快递物流管理系统

总结报告

AXIS工作组

南京大学软件学院

2015年12月30日

## 项目概况：

快递物流管理系统就是为满足xx企业业务发展需求而开发的，它包括一个数据集中服务器和多个客户端。数据集中服务器将所有的数据储存起来进行维护。用户通过客户端完成日常任务，客户端与数据集中服务器才是实时通信的方式完成数据交换。

## 团队介绍：

|  |  |
| --- | --- |
| 陈自强 | 141250017 |
| 邢程 | 141250137 |
| 余旻晨 | 141250177 |
| 张瑞 | 151240189 |

## 项目时间表

### 9.15-10.14 需求开发阶段 ：

作为无知的小白，在这个阶段什么都不懂，用例文档写的很想当然= =，后期改了N次，这期间完成了用例文档、需求规格说明文档、功能测试用例测试套件、各种概念类图。。。然后来发现，需求阶段果然只是做做而已

### 10.15 – 10.26 体系结构设计阶段：

项目开始有了雏形，参照着去年师兄的文档完成了体系架构的设计，对分层只有一个表象的了解：

AXIS： 我们这次用什么模式啊？

LiuQin： 分层模式

AXIS： 有什么用吗？

LiuQin： 能并行开发、加快开发进度

AXIS： 我们这次用什么模式啊？

LiuQin： 接口稳定很重要

AXIS： 哦

AXIS： VOPO有什么用吗？

LiuQin： 包装属性、PO是可持久化存储对象，VO是用于传给上层，防止底层数据暴露的媒介

AXIS： 哦（还是没懂有什么区别）

LiuQin： （今年学生咋这么6，居然都懂了）

### 10.15-12.31的日常对话：

某A：orderPO我加了一个属性，还有那个金额的从int改成double，你帮我解决一下

某C：哦。

（几小时后）

某C:改完了，终于……

某A：order的逻辑层加了一个接口，还有我把XX方法返回值改成了ArrayList

某C：哦

……

……

……

某C：悔不听LiuQin的话啊……

以上反映了这次体系设计阶段对设计模式了解不透彻，不明白其含义所导致的后期结构修改频繁导致开发效率低下的问题。

### 10.26-11.16 详细设计阶段、人机交互设计阶段：

这次我们小组的详细设计阶段与人机交互设计阶段形同虚设，主要原因是老师详细设计阶段当时还没讲到（怒甩锅），小组成员对设计阶段重要性没有认识，加之连续文档一个月，对码文档深恶痛绝，于是应付从而草草了事，结果就是我们的详细设计阶段没有起到指导设计的作用，没有在详细设计阶段针对体系结构设计阶段所存在的一些重大缺陷进行发现，并且UI层设计为0！！！导致UI层开发迟缓，直到11月下旬才正式开始，组员们因此承担了过重的任务。

人机交互设计也存在应付的问题，原型图过于简单，美其名曰：等到写代码时再来考虑，结果导致写代码时还不得不来考虑界面。人机交互设计没有做好也导致了功能覆盖的缺失：有部分功能直到写界面才发现没有接口，导致重复加接口的问题。

### 11.16-11.22 空闲期

期中考压身，什么都没干 = ==（不过数据层写完了）

### 11.22-12.25 疯狂码代码

期间遇到的问题多不胜数，不提逻辑漏洞，光是数据库、maven、jeckins、以及时不时崩的github就让我们奔溃很久，庆幸的是我们还是稳步前进，顺利的通过了一个个DDL。但是相对而言，我们组对于代码之外的其他就关注的比较少了，例如Junit之类的，亦或是对于代码的重构等等

## 项目成果展示

详见PPT

## 项目亮点

总体而言，我们的项目“结构化”较强，在UI、逻辑层、数据层都出现了大规模继承的情况，例如UI，大部分panel都是继承自mypanel，逻辑层所有控制器都有父类，数据层亦是，尤其是UI层，父类的作用主要是定义了一系列抽象的步骤，使得写UI变得更加机械。好处是不会漏掉一些组件或者步骤，例如我们抽象出的confirmlistener有四个步骤：检查数据格式、存至数据库、更新信息，重新初始化，四个抽象方法保证在写这些确认时不会出现漏写的情况

### UI的亮点：

1. 框架结构化：

保证流程的完整、清晰

1. 每个类结构化：

体现每个panel编写的流程（包括button、label、whitepanel等一系列组件的初始化，添加进面板，添加监听），避免流程中细节的遗漏。

1. 封装数据检查：

为不同的数据设置枚举类型，集中进行检查，方便管理。

1. 集中式控制：

每个人都有固定controller，控制对应mainpanel和下属panel，统一controller使用cardlayout来进行控制切换。人员与人员之间比较独立，实现了UI层完全并行开发的可能性。

1. 采用xml：

将ui的数据和操作分离，方便调节界面的参数。

1. 所有组件重绘：

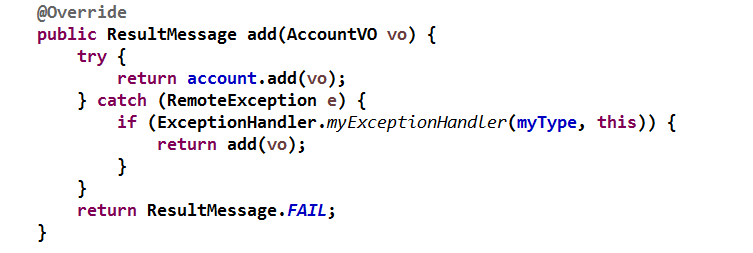
全部自定义组件，对反复需要使用的功能抽象出来进行封装。使得组件与界面的代码耦合度降低

1. 采用统一的命名风格:

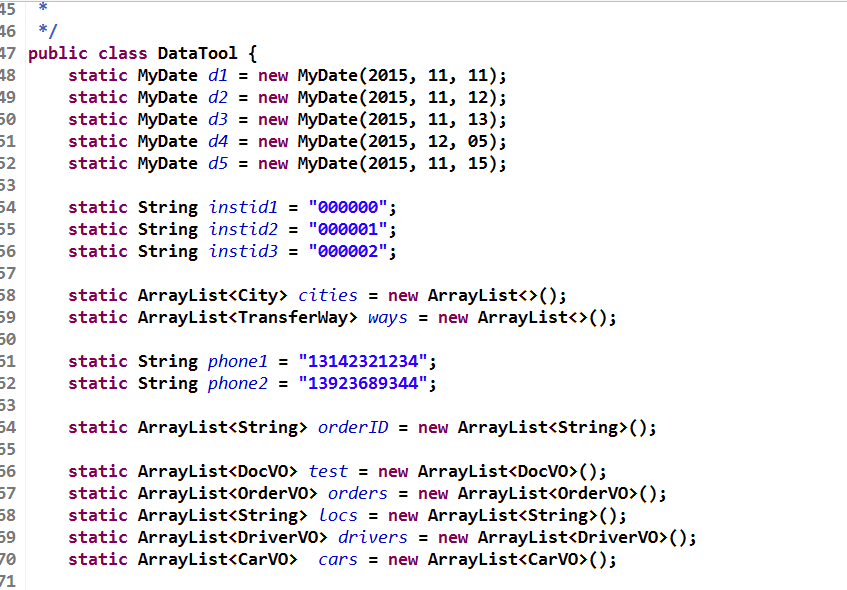
后缀名 panel /table/controller（控制器）/主界面（main），建立精神模型，便于团队合作开发

### Bl的亮点：

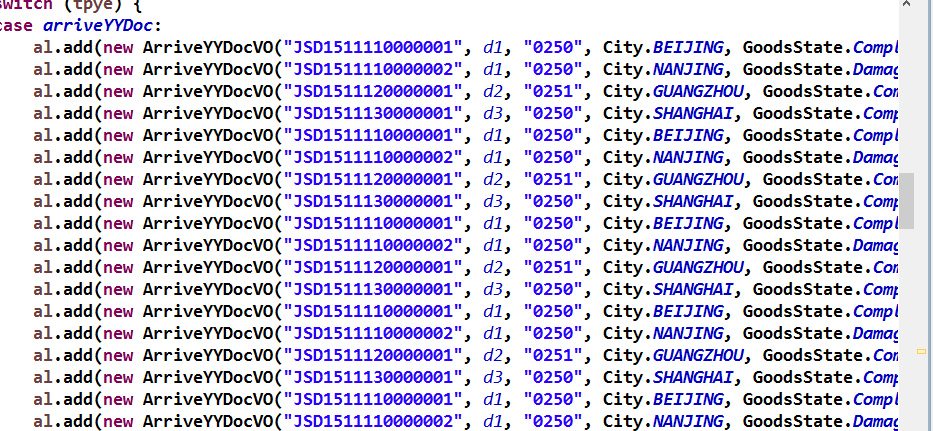
1. 采用blcontroller,对网络异常进行集中处理，使得网络异常能够较为简单的处理完成

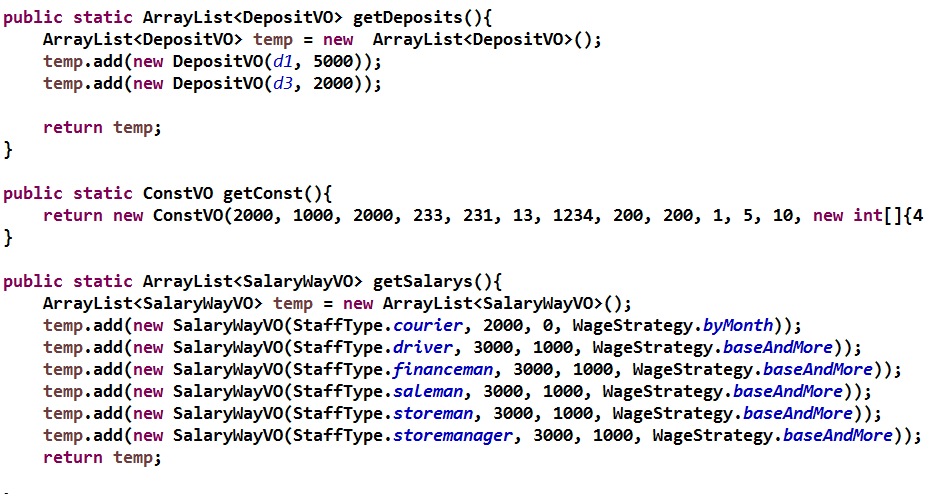


1. 采用集中式模拟数据工厂，由于开发时需要大量的数据来测试，我们专门写了一个类来生产各种各样的数据，这样的好处是大大减少了写桩的工作量和编写数据的工作量，使得写桩时不需要编数据，也便于数据统一管理



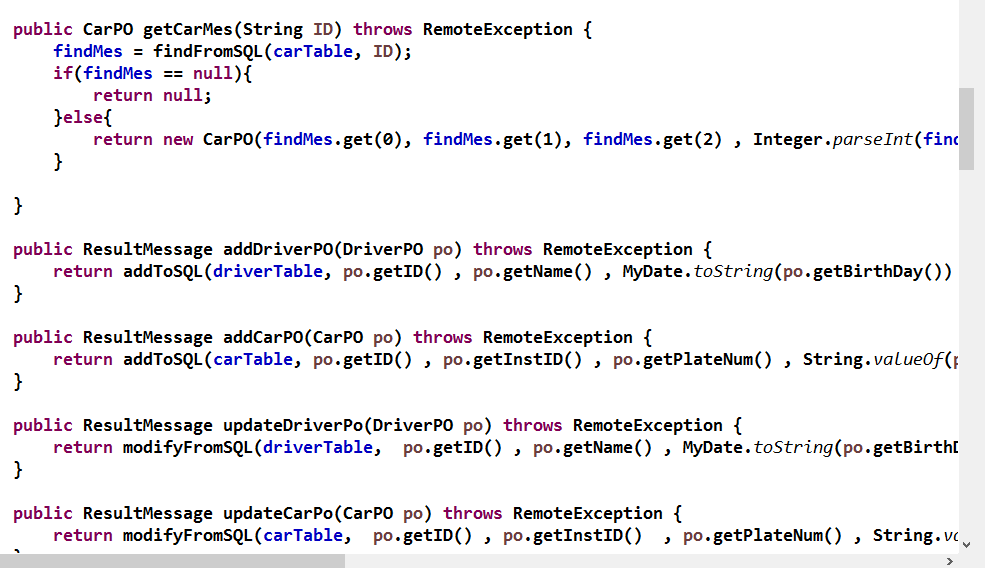




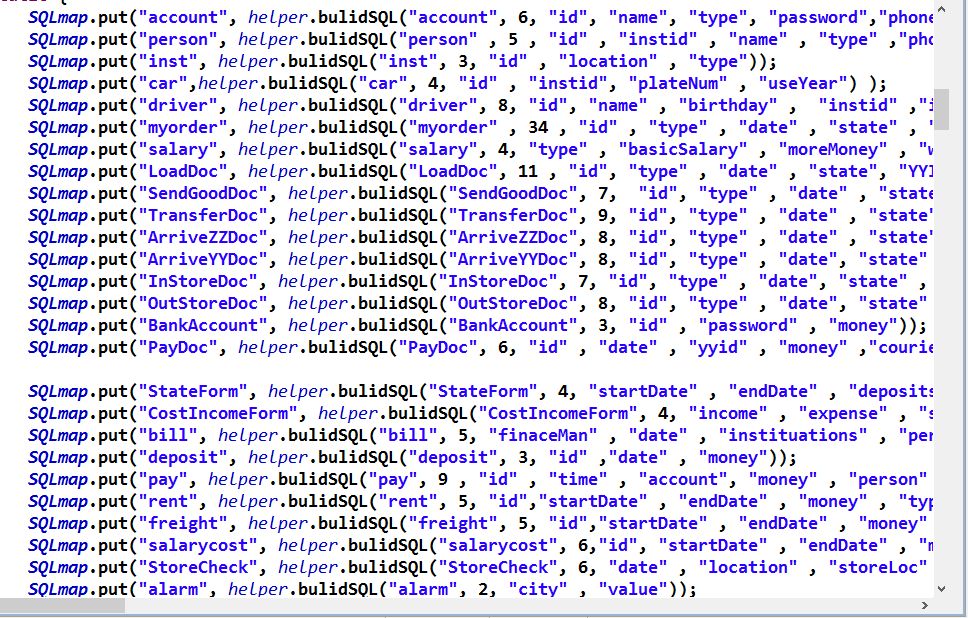


### Data的亮点：

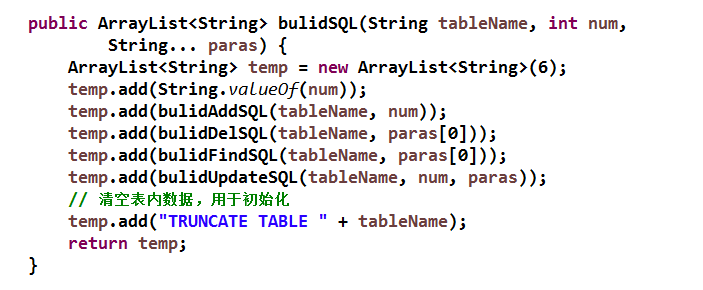
1. 提取父类，集中管理，父类以及helper承担起数据库大部分功能，使得子类的代码较为简单，甚至于仅涉及到调用构造器和赋值，不存在任何逻辑，代码可修改性大大得到提高，另一方面，将子类会用到的成员变量全部声明在父类，虽有一次声明，到处使用的嫌疑，毕竟还是方便和已修改（前提是父类足够稳定，本次的数据层符合这个特点，全过程中父类几乎无修改）

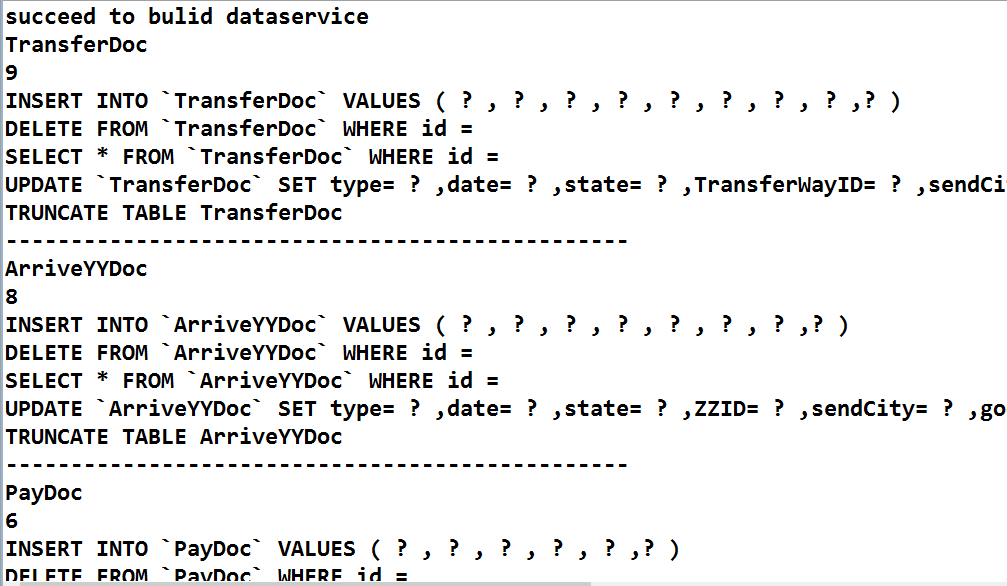


1. 数据集中，所有的数据库数据全都放在一个map中集中处理，便于修改



1. 数据库语句统一生成：利用可变参数列表，实现了只需要推入参数，就可以自动常用生成常用数据库语句语句，这样，在执行数据库语句时非常方便，可以统一使用，仅需要像上图一样把表给加入map中就可以了





## 经验

经验基本在亮点里面说到了，除此之外，最大的经验就是还是要好好做设计啊

====================================================

## 教训

本次项目总体而言不是十分的成功，其背后原因是众多的，除去人为原因、工具老出bug这些原因，还有以下几点教训

1. 接口、POVO要稳定！！！！！！！！！不多说了，心累
2. 工具要提早搞定，我们组的数据层一开始只是由负责数据层开发的人操作，其他人一直没有装数据库，后来为了数据库浪费了很多时间，必要的工具应当今早部署完毕
3. 要正确估计自身的实力，拿出合适的计划，量力而行。

我们组在前期对自身实力估计过高，想着实现什么并发处理、全自动化等等，后来发现真的太年轻，没有考虑到其他学业等压力的存在，从而导致很多前期的设想后期完全无法实现

1. （打一下脸）UI层的设计其实做的还是不好，有很多提高的空间，如界面跳转没有完全独立出来，一个组件从生成，加进去等一个不走应该可以用反射来实现，减少冗余代码。界面复用弱，基本上一个界面一个类，其实像很多表格类都是可以抽象出来直接使用的。

## 团队分工

陈自强：队长，负责数据层和UI层架构和部分代码编写

邢程：负责文档、UI层代码编写

余旻晨：负责逻辑层、UI层代码编写

张瑞：美工，负责P图、统一界面风格、文档编写

## 展望

无论如何，大作业确确实实是我们大二上生活非常大的一部分，付出了这么多的时间，无论做的怎样，都是有所收获的，尽管还有所不足，在做大作业的期间也有几次因为bug和工具而几乎崩溃，但是很高兴我们坚持了下来，也至少算是交上了一份完整的作业，已经知足了，我们会汲取在这次大作业中的经验和教训，继续奋斗！

AXIS小组

2015年12月30日