国际物流云商系统第七天

一．回顾

邮件发送接收原理

使用 JavaMail 实现邮件发送

Spring+JavaMail 整合实现邮件发送功能

HttpClient 的 GET 与 POS 请求处理

使用 HttpClient 调用接口发送手机短信验证码

二．购销合同模块业务分析

1.购销合同业务介绍

公司销售和海外客户签订订单（合同），客户订单中的货物，公司就联系这些（多个）厂家来生产，和生产厂家签订合同，这个合同就叫“购销合同”。购销合同的内容主要由三部分组成，购销合同的主信息，和多个货物的信息，和多个附件的信息。（附件实际就是货物）
购销合同打印出后，将每张纸交给对应生产厂家的销售代表（公司）
购销合同打印要求，可以每页只打印一款货物，也可以打印两款货物。用户可以自己选择。如果每页打印两款货物，必须是同一个生产厂家，如果不是同一个生产厂家，必须另起一页。
在需求调研时，客户给我们提供纸质资料，电子资料（doc,excel）

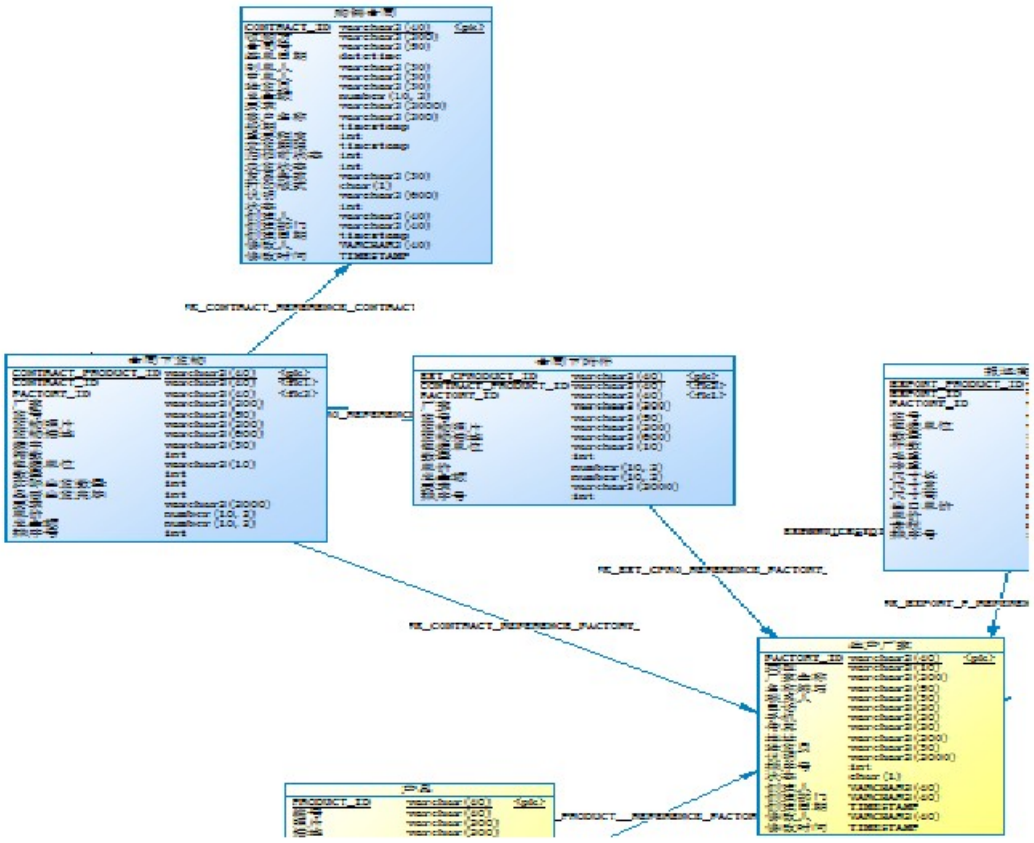
SHAANXI
HEIMA INTERNATIONAL
北京市昌平区建材城西路金燕龙办公楼一层 CO., LTD.

购 销 合 同

收 购 方：黑马商务发展有限公司	生产工厂：宏艺厂
合 同 号：12HEIMA081	联 系 人：李诚
签单日期：2012年9月13日	电 话：054-31899

产品	★★★ 货物描述	数量		单价	总金额
	全用料红酒杯 口部箍1CM宽白金边 直径7X26CM高 4只/五层内盒 16只/大箱 产品白纸,瓦楞纸，汽泡纸包裹入内盒， 内盒， 大箱用胶带纸工字封口 大箱尺码：44X44X30CM 如有不符请提前通知	400	只	¥10.00	¥4,000.00
HEIMA1050152/11306					

2. 购销合同的表结构



二．数据库设计的三大范式

1.数据库设计三大范式

第一范式 1NF：做到每列不可拆分

第二范式 2NF：确保一个表只做一件事情

第三范式 3NF：在满足 2NF，消除表中的传递依赖

Amstrong 推理规则： A--→B ， B--→C 则 A--→C ，这种情况如果存在，就要想办法消除这种关系 ，它的目标是为了空间最省。

2.数据库设计的反三范式

反三范式：在进行数据库设计时，首先确保你所设计的表结构都达到第三范式的要求，但是实际开发时，为了满足一些用户的特殊需求，开发时又会在符合第三范式的表中，加入冗余字段，结果就导致又不符合第三范式要求了，这种情况下，咱们把它称为反三范式。

因为存在冗余字段，所以就要去维护它，如购销合同中的合同总金额就是一个冗余字段，

就需要在平时添加货物时，添加附件时，都需要去更新购销合同中的总金额。
思考：购销合同模块的表结构设计是否使用了三范式及反三范式，它们都用在哪些地方。

三.分散计算思想

如果没有分散计算，在查询购销合同总金额时，就需要关联加载购销合同下的所有货物，并要加载所有货物下的所有附件，这样可能实现购销合同总金额查询消耗时间太多，而用户等不起，此时就可以在平时添加货物时，添加附件时，分别计算出货物总金额，附件总金额 ，再更新购销合同总金额 ，这样就相当于将一次集中计算的工作量分散到平时的多次计算过程中，所以查询购销合同总金额时速度就会很快。

早期系统上线，表现良好，反应速度很快，随着系统使用时间的增长

缺点:就是代码的编写量和维护工作量急剧上升

优点：提高了页面数据的检索速度

四．实现购销合同的 CRUD

1.购销合同模块的实体类

购销合同 Contract 类

```
@Entity
@Table(name="CONTRACT_C")
@DynamicInsert(true)
@DynamicUpdate(true)
public class Contract extends BaseEntity {
    @Id
    @Column(name="CONTRACT_ID")
    @GeneratedValue(generator="system-uuid")
    @GenericGenerator(name="system-uuid",strategy="uuid")
    private String id;//购销合同id

    @OneToMany(mappedBy="contract")
    private Set<ContractProduct> contractProducts = new HashSet<ContractProduct>(0);
```

货物 ContractProduct 类

```
@Entity
@Table(name="CONTRACT_PRODUCT_C")
public class ContractProduct implements Serializable{
    @Id
    @Column(name="CONTRACT_PRODUCT_ID")
    @GeneratedValue(generator="system-uuid")
    @GenericGenerator(name="system-uuid",strategy="uuid")
    private String id;

    @ManyToOne
    @JoinColumn(name="CONTRACT_ID")
    private Contract contract = new Contract();//购销合同对象  货物与合同  多对一

    @ManyToOne
    @JoinColumn(name="FACTORY_ID")
    private Factory factory = new Factory();//工厂对象  多对一

    @OneToMany(mappedBy="contractProduct")
    private Set<ExtCproduct> extCproducts = new HashSet<ExtCproduct>(0);//货物与附件  一对多
```

附件 ExtCproduct 类

```
@Entity
@Table(name="EXT_CPRODUCT_C")
public class ExtCproduct implements Serializable{
    @Id
    @Column(name="EXT_CPRODUCT_ID")
    @GeneratedValue(generator="system-uuid")
    @GenericGenerator(name="system-uuid",strategy="uuid")
    private String id;

    @ManyToOne
    @JoinColumn(name="CONTRACT_PRODUCT_ID")
    private ContractProduct contractProduct = new ContractProduct();//附件与货物 多对一

    @ManyToOne
    @JoinColumn(name="FACTORY_ID")
    private Factory factory;//附件与工厂  多对一
```

工厂 Factory 类

```
@Entity
@Table(name="FACTORY_C")
public class Factory extends BaseEntity {
    @Id
    @Column(name="FACTORY_ID")
    @GeneratedValue(generator="system-uuid")
    @GenericGenerator(name="system-uuid",strategy="uuid")
    private String id;

    @Column(name="CTYPE")
    private String ctype;//类型
    @Column(name="FULL_NAME")
    private String fullName;//厂家全称
```

2.购销合同的 DAO

```
public interface ContractDao extends BaseDao<Contract> {

}

public interface ContractProductDao extends BaseDao<ContractProduct> {

}

public interface ExtCproductDao extends BaseDao<ExtCproduct> {

}

public interface FactoryDao extends BaseDao<Factory> {

}
```

3.购销合同的 Service

```
@Service
public class ContractServiceImpl extends BaseServiceImpl<Contract> implements ContractService {
    private ContractDao contractDao;

    @Autowired
    public void init(ContractDao contractDao){
        super.baseDao = contractDao;
        this.contractDao = contractDao;
    }

    public void saveOrUpdate(Contract entity) {
        if(UtilFuns.isEmpty(entity.getId())){
            entity.setState(0);//0草稿 1已上报
            entity.setTotalAmount(0d);
        }
        contractDao.save(entity);
    }
```

4.购销合同的 Action

```
@Namespace("/cargo")
@Controller
@ParentPackage("default")
@Scope("prototype")
@Results({ @Result(name = "alist", type = "redirectAction", location = "contractAction_list") })
public class ContractAction extends BaseAction implements ModelDriven<Contract> {
    private Contract model = new Contract();

    public Contract getModel() {
        return model;
    }
```

分页查询

```
/**
 * 分页查询列表
 */
@Action(value = "contractAction_list", results = {
        @Result(name = "list", location = "/WEB-INF/pages/cargo/contract/jContractList.jsp") })
public String list() throws Exception {
    // 设置Pageable的取值
    Pageable pageable = new PageRequest(page.getPageNo() - 1, page.getPageSize());
    org.springframework.data.domain.Page<Contract> spage = contractService.findPage(null, pageable);

    // 实现分页组件的转换
    super.parsePage(page, spage);
    page.setUrl("contractAction_list");

    // page压入到值栈中
    super.push(page);

    return "list";
}
```

查看详情

```
/**
 * 查看详情
 */
@Action(value = "contractAction_toview", results = {
        @Result(name = "toview", location = "/WEB-INF/pages/cargo/contract/jContractView.jsp") })
public String toview() throws Exception {
    // 1.调用业务方法，查询一个部门
    Contract obj = contractService.get(model.getId());

    // 2.将obj压入值栈中
    super.push(obj);

    return "toview";
}
```

进入新增页面

```
/**
 * 进入新增页面
 */
@Action(value = "contractAction_tocreate", results = {
        @Result(name = "tocreate", location = "/WEB-INF/pages/cargo/contract/jContractCreate.jsp") })
public String tocreate() throws Exception {

    return "tocreate";
}
```

实现新增

```
/**
 * 实现新增
 */
@Action("contractAction_insert")
public String insert() throws Exception {
    // 调用业务方法，实现保存
    contractService.saveOrUpdate(model);
    return "alist";
}
```

进入修改页面

```
/**
 * 进入修改页面
 */
@Action(value = "contractAction_toupdate", results = {
        @Result(name = "toupdate", location = "/WEB-INF/pages/cargo/contract/jContractUpdate.jsp") })
public String toupdate() throws Exception {
    // 1.查询要修改的原有部门信息
    Contract obj = contractService.get(model.getId());

    // 2.将部门信息放入值栈中
    super.push(obj);

    return "toupdate";
}
```

实现修改操作

```
/**
 * 实现修改操作
 */
@Action("contractAction_update")
public String update() throws Exception {
    // 1.先从数据库中查询原有部门信息
    Contract obj = contractService.get(model.getId());
    // 2.更新要修改的属性
    obj.setCustomName(model.getCustomName());
    obj.setPrintStyle(model.getPrintStyle());
    obj.setContractNo(model.getContractNo());
    obj.setOffer(model.getOffer());
    obj.setInputBy(model.getInputBy());
    obj.setCheckBy(model.getCheckBy());
    obj.setInspector(model.getInspector());
    obj.setSigningDate(model.getSigningDate());
    obj.setImportNum(model.getImportNum());
    obj.setShipTime(model.getShipTime());
    obj.setTradeTerms(model.getTradeTerms());
    obj.setDeliveryPeriod(model.getDeliveryPeriod());
    obj.setCrequest(model.getCrequest());
    obj.setRemark(model.getRemark());
    // 3.调用业务方法，修改部门对象
    contractService.saveOrUpdate(obj);
    return "alist";
}
```

五．实现货物的 CRUD

六．附件的 CRUD(作业)

附件的相关操作与货物的相关相似，各位可以参考货物的实现思路。
关键代码的参考：
附件新增及修改的参考代码：

```
public void saveOrUpdate(ExtCproduct entity) {
    double amount = 0d;
    if(UtilFuns.isEmpty(entity.getId())){
        //新增
        if(UtilFuns.isNotEmpty(entity.getPrice())&& UtilFuns.isNotEmpty(entity.getCnumber())){
            amount= entity.getPrice()*entity.getCnumber();//货物总金额
            entity.setAmount(amount);
        }

        //修改购销合同的总金额
        Contract contract = baseDao.get(Contract.class, entity.getContractProduct().getContract().getId());//根据购销合同的id,得到购销合同对象
        contract.setTotalAmount(contract.getTotalAmount()+amount);

        //保存购销合同的总金额
        baseDao.saveOrUpdate(contract);

    }else{
        //修改
        double oldAmount = entity.getAmount();//取出货物的原有总金额
        if(UtilFuns.isNotEmpty(entity.getPrice())&& UtilFuns.isNotEmpty(entity.getCnumber())){
            amount= entity.getPrice()*entity.getCnumber();//货物总金额
            entity.setAmount(amount);
        }

        Contract contract = baseDao.get(Contract.class, entity.getContractProduct().getContract().getId());//根据购销合同的id,得到购销合同对象
        contract.setTotalAmount(contract.getTotalAmount()-oldAmount+amount);
    }
    baseDao.saveOrUpdate(entity);
}
```

附件删除的参考代码：

```
@Override
public void delete(Class<ExtCproduct> entityClass, ExtCproduct model) {
    ExtCproduct extCproduct = baseDao.get(ExtCproduct.class, model.getId());//得到附件对象

    Contract contract = baseDao.get(Contract.class, model.getContractProduct().getContract().getId());//得到购销合同对象

    //修改购销合同的总金额
    contract.setTotalAmount(contract.getTotalAmount()-extCproduct.getAmount());

    //保存总金额
    baseDao.saveOrUpdate(contract);

    //删除附件
    baseDao.deleteById(ExtCproduct.class,model.getId());

}
```

七．日历控件的补充

```
<script type="text/javascript" src="${ctx }/js/datepicker/WdatePicker.js""></script>
```


```
<td class="tableContent">
    <input type="text" style="width:90px;" name="deliveryPeriod"
     value=""
    onclick="WdatePicker({el:this,isShowOthers:true,dateFmt:'yyyy-MM-dd'});"/>
</td>
```

日历控件

八.下拉框的取值

```
<td class="columnTitle">生产厂家：</td>
<td class="tableContent">
     <s:select name="factory.id" list="factoryList"
               onchange="setFactoryName(this.options[this.selectedIndex].text);"
               listKey="id" listValue="factoryName"
               headerKey="" headerValue="--请选择--"/>
    <input type="hidden" id="factoryName" name="factoryName" value="${factoryName }"/>
</td>
```

九．相关问题的思考

实现购销合同查看

1.货物数/附件数

```
${o.contractProducts.size() }
/
<c:set var="extNo" value="0"></c:set>
<c:forEach items="${o.contractProducts }"  var="cp" >
    <c:if test="${cp.extCproducts.size() !=0 }">
        <c:set var="extNo" value="${extNo+cp.extCproducts.size() }"></c:set>
    </c:if>

</c:forEach>
${extNo }
```

上面的代码是可以优化的！

如何优化？

分散计算原理指导下，就可以在购销合同表中，加入两个冗余字段（货物数，附件数）

添加货物时就要同步更新购销合同的货物数，

添加附件时就要同步更新购销合同的附件数。