处。
  - Spring框架同时提供声明式和编程式事务管理，声明事务管理是多数使用者的首选。

Spring事务模型

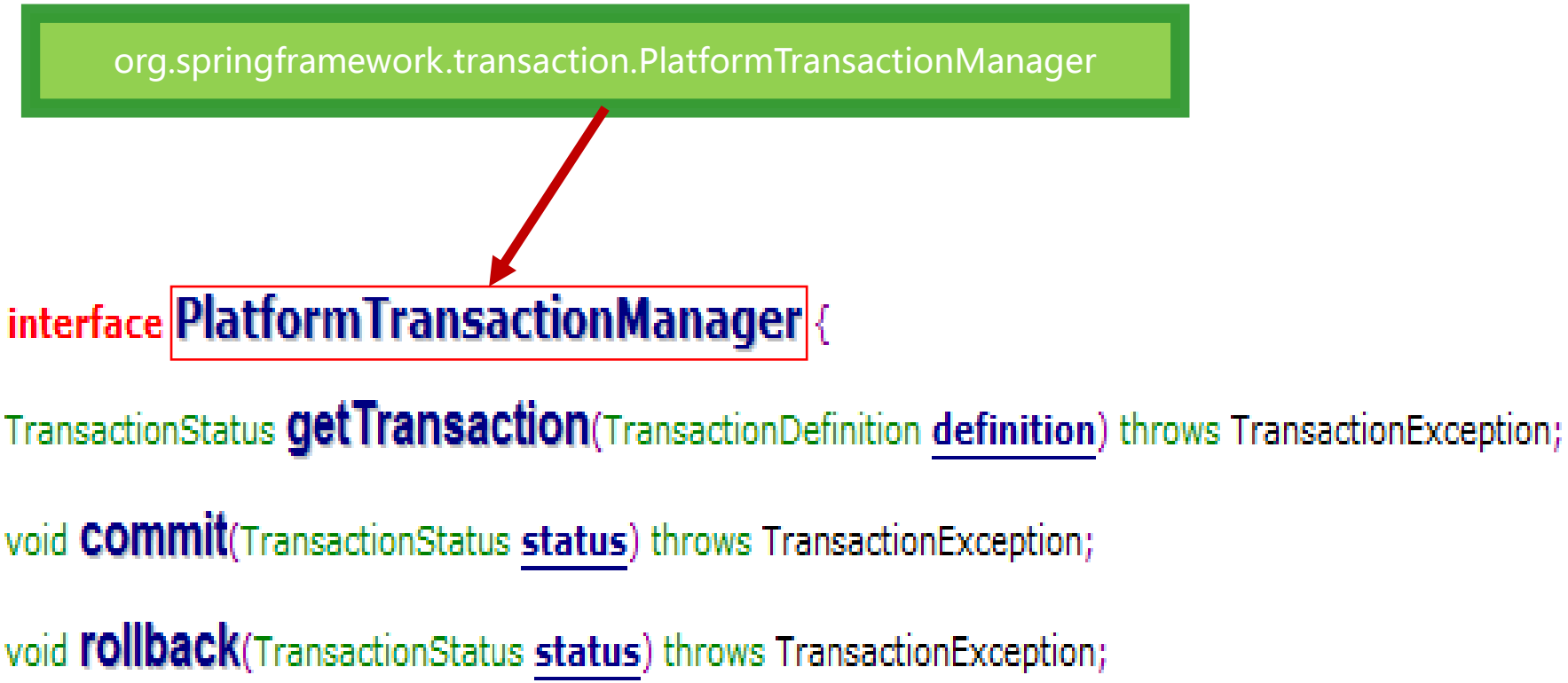
- 你还需要使用应用服务器进行事务管理吗？
  - Spring框架对事务的管理，改变了传统上认为企业级Java应用,必须使用应用服务器的认识。
  - 典型情况，只有当你需要管理多个资源的事务时，你才需要应用服务器的JTA功能。
  - Spring Framework允许你在需要时，把代码很容易地迁移到应用服务器去。用EJB的CMT或 JTA管理本地事务（如JDBC连接），将变得毫无意义。

Spring事务模型

- *总结：*
  - *Spring提供了轻量级的声明性事务方案，解决了事务对重量级应用服务器的依赖；解决了编程式事务的代码耦合；通过一致的编程模型，解决了不同事务环境迁移问题。*

Spring事务抽象模型

- Spring事务策略，由如下接口描述



Spring事务抽象模型

- TransactionStatus

```
public interface TransactionStatus extends SavepointManager, Flushable {
    boolean isNewTransaction();
    boolean hasSavepoint();
    void setRollbackOnly();
    boolean isRollbackOnly();
    void flush();
    boolean isCompleted();
```

描述事务状态

Spring事务抽象模型

- TransactionDefinition接口定义了事务的属性信息：
  - **事务隔离**：当前事务和其它事务的隔离的程度。例如，这个事务能否看到其他事务未提交的写数据。
  - **事务传播**：通常在一个事务中执行的所有代码都会在这个事务中运行。但是，如果一个事务上下文已经存在，有几个选项可以指定一个事务性方法的执行行为：例如，简单地在现有的事务中继续运行，或者挂起现有事务，创建一个新的事务。*Spring提供EJB CMT中常见的事务传播选项。*
  - **事务超时**: 事务在超时前能运行多久
  - **只读状态**: 只读事务不修改任何数据。只读事务在某些情况下（例如当使用Hibernate时），是一种非常有用的优化

Spring事务抽象模型

- 示例A：

```
<bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName"
        value="com.mysql.cj.jdbc.Driver" />
    <property name="url"
        value="jdbc:mysql://localhost:3306/staff?useSSL=false&amp;serverTimezone=UTC" />
    <property name="username" value="root" />
    <property name="password" value="123456" />
</bean>
```

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

DataSourceTransactionManager是PlatformTransactionManager的实现类，用于JDBC和myBatis访问数据库的事务配置。

Spring事务抽象模型

- 示例B：

```
<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="configLocation"
        value="classpath:hibernate.cfg.xml">
    </property>
</bean>
```

HibernateTransactionManager是PlatformTransactionManager的实现类，用于Spring管理hibernate的事务。

```
<bean id="txManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory" />
</bean>
```

Spring事务抽象模型

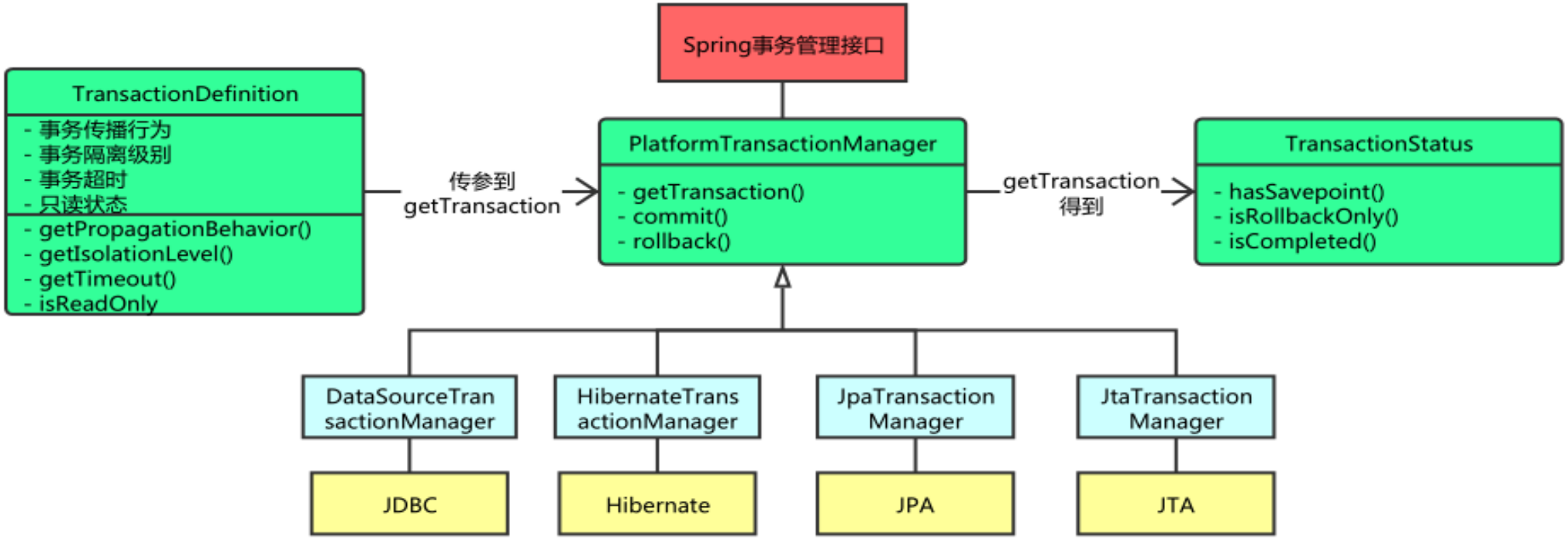
- 示例C：

```
<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" />

</beans>
```

JtaTransactionManager是PlatformTransactionManager的实现类，这用于Spring集成数据源被应用服务器管理的JTA事务。

Spring事务抽象模型

- 示例图



事务与资源管理

- 不同的事务管理器，如何管理资源？
  - 方案 1：使用高层级的抽象，把底层资源的本地API进行封装，提供模板方法。如JdbcTemplate、HibernateTemplate、JdoTemplate、SqlSessionTemplate等。
  - 方案2：直接使用资源的本地持久化API，包装类如下：
    - DataSourceUtils（用于JDBC）、EntityManagerFactoryUtils（用于JPA）、SessionFactoryUtils（用于hibernate）、PersistenceManagerFactoryUtils（用于JDO）

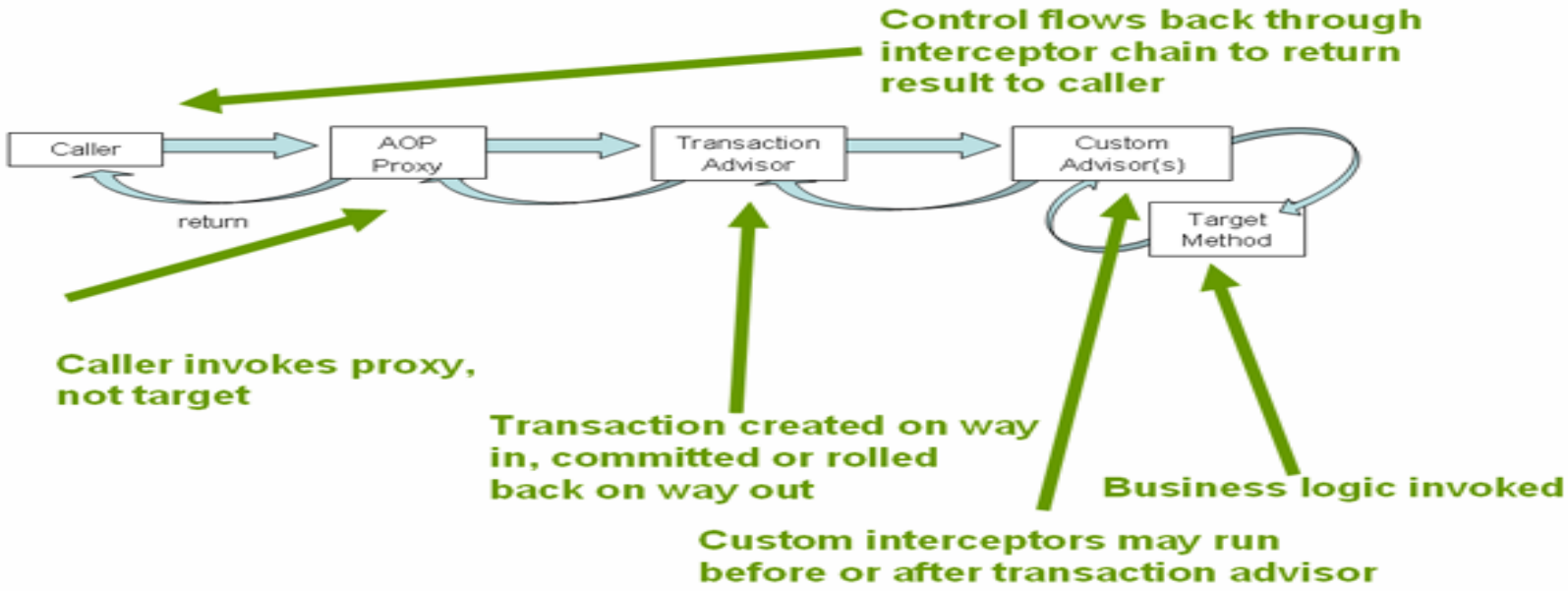
```
public abstract class DataSourceUtils {

    public static Connection getConnection(DataSource dataSource) throws CannotGetJdbcConnectionException {
        try {
            return doGetConnection(dataSource);
        }
        catch (SQLException ex) {
            throw new CannotGetJdbcConnectionException("Could not get JDBC Connection", ex);
        }
    }
```

事务与资源管理

- 总结：
  - JdbcTemplate封装JDBC操作后，代码并不简洁，开发难度增加，好处不明显
  - HibernateTemplate封装hiberante操作后，对与使用复杂的原生SQL，如复杂的多表多条件嵌套查询，HibernateTemplate的使用局限性很大，开发好处也不明显
  - 个人推荐，尽量使用原生API操作资源数据

Spring声明性事务

- 说明：
  - Spring的声明式事务管理是通过Spring AOP实现的。
  - Spring声明式事务管理可以在任何环境下使用。只需更改配置文件， 它就可以和JDBC、JDO、Hibernate或其他的事务机制一起工作。
  - 在Spring事务代理上调用方法的工作过程如下：

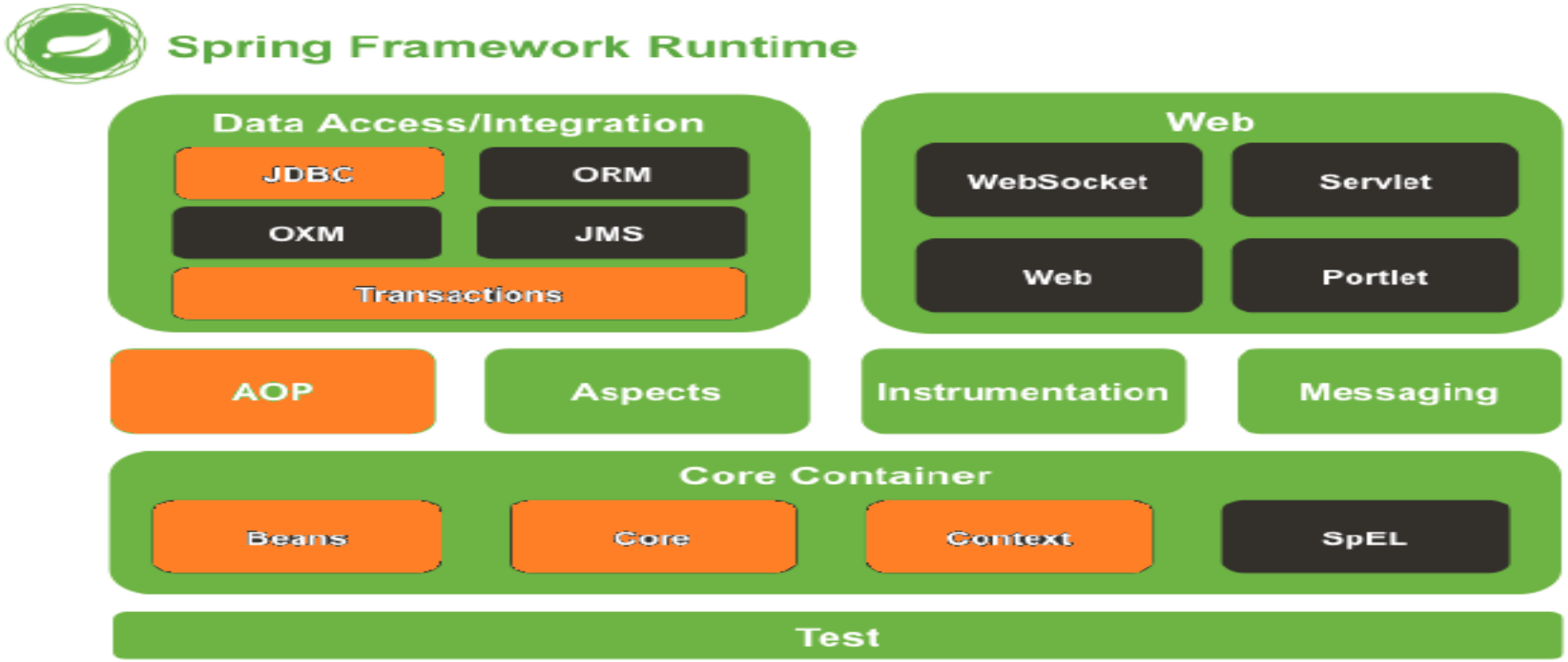


Spring声明性事务

- XML方式管理声明性事务
  - SSM开源框架中有标题，但实际内容缺失

Spring声明性事务

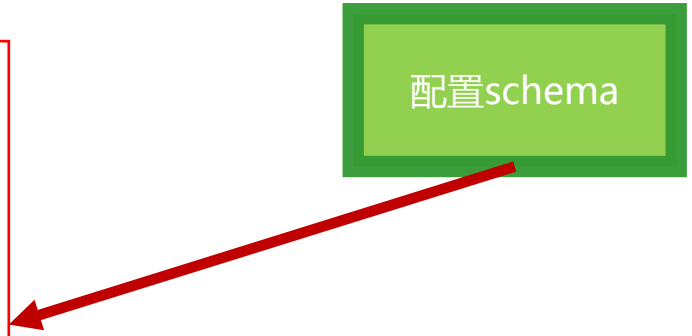
- StaffUser事务XML方案项目案例-导包
  - 导入核心包：spring-beans-4.3.19.jar，spring-core-4.3.19.jar，spring-context-4.3.19.jar，spring-context-support-4.3.19.jar
  - 导入AOP包：spring-aop-4.3.19.jar
  - 导入数据整合包：spring-jdbc-4.3.19.jar，spring-tx-4.3.19.jar
  - 导入依赖包：commons.logging-1.1.jar ，org.aspectj.weaver-1.6.8.jar



Spring声明性事务

- StaffUser事务XML方案项目案例-配置XML
  - 步骤一：

```
<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-4.3.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-4.3.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">
```



Spring声明性事务

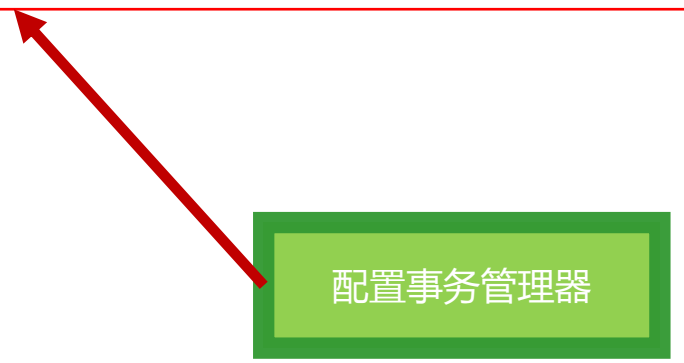
- 步骤二：


```
<bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName"
        value="com.mysql.cj.jdbc.Driver" />
    <property name="url"
        value="jdbc:mysql://localhost:3306/staff?useSSL=false&amp;serverTimezone=UTC" />
    <property name="username" value="root" />
    <property name="password" value="123456" />
</bean>
```

Spring声明性事务

- 步骤三：

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



Spring声明性事务

- 步骤四：

```
<aop:config>
    <aop:pointcut id="serviceOperation"
                  expression="execution(* com.icss.biz.*.*(..)))" />
    <aop:advisor advice-ref="txAdvice" pointcut-ref="serviceOperation" />
</aop:config>

<tx:advice id="txAdvice" transaction-manager="txManager">
    <tx:attributes>
        <tx:method name="delete*"  rollback-for="Throwable" />
        <tx:method name="update*"  rollback-for="Throwable" />
        <tx:method name="add*"  rollback-for="Throwable" />
        <tx:method name="*"  read-only="true" />
    </tx:attributes>
</tx:advice>
```

注意：Spring Framework的事务，默认只有runtime和unchecked 异常才会事务回滚。如果从事务方法中抛出Checked 异常，将不会进行事务回滚。因此，要配置rollback-for=Throwable

read-only为只读事务，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。readonly并不是所有数据库都支持，不同的数据库下会有不同的结果。

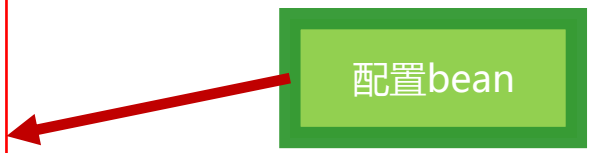
Spring声明性事务

属性	是否需要？	默认值	描述
name	是		与事务属性关联的方法名。通配符（*）可以用来指定一批关联到相同的事务属性的方法。 如：'get*'、'handle*'、'on*Event'等等。
propagation	不	REQUIRED	事务传播行为
isolation	不	DEFAULT	事务隔离级别
timeout	不	-1	事务超时的时间（以秒为单位）
read-only	不	false	事务是否只读？
rollback-for	不		将被触发进行回滚的 Exception(s)；以逗号分开。 如：'com.foo.MyBusinessException,ServletException'
no-rollback-for	不		不被触发进行回滚的 Exception(s)；以逗号分开。 如：'com.foo.MyBusinessException,ServletException'

Spring声明性事务

- 步骤五：

```
<bean id="userDao" class="com.icss.dao.UserDao">
   <property name="dataSource" ref="dataSource"></property>
</bean>
<bean id="staffDao" class="com.icss.dao.StaffDao">
    <property name="dataSource" ref="dataSource"></property>
</bean>
<bean id="staffBiz" class="com.icss.biz.StaffBiz">
   <property name="userDao" ref="userDao"></property>
   <property name="staffDao" ref="staffDao"></property>
</bean>
```



Spring声明性事务

- StaffUser事务XML方案项目案例-事务控制业务代码

```
public class StaffBiz implements IStaff{
    private StaffDao staffDao;
    private UserDao userDao;

    public void setStaffDao(StaffDao staffDao) {
        this.staffDao = staffDao;
    }

    public void setUserDao(UserDao userDao) {
        this.userDao = userDao;
    }

    /**
     * 声明性事务，保证员工和用户信息同时写入成功
     */

    public void addStaffUser(TStaff staff, TUser user) throws Exception {
        Log.logger.info("StaffBiz-->>addStaffUser()");

        staffDao.addStaff(staff);
        userDao.addUser(user);
    }
```

Spring声明性事务

- StaffUser事务XML方案项目案例-事务测试
  - 步骤一：事务提交测试

```
TStaff staff = new TStaff();
staff.setSno("121000165");
staff.setName("赵65");
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
try {
    staff.setBirthday(sdf.parse("1995-10-1"));
} catch (Exception e) {

}
staff.setAddress("北京朝阳区建国门");
staff.setTel("1352245466221");
TUser user = new TUser();
user.setSno("121000165");
user.setUname("zhao65");
user.setRole(2);
user.setPwd("1234");

IStaff staffProxy = (IStaff)BeanFactory.getBean("staffBiz");
try {
    staffProxy.addStaffUser(staff, user);
    System.out.println(staff.getSno() + "创建成功....");
} catch (Exception e) {
    e.printStackTrace();
}
```

```
INFO - Loading XML bean definitions from class path resource [beans.xml]
INFO - Loaded JDBC driver: com.mysql.cj.jdbc.Driver
INFO - StaffBiz-->>addStaffUser()
INFO - StaffDao-->>addStaff()
INFO - UserDao-->>addUser()
121000163创建成功....
```

测试结果

Spring声明性事务

- 步骤二：事务回滚测试

```java
public class UserDao extends BaseDao{

    /**
     * 添加用户
     * @param user
     * @throws Exception
     */
    public void addUser(TUser user) throws Exception {
        Log.logger.info("UserDao-->>addUser()");
        String sql = "insert into tuser values(?,?,?,?)";
        Connection conn = this.openConnection();
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, user.getUname());
        ps.setString(2, user.getSno());
        ps.setString(3, user.getPwd());
        ps.setInt(4,user.getRole());
        ps.executeUpdate();
        ps.close();
        throw new RuntimeException("异常测试....");
    }
```

```
INFO - Loading XML bean definitions from class path resource [beans.xml]
INFO - Loaded JDBC driver: com.mysql.cj.jdbc.Driver
INFO - StaffBiz-->>addStaffUser()
INFO - StaffDao-->>addStaff()
INFO - UserDao-->>addUser()
java.lang.RuntimeException: 异常测试....
        at com.icss.dao.UserDao.addUser(UserDao.java:30)
        at com.icss.biz.StaffBiz.addStaffUser(StaffBiz.java:29)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
```

测试结果需要在mysql数据库中检查，TStaff数据是否已回滚

Spring声明性事务

- 注解管理声明性事务
  - @Service
  - @Repository
  - @Transactional

Spring声明性事务

- StaffUser事务注解方案项目案例-配置XML
  - 步骤一：

```
<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"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-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/context
        http://www.springframework.org/schema/context/spring-context.xsd">
```

Spring声明性事务

- 步骤二：

```
<bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName"
        value="com.mysql.cj.jdbc.Driver" />
    <property name="url"
        value="jdbc:mysql://localhost:3306/staff?useSSL=false&amp;serverTimezone=UTC" />
    <property name="username" value="root" />
    <property name="password" value="123456" />
</bean>

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

数据源与事务管理器配置不变

Spring声明性事务

- 步骤三：

```
<tx:annotation-driven transaction-manager="txManager" />
```

配置事务的注解驱动
表示事务管理，使用@Transactional注解方式

```
<context:component-scan base-package="com.icss.biz"/>
<context:component-scan base-package="com.icss.dao"/>
```

配置组件扫描

Spring声明性事务

- StaffUser事务注解方案项目案例-组件注入

```
public abstract class BaseDao extends JdbcDaoSupport {

    //注入dataSource给JdbcDaoSupport赋值
    @Autowired
    public void setDataSource(org.springframework.jdbc.datasource.DriverManagerDataSource dataSource) {
        super.setDataSource(dataSource);
    }
```

```
@Service("staffBiz")
public class StaffBiz implements IStaff{

    @Autowired
    private StaffDao staffDao;
    @Autowired
    private UserDao userDao;
```

```
@Repository("staffDao")
public class StaffDao extends BaseDao{
```

```
@Repository("userDao")
public class UserDao extends BaseDao{
```

事务分类

- StaffUser事务注解方案项目案例-切面与事务策略
  - @Transactional 注解应该只被应用到 *public* 方法上。 如果你在 protected、private 或者 package-visible 的方法上使用 @Transactional 注解，系统也不会报错， 但是这个被注解的方法将不会执行已配置的事务设置。

```
/**
 * 声明性事务，保证员工和用户信息同时写入成功
 */
@Transactional(rollbackFor=Throwable.class)
public void addStaffUser(TStaff staff, TUser user) throws Exception {
    Log.logger.info("StaffBiz-->>addStaffUser()");

    staffDao.addStaff(staff);
    userDao.addUser(user);
}
```

事务分类

属性	类型	描述
Propagation	枚举型：Propagation	可选的传播性设置
Isolation	枚举型：Isolation	可选的隔离性级别（默认值：ISOLATION_DEFAULT）
readOnly	布尔型	读写型事务 vs. 只读型事务
timeout	int型（以秒为单位）	事务超时
rollbackFor	一组 Class 类的实例，必须是Throwable 的子类	一组异常类，遇到时 必须 进行回滚。默认情况下checked exceptions不进行回滚，仅unchecked exceptions（即RuntimeException的子类）才进行事务回滚。
rollbackForClassname	一组 Class 类的名字，必须是Throwable的子类	一组异常类名，遇到时 必须 进行回滚
noRollbackFor	一组 Class 类的实例，必须是Throwable 的子类	一组异常类，遇到时 必须不 回滚。
noRollbackForClassname	一组 Class 类的名字，必须是Throwable 的子类	一组异常类，遇到时 必须不 回滚

事务分类

- StaffUser事务注解方案项目案例-事务测试
  - 事务测试

```
TStaff staff = new TStaff();
staff.setSno("121000165");
staff.setName("赵65");
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
try {
    staff.setBirthday(sdf.parse("1995-10-1"));
} catch (Exception e) {

}
staff.setAddress("北京朝阳区建国门");
staff.setTel("1352245466221");
TUser user = new TUser();
user.setSno("121000165");
user.setUname("zhao65");
user.setRole(2);
user.setPwd("1234");

IStaff staffProxy = (IStaff)BeanFactory.getBean("staffBiz");
try {
    staffProxy.addStaffUser(staff, user);
    System.out.println(staff.getSno() + "创建成功....");
} catch (Exception e) {
    e.printStackTrace();
}
```

```
INFO - Loading XML bean definitions from class path resource [beans.xml]
INFO - Loaded JDBC driver: com.mysql.cj.jdbc.Driver
INFO - StaffBiz-->>addStaffUser()
INFO - StaffDao-->>addStaff()
INFO - UserDao-->>addUser()
121000165创建成功....
```

事务分类

- 事务回滚测试

```java
@Repository("userDao")
public class UserDao extends BaseDao{

    /**
     * 添加用户
     * @param user
     * @throws Exception
     */
    public void addUser(TUser user) throws Exception {
        Log.logger.info("UserDao-->>addUser()");
        String sql = "insert into tuser values(?,?,?,?)";
        Connection conn = this.openConnection();
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, user.getUname());
        ps.setString(2, user.getSno());
        ps.setString(3, user.getPwd());
        ps.setInt(4,user.getRole());
        ps.executeUpdate();
        ps.close();
        throw new RuntimeException("异常测试....");
    }
```

```
INFO - Loading XML bean definitions from class path resource [beans.xml]
INFO - Loaded JDBC driver: com.mysql.cj.jdbc.Driver
INFO - StaffBiz-->>addStaffUser()
INFO - StaffDao-->>addStaff()
INFO - UserDao-->>addUser()
java.lang.RuntimeException: 异常测试....
        at com.icss.dao.UserDao.addUser(UserDao.java:32)
        at com.icss.biz.StaffBiz.addStaffUser(StaffBiz.java:30)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
```

在mysql数据库中检查，TStaff数据是否已回滚。

Spring编程式事务

- 编程式事务介绍
  - Spring Framework提供了两种编程式事务管理方法：
    - 使用TransactionTemplate
    - 直接使用PlatformTransactionManager
  - Spring的编程式事务应用并不广泛，同我们前面示例中的编程事务对比，优势仅仅是使用了一致的事务管理抽象。

Spring编程式事务

- Spring编程式事务新增StaffUser员工案例
  - 步骤一：编写配置文件

```
<context:component-scan
    base-package="com.icss.biz" />

<context:component-scan
    base-package="com.icss.dao" />

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="com.mysql.cj.jdbc.Driver" />
    <property name="url"
      value="jdbc:mysql://localhost:3306/staff?useSSL=false&amp;serverTimezone=UTC&amp;allowPublicKeyRetrieval=true" />
    <property name="username" value="root" />
    <property name="password" value="123456" />
</bean>

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

Spring编程式事务

- 步骤二：注入数据源

```
public abstract class BaseDao extends JdbcDaoSupport {

    public Connection openConnection() throws Exception {
        //从spring的context中获取connection
        return this.getConnection();
    }

    @Autowired
    public void setMyDataSource(org.springframework.jdbc.datasource.DriverManagerDataSource dataSource){
        super.setDataSource(dataSource);
    }
```

Spring编程式事务

- 步骤三：注入PlatformTransactionManager

```
@Service("staffBiz")
public class StaffBiz implements IStaff{

    @Autowired
    private PlatformTransactionManager txManager;

    @Autowired
    StaffDao staffDao;

    @Autowired
    UserDao userDao;
```

Spring编程式事务

- 步骤四：业务逻辑控制

```
/**
 * 控制本地事务，保证员工和用户信息同时写入成功
 */
public void addStaffUser(Staff staff, User user) throws Exception {
    Log.logger.info(Thread.currentThread().getId() + ":addStaffUser()");

    DefaultTransactionDefinition def = new DefaultTransactionDefinition();
    def.setName("SomeTxName");
    def.setPropagationBehavior(TransactionDefinition.PROPAGATION_REQUIRED);
    TransactionStatus status = txManager.getTransaction(def);
    try {
        staffDao.addStaff(staff);
        userDao.addUser(user);
        txManager.commit(status);
    }
    catch (Exception ex) {
        txManager.rollback(status);
        throw ex;
    }
}
```

Spring编程式事务

- 步骤五：测试

```
public static void main(String[] args) {
    Staff staff = new Staff();
    staff.setSno("121000158");
    staff.setName("马58");
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
    try {
        staff.setBirthday(sdf.parse("1995-10-1"));
    } catch (Exception e) {
    }
    staff.setAddress("北京朝阳区建国门");
    staff.setTel("13522454666");
    User user = new User();
    user.setSno("121000158");
    user.setUname("ma58");
    user.setRole(2);
    user.setPwd("1234");
    //获得Spring的bean对象
    IStaff staffProxy = (IStaff)BeanFactory.getBean(IStaff.class);
    try {
        staffProxy.addStaffUser(staff, user);
        System.out.println(staff.getSno() + "创建成功....");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
INFO - Loading XML bean definitions from class path resource [beans.xml]
INFO - Loaded JDBC driver: com.mysql.cj.jdbc.Driver
INFO - 1:addStaffUser()
INFO - 1:addStaff()
INFO - 1:addUser()
121000158创建成功....
```

Spring编程式事务

- 步骤六：异常测试

```
@Repository("userDao")
public class UserDao extends BaseDao{

    /**
     * 添加用户
     * @param user
     * @throws Exception
     */
    public void addUser(User user) throws Exception {
        Log.logger.info(Thread.currentThread().getId() + ":addUser()");
        String sql = "insert into tuser values(?,?,?,?)";
        Connection conn = this.openConnection();
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, user.getUname());
        ps.setString(2, user.getSno());
        ps.setString(3, user.getPwd());
        ps.setInt(4,user.getRole());
        ps.executeUpdate();
        ps.close();
        throw new RuntimeException("事务回滚测试...");
    }
```

```
INFO - Loading XML bean definitions from class path resource [beans.xml]
INFO - Loaded JDBC driver: com.mysql.cj.jdbc.Driver
INFO - 1:addStaffUser()
INFO - 1:addStaff()
INFO - 1:addUser()
java.lang.RuntimeException: 事务回滚测试...
        at com.icss.dao.UserDao.addUser(UserDao.java:30)
        at com.icss.biz.impl.StaffBiz.addStaffUser(StaffBiz.java:41)
        at com.icss.ui.TestAddStaffUser.main(TestAddStaffUser.java:30)
```

声明性事务与编程式事务选择

- 当你只有很少的事务操作时，编程式事务管理通常比较合适。例如，如果你有一个Web应用，其中只有特定的更新操作有事务要求， 你可能不愿使用Spring或其他技术设置事务代理。这种情况下，使用 TransactionTemplate 可能是个好办法。只有编程式事务管理才能显式的设置事务名称。
- 反之，如果你的应用中存在大量事务操作，那么声明式事务管理通常是值得的。它将事务管理与业务逻辑分离， 使用Spring，而不是EJB 的CMT，在配置上的成本大大地降低了。
- *注意：绝大多数开发者的业务场景，事务操作的情况并不多，而他们普遍使用Spring声明性事务，这其实是对性能的一种浪费，尤其是高并发环境要小心。EJB3.0后，其CMT配置非常简单，在企业级应用中仍然是非常好的选择！*

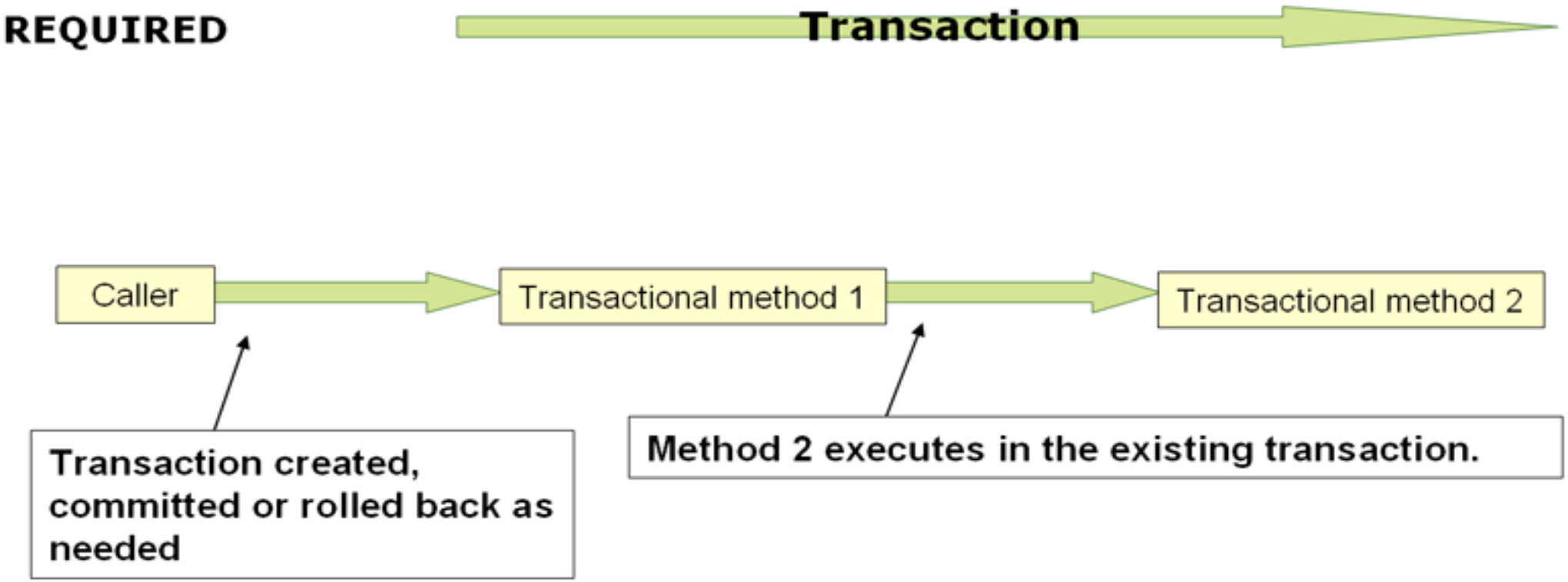
Spring事务传播性

- 说明：
  - Spring事务传播性模拟JTA事务传播性设置。
  - JTA事务传播性有7种，Spring主要支持其中三种：

PROPAGATION_REQUIRED	1. 若存在外部事务，则内部事务使用外部事务的物理环境 2. 若不存在外部事务，不会创建新的事务环境??? 3. 内部事务和外部事务，两个逻辑事务环境可以独立的设置回滚状态。物理事务是否回滚，由外部方法决定。
PAOPAGATION_REQUIRE_NEW	不论是否存在事务环境，都新建一个事务，新老事务相互独立。内部事务抛出异常回滚，不会影响外部事务的正常提交
PROPAGATION_NESTED	如果当前存在事务，则嵌套在当前事务中执行。如果当前没有事务，则新建一个事务
PROPAGATION_SUPPORTS	支持当前事务，若当前不存在事务，以非事务的方式执行
PROPAGATION_NOT_SUPPORTED	以非事务的方式执行，若当前存在事务，则把当前事务挂起
PROPAGATION_MANDATORY	强制事务执行，若当前不存在事务，则抛出异常
PROPAGATION_NEVER	以非事务的方式执行，如果当前存在事务，则抛出异常

Spring事务传播性

- Propagation.REQUIRED设置



Spring事务传播性

- 步骤一：业务方法事务设置

```
@Service("staffBiz")
public class StaffBiz implements IStaff{

    @Transactional(rollbackFor=Throwable.class)
    public void operataionA() {
        Log.logger.info("operataionA...");
        operataionB();

    }

    public void operataionA2() {
        Log.logger.info("operataionA2...");
        operataionB();
    }


    @Transactional(propagation=Propagation.REQUIRED,rollbackFor=Throwable.class)
    public void operataionB() {
        Log.logger.info("operataionB...");
    }
```

Spring事务传播性

- 步骤二：测试（外部方法与内部方法，都有事务设置）

```
public static void main(String[] args) {
    IStaff staffProxy = (IStaff)BeanFactory.getBean("staffBiz");
    staffProxy.operataionA();
    System.out.println("测试结束...");
}
```

如上所示，operataionA与operataionB都有事务设置，operataionB使用了operataionA的事务环境

```
DEBUG - Creating new transaction with name [com.icss.biz.StatfBiz.operataionA]: PROPAGATION_REQUIRED,ISOLATION_DEFAULT,-j
DEBUG - Creating new JDBC DriverManager Connection to [jdbc:mysql://localhost:3306/staff?useSSL=false&serverTimezone=UTC]
DEBUG - Acquired Connection [com.mysql.cj.jdbc.ConnectionImpl@15ef5] for JDBC transaction
DEBUG - Switching JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@15ef5] to manual commit
DEBUG - Bound value [org.springframework.jdbc.datasource.ConnectionHolder@11a5887] for key [org.springframework.jdbc.data
DEBUG - Initializing transaction synchronization
DEBUG - Getting transaction for [com.icss.biz.StaffBiz.operataionA]
INFO - operataionA...
INFO - operataionB...
DEBUG - Completing transaction for [com.icss.biz.StaffBiz.operataionA]
DEBUG - Triggering beforeCommit synchronization
DEBUG - Triggering beforeCompletion synchronization
DEBUG - Initiating transaction commit
DEBUG - Committing JDBC transaction on Connection [com.mysql.cj.jdbc.ConnectionImpl@15ef5]
DEBUG - Triggering afterCommit synchronization
DEBUG - Clearing transaction synchronization
DEBUG - Triggering afterCompletion synchronization
DEBUG - Removed value [org.springframework.jdbc.datasource.ConnectionHolder@11a5887] for key [org.springframework.jdbc.da
DEBUG - Releasing JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@15ef5] after transaction
DEBUG - Returning JDBC Connection to DataSource
测试结束...
```

Spring事务传播性

- 步骤三：测试（外部方法无事务设置）

```
public static void main(String[] args) {
    IStaff staffProxy = (IStaff)BeanFactory.getBean("staffBiz");
    staffProxy.operataionA2();
    System.out.println("测试结束...");
}
```

如上所示，operataionA2无事务设置，调用operataionB。结果显示，虽然operataionB设置了事务需求，但是并没有生成事务环境

```
DEBUG - Returning cached instance of singleton bean 'staffBiz'
INFO - operataionA2...
INFO - operataionB...
测试结束...
```

```
@Transactional(propagation=Propagation.REQUIRED,rollbackFor=Throwable.class)
public void operataionB() {
```

Spring事务传播性

- 步骤四：异常测试

```java
@Service("staffBiz")
public class StaffBiz implements IStaff{

    @Transactional(rollbackFor=Throwable.class)
    public void operataionA() {
        Log.logger.info("operataionA...");
        try {
            operataionB();
        } catch (Exception e) {
            Log.logger.error(e.getMessage());
        }
    }

    @Transactional(propagation=Propagation.REQUIRED,rollbackFor=Throwable.class)
    public void operataionB() {
        Log.logger.info("operataionB...");
        throw new RuntimeException("operataionB抛出异常...");
    }
```

```
DEBUG - Creating new transaction with name [com.icss.biz.StaffBiz.operataionA]: PROPAGATION_REQUIRED,ISOLATION_
DEBUG - Creating new JDBC DriverManager Connection to [jdbc:mysql://localhost:3306/staff?useSSL=false&serverTim
DEBUG - Acquired Connection [com.mysql.cj.jdbc.ConnectionImpl@15ef5] for JDBC transaction
DEBUG - Switching JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@15ef5] to manual commit
DEBUG - Bound value [org.springframework.jdbc.datasource.ConnectionHolder@11a5887] for key [org.springframework
DEBUG - Initializing transaction synchronization
DEBUG - Getting transaction for [com.icss.biz.StaffBiz.operataionA]
INFO - operataionA...
INFO - operataionB...
ERROR - operataionB抛出异常...
DEBUG - Completing transaction for [com.icss.biz.StaffBiz.operataionA]
DEBUG - Triggering beforeCommit synchronization
DEBUG - Triggering beforeCompletion synchronization
DEBUG - Initiating transaction commit
DEBUG - Committing JDBC transaction on Connection [com.mysql.cj.jdbc.ConnectionImpl@15ef5]
DEBUG - Triggering afterCommit synchronization
DEBUG - Clearing transaction synchronization
DEBUG - Triggering afterCompletion synchronization
DEBUG - Removed value [org.springframework.jdbc.datasource.ConnectionHolder@11a5887] for key [org.springframewo
DEBUG - Releasing JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@15ef5] after transaction
DEBUG - Returning JDBC Connection to DataSource
测试结束...
```

operataionB中抛出异常，在operataionA中处理后，不影响operataionA的事务提交。

Spring事务传播性

- 步骤五：异常测试2

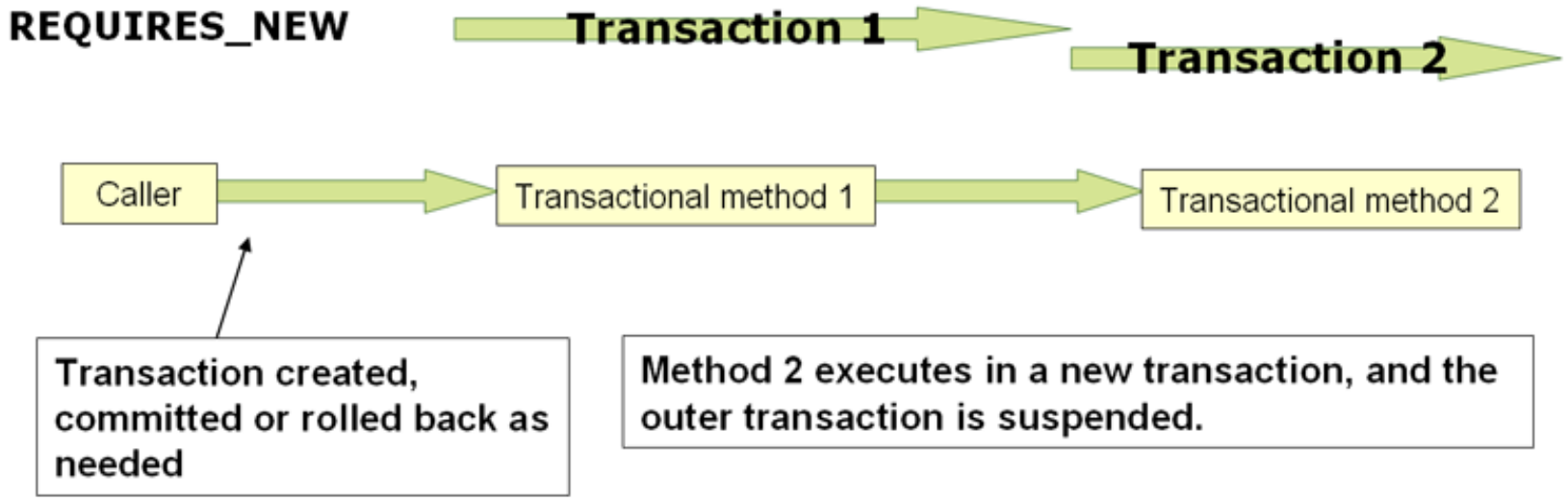
```java
@Service("staffBiz")
public class StaffBiz implements IStaff{

    @Transactional(rollbackFor=Throwable.class)
    public void operataionA() {
        Log.logger.info("operataionA...");
        operataionB();
    }

    @Transactional(propagation=Propagation.REQUIRED,rollbackFor=Throwable.class)
    public void operataionB() {
        Log.logger.info("operataionB...");
        throw new RuntimeException("operataionB抛出异常...");
    }
```

```
DEBUG - Creating new transaction with name [com.icss.biz.StaffBiz.operataionA]: PROPAGATION_REQUIRED,ISOLATION_DEFAULT
DEBUG - Creating new JDBC DriverManager Connection to [jdbc:mysql://localhost:3306/staff?useSSL=false&serverTimezone=U
DEBUG - Acquired Connection [com.mysql.cj.jdbc.ConnectionImpl@15ef5] for JDBC transaction
DEBUG - Switching JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@15ef5] to manual commit
DEBUG - Bound value [org.springframework.jdbc.datasource.ConnectionHolder@11a5887] for key [org.springframework.jdbc.d
DEBUG - Initializing transaction synchronization
DEBUG - Getting transaction for [com.icss.biz.StaffBiz.operataionA]
INFO - operataionA...
INFO - operataionB...
DEBUG - Completing transaction for [com.icss.biz.StaffBiz.operataionA] after exception: java.lang.RuntimeException: op
DEBUG - Applying rules to determine whether transaction should rollback on java.lang.RuntimeException: operataionB抛出异
DEBUG - Winning rollback rule is: RollbackRuleAttribute with pattern [java.lang.Throwable]
DEBUG - Triggering beforeCompletion synchronization
DEBUG - Initiating transaction rollback
DEBUG - Rolling back JDBC transaction on Connection [com.mysql.cj.jdbc.ConnectionImpl@15ef5]
DEBUG - Clearing transaction synchronization
DEBUG - Triggering afterCompletion synchronization
DEBUG - Removed value [org.springframework.jdbc.datasource.ConnectionHolder@11a5887] for key [org.springframework.jdbc
DEBUG - Releasing JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@15ef5] after transaction
DEBUG - Returning JDBC Connection to DataSource
Exception in thread "main" java.lang.RuntimeException: operataionB抛出异常...
        at com.icss.biz.StaffBiz.operataionB(StaffBiz.java:20)
        at com.icss.biz.StaffBiz.operataionA(StaffBiz.java:14)
```

operataionB中抛出异常，在operataionA中未处理，operataionA的事务回滚

Spring事务传播性

- Propagation.REQUIRES_NEW设置
  - PROPAGATION_REQUIRES_NEW与 PROPAGATION_REQUIRED相反，在其各自事务范围内，永远使用各自独立的物理事务环境。内部事务永远不会参与外部已存在事务环境。



Spring事务传播性

- 步骤一：业务方法事务设置

```
@Transactional(rollbackFor=Throwable.class)
public void operataionA() {
    Log.logger.info("operataionA...");
    operataionC();
}

public void operataionA2() {
    Log.logger.info("operataionA2...");
    operataionC();
}

@Transactional(propagation=Propagation.REQUIRES_NEW,rollbackFor=Throwable.class)
public void operataionC() {
    Log.logger.info("operataionC...");
}
```

Spring事务传播性

- 步骤二：测试（外部方法与内部方法，都有事务设置）

```
public static void main(String[] args) {
    IStaff staffProxy = (IStaff)BeanFactory.getBean("staffBiz");
    staffProxy.operataionA();
    System.out.println("测试结束...");
}
```

测试结果很意外，operataionA和operataionC都有事务设置，operataionC使用了operataionA的物理环境？why

```
DEBUG - Creating new transaction with name [com.icss.biz.StaffBiz.operataionA]: PROPAGATION_REQUIRED,ISOLATION_DEFAULT
DEBUG - Creating new JDBC DriverManager Connection to [jdbc:mysql://localhost:3306/staff?useSSL=false&serverTimezone=U
DEBUG - Acquired Connection [com.mysql.cj.jdbc.ConnectionImpl@15ef5] for JDBC transaction
DEBUG - Switching JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@15ef5] to manual commit
DEBUG - Bound value [org.springframework.jdbc.datasource.ConnectionHolder@11a5887] for key [org.springframework.jdbc.da
DEBUG - Initializing transaction synchronization
DEBUG - Getting transaction for [com.icss.biz.StaffBiz.operataionA]
INFO - operataionA...
INFO - operataionC...
DEBUG - Completing transaction for [com.icss.biz.StaffBiz.operataionA]
DEBUG - Triggering beforeCommit synchronization
DEBUG - Triggering beforeCompletion synchronization
DEBUG - Initiating transaction commit
DEBUG - Committing JDBC transaction on Connection [com.mysql.cj.jdbc.ConnectionImpl@15ef5]
DEBUG - Triggering afterCommit synchronization
DEBUG - Clearing transaction synchronization
DEBUG - Triggering afterCompletion synchronization
DEBUG - Removed value [org.springframework.jdbc.datasource.ConnectionHolder@11a5887] for key [org.springframework.jdbc
DEBUG - Releasing JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@15ef5] after transaction
DEBUG - Returning JDBC Connection to DataSource
测试结束...
```

Spring事务传播性

- 步骤三：外部方法无事务设置

```
public static void main(String[] args) {
    IStaff staffProxy = (IStaff)BeanFactory.getBean("staffBiz");
    staffProxy.operataionA2();
    System.out.println("测试结束...");
}
```

```
DEBUG - Returning cached instance of singleton bean 'staffBiz'
INFO - operataionA2...
INFO - operataionC...
测试结束...
```

测试结果显示，Propagation.REQUIRES_NEW设置无效

Spring事务传播性

- 步骤四：修改业务逻辑代码，注入自身

```
@Service("staffBiz")
public class StaffBiz implements IStaff{

    @Autowired
    private IStaff staffBiz;

    @Transactional(rollbackFor=Throwable.class)
    public void operataionA() {
        Log.logger.info("operataionA...");
        staffBiz.operataionC();
    }

    public void operataionA2() {
        Log.logger.info("operataionA2...");
        staffBiz.operataionC();
    }

    @Transactional(propagation=Propagation.REQUIRES_NEW,rollbackFor=Throwable.class)
    public void operataionC() {
        Log.logger.info("operataionC...");
        //throw new RuntimeException("operataionC抛出异常...");
    }
```

Spring事务传播性

- 步骤五：测试（外部方法与内部方法，都有事务设置）

```
public static void main(String[] args) {
    IStaff staffProxy = (IStaff)BeanFactory.getBean("staffBiz");
    staffProxy.operataionA();
    System.out.println("测试结束...");
}
```

```
DEBUG - Creating new transaction with name [com.icss.biz.StaffBiz.operataionA]: PROPAGATION_REQUIRED,ISOLATION_DEFAULT,
DEBUG - Creating new JDBC DriverManager Connection to [jdbc:mysql://localhost:3306/staff?useSSL=false&serverTimezone=UT
DEBUG - Acquired Connection [com.mysql.cj.jdbc.ConnectionImpl@116132a] for JDBC transaction
DEBUG - Switching JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@116132a] to manual commit
DEBUG - Bound value [org.springframework.jdbc.datasource.ConnectionHolder@1d6015a] for key [org.springframework.jdbc.da
DEBUG - Initializing transaction synchronization
DEBUG - Getting transaction for [com.icss.biz.StaffBiz.operataionA]
INFO - operataionA...
DEBUG - Adding transactional method 'com.icss.biz.StaffBiz.operataionC' with attribute: PROPAGATION_REQUIRES_NEW,ISOLAT
DEBUG - Retrieved value [org.springframework.jdbc.datasource.ConnectionHolder@1d6015a] for key [org.springframework.jdb
DEBUG - Suspending current transaction, creating new transaction with name [com.icss.biz.StaffBiz.operataionC]
DEBUG - Clearing transaction synchronization
DEBUG - Removed value [org.springframework.jdbc.datasource.ConnectionHolder@1d6015a] for key [org.springframework.jdbc.
DEBUG - Creating new JDBC DriverManager Connection to [jdbc:mysql://localhost:3306/staff?useSSL=false&serverTimezone=UT
DEBUG - Acquired Connection [com.mysql.cj.jdbc.ConnectionImpl@136d012] for JDBC transaction
DEBUG - Switching JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@136d012] to manual commit
DEBUG - Bound value [org.springframework.jdbc.datasource.ConnectionHolder@102734] for key [org.springframework.jdbc.dat
DEBUG - Initializing transaction synchronization
DEBUG - Getting transaction for [com.icss.biz.StaffBiz.operataionC]
INFO - operataionC...
DEBUG - Completing transaction for [com.icss.biz.StaffBiz.operataionC]
DEBUG - Triggering beforeCommit synchronization
DEBUG - Triggering beforeCompletion synchronization
DEBUG - Initiating transaction commit
DEBUG - Committing JDBC transaction on Connection [com.mysql.cj.jdbc.ConnectionImpl@136d012]
DEBUG - Triggering afterCommit synchronization
DEBUG - Clearing transaction synchronization
DEBUG - Triggering afterCompletion synchronization
DEBUG - Removed value [org.springframework.jdbc.datasource.ConnectionHolder@102734] for key [org.springframework.jdbc.datas
DEBUG - Releasing JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@136d012] after transaction
DEBUG - Returning JDBC Connection to DataSource
DEBUG - Resuming suspended transaction after completion of inner transaction
DEBUG - Bound value [org.springframework.jdbc.datasource.ConnectionHolder@1d6015a] for key [org.springframework.jdbc.datasc
DEBUG - Initializing transaction synchronization
DEBUG - Completing transaction for [com.icss.biz.StaffBiz.operataionA]
DEBUG - Triggering beforeCommit synchronization
DEBUG - Triggering beforeCompletion synchronization
DEBUG - Initiating transaction commit
DEBUG - Committing JDBC transaction on Connection [com.mysql.cj.jdbc.ConnectionImpl@116132a]
DEBUG - Triggering afterCommit synchronization
DEBUG - Clearing transaction synchronization
DEBUG - Triggering afterCompletion synchronization
DEBUG - Removed value [org.springframework.jdbc.datasource.ConnectionHolder@1d6015a] for key [org.springframework.jdbc.data
DEBUG - Releasing JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@116132a] after transaction
DEBUG - Returning JDBC Connection to DataSource
测试结束...
```

Spring事务传播性

- 步骤六：测试（外部方法无事务设置）

```
public static void main(String[] args) {
    IStaff staffProxy = (IStaff)BeanFactory.getBean("staffBiz");
    staffProxy.operataionA2();
    System.out.println("测试结束...");
}
```

```
DEBUG - Returning cached instance of singleton bean 'staffBiz'
INFO - operataionA2...
DEBUG - Adding transactional method 'com.icss.biz.StaffBiz.operataionC' with attribute: PROPAGATION_REQUIRES_NEW,ISOLATION_
DEBUG - Returning cached instance of singleton bean 'org.springframework.transaction.interceptor.TransactionInterceptor#0'
DEBUG - Returning cached instance of singleton bean 'txManager'
DEBUG - Creating new transaction with name [com.icss.biz.StaffBiz.operataionC]: PROPAGATION_REQUIRES_NEW,ISOLATION_DEFAULT
DEBUG - Creating new JDBC DriverManager Connection to [jdbc:mysql://localhost:3306/staff?useSSL=false&serverTimezone=UTC]
DEBUG - Acquired Connection [com.mysql.cj.jdbc.ConnectionImpl@11a5887] for JDBC transaction
DEBUG - Switching JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@11a5887] to manual commit
DEBUG - Bound value [org.springframework.jdbc.datasource.ConnectionHolder@a7bfbc] for key [org.springframework.jdbc.dataso
DEBUG - Initializing transaction synchronization
DEBUG - Getting transaction for [com.icss.biz.StaffBiz.operataionC]
INFO - operataionC...
DEBUG - Completing transaction for [com.icss.biz.StaffBiz.operataionC]
DEBUG - Triggering beforeCommit synchronization
DEBUG - Triggering beforeCompletion synchronization
DEBUG - Initiating transaction commit
DEBUG - Committing JDBC transaction on Connection [com.mysql.cj.jdbc.ConnectionImpl@11a5887]
DEBUG - Triggering afterCommit synchronization
DEBUG - Clearing transaction synchronization
DEBUG - Triggering afterCompletion synchronization
DEBUG - Removed value [org.springframework.jdbc.datasource.ConnectionHolder@a7bfbc] for key [org.springframework.jdbc.data
DEBUG - Releasing JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@11a5887] after transaction
DEBUG - Returning JDBC Connection to DataSource
测试结束...
```

Spring事务传播性

- 步骤七：异常测试

```
@Service("staffBiz")
public class StaffBiz implements IStaff{

    @Autowired
    private IStaff staffBiz;

    @Transactional(rollbackFor=Throwable.class)
    public void operataionA() {
        Log.logger.info("operataionA...");
        staffBiz.operataionC();
    }

    @Transactional(propagation=Propagation.REQUIRES_NEW,rollbackFor=Throwable.class)
    public void operataionC() {
        Log.logger.info("operataionC...");
        throw new RuntimeException("operataionC抛出异常...");
    }
```

```
public static void main(String[] args) {
    IStaff staffProxy = (IStaff)BeanFactory.getBean("staffBiz");
    staffProxy.operataionA();
    System.out.println("测试结束...");
}
```

```
DEBUG - Creating new JDBC DriverManager Connection to [jdbc:mysql://localhost:3306/staff?useSSL=false&serverTimezone=UT
DEBUG - Acquired Connection [com.mysql.cj.jdbc.ConnectionImpl@116132a] for JDBC transaction
DEBUG - Switching JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@116132a] to manual commit
DEBUG - Bound value [org.springframework.jdbc.datasource.ConnectionHolder@1d6015a] for key [org.springframework.jdbc.da
DEBUG - Initializing transaction synchronization
DEBUG - Getting transaction for [com.icss.biz.StaffBiz.operataionA]
INFO - operataionA...
DEBUG - Adding transactional method 'com.icss.biz.StaffBiz.operataionC' with attribute: PROPAGATION_REQUIRES_NEW,ISOLAT
DEBUG - Retrieved value [org.springframework.jdbc.datasource.ConnectionHolder@1d6015a] for key [org.springframework.jdb
DEBUG - Suspending current transaction, creating new transaction with name [com.icss.biz.StaffBiz.operataionC]
DEBUG - Clearing transaction synchronization
DEBUG - Removed value [org.springframework.jdbc.datasource.ConnectionHolder@1d6015a] for key [org.springframework.jdbc.
DEBUG - Creating new JDBC DriverManager Connection to [jdbc:mysql://localhost:3306/staff?useSSL=false&serverTimezone=UT
DEBUG - Acquired Connection [com.mysql.cj.jdbc.ConnectionImpl@136d012] for JDBC transaction
DEBUG - Switching JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@136d012] to manual commit
DEBUG - Bound value [org.springframework.jdbc.datasource.ConnectionHolder@102734] for key [org.springframework.jdbc.dat
DEBUG - Initializing transaction synchronization
DEBUG - Getting transaction for [com.icss.biz.StaffBiz.operataionC]
INFO - operataionC...
DEBUG - Completing transaction for [com.icss.biz.StaffBiz.operataionC] after exception: java.lang.RuntimeException: ope
DEBUG - Applying rules to determine whether transaction should rollback on java.lang.RuntimeException: operataionC抛出异常
DEBUG - Winning rollback rule is: RollbackRuleAttribute with pattern [java.lang.Throwable]
DEBUG - Triggering beforeCompletion synchronization
DEBUG - Initiating transaction rollback
DEBUG - Rolling back JDBC transaction on Connection [com.mysql.cj.jdbc.ConnectionImpl@136d012]
DEBUG - Clearing transaction synchronization
DEBUG - Triggering afterCompletion synchronization
DEBUG - Removed value [org.springframework.jdbc.datasource.ConnectionHolder@102734] for key [org.springframework.
DEBUG - Releasing JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@136d012] after transaction
DEBUG - Returning JDBC Connection to DataSource
DEBUG - Resuming suspended transaction after completion of inner transaction
DEBUG - Bound value [org.springframework.jdbc.datasource.ConnectionHolder@1d6015a] for key [org.springframework.j
DEBUG - Initializing transaction synchronization
DEBUG - Completing transaction for [com.icss.biz.StaffBiz.operataionA] after exception: java.lang.RuntimeExceptio
DEBUG - Applying rules to determine whether transaction should rollback on java.lang.RuntimeException: operataion
DEBUG - Winning rollback rule is: RollbackRuleAttribute with pattern [java.lang.Throwable]
DEBUG - Triggering beforeCompletion synchronization
DEBUG - Initiating transaction rollback
DEBUG - Rolling back JDBC transaction on Connection [com.mysql.cj.jdbc.ConnectionImpl@116132a]
DEBUG - Clearing transaction synchronization
DEBUG - Triggering afterCompletion synchronization
DEBUG - Removed value [org.springframework.jdbc.datasource.ConnectionHolder@1d6015a] for key [org.springframework
DEBUG - Releasing JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@116132a] after transaction
DEBUG - Returning JDBC Connection to DataSource
Exception in thread "main" java.lang.RuntimeException: operataionC抛出异常...
        at com.icss.biz.StaffBiz.operataionC(StaffBiz.java:24)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
```

Spring事务传播性

- 结论：两个独立的物理事务环境，异常时分别回滚。

```
INFO - operataionA...
DEBUG - Adding transactional method 'com.icss.biz.StaffBiz.operataionC' with attribute: PROPAGATION_
DEBUG - Retrieved value [org.springframework.jdbc.datasource.ConnectionHolder@1d6015a] for key [org.
DEBUG - Suspending current transaction, creating new transaction with name [com.icss.biz.StaffBiz.op
DEBUG - Clearing transaction synchronization
DEBUG - Removed value [org.springframework.jdbc.datasource.ConnectionHolder@1d6015a] for key [org.sp
DEBUG - Creating new JDBC DriverManager Connection to [jdbc:mysql://localhost:3306/staff?useSSL=fals
DEBUG - Acquired Connection [com.mysql.cj.jdbc.ConnectionImpl@136d012] for JDBC transaction
DEBUG - Switching JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@136d012] to manual commit
DEBUG - Bound value [org.springframework.jdbc.datasource.ConnectionHolder@102734] for key [org.sprir
DEBUG - Initializing transaction synchronization
DEBUG - Getting transaction for [com.icss.biz.StaffBiz.operataionC]
INFO - operataionC...
```

operataionC操作前，先暂停operataionA的事务

```
INFO - operataionC...
DEBUG - Completing transaction for [com.icss.biz.StaffBiz.operataionC] after exception: java.lang.
DEBUG - Applying rules to determine whether transaction should rollback on java.lang.RuntimeExcept
DEBUG - Winning rollback rule is: RollbackRuleAttribute with pattern [java.lang.Throwable]
DEBUG - Triggering beforeCompletion synchronization
DEBUG - Initiating transaction rollback
DEBUG - Rolling back JDBC transaction on Connection [com.mysql.cj.jdbc.ConnectionImpl@136d012]
DEBUG - Clearing transaction synchronization
DEBUG - Triggering afterCompletion synchronization
DEBUG - Removed value [org.springframework.jdbc.datasource.ConnectionHolder@102734] for key [org.s
DEBUG - Releasing JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@136d012] after transaction
DEBUG - Returning JDBC Connection to DataSource
DEBUG - Resuming suspended transaction after completion of inner transaction
```

operataionC操作完成后，恢复operataionA的事务

Spring事务传播性

- Propagation.NESTED设置
  - PROPAGATION_NESTED 使用了一*个单一的物理*事务，这个事务拥有多个可以回滚的保存点。允许内部事务*在它的事务范围内部分回滚*，并且外部事务能够不受影响。
  - 这个设置仅仅在Spring管理JDBC资源时有效，它对应JDBC的savepoint

数据库连接管理

- 说明：
  - Spring如何管理数据库连接的打开、关闭？
  - 数据库连接是业务系统开发中最为宝贵的资源，很容易成为系统的性能瓶颈。
  - 这个问题，如果处理不好，在并发高的环境下，系统很容易崩溃。

数据库连接管理

- JdbcDaoSupport
  - JdbcDaoSupport的核心方法

```
public abstract class JdbcDaoSupport extends DaoSupport {

    private JdbcTemplate jdbcTemplate;


    /**
     * Set the JDBC DataSource to be used by this DAO.
     */
    public final void setDataSource(DataSource dataSource) {
        if (this.jdbcTemplate == null || dataSource != this.jdbcTemplate.getDataSource()) {
            this.jdbcTemplate = createJdbcTemplate(dataSource);
            initTemplateConfig();
        }
    }
```

```
    /**
     * Get a JDBC Connection, either from the current transaction or a new one.
     * @return the JDBC Connection
     * @throws CannotGetJdbcConnectionException if the attempt to get a Connection failed
     * @see org.springframework.jdbc.datasource.DataSourceUtils#getConnection(javax.sql.DataSource)
     */
    protected final Connection getConnection() throws CannotGetJdbcConnectionException {
        return DataSourceUtils.getConnection(getDataSource());
    }
```

```
    /**
     * Close the given JDBC Connection, created via this DAO's DataSource,
     * if it isn't bound to the thread.
     * @param con Connection to close
     * @see org.springframework.jdbc.datasource.DataSourceUtils#releaseConnection
     */
    protected final void releaseConnection(Connection con) {
        DataSourceUtils.releaseConnection(con, getDataSource());
    }
```

数据库连接管理

- 所有的持久层方法都需要继承JdbcDaoSupport

```
public abstract class BaseDao extends JdbcDaoSupport {

    //注入dataSource给JdbcDaoSupport赋值
    @Autowired
    public void setDataSource(org.springframework.jdbc.datasource.DriverManagerDataSource dataSource) {
        super.setDataSource(dataSource);
    }

    /**
     * 从spring环境上下文查找Connection
     * @return
     */
    public Connection openConnection() {
       return this.getConnection();
    }
```

数据库连接管理

- 持久层需要获得Connection，通过原生API进行数据操作

```
public void addStaff(TStaff staff) throws Exception{
    Log.logger.info("StaffDao-->>addStaff()");

    String sql = "insert into tstaff values(?,?,?,?,?)";
    Connection conn = this.openConnection();

 public void addUser(TUser user) throws Exception {
     Log.logger.info("UserDao-->>addUser()");
     String sql = "insert into tuser values(?,?,?,?)";
     Connection conn = this.openConnection();
```

数据库连接管理

- 数据库连接控制-说明
  - 业务层控制数据库连接使用，如何操作，非常关键。
  - 业务方法有些需要事务（如addStaffUser），有些仅仅需要数据库连接（如login），有些不需要数据库连接(如operateNoDb)

数据库连接管理

- 数据库连接控制-业务类使用@Transactional(readOnly=true)

在业务类使用@Transactional(readOnly=true)，是否合适？

```
@Service("staffBiz")
@Transactional(readOnly=true)
public class StaffBiz implements IStaff{

    @Autowired
    private StaffDao staffDao;
    @Autowired
    private UserDao userDao;

    /**
     * 无数据库访问操作
     */
    public void operateNoDb() {
        Log.logger.info("operateOnDb....");
    }
```

代码测试

```
IStaff staffProxy = (IStaff)BeanFactory.getBean("staffBiz");
staffProxy.operateNoDb();
```

```
DEBUG - Creating new transaction with name [com.icss.biz.StaffBiz.operateNoDb]: PROPAGATION_REQUIRED,ISOLATION_DEFAULT,readOnly
DEBUG - Creating new JDBC DriverManager Connection to [jdbc:mysql://localhost:3306/staff?useSSL=false&serverTimezone=UTC]
DEBUG - Acquired Connection [com.mysql.cj.jdbc.ConnectionImpl@1dabb18] for JDBC transaction
DEBUG - Setting JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@1dabb18] read-only
DEBUG - Switching JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@1dabb18] to manual commit
DEBUG - Bound value [org.springframework.jdbc.datasource.ConnectionHolder@7d121c] for key [org.springframework.jdbc.datasource.Dr
DEBUG - Initializing transaction synchronization
DEBUG - Getting transaction for [com.icss.biz.StaffBiz.operateNoDb]
INFO - operateOnDb....
DEBUG - Completing transaction for [com.icss.biz.StaffBiz.operateNoDb]
DEBUG - Triggering beforeCommit synchronization
DEBUG - Triggering beforeCompletion synchronization
DEBUG - Initiating transaction commit
DEBUG - Committing JDBC transaction on Connection [com.mysql.cj.jdbc.ConnectionImpl@1dabb18]
DEBUG - Triggering afterCommit synchronization
DEBUG - Clearing transaction synchronization
DEBUG - Triggering afterCompletion synchronization
DEBUG - Removed value [org.springframework.jdbc.datasource.ConnectionHolder@7d121c] for key [org.springframework.jdbc.datasource.I
DEBUG - Resetting read-only flag of JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@1dabb18]
DEBUG - Releasing JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@1dabb18] after transaction
DEBUG - Returning JDBC Connection to DataSource
```

通过日志跟踪，如果在业务类StaffBiz使用@Transactional(readOnly=true)注解，即使没有数据库操作，也会打开一个数据库连接，这将造成极大的浪费。

数据库连接管理

- 数据库连接控制-无事务方法控制

```
@Service("staffBiz")
public class StaffBiz implements IStaff{

    @Autowired
    private StaffDao staffDao;
    @Autowired
    private UserDao userDao;

    /**
     * 用户登录
     */
    public TUser login(String uname, String pwd) throws Exception {
        if(uname == null || pwd == null || uname.trim().equals("") || pwd.trim().equals("")) {
            throw new Exception("用户名或密码不能为空......");     }

        TUser user = userDao.login(uname, pwd);

        return user;
    }
```

取消业务类StaffBiz的@Transactional(readOnly=true)注解，进行用户登录测试

数据库连接管理

```
public TUser login(String uname, String pwd) throws Exception {
    TUser user = null;

    String sql = "select * from tuser where uname=? and pwd=?";
    Connection conn = this.openConnection();
    Log.logger.info("UserDao-->>login():" + conn.hashCode());
```

在持久层获取数据库连接

数据库连接管理

```
IStaff staffProxy = (IStaff)BeanFactory.getBean("staffBiz");
staffProxy.login("tom", "123");
System.out.println("登录成功....");

DEBUG - Returning cached instance of singleton bean 'staffBiz'
DEBUG - Fetching JDBC Connection from DataSource
DEBUG - Creating new JDBC DriverManager Connection to [jdbc:mysql://localhost:3306/staff?useSSL=false&s
INFO - UserDao-->>login():16240601
用户登录成功....
```

测试

总结：
如上操作，只是获得了一个新的数据库连接，但是没有释放数据库连接。这样的操作非常危险，很容易导致系统崩溃！

数据库连接管理

最合理的注释方法如下！

```
DEBUG - Creating new transaction with name [com.icss.biz.StaffBiz.login]: PROPAGATION_REQUIRED,ISOLATION_DEFAULT,readOnly
DEBUG - Creating new JDBC DriverManager Connection to [jdbc:mysql://localhost:3306/staff?useSSL=false&serverTimezone=UTC]
DEBUG - Acquired Connection [com.mysql.cj.jdbc.ConnectionImpl@1e35cd9] for JDBC transaction
DEBUG - Setting JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@1e35cd9] read-only
DEBUG - Switching JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@1e35cd9] to manual commit
DEBUG - Bound value [org.springframework.jdbc.datasource.ConnectionHolder@9bec2e] for key [org.springframework.jdbc.datasc
DEBUG - Initializing transaction synchronization
DEBUG - Getting transaction for [com.icss.biz.StaffBiz.login]
DEBUG - Retrieved value [org.springframework.jdbc.datasource.ConnectionHolder@9bec2e] for key [org.springframework.jdbc.da
INFO - UserDao-->>login():31677657
DEBUG - Completing transaction for [com.icss.biz.StaffBiz.login]
DEBUG - Triggering beforeCommit synchronization
DEBUG - Triggering beforeCompletion synchronization
DEBUG - Initiating transaction commit
DEBUG - Committing JDBC transaction on Connection [com.mysql.cj.jdbc.ConnectionImpl@1e35cd9]
DEBUG - Triggering afterCommit synchronization
DEBUG - Clearing transaction synchronization
DEBUG - Triggering afterCompletion synchronization
DEBUG - Removed value [org.springframework.jdbc.datasource.ConnectionHolder@9bec2e] for key [org.springframework.jdbc.data
DEBUG - Resetting read-only flag of JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@1e35cd9]
DEBUG - Releasing JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@1e35cd9] after transaction
DEBUG - Returning JDBC Connection to DataSource
登录成功....
```

```java
@Service("staffBiz")
public class StaffBiz implements IStaff{

    @Autowired
    private StaffDao staffDao;
    @Autowired
    private UserDao userDao;

    /**
     * 用户登录
     */
    @Transactional(readOnly=true)
    public TUser login(String uname, String pwd) throws Exception {
        if(uname == null || pwd == null || uname.trim().equals("") || pwd.trim().equals("")) {
            throw new Exception("用户名或密码不能为空......");     }

        TUser user = userDao.login(uname, pwd);

        return user;
    }
```

数据库连接管理

- 数据库连接控制-事务方法控制

```java
/**
 * 声明性事务，保证员工和用户信息同时写入成功
 */
@Transactional(rollbackFor=Throwable.class)
public void addStaffUser(TStaff staff, TUser user) throws Exception {
    Log.logger.info("StaffBiz-->>addStaffUser()");

    staffDao.addStaff(staff);
    userDao.addUser(user);
}
```

```java
IStaff staffProxy = (IStaff)BeanFactory.getBean("staffBiz");
staffProxy.addStaffUser(staff, user);
System.out.println(staff.getSno() + "创建成功....");
```

使用了事务注解后

```
DEBUG - Creating new transaction with name [com.icss.biz.StaffBiz.addStaffUser]: PROPAGATION_REQUIRED,ISOLATION_DEFAUL
DEBUG - Creating new JDBC DriverManager Connection to [jdbc:mysql://localhost:3306/staff?useSSL=false&serverTimezone=U
DEBUG - Acquired Connection [com.mysql.cj.jdbc.ConnectionImpl@821a96] for JDBC transaction
DEBUG - Switching JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@821a96] to manual commit
DEBUG - Bound value [org.springframework.jdbc.datasource.ConnectionHolder@13fe6c9] for key [org.springframework.jdbc.d
DEBUG - Initializing transaction synchronization
DEBUG - Getting transaction for [com.icss.biz.StaffBiz.addStaffUser]
INFO - StaffBiz-->>addStaffUser()
DEBUG - Retrieved value [org.springframework.jdbc.datasource.ConnectionHolder@13fe6c9] for key [org.springframework.jd
INFO - StaffDao-->>addStaff():8526486
DEBUG - Retrieved value [org.springframework.jdbc.datasource.ConnectionHolder@13fe6c9] for key [org.springframework.jd
INFO - UserDao-->>addUser():8526486
DEBUG - Completing transaction for [com.icss.biz.StaffBiz.addStaffUser]
DEBUG - Triggering beforeCommit synchronization
DEBUG - Triggering beforeCompletion synchronization
DEBUG - Initiating transaction commit
DEBUG - Committing JDBC transaction on Connection [com.mysql.cj.jdbc.ConnectionImpl@821a96]
DEBUG - Triggering afterCommit synchronization
DEBUG - Clearing transaction synchronization
DEBUG - Triggering afterCompletion synchronization
DEBUG - Removed value [org.springframework.jdbc.datasource.ConnectionHolder@13fe6c9] for key [org.springframework.jdbc
DEBUG - Releasing JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@821a96] after transaction
DEBUG - Returning JDBC Connection to DataSource
12100070创建成功....
```

总结：使用@Transactional(rollbackFor=Throwable.class)控制事务，数据库连接唯一，事务环境唯一，数据库资源在业务方法结束后，正确释放。