CRM 第二天

第1章 字典表和客户表的关系映射

1.1 字典表介绍

字典表顾名思义，就是用于给其他表作为参照的表。在开发中我们可能会有很多的字典表。例如我们的 CRM 中，可以有客户来源字典表，客户级别字典表，客户规模字典表等等。如果这些表独立出来，维护起来不很方便，所以我们可以把它建立成一张表，用一列来区分是客户来源，还是客户级别，还是客户规模的字典数据。

cust_id	cust_name	cust_source	cust_industry	cust_level	cust_address	cust_phone	规模ID
1	传智播客集团	6	IT培训	23	北京市昌平区	010-666688	1
2	黑马训练营	6	IT培训	23	北京市昌平区	010-666688	1
3	京西集团	7	电商	23	京西玉泉山	010-650855	2
4	修正药业	7	医药	22	北京市昌平区	010-689090	3

客户来源表

来源ID FK	来源名称
1	网络营销
2	电话营销

客户级别表

级别ID FK	级别名称
1	普通客户
2	vip客户

公司规模表

规模ID FK	规模名称
1	1-50人
2	50-150人
3	150-500人

数据字典表：里面包含的都是字典数据

ID FK	名称	字典类型	类型编码 varchar	
1	电话营销	信息来源类型	001	
2	网络营销	信息来源类型	001	
3	普通客户	客户级别类型	002	
4	VIP客户	客户级别类型	002	

表的 SQL 语句如下：

```
/*创建数据字典表*/
CREATE TABLE `base_dict` (
  `dict_id` varchar(32) NOT NULL COMMENT '数据字典 id(主键)',
  `dict_type_code` varchar(10) NOT NULL COMMENT '数据字典类别代码',
  `dict_type_name` varchar(64) NOT NULL COMMENT '数据字典类别名称',
  `dict_item_name` varchar(64) NOT NULL COMMENT '数据字典项目名称',
  `dict_item_code` varchar(10) DEFAULT NULL COMMENT '数据字典项目(可为空)',
  `dict_sort` int(10) DEFAULT NULL COMMENT '排序字段',
  `dict_enable` char(1) NOT NULL COMMENT '1:使用 0:停用',
```

```
  `dict_memo` varchar(64) DEFAULT NULL COMMENT '备注',
  PRIMARY KEY (`dict_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/*为字典表插入数据：Data for the table `base_dict` */

LOCK TABLES `base_dict` WRITE;

insert into `base_dict`(`dict_id`,`dict_type_code`,`dict_type_name`,`dict_item_name`,`dict_item_code`,`dict_sort`,`dict_enable`,`dict_memo`)
values ('1','001','客户行业','教育培训 ',NULL,1,'1',NULL),
('10','003','公司性质','民企',NULL,3,'1',NULL),
('12','004','年营业额','1-10 万',NULL,1,'1',NULL),
('13','004','年营业额','10-20 万',NULL,2,'1',NULL),
('14','004','年营业额','20-50 万',NULL,3,'1',NULL),
('15','004','年营业额','50-100 万',NULL,4,'1',NULL),
('16','004','年营业额','100-500 万',NULL,5,'1',NULL),
('17','004','年营业额','500-1000 万',NULL,6,'1',NULL),
('18','005','客户状态','基础客户',NULL,1,'1',NULL),
('19','005','客户状态','潜在客户',NULL,2,'1',NULL),
('2','001','客户行业','电子商务',NULL,2,'1',NULL),
('20','005','客户状态','成功客户',NULL,3,'1',NULL),
('21','005','客户状态','无效客户',NULL,4,'1',NULL),
('22','006','客户级别','普通客户',NULL,1,'1',NULL),
('23','006','客户级别','VIP 客户',NULL,2,'1',NULL),
('24','007','商机状态','意向客户',NULL,1,'1',NULL),
('25','007','商机状态','初步沟通',NULL,2,'1',NULL),
('26','007','商机状态','深度沟通',NULL,3,'1',NULL),
('27','007','商机状态','签订合同',NULL,4,'1',NULL),
('3','001','客户行业','对外贸易',NULL,3,'1',NULL),
('30','008','商机类型','新业务',NULL,1,'1',NULL),
('31','008','商机类型','现有业务',NULL,2,'1',NULL),
('32','009','商机来源','电话营销',NULL,1,'1',NULL),
('33','009','商机来源','网络营销',NULL,2,'1',NULL),
('34','009','商机来源','推广活动',NULL,3,'1',NULL),
('4','001','客户行业','酒店旅游',NULL,4,'1',NULL),
('5','001','客户行业','房地产',NULL,5,'1',NULL),
('6','002','客户信息来源','电话营销',NULL,1,'1',NULL),
('7','002','客户信息来源','网络营销',NULL,2,'1',NULL),
('8','003','公司性质','合资',NULL,1,'1',NULL),
('9','003','公司性质','国企',NULL,2,'1',NULL);
UNLOCK TABLES;
```

1.2 字典表和客户表的关系分析

通过上图我们可以得知：

一个客户只能有一个来源（级别）。多个客户可能有同一个来源（级别）。

所以客户表和字典表的之间的关系是多对一。（此处要注意方向性）

无论多对一还是一对多，在数据库中都是依靠外键约束来实现的。

1.3 字典表和客户表的实体类映射配置

```
/**
 * 客户的实体类
 *
 *
 * 明确使用的注解都是 JPA 规范的
 * 所以导包都要导入 javax.persistence 包下的
 *
 */
@Entity//表示当前类是一个实体类
@Table(name="cst_customer")//建立当前实体类和表之间的对应关系
public class Customer implements Serializable {

    @Id//表明当前私有属性是主键
    @GeneratedValue(strategy=GenerationType.IDENTITY)//指定主键的生成策略
    @Column(name="cust_id")//指定和数据库表中的 cust_id 列对应
    private Long custId;
    @Column(name="cust_name")//指定和数据库表中的 cust_name 列对应
    private String custName;

    @Column(name="cust_industry")//指定和数据库表中的 cust_industry 列对应
    private String custIndustry;

    @Column(name="cust_address")//指定和数据库表中的 cust_address 列对应
    private String custAddress;
    @Column(name="cust_phone")//指定和数据库表中的 cust_phone 列对应
    private String custPhone;

    @ManyToOne(targetEntity=BaseDict.class)
    @JoinColumn(name="cust_source",referencedColumnName="dict_id")
    private BaseDict custSource;

    @ManyToOne(targetEntity=BaseDict.class)
    @JoinColumn(name="cust_level",referencedColumnName="dict_id")
    private BaseDict custLevel;
```

```
    public Long getCustId() {
        return custId;
    }
    public void setCustId(Long custId) {
        this.custId = custId;
    }
    public String getCustName() {
        return custName;
    }
    public void setCustName(String custName) {
        this.custName = custName;
    }
    public String getCustIndustry() {
        return custIndustry;
    }
    public void setCustIndustry(String custIndustry) {
        this.custIndustry = custIndustry;
    }
    public String getCustAddress() {
        return custAddress;
    }
    public void setCustAddress(String custAddress) {
        this.custAddress = custAddress;
    }
    public String getCustPhone() {
        return custPhone;
    }
    public void setCustPhone(String custPhone) {
        this.custPhone = custPhone;
    }
    public BaseDict getCustSource() {
        return custSource;
    }
    public void setCustSource(BaseDict custSource) {
        this.custSource = custSource;
    }
    public BaseDict getCustLevel() {
        return custLevel;
    }
    public void setCustLevel(BaseDict custLevel) {
        this.custLevel = custLevel;
    }
    @Override
    public String toString() {
```

```
            return "Customer [custId=" + custId + ", custName=" + custName
                    + ", custIndustry=" + custIndustry + ", custAddress="
                    + custAddress + ", custPhone=" + custPhone + "]";
        }
    }

    /**
     * 字典表的数据模型
     *
     */
    @Entity
    @Table(name="base_dict")
    public class BaseDict implements Serializable {

        @Id
        @GeneratedValue(strategy=GenerationType.AUTO)
        @Column(name="dict_id")
        private String dictId;
        @Column(name="dict_type_code")
        private String dictTypeCode;
        @Column(name="dict_type_name")
        private String dictTypeName;
        @Column(name="dict_item_name")
        private String dictItemName;
        @Column(name="dict_item_code")
        private String dictItemCode;
        @Column(name="dict_sort")
        private Integer dictSort;
        @Column(name="dict_enable")
        private String dictEnable;
        @Column(name="dict_memo")
        private String dictMemo;
        public String getDictId() {
            return dictId;
        }
        public void setDictId(String dictId) {
            this.dictId = dictId;
        }
        public String getDictTypeCode() {
            return dictTypeCode;
        }
        public void setDictTypeCode(String dictTypeCode) {
            this.dictTypeCode = dictTypeCode;
        }
```

```
        public String getDictTypeName() {
            return dictTypeName;
        }
        public void setDictTypeName(String dictTypeName) {
            this.dictTypeName = dictTypeName;
        }
        public String getDictItemName() {
            return dictItemName;
        }
        public void setDictItemName(String dictItemName) {
            this.dictItemName = dictItemName;
        }
        public String getDictItemCode() {
            return dictItemCode;
        }
        public void setDictItemCode(String dictItemCode) {
            this.dictItemCode = dictItemCode;
        }
        public Integer getDictSort() {
            return dictSort;
        }
        public void setDictSort(Integer dictSort) {
            this.dictSort = dictSort;
        }
        public String getDictEnable() {
            return dictEnable;
        }
        public void setDictEnable(String dictEnable) {
            this.dictEnable = dictEnable;
        }
        public String getDictMemo() {
            return dictMemo;
        }
        public void setDictMemo(String dictMemo) {
            this.dictMemo = dictMemo;
        }
        @Override
        public String toString() {
            return "BaseDict [dictId=" + dictId + ", dictTypeCode=" + dictTypeCode
+ ", dictTypeName=" + dictTypeName
                    + ", dictItemName=" + dictItemName + ", dictItemCode=" +
dictItemCode + ", dictSort=" + dictSort
                    + ", dictEnable=" + dictEnable + ", dictMemo=" + dictMemo +
"]";
```

```
        }
    }
```

第2章 客户的增删改查操作

2.1 写在最前

本章节提供的是客户增删改查的代码实现。
下面出现的 Action 指的的是：CustomerAction
出现的 Service 指的是：ICustomerService 和 CustomerServiceImpl
出现的 Dao 指的是：CustomerDao 和 CustomerDaoImpl。
这些类都需要交给 spring 来管理。
在没有提供新的类（或接口）时，从 2.2 章节开始的 Action,Service 和 Dao 的代码都是出现在以下的类中。

Action

```
/**
 * 客户的动作类
 *
 *
 */
@Controller("customerAction")
@Scope("prototype")
@ParentPackage("struts-default")
@Namespace("/customer")
public class CustomerAction extends ActionSupport implements
ModelDriven<Customer> {
        private Customer customer = new Customer();

        @Autowired
        private ICustomerService customerService ;

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

Service

```
/**
 * 客户的业务层接口
```

```
 *
 */
public interface ICustomerService {
}

/**
 * 客户的业务层实现类
 *
 */
@Service("customerService")
@Transactional(propagation=Propagation.SUPPORTS,readOnly=true)
public class CustomerServiceImpl implements ICustomerService {

    @Autowired
    private ICustomerDao customerDao;
}
```

Dao

```
/**
 * 客户的持久层接口
 *
 */
public interface ICustomerDao {
}

/**
 * 客户的持久层实现类
 *
 */
@Repository("customerDao")
public class CustomerDaoImpl implements ICustomerDao {
    @Autowired
    private HibernateTemplate hibernateTemplate;
}
```

2.2 显示添加客户页面

2.2.1 menu.jsp 页面

```
<A
href="${pageContext.request.contextPath}/customer/addUICustomer.action"
```

```
        class=style2 target=main>
        — 新增客户
    </A>
```

2.2.2 Action

```
    /**
     * 获取添加客户页面
     * @return
     */
    private List<BaseDict> custSources;
    private List<BaseDict> custLevels;

    @Action(value="addUICustomer",results={
        @Result(name="addUICustomer",type="dispatcher",location="/jsp/custome
r/add.jsp")
    })
    public String addUICustomer(){
        //1.查询所有客户来源
        custSources = customerService.findAllCustomerSource();
        //2.查询所有客户级别
        custLevels = customerService.findAllCustomerLevel();
        return "addUICustomer";
    }

    public List<BaseDict> getCustLevels() {
        return custLevels;
    }
    public void setCustLevels(List<BaseDict> custLevels) {
        this.custLevels = custLevels;
    }
    public List<BaseDict> getCustSources() {
        return custSources;
    }
    public void setCustSources(List<BaseDict> custSources) {
        this.custSources = custSources;
    }
```

2.2.3 Service

```
        /**
         * 查询所有客户来源
         * @return
```

```
     */
    List<BaseDict> findAllCustomerSource();

    /**
     * 查询所有客户级别
     * @return
     */
    List<BaseDict> findAllCustomerLevel();

    @Override
    @Transactional(propagation=Propagation.SUPPORTS,readOnly=true)
    public List<BaseDict> findAllCustomerSource() {
        return baseDictDao.findBaseDictByTypeCode("002");
    }

    @Override
    @Transactional(propagation=Propagation.SUPPORTS,readOnly=true)
    public List<BaseDict> findAllCustomerLevel() {
        return baseDictDao.findBaseDictByTypeCode("006");
    }
```

2.2.4 Dao

```
/**
 * 数据字典实体的持久层接口
 *
 *
 */
public interface IBaseDictDao {
    /**
     * 根据字典类型查询字典表数据
     * @param typeCode      字典类型编码
     * @return
     */
    List<BaseDict> findBaseDictByTypeCode(String typeCode);
}

/**
 * 数据字典实体的持久层实现类
 *
 */
@Repository("baseDictDao")
public class BaseDictDaoImpl implements IBaseDictDao {
```

```
        @Autowired
        private HibernateTemplate hibernateTemplate;

        @Override
        public List<BaseDict> findBaseDictByTypeCode(String typeCode) {
            return (List<BaseDict>) hibernateTemplate.find("from BaseDict where
dictTypeCode = ? ", typeCode);
        }
    }
```

2.3保存客户

2.3.1 add.jsp

```
    <s:form action="addCustomer" namespace="/customer">
        <TABLE cellSpacing=0 cellPadding=5 border=0>
            <TR>
                <td>客户名称：</td>
                <td>
                    <s:textfield name="custName" class="textbox" id="sChannel2"
style="WIDTH: 180px" maxLength="50" />
                </td>
                <td>所属行业 ：</td>
                <td>
                    <s:textfield name="custIndustry" class="textbox"
id="sChannel2" style="WIDTH: 180px" maxLength="50" />
                </td>
            </TR>
            <TR>
                td>信息来源 ：</td>
                <td>
                    <s:select list="custSources" name="custSource.dictId"
listKey="dictId" listValue="dictItemName" headerKey="" headerValue="---请选择
---" class="textbox" id="sChannel2" style="WIDTH: 180px"></s:select>
                </td>
                <td>客户级别：</td>
                <td>
                    <s:select list="custLevels" name="custLevel.dictId"
listKey="dictId" listValue="dictItemName" headerKey="" headerValue="---请选择
---" class="textbox" id="sChannel2" style="WIDTH: 180px"></s:select>
                            </td>
            </TR>
```

```
            <TR>
                <td>联系地址 : </td>
                <td>
                    <s:textfield       name="custAddress"       class="textbox"
id="sChannel2" style="WIDTH: 180px" maxLength="50" />
                </td>
                <td>联系电话 : </td>
                <td>
                    <s:textfield       name="custPhone"       class="textbox"
id="sChannel2" style="WIDTH: 180px" maxLength="50" />

                </td>
            </TR>
            <tr>
                <td rowspan=2>
                    <s:submit value="保存"/>
                </td>
            </tr>
        </TABLE>
    </s:form>
```

2.3.2 Action

```
    @Action(value="addCustomer",results={
        @Result(name="addCustomer",type="redirect",location="/jsp/success.jsp
")
    })
    public String addCustomer(){
        customerService.saveCustomer(customer);
        return "addCustomer";
    }
```

2.3.3 Service

```
    /**
     * 保存客户
     * @param customer
     */
    void saveCustomer(Customer customer);


    @Override
```

```
    public void saveCustomer(Customer customer) {
        customerDao.save(customer);
    }
```

2.3.4 Dao

```
    /**
     * 保存客户
     * @param customer
     */
    void save(Customer customer);


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

2.4客户列表展示

2.4.1 menu.jsp

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

2.4.2 Action

```
    /**
     * 查询所有客户
     * @return
     */
    private List<Customer> customers;
    @Action(value="findAllCustomer",results={
        Result(name="findAllCustomer",location="/jsp/cusotmer/list.jsp")
    })
    public String findAllCustomer(){
        //1.查询所有客户
        customers = customerService.findAllCustomer();
        return "findAllCustomer";
```

```
}

public List<Customer> getCustomers() {
    return customers;
}
public void setCustomers(List<Customer> customers) {
    this.customers = customers;
}
```

2.4.3 Service

```
/**
 * 查询所有客户信息
 * @return
 */
List<Customer> findAllCustomer();

@Override
@Transactional(propagation=Propagation.SUPPORTS,readOnly=true)
public List<Customer> findAllCustomer() {
    return customerDao.findAllCustomer();
}
```

2.4.4 Dao

```
/**
 *查询所有客户
 * @return
 */
List<Customer> findAllCustomer();

@Override
public List<Customer> findAllCustomer() {
    return (List<Customer>) hibernateTemplate.find("from Customer ");
}
```

2.5 客户删除

2.5.1 list.jsp

```
<s:a href="javascript:delOne('${custId}')">删除</s:a>
<SCRIPT language=javascript>
```

```
        function delOne(custId){
            var sure = window.confirm("确定删除吗？");
            if(sure){
        window.location.href="${pageContext.request.contextPath}/customer/rem
oveCustomer.action?custId="+custId;
            }
        }
    </SCRIPT>
```

2.5.2 Action

```
    /**
     * 删除客户
     * @return
     */
    @Action(value="removeCustomer",results={
        @Result(name="removeCustomer",type="redirect",location="/jsp/success.
jsp")
    })
    public String removeCustomer(){
        customerService.removeCustomer(customer);
        return "removeCustomer";
    }
```

2.5.3 Service

```
    /**
     * 删除客户
     * @param customer
     */
    void removeCustomer(Customer customer);

    @Override
    public void removeCustomer(Customer customer) {
        customerDao.delete(customer);
    }
```

2.5.4 Dao

```
    /**
     * 删除客户
     * @param customer
```

```
     */
    void delete(Customer customer);

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

2.6显示客户修改页面

2.6.1 list.jsp

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

2.6.2 Action

```
    /**
     * 获取编辑客户页面
     * @return
     */
    @Action(value="editUICustomer",results={
        @Result(name="editUICustomer",type="dispatcher",location="/jsp/custom
er/edit.jsp")
    })
    public String editUICustomer(){
        //1.根据 id 查询要编辑的客户对象
        Customer                          dbCustomer                          =
customerService.findCustomerById(customer.getCustId());
        //2.获取值栈对象
        ValueStack vs = ActionContext.getContext().getValueStack();
        //3.把查询出来的客户对象压入栈顶
        vs.push(dbCustomer);
        //4.查询所有客户来源和客户级别
        custSources = customerService.findAllCustomerSource();
        custLevels = customerService.findAllCustomerLevel();
        return "editUICustomer";
    }
```

2.6.3 Service

```
    /**
     * 根据id查询客户信息
     * @param custId
     * @return
     */
    Customer findCustomerById(Long custId);

    @Override
    @Transactional(propagation=Propagation.SUPPORTS,readOnly=true)
    public Customer findCustomerById(Long custId) {
        return customerDao.findById(custId);
    }
```

2.6.4 Dao

```
    /**
     * 根据id查询客户
     * @param customer
     */
    void findById(Long custId);


    @Override
    public void findById (Long custId) {
        hibernateTemplate.get(Customer.class,custId);
    }
```

2.7客户修改

2.7.1 edit.jsp

```
    <s:form action="editCustomer" namespace="/customer">
        <s:hidden name="custId" value="%{custId}"></s:hidden>
        <TABLE cellSpacing=0 cellPadding=5  border=0>
            <TR>
                <td>客户名称：</td>
                <td>
                    <s:textfield name="custName" class="textbox" id="sChannel2"
style="WIDTH: 180px" maxLength="50" />
```

```
                </td>
                <td>所属行业 : </td>
                <td>
                    <s:textfield      name="custIndustry"      class="textbox"
id="sChannel2" style="WIDTH: 180px" maxLength="50" />
                </td>
            </TR>
            <TR>
                td>信息来源 : </td>
                <td>
                    <s:select   list="custSources"    name="custSource.dictId"
listKey="dictId" listValue="dictItemName" headerKey="" headerValue="---请选择
---" class="textbox" id="sChannel2" style="WIDTH: 180px"></s:select>
                </td>
                <td>客户级别：</td>
                <td>
                    <s:select   list="custLevels"      name="custLevel.dictId"
listKey="dictId" listValue="dictItemName" headerKey="" headerValue="---请选择
---" class="textbox" id="sChannel2" style="WIDTH: 180px"></s:select>
                            </td>
            </TR>
            <TR>
                <td>联系地址 : </td>
                <td>
                    <s:textfield      name="custAddress"      class="textbox"
id="sChannel2" style="WIDTH: 180px" maxLength="50" />
                </td>
                <td>联系电话 : </td>
                <td>
                    <s:textfield       name="custPhone"       class="textbox"
id="sChannel2" style="WIDTH: 180px" maxLength="50" />

                </td>
            </TR>
            <tr>
                <td rowspan=2>
                    <s:submit value="保存"/>
                </td>
            </tr>
        </TABLE>
    </s:form>
```

2.7.2 Action

```
    /**
     * 编辑客户
     * @return
     */
    @Action(value="editCustomer",results={
        @Result(name="editCustomer",type="redirect",location="/jsp/success.jsp")
    })
    public String editCustomer(){
        customerService.updateCustomer(customer);
        return "editCustomer";
    }
```

2.7.3 Service

```
    /**
     * 编辑客户
     * @param customer
     */
    void updateCustomer(Customer customer);

    @Override
    public void updateCustomer(Customer customer) {
        customerDao.update(customer);
    }
```

2.7.4 Dao

```
    /**
     * 更新客户
     * @param customer
     */
    void update(Customer customer);


    @Override
    public void delete(Customer customer) {
        hibernateTemplate.update(customer);
    }
```

2.8客户的列表条件查询

2.8.1 list.jsp

```
    <s:form action="findAllCustomer" namespace="/customer">
        <TR>
            <td>客户名称：</td>
            <td>
                <s:textfield  name="custName"  class="textbox"  id="sChannel2"
style="WIDTH: 80px" maxLength="50" />
            </td>
            <td>所属行业 ：</td>
            <td>
                <s:textfield name="custIndustry" class="textbox" id="sChannel2"
style="WIDTH: 80px" maxLength="50" />
            </td>
            <td>信息来源 ：</td>
            <td>
                <s:select      list="custSources"           name="custSourceId"
listKey="dictId" listValue="dictItemName" headerKey="" headerValue="---请选择
---" class="textbox" id="sChannel2" style="WIDTH: 80px"></s:select>
            </td>
            <td>客户级别：</td>
            <td>
                <s:select      list="custLevels"            name="custLevelId"
listKey="dictId" listValue="dictItemName" headerKey="" headerValue="---请选择
---" class="textbox" id="sChannel2" style="WIDTH: 80px"></s:select>

            </td>
            <TD>
                <s:submit value=" 筛选 "></s:submit>
            </TD>
        </TR>
    </s:form>
```

2.8.2 Action

```
    /**
     * 查询所有客户
     * @return
     */
```

```
    @Action(value="findAllCustomer",results={
        @Result(name="findAllCustomer",type="dispatcher",location="/jsp/custo
mer/list.jsp")
    })
    public String findAllCustomer(){
        //1.创建离线查询对象
        DetachedCriteria                    dCriteria                    =
DetachedCriteria.forClass(Customer.class);//就相当于查询所有  from Customer;
        //2.拼装查询条件
        //判断是否提供了客户名称
        if(StringUtils.isNotBlank(customer.getCustName())){
            //添加条件：模糊查询客户名称
        dCriteria.add(Restrictions.like("custName","%"+customer.getCustName()
+"%"));
        }
        //判断是否提供了客户行业
        if(StringUtils.isNotBlank(customer.getCustIndustry())){
            //添加条件：模糊查询客户行业
        dCriteria.add(Restrictions.like("custIndustry","%"+customer.getCustIn
dustry()+"%"));
        }
        //判断是否提供了客户来源
        if(StringUtils.isNotBlank(custSourceId)){
            //添加条件：精确查询客户来源
        dCriteria.add(Restrictions.eq("custSource.dictId", custSourceId));
        }
        //判断是否提供了客户来源
        if(StringUtils.isNotBlank(custLevelId)){
            //添加条件：精确查询客户来源
            dCriteria.add(Restrictions.eq("custLevel.dictId", custLevelId));
        }
        //3.查询所有客户
        customers = customerService.findAllCustomer(dCriteria);
        //4.查询所有客户来源和所有客户级别
        custSources = customerService.findAllCustomerSource();
        custLevels = customerService.findAllCustomerLevel();
        return "findAllCustomer";
    }
```

2.8.3 Service

```
    /**
     * 查询所有客户，带条件
     * @param dCriteria 查询条件
```

```
     * @return
     */
    List<Customer> findAllCustomer(DetachedCriteria dCriteria);

    @Override
    @Transactional(propagation=Propagation.SUPPORTS,readOnly=true)
    public Page findAllCustomer(DetachedCriteria dCriteria) {
        return customerDao.findAllCustomer(dCriteria);
    }
```

2.8.4 Dao

```
    /**
     * 查询所有客户
     * @param dCriteria 查询条件
     * @return
     */
    List<Customer> findAllCustomer(DetachedCriteria dCriteria);
```