第四章 Spring 框架（四）

课程目标

1、 上章回顾

2、 Spring-Jdbc 模板

3、 Spring 事务管理

课程内容

1、 上章回顾

==========

上午讲解内容：

==========

注解：

1）将 javaBean 放到 spring 容器中的注解：

@Repository

@Service

@Component

@Controller

2）从 spring 容器取出 javaBean 的注解：

@Autowired(byType)　@Qualifier(取时指定 bean 的名称)

@Resource（按名称取 byName，然后按类型取 byType）

3）关于 javaBean 生命周期的注解：

@PostConstruct　(相当于 bean 标签中的 init-method 属性)

@PreDestory（相当于 bean 标签中的 destory-method 属性）

4）关于 javaBean 作用范围的注解：

@Scope("request/session/singleton/prototype")

① singleton ： 代表生成的 javaBean 为单例的。

② prototype： 代表生成的 javaBean 为多例的。

5）关于取出属性文件中的值的注解：

①读取配置文件：

```
<context:property-placeholder      location="classpath:db.properties">
```

②在 java 代码中引用：

```
public class StudentServiceImpl implements StudentService{
    @Value("${pagesize}")
    private long pagesize;
    ...
}
```

==========

下午讲解内容：

==========

AOP：

1、代理：

1.1）静态代理：

原理：实现类与代理对象要实现同样的接口。

1.2）动态代理：

第一种：JDK 的动态代理：（需要目标对象要定义接口）

```
public class UserDaoImplJDKDynamicProxy implements InvocationHandler{
    private UserDao userDao;            //目标对象
    //创建代理对象
    public UserDao getInstance(UserDao userDao){
        this.userDao = userDao;
        return Proxy.newProxyInstance(
            userDao.getClass().getClassLoader(),
            userDao.getClass().getInterfaces(),
            this
        )
    }
    public Object invoke(Object obj,Method method,Object[] args){
        if(method.getName().equals("query")){
            return invoke(userDao,args);
        }
        checkSecurity();
        return invoke(userDao,args);
    }
  }
```

第二种：CGLIB 的动态代理：（需要目标对象可以被继承）

```
public class UserDaoImplCGLIBProxy implements MethodInceptor{
    //创建代理对象
    public static UserDao getInstance(){
        //1.创建一个 Enhancer 对象
        Enhancer enhancer = new Enhancer();
        //2.指定 EnHancer 对象的父类
```

```
            enhancer.setSuperClass(UserDaoImpl.class);
            //3.让 EnHancer 对象执行相应的方法回调
            enhancer.setCallback(this);
            //4.创建代理对象
            return enhancer.create();
      }
      //方法调用
      public Object invoke(Object proxy,Method method,Object[] args,MethodProxy
methodProxy){
            //如果是 query 对象，就调用其代理父类的方法
            if(method.getName().equals("query")){
                  return proxy.invokeSuper(obj, arg2);
            }
            checkSecurity();
            return proxy.invokeSuper(obj, arg2);
      }
        }
```

2、AOP：

2.1) 名词：

切入点：已经增强的方法，叫切入点。(pointcut)

连接点：可以增强的方法，叫连接点。(joinpoint)

通知：增强的代码。（advisor）

织入：将通知应用到切入点的过程，叫织入。（weaver）

切面：通知 + 切入点。（Aspect）

配置方法一：【xml 文件配置】

```
<aop:config>
<!--配置切入点-->
<aop:pointcut id="pointcut1" expression="execution(*
com.zelin.dao.impl.*DaoImpl.*(..))"/>
<!--配置切面-->
<aop:aspect ref="myAdvice">
      <aop:before method="before" pointcut-ref="pointcut1"/>
      ...
</aop:aspect>
  </aop:config>
```

配置方法二：【使用注解配置】

见《讲义》。

2、 Spring-Jdbc 模板（非常重要）

补充：

2.1）第一种配置方案：

2.1.1）在 applicationContext.xml 文件中配置连接及 jdbcTemplate

```
<!-- 配置扫描包 -->
<context:component-scan base-package="com.zelin"/>
<!-- 1.配置连接池 -->
<bean name="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <property name="driverClass" value="oracle.jdbc.driver.OracleDriver"/>
    <property name="jdbcUrl" value="jdbc:oracle:thin:@192.168.114.134:1521:orcl"/>
    <property name="user" value="scott"/>
    <property name="password" value="tiger"/>
</bean>
<!-- 2.配置jdbc模板 -->
<bean name="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource" ref="dataSource"/>
</bean>
```

2.1.2）定义 StudentDaoImpl.java 实现类：

```
@Repository("studentDao")
public class StudentDaoImpl implements StudentDao {

    @Resource
    private JdbcTemplate jdbcTemplate;
    @Override
    public void save(Student student) throws SQLException {
        String sql = "insert into student values(student_seq.nextval,?,?,?,?,?)";
        jdbcTemplate.update(sql, student.getSname(),student.getSex(),
                                 student.getAge(),student.getAddr(),
                                 student.getCid());

    }
```

将当前的dao放入到spring容器中

定义Jdbc模板对象

jdbcTemplate与dbutils框架用法在增、删、改时一样。

2.1.3）定义 StudentServiceImpl.java 实现类：

```
@Service("studentService")
public class StudentServiceImpl implements StudentService {
    @Resource
    private StudentDao studentDao;
    @Override
    public void save(Student student) throws SQLException {
        studentDao.save(student);
    }
```

将studentService对象放到Spring容器中

从spring容器中取出名为"studentDao"的javabean

2.1.4）测试 JDBCTemplate：

```
//整合spring与junit4
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class TestStudentService {

    @Resource
    private StudentService studentService;
    @Test
    public void testSave() throws SQLException{
        Student student = new Student("小明", "男", 23, "岳阳");
        student.setCid(1);
        studentService.save(student );
        System.out.println("添加成功！");
    }
}
```

加载Spring容器，读取配置文件。

从容器中取出名为studentService的bean对象。

2.1.5）运行结果：

```
log4j:WARN No appenders could b
log4j:WARN Please initialize th
添加成功！
```

```
SQL> select * from student;

       SID SNAME                SE        AGE ADDR
---------- -------------------- -- ---------- ----------------------
--------
      1005 王五                 男         20 深圳
      1001 张三                 男         21 上海
      1002 李四                 男         20 杭州
      1003 赵六                 女         19 广州
      1006 小明                 男         23 岳阳

SQL>
```

2.2）第二种配置方案：

2.2.1)在 applicationContext.xml 文件中配置 Dao 及依赖的 DataSource

```
<!-- 配置扫描包 -->
    <context:component-scan base-package="com.zelin"/>
    <!-- 1.配置连接池 -->
    <bean name="dataSource"
class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass"
value="oracle.jdbc.driver.OracleDriver"/>
        <property name="jdbcUrl"
value="jdbc:oracle:thin:@192.168.114.134:1521:orcl"/>
        <property name="user" value="scott"/>
        <property name="password" value="tiger"/>
    </bean>
    <!-- 2.配置 jdbc 模板 -->
    <bean name="jdbcTemplate"
class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource"/>
    </bean>
    <!-- 3.配置 dao -->
    <bean name="classesDao" class="com.zelin.dao.impl.ClassesDaoImpl">
        <property name="dataSource" ref="dataSource"/>
    </bean>
```

2.2.2 ）ClassesDaoImpl.java 类的定义：

```
public class ClassesDaoImpl extends JdbcDaoSupport implements
ClassesDao,RowMapper<Classes> {
    @Override
    public void save(Classes clazz) throws SQLException {
        getJdbcTemplate().update("insert into classes values(?,?)",
```

```
    clazz.getCid(),clazz.getCname());
    }
    @Override
    public void update(Classes clazz) throws SQLException {
        getJdbcTemplate().update("update classes set cname = ? where cid = ?",
                                 clazz.getCname(),clazz.getCid());
    }
    @Override
    public void delete(int cid) throws SQLException {
        getJdbcTemplate().update("delete from classes where cid = ?",cid);
    }
    @Override
    public List<Classes> findAll() throws SQLException {
        return getJdbcTemplate().query("select * from classes", this);
    }
    @Override
    public Classes findClassesByCid(int cid) throws SQLException {
        return getJdbcTemplate().queryForObject("select * from classes where
cid=?", this,cid);
    }
    //实现 jdbcTemplate 查询的映射方法(作用：将 ResultSet 转换为 Classes 对象)
    @Override
    public Classes mapRow(ResultSet rs, int rowNum) throws SQLException {
        return new Classes(rs.getInt("cid"),rs.getString("cname"));
    }
}
```

2.2.3）测试代码：

```
public class TestClassesService {
    @Resource
    private ClassesService classesService;
    @Test
    //第一部分：增删改测试
    public void testSave() throws SQLException{
        //Classes clazz = new Classes(4,"1304 班");
        /*classesService.update(clazz);
        System.out.println("修改成功！");*/
        /*classesService.save(clazz);
        System.out.println("添加成功！");*/
        //classesService.delete(4);
        //System.out.println("删除成功！");

    }
    //第二部分：查询测试
```

```
    @Test
    public void testFind() throws SQLException{
        //2.1)查询所有班级
        /*List<Classes> classes = classesService.findAll();
        for(Classes c:classes){
            System.out.println(c);
        }*/
        //2.2)查询单个班级
        Classes classes = classesService.findClassesByCid(3);
        System.out.println(classes);
    }
}
```

2.2.4）运行结果：

查询所有班级：

```
log4j:WARN Please initialize the log4j system proper
Classes [cid=1, cname=1301班]
Classes [cid=2, cname=1302班]
Classes [cid=3, cname=Java-1303班]
```

查询单个班级：

```
log4j:WARN No appenders could be found for logger
log4j:WARN Please initialize the log4j system pro
Classes [cid=3, cname=Java-1303班]
```

2.3）第三种配置方案：

2.3.1) 定义 StudentDaoImpl2.java 类：

```
/*****************************************
 * 公司：  深圳市泽林信息公司              <br>
 * 作者：  王峰                           <br>
 * 类名：  StudentDaoImpl                 <br>
 * 日期：  2018 年 8 月 29 日 上午 9:45:36   <br>
 * 功能：
 *****************************************/

//第二种使用 jdbcTemplate 的方法
@Repository("studentDao2")
public class StudentDaoImpl2 extends JdbcDaoSupport implements
StudentDao ,RowMapper<Student>{
```

```
    @Autowired
    private DataSource dataSource;
    @PostConstruct
    public void init(){
        setDataSource(dataSource);
    }
    @Override
    public void add(Student student) throws Exception {
        getJdbcTemplate().update("insert into student values(null,?,?,?,?,?,?)",
                    student.getSname(),student.getSex(),student.getAge(),
                    student.getAddr(),student.getCid(),student.getBirth());
    }
    @Override
    public void update(Student student) throws Exception {
        getJdbcTemplate().update("update student set
sname=?,sex=?,age=?,cid=?,addr=?,birth=? where sid=?",

    student.getSname(),student.getSex(),student.getAge(),

    student.getCid(),student.getAddr(),student.getBirth(),
                                student.getSid());
    }
    @Override
    public void delete(int sid) throws Exception {
        getJdbcTemplate().update("delete from student where sid = ?" ,sid);
    }
    @Override
    public List<Student> findAll() throws Exception {
        return getJdbcTemplate().query("select * from student", this);
    }
    //注意：查询的两个方法的应用
    @Override
    public Student findStudentBySid(int sid) throws Exception {
        return getJdbcTemplate().queryForObject("select * from student where sid=?",
this,sid);
    }
    //此方法为 RowMapper 接品的抽象方法，其作用就是将查询得到的结果集转换对应的实体类
    @Override
    public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
        return new Student(rs.getInt("sid"), rs.getString("sname"),
rs.getString("sex"),
                           rs.getInt("age"), rs.getString("addr"), rs.getInt("cid"),
                           rs.getString("birth"));
    }
```

```
    //查询学生数量
    @Override
    public Long findCount() throws Exception {
         return (Long) getJdbcTemplate().queryForObject("select count(*) as aa from
student",
                 new RowMapper<Long>(){
                    @Override
                    public Long mapRow(ResultSet rs, int rowNum) throws SQLException
{
                        return rs.getLong(1);
                    }
             });
    }
}
```

2.3.2) StudentServiceImpl.java:

```
@Service("studentService2")
public class StudentServiceImpl2 implements StudentService {
    @Resource
    private StudentDao studentDao2;
    @Override
    public void add(Student student) throws Exception {
        studentDao2.add(student);
    }
    @Override
    public void update(Student student) throws Exception {
        studentDao2.update(student);
    }
    @Override
    public void delete(int sid) throws Exception {
        studentDao2.delete(sid);
    }
    @Override
    public List<Student> findAll() throws Exception {
        return studentDao2.findAll();
    }
    @Override
    public Student findStudentBySid(int sid) throws Exception {
        return studentDao2.findStudentBySid(sid);
    }
    @Override
    public Long findCount() throws Exception {
        return studentDao2.findCount();
    }
```

```
}
```

2.3.3) 配置 applicationContext-02.xml 文件：

```
<?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-4.2.xsd
        http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.2.xsd">
    <!-- 0.指定扫描包 -->
    <context:component-scan base-package="com.zelin"/>
    <!-- 1.读取数据库访问的属性文件 -->
    <context:property-placeholder location="classpath:db.properties"/>
    <!-- 2.配置数据源 -->
    <bean name="dataSource"
class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="${db.driver}"/>
        <property name="jdbcUrl" value="${db.url}"/>
        <property name="user" value="${db.user}"/>
        <property name="password" value="${db.password}"/>
    </bean>
    <!-- 3.配置 JdbcTemplate -->
    <!-- <bean name="jdbcTemplate"
class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource"/>
    </bean> -->
</beans>
```

2.3.4) 进行单元测试：

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext-02.xml")
public class TestStudentService2 {
    @Resource
    private StudentService studentService2;
    //测试添加学生
    @Test
    public void testAdd() throws Exception {
        String sname = "胡大海";
        String sex = "男";
        int age = 110;
```

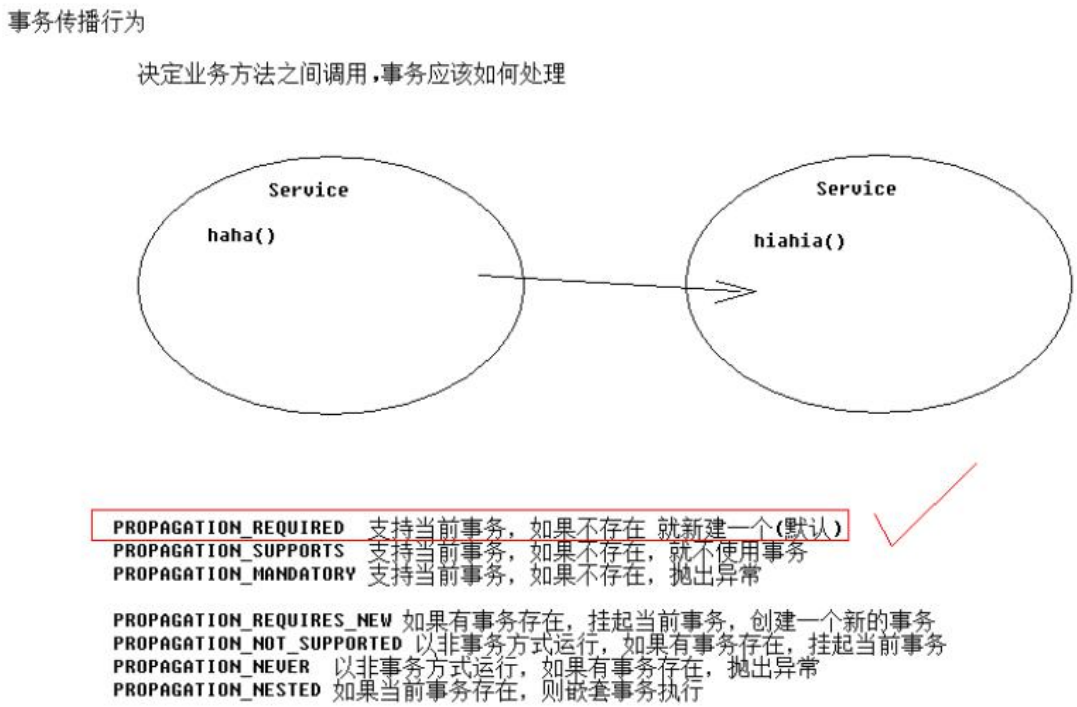
```
        String addr = "山东";
        int cid = 2;
        String birth = "1908-2-28";
        Student student = new Student(sname, sex, age, addr, cid, birth);
        studentService2.add(student);
        System.out.println("添加学生成功！");
    }
    //测试修改学生
    @Test
    public void testUpdate() throws Exception {
        String sname = "胡大海 111";
        String sex = "男";
        int age = 120;
        String addr = "东莞";
        int cid = 2;
        String birth = "1808-2-28";
        Student student = new Student(23,sname, sex, age, addr, cid, birth);
        studentService2.update(student);
        System.out.println("修改成功！");
    }
    //测试删除学生
    @Test
    public void testDelete() throws Exception {
        studentService2.delete(23);
        System.out.println("删除成功！");
    }
    //查询所有
    @Test
    public void testFindAll() throws Exception {
        List<Student> students = studentService2.findAll();
        for (Student student : students) {
            System.out.println(student);
        }
    }
    //查询所有
    @Test
    public void testFindStudentBySid() throws Exception {
        Student student = studentService2.findStudentBySid(22);
        System.out.println(student);
    }
    //查询学生数量
    @Test
    public void testFindCount() throws Exception {
        Long count = studentService2.findCount();
```

```
        System.out.println(count);
    }
}
```

3、 Spring 事务管理

第一种方式：使用 XML 配置实现：（非常重要）

3.1) 了解事务事务的传播行为：



3.2) UsersDaoImpl.java 编写：

```
public class UsersDaoImpl extends JdbcDaoSupport implements UsersDao {
    //转出
    @Override
    public void outMoney(String from, int money) throws SQLException {
        String sql = "update users set balance = balance - ? where id = ?";
        getJdbcTemplate().update(sql, money,from);
    }
    //转入
    @Override
    public void inMoney(String to, int money) throws SQLException {
```

```
        String sql = "update users set balance = balance + ? where id = ?";
        getJdbcTemplate().update(sql,money,to);
    }
}
```

3.3) UsersServiceImpl.java 编写：

```
@Service("usersService")
public class UsersServiceImpl implements UsersService {

    @Resource
    private UsersDao usersDao;
    @Override
    public void transfer(String from, String to, int money) throws
SQLException {
        //转出
        usersDao.outMoney(from, money);
        //int c = 10 / 0;
        //转入
        usersDao.inMoney(to, money);
    }
}
```

3.4) applicationContext3.xml 编写：

```
<!-- 配置扫描包 -->
    <context:component-scan base-package="com.zelin"/>
    <!-- 读取属性文件 -->
    <context:property-placeholder location="classpath:db.properties"/>
    <!-- 1.配置连接池 -->
    <bean name="dataSource"
class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="${db.driver}"/>
        <property name="jdbcUrl" value="${db.url}"/>
        <property name="user" value="${db.user}"/>
        <property name="password" value="${db.pwd}"/>
    </bean>
    <!-- 2.配置 Dao -->
    <bean name="usersDao" class="com.zelin.dao.impl.UsersDaoImpl">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <!-- 3.开始配置事务相关 -->
```

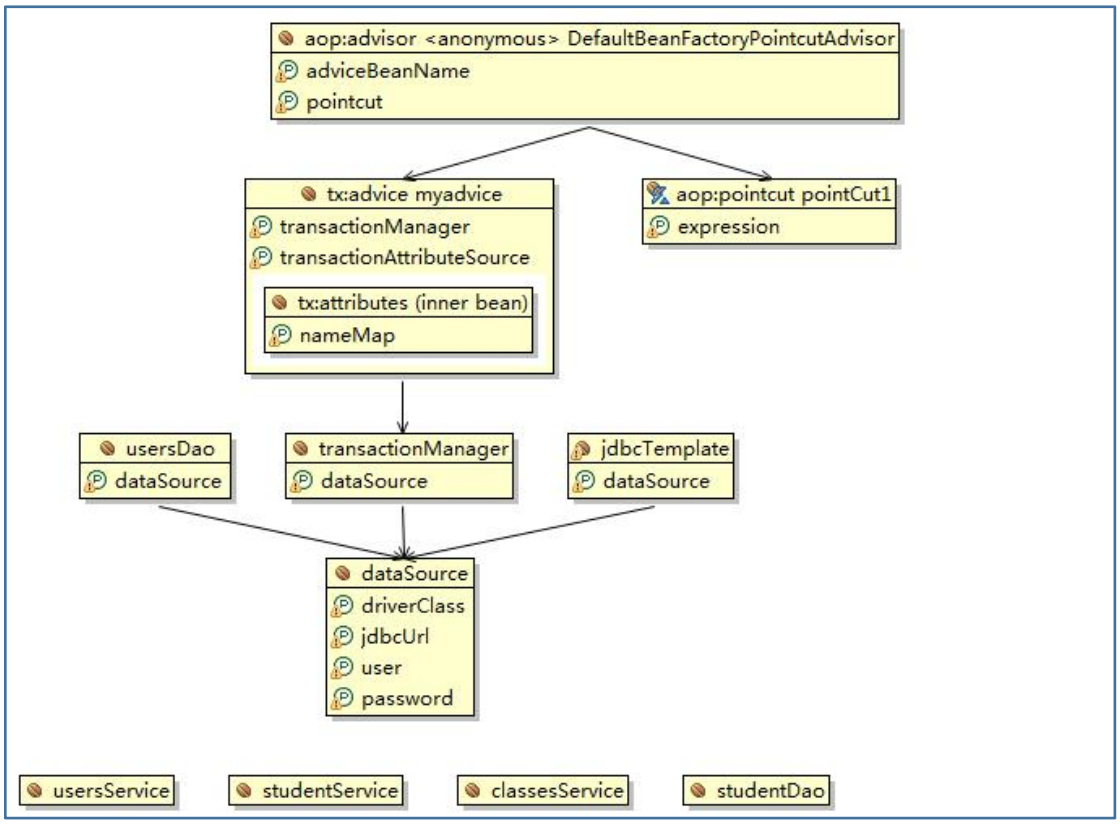
```
    <!-- 3.1)配置事务管理器 -->
    <!-- DataSourceTransactionManager：作用于 JDBC 或 MyBatis，作用于 Hihbernate
的叫 ： HibernateTransactionManager -->
    <bean name="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>
    <!-- 配置通知 (对事务管理器增强)-->
    <tx:advice id="myadvice" transaction-manager="transactionManager">
        <tx:attributes>
            <tx:method name="save*" propagation="REQUIRED"
    read-only="false"/>
            <tx:method name="insert*" propagation="REQUIRED"
    read-only="false"/>
            <tx:method name="update*" propagation="REQUIRED"
    read-only="false"/>
            <tx:method name="modify*" propagation="REQUIRED"
    read-only="false"/>
            <tx:method name="delete*" propagation="REQUIRED"
    read-only="false"/>
            <tx:method name="remove*" propagation="REQUIRED"
    read-only="false"/>
            <tx:method name="find*" propagation="REQUIRED" read-only="true"/>
            <tx:method name="get*" propagation="REQUIRED" read-only="true"/>
            <tx:method name="transfer" propagation="REQUIRED"
read-only="false"/>
        </tx:attributes>
    </tx:advice>
    <!-- 3.2)使用 aop 配置事务管理器 -->
    <aop:config>
        <!-- 3.3)配置切入点表达式 -->
        <aop:pointcut expression="execution(*
    com.zelin.service.impl.*ServiceImpl.*(..))" id="pointCut1"/>
        <!-- 3.4)配置切面(切入点 + 通知) -->
        <aop:advisor advice-ref="myadvice" pointcut-ref="pointCut1"/>
    </aop:config>
    <!-- 引入其它模块 -->
    <import resource="jdbcTemplateConfig.xml"/>
```

3.5) jdbcTemplateConfig.xml 编写：

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

```
        <property name="dataSource" ref="dataSource"/>
</bean>
```

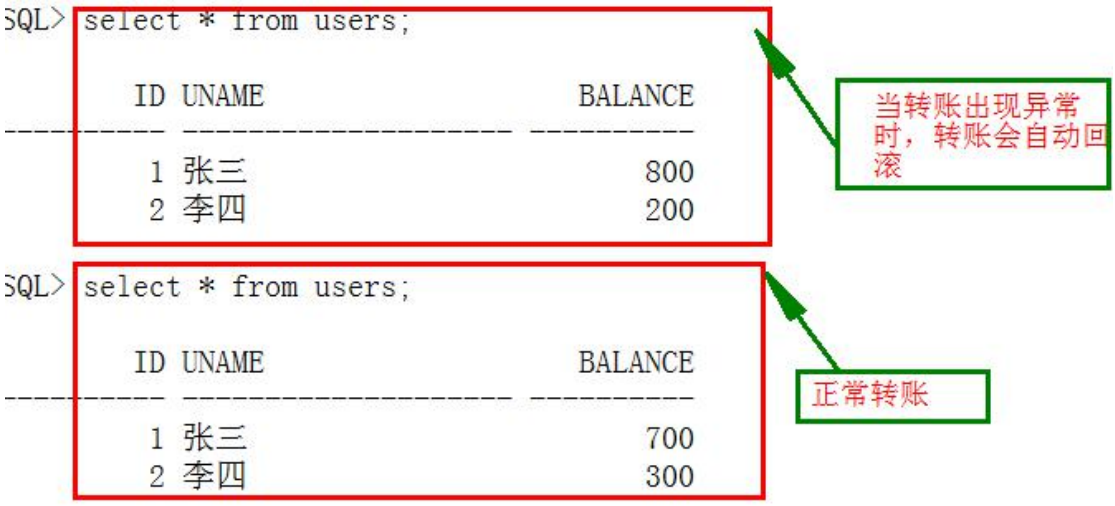
3.6) Beans Graph 图例：



3.7) 测试代码：

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext*.xml")
public class TestSpringTx {
    @Resource
    private UsersService usersService;
    @Test
    public void test01() throws SQLException{
        usersService.transfer("1", "2", 100);
    }
}
```

3.8) 效果如下：



第二种方式：使用注解实现：（非常重要）

3.1）在 applicationContext4.xml 文件中添加关于注解的配置：

```
<!-- 3.开始配置事务相关 -->
    <!-- Spring 事务管理方式二：注解式事务配置 -->
    <!-- 3.1)配置事务管理器 -->
    <!-- DataSourceTransactionManager：作用于 JDBC 或 MyBatis，作用于
Hihbernate 的叫 ：HibernateTransactionManager -->
    <bean name="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager
">
        <property name="dataSource" ref="dataSource"/>
    </bean>
    <!--3.2) 添加事务驱动 -->
    <tx:annotation-driven/>
```

3.2）在 UsersServiceImpl.java 层添加关于事务的注解：

```
@Service("usersService")
@Transactional
public class UsersServiceImpl implements UsersService {
    @Resource
    private UsersDao usersDao;
```

```
    @Override
    public void transfer(String from, String to, int money) throws
SQLException {
        //转出
        usersDao.outMoney(from, money);
        int c = 10 / 0;
        //转入
        usersDao.inMoney(to, money);
    }
    @Transactional(readOnly=true)
    public void findAll(){    }
}
```

3.3）测试代码、运行效果同上。