CRM 第三天

第1章 客户和联系人的关系映射

1.1 客户和联系人之间的关系分析

在 **hibernate** 课程我们学习一对多关系映射时，已经明确过客户和联系人的关系，今天我们再来回顾一下。

客户：即一家公司。

联系人：即该公司的员工。

（排除兼职的情况）

一家公司可以有多名员工。多名员工可以属于同一家公司。

所以，我们认定客户和联系人之间的关系是一对多。

1.2 客户和联系人的实体映射配置

```
    /**
     * 客户的实体类加入下面代码
     */
    @OneToMany(targetEntity=LinkMan.class,mappedBy="customer",cascade=CascadeType.ALL)
    private Set<LinkMan> linkmans = new HashSet<LinkMan>(0);

    public Set<LinkMan> getLinkmans() {
        return linkmans;
    }
    public void setLinkmans(Set<LinkMan> linkmans) {
        this.linkmans = linkmans;
    }

    /**
     * 联系人的实体类（数据模型）
     *
     */
    @Entity
    @Table(name="cst_linkman")
    public class LinkMan implements Serializable {

        @Id
```

```
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="lkm_id")
    private Long lkmId;
    @Column(name="lkm_name")
    private String lkmName;
    @Column(name="lkm_gender")
    private String lkmGender;
    @Column(name="lkm_phone")
    private String lkmPhone;
    @Column(name="lkm_mobile")
    private String lkmMobile;
    @Column(name="lkm_email")
    private String lkmEmail;
    @Column(name="lkm_position")
    private String lkmPosition;
    @Column(name="lkm_memo")
    private String lkmMemo;

    //多对一关系映射：多个联系人对应客户
    @ManyToOne(targetEntity=Customer.class)
    @JoinColumn(name="lkm_cust_id",referencedColumnName="cust_id")
    private Customer customer;//用它的主键，对应联系人表中的外键

    public Long getLkmId() {
        return lkmId;
    }
    public void setLkmId(Long lkmId) {
        this.lkmId = lkmId;
    }
    public String getLkmName() {
        return lkmName;
    }
    public void setLkmName(String lkmName) {
        this.lkmName = lkmName;
    }
    public String getLkmGender() {
        return lkmGender;
    }
    public void setLkmGender(String lkmGender) {
        this.lkmGender = lkmGender;
    }
    public String getLkmPhone() {
        return lkmPhone;
    }
```

```
        public void setLkmPhone(String lkmPhone) {
            this.lkmPhone = lkmPhone;
        }
        public String getLkmMobile() {
            return lkmMobile;
        }
        public void setLkmMobile(String lkmMobile) {
            this.lkmMobile = lkmMobile;
        }
        public String getLkmEmail() {
            return lkmEmail;
        }
        public void setLkmEmail(String lkmEmail) {
            this.lkmEmail = lkmEmail;
        }
        public String getLkmPosition() {
            return lkmPosition;
        }
        public void setLkmPosition(String lkmPosition) {
            this.lkmPosition = lkmPosition;
        }
        public String getLkmMemo() {
            return lkmMemo;
        }
        public void setLkmMemo(String lkmMemo) {
            this.lkmMemo = lkmMemo;
        }
        public Customer getCustomer() {
            return customer;
        }
        public void setCustomer(Customer customer) {
            this.customer = customer;
        }
        @Override
        public String toString() {
            return "LinkMan [lkmId=" + lkmId + ", lkmName=" + lkmName + ",
lkmGender=" + lkmGender + ", lkmPhone="
                    + lkmPhone + ", lkmMobile=" + lkmMobile + ", lkmEmail=" +
lkmEmail + ", lkmPosition=" + lkmPosition
                    + ", lkmMemo=" + lkmMemo + "]";
        }
    }
```

第2章 联系人的增删改查操作

2.1 写在最前

本章节提供的是联系人增删改查的代码实现。
下面出现的 Action 指的的是：LinkManAction
出现的 Service 指的是：ILinkManService 和 LinkManServiceImpl
出现的 Dao 指的是：LinkManDao 和 LinkManDaoImpl。
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

```
/**
 * 联系人的动作类
 *
 *
 */
@Controller("linkmanAction")
@Scope("prototype")
@ParentPackage("struts-default")
@Namespace("/linkman")
public class LinkManAction extends ActionSupport implements
ModelDriven<LinkMan> {

    private LinkMan linkman = new LinkMan();

    @Autowired
    private ICustomerService customerService;
    @Autowired
    private ILinkManService linkmanService;

    @Override
    public LinkMan getModel() {
        return linkman;
    }
}
```

Service

```
/**
 * 联系人的业务层接口
 *
```

```
 */
public interface ILinkManService {
}

/**
 * 联系人的业务层实现类
 *
 */
@Service("linkmanService")
@Transactional(propagation=Propagation.SUPPORTS,readOnly=true)
public class LinkManServiceImpl implements ILinkManService {

    @Autowired
    private ILinkManDao linkmanDao;
}
```

Dao

```
/**
 * 联系人的持久层接口
 *
 */
public interface ILinkManDao {
}

/**
 * 联系人的持久层实现类
 *
 */
@Repository("linkmanDao")
public class LinkManDaoImpl implements ILinkManDao {
    @Autowired
    private HibernateTemplate hibernateTemplate;
}
```

2.2 显示添加联系人页面

2.2.1 menu.jsp 页面

```
<A  href="${pageContext.request.contextPath}/linkman/addUILinkMan.action"
    class=style2 target=main>
    — 新增联系人
</A>
```

2.2.2 Action

```
    /**
     * 获取添加联系人页面
     * @return
     */
    private List<Customer> customers;
    @Action(value="addUILinkMan",results={
    @Result(name="addUILinkMan",type="dispatcher",location="/jsp/linkman/add.
jsp")
    })
    public String addUILinkMan(){
        //1.查询所有客户
        customers = customerService.findAllCustomer();
        return "addUILinkMan";
    }

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

2.3保存联系人

2.3.1 add.jsp

```
    <s:form action="addLinkMan" namespace="/linkman">
        <TABLE cellSpacing=0 cellPadding=5  border=0>
            <tr>
                <td>所属客户：</td>
                <td colspan="3">
                    <s:select      list="customers"      name="customer.custId"
listKey="custId" listValue="custName" headerKey="" headerValue="---请选择---"
class="textbox" id="sChannel2" style="WIDTH: 180px" maxLength="50"/>
                </td>
            </tr>
            <TR>
                <td>联系人名称：</td>
                <td>
                    <s:textfield name="lkmName" class="textbox" id="sChannel2"
```

```
style="WIDTH: 180px" maxLength="50"/>
                </td>
                <td>联系人性别：</td>
                <td>
                    <s:radio    list="#{'male':'   男   ','female':'   女   '}"
name="lkmGender"/ >
                </td>
            </TR>
            <TR>
                <td>联系人办公电话 ：</td>
                <td>
                    <s:textfield name="lkmPhone" class="textbox" id="sChannel2"
style="WIDTH: 180px" maxLength="50"/>
                </td>
                <td>联系人手机 ：</td>
                <td>
                    <s:textfield       name="lkmMobile"       class="textbox"
id="sChannel2" style="WIDTH: 180px" maxLength="50"/>
                </td>
            </TR>
            <TR>
                <td>联系人邮箱 ：</td>
                <td>
                    <s:textfield name="lkmEmail" class="textbox" id="sChannel2"
style="WIDTH: 180px" maxLength="50"/>
                </td>
                <td>联系人职位 ：</td>
                <td>
                    <s:textfield      name="lkmPosition"      class="textbox"
id="sChannel2" style="WIDTH: 180px" maxLength="50"/>
                </td>
            </TR>
            <TR>
                <td>联系人简介 ：</td>
                <td colspan="2">
                    <s:textarea name="lkmMemo" rows="5" cols="27"></s:textarea>
                </td>
            </TR>
            <tr>
                <td rowspan=2>
                    <s:submit value="保存"></s:submit>
                </td>
            </tr>
        </TABLE>
```

```
    </s:form>
```

2.3.2 Action

```
    /**
     * 保存联系人
     * @return
     */
    @Action(value="addLinkMan",results={
        @Result(name="addLinkMan",type="redirect",location="/jsp/success.jsp"
)
    })
    public String addLinkMan(){
        linkmanService.saveLinkMan(linkman);
        return "addLinkMan";
    }
```

2.3.3 Service

```
    /**
     * 保存联系人
     * @param linkman
     */
    void saveLinkMan(LinkMan linkman);

    @Override
    @Transactional(propagation=Propagation.REQUIRED,readOnly=false)
    public void saveLinkMan(LinkMan linkman) {
        linkmanDao.save(linkman);
    }
```

2.3.4 Dao

```
    /**
     * 保存联系人
     * @param linkman
     */
    void save(Linkman linkman);

    @Override
    public void save(Linkman linkman) {
        hibernateTemplate.save(linkman);
```

```
    }
```

2.4联系人列表展示

2.4.1 menu.jsp

```
    <A
href="${pageContext.request.contextPath}/linkman/findAllLinkMan.action"
        class=style2 target=main>
        — 联系人列表
    </A>
```

2.4.2 Action

```
    private List<LinkMan> linkmans;
    @Action(value="findAllLinkMan",results={
        @Result(name="findAllLinkMan",type="dispatcher",location="/jsp/linkma
n/list.jsp")
    })
    public String findAllLinkMan(){
        //1.查询所有联系人，使用离线对象
        linkmans = linkmanService.findAllLinkMan(dCriteria);
        return "findAllLinkMan";
    }

    public List<LinkMan> getLinkmans() {
        return linkmans;
    }
    public void setLinkmans(List<LinkMan> linkmans) {
        this.linkmans = linkmans;
    }
```

2.4.3 Service

```
    /**
     * 查询所有联系人
     * @param dCriteria 查询条件
     * @return
     */
    List<LinkMan> findAllLinkMan(DetachedCriteria dCriteria);
```

```
    @Override
    public List<LinkMan> findAllLinkMan(DetachedCriteria dCriteria) {
        return linkmanDao.findAllLinkMan(dCriteria);
    }
```

2.4.4 Dao

```
    /**
     * 查询所有联系人
     * @param dCriteria  查询条件
     * @return
     */
    List<LinkMan> findAllLinkMan(DetachedCriteria dCriteria);

    @Override
    public List<LinkMan> findAllLinkMan(DetachedCriteria dCriteria) {
        return (List<LinkMan>)
getHibernateTemplate().findByCriteria(dCriteria);
    }
```

2.5联系人删除

2.5.1 list.jsp

```
    <s:a href="javascript:delOne('%{lkmId}')">删除</s:a>
    <SCRIPT language=javascript>
        function delOne(lkmId){
            var sure = window.confirm("确定删除吗？");
            if(sure){
        window.location.href="${pageContext.request.contextPath}/linkman/remo
veLinkMan.action?lkmId="+lkmId;
            }
        }
    </SCRIPT>
```

2.5.2 Action

```
    /**
     * 删除联系人
     * @return
     */
    @Action(value="removeLinkMan",results={
```

```
        @Result(name="removeLinkMan",type="redirect",location="/jsp/success.j
sp")
    })
    public String removeLinkMan(){
        linkmanService.removeLinkMan(linkman);
        return "removeLinkMan";
    }
```

2.5.3 Service

```
    /**
     * 删除联系人
     * @param linkman
     */
    void removeLinkMan(LinkMan linkman);

    @Override
    @Transactional(propagation=Propagation.REQUIRED,readOnly=false)
    public void removeLinkMan(LinkMan linkman) {
        linkmanDao.delete(linkman);
    }
```

2.5.4 Dao

```
    /**
     * 删除联系人
     * @param linkman
     */
    void delete(Linkman linkman);


    @Override
    public void delete(Linkman linkman) {
        hibernateTemplate.delete(linkman);
    }
```

2.6显示联系人修改页面

2.6.1 list.jsp

```
    <s:a action="editUILinkMan" namespace="/linkman">
```

```
        <s:param name="lkmId" value="%{lkmId}"></s:param>
        修改
    </s:a>
```

2.6.2 Action

```
    /**
     * 获取编辑联系人页面
     * @return
     */
    @Action(value="editUILinkMan",results={
        @Result(name="editUILinkMan",type="dispatcher",location="/jsp/linkman
/edit.jsp")
    })
    public String editUILinkMan(){
        //1.根据 id 查询联系人信息
        LinkMan                          dbLinkMan                          =
linkmanService.findLinkManById(linkman.getLkmId());
        //2.把查询出来的联系人压栈
        ValueStack vs = ActionContext.getContext().getValueStack();
        vs.push(dbLinkMan);
        //3.查询所有客户
        customers = customerService.findAllCustomer();
        return "editUILinkMan";
    }
```

2.6.3 Service

```
    /**
     * 根据 id 查询联系人信息
     * @param lkmId
     * @return
     */
    LinkMan findLinkManById(Long lkmId);

    @Override
    public LinkMan findLinkManById(Long lkmId) {
        return linkmanDao.findById(lkmId);
    }
```

2.6.4 Dao

```
/**
 * 根据id查询联系人
 * @param lkmId
 */
Linkman findById(Long lkmId);



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

2.7联系人修改

2.7.1 edit.jsp

```
<s:form action="editLinkMan" namespace="/linkman">
    <s:hidden name="lkmId" value="%{lkmId}"></s:hidden>
    <TABLE cellSpacing=0 cellPadding=5 border=0>
        <tr>
            <td>所属客户：</td>
            <td colspan="3">
                <s:select list="customers" name="customer.custId"
listKey="custId" listValue="custName" headerKey="" headerValue="---请选择---"
class="textbox" id="sChannel2" style="WIDTH: 180px" maxLength="50"/>
            </td>
        </tr>
        <TR>
            <td>联系人名称：</td>
            <td>
                <s:textfield name="lkmName" class="textbox" id="sChannel2"
style="WIDTH: 180px" maxLength="50"/>
            </td>
            <td>联系人性别：</td>
            <td>
                <s:radio list="#{'male':'男','female':'女'}"
name="lkmGender"/ >
            </td>
        </TR>
        <TR>
```

```
                <td>联系人办公电话 ： </td>
                <td>
                    <s:textfield name="lkmPhone" class="textbox" id="sChannel2"
style="WIDTH: 180px" maxLength="50"/>
                </td>
                <td>联系人手机 ： </td>
                <td>
                    <s:textfield       name="lkmMobile"       class="textbox"
id="sChannel2" style="WIDTH: 180px" maxLength="50"/>
                </td>
            </TR>
            <TR>
                <td>联系人邮箱 ： </td>
                <td>
                    <s:textfield name="lkmEmail" class="textbox" id="sChannel2"
style="WIDTH: 180px" maxLength="50"/>
                </td>
                <td>联系人职位 ： </td>
                <td>
                    <s:textfield      name="lkmPosition"      class="textbox"
id="sChannel2" style="WIDTH: 180px" maxLength="50"/>
                </td>
            </TR>
            <TR>
                <td>联系人简介 ： </td>
                <td colspan="2">
                    <s:textarea name="lkmMemo" rows="5" cols="27"></s:textarea>
                </td>
            </TR>
            <tr>
                <td rowspan=2>
                    <s:submit value="保存"></s:submit>
                </td>
            </tr>
        </TABLE>
    </s:form>
```

2.7.2 Action

```
    /**
     * 编辑联系人
     */
    @Action(value="editLinkMan",results={
        @Result(name="editLinkMan",type="redirect",location="/jsp/success.jsp
```

```
")
    })
    public String editLinkMan(){
        linkmanService.updateLinkMan(linkman);
        return "editLinkMan";
    }
```

2.7.3 Service

```
    /**
     * 编辑联系人
     * @param linkman
     */
    void updateLinkMan(LinkMan linkman);

    @Override
    public void updateLinkMan(LinkMan linkman) {
        linkmanDao.update(linkman);
    }
```

2.7.4 Dao

```
    /**
     * 更新联系人
     * @param linkman
     */
    void update(Linkman linkman);


    @Override
    public void update(Linkman linkman) {
        hibernateTemplate.update(linkman);
    }
```

2.8 联系人的列表条件查询

2.8.1 list.jsp

2.8.2 Action

2.8.3 Service

2.8.4 Dao

第3章 客户联系人的分页实现

3.1 分页的意义

分页的意义：

我们都知道，查询某个列表数据时，有可能会有很多数据。如果一次查询出很大的结果集，对执行效率上有影响，同时对内存的开销也很大。这时候我们就要将大结果集拆分成小结果集，分批次查询显示出来。

分页的分类：

基于内存分页：减少与数据库的交互次数，没有解决内存开销问题。

基于数据库分页：增加与数据库的交互，解决内存开销问题。依赖的是数据库的分页语句。（不同的数据库，分页的语句是不同的）

3.2 实现分页的 3 个必要条件

在介绍这三个必要条件之前，我们得先明确几个问题。

第一个： mysql 数据库分页的关键字：

```
limit  startindex , resultcount
```

startindex：开始记录的索引，注意第一条记录的索引为 0。

resultcount：一次查询多少条记录。（也就是每页要显示的条数，这个值一般是固定的）

第二个：开始记录索引的计算公式：

```
开始记录索引 = （当前页-1） * 每页显示的条数
```

第三个：总共有多少页

```
        总页数 = 总记录数条数%每页显示的条数 == 0
                                ? 总记录数条数%每页显示的条数
                                : 总记录数条数%每页显示的条数+1
```

总结：

通过上面的介绍，我们发现每页显示的条数是一个固定值，当前页需要由使用者提供，而总记录条数需要我们自己去数据库中查询。这三个信息就构成了分页的 3 个必要条件。

3.3 分页的代码实现

3.3.1 抽取 Page 类

```
/**
 * 用于封装分页的对象
 *   要想实现分页，必不可少的三个条件：
 *        1、客户要看哪一页
 *        2、每页显示的条数
 *        3、总记录条数
 *
 */
public class Page implements Serializable {

    private int currentPageNum;//当前页      *
    private int pageSize = 5;//每页显示的条数*
    private int totalRecords;//总记录条数        *
    private int totalPageNum;//总页数            *
    private int startIndex;//查询的开始记录索引   *
    private int prePageNum;//上一页          *
    private int nextPageNum;//下一页         *
    private List records;//带有分页的结果集

    //要想使用此 Page 类，需要提供两个参数
    public Page(int currentPageNum,int totalRecords){
        this.currentPageNum = currentPageNum;
        this.totalRecords = totalRecords;

        //总页数
        totalPageNum = totalRecords%pageSize == 0
                                ? totalRecords/pageSize
                                : totalRecords/pageSize+1;

        //开始记录索引
        startIndex = (currentPageNum-1)*pageSize;
```

```
    }

    public int getPrePageNum() {//计算上一页
        prePageNum = currentPageNum - 1;
        if(prePageNum < 1){
            prePageNum = 1;
        }
        return prePageNum;
    }

    public int getNextPageNum() {//计算下一页
        nextPageNum = currentPageNum + 1;
        if(nextPageNum > totalPageNum){
            nextPageNum = totalPageNum;
        }
        return nextPageNum;
    }

    public int getCurrentPageNum() {
        return currentPageNum;
    }
    public void setCurrentPageNum(int currentPageNum) {
        this.currentPageNum = currentPageNum;
    }
    public int getPageSize() {
        return pageSize;
    }
    public void setPageSize(int pageSize) {
        this.pageSize = pageSize;
    }
    public int getTotalRecords() {
        return totalRecords;
    }
    public void setTotalRecords(int totalRecords) {
        this.totalRecords = totalRecords;
    }
    public int getTotalPageNum() {
        return totalPageNum;
    }
    public void setTotalPageNum(int totalPageNum) {
        this.totalPageNum = totalPageNum;
    }
    public int getStartIndex() {
        return startIndex;
```

```
        }
        public void setStartIndex(int startIndex) {
            this.startIndex = startIndex;
        }
        public List getRecords() {
            return records;
        }
        public void setRecords(List records) {
            this.records = records;
        }
        public void setPrePageNum(int prePageNum) {
            this.prePageNum = prePageNum;
        }
        public void setNextPageNum(int nextPageNum) {
            this.nextPageNum = nextPageNum;
        }
    }
```

3.3.2 改造持久层接口和实现类

```
    客户的持久层接口
    /**
     * 查询总记录条数
     * @param dCriteria 查询的条件
     * @return
     */
    int findTotalRecords(DetachedCriteria dCriteria);

    /**
     * 查询所有客户
     * @param dCriteria 查询条件
     * @param firstResult   查询的开始记录索引
     * @param maxResults    查询的条数
     * @return
     */
    List<Customer> findAllCustomer(DetachedCriteria dCriteria,
                                   int             firstResult,int
maxResults);
    }

    客户的持久层实现类
    @Override
    public int findTotalRecords(DetachedCriteria dCriteria) {
        dCriteria.setProjection(Projections.count("custId"));
```

```
        List<Long>              list              =              (List<Long>)
hibernateTemplate.findByCriteria(dCriteria);
        return list.isEmpty()?0:list.get(0).intValue();
    }

    @Override
    public List<Customer> findAllCustomer(DetachedCriteria dCriteria,
                                          int firstResult, int maxResults)
{
        dCriteria.setProjection(null);//清空之前的设置
        return (List<Customer>) hibernateTemplate.findByCriteria(dCriteria,
                                                  firstResult,
maxResults);
    }
```

3.3.3 改造 Service 接口和实现类

```
    客户的业务层接口
    /**
     * 查询所有客户，带分页
     * @param dCriteria 查询条件
     * @param num        当前客户要看的页码
     * @return  page    封装好的分页对象
     */
    Page findAllCustomer(DetachedCriteria dCriteria,String num);

    客户的业务层实现类
    @Override
    @Transactional(propagation=Propagation.SUPPORTS,readOnly=true)
    public Page findAllCustomer(DetachedCriteria dCriteria, String num) {
        //1.判断用户是否提供了页码
        int currentPageNum = 1;//默认页码
        if(StringUtils.isNotBlank(num)){
            currentPageNum = Integer.parseInt(num);
        }
        //2.查询总记录条数
        int totalRecords = customerDao.findTotalRecords(dCriteria);//此处有个小
问题
        //3.创建 page 对象
        Page page = new Page(currentPageNum,totalRecords);
        //4.使用 page 对象中的数据，查询带有分页的结果集
        List<Customer> records = customerDao.findAllCustomer(dCriteria,
                                        page.getStartIndex(),
page.getPageSize());
```

```
        //5.把查询出来的结果集存入 page 中
        page.setRecords(records);
        //6.返回 page 对象
        return page;
    }
```

3.3.4 改造 Action 和 jsp

```
    客户的动作类
    /**
     * 查询所有客户
     * @return
     */
    private Page page;
    private String num;

    @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);
        //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.getCustIndustry()+"%"));
        }
        //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
        page = customerService.findAllCustomer(dCriteria,num);
        //4.查询所有客户来源和所有客户级别
        custSources = customerService.findAllCustomerSource();
        custLevels = customerService.findAllCustomerLevel();
        return "findAllCustomer";
    }

    public Page getPage() {
        return page;
    }
    public void setPage(Page page) {
        this.page = page;
    }
    public String getNum() {
        return num;
    }
    public void setNum(String num) {
        this.num = num;
    }
```

客户的列表页面

```
<script type="text/javascript">
    function topage(num){
        document.getElementById("pagenum").value = num;
        document.forms[0].submit();
    }
</script>
<style>
    div{
        margin-top:15px;
    }
    div a{
        border:1px solid lightblue;
        padding: 3px 3px 3px 3px;
        border-radius:25%;
    }
    a:active{
        border-left:2px solid lightblue;
        border-right:1px solid white;
        border-top:2px solid lightblue;
```

```
            border-bottom:1px solid white;
        }
    </style>
    <div id="pageDiv">
        <a href="javascript:topage('1')">首页</a>
        <a href="javascript:topage('${page.prePageNum}')">上一页</a>
        <a href="javascript:topage('${page.nextPageNum}')">下一页</a>
        <a href="javascript:topage('${page.totalPageNum}')">末页</a>
              
        共${page.totalPageNum}页/第${page.currentPageNum}页
    </div>
```

3.4分页的完善

3.4.1 添加 jsp 页面的页码按钮

在 Page 类中加入：

```
    //用于显示页号的属性。需求：只显示 9 个页号
    private int beginPageNum;//开始页号
    private int endPageNum;//结束页号

    public int getBeginPageNum() {
        return beginPageNum;
    }
    public void setBeginPageNum(int beginPageNum) {
        this.beginPageNum = beginPageNum;
    }
    public int getEndPageNum() {
        return endPageNum;
    }
    public void setEndPageNum(int endPageNum) {
        this.endPageNum = endPageNum;
    }
```

在 Page 类的构造函数中加入：

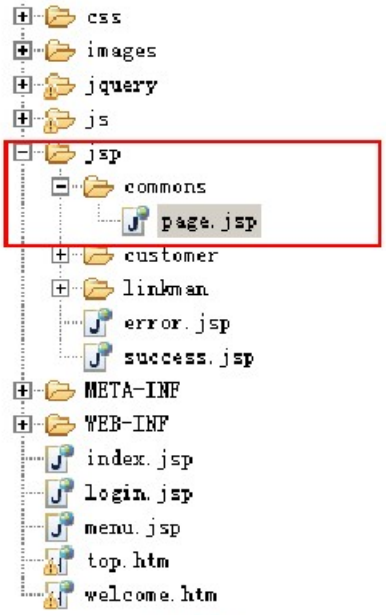
```
    if(totalPageNum < 9){//总页数不够 9 页
        beginPageNum = 1;
        endPageNum = totalPageNum;
    }else{//总页数超过了 9 页
        beginPageNum = currentPageNum - 4;
```

```
            endPageNum = currentPageNum + 4;
            if(beginPageNum < 1){
                beginPageNum = 1;
                endPageNum = beginPageNum + 8;
            }
            if(endPageNum > totalPageNum){
                endPageNum = totalPageNum;
                beginPageNum = endPageNum - 8;
            }
        }
```

3.4.2 改造 list.jsp 显示页号

```
    <div id="pageDiv">
        <a href="javascript:topage('1')">首页</a>
        <a href="javascript:topage('${page.prePageNum}')">上一页</a>
        <s:iterator   begin="%{page.beginPageNum}"   end="%{page.endPageNum}"
var="snum">
            <a href="javascript:topage('${snum}')">${snum}</a>
        </s:iterator>
        <a href="javascript:topage('${page.nextPageNum}')">下一页</a>
        <a href="javascript:topage('${page.totalPageNum}')">末页</a>
              
        共${page.totalPageNum}页/第${page.currentPageNum}页
    </div>
```

3.4.3 抽取 page.jsp



page.jsp 中的代码

```
<%@page language="java"
        contentType="text/html;charset=UTF-8"
        pageEncoding="UTF-8"%>
<script type="text/javascript">
    function topage(num){
        document.getElementById("pagenum").value = num;
        document.forms[0].submit();
    }
</script>
<style>
    div{
        margin-top:15px;
    }
    div a{
        border:1px solid lightblue;
        padding: 3px 3px 3px 3px;
        border-radius:25%;
    }
    a:active{
        border-left:2px solid lightblue;
        border-right:1px solid white;
        border-top:2px solid lightblue;
        border-bottom:1px solid white;
    }
</style>
<div id="pageDiv">
```

```
        <a href="javascript:topage('1')">首页</a>
        <a href="javascript:topage('${page.prePageNum}')">上一页</a>
        <s:iterator   begin="%{page.beginPageNum}"   end="%{page.endPageNum}"
var="snum">
            <a href="javascript:topage('${snum}')">${snum}</a>
        </s:iterator>
        <a href="javascript:topage('${page.nextPageNum}')">下一页</a>
        <a href="javascript:topage('${page.totalPageNum}')">末页</a>
              
        共${page.totalPageNum}页/第${page.currentPageNum}页
    </div>

    list.jsp 中的内容
    <TD align="center">
        <%--分页的开始 --%>
        <%@ include file="/jsp/commons/page.jsp" %>
        <%--分页的结束 --%>
    </TD>
```