

# 目录

第一章 项目介绍.....	3
1.1 编写目的 .....	3
1.2 背景说明 .....	3
1.3 开发环境 .....	3
第二章 项目开发介绍 .....	4
2.1 目标.....	4
2.2 用户的特点.....	4
2.3 假定与约束.....	4
第三章 信用卡业务调研.....	5
3.1 信用卡基础知识 .....	5
3.2 信用卡在我国的发展 .....	6
3.3 我国信用卡发展遇到的问题 .....	7
第四章 需求分析.....	8
4.1 对功能的规定 .....	8
4.2 系统的基本结构 .....	10
4.3 对性能的规定 .....	12
第五章 系统详细设计 .....	13
5.1 总体流程图.....	13
5.2 数据库设计.....	13
5.2.1 数据库表设计 .....	13
5.2.2、数据库表之间的关系示意图 .....	14
5.3 静态结构建模.....	15
5.3.1 定义系统对象类.....	15
5.3.2 建立类图.....	16
5.4 动态行为建模.....	17
第六章 系统实现.....	19
6.1、编写 REXX 生成测试数据.....	19
6.2、COBOL 处理后台业务逻辑.....	20
6.3、系统最终输出结果 .....	21
6.4、开发过程中遇到的问题和解决方案 .....	23
第七章 系统测试.....	25
7.1 测试产品信息 .....	25
7.2 测试内容 .....	25
7.3 测试用例 .....	26
7.4 遗留缺陷和建议 .....	27
第八章 项目核心技术 .....	28
8.1 REXX 语言 .....	28
8.2 COBOL 语言.....	28
8.3 中间件.....	28
第九章 开发心得.....	29
9.1 成员心得 .....	29

参考文献.....	31
附录 I 冷启动参数和 COBOL 操作文件 .....	32
冷启动参数.....	32
COBOL 操作文件.....	33
附录 II 虚拟机的安装 <sup>[3]</sup> .....	34
附录 III 部分编码.....	38
模拟数据编码 .....	38
后台业务处理编码.....	40

## 第一章 项目介绍

### 1.1 编写目的

为了保证项目团队按时保质地完成项目目标,便于团队成员更好地了解项目的情况,使项目工作开展的各个过程合理有序,方便用户快速了解产品的特性,我们以文件化的形式,对于项目生命周期内的工作任务范围、各项工作的任务分解、调研资料、项目团队组织结构、各团队成员的工作责任、风险对策等内容做出的安排进行规范清晰地表达,作为项目周期内所有项目活动的行动基础,项目团队开展和检查项目工作的依据。

### 1.2 背景说明

本项目的名称是银行信用卡业务后台子系统的实现,也是电子科技大学信息与软件工程学院综合课程设计II的课设项目。全部任务由大型主机系的三名同学:杨林彬、罗阳星、郑伟合作完成,项目开发历时3个月。

银行信用卡是中国个人金融服务市场中成长最快的产品之一,它能够促进商品销售,刺激社会需求,对于我们国家拉动内需,解决产能过剩的问题具有重要的意义。信用卡业务系统一个重要的指标是可靠性。我们的系统利用COBOL语言完成后台业务逻辑的开发,批量处理测试数据,并根据不同的消费情况,在每个月的账单日打印出用户的账单信息。这个模拟系统的设计与实现为我们更好地了解当今信用卡的发展趋势和业务流程打下了基础,也是对我们团队协作、系统的设计能力的一种锻炼。

### 1.3 开发环境

本系统的开发环境为IBM主机系统Z900,使用安装在WINDOWS微机上的PCOM 3270仿真终端软件连接主机系统进行相应操作;同时主机需要COBOL语言和REXX语言环境的支持。

## 第二章 项目开发介绍

### 2.1 目标

1. 对银行信用卡业务的后台子系统部分进行需求调研，进行合理假设。要求实现如下功能：
  - a) 信用卡消费与取现。
  - b) 用户账单生成。
  - c) 还款。
  - d) 各等级情况的利息计算。
  - e) 统计利息文件生成。
2. 完成概要设计，进行模块划分。
3. 完成详细设计，使用 CASE 工具对各流程进行充分表达。
4. 完成存储设计，设计涉及的顺序文件、VSAM 文件的参数与记录格式。
5. 完成系统实现，用 COBOL 语言开发出最终记录处理代码。
6. 系统测试。
  - a) 自行开发出测试数据生成程序，产生前端交易数据。
  - b) 用开发的后台子系统程序对产生的交易数据集进行处理，产生期望输出。
7. 预期成果或目标：
  - a) 需求分析、概要设计、详细设计文档、完整系统编码、测试使用情况。
8. 最终完成对银行信用卡业务后台子系统的完整开发。

### 2.2 用户的特点

本后台子系统的使用对象是相关信用卡部门及大型主机编程爱好者，具有很高的行业特性。

### 2.3 假定与约束

我们从实际的开发情况出发，考虑到项目的风险等问题，指定了一些假定。假定主要包括：用户假定、交易记录假定、条件假定、时间假定、技术假定。

1. 用户假定：每个用户只拥有一个帐户，且额度暂不会发生变化。
2. 交易记录假定：在 3 个月之中，每天产生一条消费或者是取现的交易记录，且交易数目在 50.00-99.99 之间（四位数），用户消费总额不会超过帐户额度，将所有还款记录等价为一笔还款日当天还款记录。
3. 条件假定：手动制定还款金额，确保覆盖每一种利息计算情况。
4. 时间假定：每个月的 10 号为还款日，30 号为账单日，每个月恒定 30 天。
5. 技术约束：本项目的开发是在 COBOL 程序设计语言的环境下进行的，结合大型主机编程语言 REXX 进行整合开发。
6. 环境约束：运行该子系统所适用的具体环境是 IBM 主机系统 z900、PCOM 或其他 3270 终端仿真软件。

## 第三章 信用卡业务调研

### 3.1 信用卡基础知识

我们选择的课题名称为银行信用卡业务后台子系统的设计与实现。经过仔细的调研，我们对于银行信用卡的基本业务流程有了一定的了解。



图 3-1 日常生活中的信用卡

<sup>[1]</sup>银行信用卡（Credit Card）是银行向个人和单位发行的，凭此向特约单位购物、消费和向银行存取现金，其形式是一张正面印有发卡银行名称、有效期、号码、持卡人姓名等内容，由芯片、磁条、签名条等组成的磁卡。持卡人持信用卡消费时无需支付现金，待账单日时再还款。

信用卡业务中的关键词：

1. 账单日：银行每月会固定一天对持卡人的信用卡账户中当期发生的各项交易、费用等进行汇总结算，并结记利息，计算持卡人当期总欠款金额和最小还款额，打印出对账单邮寄给客户以便持卡人进行核对，并提示持卡人还款的最后日期和还款金额。
2. 到期还款日：发卡行要求持卡人归还当期信用卡应付款项的最后日期，一般为对账单日起的第 20 天。
3. 全额还款：持卡人再信用卡到期还款日前偿还对账单上所列示的全部应付款项。采取全额还款的持卡人其当期对账单所列示的信用卡消费可以享受免息还款的待遇。

4. 最低还款：持卡人在到期还款日前偿还当期对账单上的全部应付款项有困难时，可以按发卡行规定的最低还款额进行还款，如持卡人选择最低还款，就不能享受免息还款的待遇。
5. 免息还款期：持卡人利用信用卡进行交易消费，从银行记账日起至到期还款日的期间，在此期间，持卡人只要全额还清当期对账单上所列示的全部应付款项，便不用向银行支付消费交易所产生的贷款利息。

## 3.2 信用卡在我国的发展

随着中国经济的迅速发展，居民的购买力的不断增强以及消费观念的转变，信用卡作为一种便捷的交易方式，在国内有了很大的发展。

信用卡是一种由银行或信用卡公司签发，证明持卡人信誉良好，可以在指定的商店、服务场所消费或在各地的金融机构取现，办理结算的信用凭证和支付工具。

1952 年最早由美国加利福尼亚州的富兰克林国民银行作为金融机构首先进入发行信用卡的领域，由于信用卡具有方便性、通用性以及可以进行善意的透支，信用卡很快收到社会各界的普遍欢迎，并得到迅速地发展。



图 3-2 美国富兰克林国民银行

中国第一张信用卡是中国珠海银行于 1985 年发行的中银卡。中国信用卡市场的开始产生，经过了漫长的启动阶段，从 2003 年开始受益于国名经济的持续增长和受理环境的改善，中国的信用卡用户开始激增。根据央行最新的统计数据，2006 年底我国各银行信用卡发行量为 4958 万张。2007 年前 3 个月，信用卡发行量每月平均增长在 100 万张以上。截至 2008 年第二季度末，我国信用卡总发卡

量达到了 1.04 亿，占了全国银行卡总发卡量的 8%。到 2014 年末全国累计发行银行卡 49.36 亿张，其中信用卡累计发卡 4.55 亿张。全国人均持有信用卡 0.34 张。以下为我国近 2008 年到 2015 年信用卡发卡量示意图：

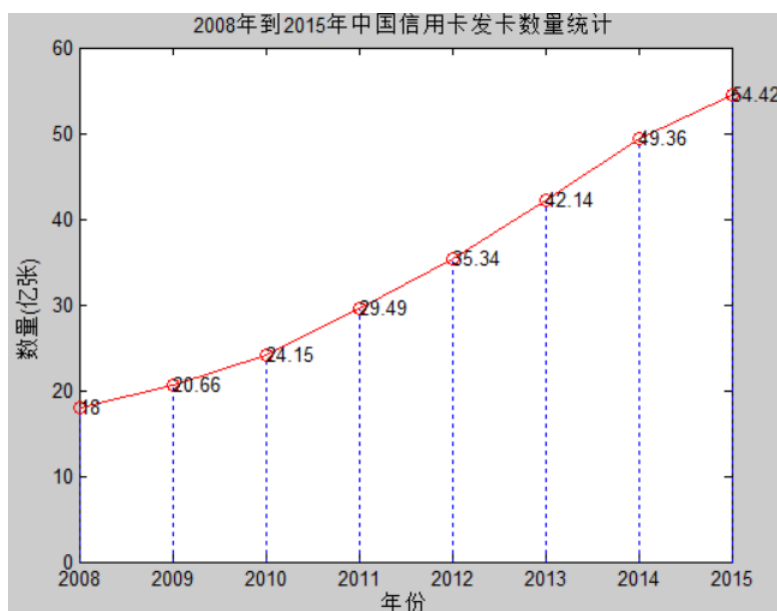


图 3-3 信用卡发卡量数据统计

### 3.3 我国信用卡发展遇到的问题

1. 信用卡的睡眠卡数量巨大。虽然中国的信用卡近年来获得了迅速的发展，但是信用卡的总体渗透率依然较低，这主要是因为一人多卡的现象在中国较为普遍。
2. 社会个人信用征信体系不够健全。中国的信用评估面临着数据瓶颈。目前个人信用信息基础数据库已经实现全国联网，这在一定程度上完善了中国信用卡市场。但是受到个人收入不透明、个人征税机制不完善等主管客观因素的影响，数据的有效性和真实性还是打了折扣。
3. 相应的法律法规还有待进一步完善。目前我国刑法对于信用卡业务的各个专业环节中出现的犯罪行为尚未做到全面覆盖。另外征信数据的使用、个人隐私的保护等方面做得还不够完善。

总的来说，近 20 年来我们国家的信用卡产业得到了迅速的发展，但是其中暴露出的一些问题也值得我们注意。信用卡对于刺激消费、拉动内需具有重要的意义。了解信用卡的业务流程是未来人才所必须的，此外对于我们计算机专业人员来说，注意到信用卡的系统化、自动化的管理是非常有必要的，如果信用卡的核心业务都是由人工驱动，那么风险是巨大的。所以信用卡行业信息系统的学习与开发对于我们而言是非常具有价值的。

## 第四章 需求分析

### 4.1 对功能的规定

本次综合应用设计项目的名称为银行信用卡后台子系统的设计与实现，目的是在大型主机上模拟银行信用卡基础的一些交易业务。通过网络、实地调查和电话咨询，我们认真研究了信用卡的业务流程(见图 3-1)，并结合课题任务书(见附录)，拟定了该业务模拟系统需要实现的一些基本功能需求：

- 1. 消费与取现
- 2. 账单的生成
- 3. 还款
- 4. 利息计算
- 5. 统计文件的生成

信用卡是独立的金融产业，交易的过程以及交易流程中电子信息的传递方式都非常有特色。

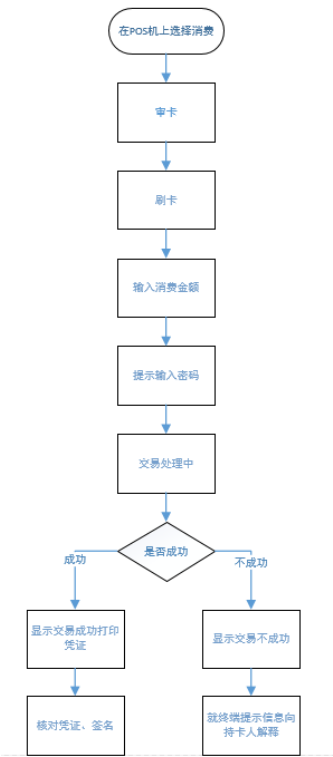


图 4-1 银行信用卡交易业务流程

US1:首先消费与取现在银行信用卡业务中是两个截然不同的概念，它们之间的不同主要是体现在利息的计算上。假设每个月的 10 号是到期还款日，30 号是



账单日。那么 2 月 30 日在计算第本账单周期所负的利息时（我们假设用户在 2 月 10 日没有还清 1 月份的欠款）就要严格分为消费利息和取现利息，而这两者计算方式是截然不同的。用户 Alice 在一月的 DAY1 和 DAY2 消费、取现各 1000 元，在 2 月 10 日只还了 500 元，那么 Alice 在 3 月 10 日前需要承担的消费利息就是：

$$S_{\text{二月消费}} = (2 \text{ 月 } 10 - \text{DAY1}) * 0.05\% * 1000$$

需要承担的取现利息就为：

$$S_{\text{二月取现}} = (2 \text{ 月 } 30 - \text{DAY2}) * 0.05\% * 1000$$

可见取现利息的计算是按照天数计算的，不像消费利息有着交易周期边界。

US2:账单的生成。项目是运行在大型主机的 Z/OS 操作系统中，我们模拟的交易数据计划采用 REXX 来直接生成账单，并把相关数据直接写道顺序数据集里面。这里就涉及到了数据格式的设计问题，数据格式设计是否合理直接影响接下来 COBOL 对账单的业务处理。经过小组的讨论与协商，我们提出来一下的数据格式设计方案（见表格 4-1）

表 4-1

交易流水号	交易人	交易数额	交易日期	交易类型	信用卡额度
交易流水号 由时间+交易人+随机数组 成	交易人 XR+ 交易人编 号	交易数额 统一的为 两位小数	交易日期 的格式 yyyymmdd	0- 消费 1- 取现 2- 还款	信用卡额度 是固定的（这 里指的是办 卡时银行告 知的额度） 3000 元

US3:利息的计算。利息的计算是本项目的核心内容，我们计划采用 COBOL 语言对于准备好的账单数据集进行集中处理。利息的计算除了 US1 中提到的消费利息和现金利息，还有上月没有还清的欠款（包括利息）的利息，也就是常说的“驴滚利”。

我们还是用 Alice 的例子来计算一下她 2 月 30 日所背负的利息。在计算该利息之前我们先要明确 Alice 在一月份的欠款是多少，以下是 Alice 一月份和二月份的交易情况示意图（见下一页）：

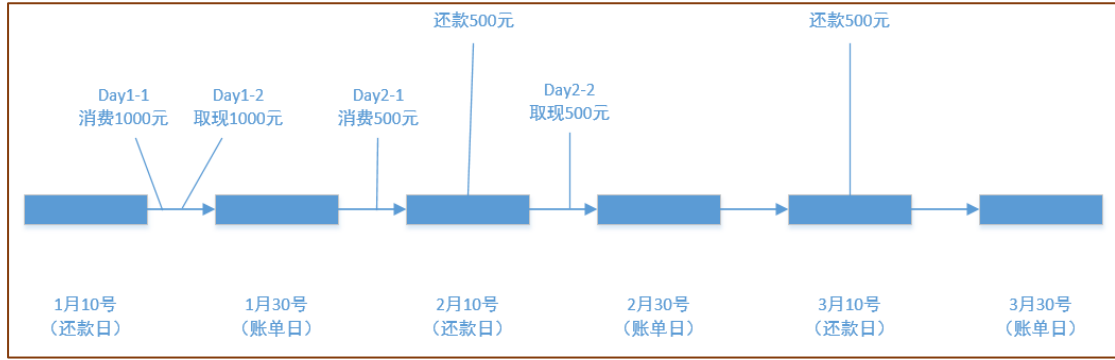


图 4-2 Alice 的交易情况示意图

一月欠款计算如下所示：

$$S_{\text{一月取现}} = (2 \text{月} 30 - \text{DAY2}) * 0.05\% * 1000$$

$$S_{\text{一月消费}} = 0$$

$$S_{\text{一月}} = S_{\text{一月取现}} + S_{\text{一月消费}}$$

$$D_{\text{一月}} = 1000 + 500 + S_{\text{一月}}$$

2 月 30 日利息计算如下所示：

$$S_{\text{二月}} = S_{\text{二月消费}} + S_{\text{二月取现}} + S_{\text{欠款利息}}$$

$$S_{\text{欠款利息}} = (D_{\text{一月}} - 500) * 0.05\% * (2 \text{月} 30 - 2 \text{月} 10) + S_{\text{一月}} * 0.05\% * 10$$

US4:统计文件的生成。在完成上述的功能需求后，最后一步就是把交易结果写入一个新的数据集中，其实就是在模拟用户在账单日的时候查看自己的交易记录。这个记录的内容包括：交易人姓名、信用卡额度、交易流水记录、利息、最低还款额。这其中利息对于用户来说至关重要，因为如果在下个到期还款日，用户必须还清利息和欠款，否则又会进行“利滚利”式的利息迭代，用户信用等级降低了不说，高额的利息也让人难以承担，因此我们在使用信用卡消费的时候一定要小心谨慎，及时还清卡上的欠款。

## 4.2 系统的基本结构

整个系统的用例图如下所示（见下一页）：

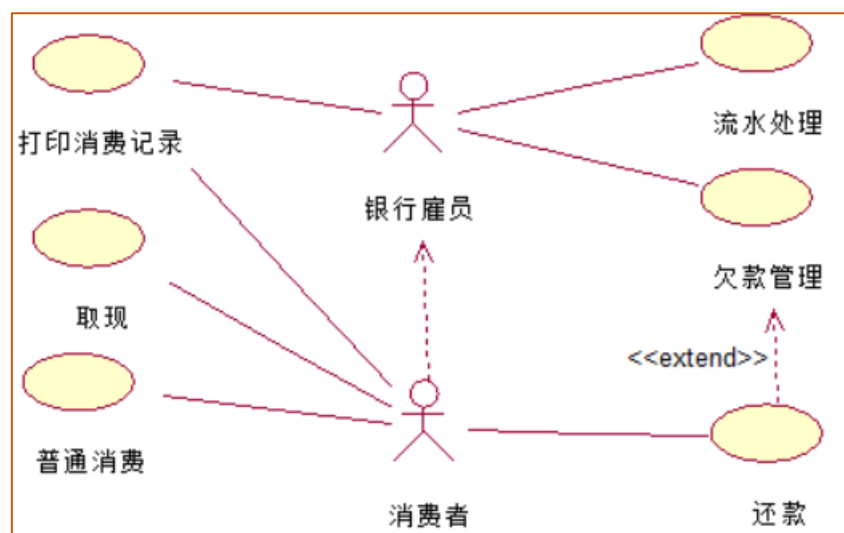


图 4-3 系统的用例总图

模拟的系统中一共有两个角色：银行雇员和消费者。根据 4.1 节对于系统功能的规定，我们有消费者的行为一共可以分为三种：

1. 普通消费
2. 取现
3. 还款

银行雇员代表信用卡后台逻辑处理单元，银行雇员可以处理每月的客户消费流水，打印消费记录并对客户欠款进行管理。

本次综合应用设计 II 我们着重于对数据的批处理，但是如果这个系统要完全模拟出来的话还需要考虑更多的因素。例如不同角色权限的设定、数据库（这里可以采用 VSAM）的结构的设计、UI 界面的设计等等。整个系统的部署图见图 4-5。考虑到我们只是对数据进行简单批处理，我们系统开发的框架大致如下所示：

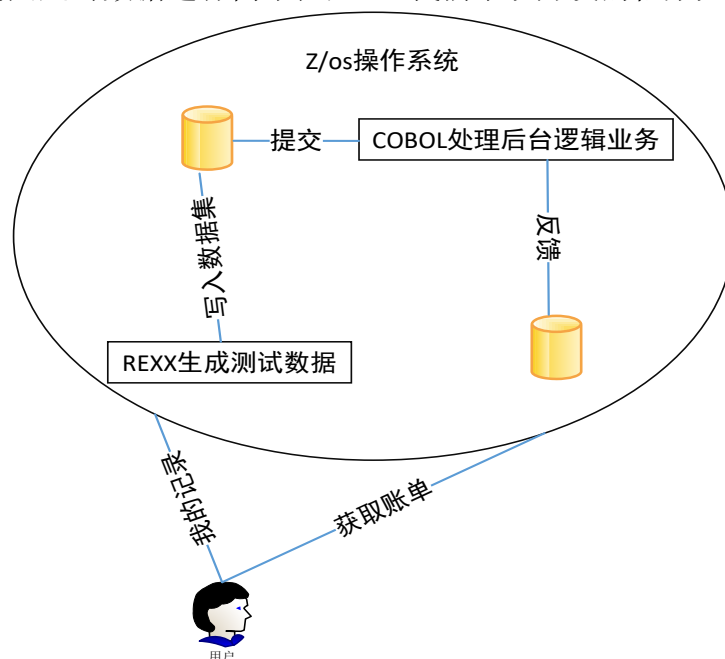


图 4-4 本系统结构图

完整系统的部署图如下所示：

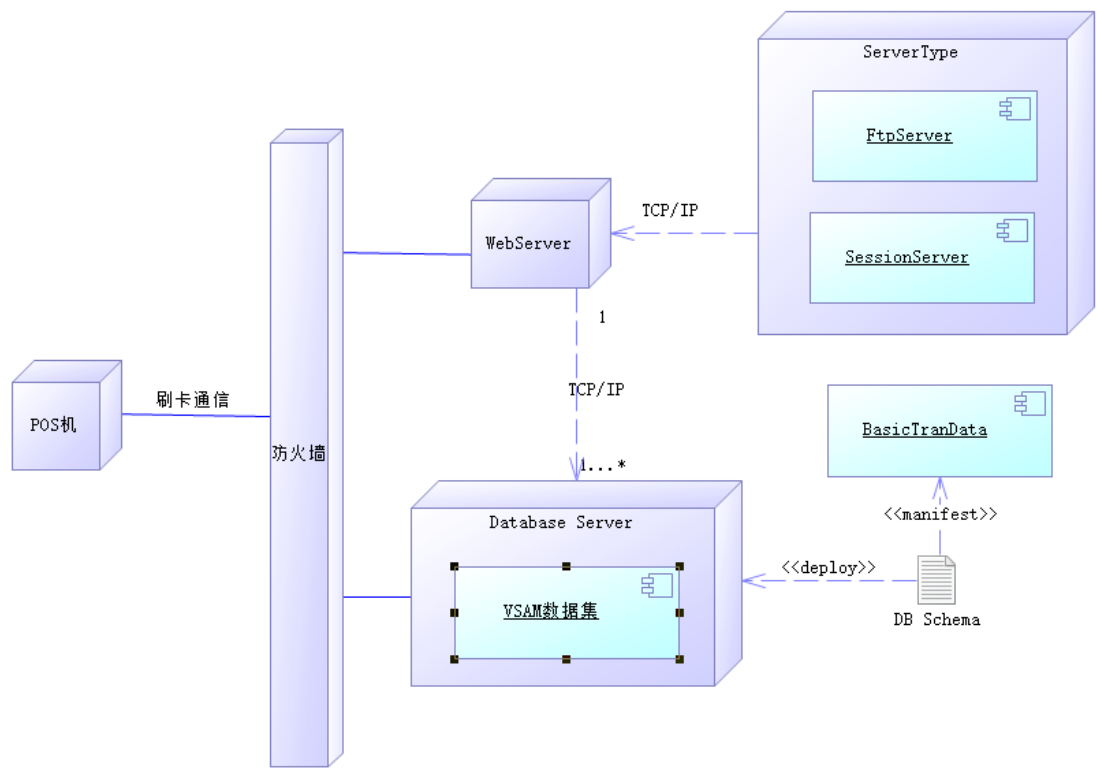


图 4-5 完整系统的部署图

从上图可以看出这个系统是一个典型的中间件系统，这里后台的数据库我们暂时考虑使用 VSAM 数据集。不过随着业务的拓展 VSAM 将不能满足系统功能的需要，我们会使用 DB2 数据库来替代它。

### 4.3 对性能的规定

如果只是针对批处理程序，对于我们这个系统来说，我们需要起满足以下的性能要求：

1. REXX 正确模拟用户交易数据，写入顺序数据集中
2. COBOL 逐行读取用户交易数据，监测用户“行为”
3. COBOL 返回用户交易日志，并判断用户交易是否可以继续进行下去

## 第五章 系统详细设计

### 5.1 总体流程图

根据系统的需求，将系统总体流程如下图 3.1 的所示：

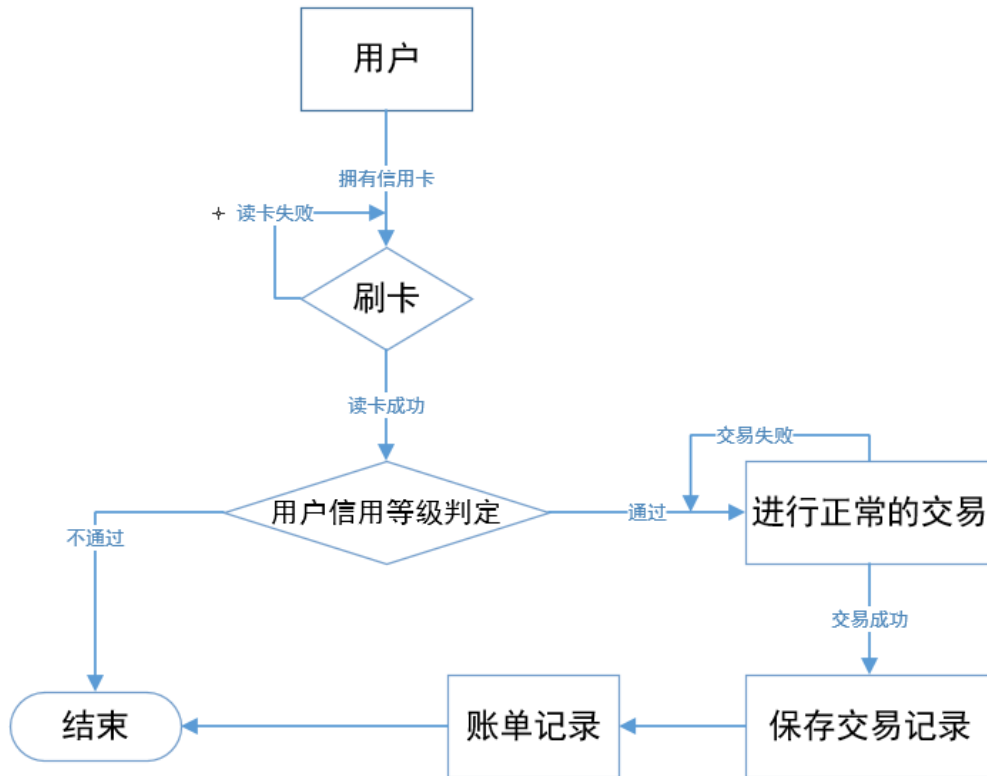


图 5-1 系统结构图

### 5.2 数据库设计

#### 5.2.1 数据库表设计

表 5-1 Interest 表

表名	Interest			
列名	数据类型 (精度范围)	空/非空	约束条件	其它说明
cardID	Varchar	非空	主键	信用卡 ID
TMdrawinterest	Long float	非空		本月取现利息
LMdebtinterest	Long float	非空		上月欠款利息
补充说明	该表用于保存信用卡利息信息			

表 5-2 发卡行信息表

表名	Bank			
列名	数据类型（精度范围）	空/非空	约束条件	其它说明
BankID	int	非空	主键	自动递增
BankName	varchar	非空		银行名称
BankAddress	varchar	非空		银行地址
CardType	varchar	非空		银行卡种类
BankPhone	int	非空		联系电话
补充说明	该表用于保存发卡银行的基本信息			

表 5-3 账户表

表名	Account			
列名	数据类型（精度范围）	空/非空	约束条件	其它说明
AccountID	int	非空	主键	账户 ID
AccountName	varchar	非空		账户持有人姓名
AccountNum	varchar	非空		账户卡号
补充说明	该表用于保存信用卡持有者账户信息			

表 5-4 账单表

表名	BankBill			
列名	数据类型（精度范围）	空/非空	约束条件	其它说明
TradeID	Varchar	非空	主键	交易 ID
TradeUser	Varchar	非空		用户名
TradeAmount	Varchar	非空		交易数额
TradeTime	Date	非空		交易时间
TradeType	Varchar	非空		交易类型
MaxLimit	Varcharate	非空		交易额度
补充说明	该表用于保存信用卡消费信息			

### 5.2.2、数据库表之间的关系示意图（见下一页）

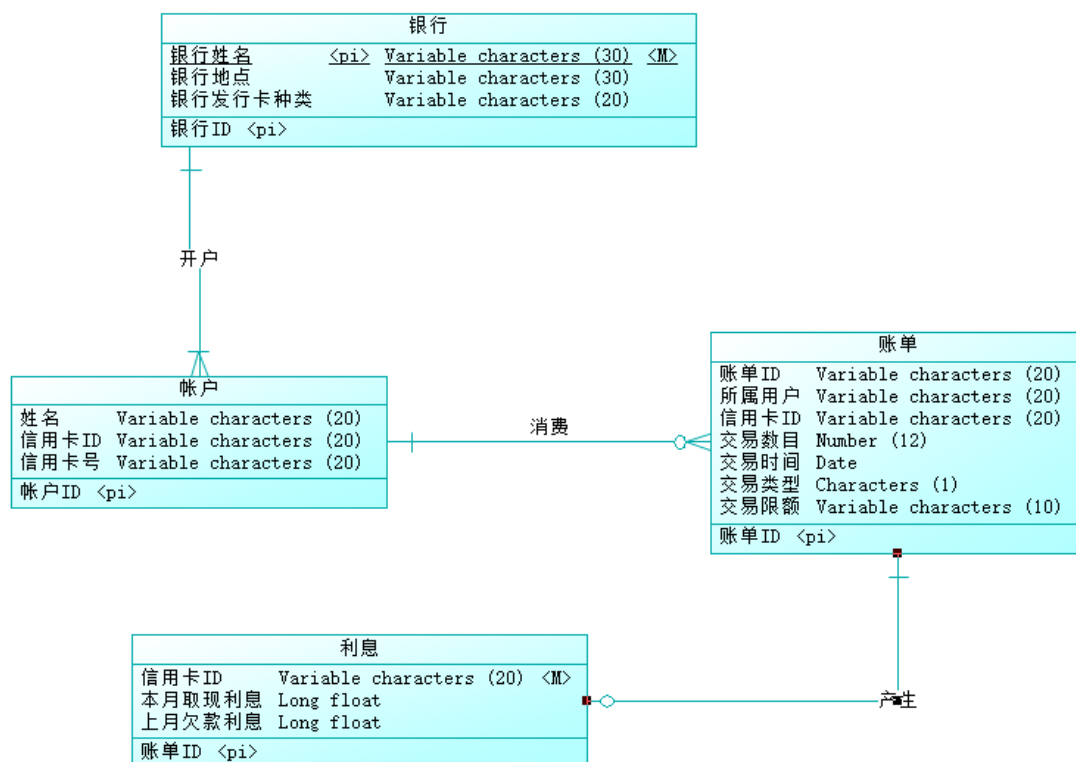


图 5-2 数据库关系图

## 5.3 静态结构建模

进一步分析系统需求，识别出类以及类之间的关系，确定它们的静态结构和动态行为，是面向对象分析的基本任务。系统的静态结构模型主要用类图或对象图来描述。

### 5.3.1 定义系统对象类

定义过系统需求，就可以根据系统需求来识别系统中所存在的对象。系统对象的识别可以通过寻找系统域描述和需求描述中的名词来进行，从前述的系统需求的描述中可以找到的名词有客户、信用卡、账单、消费记录。

## 5.3.2 建立类图

## 3.3.2.1 系统静态类图

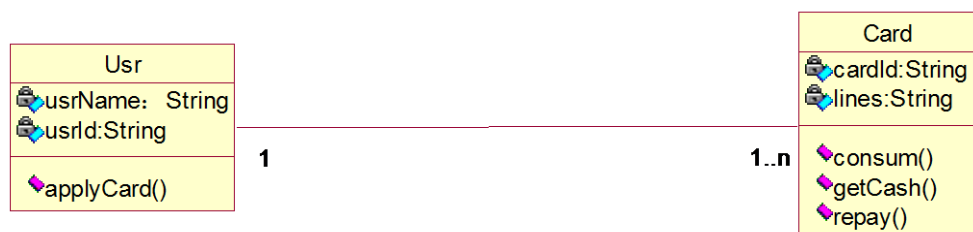


图 5-3 用户与信用卡关系



图 5-4 账单与账单输出文件结构

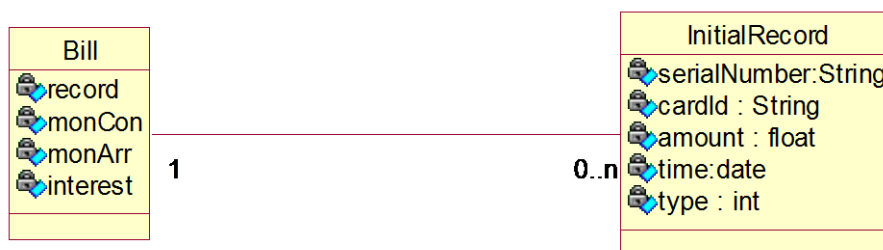


图 5-5 账单与消费记录



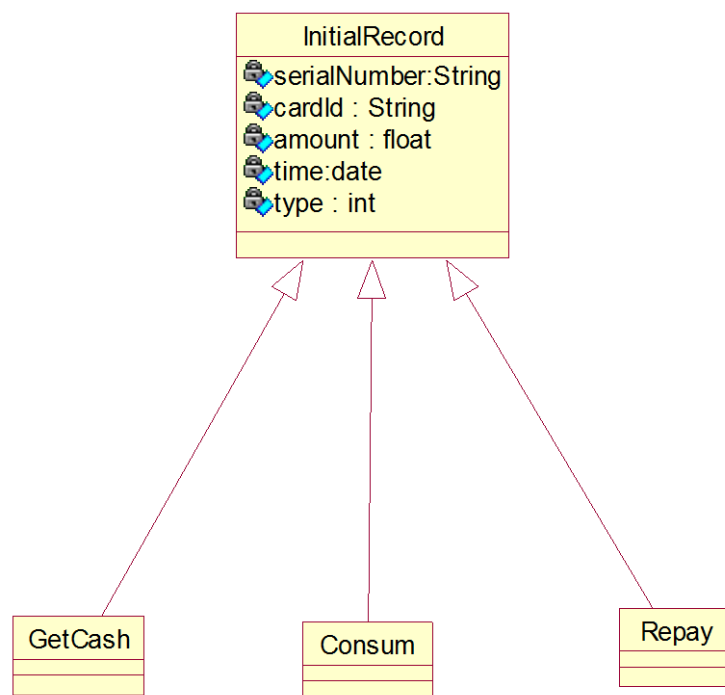


图 5-6 业务类型泛化

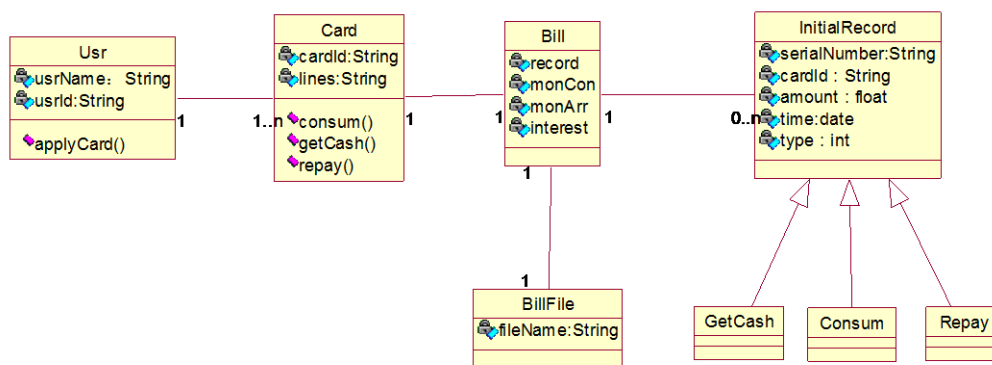


图 5-7 整体类图

## 5.4 动态行为建模

系统的动态行为模型可以用交互作用图、状态图和活动图来描述。活动图强调了从活动到活动的控制流，而交互图则强调从对象到对象的控制流，我们采用时序图来描述为完成某个特定功能发生在系统对象之间的信息交换。系统整体的

顺序图如下

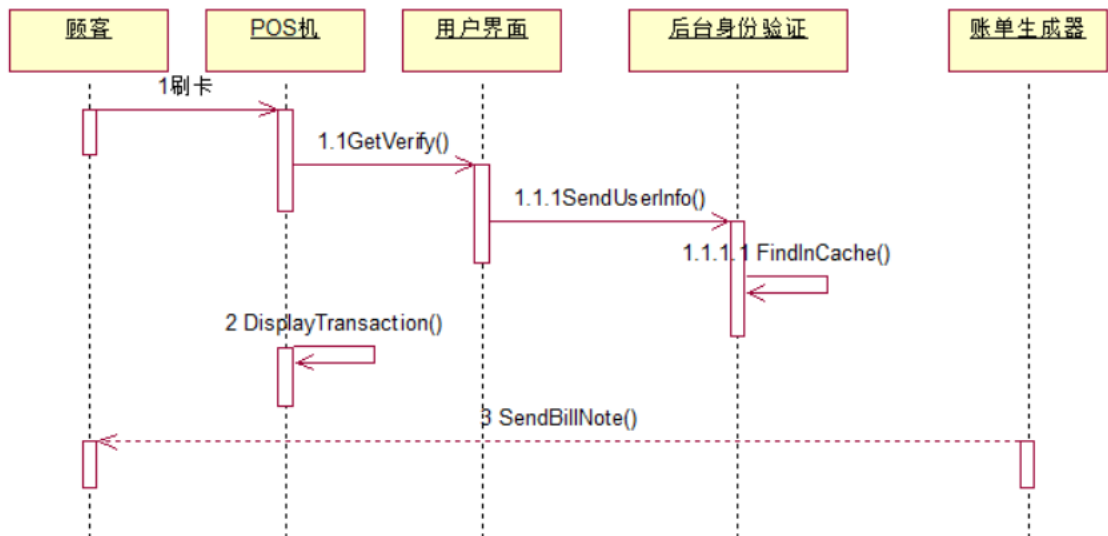


图 5-8 系统整体顺序图

## 第六章 系统实现

### 6.1、编写 REXX 生成测试数据

```
VIEW      XR307.CREDIT.REXX(CREDITB) - 01.99      Columns 00001 00072
==MSG>      your edit profile using the command RECOVERY ON.
000100 /***** REXX *****/
000200 TradeInfo.0=1
000300 SPEND.0 = 3
000400 SPEND.1 = 0
000500 SPEND.2 = 0
000600 SPEND.3 = 0
000601 Winterest.0 = 3
000602 Winterest.1 = 0
000603 Winterest.2 = 0
000604 Winterest.3 = 0
000610 YEAR = 2017
000620 MONTH = 01
000630 DAY = 01
000700 TradeInfo.USER.0 = 3
000800 TradeInfo.USER.1 = 'XR001'
000900 TradeInfo.USER.2 = 'XR002'
001000 TradeInfo.USER.3 = 'XR003'
Command ==>
F1=Help      F2=Split      F3=Exit      F5=Rfind      F6=Rchange      F7=Up
F8=Down      F9=Swap      F10=Left     F11=Right     F12=Cancel
MA A      英文 半形      08/025
```

图 6-1 REXX 源代码示意图

```
VIEW      XR307.CREDIT.REXX(DATAS1) - 01.00      Columns 00001 00072
000018 20170103XR0031 XR003 0084.01 20170103 1 3000
000019 20170104XR0011 XR001 0054.33 20170104 0 3000
000020 20170104XR0021 XR002 0092.35 20170104 0 3000
000021 20170104XR0031 XR003 0075.37 20170104 1 3000
000022 20170105XR0011 XR001 0091.01 20170105 0 3000
000023 20170105XR0021 XR002 0060.68 20170105 0 3000
000024 20170105XR0031 XR003 0087.20 20170105 0 3000
000025 20170106XR0011 XR001 0050.62 20170106 0 3000
000026 20170106XR0021 XR002 0087.29 20170106 0 3000
000027 20170106XR0031 XR003 0063.05 20170106 0 3000
000028 20170107XR0011 XR001 0078.99 20170107 0 3000
000029 20170107XR0021 XR002 0055.30 20170107 0 3000
000030 20170107XR0031 XR003 0082.15 20170107 0 3000
000031 20170108XR0011 XR001 0075.82 20170108 0 3000
000032 20170108XR0021 XR002 0065.75 20170108 0 3000
000033 20170108XR0031 XR003 0086.07 20170108 0 3000
000034 20170109XR0011 XR001 0072.30 20170109 0 3000
000035 20170109XR0021 XR002 0053.78 20170109 0 3000
Command ==>
F1=Help      F2=Split      F3=Exit      F5=Rfind      F6=Rchange      F7=Up
F8=Down      F9=Swap      F10=Left     F11=Right     F12=Cancel
MA A      英文 半形      22/015
```

图 6-2 生成的测试数据集

6.2、COBOL 处理后台业务逻辑

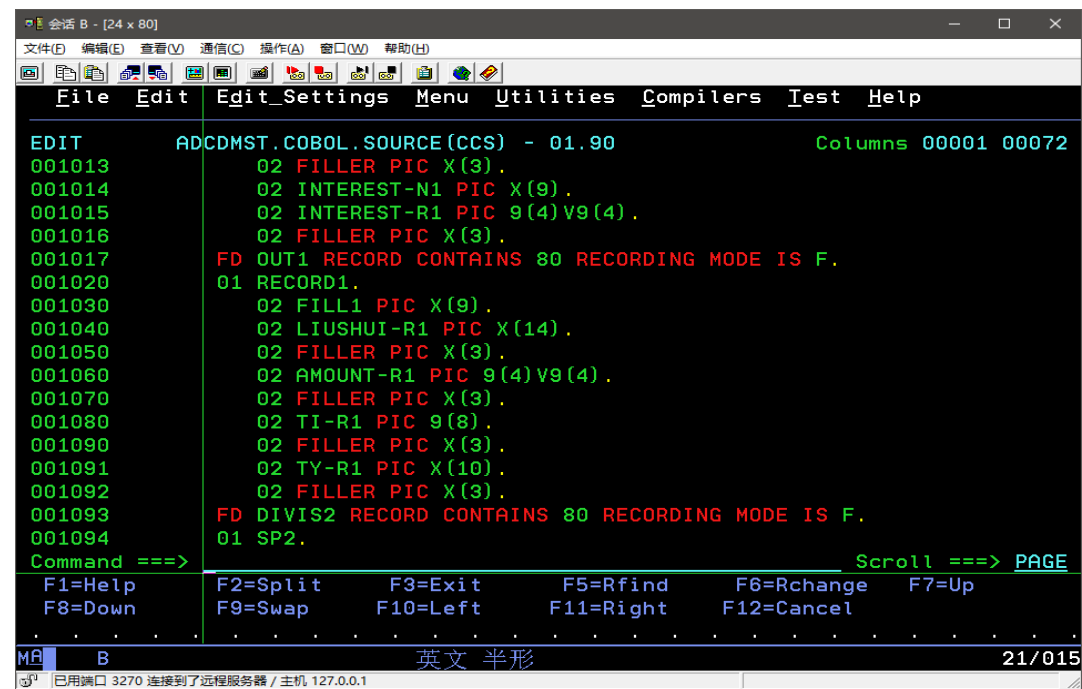


图 6-3 编写 COBOL 业务逻辑代码

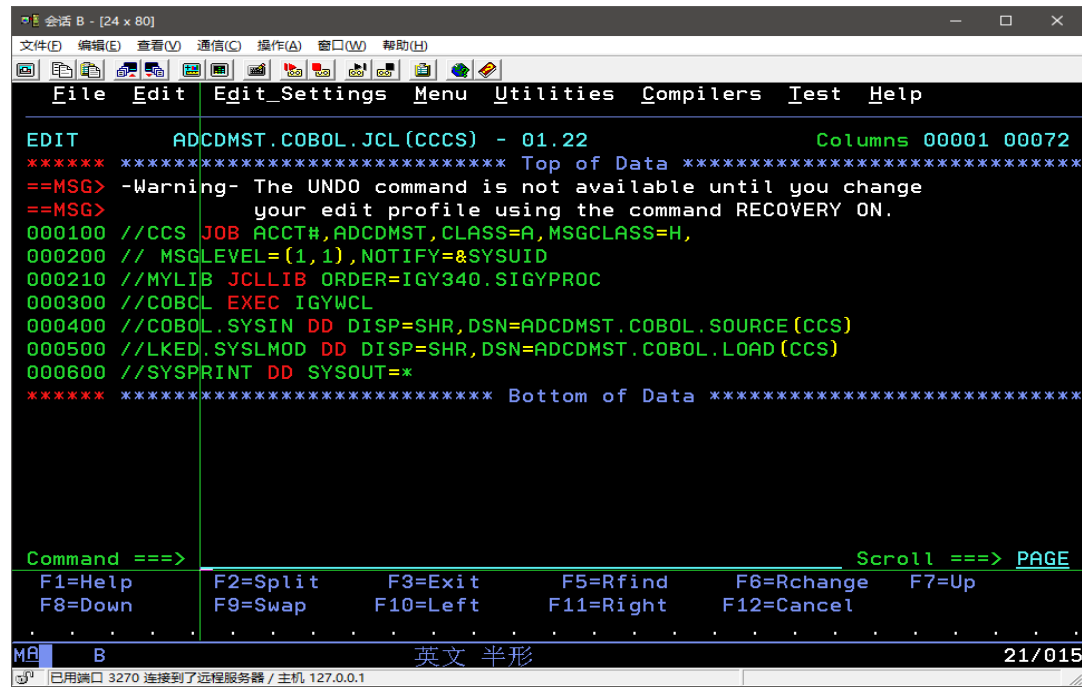


图 6-4 编写 JCL 编译 COBOL 代码

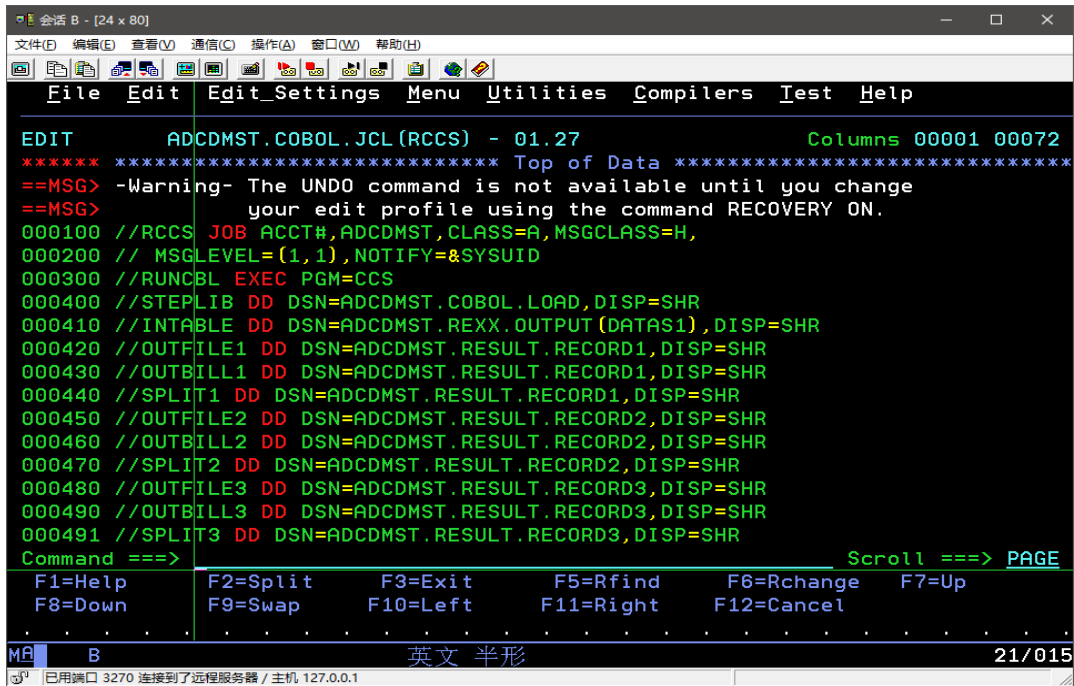


图 6-5 编写 JCL 编译 COBOL 代码

6.3、系统最终输出结果

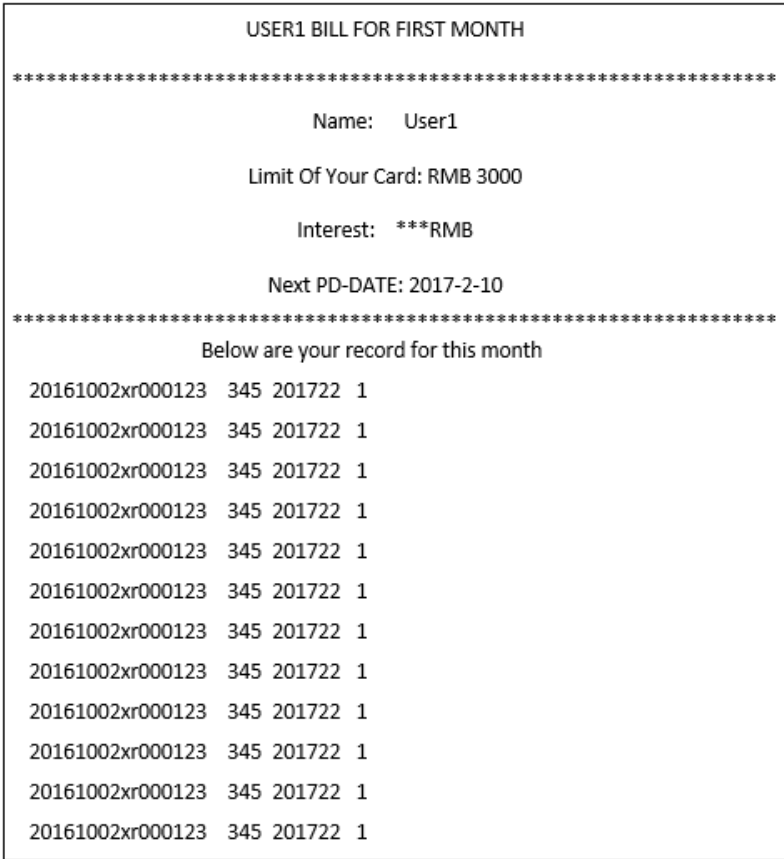


图 6-6 账单设计方案示意图

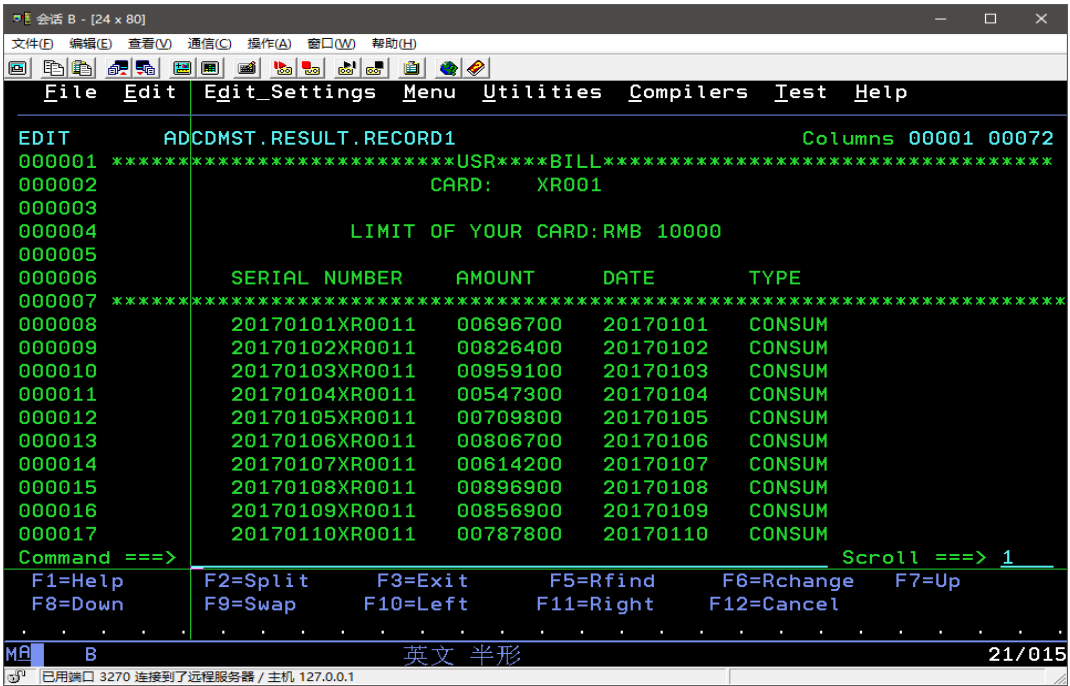


图 6-7 实际账单效果图 I

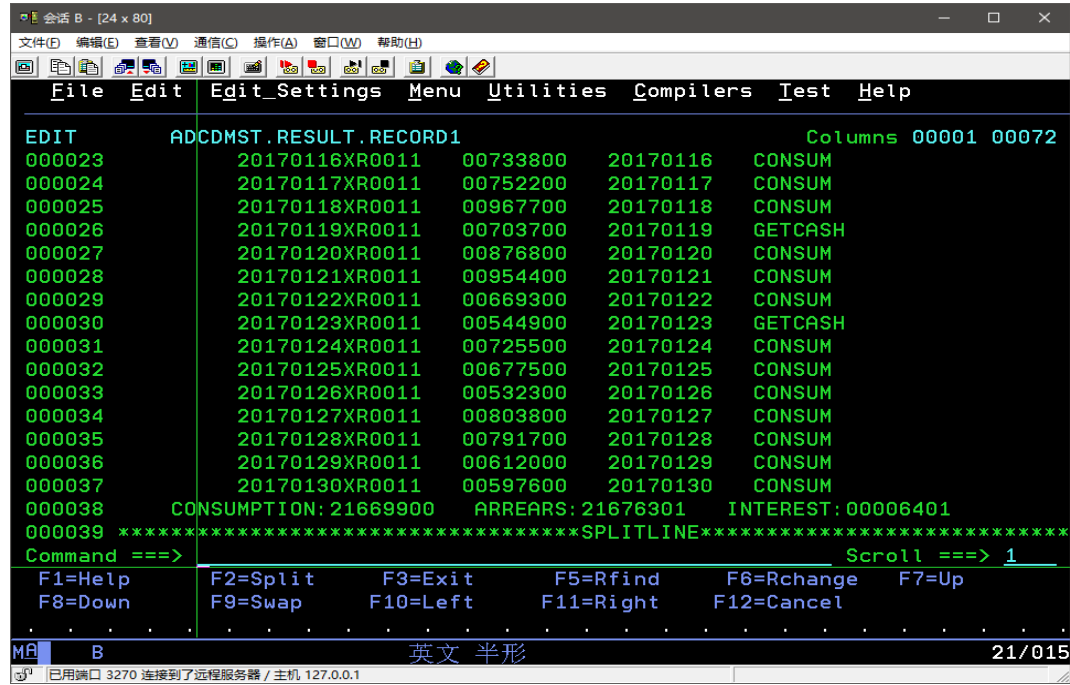
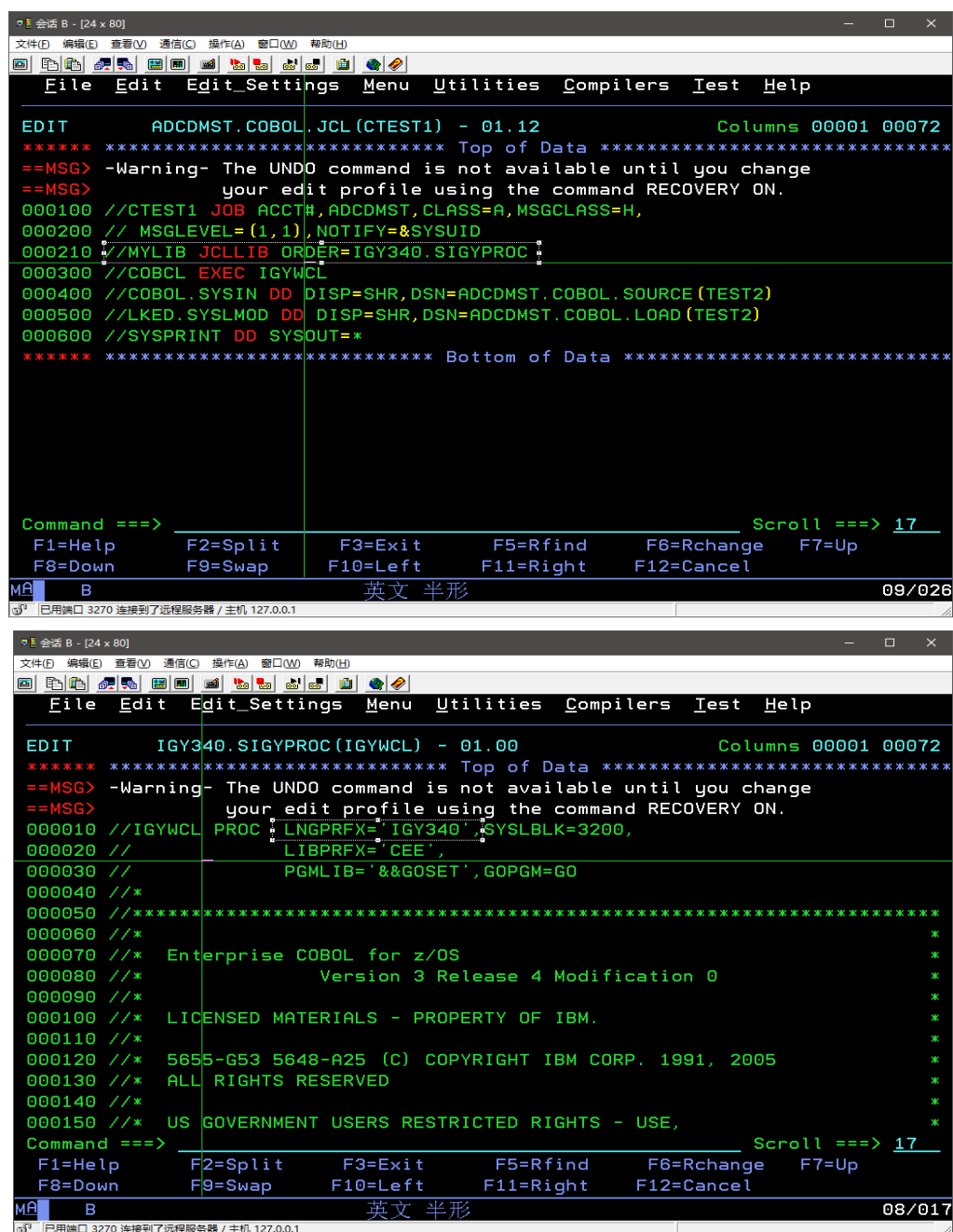


图 6-7 实际账单效果图 II

## 6.4、开发过程中遇到的问题和解决方案

1. 一开始我们是在学院的西 103 机房进行开发，后来由于实验室整体要搬迁的缘故，我们研究了 HELCULAR 的安装与配置的方法，由于没有现成的经验可以借鉴，我们在 IBM 技术官网上进行学习最终成功部署了 HELCULAR, 具体的部署方法我们专门写了一个文档在报告的附录里面。
2. 我们在安装好虚拟机后，发现写好的 COBOL 程序并不能正常执行，后来在老师的帮助下，我们找到了自己系统相关库的位置，对于虚拟机系统的文件进行了一些参数上的调整，最终实现 COBOL 在虚拟机上的执行。



```
EDIT      ADCDMST.COBOL.JCL (CTEST1) - 01.12      Columns 00001 00072
*****
==MSG> -Warning- The UNDO command is not available until you change
==MSG> your edit profile using the command RECOVERY ON.
000100 //CTEST1 JOB ACCT#,ADCDMST,CLASS=A,MSGCLASS=H,
000200 // MSGLEVEL=(1,1),NOTIFY=&SYSUID
000210 //MYLIB JCLLIB ORDER=IGY340.SIGYPROC
000300 //COBCL EXEC IGYWCL
000400 //COBOL.SYSIN DD DISP=SHR,DSN=ADCDMST.COBOL.SOURCE (TEST2)
000500 //LKED.SYSLMOD DD DISP=SHR,DSN=ADCDMST.COBOL.LOAD (TEST2)
000600 //SYSPRINT DD SYSOUT=*
*****

Command ==>
F1=Help      F2=Split      F3=Exit      F5=Rfind      F6=Rchange      F7=Up
F8=Down      F9=Swap      F10=Left     F11=Right     F12=Cancel

MA B 英文 半形 09/026
已用端口 3270 连接到了远程服务器 / 主机 127.0.0.1

EDIT      IGY340.SIGYPROC (IGYWCL) - 01.00      Columns 00001 00072
*****
==MSG> -Warning- The UNDO command is not available until you change
==MSG> your edit profile using the command RECOVERY ON.
000010 //IGYWCL PROC LNGPRFX='IGY340',SYSLBLK=3200,
000020 //
000030 //      PGMLIB='&&GOSET',GOPGM=GO
000040 //
000050 //*****
000060 //
000070 //* Enterprise COBOL for z/OS
000080 //* Version 3 Release 4 Modification 0
000090 //*
000100 //* LICENSED MATERIALS - PROPERTY OF IBM.
000110 //*
000120 //* 5655-G53 5648-A25 (C) COPYRIGHT IBM CORP. 1991, 2005
000130 //* ALL RIGHTS RESERVED
000140 //*
000150 //* US GOVERNMENT USERS RESTRICTED RIGHTS - USE,
Command ==>
F1=Help      F2=Split      F3=Exit      F5=Rfind      F6=Rchange      F7=Up
F8=Down      F9=Swap      F10=Left     F11=Right     F12=Cancel

MA B 英文 半形 08/017
已用端口 3270 连接到了远程服务器 / 主机 127.0.0.1
```

图 6-8 配置编译器位置

3. 我们第一次在虚拟机上运行 COBOL 代码的时候，由于循环体内 SET TX TO 1 这条语句的存在，导致程序陷入无限循环无法退出，由于虚拟机的日志和缓存被塞满，我们无法重新登陆，实验也无法进行下去。我们也第一时间咨询了指导老师，再结合自己从 IBM 官网上找到的资料，最终选择冷启动的方式解决这个问题。

在启动 z/OS 的时候，本文所用的启动参数是 0A8032M1，这里的 32 代表冷启动，启动了 TSO，CICS，DB2，WAS，JES2。也可以将这里的 32 换成以下参数来根据需要启动不同的组件（见附录）；冷启动之后，系统里的日志将会被清空，即可再次提交作业和登录系统。

图 6-9 系统缓存和日志塞满

图 6-10 重新登陆虚拟机



## 第七章 系统测试

### 7.1 测试产品信息

产品或系统模块名称：本次测试主要是针对信用卡后台业务子系统的一些特定模块，主要包括：

- 1、文件处理模块
  - a) 业务数据生成
  - b) 业务数据读入
  - c) 账单数据生成
- 2、账单计算模块
  - a) 账单计算准确性
  - b) 消费类型的覆盖

版本信息：V0.1，本次测试主要是针对信用卡后台业务子系统进行简单测试。

### 7.2 测试内容

表 7-1 测试计划

测试计划	原测试内容	覆盖与否	测试的情况	未成功覆盖的原因
非功能性测试	账单数据是否满足用户需要	是	确保输出账单的数据能够保证用户的需要	
功能测试	业务数据的生成	是	确保业务数据按数据规则成功生成	
	业务数据的读入	是	确保业务数据能够成功读入程序	
	账单数据生成	是	确保账单数据能够按规则成功生成	
	账单计算的准确性	是	确保账单的计算准确无误	
	消费类型的覆盖	是	确保能够覆盖所有消费情况	

## 7.3 测试用例

表 7-2 测试用例

用例名	前置条件	操作步骤	输入数据集	预期结果	实际结果
生成业务数据	登录系统成功	1、配置文件输出目录 2、选择源文件执行	无	业务数据按规定格式生成成功	通过
读入业务数据	业务数据文件已经成功生成并且链接到源程序	1、在源程序中创建数组 2、数组接受读入的业务数据并检查数据正确性	程序中需要输入业务数据源文件	业务数据源文件读取成功并且成功按预期形式保存到内存	通过
账单将计算的准确性	业务数据文件读取成功	1、执行计算账单的程序模块 2、与生成的账单核对计算结果	存储业务数据的数组	账单中每月的利息、欠款、消费等信息计算准确。	在月初到最低还款日执行还款和不还款大的情况下，其余情况不能正确识别还款操作
覆盖所有消费类型	业务数据文件读取成功	1、输入包含不同消费类型的业务数据 2、查看输出账单是否包正确判断消费	存储业务数据的数组	有业务数据中的每一种消费类型都被正确判断且正确计算账单	通过

		类型			
生成账单数据	账单计算成功	执行账单导出模块，将账单输出到账单文件	账单的目标文件路径	每个用户的账单被成功追加到账单文件	通过
账单数据满足用户需要的信息	账单文件成功导出	进入相应数据集查看处理账单	无	在账单文件中可以看到每个月自己的消费流水记录和每月欠款以及利息	通过

## 7.4 遗留缺陷和建议

表 7-3 缺陷与建议

缺陷	建议
在账单文件中由于输出的数字的类型，小数点没有显示	将数字类型转换为字符串后再进行输出
需要预先知道业务文件的行数再进行读取	1、使用可变长数组存取业务数据 2、不使用数组存取业务数据，逐行读取
代码冗余程度高，占用了不必要的空间	使用子程序进行代码复用
每个用户都需要定义一个文件进行存储账单，当用户量增加后代码会很冗余	引入 Z/OS DB2 数据库控制对用户业务的访问
缺少对用户的还款提示	在账单中加入最低还款日以及最低还款金额

## 第八章 项目核心技术

### 8.1 REXX 语言

REXX (Restructured Extended Executor) 是 IBM 在上个世纪 80 年代发明的一种程序设计语言, 主要用在 IBM 的大型计算机 (Mainframe Computer) 上。

REXX 不同于 COBOL、JAVA 等高级语言, 它是解释型的, 无需编译的一种脚本语言。它的书写风格比较自由, 语句简单易懂, 开发人员上手比较快。一开始我们是在 Windows 系统下把数据准备好再利用 FTP 传到大机上, 后来发现我们可以直接再 Z/OS 下进行 REXX 开发。这一转变也体现了 REXX 多平台的特性。

### 8.2 COBOL 语言

COBOL (Common Business Oriented Language) 是第一个广泛使用的高级编程语言。它主要是为了解决经企管理的问题。COBOL 重视数据项和输入输出记录的处理, 对具有大量数据文件提供了简单的处理方式。

在本项目的开发中, 尽管 REXX 具有很多强大的文本处理任务的工具, 但是我们考虑到后台系统业务设计的连贯性, 所以逻辑实现和账单生成这两大功能均是采用 COBOL 来设计的, 事实也证明因为数据项等特有的数据格式的存在, COBOL 处理文本数据同样出色。

### 8.3 中间件

中间件是一种独立的系统软件或服务程序, 是连接两个独立应用程序或独立系统的软件。通过中间件, 不同层软件之间可以通过接口交换信息, 应用程序可以实现工作于多平台或者 OS 环境。

中间件的思想更多是应用在前瞻性的设计中, 因为本次开发并没有涉及到客户端、服务器等应用。中间件作为一种设计模式, 应该是这个项目未来深入开发指导思想。

## 第九章 开发心得

### 9.1 成员心得

“本次项目给我带来的最大感触就是工程项目的负责人光有组织和规划能力是不够的，扎实的专业技能也很有必要，因为你只有了解这项技术，你才清楚项目某一部分的工作量大小、难度、技术特点，从而帮助你更加合理地分配任务。任务安排不合理不仅会降低项目开发效率而且会有损小组成员工作的积极性。

此外，在项目开发的过程中，一个组长要随时做好冲锋向前，接替队友工作的准备。这里的意思是说，一个组长既要做带头人，为队友们更加深入的工作打基础，还要在队友没有时间的时候能够迅速用自己的努力去弥补工作的空缺部分。

总之，此次开发经历不仅让我对于软件开发与设计的过程有了更加深刻的认识，夯实了自己的专业技能，我还体悟到如何去做一名合格的、负责任的项目组长。”

----- 杨林彬

“在模拟数据这一块，开发过程还是比较顺利的，最后根据后台业务处理编码的需求，不断的修改格式，最终完成了模拟数据的开发与设计。

模拟数据编码的过程中主要遇到的几个困难及解决方法：

1. 频繁的修改数据的格式，造成这个问题的主要是由于需求的不清晰以及 COBOL 语言处理数据的方式不熟练。解决的方法是采用敏捷开发模型，不断的迭代，逐渐形成可用的数据格式，并增删有无用的数据信息。
2. 从 Windows 到 Z/os 运行环境的转变导致 REXX 程序片段的不可执行。由于之前没有意识到 REXX 还可以在大机环境下编写，模拟数据的程序是在 Windows 环境下运行的，而 REXX 在 Windows 操作系统读写文件的方法与在 Z/os 下操作数据集有所区别，导致转移环境后，原来的代码不可用，需要重新编写读写文件相关信息。
3. REXX 复合变量的处理问题。REXX 的复合变量使用起来十分的灵活，而且下标可以是数字也可以是任何有意义的字符串，但可能造成数据的混淆，导致空值的出现，在实际的开发过程中就需要进行区分与组合。

本次的综合设计经过了需求分析，系统设计，系统实现，系统测试等阶段，完成了对后天业务子系统的完整开发，虽说工程量不是很大，但是通过这个过程，进一步加深了我对实际项目的开发过程理解，项目团队合作之间的重要性的认识，在项目实际开发的过程中，在杨林彬队长的领导下，我们不定时的组织开会，发表各自的意见和看法，讨论项目的进展，讨论项目关键部分的解决办法，比如在设计项目的核心也就是信用卡具体利息计算的方法中，我们首先通过讨论理解信用卡的结算过程以及利息计算规则，之后再去看资料了解实际银行的利息计算，再设计出详细的计算逻辑，使用模拟用例加以理解，成员发表个人的看法再加以完善，最后达成最终的计算逻辑。同时也加强了项目的时间观念，在项目开发的过程中需要规划好时间，在不同的时期尽量完成该时期的任务，以加强项目计划执行力度，以及减少项目风险。

我在这次项目中主要完成消费记录数据的生成以及文件的读写，在开发过程中也遇到了一些问题，不过通过学习开发工具 REXX 与 COBOL，加深了对其的理

解，不断编译调试解决了其中的问题，完成了模拟数据的生成与测试，同时也为以后的项目开发打下基础。

在开始一个完整的综合设计课题时，需要对其进行拆分，按照软件工程的流程进行开发，可以让开发过程更加的清晰，更加的快速与便捷。”

-----郑伟

“在开发该系统的过程中，我主要负责开发对原始记录进行利息计算和账单生成的 COBOL 程序的开发，在开发过程中主要有以下心得体会：

1. 开发首先需要明确问题：
  - a) 明确问题即明确需求是什么，需要满足的需要是什么，在开发初期，由于没有明确的需求，仅仅将该项目理解为读入数据，计算利息；对于利息的计算方法和程序内部逻辑都没有一致的标准，从而导致了开发进度缓慢，效率低下等问题，因此开发首先需要明确需求和面对的问题；
2. 文档的重要性：
  - a) 开发是一个持续性和开发成员互相合作的过程，仅凭口头交流难以达成开发过程中高度配合，因此需要在开发过程中建立文档以保持自己对以前的工作进行记录，开发的过程中不可能记住自己所做过的每一项工作，通过文档，可以让合作人员也可以让自己方便的追踪项目进度。
3. 团队合作的重要性：
  - a) 在开发过程中，一个人的思想可能对项目的实际情况考虑有所欠缺，因此需要团队合作进行思想的交流和碰撞，才不会囿于自己的定时思维之中。比如关于利息的计算方法，我们每个人最开始都有一种方案，最后大家通过互相交流分析，发现大家的算法都有所欠缺，最后结合所有人的想法形成了一个相对完善的算法。所以团队合作在开发过程中是不可或缺的。”

-----罗阳星

## 参考文献

- [1]信用卡基本常识[EB/OL].(2016-11-25). <http://jingyan.baidu.com/article/4ae03de31.html>
- [2]IBM 官网论坛[EB/OL].(2016-11-25). [www.ibm.com](http://www.ibm.com)
- [3]大型机在 Windows 下环境的搭建[EB/OL].(2016-11-25). <http://blog.csdn.net/tuliangde/article/details>

## 附录 I 冷启动参数和 COBOL 操作文件

## 冷启动参数

表 附录-1

冷启动参数	参数含义
CS	CLPA and cold start of JES2. Base z/OS system functions i.e. no CICS, DB2, IMS, WAS, etc.
00	Warm start of JES2. Base z/OS system functions i.e. no CICS, DB2, IMS, WAS, etc.
WS	Warm start of JES2. Base z/OS system functions i.e. no CICS, DB2, IMS, WAS, etc.
DC	CLPA, brings in CICS LPA modules, cold start of JES2, starts up DB2 and CICS.
DB	Warm start of JES2 and starts the DB2 and CICS.
DI	CLPA and cold start of JES2 and loads the IMS Libraries. IMS must be manually started.
CC	CLPA and cold start of JES2, loads the CICS Libraries, starts up CICS, no DB2.
CW	Warm start of JES2, and starts up CICS.
7C	CLPA, cold start of JES2, starts up DB2 V7, no CICS.
7W	Warm start of JES2, starts up DB2 V7, no CICS.
8C	CLPA, cold start of JES2, starts up DB2 v8, no CICS.
8W	Warm start of JES2, starts up DB2 v8, no CICS.
IC	CLPA and cold start of JES2 and load the IMS Libraries, start IMS, no DB2 or CICS.
IW	Warm start of JES2 start IMS, no DB2 or CICS.
AC	CLPA and cold start of JES2 load IMS and CICS libraries, start IMS, DB/2, and CICS.
AW	Warm start of JES2. start IMS, DB/2, and CICS.
BC	CLPA and cold start of JES2, load WAS libraries, WAS is manually started
BW	Warm start of JES2. WAS is manually started.
99	Points to IODF99 for IPL on MP3000. Reply 00,SYSP=xx were xx is any of the above options i.e. for cics only



## COBOL 操作文件

我们在使用 COBOL 写文件的时候，曾经遇到 SOC4 ABEND 的异常，困扰了我们很久，后来在 IBM 官网上查阅资料后

SOC4 是在虚拟地址与物理地址不能映射时发生的保护异常

以下的错误会导致 SOC4 异常的发生：

1. 由于下标错误而引用的无效地址。
2. 在组移动中，接收字段的长度定义不正确。
3. 移动可变长度记录，大于接收字段的长度。
4. 读/写在程序中未打开的文件。
5. 在 EOF 之后读/写文件。

经过分析源程序，出现该错误的原因是没有在 COBOL 程序中执行打开文件的操作，就对文件中的变量进行赋值，所以导致了变量的虚拟地址不能映射到文件物理地址，即未打开文件就进行了读写操作。以下是 COBOL 中文件的打开模式以及应用场景

表 附录-2

File Access Mode	OPEN Mode				
	Statement	Input	Output	Input-Output	Extend
Sequential	READ	X		X	
	WRITE		X		X
	REWRITE			X	
Random (Non-Sequential Files)	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START				
	DELETE			X	
Dynamic (Non-Sequential Files)	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START	X		X	
	DELETE			X	

## 附录 II 虚拟机的安装<sup>[3]</sup>

IBM 主机系统可以借助仿真软件 Hercules 在 PC 上面运行。在安装主机系统之前，必须遵循 IBM 对于主机软件的有关规定。

### 1、需要的相关软件 and 资源：

- a) 一台 PC 机，1G 内存以上，2GHz CPU, 硬盘空间 10G
- b) ADCD z/OS 系统的卷文件：
- c) 3270 客户端：IBM Personal Communication，简称 PCOM，该软件只能运行于 Windows 平台，并且功能强大，也是最常用的 3270 客户端
- d) Hercules
- e) TCP/IP 支持软件

### 2、为了方便管理，约定好文件放置和软件的安装路径

表 附-3

D:/ADCDV1R6	所有与模拟环境相关的东西都放在这个目录下
D:/ADCDV1R6/ZOSV1R6	存放卷文件，即 CCKD 文件
D:/ADCDV1R6/Config	存放启动配置文件
D:/ADCDV1R6/HercGUI-1.11.1	存放 Hercules GUI 和 FishLib
D:/ADCDV1R6/hercules-3.07	存放 Hercules 的可执行文件和 CTCI-W32
D:/ADCDV1R6/Log	存放系统运行日志

### 3、配置 ZOS19.cnf，设置启动虚拟机时的配置

在输入配置文件时，需要注意以下几个字段，

- a) LOADPARM 是 z/OS 在 IPL 时需要用到的参数，参数选择不同，z/OS 中启动后，运行的组件也不同。
- b) MAINSIZE 表示 Hercules 占用的物理内存大小，单位是 MB，如果内存够大，最好设为 1024，在这里暂且设为 832，建议不要小于 512，否则 MIPS 会很低，系统运行会非常的慢。
- c) NUMCPU 表示虚拟的主机有多少个 CPU。
- d) Display Terminals 表示 3270 终端的数量，演示所用的 3270 终端号是从 0700 开始的，以十六进制记录，如 0700-0710 则表示 16 个端口，其中 0700 是控制台专用端口，0701 至 0710 为用户连接端口。
- e) DASD Device，要把所有下载的 CCKD 文件都列进去，这一段的每一行分为三个部分，第一个是设备编号，如 0A80，第二个是设备类型，如 3390，第三个是卷文件的路径和文件名，这里可以写绝对路径也可以是相对路径。

### 4、安装 3270 仿真软件 PCOMM

- a) 运行 3270 仿真软件，在通信菜单→配置为：主机类型：zSeries 或 OS390，接口：LAN，连接：Telnet3270，链路参数：IP 地址：（主要）127.0.0.1，端口：3270。一般运行至少有 2 个 terminal，通常其中一个为 3270 Console（控制大机用的），其它为 TSO Terminal。

- b) 登录 TSO 时会弹出打印窗口，可以通过下面设置去掉：

在 PCOMM 中，选择 File -> Save As... 你就可以看到你的配置文件(.ws)保存的地方。使用记事本打开 pcomm 连接配置文件，并在此文件最后加入以下内容：

```
[LT]
IgnoreWCCStartPrint=Y
UndefinedCode=Y
UndefinedDBCSChar=Y
```

## 5、启动大机

- a) 在 HercGUI 窗口中直接点 Power ON（灰白色按钮）
- b) 选择配置文件（前面写好的 zOS19.cnf）
- c) 点确定
- d) 选 Yes，点 OK
- e) 打开两个 PCOM 窗口，可以看到 Hercules 界面，这时还不能操作，点 HercGUI 窗口的右上角的 Load（蓝色按钮），第一次点时会弹出一个 IPL 窗口，填以下内容：

Device Number:0A80

Load Parm:0A8299M1

勾上 Don't ask me again

- f) 这时大机就开始启动了，等待其中一个 PCOM 窗口（通常为第 2 个）显示 z/OS 界面就可以登录使用了，启动过程信息可以在其中一个 PCOM 窗口（通常为第 1 个）中看到。在启动 hercules 时，系统可能出现各种需要应答的信息，具体情况要具体分析，不同信息的处理方式是不同的，需要根据提示进行应答。
- g) 登录 TSO：默认用户为 ADCDMST，密码：ADCDMST

上述步骤中提到的配置文件 zos19.cnf 如下所示：

```
#
# Hercules Emulator Control file...
# Description: z/OS 1.9 Created by XiaoCai
# MaxShutdownSecs: 15
#
# System parameters
#
ARCHMODE z/Arch
ALRF      ENABLE
CNSLPORT  3270
CONKPALV  (3,1,10)
CPUMODEL  3090
CPUSERIAL 012345
DIAG8CMD  ENABLE
```

```

LOADPARM 0A82CSM1
LPARNAME HERCULES
MAINSIZE 256
MOUNTED_TAPE_REINIT DISALLOW
NUMCPU 4
OSTAILOR Z/OS
PANRATE 80
PGMPRDOS LICENSED
SHCMDOPT NODIAG8
SYSEPOCH 1900
TIMERINT 50
TZOFFSET +1400
YROFFSET 0
HERCPRIO 0
TODPRIO -20
DEVPRIO 8
CPUPRIO 0
PANTITLE "z/OS 1.9 IPL A80"
# Display Terminals
0700 3270
0701 3270
0702 3270
0703 3270
# DASD Devices
0A80 3390 D:\ADCDV1R6\ZOS1.9\Z9RES1.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9RES1_Shadow.CCKD
0A81 3390 D:\ADCDV1R6\ZOS1.9\Z9RES2.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9RES2_Shadow.CCKD
0A82 3390 D:\ADCDV1R6\ZOS1.9\Z9SYS1.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9SYS1_Shadow.CCKD
0A83 3390 D:\ADCDV1R6\ZOS1.9\Z9RES3.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9RES3_Shadow.CCKD
0A84 3390 D:\ADCDV1R6\ZOS1.9\Z9USS1.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9USS1_Shadow.CCKD
0A85 3390 D:\ADCDV1R6\ZOS1.9\Z9PRD1.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9PRD1_Shadow.CCKD
0A86 3390 D:\ADCDV1R6\ZOS1.9\Z9DIS1.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9DIS1_Shadow.CCKD
0A87 3390 D:\ADCDV1R6\ZOS1.9\Z9DIS2.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9DIS2_Shadow.CCKD

```

0A88	3390	D:\ADCDV1R6\ZOS1.9\Z9DIS3.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9DIS3_Shadow.CCKD		
0A89	3390	D:\ADCDV1R6\ZOS1.9\Z9DIS4.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9DIS4_Shadow.CCKD		
0A8A	3390	D:\ADCDV1R6\ZOS1.9\Z9DIS5.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9DIS5_Shadow.CCKD		
0A8B	3390	D:\ADCDV1R6\ZOS1.9\Z9DIS6.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9DIS6_Shadow.CCKD		
0A8C	3390	D:\ADCDV1R6\ZOS1.9\SARES1.CCKD
sf=D:\ADCDV1R6\ZOS1.9\SARES1_Shadow.CCKD		
0A8D	3390	D:\ADCDV1R6\ZOS1.9\Z9CIC1.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9CIC1_Shadow.CCKD		
0A8E	3390	D:\ADCDV1R6\ZOS1.9\Z9DB81.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9DB81_Shadow.CCKD		
0A8F	3390	D:\ADCDV1R6\ZOS1.9\Z9DB82.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9DB82_Shadow.CCKD		
0A90	3390	D:\ADCDV1R6\ZOS1.9\Z9DB91.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9DB91_Shadow.CCKD		
0A91	3390	D:\ADCDV1R6\ZOS1.9\Z9DB92.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9DB92_Shadow.CCKD		
0A92	3390	D:\ADCDV1R6\ZOS1.9\Z9IMS1.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9IMS1_Shadow.CCKD		
0A93	3390	D:\ADCDV1R6\ZOS1.9\Z9WAS1.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9WAS1_Shadow.CCKD		
0A94	3390	D:\ADCDV1R6\ZOS1.9\Z9WAS2.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9WAS2_Shadow.CCKD		
0A95	3390	D:\ADCDV1R6\ZOS1.9\Z9WAS3.CCKD
sf=D:\ADCDV1R6\ZOS1.9\Z9WAS3_Shadow.CCKD		
# CTC Adapters		
0E20	LCS	-n 192.168.1.2 -m 00-00-5E-90-09-5D 192.168.1.5
0E21	LCS	-n 192.168.1.2 -m 00-00-5E-90-09-5D 192.168.1.5

## 附录III 部分编码

## 模拟数据编码

```

/***** REXX *****/
TradeInfo.0=1
SPEND.0 = 3
SPEND. = 0
Winterest.0 = 3
Winterest. = 0
YEAR = 2017 ; MONTH = 01 ; DAY = 01
TradeInfo.USER.0 = 3
TradeInfo.USER.1 = 'XR001'
TradeInfo.USER.2 = 'XR002'
TradeInfo.USER.3 = 'XR003'
K.0=3
L=1

/**** Set the output file ****/
"ALLOC DA('XR307.CREDIT.REXX(DATAS1)') F(OUTADATA) OLD REUSE"

/**** Generate data ****/
DO I=1 TO 90
  K.1=1
  K.2=1
  K.3=1
  TradeDay = YEAR||MONTH||DAY
  DO
    DO U=1 TO 3
      IF SPEND.U < 3000 THEN
        /**** sets the data format ****/
        TradeInfo.L.TradeID = TradeDay || ,
        RIGHT(TradeInfo.USER.U,5) || RIGHT(K.U,1,'0')
        TradeInfo.L.TradeUser = TradeInfo.USER.U
        TradeInfo.L.TradeAmount=RANDOM(50,99)+(RANDOM(0,99))/100
        SPEND.U = SPEND.U + TradeInfo.L.TradeAmount
        TradeInfo.L.TradeTime = TradeDay
        IF random(1,10) > 1 then
          TradeInfo.L.TradeType = 0
        ELSE
          TradeInfo.L.TradeType = 1
        TradeInfo.L.MaxLimit = 3000
        TradeInfo.0 = TradeInfo.0+1
        K.U = K.U + 1
    END
  END
END

```

```

        IF SPEND.U > 3000 THEN
            DO
                SPEND.U = SPEND.U - TradeInfo.L.TradeAmount
                TradeInfo.L.TradeAmount = 3000 - SPEND.U
                SPEND.U = 3000
            END
            IF TradeInfo.L.TradeType = 1 THEN
                /***** Interest *****/
                DO
                    Atemp = TradeInfo.L.TradeAmount
                    Thismonthlasttime = YEAR||MONTH||31
                    Itemp = Atemp * 0.0005 * (Thismonthlasttime - TradeDay)
                    Winterest.U = Itemp + Winterest.U
                END
                L = L+1
            END
        END
        END
        END
        /**** total Amount ****/
    IF SUBSTR(TradeDay,7,2) = '30' then
        DO
            DO K=1 TO 3
                Spend_Output.K = SPEND.K || ' ' ||,
                TradeInfo.USER.K || ' ' ||,
                SUBSTR(TradeDay,1,6) || ' ' ||,
                RIGHT(TRUNC(Winterest.K,2),7,0)
                SPEND.K = 0
                Winterest.K = 0
            END
            "EXECIO * DISKW OUTADATA (STEM Spend_Output."
            Spend_Output.K = ""
        END
        /**** Specifies the date ****/
        DAY = DAY + 1
        IF DAY < 10 THEN
            DO
                DAY = 0||DAY
            END
        IF DAY == 31 THEN
            DO
                MONTH = MONTH+1
                MONTH = 0||MONTH
                DAY = 01
            END
        END
    END

```

```

END
END
/** Put each output data in an output array */
TradeInfo_Output.0=TradeInfo.0
DO I=1 TO TradeInfo_Output.0-1
  TradeInfo_Output.I=TradeInfo.I.TradeID      || ' ' ||,
                    TradeInfo.I.TradeUser    || ' ' ||,
  SUBSTR(RIGHT(TRUNC(TradeInfo.I.TradeAmount,2),7,0),1,4) ||,
  SUBSTR(RIGHT(TRUNC(TradeInfo.I.TradeAmount,2),7,0),6,2) || ' ' ||,
                    TradeInfo.I.TradeTime    || ' ' ||,
                    TradeInfo.I.TradeType    || ' ' ||,
                    TradeInfo.I.MaxLimit
END
/** Output data to the file */
"EXECIO * DISKW OUTADATA (STEM TradeInfo_Output."
SAY "OK"
EXIT

```

设计思路：该程序一次性生成了三个用户 USER1、USER2、USER3 三个用户的三个月的交易数据。数据的格式严格遵守在需求分析中所设计的数据格式（见表 3-1）。数据的写入用到了 REXX FOR Z/OS 中的 EXECIO 命令。

## 后台业务处理编码

```

IDENTIFICATION DIVISION.
PROGRAM-ID. CCS
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INTAB ASSIGN TO INTABLE.
    SELECT OUT1 ASSIGN TO OUTFILE1.
    SELECT BILL1 ASSIGN TO OUTBILL1.
    SELECT DIVIS1 ASSIGN TO SPLIT1.
    SELECT OUT2 ASSIGN TO OUTFILE2.
    SELECT BILL2 ASSIGN TO OUTBILL2.
    SELECT DIVIS2 ASSIGN TO SPLIT2.
    SELECT OUT3 ASSIGN TO OUTFILE3.
    SELECT BILL3 ASSIGN TO OUTBILL3.
DATA DIVISION.
FILE SECTION.

```



FD INTAB LABEL RECORD IS STANDARD.  
01 INTAB-REC PIC X(80).  
FD DIVIS1 RECORD CONTAINS 80 RECORDING MODE IS F.  
01 SP1.  
    02 C1 PIC X(40).  
    02 D1 PIC X(40).  
FD BILL1 RECORD CONTAINS 80 RECORDING MODE IS F.  
01 BILL-R1.  
    02 BL1 PIC X(4).  
    02 MONCON-N1 PIC X(12).  
    02 MONCON-R1 PIC 9(4)V9(4).  
    02 FILLER PIC X(3).  
    02 MONARR-N1 PIC X(8).  
    02 MONARR-R1 PIC 9(4)V9(4).  
    02 FILLER PIC X(3).  
FD OUT1 RECORD CONTAINS 80 RECORDING MODE IS F.  
01 RECORD1.  
    02 FILL1 PIC X(9).  
    02 LIUSHUI-R1 PIC X(14).  
    02 FILLER PIC X(3).  
    02 AMOUNT-R1 PIC 9(4)V9(4).  
    02 FILLER PIC X(3).  
    02 TI-R1 PIC 9(8).  
    02 FILLER PIC X(3).  
    02 TY-R1 PIC X(10).  
    02 FILLER PIC X(3).  
FD DIVIS2 RECORD CONTAINS 80 RECORDING MODE IS F.  
01 SP2.  
    02 C2 PIC X(40).  
    02 D2 PIC X(40).  
FD BILL2 RECORD CONTAINS 80 RECORDING MODE IS F.  
01 BILL-R2.  
    02 BL2 PIC X(4).  
    02 MONCON-N2 PIC X(12).  
    02 MONCON-R2 PIC 9(4)V9(4).  
    02 FILLER PIC X(3).  
    02 MONARR-N2 PIC X(8).  
    02 MONARR-R2 PIC 9(4)V9(4).  
    02 FILLER PIC X(3).  
    02 INTEREST-N2 PIC X(9).  
    02 INTEREST-R2 PIC 9(4)V9(4).  
    02 FILLER PIC X(3).  
FD OUT2 RECORD CONTAINS 80 RECORDING MODE IS F.

01 RECORD2.

02 FILL2 PIC X(9).

02 LIUSHUI-R2 PIC X(14).

02 FILLER PIC X(3).

02 AMOUNT-R2 PIC 9(4)V9(4).

02 FILLER PIC X(3).

02 TI-R2 PIC 9(8).

02 FILLER PIC X(3).

02 TY-R2 PIC X(10).

02 FILLER PIC X(3).

FD DIVIS3 RECORD CONTAINS 80 RECORDING MODE IS F.

01 SP3.

02 C3 PIC X(40).

02 D3 PIC X(40).

FD BILL3 RECORD CONTAINS 80 RECORDING MODE IS F.

01 BILL-R3.

02 BL3 PIC X(4).

02 MONCON-N3 PIC X(12).

02 MONCON-R3 PIC 9(4)V9(4).

02 FILLER PIC X(3).

02 MONARR-N3 PIC X(8).

02 MONARR-R3 PIC 9(4)V9(4).

02 FILLER PIC X(3).

02 INTEREST-N3 PIC X(9).

02 INTEREST-R3 PIC 9(4)V9(4).

02 FILLER PIC X(3).

FD OUT3 RECORD CONTAINS 80 RECORDING MODE IS F.

01 RECORD3.

02 FILL3 PIC X(9).

02 LIUSHUI-R3 PIC X(14).

02 FILLER PIC X(3).

02 AMOUNT-R3 PIC 9(4)V9(4).

02 FILLER PIC X(3).

02 TI-R3 PIC 9(8).

02 FILLER PIC X(3).

02 TY-R3 PIC X(10).

02 FILLER PIC X(3).

WORKING-STORAGE SECTION.

01 U PIC 9.

01 TIM PIC 9(8).

01 MONTH PIC 9(2) VALUE 01.

01 DA-Y PIC 9(2).

01 CUSTTAB.

02 CUSTOMER OCCURS 3 TIMES INDEXED BY IX.

10 MONCON PIC 9(4)V9(4) VALUE 0.  
 10 LASTCON PIC 9(4)V9(4) VALUE 0.  
 10 MONARR PIC 9(4)V9(4) VALUE 0.  
 10 LASTARR PIC 9(4)V9(4) VALUE 0.  
 10 INTERESTQ PIC 9(4)V9(4) VALUE 0.  
 10 INTERESTX PIC 9(4)V9(4) VALUE 0.  
 10 LASTINTEX PIC 9(4)V9(4) VALUE 0.  
 10 INTEREST PIC 9(4)V9(4) VALUE 0.  
 10 REPAY PIC 9(4)V9(4) VALUE 0.

01 WORK-REC.

02 WORK-TABLE OCCURS 270 TIMES  
 INDEXED BY TX.

04 LIUSHUI PIC X(14).  
 04 FILLER PIC X(3).  
 04 USR PIC X(5).  
 04 FILLER PIC X(3).  
 04 AMOUNT PIC 9(4)V99.  
 04 FILLER PIC X(3).  
 04 TI PIC 9(8).  
 04 FILLER PIC X(3).  
 04 TY PIC X.  
 04 FILLER PIC X(3).  
 04 MAX PIC 9(4).  
 04 FILLER PIC X(3).

PROCEDURE DIVISION.

S. OPEN INPUT INTAB.

PERFORM VARYING TX FROM 1 BY 1 UNTIL TX > 270  
 READ INTAB  
 MOVE INTAB-REC TO WORK-TABLE (TX)  
 END-PERFORM  
 CLOSE INTAB.  
 SET TX TO 1  
 PERFORM VARYING TX FROM 1 BY 1 UNTIL TX > 270  
 PERFORM W1  
 MOVE TI(TX) TO TIM  
 MOVE TIM(7:2) TO DA-Y  
 IF MONTH = TIM(5:2)  
 IF USR(TX) = 'XR001' AND TY(TX) NOT EQUAL 2  
 COMPUTE MONCON(1) = MONCON(1) + AMOUNT(TX)  
 IF TY(TX) = 0  
 COMPUTE INTERESTX(1) = INTERESTX(1)

```

-          + AMOUNT(TX)*(0.0005)
-          *(40 - (DA-Y) + 1)
END-IF
IF TY(TX) = 1
    COMPUTE INTERESTQ(1) = INTERESTQ(1)
-          + AMOUNT(TX)*(0.0005)
-          *(30 - (DA-Y) + 1)
END-IF
IF TY(TX) = 2
    COMPUTE REPAY(1) = REPAY(1) + AMOUNT(TX)
END-IF
END-IF
IF USR(TX) = 'XR002' AND TY(TX) NOT EQUAL 2
    COMPUTE MONCON(2) = MONCON(2) + AMOUNT(TX)
    IF TY(TX) = 0
        COMPUTE INTERESTX(2) = INTERESTX(2)
-          + AMOUNT(TX)*(0.0005)
-          *(40 - (DA-Y) + 1)
    END-IF
    IF TY(TX) = 1
        COMPUTE INTERESTQ(2) = INTERESTQ(2)
-          + AMOUNT(TX)*(0.0005)
-          *(30 - (DA-Y) + 1)
    END-IF
    IF TY(TX) = 2
        COMPUTE REPAY(2) = REPAY(2) + AMOUNT(TX)
    END-IF
END-IF
IF USR(TX) = 'XR003' AND TY(TX) NOT EQUAL 2
    COMPUTE MONCON(3) = MONCON(3) + AMOUNT(TX)
    IF TY(TX) = 0
        COMPUTE INTERESTX(3) = INTERESTX(3)
-          + AMOUNT(TX)*(0.0005)
-          *(40 - (DA-Y) + 1)
    END-IF
    IF TY(TX) = 1
        COMPUTE INTERESTQ(3) = INTERESTQ(3)
-          + AMOUNT(TX)*(0.0005)
-          *(30 - (DA-Y) + 1)
    END-IF
    IF TY(TX) = 2

```

```

        COMPUTE REPAY(3) = REPAY(3) + AMOUNT(TX)
    END-IF
END-IF
IF TIM(7:2) = 30 AND USR(TX) = 'XR003'
    SET IX TO 1
    MOVE 0 TO U
    PERFORM VARYING IX FROM 1 BY 1 UNTIL IX > 3
        DISPLAY LASTARR(IX) 'AAA'
        IF REPAY(IX) >= LASTARR(IX)
            DISPLAY REPAY(IX)
            COMPUTE MONARR(IX) = MONCON(IX)
            + INTERESTQ(IX)
            MOVE INTERESTQ(IX) TO INTEREST(IX)
        ELSE
            COMPUTE MONARR(IX) =
                MONCON(IX)
                + LASTARR(IX)
                - REPAY(IX)
                + INTERESTQ(IX)
                + (LASTARR(IX)
                - REPAY(IX))
                *(0.0005)
                *(20)
                + (LASTARR(IX)
                - LASTCON(IX))
                *(0.0005)
                *(10)
                + LASTINTEX(IX)
            COMPUTE INTEREST(IX) =
                MONARR(IX) - MONCON(IX)
                - LASTARR(IX) + REPAY(IX)
        END-IF
        COMPUTE U = U + 1
        PERFORM W12
    END-PERFORM
END-IF
ELSE
    MOVE TIM(5:2) TO MONTH
    SET IX TO 1
    PERFORM VARYING IX FROM 1 BY 1 UNTIL IX > 3
        MOVE MONCON(IX) TO LASTCON(IX)
        MOVE MONARR(IX) TO LASTARR(IX)

```

```

        MOVE INTERESTX(IX) TO LASTINTEX(IX)
        MOVE 0 TO INTERESTX(IX)
        MOVE 0 TO MONCON(IX)
        MOVE 0 TO MONARR(IX)
        MOVE 0 TO REPAY(IX)
        MOVE 0 TO INTERESTQ(IX)
        MOVE 0 TO INTEREST(IX)
END-PERFORM
IF USR(TX) = 'XR001' AND TY(TX) NOT EQUAL 2
    COMPUTE MONCON(1) = MONCON(1) + AMOUNT(TX)
    IF TY(TX) = 0
        COMPUTE INTERESTX(1) = INTERESTX(1)
        -      + AMOUNT(TX)*(0.0005)
        -      *(40 - (DA-Y) + 1)
    END-IF
    IF TY(TX) = 1
        COMPUTE INTERESTQ(1) = INTERESTQ(1)
        -      + AMOUNT(TX)*(0.0005)
        -      *(30 - (DA-Y) + 1)
    END-IF
    IF TY(TX) = 2
        COMPUTE REPAY(1) = REPAY(1) + AMOUNT(TX)
    END-IF
END-IF
IF USR(TX) = 'XR002' AND TY(TX) NOT EQUAL 2
    COMPUTE MONCON(2) = MONCON(2) + AMOUNT(TX)
    IF TY(TX) = 0
        COMPUTE INTERESTX(2) = INTERESTX(2)
        -      + AMOUNT(TX)*(0.0005)
        -      *(40 - (DA-Y) + 1)
    END-IF
    IF TY(TX) = 1
        COMPUTE INTERESTQ(2) = INTERESTQ(2)
        -      + AMOUNT(TX)*(0.0005)
        -      *(30 - (DA-Y) + 1)
    END-IF
    IF TY(TX) = 2
        COMPUTE REPAY(2) = REPAY(2) + AMOUNT(TX)
    END-IF
END-IF
IF USR(TX) = 'XR003' AND TY(TX) NOT EQUAL 2
    COMPUTE MONCON(3) = MONCON(3) + AMOUNT(TX)
    IF TY(TX) = 0

```

```
        COMPUTE INTERESTX(3) = INTERESTX(3)
-          + AMOUNT(TX)*(0.0005)
-          *(40 - (DA-Y) + 1)
        END-IF
        IF TY(TX) = 1
            COMPUTE INTERESTQ(3) = INTERESTQ(3)
-          + AMOUNT(TX)*(0.0005)
-          *(30 - (DA-Y) + 1)
        END-IF
        IF TY(TX) = 2
            COMPUTE REPAY(3) = REPAY(3) + AMOUNT(TX)
        END-IF
    END-IF
END-PERFORM
STOP RUN.
```

W1.

```
    IF USR(TX) = 'XR001'
        OPEN EXTEND OUT1
        MOVE SPACE TO LIUSHUI-R1
        MOVE SPACE TO FILL1
        MOVE ZERO TO AMOUNT-R1
        MOVE ZERO TO TI-R1
        MOVE SPACE TO TY-R1
        MOVE LIUSHUI(TX) TO LIUSHUI-R1
        MOVE AMOUNT(TX) TO AMOUNT-R1
        MOVE TI(TX) TO TI-R1
        IF TY(TX) = 0
            MOVE 'CONSUM' TO TY-R1
        END-IF
        IF TY(TX) = 1
            MOVE 'GETCASH' TO TY-R1
        END-IF
        IF TY(TX) = 2
            MOVE 'REPAY' TO TY-R1
        END-IF
        WRITE RECORD1
        CLOSE OUT1
    END-IF
    IF USR(TX) = 'XR002'
        OPEN EXTEND OUT2
        MOVE SPACE TO LIUSHUI-R2
        MOVE SPACE TO FILL2
        MOVE ZERO TO AMOUNT-R2
```

```
MOVE ZERO TO TI-R2
MOVE SPACE TO TY-R2
MOVE LIUSHUI(TX) TO LIUSHUI-R2
MOVE AMOUNT(TX) TO AMOUNT-R2
MOVE TI(TX) TO TI-R2
IF TY(TX) = 0
    MOVE 'CONSUM' TO TY-R2
END-IF
IF TY(TX) = 1
    MOVE 'GETCASH' TO TY-R2
END-IF
IF TY(TX) = 2
    MOVE 'REPAY' TO TY-R2
END-IF
WRITE RECORD2
CLOSE OUT2
END-IF
IF USR(TX) = 'XR003'
    OPEN EXTEND OUT3
    MOVE SPACE TO LIUSHUI-R3
    MOVE SPACE TO FILL3
    MOVE ZERO TO AMOUNT-R3
    MOVE ZERO TO TI-R3
    MOVE SPACE TO TY-R3
    MOVE LIUSHUI(TX) TO LIUSHUI-R3
    MOVE AMOUNT(TX) TO AMOUNT-R3
    MOVE TI(TX) TO TI-R3
    IF TY(TX) = 0
        MOVE 'CONSUM' TO TY-R3
    END-IF
    IF TY(TX) = 1
        MOVE 'GETCASH' TO TY-R3
    END-IF
    IF TY(TX) = 2
        MOVE 'REPAY' TO TY-R3
    END-IF
    WRITE RECORD3
    CLOSE OUT3
END-IF
EXIT.
W12.
IF U = 1
    DISPLAY U
```



```
OPEN EXTEND BILL1
MOVE SPACE TO BL1
MOVE 'CONSUMPTION:' TO MONCON-N1
MOVE MONCON(IX) TO MONCON-R1
MOVE 'ARREARS:' TO MONARR-N1
MOVE MONARR(IX) TO MONARR-R1
MOVE 'INTEREST:' TO INTEREST-N1
MOVE INTEREST(IX) TO INTEREST-R1
WRITE BILL-R1
CLOSE BILL1
OPEN EXTEND DIVIS1
MOVE 'LINE*****' TO D1
MOVE '*****SPLIT' TO C1
WRITE SP1
CLOSE DIVIS1
END-IF
IF U = 2
    DISPLAY U
    OPEN EXTEND BILL2
    MOVE SPACE TO BL2
    MOVE 'CONSUMPTION:' TO MONCON-N2
    MOVE MONCON(IX) TO MONCON-R2
    MOVE 'ARREARS:' TO MONARR-N2
    MOVE MONARR(IX) TO MONARR-R2
    MOVE 'INTEREST:' TO INTEREST-N2
    MOVE INTEREST(IX) TO INTEREST-R2
    WRITE BILL-R2
    CLOSE BILL2
    OPEN EXTEND DIVIS2
    MOVE 'LINE*****' TO D2
    MOVE '*****SPLIT' TO C2
    WRITE SP2
    CLOSE DIVIS2
END-IF
IF U = 3
    DISPLAY U
    OPEN EXTEND BILL3
    MOVE SPACE TO BL3
    MOVE 'CONSUMPTION:' TO MONCON-N3
    MOVE MONCON(IX) TO MONCON-R3
    MOVE 'ARREARS:' TO MONARR-N3
    MOVE MONARR(IX) TO MONARR-R3
    MOVE 'INTEREST:' TO INTEREST-N3
    MOVE INTEREST(IX) TO INTEREST-R3
```

---

```

WRITE BILL-R3
CLOSE BILL3
OPEN EXTEND DIVIS3
MOVE 'LINE*****' TO D3
MOVE '*****SPLIT' TO C3
WRITE SP3
CLOSE DIVIS3
END-IF
EXIT.

```

编译使用的 JCL

```

//CCS JOB ACCT#,ADCDMST,CLASS=A,MSGCLASS=H,
// MSGLEVEL=(1,1),NOTIFY=&SYSUID
//MYLIB JCLLIB ORDER=IGY340.SIGYPROC
//COBCL EXEC IGYWCL
//COBOL.SYSIN DD DISP=SHR,DSN=ADCDMST.COBOLO.SOURCE(CCS)
//LKED.SYSLMOD DD DISP=SHR,DSN=ADCDMST.COBOLO.LOAD(CCS)
//SYSPRINT DD SYSOUT=*

```

```

//RCCS JOB ACCT#,ADCDMST,CLASS=A,MSGCLASS=H,
// MSGLEVEL=(1,1),NOTIFY=&SYSUID
//RUNCBL EXEC PGM=CCS
//STEPLIB DD DSN=ADCDMST.COBOLO.LOAD,DISP=SHR
//INTABLE DD DSN=ADCDMST.REXX.OUTPUT(DATAS1),DISP=SHR
//OUTFILE1 DD DSN=ADCDMST.RESULT.RECORD1,DISP=SHR
//OUTBILL1 DD DSN=ADCDMST.RESULT.RECORD1,DISP=SHR
//SPLIT1 DD DSN=ADCDMST.RESULT.RECORD1,DISP=SHR
//OUTFILE2 DD DSN=ADCDMST.RESULT.RECORD2,DISP=SHR
//OUTBILL2 DD DSN=ADCDMST.RESULT.RECORD2,DISP=SHR
//SPLIT2 DD DSN=ADCDMST.RESULT.RECORD2,DISP=SHR
//OUTFILE3 DD DSN=ADCDMST.RESULT.RECORD3,DISP=SHR
//OUTBILL3 DD DSN=ADCDMST.RESULT.RECORD3,DISP=SHR
//SPLIT3 DD DSN=ADCDMST.RESULT.RECORD3,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*

```