国际物流云商系统第十天

一.回顾

1.打断设计

所谓打断设计就是表中字段的冗余，把外键消除了，当存在一对多时，它是在一的一方加入多的一方的集合，这个集合用于存储多的主键，并且使用分隔符进行分隔（如用逗号分隔）

2.跳跃查询

跳跃查询是在打断设计基础上进行查询数据时的优化。它比传统的关联数据加载效率翻倍。

使用打断设计原则：
当关联级别数据加载的层级大于 4 层时，就必须考虑打断设计，否则加载数据速度过慢。

3.再次优化

数据搬家
数据搬家就是实现表级别的数据冗余，在该项目中用于当添加出口报运单时，也要实现报运单下的货物和附件的数据搬家，搬家的数据来源于购销合同的货物和附件。这样出口报运单就可以直接关联查询到它的货物和附件。

4.出口报运

当用户选择购销合同后，就可以进行出口报运，此时的出口报运单信息是不全面的，所以它的状态只能是一个草稿，其它信息在更新时再去完善。

5.百万数据的 POI

HSSFWorkbook:只能操作 excel2003
XSSFWorkbook:可以操作 excel 2007+ 虽然可以操作大量数据，但实际操作时会出问题，
问题原因是创建对象过多，而这些对象都在内存中，所以可能导致溢出。
SXSSFWorkbook:它是在 XSSFWorkbook 基础上进行优化，它的原理是首先设置一个内存中对象的数量值，默认为 100 个对象，当内存中所产生的对象数超出规定的限制时，就会将这些对象写入到临时的 xml 文件中，此时内存中的这些对象就可以销毁了，以后不断这样进行。

同时它也存在缺点：1.不能使用模板打印了。 2.在写磁盘过程中消耗的 IO 操作时间过多，会导致内存中又产生很多对象，但是原来的对象还没有完整写入磁盘中。

6.内存监视工具

Jdk 自带的 jvisualvm.exe 工具
通过这个工具就可以查看到 cpu 的使用情况，垃圾回收器的工作情况，堆内存使用情况。

二.实现动态更新出口报运单

1.进入更新页面

```
<script type="text/javascript" src="${ctx}/components/jquery-ui/jquery-1.2.6.js"></script>
<script type="text/javascript" src="${ctx}/js/tabledo.js"></script>
<script type="text/javascript" src="${ctx}/js/datepicker/WdatePicker.js"></script>
```

2.进行更新操作

劳保产品的申领
分组领取
劳保用品包括：60 个种类
班组：100 个班组

日用品	军帽	手套	大衣	牙膏
1 组	100			
2 组	20		30	

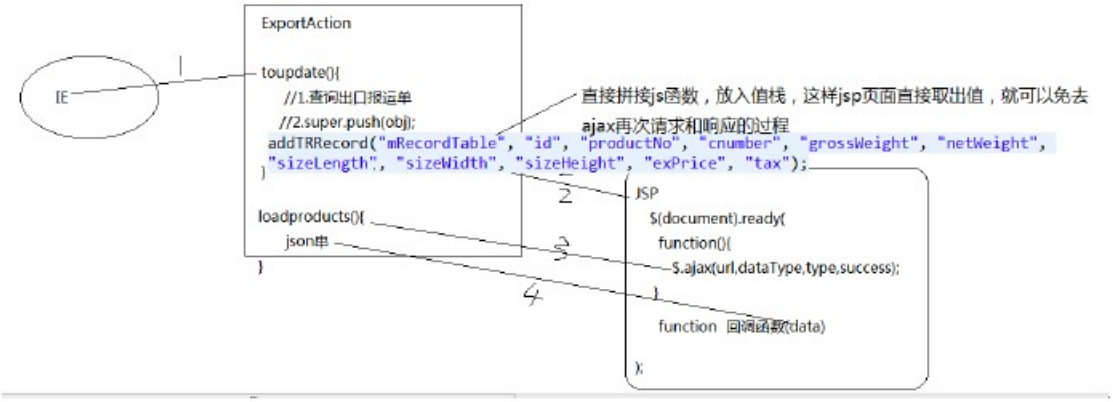
此时当领取完成后，就要实现数据的更新，结果反映更新速度相当慢。问题是有些组用品已领完，以后是不会更新的，这样如果每次也更新它，就浪费时间了。
解决办法：
1.在每个用品每组的对应的小格子中放入一个隐藏域，就可以放入 6000 个隐藏域，结果是处理速度反而越慢了。不考虑使用

2.在每组放入一个隐藏域，这样放入 100 个隐藏域（0 代表当前行的数据没有更新，1 代表当前行的数据有更新）

引用 tabledo.js 来实现出口报运单下的货物列表展示

```
<script type="text/javascript" src="${ctx}/components/jquery-ui/jquery-1.2.6.js"></script>
<script type="text/javascript" src="${ctx}/js/tabledo.js"></script>
```

此处不使用 ajax 方式实现处理
Ajax 请求需要经历 1,2,3,4 步骤，而传统方式只需要经历 1,2 步骤就可以了。



3.程序代码的实现

<input type=\"hidden\" name=\"mr_id\" value=\""+id+"\">代表货物编号
<input class=\"input\" type=\"hidden\" id=\"mr_changed\" name=\"mr_changed\">修改标志
<input type=\"text\" name=\"mr_cnumber\"　数量

想办法在 Action 中接收数据

三.读程

读程是一个程序员的基本功！

1.读程的方法

1. 通读（需要大量时间）
2. 带着问题去读代码（客户反馈一些 bug，带着一些学习上的问题去读程）

2.购销合同的读程相关问题

1）　是否可以使用模板？

否。由于用户的习惯，造成不能直接使用模板。

2）　数据和打印代码分离？

封装每页数据到 map 中，封装所有页到 list 中

```
//填写每页的内容，之后在循环每页读取打印
Map<String,String> pageMap = null;
List<Map> pageList = new ArrayList();　　//打印页
```

3）　图片（logo 图片和货物图片）如何插入到 excel 中？

工具类中直接封装，直接调用工具类方法

```
poiUtil.setPicture(wb, patriarch, path+"make/xlsprint/logo.jpg", curRow, 2, curRow+4, 2);
```

4） 如何实现画线？

```
poiUtil.setLine(wb, patriarch, curRow, 2, curRow, 8);   //draw line
```

5） 合并单元格画线？

合并单元格，只实现第一个单元格的画线，其他的单元格必须手工补。
通过直接创建这个单元格，设置画线的样式即可

6） 加人民币前缀？

```
public short rmb2Format(HSSFWorkbook wb) {
    HSSFDataFormat format = wb.createDataFormat();
    return format.getFormat("\"￥\"#,###,###.00"); //  设置格式
}
curStyle.setDataFormat(rmb2Format);
```

，代表千位符
#，代表数字，如果这个值为 0，它又在最前面，它的结果空
0，代表，当这位有值时，直接显示值，如果最后一位没有值时显示 0

7） 如何写公式？

```
普通单元格内容，直接写；
nCell.setCellType(HSSFCell.CELL_TYPE_FORMULA);
nCell.setCellFormula("F"+String.valueOf(curRow)+"*H"+String.valueOf(curRow));
```

8） 如何多行文本自动换行？

```
curStyle.setWrapText(true); //  换行
```

9） 如何实现单元格自适应高度？

```
float height = poiUtil.getCellAutoHeight(printMap.get("Crequest"), 12f);        //自动高度
nRow.setHeightInPoints(height);
```

10） 如何插入一个分页符？

```
if(p>0){
    sheet.setRowBreak(curRow++);    //在第 startRow 行设置分页符
}
```

11） 审单人增加名称后，验货员不会向后推动？

```
utilFuns.fixSpaceStr(contract.getCheckBy(),26)
```

12） 日期如何转为中文格式？

```
UtilFuns.formatDateTimeCN(UtilFuns.dateTimeFormat(contract.getSigningDate()));
```

13） 打印一款货物或者打印两款货物

通过 printStyle 来判断

14） 同一个生产厂家的打印在一页上，不同则另起一页

存在的问题：主要是迭代器的问题，可以改成 List 集合配合下标读取元素。
在判断厂家前，查询的多个货物信息时，必须先排序，按生产厂家名称。

四．Quartz 实现定时任务调度

1.需求及实现思路

当指定的交期到时，给相关人员发送一封邮件进行提醒，需要与工厂进行联系
定时查询库存预警信息，如果存在库存预警信息，发送邮件通知给相关工作人员。

2.什么是 Quartz 框架

Quartz 是一个开源的作业调度框架，它完全由 Java 写成，并设计用于 J2SE 和 J2EE 应用中。它提供了巨大的灵活性而不牺牲简单性。你能够用它来为执行一个作业而创建简单的或复杂的调度。

1、Job

表示一个任务（工作），要执行的具体内容。

2、JobDetail

JobDetail 表示一个具体的可执行的调度程序，Job 是这个可执行程调度程序所要执行的内容，另外 JobDetail 还包含了这个任务调度的方案和策略。

3、Trigger

代表一个调度参数的配置，什么时候去调。

4、Scheduler

代表一个调度容器，一个调度容器中可以注册多个 JobDetail 和 Trigger。当 Trigger 与 JobDetail 组合，就可以被 Scheduler 容器调度了。
3.准备工作及测试

3.引入 Quartz 框架

修改 ilcbs_parent 工程中的 pom.xml，添加依赖

```
        <dependency>
            <groupId>org.quartz-scheduler</groupId>
            <artifactId>quartz</artifactId>
```

```
            <version>2.2.3</version>
        </dependency>
```

4.编写测试代码

在 ilcbs_server 工程中创建 cn.itcast.ilcbs.job 包，包下创建类 JobTest

```
package cn.itcast.ilcbs.job;
import java.util.Date;
public class JobTest {
    public void execute(){
        System.out.println("执行了调度"+new Date());
    }
}
```

5.编写 Spring 与 Quartz 整合配置文件

创建配置文件 applicationContext_job.xml

```
    <!-- 定义一个任务类 -->
    <bean id="mailJobBean" class="cn.itcast.ilcbs.job.JobTest "></bean>
    <!-- 任务类描述 -->
    <bean id="mailJobDetail"
class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFac
toryBean">
        <property name="targetObject" ref="mailJobBean"></property>
        <property name="targetMethod" value="execute"></property>
    </bean>
    <!-- 触发器  -->
    <bean id="mailTrigger"
class="org.springframework.scheduling.quartz.CronTriggerFactoryBean">
        <property name="jobDetail" ref="mailJobDetail"></property>
        <!-- 表达式，每 10 秒执行一次 -->
        <property name="cronExpression" value="0/10 * * * * ?"></property>
    </bean>
    <!-- 总管理容器 -->
    <bean id="startQuartz"
class="org.springframework.scheduling.quartz.SchedulerFactoryBean" >
        <property name="triggers">
            <list>
                <ref bean="mailTrigger"/>
            </list>
        </property>
    </bean>
```

启动 TOMCAT 测试

```
执行了发送邮件的方法Fri Jun 24 17:51:10 CST 2016
执行了发送邮件的方法Fri Jun 24 17:51:20 CST 2016
执行了发送邮件的方法Fri Jun 24 17:51:30 CST 2016
执行了发送邮件的方法Fri Jun 24 17:51:40 CST 2016
```

五．Cron 表达式写法

1.Cron 表达式的域

Quartz Cron 表达式支持到七个域

* * * ? * *

这个表达会每秒钟(每分种的、每小时的、每天的)激发一个部署的 job。

名称	是否必须	允许值	特殊字符
秒	是	0-59	, - * /
分	是	0-59	, - * /
时	是	0-23	, - * /
日	是	1-31	, - * ? / L W C
月	是	1-12 或 JAN-DEC	, - * /
周	是	1-7 或 SUN-SAT	, - * ? / L C #
年	否	空 或 1970-2099	, - * /

理解特殊字符

同 UNIX cron 一样，Quartz cron 表达式支持用特殊字符来创建更为复杂的执行计划。然而，Quartz 在特殊字符的支持上比标准 UNIX cron 表达式更丰富了。

* 星号

使用星号(*) 指示着你想在这个域上包含所有合法的值。例如，在月份域上使用星号意味着每个月都会触发这个 trigger。

表达式样例：

0 * 17 * * ?

意义：每天从下午 5 点到下午 5:59 中的每分钟激发一次 trigger。它停在下午 5:59 是因为值 17 在小时域上，在下午 6 点时，小时变为 18 了，也就不再理会这个 trigger，直到下一天的下午 5 点。

在你希望 trigger 在该域的所有有效值上被激发时使用 * 字符。

? 问号

? 号只能用在日和周域上，但是不能在这两个域上同时使用。你可以认为 ? 字符是 "我并不关心在该域上是什么值。" 这不同于星号，星号是指示着该域上的每一个值。? 是说不为该域指定值。

不能同时这两个域上指定值的理由是难以解释甚至是难以理解的。基本上，假定同时指定值的话，意义就会变得含混不清了：考虑一下，如果一个表达式在日域上有值 11，同时在周域上指定了 WED。那么是要 trigger 仅在每个月的 11 号，且正好又是星期三那天被激发？还是在每个星期三的 11 号被激发呢？要去除这种不明确性的办法就是不能同时在这两个域上指定值。

只要记住，假如你为这两域的其中一个指定了值，那就必须在另一个字值上放一个 ?。

表达式样例：

0 10,44 14 ? 3 WEB

意义：在三月中的每个星期三的下午 2:10 和 下午 2:44 被触发。

, 逗号

逗号 (,) 是用来在给某个域上指定一个值列表的。例如，使用值 0,15,30,45 在秒域上意味着每 15 秒触发一个 trigger。

表达式样例：

0 0,15,30,45 * * * ?

意义：每刻钟触发一次 trigger。

/ 斜杠

斜杠 (/) 是用于时间表的递增的。我们刚刚用了逗号来表示每 15 分钟的递增，但是我们也能写成这样 0/15。

表达式样例：

0/15 0/30 * * * ?

意义：在整点和半点时每 15 秒触发 trigger。

- 中划线

中划线 (-) 用于指定一个范围。例如，在小时域上的 3-8 意味着 "3,4,5,6,7 和 8 点。" 域的值不允许回卷，所以像 50-10 这样的值是不允许的。

表达式样例：

0 45 3-8 ? * *

意义：在上午的 3 点至上午的 8 点的 45 分时触发 trigger。

L 字母

L 说明了某域上允许的最后一个值。它仅被日和周域支持。当用在日域上，表示的是在月域上指定的月份的最后一天。例如，当月域上指定了 JAN 时，在日域上的 L 会促使 trigger 在 1 月 31 号被触发。假如月域上是 SEP，那么 L 会预示着在 9 月 30 号触发。换句话说，就是不管指定了哪个月，都是在相应月份的时最后一天触发 trigger。

表达式 0 0 8 L * ?

意义是在每个月最后一天的上午 8:00 触发 trigger。在月域上的 * 说明是 "每个月"。

当 L 字母用于周域上，指示着周的最后一天，就是星期六 (或者数字 7)。所以如果你需要在每个月的最后一个星期六下午的 11:59 触发 trigger，你可以用这样的表达式 0 59 23 ? * L。

当使用于周域上，你可以用一个数字与 L 连起来表示月份的最后一个星期 X。例如，表达式 0 0 12 ? * 2L 说的是在每个月的最后一个星期一触发 trigger。

> 不要让范围和列表值与 L 连用
>
> 虽然你能用星期数(1-7)与 L 连用，但是不允许你用一个范围值和列表值与 L 连用。这会产生不可预知的结果。

W 字母

W 字符代表着平日 (Mon-Fri)，并且仅能用于日域中。它用来指定离指定日的最近的一个平日。大部分的商业处理都是基于工作周的，所以 W 字符可能是非常重要的。例如，日域中的 15W 意味着 "离该月 15 号的最近一个平日。" 假如 15 号是星期六，那么 trigger 会在 14 号(星期四)触发，因为距 15 号最近的是星期一，这个例子中也会是 17 号（译者 Unmi 注：不会在 17 号触发的，如果是 15W，可能会是在 14 号(15 号是星期六)或者 15 号(15 号是星期天)触发，也就是只能出现在邻近的一天，如果 15 号当天为平日直接就会当日执行）。W 只能用在指定的日域为单天，不能是范围或列表值。

井号

字符仅能用于周域中。它用于指定月份中的第几周的哪一天。例如，如果你指定周域的值为 6#3，它意思是某月的第三个周五 (6=星期五，#3 意味着月份中的第三周)。另一个例子 2#1 意思是某月的第一个星期一 (2=星期一，#1 意味着月份中的第一周)。注意，假如你指定 #5，然而月份中没有第 5 周，那么该月不会触发。

2.测试示例

我们自测下面表达式意义？

"0 0 12 * *?" 每天中午 12 点触发

"0 15 10 ? **" 每天上午 10:15 触发

"0 15 10 * *?" 每天上午 10:15 触发

"0 15 10 * * ?*" 每天上午 10:15 触发

"0 15 10 * * ?2005" 2005 年的每天上午 10:15 触发

"0 * 14 * *?" 在每天下午 2 点到下午 2:59 期间的每 1 分钟触发

"0 0/5 14 * *?" 在每天下午 2 点到下午 2:55 期间的每 5 分钟触发

"0 0/5 14,18 ** ?" 在每天下午 2 点到 2:55 期间和下午 6 点到 6:55 期间的每 5 分钟触发

"0 0-5 14 * *?" 在每天下午 2 点到下午 2:05 期间的每 1 分钟触发

"0 10,44 14 ? 3WED" 每年三月的星期三的下午 2:10 和 2:44 触发

"0 15 10 ? *MON-FRI" 周一至周五的上午 10:15 触发

"0 15 10 15 *?" 每月 15 日上午 10:15 触发

"0 15 10 L *?" 每月最后一日的上午 10:15 触发

"0 15 10 ? *6L" 每月的最后一个星期五上午 10:15 触发

"0 15 10 ? * 6L2002-2005" 2002 年至 2005 年的每月的最后一个星期五上午 10:15 触发

"0 15 10 ? *6#3" 每月的第三个星期五上午 10:15 触发

六.作业

1.题目

自己动手实现交期到期后，早上 **8:30** 分发送一封邮件到船运部经理的邮件

2.参考答案

```
public class DeliveryPeriodJob {
	//注入 ContractService
	private ContractService contractService;
	public void setContractService(ContractService contractService) {
		this.contractService = contractService;
	}
	/**
	 * 以当前时间为标准，查询出交期到期的购销合同，并进行邮件发送，以提醒负责人
	 * @throws Exception
	 */
	public void execute() throws Exception{
		String hql = "from Contract where to_char(deliveryPeriod,'yyyy-MM-dd')=?";
		//获取当前时间
		String deteStr	= new SimpleDateFormat("yyyy-MM-dd").format(new Date());
		List<Contract> list = contractService.find(hql, Contract.class, new String[]{deteStr});

		//判断集合，如果不为空，说明有购销合同今天到期
		if(list!=null && list.size()>0){
			for(final Contract c :list){
				Thread.sleep(3000);//让当前线程休眠	3 秒

				Thread th = new Thread(new Runnable() {
					public void run() {
						//发送邮件的代码
						try {
						MailUtil.sendMail("3462420264@qq.com", "提醒：交期到了", "主人您好，您有一个购销合同的交货日期于"+new SimpleDateFormat("yyyy-MM-dd hh:mm:ss").format(c.getDeliveryPeriod())+"到期");
```

```
                                } catch (Exception e) {
                                    e.printStackTrace();
                                }
                            }
                        });

                        th.start();
                    }
                }else{
                    System.out.println("主人，您目前还没有购销合同交期到期的!!!");
                }
            }
        }
```