Action类

```
/**
* 模型驱动封装用户提交的数据,必须重写的方法
*/
private Customer customer = new Customer();
@Override
public Customer getModel() {
   return customer;
}
```

Dao类继承HibernateDaoSupport

```
getHibernateTemplate().save(customer);
```

JSP页面保存使用的Struts2 form表单标签

```
<s:form action="addCustomer.html" namespace="/customer">
用户的输入.....
</s:form>
```

使用**struts2**的动作请求形式：（不管是哪种请求模式最终都将封装进模型驱动的实体类中）

post请求（**struts2**默认就是**post**请求的动作）使用**html**的超链接标签

```
<A class=style2 href="${pageContext.request.contextPath}/customer/findAllCustomer.html"
target=main>一 客户列表</A>
```

相当于带参的**post**请求，使用**struts2**的**form**表单形式，多用于对表数据的修改操作

提交是**POST**请求，但有一个隐藏的参数可以被类接收并封装处理，其实相当于带请求参数的**get**请求

```
<s:form action="editCustomer.html" namespace="/customer">
 <s:hidden name="custId" value="%{custId}"></s:hidden>
</s:form>
```

get请求（使用**OGNL**表达式封装参数传递参数给**javascript**并使用页面的**get**请求响应处理）

```
<s:a href="javascript:delOne('%{custId}')">删除</s:a>
function delOne(custId){
  var sure = window.confirm("确定删除吗？");
  if(sure){
     window.location.href="${pageContext.request.contextPath}/customer/removeCustomer.html?
custId="+custId;
  }
}
```

get请求：使用struts2的超链接标签并包含一个参数标签来定义请求的参数

struts2的超链接标签

<s:a>

action：指定动作名称

namespace：指定名称空间

href：和html的超链接标签作用是一样的。

<s:param>标签用于给超链接标签拼接请求参数。用的是get的方式拼接

属性：

name：指定请求参数key的部分

value：指定请求参数值的部分

```
<s:a action="editUICustomer" namespace="/customer">
<s:param name="custId" value="%{custId}"></s:param>
编辑
</s:a>
```

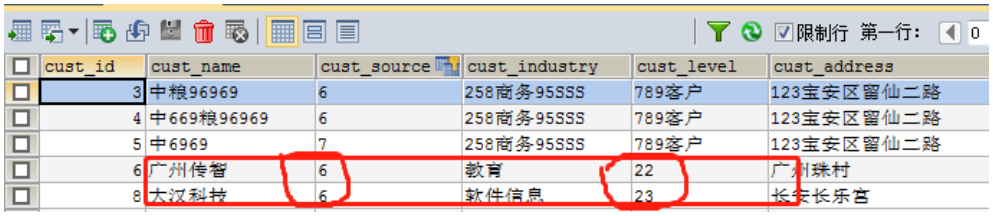
字典表技术：

```
/**
 * 一个字典表的数据模型
 */
public class BaseDict implements Serializable {

    private String dictId;
    private String dictTypeCode;
    private String dictTypeName;
    private String dictItemName;
    private String dictItemCode;
    private Integer dictSort;
    private String dictEnable;
    private String dictMemo;
```

```
/**
 * 对应数据库表的实体JAVABean类 实现序列化 并生成get/set方法
 * 主表（客户）信息
 */
public class Customer implements Serializable {
    private Integer custId;
    private String custName;
//  private String custSource;
    private String custIndustry;
    private String custLevel;
    private String custAddress;
    private String custPhone;

    //单向多对一关系（多个客户来源信息对应一个字典表）
    private BaseDict custSource;
```

客户与字典表的关系

dict_id	dict_type_code	dict_type_name	dict_item_name	dict_item_
22	006	客户级别	普通客户	(NULL)
23	006	客户级别	VIP客户	(NULL)
24	007	商机状态	意向客户	(NULL)
25	007	商机状态	初步沟通	(NULL)
26	007	商机状态	深度沟通	(NULL)
27	007	商机状态	签订合同	(NULL)



cust_id	cust_name	cust_source	cust_industry	cust_level	cust_address
3	中粮96969	6	258商务95SSS	789客户	123宝安区留仙二路
4	中669粮96969	6	258商务95SSS	789客户	123宝安区留仙二路
5	中6969	7	258商务95SSS	789客户	123宝安区留仙二路
6	广州传智	6	教育	22	广州珠村
8	大汉科技	6	软件信息	23	长安长乐宫

```
<!--

客户信息来源和客户级别都使用了字典表技术实现

name = custSource.dictId:
页面提交数据模型封装到对象位置
改页面是添加客户信息的页面有提交保存功能，所以对这个类的数据有模型封装的需求，
模型封装的需求就必须要表单的name能和实体类变量名一一对应才可以
list="customerSources":
这个是栈值数据（查询字典表结束后的数据集合的结果 ）

-->
<s:select name="custSource.dictId" list="customerSources" listKey="dictId" listValue="dictItemName"
headerKey="" headerValue="---请选择---" class="textbox" id="sChannel2" style="WIDTH: 180px"></s:select>
```

```
/**
 * 进入添加客户的页面
 */
private List<BaseDict> customerSources;
private List<BaseDict> customerLevels;
public String addUICustomer() {
    // 从字典表获取所有来源信息
    customerSources = customerService.findAllCustomerSource();
    // 从字典表获取所有级别信息
    customerLevels = customerService.findAllCustomerLevel();

    return "addUICustomer";
}
```

由于有多次查询，所以需用配置Session的开关时间

在Struts2的过滤器中配置

在项目 中是配置Web.xml里处理

数据回显：

```
/**
 * 获取编辑客户页面
 * @return
 */
public String editUICustomer(){
    //查询所有客户来源
    customerSources = customerService.findAllCustomerSource();
    //查询所有客户级别
    customerLevels = customerService.findAllCustomerLevel();
    //1.根据id查询客户信息
    Customer dbCustomer = customerService.findCustomerById(customer.getCustId());
    //2.获取值栈对象
    ValueStack vs = ActionContext.getContext().getValueStack();
    //3.把查询出来的客户压栈
    vs.push(dbCustomer);
    return "editUICustomer";
}
```

查客户来源，级别 回显及编辑的字典技术使用

通过id获取用户要编辑的是谁的数据

压栈是为了页面可以回显要编辑的原数据

```
<td>客户名称：</td>
<td>
    <s:textfield name="custName" class="textbox" id="sChannel2" style="WIDTH:
    180px" maxLength="50"></s:textfield>
</td>
<td>所属行业 ：</td>
<td>
    <s:textfield name="custIndustry" class="textbox" id="sChannel2"
    style="WIDTH: 180px" maxLength="50"></s:textfield>
</td>
```

使用Struts2的标签来实现回显数据

配置过滤器：用于控制**Session**的获取和关闭

```
<!-- 配置过滤器：用于控制Session的获取和关闭 -->
<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>/*</url-pattern>
</filter-mapping>
```

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

多条件查询技术：DetachedCriteria dCriteria对查询条件封装采用QBC的查询方式

Action类的条件封装

```
DetachedCriteria dCriteria = DetachedCriteria.forClass(Customer.class);// 查询所有
if (StringUtils.isNotBlank(customer.getCustName())) {
    // 如果客户名不为空的话，模糊查询条件：客户名称
    dCriteria.add(Restrictions.like("custName", "%" + customer.getCustName() + "%"));
}
if (StringUtils.isNotBlank(customer.getCustIndustry())) {
    // 如果客户行业不为空的话，模糊查询条件：客户行业
    dCriteria.add(Restrictions.like("custIndustry", "%" + customer.getCustIndustry() + "%"));
}
if (StringUtils.isNotBlank(customer.getCustLevel())) {
    // 如果客户级别不为空的话，精确查询条件：客户级别
    dCriteria.add(Restrictions.eq("custLevel", customer.getCustLevel()));
}
if (customer.getCustSource() != null && StringUtils.isNotBlank(customer.getCustSource().getDictId())) {
    // 如果客户来源不为空的话，精确查询条件：客户来源
    dCriteria.add(Restrictions.eq("custSource.dictId", customer.getCustSource().getDictId()));
}

return dCriteria;
```

如果条件都不符合就是查所有，否则根据提供的条件查

```java
/**
 * 查询所有客户
 */
private List<Customer> customers;    //客户信息值栈操作
public String findAllCustomer() {
    //处理条件查询的数据回显，及数据类型确定
    getDictionaryTableInfo();
    // 4.查询所有客户-带条件
    customers = customerService.findAllorCustomer(conditionAssemble());
    return "findAllCustomer";
}
```

jsp的form表单请求

```
<s:form action="findAllCustomer" namespace="/customer">
<s:textfield name="custName" class="textbox" id="sChannel2" style="WIDTH: 80px" maxLength="50"/>
  使用struts2的标签  可以很好的回显数据
</s:form>
```

Dao持久层的查

```
return (List<Customer>) getHibernateTemplate().findByCriteria(dCriteria);
```