

Spring_day04 总结

今日任务

- 使用 SSH 整合完成客户的保存操作

教学导航

教学目标	
教学方法	案例驱动法

案例一使用 SSH 的整合完成客户的保存操作

1.1 案例需求

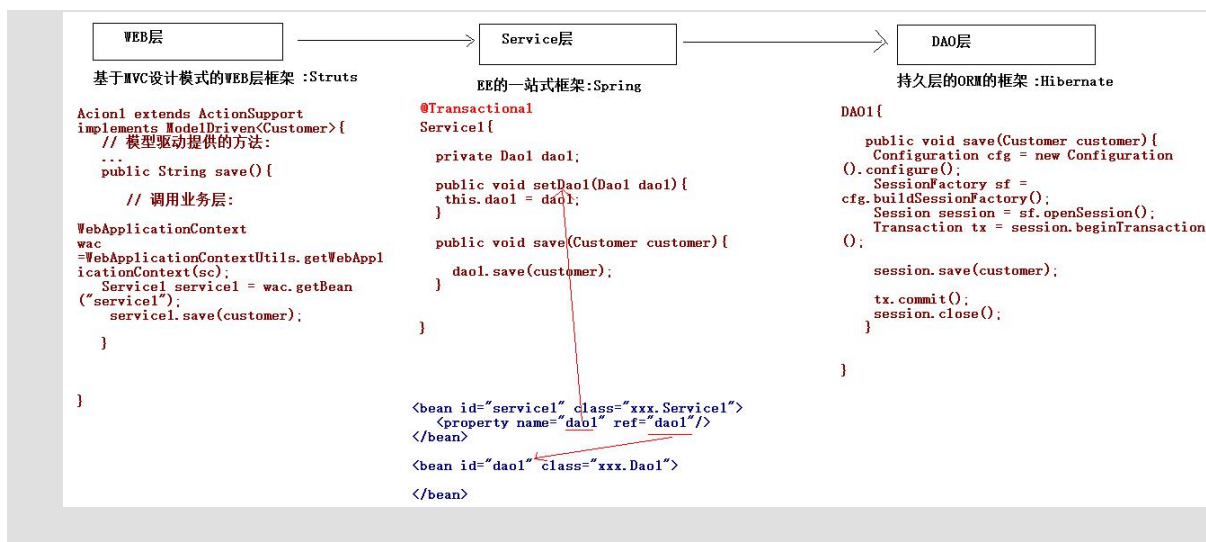
1.1.1 需求描述

使用 SSH 整合完成 CRM 的客户保存操作

1.2 相关知识点:

1.2.1 SSH 简单的回顾:

1.2.1.1 SSH 的基本开发回顾



1.2.2 SSH 框架的整合方式一: 零障碍整合(带有 Hibernate 配置文件)

1.2.2.1 创建 web 项目, 引入相关 jar 包.

【Struts2】

D:\struts2\struts-2.3.24\apps\struts2-blank\WEB-INF\lib*.jar










asm-3.3.jar	2013/11/23 17:55	Executable Jar File	43 KB
asm-commons-3.3.jar	2013/11/23 17:55	Executable Jar File	38 KB
asm-tree-3.3.jar	2013/11/23 17:55	Executable Jar File	21 KB
commons-fileupload-1.3.1.jar	2014/2/19 16:21	Executable Jar File	68 KB
commons-io-2.2.jar	2013/11/23 17:55	Executable Jar File	170 KB
commons-lang3-3.2.jar	2014/1/2 21:45	Executable Jar File	376 KB
freemarker-2.3.22.jar	2015/4/3 7:09	Executable Jar File	1,271 KB
javassist-3.11.0.GA.jar	2013/11/23 17:55	Executable Jar File	600 KB
log4j-api-2.2.jar	2015/4/19 12:04	Executable Jar File	131 KB
log4j-core-2.2.jar	2015/4/19 12:04	Executable Jar File	808 KB
ognl-3.0.6.jar	2013/11/23 17:55	Executable Jar File	223 KB
struts2-core-2.3.24.jar	2015/5/3 12:25	Executable Jar File	813 KB
xwork-core-2.3.24.jar	2015/5/3 12:23	Executable Jar File	661 KB

Struts2 需要了解的 jar 包:



struts2-convention-plugin-2.3.24.jar	---Struts2 注解的开发包.
struts2-json-plugin-2.3.24.jar	---Struts2 整合 AJAX 返回 JSON 数据.
struts2-spring-plugin-2.3.24.jar	---Struts2 整合 Spring 的插件包.

【Hibernate】

D:\hibernate-release-5.0.7.Final\lib\required*.jar

 antlr-2.7.7.jar	2014/4/28 20:30	Executable Jar File	435 KB
 dom4j-1.6.1.jar	2014/4/28 20:28	Executable Jar File	307 KB
 geronimo-jta_1.1_spec-1.1.1.jar	2015/5/5 11:26	Executable Jar File	16 KB
 hibernate-commons-annotations-5.0....	2015/11/30 10:22	Executable Jar File	74 KB
 hibernate-core-5.0.7.Final.jar	2016/1/13 12:35	Executable Jar File	5,453 KB
 hibernate-jpa-2.1-api-1.0.0.Final.jar	2014/4/28 20:30	Executable Jar File	111 KB
 jandex-2.0.0.Final.jar	2015/11/30 10:22	Executable Jar File	184 KB
 javassist-3.18.1-GA.jar	2014/4/28 20:28	Executable Jar File	698 KB
 jboss-logging-3.3.0.Final.jar	2015/5/28 12:35	Executable Jar File	66 KB

日志记录:

 slf4j-api-1.6.1.jar	2015/8/6 14:05	Executable Jar File	25 KB
 slf4j-log4j12-1.7.2.jar	2015/8/6 14:05	Executable Jar File	9 KB




log4j 的包由 Spring 引入.

数据库驱动:

 mysql-connector-java-5.1.7-bin.jar	2014/7/8 18:41	Executable Jar File	694 KB
--	----------------	---------------------	--------




Hibernate 引入连接池:

D:\hibernate-release-5.0.7.Final\lib\optional\c3p0*.jar




 c3p0-0.9.2.1.jar	2014/4/28 20:30	Executable Jar File	414 KB
 hibernate-c3p0-5.0.7.Final.jar	2016/1/13 12:42	Executable Jar File	12 KB
 mchange-commons-java-0.2.3.4.jar	2014/4/28 20:30	Executable Jar File	568 KB







【Spring】

基本的开发:

 com.springsource.org.apache.commons.logging-1.1.1.jar
 com.springsource.org.apache.log4j-1.2.15.jar
 spring-beans-4.2.4.RELEASE.jar
 spring-context-4.2.4.RELEASE.jar
 spring-core-4.2.4.RELEASE.jar
 spring-expression-4.2.4.RELEASE.jar

AOP 开发:

 spring-aop-4.2.4.RELEASE.jar	2015/12/17 0:44	Executable Jar File	362 KB
 spring-aspects-4.2.4.RELEASE.jar	2015/12/17 0:48	Executable Jar File	58 KB
 com.springsource.org.aopalliance-1.0.0.jar	2010/4/2 11:09	Executable Jar File	5 KB

 com.springsource.org.aspectj.weaver-1.6.8.RELEASE.jar	2010/4/2 11:09	Executable Jar File	1,604 KB
JDBC 开发:			
 spring-tx-4.2.4.RELEASE.jar	2015/12/17 0:45	Executable Jar File	260 KB
 spring-jdbc-4.2.4.RELEASE.jar	2015/12/17 0:45	Executable Jar File	414 KB
事务管理的开发:			
 spring-tx-4.2.4.RELEASE.jar	2015/12/17 0:45	Executable Jar File	260 KB
整合 Hibernate:			
 spring-orm-4.2.4.RELEASE.jar	2015/12/17 0:46	Executable Jar File	456 KB
整合 web 项目:			
 spring-web-4.2.4.RELEASE.jar	2015/12/17 0:46	Executable Jar File	750 KB

1.2.2.2 引入相关的配置文件:

【Struts2】

```
web.xml
<!-- 配置 Struts2 的核心过滤器 -->
<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>
struts.xml
```

【Hibernate】

核心配置: hibernate.cfg.xml

映射文件:

【Spring】

```
web.xml
<!-- 配置 Spring 的核心监听器 -->
<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-
```

```
class>
    </listener>

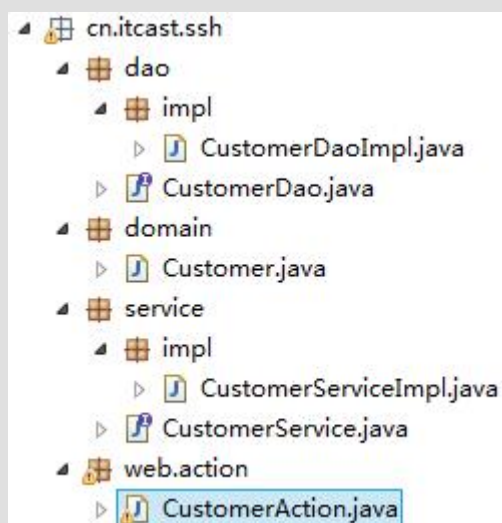
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:applicationContext.xml</param-value>
    </context-param>
applicationContext.xml

log4j.properties
```

1.2.2.3 引入相关的页面并进行修改:

```
<FORM id=form1 name=form1
    action="${pageContext.request.contextPath }/customer_save.action|"
```

1.2.2.4 创建包结构和相关的类:



```
cn.itcast.ssh
├── dao
│   ├── impl
│   │   ├── CustomerDaoImpl.java
│   │   └── CustomerDao.java
│   └── Customer.java
├── domain
│   └── Customer.java
├── service
│   ├── impl
│   │   ├── CustomerServiceImpl.java
│   │   └── CustomerService.java
│   └── CustomerService.java
└── web.action
    └── CustomerAction.java
```

1.2.2.5 Struts2 和 Spring 的整合:方式一: Action 类由 Struts2 自己创建

【编写 Action 中的 save 方法】

```
/**
 * 保存客户的执行的方法:save
 */
public String save() {
    System.out.println("Action 中的 save 方法执行了...");
    return NONE;
}
```

【配置 Action 类】

```
<package name="ssh" extends="struts-default" namespace="/">
    <action name="customer_*" class="cn.itcast.ssh.web.action.CustomerAction"
method="{1}">

        </action>
    </package>
```

【在 Action 中调用业务层的类】

```
配置 Service:
    <!-- 配置 Service -->
    <bean                                id="customerService"
class="cn.itcast.ssh.service.impl.CustomerServiceImpl">

        </bean>
```

在 Action 中调用

// 传统方式的写法

```
WebApplicationContext webApplicationContext = WebApplicationContextUtils
.getWebApplicationContext(ServletActionContext.getServletContext());
CustomerService customerService = (CustomerService)
webApplicationContext.getBean("customerService");
```

***** 这种写法很麻烦的，因为需要在每个 Action 中的每个方法上获取工厂，通过工厂获得类。

为了简化这个代码引入一个插件的包：

struts2-spring-plugin-2.3.24.jar

在这个插件中开启一个 Struts2 常量

```
* <constant name="struts.objectFactory" value="spring" />
* 默认的情况下 struts2 将这个常量关闭的，现在引入插件以后，将常量开启了，引发下面的一些
常量生效。
```

```
struts.objectFactory.spring.autoWire = name
```

那么就可以在 Action 中提供想注入的属性了：

```
public class CustomerAction extends ActionSupport implements ModelDriven<Customer>
{
    // 模型驱动使用的对象
    private Customer customer = new Customer();

    @Override
    public Customer getModel() {
        return customer;
    }

    // 注入业务层的类：
```

```
private CustomerService customerService;

public void setCustomerService(CustomerService customerService) {
    this.customerService = customerService;
}

/**
 * 保存客户的执行的方法:save
 */
public String save() {
    System.out.println("Action 中的 save 方法执行了...");
    // 传统方式的写法
    /*WebApplicationContext webApplicationContext = WebApplicationContextUtils
        .getWebApplicationContext(ServletActionContext.getServletContext());
    CustomerService customerService = (CustomerService)
        webApplicationContext.getBean("customerService");*/

    // 自动注入
    customerService.save(customer);
    return NONE;
}
}
```

【在 Service 中编写 save 方法】

```
public class CustomerServiceImpl implements CustomerService {

    @Override
    public void save(Customer customer) {
        System.out.println("Service 中的 save 方法执行了...");
    }

}
```

1.2.2.6 Struts2 和 Spring 的整合方式二:Action 类由 Spring 创建。(推荐)

【引入插件包】

```
struts2-spring-plugin-2.3.24.jar
```

【Action 交给 Spring 管理】

将 Action 配置到 Spring 中.

```
<!-- 配置 Action -->
```

```
<bean id="customerAction" class="cn.itcast.ssh.web.action.CustomerAction"
```



```
scope="prototype">
    <!--必须手动注入属性-->
    <property name="customerService" ref="customerService"/>
</bean>

Action 的配置:
<package name="ssh" extends="struts-default" namespace="/">
    <action name="customer_*" class="customerAction" method="{1}">

    </action>
</package>
```

1.2.2.7 在业务层调用 DAO

【将 DAO 配置到 Spring 中】

```
<!-- 配置 DAO -->
<bean id="customerDao" class="cn.itcast.ssh.dao.impl.CustomerDaoImpl">

</bean>
```

【在业务层注入 Dao】

```
public class CustomerServiceImpl implements CustomerService {

    // 注入 Dao
    private CustomerDao customerDao;

    public void setCustomerDao(CustomerDao customerDao) {
        this.customerDao = customerDao;
    }

    @Override
    public void save(Customer customer) {
        System.out.println("Service 中的 save 方法执行了...");
        customerDao.save(customer);
    }

}

<!-- 配置 Service -->
<bean
class="cn.itcast.ssh.service.impl.CustomerServiceImpl"
    <property name="customerDao" ref="customerDao"/>
</bean>
```


1.2.2.8 Spring 整合 Hibernate:

【创建映射文件】

```
<hibernate-mapping>

    <class name="cn.itcast.ssh.domain.Customer" table="cst_customer">

        <id name="cust_id">
            <generator class="native"/>
        </id>

        <property name="cust_name"/>
        <property name="cust_user_id"/>
        <property name="cust_create_id"/>
        <property name="cust_source"/>
        <property name="cust_industry"/>
        <property name="cust_level"/>
        <property name="cust_linkman"/>
        <property name="cust_phone"/>
        <property name="cust_mobile"/>

    </class>
</hibernate-mapping>
```

【加载到核心配置文件】

```
<!-- 加载映射文件 -->
<mapping resource="cn/itcast/ssh/domain/Customer.hbm.xml"/>
```

【在 Spring 的配置文件中完成如下配置】

```
<!-- 配置 Hibernate 中的 sessionFactory -->
<bean id="sessionFactory"
class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
    <property name="configLocations" value="classpath:hibernate.cfg.xml"/>
</bean>
```

【改写 DAO】

```
public class CustomerDaoImpl extends HibernateDaoSupport implements CustomerDao {

    @Override
    public void save(Customer customer) {
        System.out.println("DAO 中的 save 方法执行了...");
    }

}

<!-- 配置 DAO -->
```

```
<bean id="customerDao" class="cn.itcast.ssh.dao.impl.CustomerDaoImpl">
  <property name="sessionFactory" ref="sessionFactory"/>
</bean>
```

【调用模板中的方法】

```
@Override
public void save(Customer customer) {
    System.out.println("DAO 中的 save 方法执行了...");
    // 保存:
    this.getHibernateTemplate().save(customer);
}
```

1.2.2.9 配置 Spring 的事务管理：

【配置事务管理器】

```
<!-- 配置事务管理器 -->
<bean id="transactionManager"
class="org.springframework.orm.hibernate5.HibernateTransactionManager">
  <property name="sessionFactory" ref="sessionFactory"/>
</bean>
```

【注解事务管理的开启】

```
<!-- 开启事务管理的注解 -->
<tx:annotation-driven transaction-manager="transactionManager"/>
```

【在业务层添加一个注解】

```
@Transactional
public class CustomerServiceImpl implements CustomerService {
```

1.2.3 SSH 框架的整合方式二：不带 Hibernate 的配置文件

1.2.3.1 复制一个 SSH1 的项目。

1.2.3.2 查看 Hibernate 的配置文件：

Hibernate 的配置文件包含如下内容：

- 连接数据库必要的参数：
- Hibernate 的属性：
- 连接池的配置：
- 映射文件的引入：

1.2.3.3 替换数据库连接参数和连接池的配置:

创建 jdbc.properties

```
jdbc.driverClass=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql:///ssh1
jdbc.username=root
jdbc.password=123
```

在 Spring 中引入外部属性文件

```
<!-- 引入外部属性文件 -->
```

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

配置连接池:

```
<!-- 配置 c3p0 连接池: -->
```

```
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
  <property name="driverClass" value="{jdbc.driverClass}"/>
  <property name="jdbcUrl" value="{jdbc.url}"/>
  <property name="user" value="{jdbc.username}"/>
  <property name="password" value="{jdbc.password}"/>
</bean>
```

1.2.3.4 配置 Hibernate 的其他属性及映射:

```
<!-- 配置 Hibernate 中的 sessionFactory -->
```

```
<bean id="sessionFactory"
```

```
class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
```

```
<!-- 注入连接池 -->
```

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

```
<!-- 配置 Hibernate 的相关属性 -->
```

```
<property name="hibernateProperties">
```

```
<props>
```

```
<!-- 配置 Hibernate 的方言 -->
```

```
<prop
```

```
key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
```

```
<!-- 显示 SQL -->
```

```
<prop key="hibernate.show_sql">true</prop>
```

```
<!-- 格式化 SQL -->
```

```
<prop key="hibernate.format_sql">true</prop>
```

```
<!-- 映射到 DDL 的自动创建 -->
```

```
<prop key="hibernate.hbm2ddl.auto">update</prop>
```

```
</props>
```

```
</property>
```

```
<!-- 配置引入映射文件 -->
```

```
<property name="mappingResources">
    <list>
        <value>cn/itcast/ssh/domain/Customer.hbm.xml</value>
    </list>
</property>
</bean>
```

1.2.4 HibernateTemplate 的使用:

```
public class CustomerDaoImpl extends HibernateDaoSupport implements CustomerDao {

    @Override
    public void save(Customer customer) {
        System.out.println("DAO 中的 save 方法执行了...");
        // 保存:
        this.getHibernateTemplate().save(customer);
    }

    @Override
    public void update(Customer customer) {
        this.getHibernateTemplate().update(customer);
    }

    @Override
    public void delete(Customer customer) {
        this.getHibernateTemplate().delete(customer);
    }

    @Override
    public Customer findById(Long id) {
        return this.getHibernateTemplate().load(Customer.class, id);
    }

    @Override
    public List<Customer> findAllByHQL() {
        List<Customer> list = (List<Customer>)
        this.getHibernateTemplate().find("from Customer");
        return list;
    }

    public List<Customer> findAllByQBC() {
        DetachedCriteria detachedCriteria =
        DetachedCriteria.forClass(Customer.class);
        List<Customer> list = (List<Customer>)
```

```
this.getHibernateTemplate().findByCriteria(detachedCriteria);  
    return list;  
}  
  
}
```

1.2.5 延迟加载的问题的解决：OpenSessionInViewFilter

```
<filter>  
    <filter-name>OpenSessionInViewFilter</filter-name>  
  
    <filter-class>org.springframework.orm.hibernate5.support.OpenSessionInViewFilter</filter-class>  
</filter>  
  
<filter-mapping>  
    <filter-name>OpenSessionInViewFilter</filter-name>  
    <url-pattern>*.action</url-pattern>  
</filter-mapping>
```