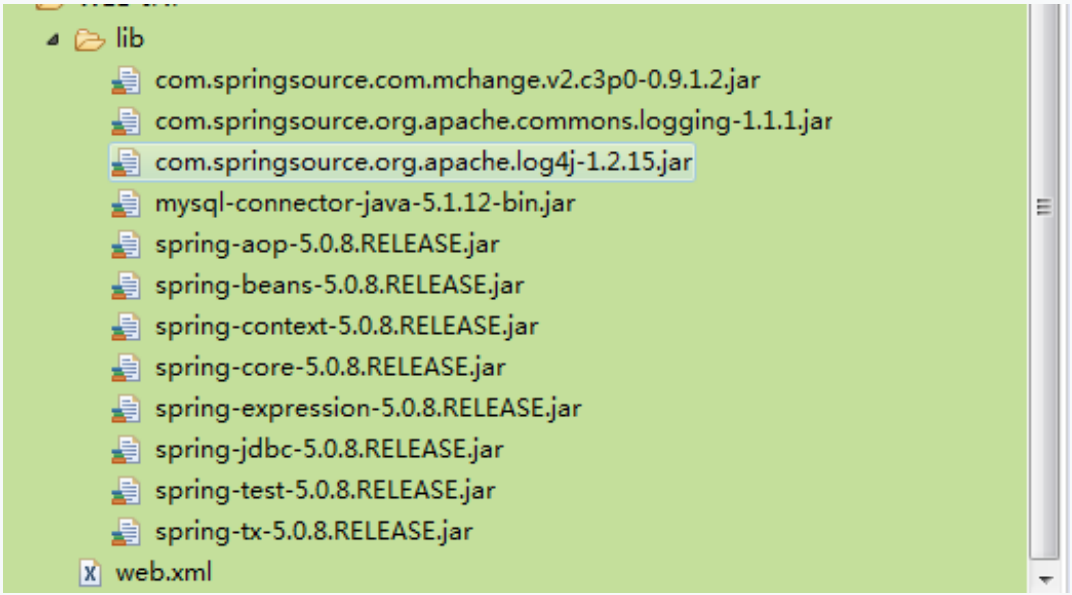
18 Spring_事务管理_HelloWord

`Spring`

一:准备工作

1、 ：导入jar包



2、实体类

```
public class Users {
  private  Integer  id;
  private String  name;
  private  float  money;
  //对应的get set 方法
  }
```

3、dao实现类

> 1、接口就为两个抽象方法

```
import org.springframework.jdbc.core.support.JdbcDaoSupport;
import com.spring.Dao.IUsersDao;
public class UsersDaoImp extends JdbcDaoSupport implements IUsersDao {
    @Override
    public void addMoney(Integer id, Float money)  {

        String sql="update users set money=money+? where id=?";
         super.getJdbcTemplate().update(sql,money,id);
    }
    @Override
    public void updateMoney(Integer id, Float money) {
        // TODO Auto-generated method stub
        String sql="update users set money=money-? where id=?";
         super.getJdbcTemplate().update(sql,money,id);
    }
}
```

4、Serivce实现类

> 1、接口就为一个抽象方法

```
import com.spring.Dao.IUsersDao;
import com.spring.serivce.IUsersSerivce;
public class UsersSerivceImp implements IUsersSerivce {

    // 必须有get set 方法
    private IUsersDao usersDao;

    @Override
    public void getTran(Integer in, Integer in2, Float money)  {
        usersDao.updateMoney(in, money);
        usersDao.addMoney(in2, money);

    }

    public IUsersDao getUsersDao() {
        return usersDao;
    }
```

```

    public void setUsersDao(IUsersDao usersDao) {
        this.usersDao = usersDao;
    }
}
```

5、配置文件

```
    <!--指定读取properties文件 -->
    <context:property-placeholder location="classpath:db.porperties" />
    <!--1.将连接池放入Spring容器中 -->
    <bean name="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="${jdbc.driverClass}"></property>
  <property name="jdbcUrl" value="${jdbc.jdbcUrl}"></property>
        <property name="user" value="${jdbc.user}"></property>
  <property name="password" value="${jdbc.password}"></property>
    </bean>
    <!--2、将IUsesDao放入Spring容器中 -->
    <bean name="usersDao" class="com.spring.Dao.imp.UsersDaoImp">
        <!-- <property name="jt" ref="jdbcTemplate"></property> -->
        <property name="dataSource" ref="dataSource"></property>
    </bean>
    <!--3、将usersSerivce放入Spring容器中 -->
    <bean name="usersSerivce" class="com.spring.serivce.impl.UsersSerivceImp">
        <property name="usersDao" ref="usersDao"></property>
    </bean>
```

6、测试类

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class UserDemo {

      @Resource(name="usersSerivce")
      private IUsersSerivce  usersSerivce;
    @Test
    public void getVoid01() throws Exception{
```

```
        usersSerivce.getTran(15,16,1000F);
    }
}
```

二:事务的管理方式

> 1、测试同上

1、方式一_编码 Transaction模板(了解)

1-1、核心的事务管理器

> 1、将核心的事务管理器配置到容器中

```
<!--将核心的事务管理器配置到容器中 -->
    <bean name="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"></property>
    </bean>
    <!-- 事务模版对象 -->
    <bean name="transactionTemplate" class="org.springframework.transaction.support.TransactionTemplate">
        <property name="transactionManager" ref="transactionManager"></property>
    </bean>
```

1-2、模版对像

> 1、将事务模版对象 组装到Serivce层

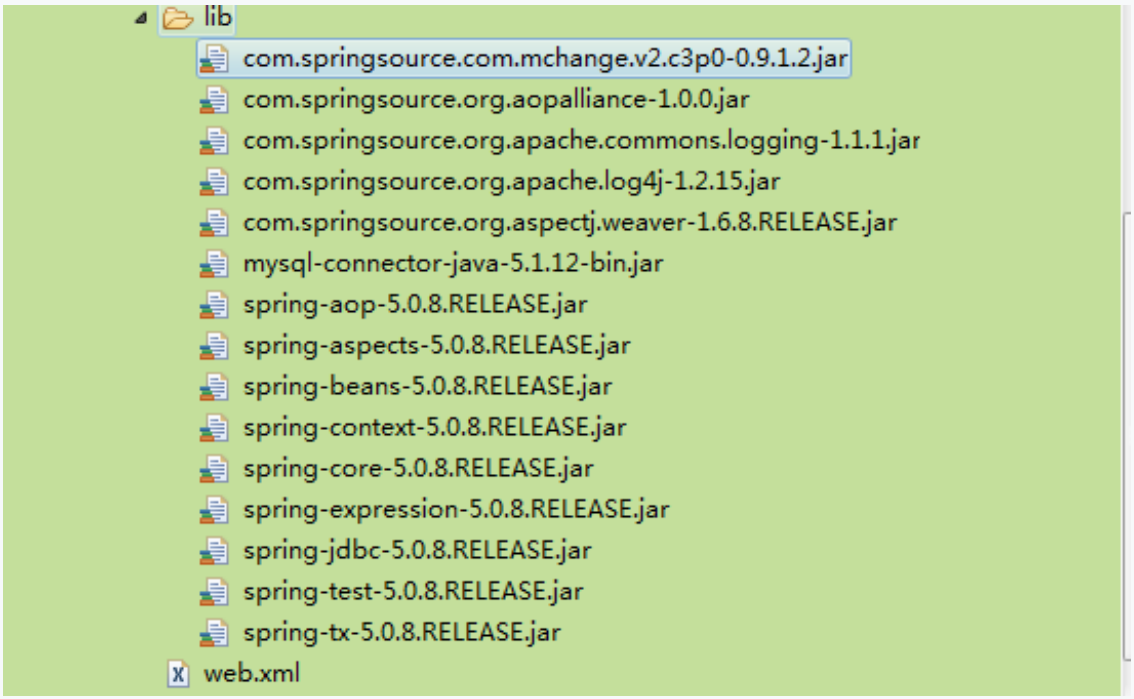
```
import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.support.TransactionCallbackWithoutResult;
import org.springframework.transaction.support.TransactionTemplate;
```

```java
import com.spring.Dao.IUsersDao;
import com.spring.serivce.IUsersSerivce;
public class UsersSerivceImp implements IUsersSerivce {
    // 必须有get set 方法
    private IUsersDao usersDao;
    private TransactionTemplate tt;

    @Override
    public void getTran(final Integer in, final Integer in2, final Float money) throws Exception {

        tt.execute(new TransactionCallbackWithoutResult() {
            @Override
            protected void doInTransactionWithoutResult(TransactionStatus arg0) {

                    usersDao.updateMoney(in, money);
                     int i=1/0;
                    usersDao.addMoney(in2, money);
            }
        });
    }
```

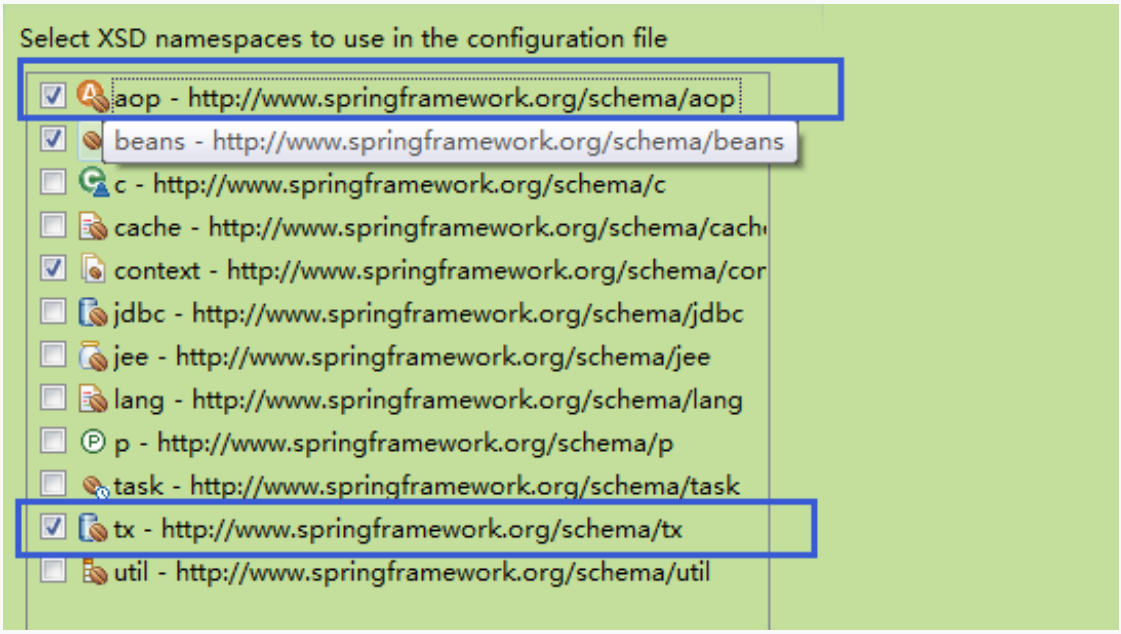
```xml
    <!--3、将usersSerivce放入Spring容器中 -->
    <bean name="usersSerivce" class="com.spring.serivce.impl.UsersSerivceImp">
        <property name="usersDao" ref="usersDao"></property>
        <!--事务模版对象 组装到Serivce层  -->
        <property name="tt" ref="transactionTemplate"></property>
    </bean>
```

2、方式二_xml配置aop事务(重点)

2-1、到如Jar包



2-2、导入tx && aop冥名空间



2-3、配置通知

```
1.      <!--将核心的事务管理器配置到容器中 -->
2.       <bean name="transactionManager"
3.
class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
>
4.          <property name="dataSource" ref="dataSource"></property>
```

```
    </bean>
   <!--配置事务通知-->
   <tx:advice id="txAdvice" transaction-manager="transactionManager">
       <tx:attributes>
       <!--
        指定方法以什么样的事务进行执行
        isolation:隔离级别
           propagation :事务的传播行为
           read-only:配置是否只读
           -->
           <!--  公司的配置方式-->
         <tx:method name="save*"  isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="false"/>
         <tx:method name="persist*"  isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="false"/>
         <tx:method name="delete*"  isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="false"/>
         <tx:method name="remove*"  isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="false"/>
         <tx:method name="update*"  isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="false"/>
         <tx:method name="modify*"  isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="false"/>
         <!--read-only: 只读 true  -->
         <tx:method name="get*"  isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="true"/>
         <tx:method name="find*"  isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="true"/>

         <tx:method name="getTran"  isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="false"/>
       </tx:attributes>
   </tx:advice>
```

2-4、配置将通知织入目标

```
<!--配置织入  -->
<aop:config>
  <aop:pointcut expression="execution(* com.spring.serivce.impl.*.*(..))" id="txPointcut"/>
   <!-- 配置切面 (通知+切点)
    advice-ref:配置通知的名称
    pointcut-ref:配置 切点的名称
    -->
  <aop:advisor advice-ref="txAdvice" pointcut-ref="txPointcut" />
```

```
</aop:config>
```

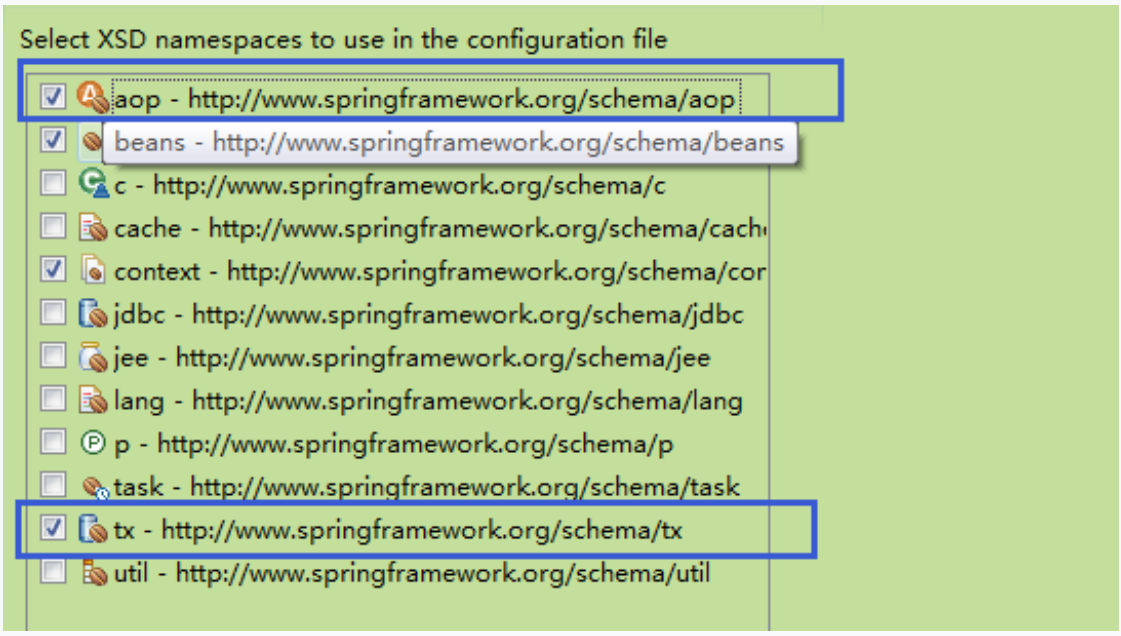
```
    <!--2、将IUsesDao放入Spring容器中 -->
    <bean name="usersDao" class="com.spring.Dao.imp.UsersDaoImp">
        <!-- <property name="jt" ref="jdbcTemplate"></property> -->
        <property name="dataSource" ref="dataSource"></property>
    </bean>
    <!--3、将usersSerivce放入Spring容器中 -->
    <bean name="usersSerivce" class="com.spring.serivce.impl.UsersSerivceImp">
        <property name="usersDao" ref="usersDao"></property>
        <!--事务模版对象 组装到Serivce层  -->
    </bean>
```

3、方式三_解配置aop事务(重点)

> 1、导包同上

3-1、到如Jar包

3-2、导入tx && aop冥名空间



3-3、配置开启注解

```
    <!--开启使用注解事务  -->
   <tx:annotation-driven/>
```

3-4、使用注解

> 1、方式一：整个类上实现该事务

```
@Transactional(isolation=Isolation.REPEATABLE_READ,propagation=Propagation.REQUIRED,readOnly=true)
public class UsersSerivceImp implements IUsersSerivce {
    // 必须有get set 方法
    private IUsersDao usersDao;
    public void getTran(final Integer in, final Integer in2, final Float money) throws Exception {
        usersDao.updateMoney(in, money);
        /*int i=1/0; */
        usersDao.addMoney(in2, money);
    }
}
```

> 2、方式二：指定方法使用该事务注解

```java
public class UsersSerivceImp implements IUsersSerivce {
    // 必须有get set 方法
    private IUsersDao usersDao;
    @Override

@Transactional(isolation=Isolation.REPEATABLE_READ,propagation=Propagation.REQUIRED,readOnly=false)
    public void getTran(final Integer in, final Integer in2, final Float money) throws Exception {
        usersDao.updateMoney(in, money);
        /*int i=1/0; */
        usersDao.addMoney(in2, money);
    }
    }
```

> 3、如在类上使用事务注解,中有方法需要的事务属性不一致,可以方法上在设置事务属性