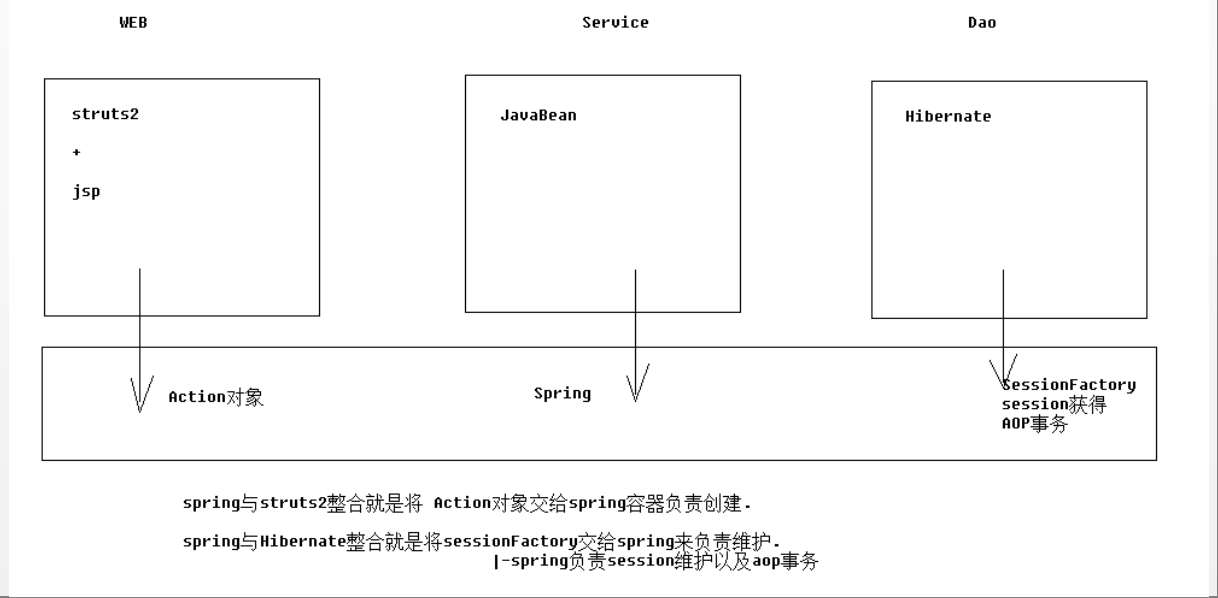
# 整合原理：



版本：

Hibernate-5.0.1

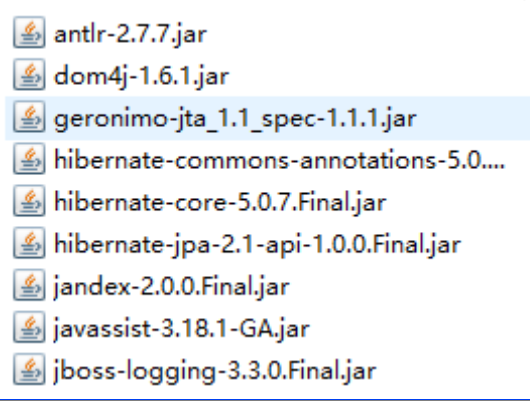
struts2-2.3.24

spring-4.2.4

# 导包：

**hibernate:**

1. hibernate/lib/required



1. hibernate/lib/jpa | java persist api java的持久化规范(接口)



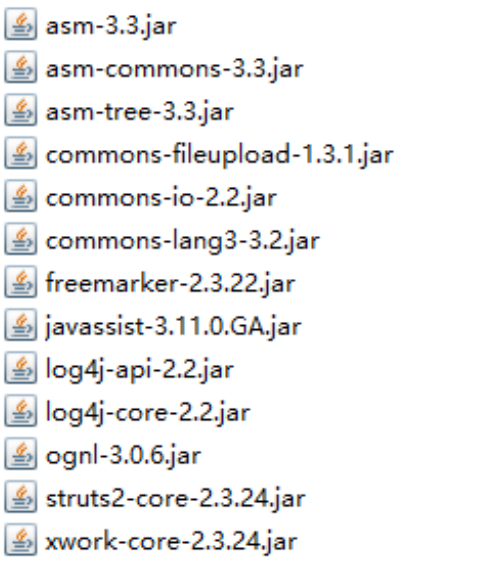
1. 数据库驱动包



**struts2：**

1. struts-blank.war/WEB-INF/lib/\*

注意:javassist-3.18.1-GA.jar包与hibernate中的重复，保留版本高的



1. struts整合spring插件包(struts2/lib/)

注意:这个包一旦导入,那么struts2在启动时就会寻找spring容器.找不到将会抛出异常



**spring:**

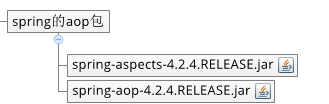
1. 基本4+2

core|beans|context|expression|logging|log4j

1. 整合web包

spring-web

1. 整合AOP





1. 整合Hibernate和事务

spring-jdbc|spring-tx|c3p0|spring-orm









1. Junit4测试包



1. 标签库包



开始配置

# 单独配置spring容器到WEB

1. 创建配置文件,并导入约束(4个)

beans|context|aop|tx

1. 配置spring随项目启动

在web.xml中配置以下内容

<!-- 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>

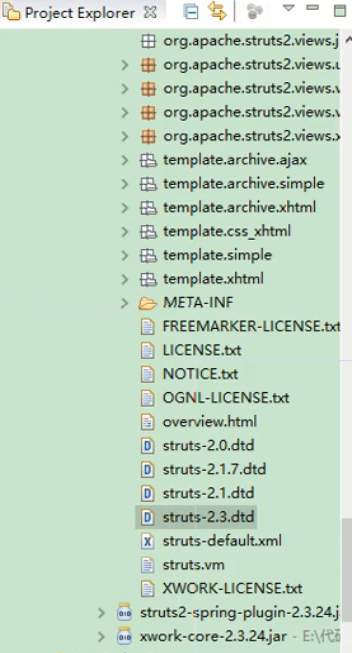
</context-param>

# 单独配置struts2到WEB

1. 配置struts2主配置文件

找到导入包sturts2-core-2.4.24.jar

找到DTD



复制下面这段到struts.xml

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

2种配置方案选其一

<struts>

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

<action name="UserAction\_\*" class="userAction" method="{1}" >

<result name="success" >/success.jsp</result>

</action>

</package>

</struts>

1. 配置struts2核心过滤器到web.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>

1. struts2与spring整合

整合方案1：不推荐使用（还是有struts2生成action，spring无法管理整个action的生命周期）

配置struct.xml，class依然使用完整类名

<struts>

<!-- # 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>

<!-- 整合方案1:class属性上仍然配置action的完整类名

struts2仍然创建action,由spring负责组装Action中的依赖属性

-->

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

<action name="UserAction\_\*" class="cn.itcast.web.userAction" method="{1}" >

<result name="success" >/success.jsp</result>

</action>

</package>

</struts>

配置applicationContext.xml

<!-- action -->

<!-- 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.itcast.web.action.UserAction" scope="prototype" >

<property name="userService" ref="userService" ></property>

</bean>

<!-- service -->

<bean name="userService" class="cn.itcast.service.impl.UserServiceImpl" >

<property name="ud" ref="userDao" ></property>

</bean>

整合方案2：推荐方案

配置struct.xml

<!--

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

-->

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

<action name="UserAction\_\*" class="userAction" method="{1}" >

<result name="success" >/success.jsp</result>

</action>

</package>

配置applicationContext.xml

<!-- action -->

<!-- 注意:Action对象作用范围一定是多例的.这样才符合struts2架构 -->

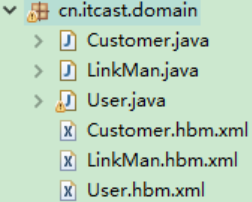
<bean name="userAction" class="cn.itcast.web.action.UserAction" scope="prototype" >

<property name="userService" ref="userService" ></property>

</bean>

**单独配置hibernate**

导入实体类&orm元数据



以User.java和User.hbm.xml为例

User.java

package cn.itcast.domain;

import java.util.HashSet;

import java.util.Set;

public class User {

/\*

\* CREATE TABLE `sys\_user` (

`user\_id` bigint(32) NOT NULL AUTO\_INCREMENT COMMENT '用户id',

`user\_code` varchar(32) NOT NULL COMMENT '用户账号',

`user\_name` varchar(64) NOT NULL COMMENT '用户名称',

`user\_password` varchar(32) NOT NULL COMMENT '用户密码',

`user\_state` char(1) NOT NULL COMMENT '1:正常,0:暂停',

PRIMARY KEY (`user\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=9 DEFAULT CHARSET=utf8;

\*/

private Long user\_id;

private String user\_code;

private String user\_name;

private String user\_password;

private Character user\_state;

public Long getUser\_id() {

return user\_id;

}

public void setUser\_id(Long user\_id) {

this.user\_id = user\_id;

}

public String getUser\_code() {

return user\_code;

}

public void setUser\_code(String user\_code) {

this.user\_code = user\_code;

}

public String getUser\_name() {

return user\_name;

}

public void setUser\_name(String user\_name) {

this.user\_name = user\_name;

}

public String getUser\_password() {

return user\_password;

}

public void setUser\_password(String user\_password) {

this.user\_password = user\_password;

}

public Character getUser\_state() {

return user\_state;

}

public void setUser\_state(Character user\_state) {

this.user\_state = user\_state;

}

@Override

public String toString() {

return "User [user\_id=" + user\_id + ", user\_code=" + user\_code + ", user\_name=" + user\_name + ", user\_password="

+ user\_password + "]";

}

}

User.hbm.xml

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="cn.itcast.domain" >

<class name="User" table="sys\_user" >

<id name="user\_id" >

<generator class="native"></generator>

</id>

<property name="user\_code" ></property>

<property name="user\_name" ></property>

<property name="user\_password" ></property>

<property name="user\_state" ></property>

</class>

</hibernate-mapping>

配置主配置文件hibernate.cfg.xml

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<!-- 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>

<!-- 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>

<!-- 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" />

</session-factory>

</hibernate-configuration>

# spring整合hibernate

**整合原理：将sessionFactory对象交给spring容器管理**

整合方案1：（不推荐方案）

配置applicationContext.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>

整合方案2：推荐方案

配置applicationContext.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>

# spring整合c3p0连接池

1. 配置db.properties

jdbc.jdbcUrl=jdbc:mysql:///crm\_32

jdbc.driverClass=com.mysql.jdbc.Driver

jdbc.user=root

jdbc.password=1234

1. 引入连接池到spring中

配置applicationContext.xml

<!-- 读取db.properties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>

1. 将连接池注入给SessionFactory

配置applicationContext.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" >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>

# spring整合hibernate环境操作数据库

1. Dao类创建:继承HibernateDaoSupport

UserDaoImpl.java

package cn.itcast.dao.impl;

import java.util.List;

import org.hibernate.HibernateException;

import org.hibernate.Query;

import org.hibernate.Session;

import org.hibernate.criterion.DetachedCriteria;

import org.hibernate.criterion.Restrictions;

import org.springframework.orm.hibernate5.HibernateCallback;

import org.springframework.orm.hibernate5.HibernateTemplate;

import org.springframework.orm.hibernate5.support.HibernateDaoSupport;

import cn.itcast.dao.UserDao;

import cn.itcast.domain.User;

//HibernateDaoSupport 为dao注入sessionFactory

public class UserDaoImpl extends HibernateDaoSupport implements UserDao {

@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;

}\*/

}

@Override

public void save(User u) {

getHibernateTemplate().save(u);

}

}

1. hibernate模板的操作

execute - 见第一步HQL函数

findByCriteria - 见第一步Criteria函数

1. spring中配置dao

配置applicationContext.xml

<!-- dao -->

<bean name="userDao" class="cn.itcast.dao.impl.UserDaoImpl" >

<!-- 注入sessionFactory -->

<property name="sessionFactory" ref="sessionFactory" ></property>

</bean>

# spring的aop事务

准备工作

配置applicationContext.xml

<!-- 核心事务管理器 -->

<bean name="transactionManager" class="org.springframework.orm.hibernate5.HibernateTransactionManager" >

<property name="sessionFactory" ref="sessionFactory" ></property>

</bean>

xml配置aop事务

配置通知

配置applicationContext.xml

<!-- 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>

配置织入

配置applicationContext.xml

<!-- 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>

注解配置aop事务

开启注解事务

配置applicationContext.xml

<!-- 开启注解事务 -->

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

Service类中使用注解

UserServiceImpl.java

package cn.itcast.service.impl;

import org.springframework.transaction.annotation.Isolation;

import org.springframework.transaction.annotation.Propagation;

import org.springframework.transaction.annotation.Transactional;

import cn.itcast.dao.UserDao;

import cn.itcast.domain.User;

import cn.itcast.service.UserService;

@Transactional(isolation=Isolation.REPEATABLE\_READ,propagation=Propagation.REQUIRED,readOnly=true)

public class UserServiceImpl implements UserService{

private UserDao ud;

@Override

public User getUserByCodePassword(User u) {

//1 根据登陆名称查询登陆用户

User existU = ud.getByUserCode(u.getUser\_code());

//2 判断用户是否存在.不存在=>抛出异常,提示用户名不存在

if(existU==null){

throw new RuntimeException("用户名不存在!");

}

//3 判断用户密码是否正确=>不正确=>抛出异常,提示密码错误

if(!existU.getUser\_password().equals(u.getUser\_password())){

throw new RuntimeException("密码错误!");

}

//4 返回查询到的用户对象

return existU;

}

@Override

@Transactional(isolation=Isolation.REPEATABLE\_READ,propagation=Propagation.REQUIRED,readOnly=false)

public void saveUser(User u) {

ud.save(u);

}

public void setUd(UserDao ud) {

this.ud = ud;

}

}

扩大session作用范围

hibernate懒加载问题，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

配置filter

配置web.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>

<!-- 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>openSessionInView</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>