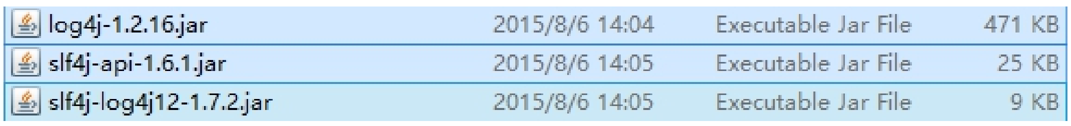
1.2 SSH整合方式一：无障碍整合

1.2.2 SSH整合

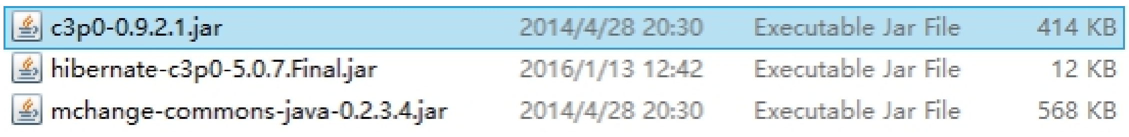
1.2.2.1 第一步：创建web项目，引入jar包

- Struts2的jar包
    - struts-2.3.24\apps\struts2-blank\WEB-INF\lib*.jar
    - Struts2中有一些包需要了解的：
        - struts2-convention-plugin-2.3.24.jar ----Struts2的注解开发包。
        - struts2-json-plugin-2.3.24.jar ----Struts2的整合AJAX的开发包。
        - struts2-spring-plugin-2.3.24.jar ----Struts2的整合Spring的开发包。
- Hibernate的jar包
    - Hibernate的开发的必须的包
        - hibernate-release-5.0.7.Final\lib\required*.jar
    - MySQL驱动
    - 日志记录



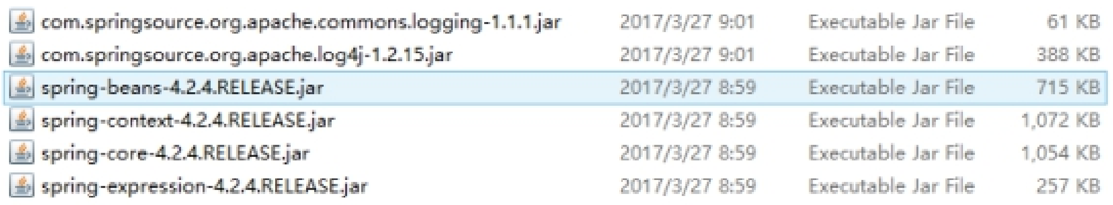
log4j-1.2.16.jar	2015/8/6 14:04	Executable Jar File	471 KB
slf4j-api-1.6.1.jar	2015/8/6 14:05	Executable Jar File	25 KB
slf4j-log4j12-1.7.2.jar	2015/8/6 14:05	Executable Jar File	9 KB

- 使用C3P0连接池：



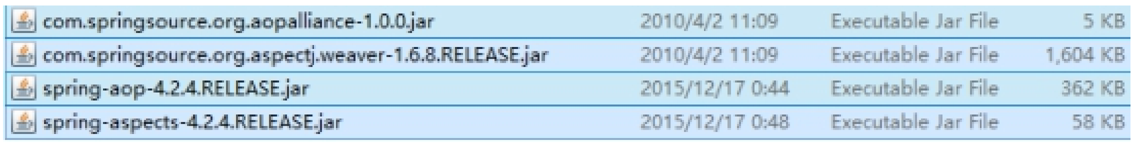
c3p0-0.9.2.1.jar	2014/4/28 20:30	Executable Jar File	414 KB
hibernate-c3p0-5.0.7.Final.jar	2016/1/13 12:42	Executable Jar File	12 KB
mchange-commons-java-0.2.3.4.jar	2014/4/28 20:30	Executable Jar File	568 KB

- **注意：Struts2和Hibernate都引入了一个相同的jar包（javassist包）。删除一个**
- Spring的jar包
    - IOC的开发



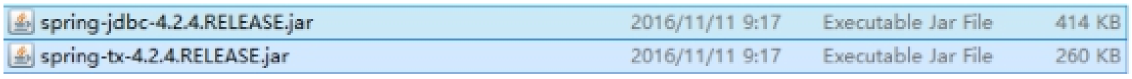
com.springsource.org.apache.commons.logging-1.1.1.jar	2017/3/27 9:01	Executable Jar File	61 KB
com.springsource.org.apache.log4j-1.2.15.jar	2017/3/27 9:01	Executable Jar File	388 KB
spring-beans-4.2.4.RELEASE.jar	2017/3/27 8:59	Executable Jar File	715 KB
spring-context-4.2.4.RELEASE.jar	2017/3/27 8:59	Executable Jar File	1,072 KB
spring-core-4.2.4.RELEASE.jar	2017/3/27 8:59	Executable Jar File	1,054 KB
spring-expression-4.2.4.RELEASE.jar	2017/3/27 8:59	Executable Jar File	257 KB

- AOP的开发



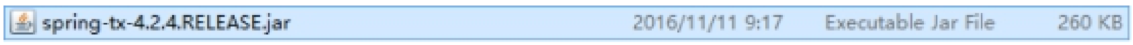
com.springsource.org.aopalliance-1.0.0.jar	2010/4/2 11:09	Executable Jar File	5 KB
com.springsource.org.aspectj.weaver-1.6.8.RELEASE.jar	2010/4/2 11:09	Executable Jar File	1,604 KB
spring-aop-4.2.4.RELEASE.jar	2015/12/17 0:44	Executable Jar File	362 KB
spring-aspects-4.2.4.RELEASE.jar	2015/12/17 0:48	Executable Jar File	58 KB

- JDBC模板的开发



spring-jdbc-4.2.4.RELEASE.jar	2016/11/11 9:17	Executable Jar File	414 KB
spring-tx-4.2.4.RELEASE.jar	2016/11/11 9:17	Executable Jar File	260 KB

- 事务管理



spring-tx-4.2.4.RELEASE.jar	2016/11/11 9:17	Executable Jar File	260 KB

- 整合web项目的开发

spring-web-4.2.4.RELEASE.jar 2015/12/17 0:46 Executable Jar File 750 KB

- 整合单元测试的开发

spring-test-4.2.4.RELEASE.jar 2015/12/17 0:48 Executable Jar File 550 KB

- 整合hibernate的开发

spring-orm-4.2.4.RELEASE.jar 2015/12/17 0:46 Executable Jar File 456 KB

1.2.2.2 第二步：引入配置文件

- Struts的配置文件
  - web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID"
version="2.5">
  <display-name>ssh1</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>

  <!-- 配置 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>
</web-app>
```

  - struts.xml

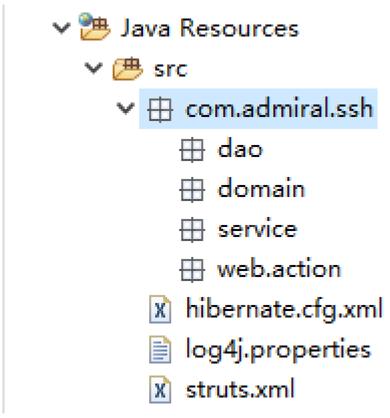
- Hibernate的配置文件
  - hibernate.cfg.xml
    - 删除那个与线程绑定的session。
  - 映射文件

- Spring的配置文件
  - web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID"
version="2.5">
  <display-name>ssh1</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>

  <!-- 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>

</web-app>
```

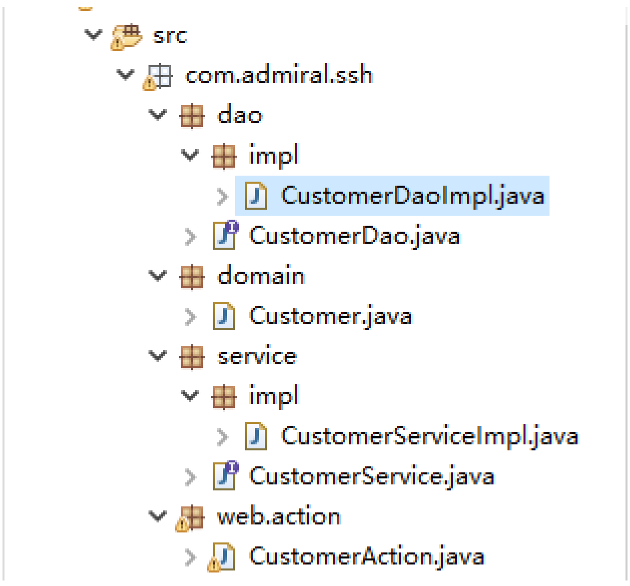
  - applicationContext.xml
- 日志记录
  - log4j.properties

1.2.2.3 第三步：创建包结构



```
Java Resources
  src
    com.admiral.ssh
      dao
      domain
      service
      web.action
    hibernate.cfg.xml
    log4j.properties
    struts.xml
```

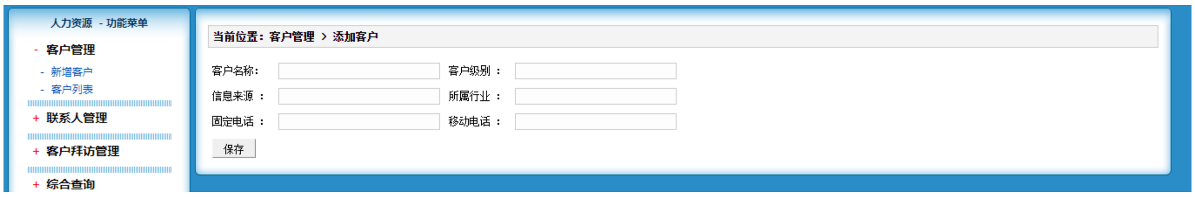
1.2.2.4 第四步：创建相关类



```
src
  com.admiral.ssh
    dao
      impl
        CustomerDaoImpl.java
      CustomerDao.java
    domain
      Customer.java
    service
      impl
        CustomerServiceImpl.java
      CustomerService.java
    web.action
      CustomerAction.java
```

1.2.2.5 第五步：引入相关的页面

1.2.2.6 第六步：修改add.jsp



当前位置：客户管理 > 添加客户

客户名称：		客户级别 ：	
信息来源 ：		所属行业 ：	
固定电话 ：		移动电话 ：	

1.2.2.7 第七步：Spring整合Struts2方式一：Action由Struts2自身创建的。

- 编写 Action

```java
/**
* @Title: CustomerAction.java
* @Package com.admiral.ssh.web.action
* @Description:
* @author Admiral
* @date 2020-10-13
* @version V1.0
*/
package com.admiral.ssh.web.action;

import com.admiral.ssh.domain.Customer;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ModelDriven;

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;
    }

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

}
```

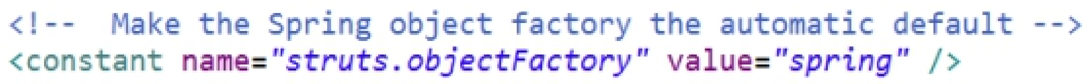
- 配置 Action

```
<?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="ssh1" extends="struts-default" namespace="/">
        <action name="customer_*"
class="com.admiral.ssh.web.action.CustomerAction" method="{1}">

        </action>
    </package>
</struts>
```

- 在Action中引入Service
    - 传统方式

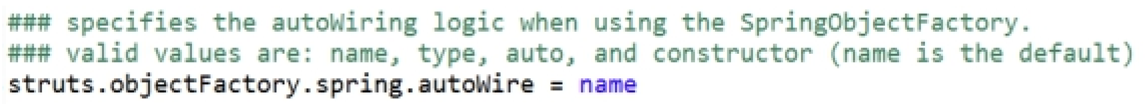
```
    /**
     * 保存客户的方法:save
     */
    public String save() {
        System.out.println("CustomerAction 中的 save 方法执行了....");
        //如果web层没有使用 Struts2 ,获取业务层的类就要进行如下编写
        WebApplicationContext webApplicationContext =
WebApplicationContextUtils.getWebApplicationContext(ServletActionContext.getServ
letContext());
        CustomerService customerService = (CustomerService)
webApplicationContext.getBean("customerService");


        return NONE;
    }
```

- 进行Spring和Struts2的整合：
    - 引入struts-spring-plugin.jar
- 在插件包中有如下配置



```
<!--  Make the Spring object factory the automatic default -->
<constant name="struts.objectFactory" value="spring" />
```

- **开启一个常量：在Struts2中只要开启这个常量就会引发下面常量生效：**



```
### specifies the autoWiring logic when using the SpringObjectFactory.
### valid values are: name, type, auto, and constructor (name is the default)
struts.objectFactory.spring.autoWire = name
```

- 将Service交给Spring管理

```
<?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"
    xmlns:tx="http://www.springframework.org/schema/tx"
    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.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx.xsd">

    <!-- 配置 CustomerService -->
    <bean id="customerService"
class="com.admiral.ssh.service.impl.CustomerServiceImpl">

    </bean>
</beans>
```

- Action注入Service

```
/**
* @Title: CustomerAction.java
* @Package com.admiral.ssh.web.action
* @Description:
* @author Admiral
* @date 2020-10-13
* @version V1.0
*/
package com.admiral.ssh.web.action;

import org.apache.struts2.ServletActionContext;
import org.springframework.web.context.WebApplicationContext;
import org.springframework.web.context.support.WebApplicationContextUtils;

import com.admiral.ssh.domain.Customer;
import com.admiral.ssh.service.CustomerService;
```

```java
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ModelDriven;

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 Service
    private CustomerService customerService;

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

    /**
     * 保存客户的方法:save
     */
    public String save() {
        System.out.println("CustomerAction 中的 save 方法执行了....");

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

        customerService.save(customer);

        return NONE;
    }

}
```

1.2.2.8 第八步：Spring整合Struts2方式二：Action交给Spring管理（推荐）

- 引入插件包
    - 引入struts-spring-plugin.jar
- 将Action交给Spring

```xml
<?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"
    xmlns:tx="http://www.springframework.org/schema/tx"
    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.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx.xsd">

    <!-- 将 Action 交给 Spring 管理 -->
    <bean id="customerAction" class="com.admiral.ssh.web.action.CustomerAction">

    </bean>

    <!-- 配置 CustomerService -->
    <bean id="customerService"
class="com.admiral.ssh.service.impl.CustomerServiceImpl">

    </bean>
</beans>
```

- 在struts.xml中配置Action

```
<?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="ssh1" extends="struts-default" namespace="/">
        <action name="customer_*" class="customerAction" method="{1}">

        </action>
    </package>
</struts>
```

- 注意:
    - 需要配置 Action 为多例

    ```
    <?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"
        xmlns:tx="http://www.springframework.org/schema/tx"
        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.xsd
    ```

```xml
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx.xsd">

    <!-- 将 Action 交给 Spring 管理 -->
    <bean id="customerAction"
class="com.admiral.ssh.web.action.CustomerAction" scope="prototype">
    </bean>

    <!-- 配置 CustomerService -->
    <bean id="customerService"
class="com.admiral.ssh.service.impl.CustomerServiceImpl">

    </bean>
</beans>
```

- 需要手动注入 Service

```xml
<?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"
    xmlns:tx="http://www.springframework.org/schema/tx"
    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.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx.xsd">

    <!-- 将 Action 交给 Spring 管理 -->
    <bean id="customerAction"
class="com.admiral.ssh.web.action.CustomerAction" scope="prototype">
        <property name="customerService" ref="customerService" />
    </bean>

    <!-- 配置 CustomerService -->
    <bean id="customerService"
class="com.admiral.ssh.service.impl.CustomerServiceImpl">

    </bean>
</beans>
```

1.2.2.9 第九步：Service调用DAO

- 将 Dao 交给 Spring 管理

```
    <!-- 将 Dao 交给 Spring 管理 -->
    <bean id="customerDao" class="com.admiral.ssh.dao.impl.CustomerDaoImpl">

    </bean>
```

- 在 Service 中注入 Dao

```

package com.admiral.ssh.service.impl;

import com.admiral.ssh.dao.CustomerDao;
import com.admiral.ssh.domain.Customer;
import com.admiral.ssh.service.CustomerService;

/**
 * 客户模块Service层的实现类
 */
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("CustomerServiceImpl 中的 save 方法执行了....");
    }

}
```

```
    <!-- 配置 CustomerService -->
    <bean id="customerService"
class="com.admiral.ssh.service.impl.CustomerServiceImpl">
        <property name="customerDao" ref="customerDao" />
    </bean>
```

1.2.2.10 第十步：Spring整合Hibernate框架

- 创建数据库和表

```
Create database ssh1;
Use ssh1;
CREATE TABLE `cst_customer` (
  `cust_id` bigint(32) NOT NULL AUTO_INCREMENT COMMENT '客户编号(主键)',
  `cust_name` varchar(32) NOT NULL COMMENT '客户名称(公司名称)',
  `cust_source` varchar(32) DEFAULT NULL COMMENT '客户信息来源',
  `cust_industry` varchar(32) DEFAULT NULL COMMENT '客户所属行业',
  `cust_level` varchar(32) DEFAULT NULL COMMENT '客户级别',
  `cust_phone` varchar(64) DEFAULT NULL COMMENT '固定电话',
  `cust_mobile` varchar(16) DEFAULT NULL COMMENT '移动电话',
  PRIMARY KEY (`cust_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

- 编写实体和映射文件

```

package com.admiral.ssh.domain;

public class Customer {

    private Long cust_id;
    private String cust_name;
    private String cust_source;
    private String cust_industry;
    private String cust_level;
    private String cust_phone;
    private String cust_mobile;
    public Long getCust_id() {
        return cust_id;
    }
    public void setCust_id(Long cust_id) {
        this.cust_id = cust_id;
    }
    public String getCust_name() {
        return cust_name;
    }
    public void setCust_name(String cust_name) {
        this.cust_name = cust_name;
    }
    public String getCust_source() {
        return cust_source;
    }
    public void setCust_source(String cust_source) {
        this.cust_source = cust_source;
    }
    public String getCust_industry() {
        return cust_industry;
    }
    public void setCust_industry(String cust_industry) {
        this.cust_industry = cust_industry;
    }
    public String getCust_level() {
        return cust_level;
    }
    public void setCust_level(String cust_level) {
        this.cust_level = cust_level;
    }
    public String getCust_phone() {
        return cust_phone;
    }
    public void setCust_phone(String cust_phone) {
        this.cust_phone = cust_phone;
    }
    public String getCust_mobile() {
        return cust_mobile;
    }
    public void setCust_mobile(String cust_mobile) {
        this.cust_mobile = cust_mobile;
    }

    @Override
    public String toString() {
        return "Customer [cust_id=" + cust_id + ", cust_name=" + cust_name + ",
cust_source=" + cust_source
                + ", cust_industry=" + cust_industry + ", cust_level=" +
cust_level + ", cust_phone=" + cust_phone
                + ", cust_mobile=" + cust_mobile + "]";
    }
}
```

```
<?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>
    <class name="com.admiral.ssh.domain.Customer" table="cst_customer">
        <!--建立类中的属性与表中的主键的映射关系 -->
        <id name="cust_id" column="cust_id">
            <!--主键的生成策略 -->
            <generator class="native"></generator>
        </id>

        <!-- 建立类中的普通属性与表中的字段的映射 -->
        <property name="cust_name" column="cust_name" />
        <property name="cust_source" column="cust_source" />
        <property name="cust_industry" column="cust_industry" />
        <property name="cust_level" column="cust_level" />
        <property name="cust_phone" column="cust_phone" />
        <property name="cust_mobile" column="cust_mobile" />
    </class>
</hibernate-mapping>
```

```
<?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>
        <!-- ================必要的配置================ -->
        <!-- 必要的配置信息,连接数据库的基本参数 -->
        <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property
name="hibernate.connection.url">jdbc:mysql:///hibernate_04</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">111111</property>
        <!-- Hibernate 的方言:根据配置的方言生成对应的 SQL 语句 -->
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

        <!-- 配置C3P0连接池 -->
        <property
name="connection.provider_class">org.hibernate.connection.C3P0ConnectionProvider
</property>
        <!--在连接池中可用的数据库连接的最少数目 -->
        <property name="c3p0.min_size">5</property>
        <!--在连接池中所有数据库连接的最大数目 -->
        <property name="c3p0.max_size">20</property>
        <!--设定数据库连接的过期时间,以秒为单位, 如果连接池中的某个数据库连接处于空闲状态的时
间超过了timeout时间,就会从连接池中清除 -->
        <property name="c3p0.timeout">120</property>
        <!--每3000秒检查所有连接池中的空闲连接 以秒为单位 -->
        <property name="c3p0.idle_test_period">3000</property>

        <!-- ================可选的配置================ -->
        <!-- Hibernate 显示 SQL 语句: -->
        <property name="hibernate.show_sql">true</property>
        <!-- Hibernate 格式化 SQL 语句: -->
        <property name="hibernate.format_sql">true</property>
        <!-- Hibernate 自动创建表 -->
        <property name="hibernate.hbm2ddl.auto">update</property>


        <!-- 加载映射文件 -->
        <mapping resource="com/admiral/ssh/domain/Customer.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

- Spring 和 Hibernate 整合
    - 在 Spring 配置文件中引入 Hibernate 的配置信息

```
    <!-- Spring 整合 Hibernate -->
    <!-- 引入 Hibernate 配置的信息 -->
    <bean id="sessionFactory"
class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
        <property name="configLocation" value="classpath:hibernate.cfg.xml" />
    </bean>
```

- 在 Spring 和 Hibernate 整合后, Spring 提供了一个 Hibernate 的模板类简化 Hibernate 的开发
    - 改写 Dao 继承 HibernateDaoSupport

```java
package com.admiral.ssh.dao.impl;

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

import com.admiral.ssh.dao.CustomerDao;
import com.admiral.ssh.domain.Customer;

/**
 * 客户模块Dao层的实现类
 */
public class CustomerDaoImpl extends HibernateDaoSupport implements
CustomerDao {

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

}
```

    - 配置的时候在 Dao 中直接注入 SessionFactory

```xml
    <!-- 将 Dao 交给 Spring 管理 -->
    <bean id="customerDao" class="com.admiral.ssh.dao.impl.CustomerDaoImpl">
        <property name="sessionFactory" ref="sessionFactory" />
    </bean>
```

    - 在 Dao 中使用 Hibernate 的模板完成保存操作

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

1.2.2.11 第十一步：配置Spring的事务管理

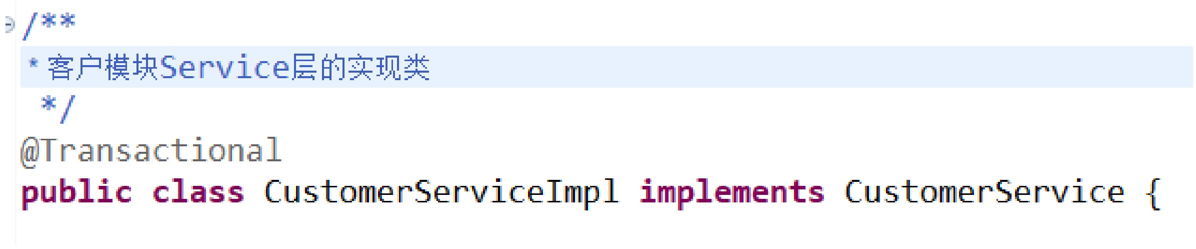
- 配置平台事务管理器

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

- 开启注解事务

```
    <!-- 开启注解事务 -->
    <tx:annotation-driven transaction-manager="tracsactionManager"/>
```

- 在 Service 层使用事务



```
/**
 * 客户模块Service层的实现类
 */
@Transactional
public class CustomerServiceImpl implements CustomerService {
```

1.3 SSH整合方式二：将hibernate的配置交给Spring管理

1.3.1 SSH整合方式二：不带hibernate配置文件

1.3.1.1 复制一个项目

1.3.1.2 hibernate配置文件中有哪些内容：

- 数据库连接的配置
- Hibernate的相关的属性的配置
    - 方言
    - 显示SQL
    - 格式化SQL
    - 。。。
- C3P0连接池
- 映射文件

1.3.1.3 将Hibernate的配置交给Spring

```
<?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"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
```

```xml
    http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx.xsd">


    <!-- =============   Spring 整合 Hibernate   ============= -->

    <!-- 引入外部属性文件 -->
    <context:property-placeholder location="classpath:jdbc.properties"/>

    <!-- 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>



    <!-- 引入 Hibernate 配置的信息 -->
    <bean id="sessionFactory"
class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">

        <!-- 注入连接池 -->
        <property name="dataSource" ref="dataSource" />

        <!-- 配置 Hibernate 相关属性 -->
        <property name="hibernateProperties">
            <props>
                <prop
key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
                <prop key="hibernate.show_sql">true</prop>
                <prop key="hibernate.format_sql">true</prop>
                <prop key="hibernate.hbm2ddl.auto">update</prop>
            </props>
        </property>

        <!-- 加载映射文件 -->
        <property name="mappingResources">
            <list>
                <value>com/admiral/ssh/domain/Customer.hbm.xml</value>
            </list>
        </property>
    </bean>


    <!-- 将 Action 交给 Spring 管理 -->
    <bean id="customerAction" class="com.admiral.ssh.web.action.CustomerAction"
scope="prototype">
        <property name="customerService" ref="customerService" />
    </bean>

    <!-- 配置 CustomerService -->
    <bean id="customerService"
class="com.admiral.ssh.service.impl.CustomerServiceImpl">
```

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

    <!-- 将 Dao 交给 Spring 管理 -->
    <bean id="customerDao" class="com.admiral.ssh.dao.impl.CustomerDaoImpl">
        <property name="sessionFactory" ref="sessionFactory" />
    </bean>

    <!-- 配置平台事务管理器 -->
    <bean id="tracsactionManager"
class="org.springframework.orm.hibernate5.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory" />
    </bean>

    <!-- 开启注解事务 -->
    <tx:annotation-driven transaction-manager="tracsactionManager"/>

</beans>
```

1.4 Hibernate的模板的使用

1.4.1 Hibernate模板的常用的方法

1.4.1.1 保存操作

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

1.4.1.2 修改操作

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

```

1.4.1.3 删除操作

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

1.4.1.4 查询操作

```
    @Override
    public Customer findById(Long cust_id) {
        return this.getHibernateTemplate().get(Customer.class, cust_id);
    }

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

    @Override
    public List<Customer> findByQBC() {
        DetachedCriteria criteria = DetachedCriteria.forClass(Customer.class);
        List<Customer> list = (List<Customer>)
this.getHibernateTemplate().findByCriteria(criteria);
        return list;
    }

    @Override
    public List<Customer> findByNamedQuery() {
        List<Customer> list = (List<Customer>)
this.getHibernateTemplate().findByNamedQuery("queryAll");
        return list;
    }
```

```
<?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>
    <class name="com.admiral.ssh.domain.Customer" table="cst_customer">
        <!--建立类中的属性与表中的主键的映射关系 -->
        <id name="cust_id" column="cust_id">
            <!--主键的生成策略 -->
            <generator class="native"></generator>
        </id>

        <!-- 建立类中的普通属性与表中的字段的映射 -->
        <property name="cust_name" column="cust_name" />
        <property name="cust_source" column="cust_source" />
        <property name="cust_industry" column="cust_industry" />
        <property name="cust_level" column="cust_level" />
        <property name="cust_phone" column="cust_phone" />
        <property name="cust_mobile" column="cust_mobile" />
    </class>

    <query name="queryAll">from Customer</query>
```

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

1.5 延迟加载问题的解决

1.5.1 Spring提供了延迟加载的解决方案

1.5.1.1 在SSH整合开发中哪些地方会出现延迟加载

- 使用load方法查询某一个对象的时候（不常用）
- **查询到某个对象以后，显示其关联对象。**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
  <display-name>ssh1</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>

  <!-- 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>

  <!-- 配置解决延迟加载问题的过滤器 -->
  <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>

  <!-- 配置 Struts2 核心过滤器 -->
  <filter>
    <filter-name>struts2</filter-name>
    <filter-
class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</fil
ter-class>
  </filter>

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