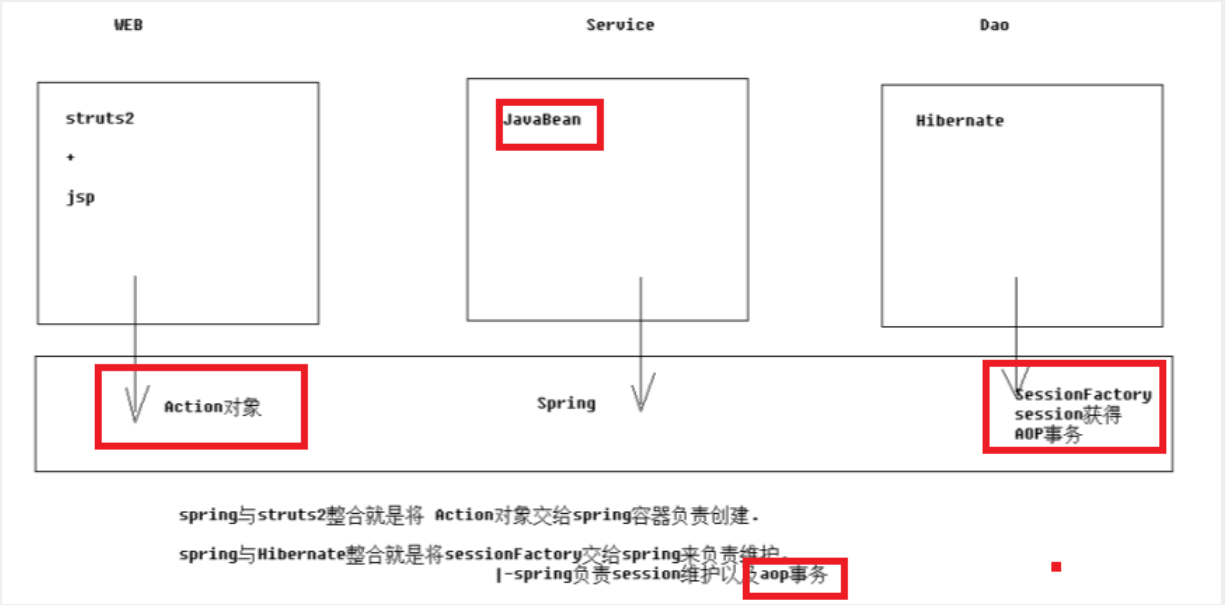
整合SSH三大框架

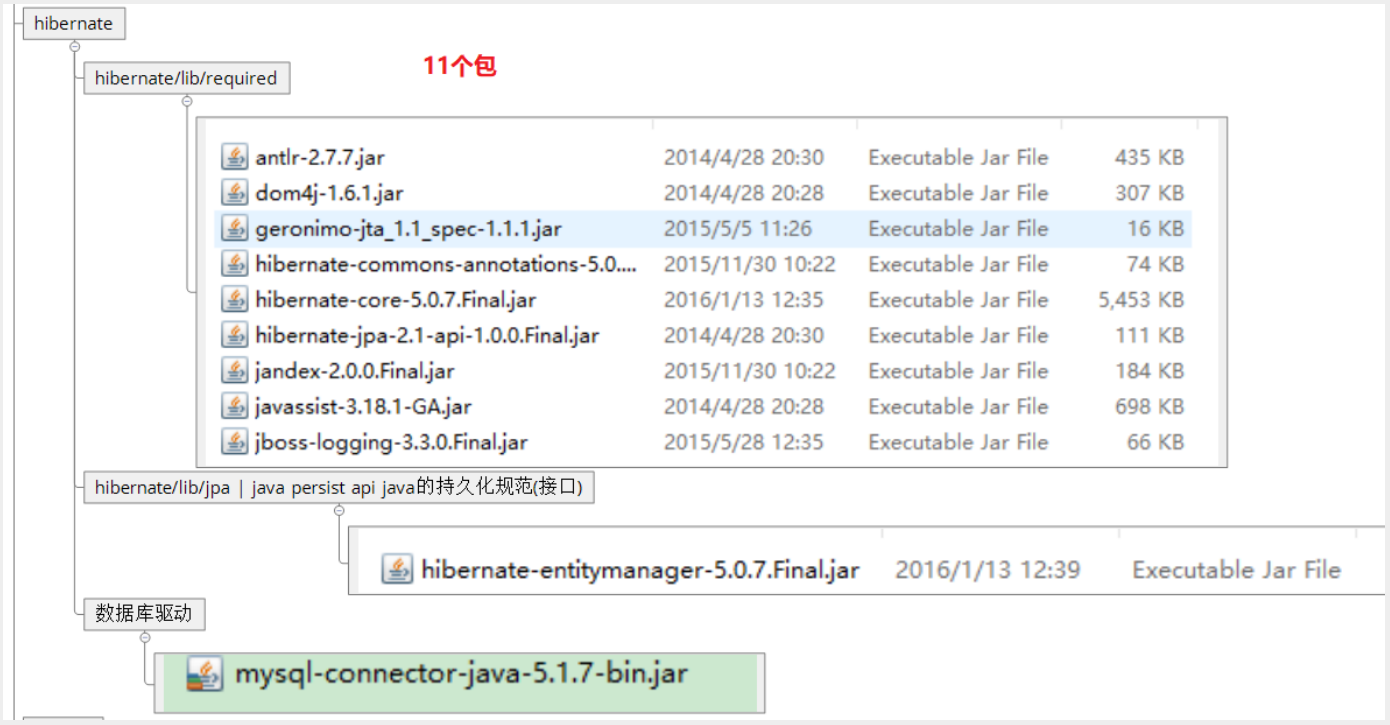
一、整合原理及导包

1.1 原理

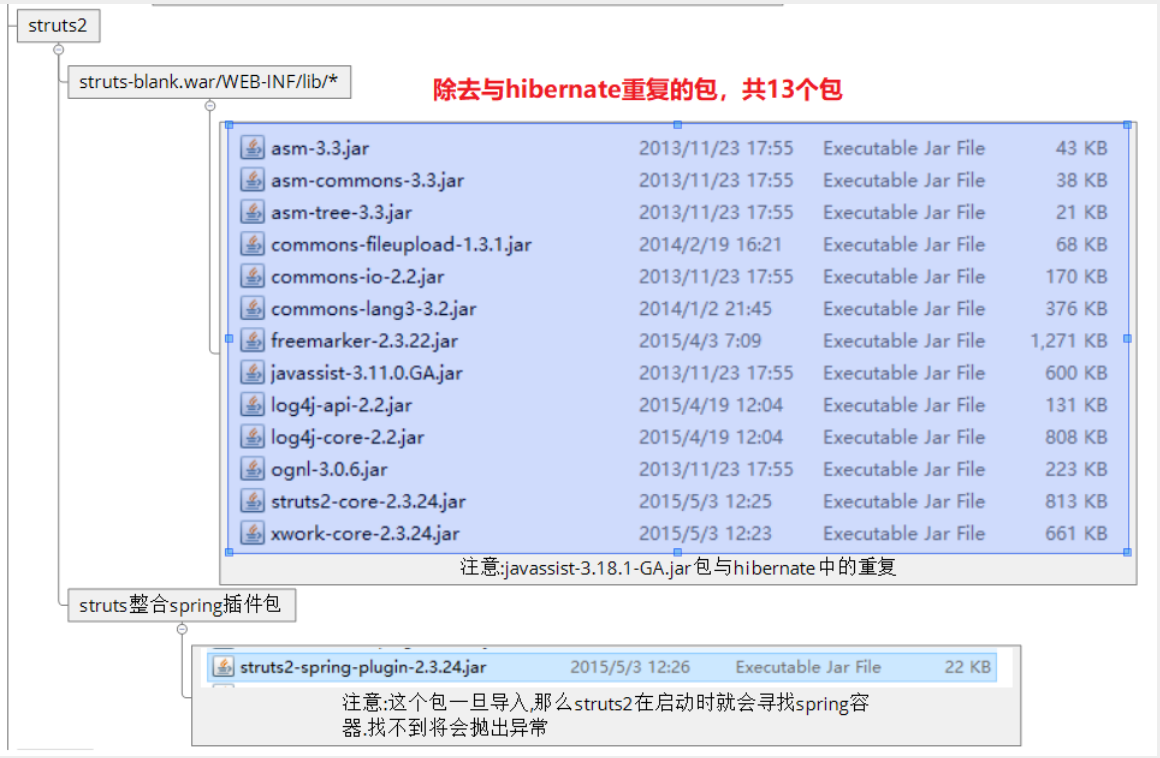


1.2 导包

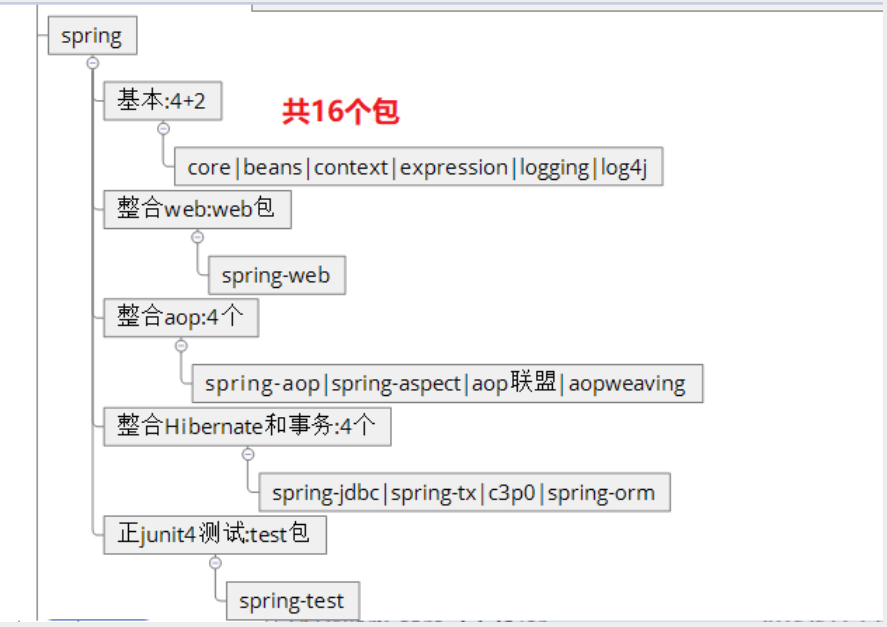
1.2.1 hibernate的包



1.2.2 struts2的包



1.2.3 spring的包



1.2.4 其它包

二、配置spring容器随项目启动

2.1 创建配置文件applicationContext.xml并导入约束

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns="http://www.springframework.org/schema/beans"
       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-4.2.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
                           http://www.springframework.org/schema/tx
                           http://www.springframework.org/schema/tx/spring-tx-4.2.xsd ">
```

2.2 在web.xml中配置spring容器随项目启动

```xml
<!-- 让spring随web启动而创建的监听器 -->
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<!-- 配置spring配置文件位置参数 -->
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>classpath:applicationContext.xml</param-value>
</context-param>
```

三、整合struts2

3.1 单独配置struts(相当于struts2单独用时后的基本配置)

单独配置struts2

配置struts2主配置文件

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>
    <package name="crm" namespace="/" extends="struts-default" >
        <action name="UserAction_*" class="cn.itcast.web.action.UserAction" method="{1}" >
            <result name="success" >/success.jsp</result>
        </action>
    </package>
</struts>
```

配置struts2核心过滤器到web.xml

```xml
<!-- struts2核心过滤器 -->

<filter>
  <filter-name>struts2</filter-name>
  <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>struts2</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

3.2 与spring整合

3.2.0 步骤零：创建action类并配置到applicationContext.xml中

```xml
<!-- 注意:Action对象作用范围一定是多例的.这样才符合struts2架构 -->
<bean name="userAction" class="cn.scct.web.action.UserAction" scope="prototype" >
    <property name="userService" ref="userService" ></property>
</bean>
                                                    //注意一定要配置成多例
```

3.2.1 步骤一：在struts.xml中配置常量(为了把struts2交给spring管理)

```xml
<!-- #  struts.objectFactory = spring    将action的创建交给spring容器
        struts.objectFactory.spring.autoWire = name  spring负责装配Action依赖属性
        -->
<constant name="struts.objectFactory" value="spring"></constant>
```

3.2.2 步骤二：在struts.xml中配置action类

3.2.2.1 方案一：struts2负责创建action，spring负责组装依赖

在struts.xml中配置

```xml
<package name="crm" namespace="/" extends="struts-default" >

    <action name="UserAction_*" class="cn.scct.web.action.UserAction" method="{1}" >
        <result name="success" >/success.jsp</result>
    </action>
</package>
```

3.2.2.2 方案二(推荐)：spring负责创建和组装依赖

```
<package name="crm" namespace="/" extends="struts-default" >
<!--
    <action name="UserAction_*" class="cn.scct.web.action.UserAction" method="{1}" >
        <result name="success" >/success.jsp</result>
    </action> -->
    <!--
        整合方案2:class属性上填写spring中action对象的BeanName
            完全由spring管理action生命周期,包括Action的创建
            注意:需要手动组装依赖属性
      -->
    <action name="UserAction_*" class="userAction" method="{1}" >
        <result name="success" >/success.jsp</result>
    </action>

</package>
```

3.2.3 步骤三：测试

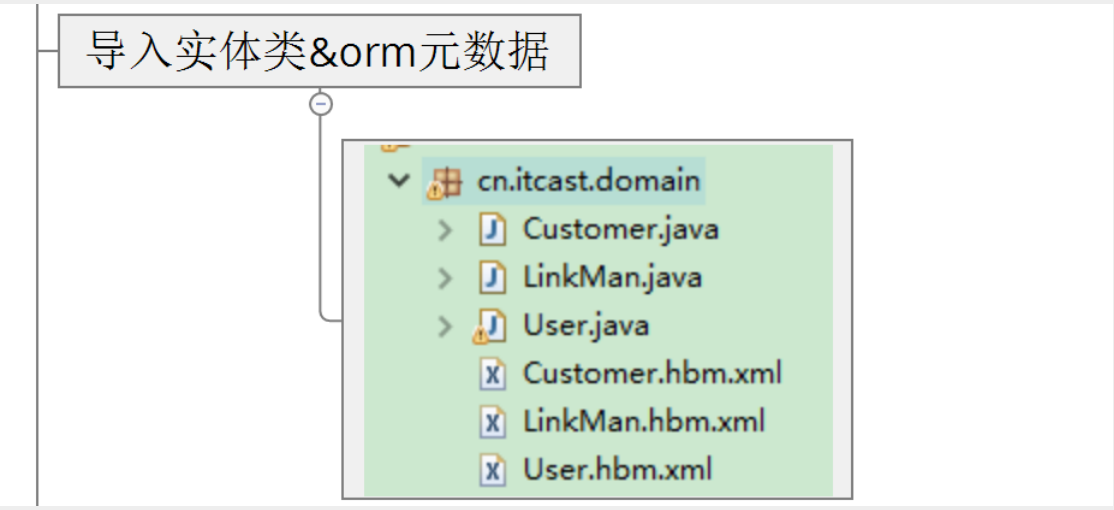
启动服务器直接访问该action类的某个方法看是否能够成功

四、整合hibernate

4.1 单独配置hibernate(需要自己创建SessionFactory对象)

4.1.1 创建实体类和映射文件

hibernate最基本的东西



4.1.2 创建核心配置文件hibernate.cfg.xml

```xml
<!-- 数据库驱动 -->
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<!-- 数据库url -->
<property name="hibernate.connection.url">jdbc:mysql:///crm_32</property>
<!-- 数据库连接用户名 -->
<property name="hibernate.connection.username">root</property>
<!-- 数据库连接密码 -->
<property name="hibernate.connection.password">1234</property>
<!-- 数据库方言
    注意：MYSQL在选择方言时,请选择最短的方言.
-->
<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
```

```xml
<!-- 将hibernate生成的sql语句打印到控制台 -->
<property name="hibernate.show_sql">true</property>
<!-- 将hibernate生成的sql语句格式化(语法缩进) -->
<property name="hibernate.format_sql">true</property>
<!--
自动导出表结构. 自动建表
 -->
<property name="hibernate.hbm2ddl.auto">update</property>
```

```xml
<!-- 引入实体配置文件 -->
<mapping resource="cn/itcast/domain/Customer.hbm.xml" />
<mapping resource="cn/itcast/domain/LinkMan.hbm.xml" />
<mapping resource="cn/itcast/domain/User.hbm.xml" />
```

注意不要再去绑定线程对象了，因为spring已经帮我们做了类似的事情。

4.1.3 测试

```java
@Test
//单独测试hibernate
public void fun1(){
    Configuration conf = new Configuration().configure();

    SessionFactory sf = conf.buildSessionFactory();

    Session session = sf.openSession();

    Transaction tx = session.beginTransaction();
    //-------------------------------------------------
    User u = new User();

    u.setUser_code("rose");
    u.setUser_name("肉丝");
    u.setUser_password("1234");

    session.save(u);

    //-------------------------------------------------
    tx.commit();

    session.close();

    sf.close();

}
```

4.2 hibernate的SessionFactory对象交由spring管理

注意hibernate的版本，这里是第五版

配置方案一：

```xml
<!-- 将SessionFactory配置到spring容器中 -->
<!-- 加载配置方案1:仍然使用外部的hibernate.cfg.xml配置信息 -->
<bean name="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean" >
    <property name="configLocation" value="classpath:hibernate.cfg.xml" ></property>
</bean>
```

配置方案二：

```xml
<!-- 加载配置方案2:在spring配置中放置hibernate配置信息 -->
<bean name="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean"
    <!-- 配置hibernate基本信息 -->
    <property name="hibernateProperties">
        <props>
            <!--  必选配置 -->
            <prop key="hibernate.connection.driver_class" >com.mysql.jdbc.Driver</prop>
            <prop key="hibernate.connection.url" >jdbc:mysql:///crm_32</prop>
            <prop key="hibernate.connection.username" >root</prop>
            <prop key="hibernate.connection.password" >1234</prop>
            <prop key="hibernate.dialect" >org.hibernate.dialect.MySQLDialect</prop>

            <!--  可选配置 -->
            <prop key="hibernate.show_sql" >true</prop>
            <prop key="hibernate.format_sql" >true</prop>
            <prop key="hibernate.hbm2ddl.auto" >update</prop>
        </props>
    </property>
    <!-- 引入orm元数据,指定orm元数据所在的包路径,spring会自动读取包中的所有配置 -->
    <property name="mappingDirectoryLocations" value="classpath:cn/itcast/domain" ></property>
</bean>
```

推荐使用方案二，毕竟少了一个配置文件了。永久告别hibernate.cfg.xml。
此时不需要创建SessionFactory对象，直接进行数据库操作即可
测试用例

```java
@Test
public void fun2(){
    Session session = sf.openSession();

    Transaction tx = session.beginTransaction();
    //----------------------------------------------------
    User u = new User();

    u.setUser_code("haoke");
    u.setUser_name("毫克");
    u.setUser_password("1234");

    session.save(u);

    //----------------------------------------------------
    tx.commit();

    session.close();


}
```

4.3 整合c3p0连接池

```xml
<!-- 读取db.properties文件 -->      //配置读取连接池的配置文件
<context:property-placeholder location="classpath:db.properties" />
<!-- 配置c3p0连接池 -->
<bean name="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" >
    <property name="jdbcUrl" value="${jdbc.jdbcUrl}" ></property>
    <property name="driverClass" value="${jdbc.driverClass}" ></property>
    <property name="user" value="${jdbc.user}" ></property>
    <property name="password" value="${jdbc.password}" ></property>
</bean>
```

//连接池的有很多都会被要求其提供依赖注入 比如我们的
SessionFactory和后面操作数据库的HibernateTemplate模板
当然后者一般继承某个XXXSuppor类之后不需要注入连接池

其实上一次日志已经总结过了
只不过此时hibernate的配置由此可以简化成：

```xml
<!-- 加载配置方案2:在spring配置中放置hibernate配置信息 -->
<bean name="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean" >
    <!-- 将连接池注入到sessionFactory, hibernate会通过连接池获得连接 -->
    <property name="dataSource" ref="dataSource" ></property>
    <!-- 配置hibernate基本信息 -->
    <property name="hibernateProperties">
        <props>
            <!-- 必选配置 -->
            <!-- <prop key="hibernate.connection.driver_class" >com.mysql.jdbc.Driver</prop>
            <prop key="hibernate.connection.url" >jdbc:mysql:///crm_32</prop>
            <prop key="hibernate.connection.username" >root</prop>
            <prop key="hibernate.connection.password" >201805</prop> -->
            <prop key="hibernate.dialect" >org.hibernate.dialect.MySQLDialect</prop>

            <!-- 可选配置 -->
            <prop key="hibernate.show_sql" >true</prop>
            <prop key="hibernate.format_sql" >true</prop>
            <prop key="hibernate.hbm2ddl.auto" >update</prop>
        </props>
    </property>
    <!-- 引入orm元数据,指定orm元数据所在的包路径,spring会自动读取包中的所有配置 -->
    <property name="mappingDirectoryLocations" value="classpath:cn/scct/domain" ></property>
</bean>
```

代替了

4.4 配置Dao层操作sql语句的HibernateTemplate对象

```java
//HibernateDaoSupport 为dao注入sessionFactory
public class UserDaoImpl extends HibernateDaoSupport implements UserDao {
 //要注意用HibernateTemplate操作数据库时要为该类注入sessionFactory注入
                                                    //和JdbcTemplate类似
    @Override
    public User getByUserCode(final String usercode) {
        //HQL
        return getHibernateTemplate().execute(new HibernateCallback<User>() {
            @Override
            public User doInHibernate(Session session) throws HibernateException {
                    String hql = "from User where user_code = ? ";
                    Query query = session.createQuery(hql);
                    query.setParameter(0, usercode);
                    User user = (User) query.uniqueResult();
                return user;
            }
        });
        //Criteria
        /*DetachedCriteria dc = DetachedCriteria.forClass(User.class);

        dc.add(Restrictions.eq("user_code", usercode));

        List<User> list = (List<User>) getHibernateTemplate().findByCriteria(dc);

        if(list != null && list.size()>0){
            return list.get(0);
        }else{
            return null;
        }*/
    }
```

获取HibernateTemplate对象

类似的API我真的没有认真去了解过，包括JdbcTemplate的API

```xml
<!-- dao -->
<bean name="userDao" class="cn.scct.dao.impl.UserDaoImpl" >
    <!-- 注入sessionFactory -->
    <property name="sessionFactory" ref="sessionFactory" ></property>
</bean>
```

五、整合事务

```xml
<!-- 核心事务管理器 -->
<bean name="transactionManager" class="org.springframework.orm.hibernate5.HibernateTransactionManager" >
    <property name="sessionFactory" ref="sessionFactory" ></property>
</bean>
                                                    //事务需要连接池技术，所以也要配置sessionFactory注入

<!-- 配置通知 -->
<!-- <tx:advice id="txAdvice" transaction-manager="transactionManager" >
    <tx:attributes>
        <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="update*" isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="false" />
        <tx:method name="modify*" 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="get*" isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="true" />
        <tx:method name="find*" isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="true" />
    </tx:attributes>
</tx:advice> -->
<!-- 配置将通知织入目标对象
配置切点
配置切面 -->
<!-- <aop:config>
    <aop:pointcut expression="execution(* cn.itcast.service.impl.*ServiceImpl.*(..))" id="txPc"/>
    <aop:advisor advice-ref="txAdvice" pointcut-ref="txPc" />
</aop:config> -->
<!-- ================================================================================================ -->
<!-- 开启注解事务 -->
<tx:annotation-driven transaction-manager="transactionManager" />
```

上一篇日志讲过了　注解开启事务最舒服

注意：
事务处理在service层中处理一般，所以一般加在service层
采用注解在service类中配置事务可以参考上一篇日志。

六、 扩大session的作用范围(在web.xml中配置)

```xml
<!-- 扩大session作用范围
  注意： 任何filter一定要在struts的filter之前调用
 -->
 <filter>
  <filter-name>openSessionInView</filter-name>
  <filter-class>org.springframework.orm.hibernate5.support.OpenSessionInViewFilter</filter-class>
</filter>
 <filter-mapping>
   <filter-name>openSessionInView</filter-name>
   <url-pattern>/*</url-pattern>
 </filter-mapping>
```

> 为了避免使用懒加载时出现no-session问题.需要扩大session的作用范围