IBM iSeries 金融系统开发最佳实践指南

全面覆盖银行核心、卡支付、证券、保险、风控与合规

版本: 1.0
发布日期: 2026年2月
适用平台: IBM Power Systems (AS/400, iSeries, IBM i)
编程语言: RPG IV, CL, SQL, Java, Node.js
目标读者: 金融系统架构师、RPG开发工程师、项目经理、技术总监

目录

第一章 引言与概述

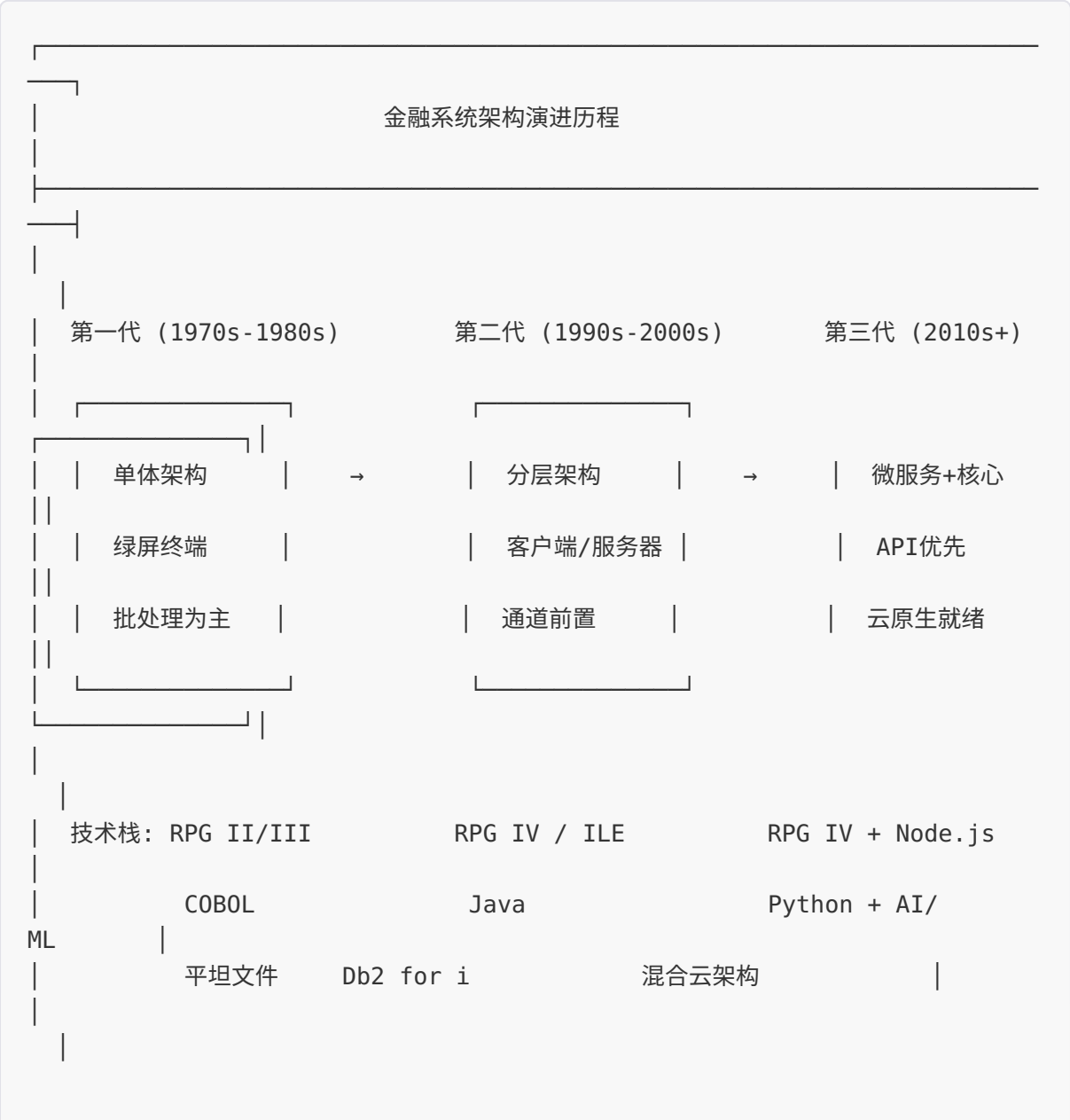
1.1 IBM iSeries在金融行业的地位

IBM iSeries（前身为AS/400）自1988年发布以来，一直是全球金融机构核心系统的首选平台。据统计，全球前100大银行中有超过85家使用IBM Power Systems运行其核心银行业务。IBM iSeries以其卓越的可靠性、安全性、可扩展性和集成能力，成为金融核心系统不可替代的基础设施。

IBM iSeries金融应用核心优势:

1. **极致可靠性**: 达到99.999%以上的可用性，支持热插拔、在线维护
2. **内置安全性**: 从芯片级到应用级的多层安全防护
3. **事务完整性**: 单级存储架构确保数据一致性和完整性
4. **纵向扩展能力**: 支持从小型社区银行到大型跨国银行的无缝扩展
5. **多语言支持**: RPG、COBOL、C、C++、Java、Node.js、Python等
6. **数据库集成**: Db2 for i是内置的关系型数据库，无需额外许可

1.2 金融系统架构演进



```
└──────────────────────────────────────────────────────────────────────────────┘
```

1.3 当前挑战与机遇

主要挑战:

1. **人才短缺**: RPG开发工程师老龄化，新生代开发者更倾向于现代语言
2. **系统耦合**: 数十年积累的业务逻辑高度耦合，难以快速响应市场变化
3. **监管压力**: 巴塞尔协议III/IV、GDPR、数据本地化等合规要求日益严格
4. **技术债务**: 遗留代码维护成本高，文档缺失，知识传承困难
5. **数字化转型**: 互联网银行、移动支付的冲击，要求7×24小时不间断服务

应对策略:

1. **现代化改造**: 采用ILE RPG、SQL RPG、自由格式RPG提升开发效率
2. **API化**: 通过REST/SOAP API将核心业务能力与前端解耦
3. **混合云**: 保留核心在iSeries，将渠道层、分析层部署在云端
4. **DevOps**: 引入自动化测试、持续集成/持续部署(CI/CD)
5. **知识管理**: 建立完善的技术文档和培训体系

1.4 文档使用指南

本文档面向不同角色的技术人员提供针对性指导:

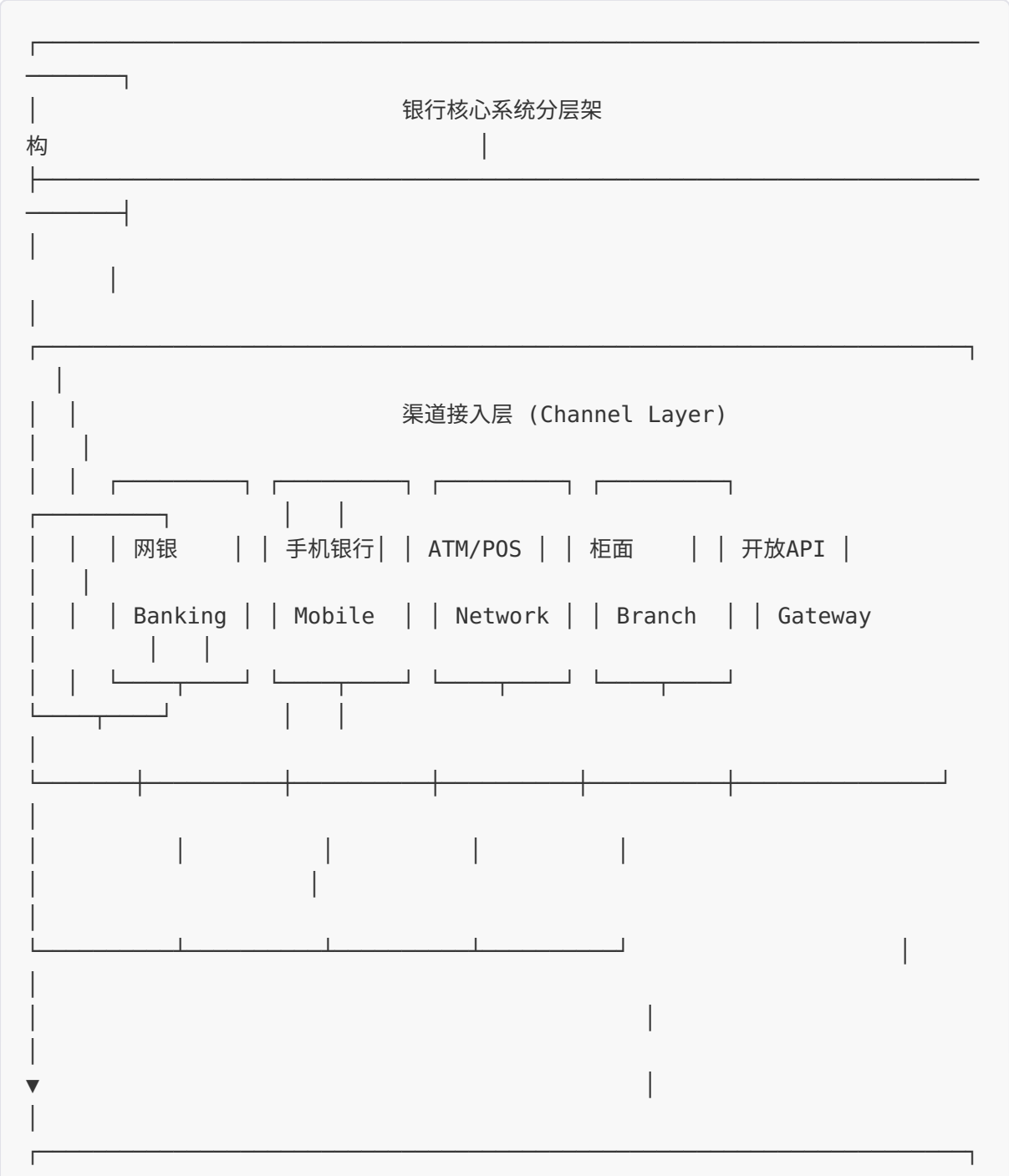
角色	重点章节	建议阅读顺序
RPG开发工程师	第2-7章代码示例	1→2→3→7→8
系统架构师	架构设计章节	1→2→3→4→5→8→9
项目经理	最佳实践和案例研究	1→各章节总结
安全合规专员	第6-7章	1→6→7→8
运维工程师	第8章性能与高可用	1→8→各章运维要点

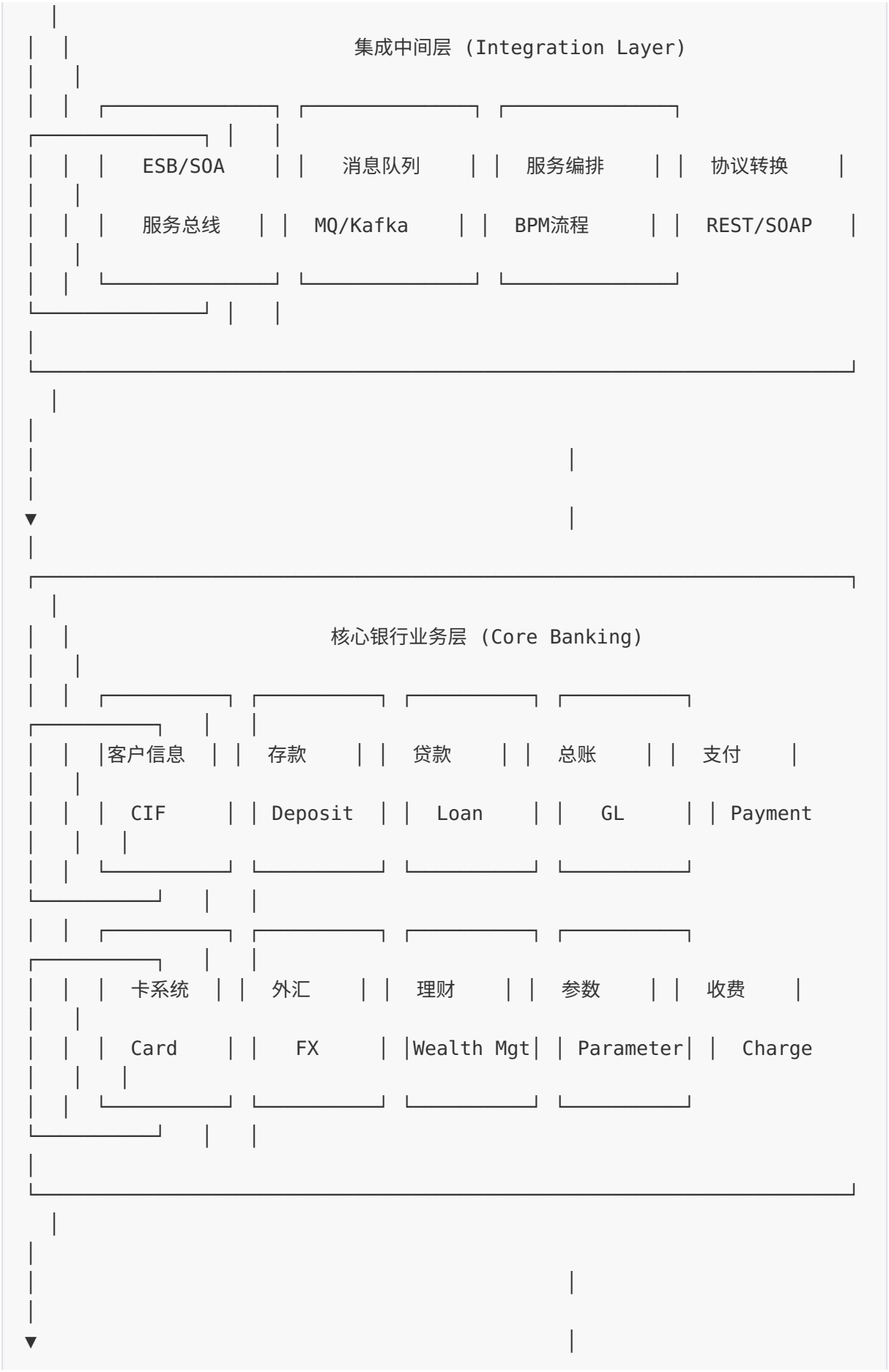
第二章 银行核心系统

2.1 核心银行架构

2.1.1 总体架构设计

银行核心系统是金融机构的中枢神经系统，负责处理所有客户账户、交易、产品定义和业务规则。一个现代化的核心银行系统通常采用分层架构:




集成中间层 (Integration Layer)
ESB/SOA
服务总线
消息队列
MQ/Kafka
服务编排
BPM流程
协议转换
REST/SOAP
核心银行业务层 (Core Banking)
客户信息
CIF
存款
Deposit
贷款
Loan
总账
GL
支付
Payment
卡系统
Card
外汇
FX
理财
Wealth Mgt
参数
Parameter
收费
Charge

2.1.2 模块划分原则

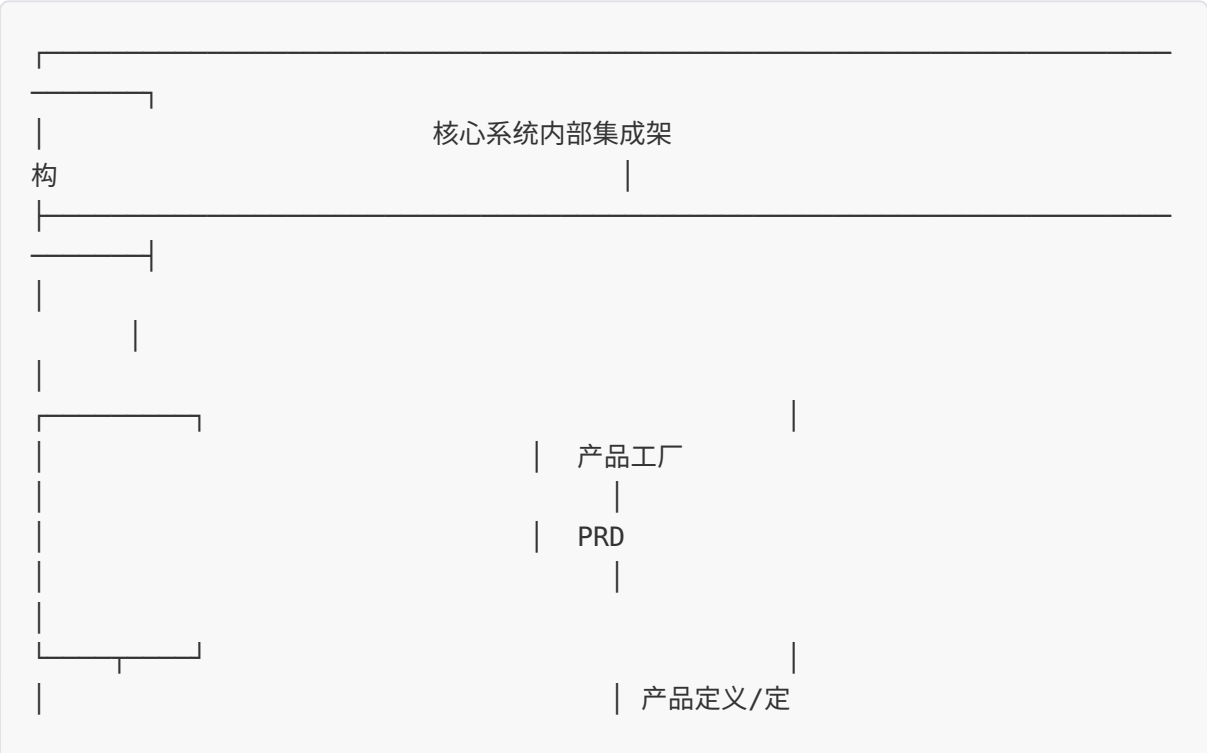
核心银行系统的模块划分应遵循高内聚、低耦合的原则:

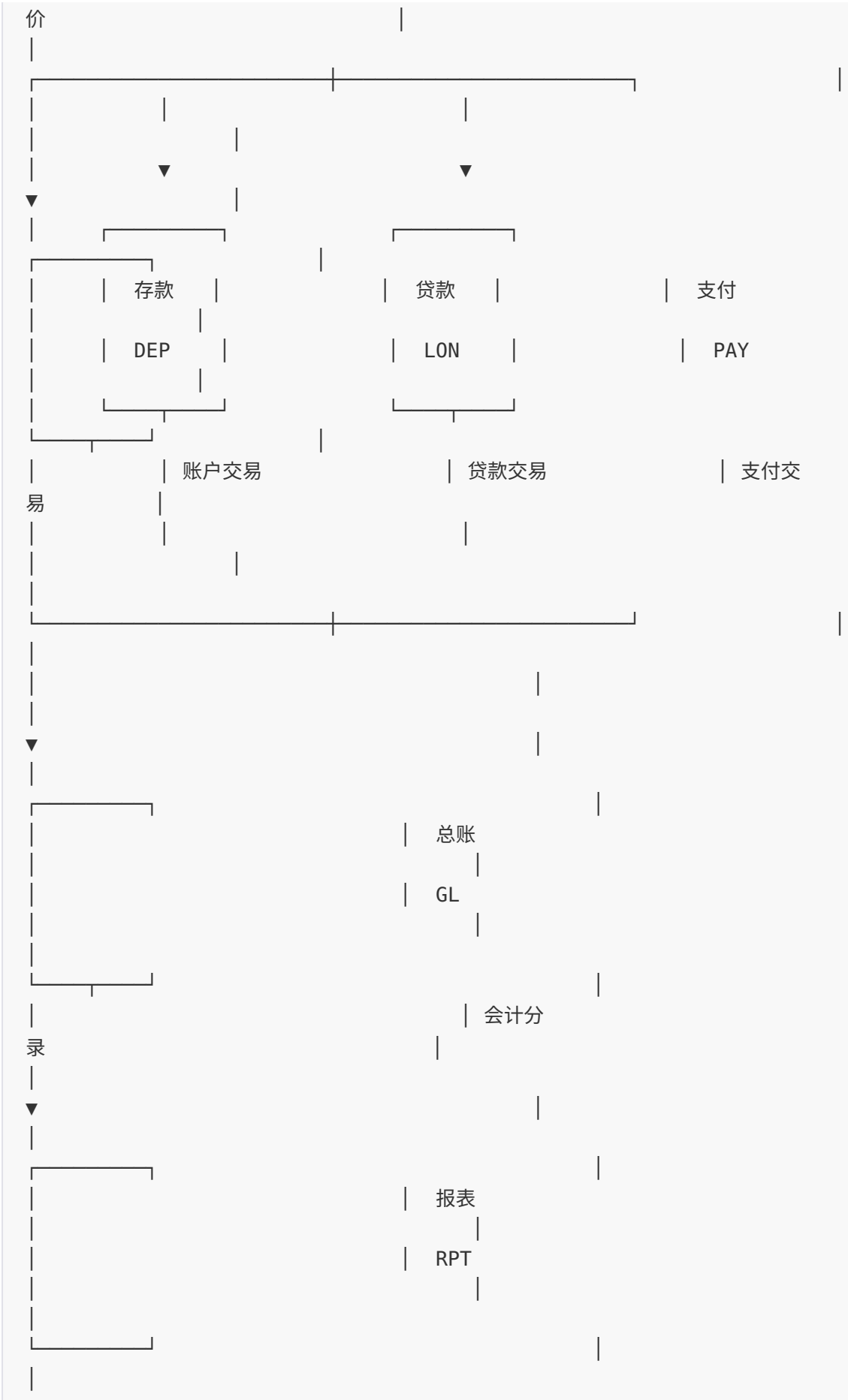
核心模块清单:

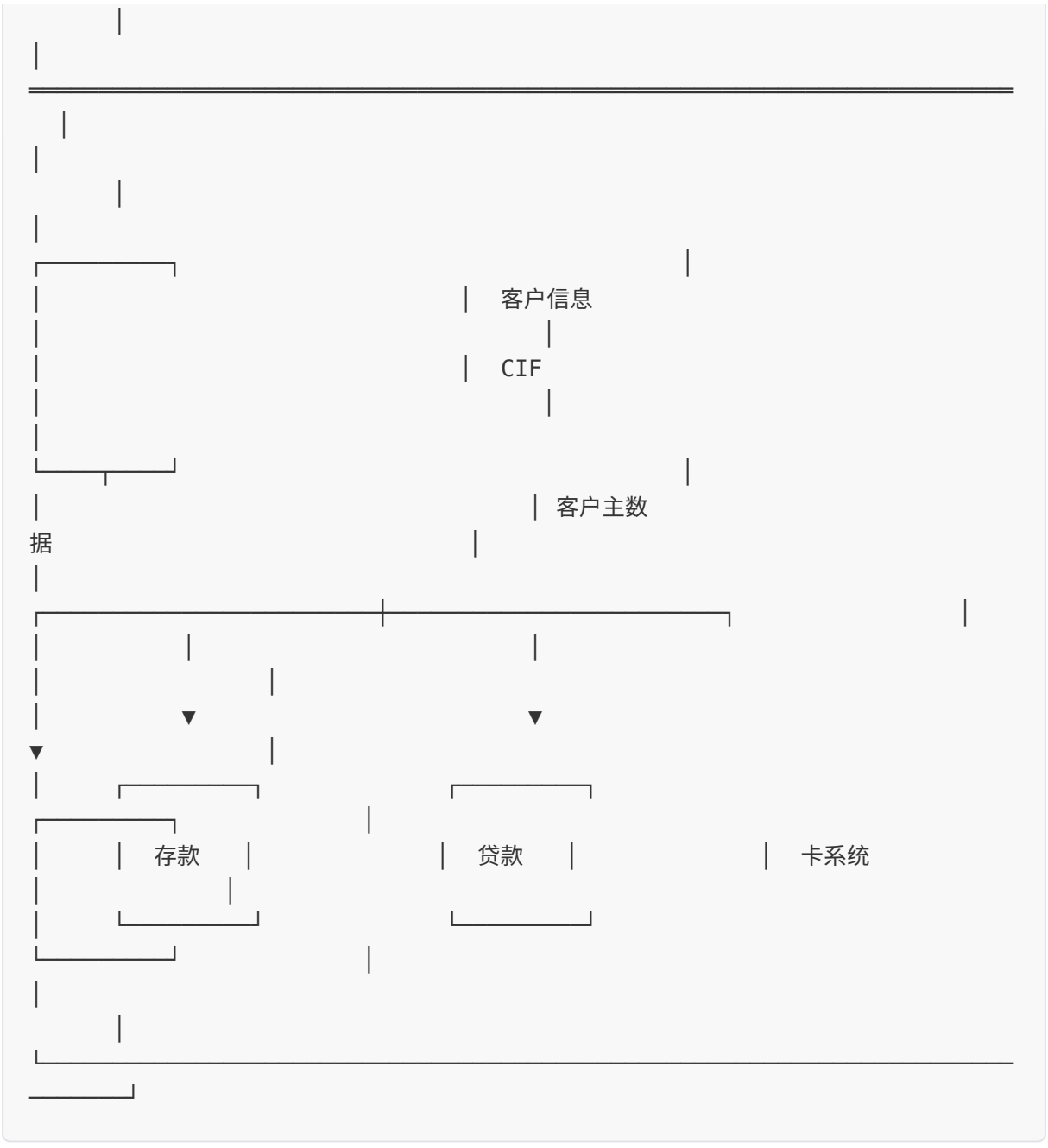
模块代码	模块名称	功能描述	关键数据表
CIF	客户信息系统	管理客户主数据、关系、评级	CUSTMAST, CUSTADDR, CUSTREL

模块代码	模块名称	功能描述	关键数据表
DEP	存款系统	活期、定期、通知存款管理	ACCTMAST, ACCTDTL, TERMINFO
LON	贷款系统	各类贷款产品生命周期管理	LOANMAST, LOANSCHD, COLLATERAL
GL	总账系统	会计科目、分录、报表	GLMAST, GLTRANS, GLBATCH
PAY	支付系统	跨行转账、汇款、票据	PAYMST, PAYTRANS, CHECKINFO
CAR	卡系统	借记卡、信用卡管理	CARDMAST, CARDTRANS, CARDLIMIT
FX	外汇系统	结售汇、外汇买卖	FXRATE, FXTRANS, FXPOSITION
PRD	产品工厂	产品参数、定价、组合	PRODUCT, PRDRULE, PRICING
PAR	参数中心	机构、柜员、日期、利率	BRANCH, TELLER, DAILYPRM
CHG	收费系统	费率计算、优惠、减免	CHGRATE, CHGTRANS, CHGWAIVER

2.1.3 系统间集成架构




存款
DEP
贷款
LON
支付
PAY
账户交易
贷款交易
支付交易
总账
GL
会计分录
报表
RPT

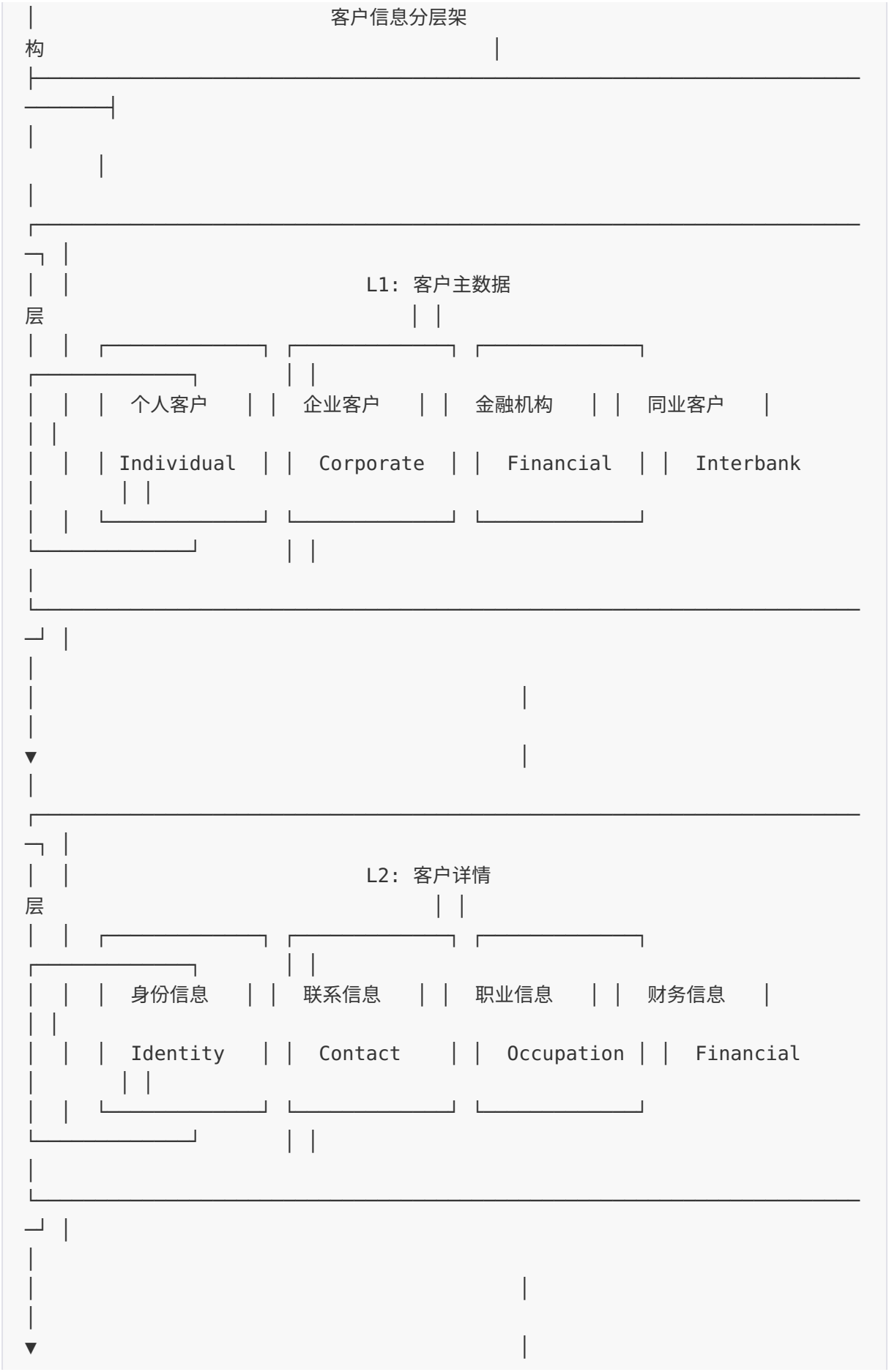


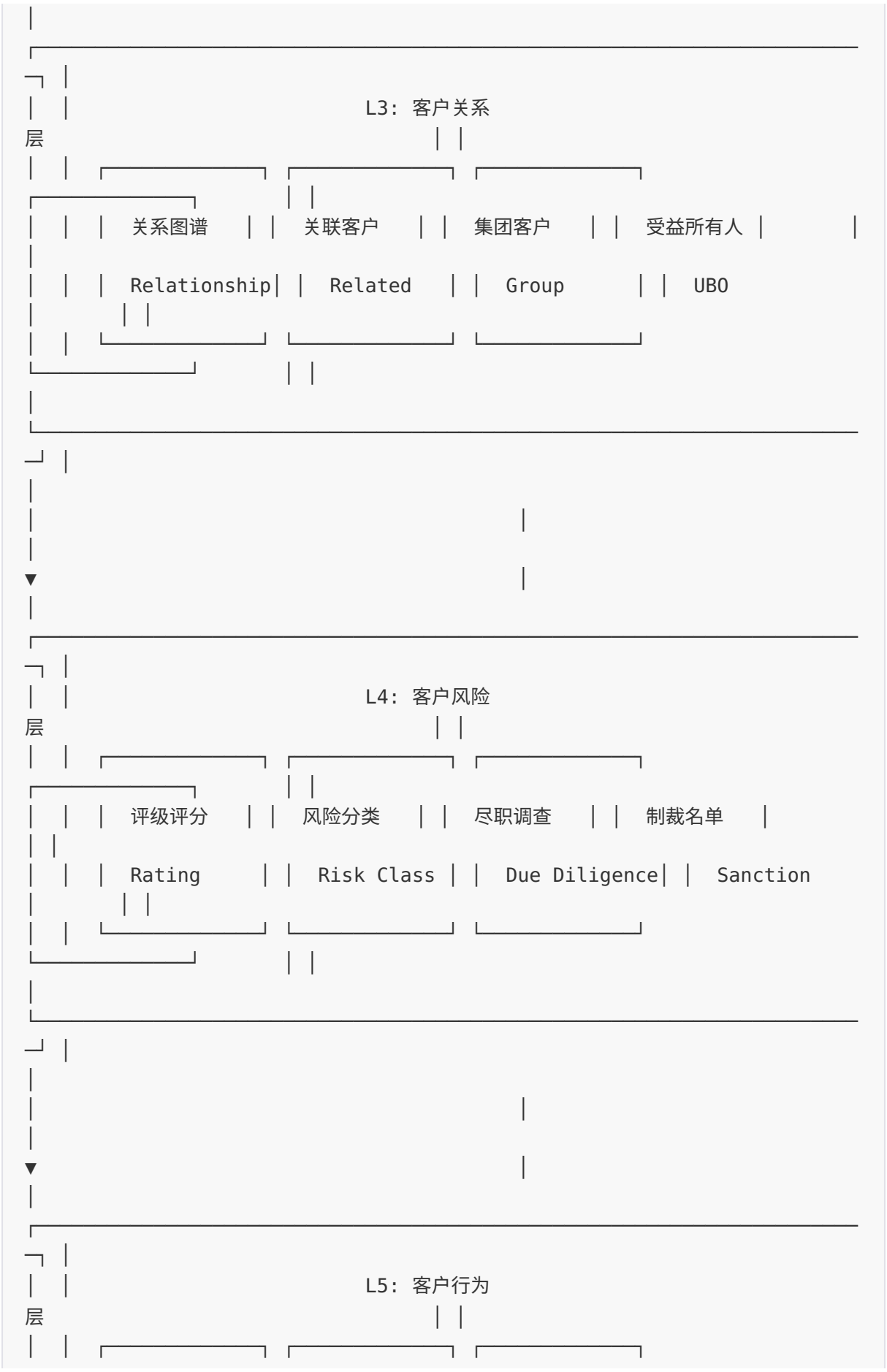
2.2 客户信息系统(CIF)

2.2.1 CIF架构设计

客户信息系统是银行最基础、最核心的数据资产，是所有业务开展的起点。CIF的设计必须考虑客户360度视图、关系图谱、风险评级等维度。

客户信息分层模型:


客户信息分层架构
L1: 客户主数据层
个人客户
Individual
企业客户
Corporate
金融机构
Financial
同业客户
Interbank
L2: 客户详情层
身份信息
Identity
联系信息
Contact
职业信息
Occupation
财务信息
Financial


L3: 客户关系层
关系图谱
Relationship
关联客户
Related
集团客户
Group
受益所有人
UBO
L4: 客户风险层
评级评分
Rating
风险分类
Risk Class
尽职调查
Due Diligence
制裁名单
Sanction
L5: 客户行为层

2.2.2 客户主表设计

物理文件(PF)定义 - 客户主表:

```
**free
// ================================================================
// 客户主表 - CUSTMAST
// 描述: 存储所有客户的基本信息
// 创建日期: 2026-01-15
// 作者: 核心系统架构组
// ================================================================

dcl-ds CUSTMAST qualified template;
  // 主键字段
  CUST_ID       char(20)  ;  // 客户号 - 主键,唯一标识
  CUST_TYPE     char(1)   ;  // 客户类型: 1-个人 2-企业 3-金融机构

  // 客户名称信息
  CUST_NAME     char(100) ;  // 客户全称
  CUST_SHORT    char(50)  ;  // 客户简称
  CUST_PY       char(100) ;  // 名称拼音/英文

  // 证件信息
  ID_TYPE       char(2)   ;  // 证件类型: 01-身份证 02-护照 03-营业执照...
  ID_NO         char(50)  ;  // 证件号码
  ID_EXP_DT     packed(8) ;  // 证件到期日YYYYMMDD
  ID_ISSUE_PLC  char(50)  ;  // 证件签发地点
```

```
// 国籍/地区信息
NATION_CD     char(3)   ;  // 国籍代码(ISO 3166-1)
REGION_CD     char(6)   ;  // 地区代码
RESIDENT_STAT char(1)   ;  // 居民标志: 0-非居民 1-居民

// 联系信息
MOBILE_NO     char(20)  ;  // 手机号
PHONE_NO      char(20)  ;  // 固定电话
EMAIL         char(100) ;  // 电子邮箱
POSTAL_CD     char(10)  ;  // 邮政编码
ADDRESS       varchar(500); // 详细地址

// 风险评级
RISK_LEVEL    char(2)   ;  // 风险等级: L-低 M-中 H-高 P-禁止
RISK_RATE_DT  packed(8) ;  // 评级日期
RISK_RATE_RSN char(200) ;  // 评级原因

// KYC信息
KYC_STAT      char(1)   ;  // KYC状态: 0-未开始 1-进行中 2-完成 3-过期
KYC_COMP_DT   packed(8) ;  // KYC完成日期
KYC_NEXT_DT   packed(8) ;  // 下次KYC日期

// 账户控制
ACCT_STAT     char(1)   ;  // 账户状态: 0-正常 1-预开户 2-休眠 9-销户
STAT_CHG_DT   packed(8) ;  // 状态变更日期
STAT_CHG_RSN  char(100) ;  // 状态变更原因

// 黑名单/制裁名单检查
SANCTION_CHK  char(1)   ;  // 制裁检查: 0-通过 1-嫌疑 2-命中
SANCTION_LIST char(200) ;  // 命中的制裁名单
SANCTION_DT   packed(8) ;  // 检查日期

// 营销信息
CUST_MGR_ID   char(10)  ;  // 客户经理编号
BRANCH_ID     char(10)  ;  // 归属机构
VIP_LEVEL     char(2)   ;  // VIP等级: 00-普通 01-金卡 02-白金 03-钻石

// 生命周期
OPEN_DT       packed(8) ;  // 开户日期
OPEN_CHNL     char(2)   ;  // 开户渠道: 01-柜面 02-网银 03-手机...
CLOSE_DT      packed(8) ;  // 销户日期

// 审计字段
CREATE_TS     timestamp ;  // 创建时间戳
```

```
  CREATE_USER   char(20)  ;  // 创建用户
  UPDATE_TS     timestamp ;  // 更新时间戳
  UPDATE_USER   char(20)  ;  // 更新用户
  UPDATE_PRG    char(20)  ;  // 更新程序
end-ds;
```

SQL DDL - 创建客户主表:

```
-- ================================================================
-- 客户主表创建脚本
-- ================================================================

CREATE TABLE CUSTMAST (
    -- 主键
    CUST_ID         CHAR(20)        NOT NULL,

    -- 客户类型
    CUST_TYPE       CHAR(1)         NOT NULL DEFAULT '1',

    -- 客户名称
    CUST_NAME       VARCHAR(100)    NOT NULL,
    CUST_SHORT      VARCHAR(50),
    CUST_PY         VARCHAR(100),

    -- 证件信息
    ID_TYPE         CHAR(2)         NOT NULL,
    ID_NO           VARCHAR(50)     NOT NULL,
    ID_EXP_DT       DECIMAL(8, 0),
    ID_ISSUE_PLC    VARCHAR(50),

    -- 国籍/地区
    NATION_CD       CHAR(3)         DEFAULT 'CHN',
    REGION_CD       CHAR(6),
    RESIDENT_STAT   CHAR(1)         DEFAULT '1',

    -- 联系信息
    MOBILE_NO       VARCHAR(20),
    PHONE_NO        VARCHAR(20),
    EMAIL           VARCHAR(100),
    POSTAL_CD       CHAR(10),
    ADDRESS         VARCHAR(500),

    -- 风险评级
    RISK_LEVEL      CHAR(2)         DEFAULT 'M1',
    RISK_RATE_DT    DECIMAL(8, 0),
```

```
    RISK_RATE_RSN   VARCHAR(200),

    -- KYC信息
    KYC_STAT        CHAR(1)         DEFAULT '0',
    KYC_COMP_DT     DECIMAL(8, 0),
    KYC_NEXT_DT     DECIMAL(8, 0),

    -- 账户控制
    ACCT_STAT       CHAR(1)         DEFAULT '1',
    STAT_CHG_DT     DECIMAL(8, 0),
    STAT_CHG_RSN    VARCHAR(100),

    -- 制裁检查
    SANCTION_CHK    CHAR(1)         DEFAULT '0',
    SANCTION_LIST   VARCHAR(200),
    SANCTION_DT     DECIMAL(8, 0),

    -- 营销信息
    CUST_MGR_ID     CHAR(10),
    BRANCH_ID       CHAR(10)        NOT NULL,
    VIP_LEVEL       CHAR(2)         DEFAULT '00',

    -- 生命周期
    OPEN_DT         DECIMAL(8, 0)   NOT NULL,
    OPEN_CHNL       CHAR(2)         DEFAULT '01',
    CLOSE_DT        DECIMAL(8, 0),

    -- 审计字段
    CREATE_TS       TIMESTAMP       DEFAULT CURRENT_TIMESTAMP,
    CREATE_USER     VARCHAR(20)     DEFAULT USER,
    UPDATE_TS       TIMESTAMP       DEFAULT CURRENT_TIMESTAMP,
    UPDATE_USER     VARCHAR(20)     DEFAULT USER,
    UPDATE_PRG      VARCHAR(20),

    -- 主键约束
    CONSTRAINT PK_CUSTMAST PRIMARY KEY (CUST_ID),

    -- 唯一约束 - 证件类型+证件号码组合唯一
    CONSTRAINT UK_CUSTMAST_ID UNIQUE (ID_TYPE, ID_NO),

    -- 检查约束
    CONSTRAINT CK_CUST_TYPE CHECK (CUST_TYPE IN ('1', '2', '3')),
    CONSTRAINT CK_RESIDENT CHECK (RESIDENT_STAT IN ('0', '1')),
    CONSTRAINT CK_ACCT_STAT CHECK (ACCT_STAT IN ('0', '1', '2', '9')),
    CONSTRAINT CK_KYC_STAT CHECK (KYC_STAT IN ('0', '1', '2', '3')),
    CONSTRAINT CK_SANCTION_CHK CHECK (SANCTION_CHK IN ('0', '1', '2'))
```

```
) RCDFMT CUSTMASTR;

-- 为表添加注释
LABEL ON TABLE CUSTMAST IS '客户主表';

-- 为列添加注释
LABEL ON COLUMN CUSTMAST (
    CUST_ID         IS '客户号',
    CUST_TYPE       IS '客户类型',
    CUST_NAME       IS '客户全称',
    CUST_SHORT      IS '客户简称',
    CUST_PY         IS '名称拼音',
    ID_TYPE         IS '证件类型',
    ID_NO           IS '证件号码',
    ID_EXP_DT       IS '证件到期日',
    ID_ISSUE_PLC    IS '证件签发地',
    NATION_CD       IS '国籍代码',
    REGION_CD       IS '地区代码',
    RESIDENT_STAT   IS '居民标志',
    MOBILE_NO       IS '手机号码',
    PHONE_NO        IS '固定电话',
    EMAIL           IS '电子邮箱',
    POSTAL_CD       IS '邮政编码',
    ADDRESS         IS '详细地址',
    RISK_LEVEL      IS '风险等级',
    RISK_RATE_DT    IS '评级日期',
    RISK_RATE_RSN   IS '评级原因',
    KYC_STAT        IS 'KYC状态',
    KYC_COMP_DT     IS 'KYC完成日期',
    KYC_NEXT_DT     IS '下次KYC日期',
    ACCT_STAT       IS '账户状态',
    STAT_CHG_DT     IS '状态变更日期',
    STAT_CHG_RSN    IS '状态变更原因',
    SANCTION_CHK    IS '制裁检查',
    SANCTION_LIST   IS '制裁名单',
    SANCTION_DT     IS '检查日期',
    CUST_MGR_ID     IS '客户经理',
    BRANCH_ID       IS '归属机构',
    VIP_LEVEL       IS 'VIP等级',
    OPEN_DT         IS '开户日期',
    OPEN_CHNL       IS '开户渠道',
    CLOSE_DT        IS '销户日期',
    CREATE_TS       IS '创建时间戳',
```

```
    CREATE_USER     IS '创建用户',
    UPDATE_TS       IS '更新时间戳',
    UPDATE_USER     IS '更新用户',
    UPDATE_PRG      IS '更新程序'
);

-- 创建索引
CREATE INDEX IDX_CUSTMAST_NAME ON CUSTMAST(CUST_NAME);
CREATE INDEX IDX_CUSTMAST_ID ON CUSTMAST(ID_TYPE, ID_NO);
CREATE INDEX IDX_CUSTMAST_MOBILE ON CUSTMAST(MOBILE_NO);
CREATE INDEX IDX_CUSTMAST_MGR ON CUSTMAST(CUST_MGR_ID);
CREATE INDEX IDX_CUSTMAST_BRANCH ON CUSTMAST(BRANCH_ID);
CREATE INDEX IDX_CUSTMAST_STAT ON CUSTMAST(ACCT_STAT);

-- 创建触发器 - 自动更新审计字段
CREATE OR REPLACE TRIGGER TRG_CUSTMAST_UPD
BEFORE UPDATE ON CUSTMAST
REFERENCING NEW ROW AS N OLD ROW AS O
FOR EACH ROW
BEGIN
    SET N.UPDATE_TS = CURRENT_TIMESTAMP;
    SET N.UPDATE_USER = USER;
END;
```

2.2.3 客户号生成服务

RPG IV - 客户号生成程序:

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('CIF');

// ============================================================
// 程序: GENCLNTID
// 描述: 客户号生成服务
// 功能: 根据客户类型生成唯一客户号
// ============================================================

// 程序接口参数
dcl-pi *n;
    inCustType  char(1)     const;  // 输入: 客户类型
    outCustID   char(20);           // 输出: 生成的客户号
    outRtnCode  char(2);            // 输出: 返回码 00-成功 其他-失败
    outRtnMsg   char(200);          // 输出: 返回消息
```

```
end-pi;

// 常量定义
dcl-c TYPE_INDIVIDUAL '1';
dcl-c TYPE_CORPORATE  '2';
dcl-c TYPE_FINANCIAL  '3';
dcl-c TYPE_INTERBANK  '4';

// 工作变量
dcl-s seqNumber packed(15);
dcl-s checkDigit packed(1);
dcl-s today packed(8);
dcl-s currentTime packed(6);
dcl-s branchCode char(4);
dcl-s workDate date;

// 数据结构 - 序列号控制表
dcl-ds CIFSEQ extname('CIFSEQ') qualified end-ds;

// SQL状态码
dcl-s SQLState char(5);

// ================================================================
// 主程序逻辑
// ================================================================

// 初始化输出参数
outRtnCode = '00';
outRtnMsg = '';
outCustID = '';

// 验证客户类型
if inCustType <> TYPE_INDIVIDUAL and
   inCustType <> TYPE_CORPORATE  and
   inCustType <> TYPE_FINANCIAL  and
   inCustType <> TYPE_INTERBANK;
    outRtnCode = '01';
    outRtnMsg = '无效的客户类型: ' + inCustType;
    return;
endif;

// 获取序列号
exsr getSequenceNumber;
if outRtnCode <> '00';
    return;
endif;
```

```
// 生成客户号
exsr generateCustomerID;

// 生成校验位
exsr calculateCheckDigit;

// 组合最终客户号
exsr formatCustomerID;

// 记录日志
exsr logGeneration;

return;

// ================================================================
// 获取序列号子程序
// ================================================================
begsr getSequenceNumber;

    // 使用序列对象获取唯一序列号
    exec SQL
        SELECT NEXT VALUE FOR CUST_ID_SEQ INTO :seqNumber
        FROM SYSIBM.SYSDUMMY1;

    if SQLState <> '00000';
        outRtnCode = '10';
        outRtnMsg = '获取序列号失败: ' + %trim(SQLState);
        leavesr;
    endif;

endsr;

// ================================================================
// 生成客户号子程序
// ================================================================
begsr generateCustomerID;

    // 获取当前日期
    exec SQL
        SELECT CURRENT DATE, CURRENT TIME
        INTO :workDate, :currentTime
        FROM SYSIBM.SYSDUMMY1;

    today = %int(%char(workDate : *iso0));
```

```
endsr;

// ============================================================
// 计算校验位子程序(模11算法)
// ============================================================
begsr calculateCheckDigit;

    dcl-s i packed(3);
    dcl-s sum packed(10);
    dcl-s baseStr varchar(20);
    dcl-s digit packed(1);
    dcl-s weight packed(2) dim(15) ctdata;

    // 加权因子: 7,9,10,5,8,4,2,1,6,3,7,9,10,5,8
    baseStr = %editc(today : 'X') + %trim(%editc(seqNumber : 'X'));

    sum = 0;
    for i = 1 to %len(%trim(baseStr));
        digit = %int(%subst(baseStr : i : 1));
        sum = sum + digit * weight(i);
    endfor;

    // 计算校验位
    checkDigit = 11 - %rem(sum : 11);
    if checkDigit = 10;
        checkDigit = 0;
    elseif checkDigit = 11;
        checkDigit = 1;
    endif;

endsr;

// ============================================================
// 格式化客户号子程序
// 客户号格式: 类型(1位) + 日期(8位) + 序号(7位) + 校验位(1位) + 填充
// 示例: 12026020700012345
// ============================================================
begsr formatCustomerID;

    // 组合客户号
    outCustID = inCustType +
                %editc(today : 'X') +
                %subst(%editc(seqNumber : 'X') : 9 : 7) +
                %editc(checkDigit : 'X');

    // 右填充空格至20位
```

```
    outCustID = %trim(outCustID);

endsr;

// ====================================================================
// 记录生成日志
// ====================================================================
begsr logGeneration;

    exec SQL
        INSERT INTO CIFIDLOG (
            GEN_TS,
            CUST_TYPE,
            CUST_ID,
            SEQ_NUMBER,
            GEN_USER,
            GEN_PRG
        ) VALUES (
            CURRENT_TIMESTAMP,
            :inCustType,
            :outCustID,
            :seqNumber,
            USER,
            'GENCLNTID'
        );

endsr;

// ====================================================================
// *INZSR - 初始化子程序
// ====================================================================
begsr *inzsr;
    // 程序初始化逻辑
endsr;

**CTDATA weight
079
108
406
203
**End
```

2.2.4 客户信息查询服务

SQL RPG - 客户信息查询:

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('CIF');

// ========================================================================
// 程序: CIFINQ
// 描述: 客户信息综合查询服务
// 支持: 按客户号、证件、手机号等多维度查询
// ========================================================================

dcl-f CUSTMAST usage(*input) keyed;
dcl-f CUSTADDR usage(*input) keyed;
dcl-f CUSTREL usage(*input) keyed;

// 查询参数结构
dcl-ds QueryParms qualified;
    searchType   char(2);    // 01-客户号 02-证件 03-手机号 04-名称
    searchValue  varchar(100);
    pageSize     packed(5)   inz(20);
    pageNum      packed(10)  inz(1);
end-ds;

// 输出结果结构
dcl-ds CustomerInfo qualified template;
    custID       char(20);
    custType     char(1);
    custName     char(100);
    idType       char(2);
    idNo         char(50);
    mobileNo     char(20);
    riskLevel    char(2);
    acctStat     char(1);
    kycStat      char(1);
    openDt       packed(8);
    branchID     char(10);
    custMgrID    char(10);
end-ds;

// 内部使用数组
dcl-ds customers likeds(CustomerInfo) dim(100);
dcl-s  customerCount packed(5);
```

```
// ============================================================
// 主程序入口
// ============================================================

// 按客户号精确查询
dcl-proc GetCustomerByID export;
    dcl-pi *n ind;
        inCustID    char(20) const;
        outCustInfo likeds(CustomerInfo);
        outRtnCode  char(2);
        outRtnMsg   char(200);
    end-pi;

    outRtnCode = '00';
    outRtnMsg = '';

    // 查询客户主表
    chain inCustID CUSTMAST;

    if not %found(CUSTMAST);
        outRtnCode = '91';
        outRtnMsg = '客户不存在: ' + %trim(inCustID);
        return *off;
    endif;

    // 填充返回数据
    outCustInfo.custID = CUST_ID;
    outCustInfo.custType = CUST_TYPE;
    outCustInfo.custName = CUST_NAME;
    outCustInfo.idType = ID_TYPE;
    outCustInfo.idNo = ID_NO;
    outCustInfo.mobileNo = MOBILE_NO;
    outCustInfo.riskLevel = RISK_LEVEL;
    outCustInfo.acctStat = ACCT_STAT;
    outCustInfo.kycStat = KYC_STAT;
    outCustInfo.openDt = OPEN_DT;
    outCustInfo.branchID = BRANCH_ID;
    outCustInfo.custMgrID = CUST_MGR_ID;

    return *on;
end-proc;

// ============================================================
// 按证件信息查询
// ============================================================
```

```
dcl-proc GetCustomerByIDCard export;
    dcl-pi *n ind;
        inIdType    char(2) const;
        inIdNo      varchar(50) const;
        outCustInfo likeds(CustomerInfo);
        outRtnCode  char(2);
        outRtnMsg   char(200);
    end-pi;

    dcl-s custID char(20);

    outRtnCode = '00';
    outRtnMsg = '';

    // 使用SQL查询(证件组合唯一)
    exec SQL
        SELECT CUST_ID INTO :custID
        FROM CUSTMAST
        WHERE ID_TYPE = :inIdType
          AND ID_NO = :inIdNo
          AND ACCT_STAT <> '9'
        FETCH FIRST 1 ROW ONLY;

    if SQLCODE <> 0;
        outRtnCode = '91';
        outRtnMsg = '未找到证件信息: ' + %trim(inIdType) + '/' +
%trim(inIdNo);
        return *off;
    endif;

    // 递归调用客户号查询
    return GetCustomerByID(custID : outCustInfo : outRtnCode :
outRtnMsg);
end-proc;

// ============================================================
// 模糊查询客户列表
// ============================================================
dcl-proc SearchCustomers export;
    dcl-pi *n packed(5);
        inParms     likeds(QueryParms) const;
        outList     likeds(CustomerInfo) dim(100);
        outTotalCnt packed(10);
        outRtnCode  char(2);
        outRtnMsg   char(200);
    end-pi;
```

```
    dcl-s whereClause varchar(1000);
    dcl-s sqlStmt varchar(2000);
    dcl-s offset packed(10);

    outRtnCode = '00';
    outRtnMsg = '';
    outTotalCnt = 0;
    customerCount = 0;

    // 构建WHERE条件
    whereClause = ' WHERE ACCT_STAT <> ''9'' ';

    select;
        when inParms.searchType = '01'; // 客户号
            whereClause = %trim(whereClause) +
                ' AND CUST_ID LIKE ''%' + %trim(inParms.searchValue) +
'%''';

        when inParms.searchType = '02'; // 证件号码
            whereClause = %trim(whereClause) +
                ' AND ID_NO LIKE ''%' + %trim(inParms.searchValue) +
'%''';

        when inParms.searchType = '03'; // 手机号
            whereClause = %trim(whereClause) +
                ' AND MOBILE_NO LIKE ''%' + %trim(inParms.searchValue)
+ '%''';

        when inParms.searchType = '04'; // 客户名称
            whereClause = %trim(whereClause) +
                ' AND (CUST_NAME LIKE ''%' + %trim(inParms.searchValue)
+
                '%'' OR CUST_PY LIKE ''%' + %trim(inParms.searchValue)
+ '%'')';

        other;
            outRtnCode = '02';
            outRtnMsg = '无效的查询类型';
            return 0;
    endsl;

    // 计算分页偏移
    offset = (inParms.pageNum - 1) * inParms.pageSize;

    // 构建并执行SQL
```

```
    sqlStmt = 'SELECT CUST_ID, CUST_TYPE, CUST_NAME, ID_TYPE, ' +
              'ID_NO, MOBILE_NO, RISK_LEVEL, ACCT_STAT, KYC_STAT, ' +
              'OPEN_DT, BRANCH_ID, CUST_MGR_ID ' +
              'FROM CUSTMAST' +
              %trim(whereClause) +
              ' ORDER BY CUST_ID' +
              ' OFFSET ' + %char(offset) + ' ROWS' +
              ' FETCH NEXT ' + %char(inParms.pageSize) + ' ROWS ONLY';

    exec SQL PREPARE S1 FROM :sqlStmt;
    exec SQL DECLARE C1 CURSOR FOR S1;
    exec SQL OPEN C1;

    // 循环读取结果
    dou SQLCODE <> 0 or customerCount >= inParms.pageSize;
        exec SQL FETCH C1 INTO :customers(customerCount + 1);
        if SQLCODE = 0;
            customerCount = customerCount + 1;
        endif;
    enddo;

    exec SQL CLOSE C1;

    // 获取总记录数
    sqlStmt = 'SELECT COUNT(*) FROM CUSTMAST' + %trim(whereClause);
    exec SQL EXECUTE IMMEDIATE :sqlStmt INTO :outTotalCnt;

    outList = customers;

    return customerCount;
end-proc;

// ============================================================
// 获取客户关系图谱
// ============================================================
dcl-proc GetCustomerRelations export;
    dcl-pi *n packed(5);
        inCustID    char(20) const;
        outRelations likeds(RelationInfo) dim(50);
        outRtnCode  char(2);
        outRtnMsg   char(200);
    end-pi;

    // 关系信息数据结构
    dcl-ds RelationInfo qualified template;
        relCustID   char(20);
```

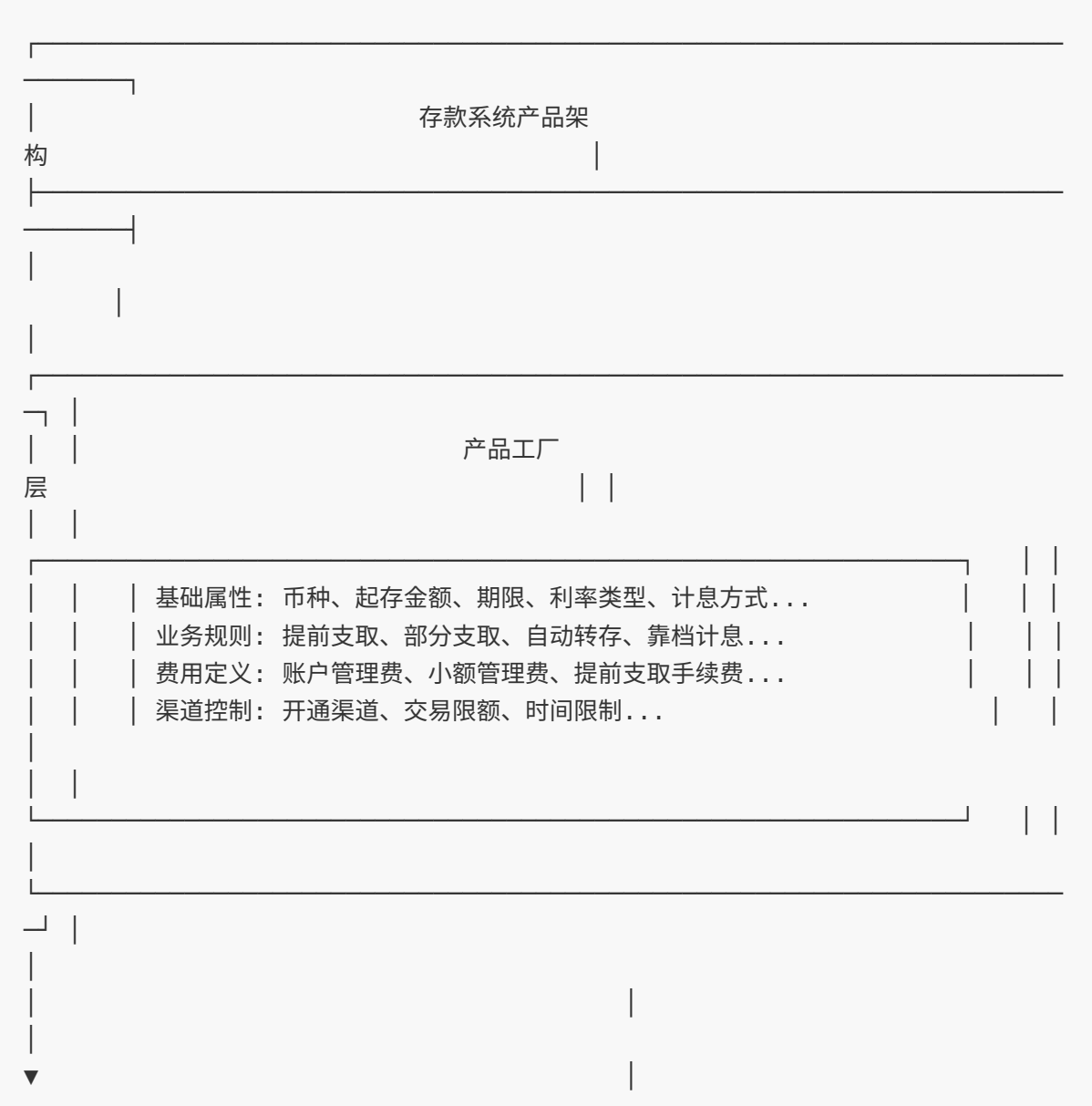
```
        relCustName char(100);
        relType     char(2);    // 01-配偶 02-子女 03-父母 04-股东...
        relTypeDesc char(50);
        relStrength char(1);    // S-强M-中W-弱
        relStartDt  packed(8);
    end-ds;

    dcl-ds rel likeds(RelationInfo) dim(50);
    dcl-s relCnt packed(5);

    relCnt = 0;
    outRtnCode = '00';

    // 查询双向关系
    exec SQL
        DECLARE REL_CUR CURSOR FOR
        SELECT
            CASE WHEN FROM_CUST = :inCustID THEN TO_CUST ELSE FROM_CUST
END,
            C.CUST_NAME,
            CASE WHEN FROM_CUST = :inCustID THEN REL_TYPE ELSE
REV_REL_TYPE END,
            P.PAR_DESC,
            R.REL_STRENGTH,
            R.START_DT
        FROM CUSTREL R
        JOIN CUSTMAST C ON (
            CASE WHEN FROM_CUST = :inCustID THEN TO_CUST ELSE FROM_CUST
END
        ) = C.CUST_ID
        JOIN PARMAST P ON (
            CASE WHEN FROM_CUST = :inCustID THEN REL_TYPE ELSE
REV_REL_TYPE END
        ) = P.PAR_CODE AND P.PAR_TYPE = 'RELTYPE'
        WHERE FROM_CUST = :inCustID OR TO_CUST = :inCustID
        ORDER BY R.REL_STRENGTH DESC, R.START_DT DESC;

    exec SQL OPEN REL_CUR;

    dou SQLCODE <> 0 or relCnt >= 50;
        exec SQL FETCH REL_CUR INTO :rel(relCnt + 1);
        if SQLCODE = 0;
            relCnt = relCnt + 1;
        endif;
    enddo;
```

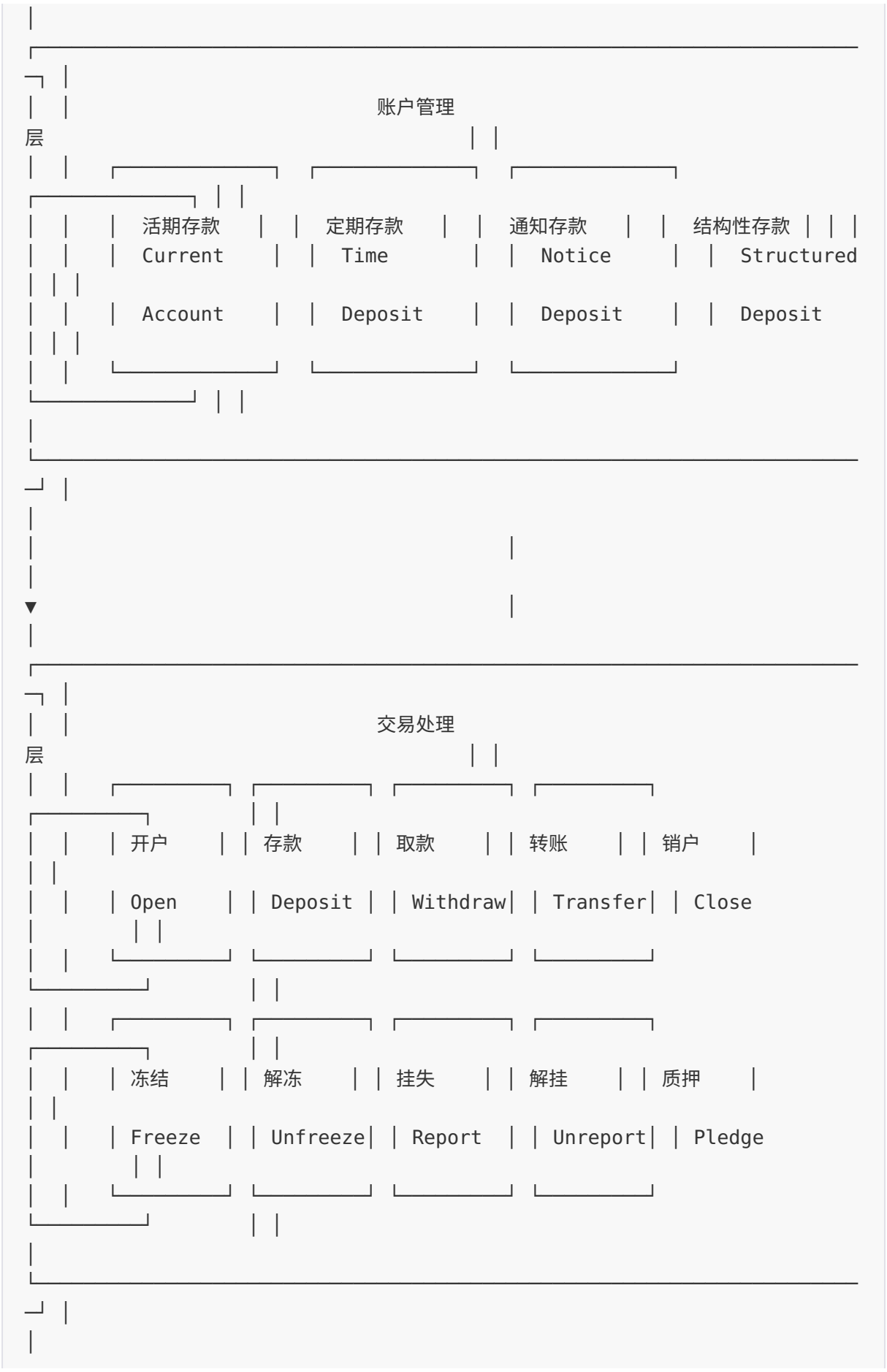
```
    exec SQL CLOSE REL_CUR;

    outRelations = rel;

    return relCnt;
end-proc;
```

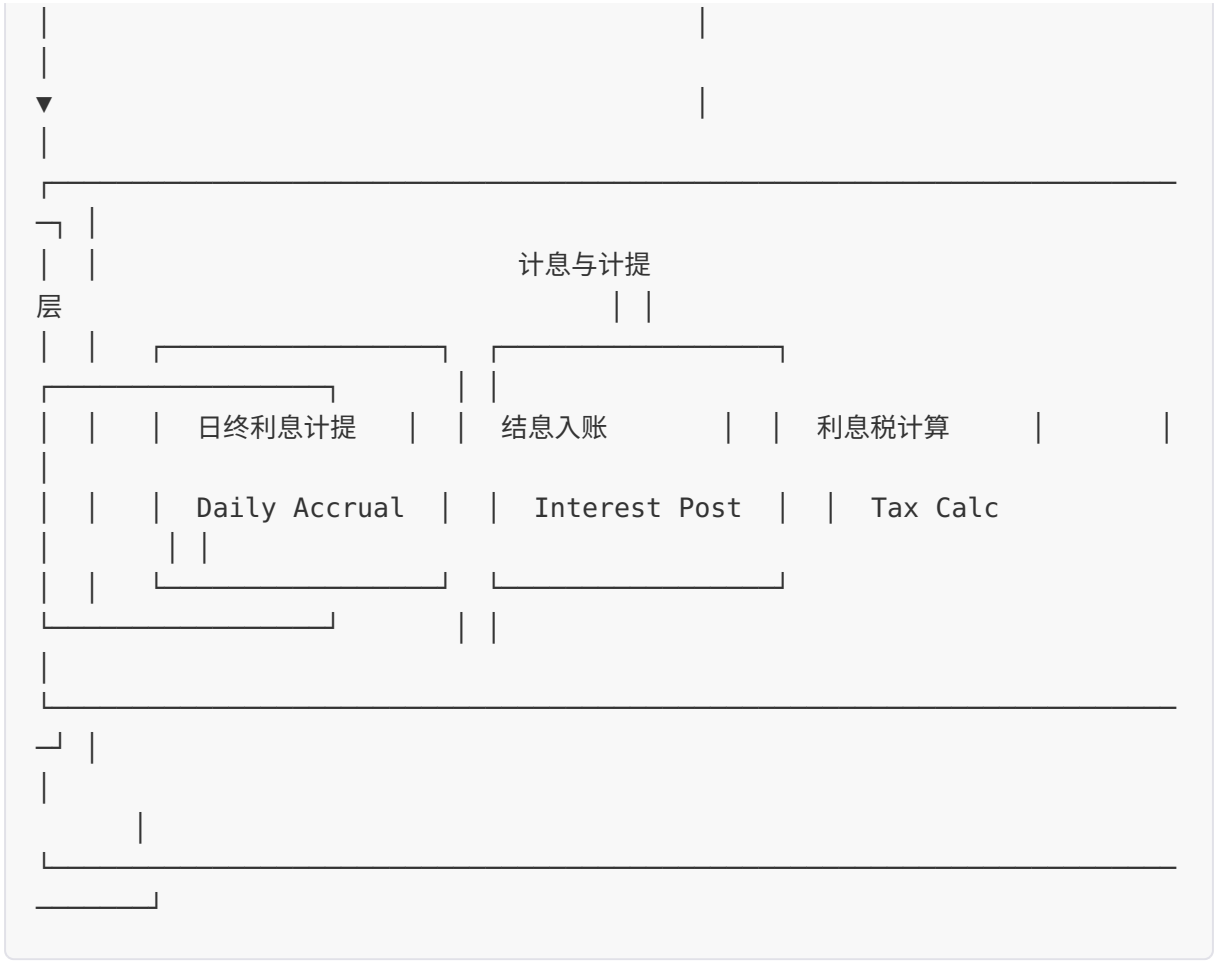
2.3 存款系统

2.3.1 存款产品架构

存款系统是银行最基础的业务模块，包括活期存款、定期存款、通知存款、结构性存款等多种产品类型。




账户管理层
活期存款 Current Account
定期存款 Time Deposit
通知存款 Notice Deposit
结构性存款 Structured Deposit
交易处理层
开户 Open
存款 Deposit
取款 Withdraw
转账 Transfer
销户 Close
冻结 Freeze
解冻 Unfreeze
挂失 Report
解挂 Unreport
质押 Pledge



2.3.2 账户主表设计

SQL DDL - 存款账户主表:

```
-- ============================================================
-- 存款账户主表 - ACCTMAST
-- ============================================================

CREATE TABLE ACCTMAST (
    -- 主键
    ACCT_NO             CHAR(32)        NOT NULL,

    -- 客户信息
    CUST_ID             CHAR(20)        NOT NULL,
    CUST_NAME           VARCHAR(100),

    -- 产品信息
    PROD_CD             CHAR(10)        NOT NULL,
    PROD_TYPE           CHAR(2)         NOT NULL,  -- 01-活期 02-定期
03-通知...
```

```
    -- 币种与余额
    CCY_CD              CHAR(3)         NOT NULL DEFAULT 'CNY',
    ACCT_BAL            DECIMAL(19, 2)  DEFAULT 0,
    AVAIL_BAL           DECIMAL(19, 2)  DEFAULT 0,
    FROZEN_AMT          DECIMAL(19, 2)  DEFAULT 0,
    HOLD_AMT            DECIMAL(19, 2)  DEFAULT 0,

    -- 利率信息
    INT_RATE_TYPE       CHAR(1)         DEFAULT '0',  -- 0-固定 1-浮动
    INT_RATE            DECIMAL(9, 6)   DEFAULT 0,
    INT_RATE_SPREAD     DECIMAL(9, 6)   DEFAULT 0,
    ODR_INT_RATE        DECIMAL(9, 6)   DEFAULT 0,    -- 透支利率

    -- 开户信息
    OPEN_DT             DECIMAL(8, 0)   NOT NULL,
    OPEN_CHNL           CHAR(2)         DEFAULT '01',
    OPEN_BRANCH         CHAR(10)        NOT NULL,

    -- 定期/通知存款特有
    MATURITY_DT         DECIMAL(8, 0),
    TERM_MONTHS         DECIMAL(3, 0),
    TERM_DAYS           DECIMAL(5, 0),
    AUTO_RENEW_FLAG     CHAR(1)         DEFAULT '0',
    RENEW_TIMES         DECIMAL(3, 0)   DEFAULT 0,
    BASE_ACCT_NO        CHAR(32),                   -- 主账户(子账户关联)

    -- 账户状态
    ACCT_STAT           CHAR(1)         DEFAULT '0',  -- 0-正常 1-预开户
2-不动户 3-冻结...
    STAT_CHG_DT         DECIMAL(8, 0),

    -- 控制信息
    PWD_FLAG            CHAR(1)         DEFAULT '1',
    CHEQUE_FLAG         CHAR(1)         DEFAULT '0',
    ODR_LIMIT_AMT       DECIMAL(19, 2)  DEFAULT 0,

    -- 限额设置
    DAILY_DR_LIMIT      DECIMAL(19, 2),
    DAILY_DR_AMT        DECIMAL(19, 2)  DEFAULT 0,    -- 当日已用额度
    SINGLE_DR_LIMIT     DECIMAL(19, 2),

    -- 账户介质
    CARD_NO             CHAR(19),
    PASSBOOK_NO         CHAR(20),
```

```
    -- 计息信息
    LAST_INT_DT         DECIMAL(8, 0),
    NEXT_INT_DT         DECIMAL(8, 0),
    ACCRUED_INT         DECIMAL(19, 2)  DEFAULT 0,

    -- 利息税信息
    TAX_RATE            DECIMAL(5, 2)   DEFAULT 20,   -- 利息税率%
    TAX_EXEMPT_FLAG     CHAR(1)         DEFAULT '0',  -- 免税标志

    -- 计息积数
    INT_BASE_CURR       DECIMAL(23, 2)  DEFAULT 0,    -- 当期积数
    INT_BASE_TOTAL      DECIMAL(23, 2)  DEFAULT 0,    -- 累计积数
    LAST_DR_CR_DT       DECIMAL(8, 0),                -- 上次动账日

    -- 销户信息
    CLOSE_DT            DECIMAL(8, 0),
    CLOSE_RSN           VARCHAR(200),

    -- 审计字段
    CREATE_TS           TIMESTAMP       DEFAULT CURRENT_TIMESTAMP,
    CREATE_USER         VARCHAR(20)     DEFAULT USER,
    UPDATE_TS           TIMESTAMP       DEFAULT CURRENT_TIMESTAMP,
    UPDATE_USER         VARCHAR(20)     DEFAULT USER,
    UPDATE_PRG          VARCHAR(20),

    -- 约束
    CONSTRAINT PK_ACCTMAST PRIMARY KEY (ACCT_NO),
    CONSTRAINT FK_ACCTMAST_CUST FOREIGN KEY (CUST_ID)
        REFERENCES CUSTMAST(CUST_ID),
    CONSTRAINT CK_ACCT_STAT CHECK (ACCT_STAT IN ('0', '1', '2', '3',
'4', '5', '9')),
    CONSTRAINT CK_PROD_TYPE CHECK (PROD_TYPE IN ('01', '02', '03',
'04', '05'))

) RCDFMT ACCTMASTR;

-- 索引
CREATE INDEX IDX_ACCTMAST_CUST ON ACCTMAST(CUST_ID);
CREATE INDEX IDX_ACCTMAST_PROD ON ACCTMAST(PROD_CD);
CREATE INDEX IDX_ACCTMAST_STAT ON ACCTMAST(ACCT_STAT);
CREATE INDEX IDX_ACCTMAST_CARD ON ACCTMAST(CARD_NO) WHERE CARD_NO IS
NOT NULL;

-- 标签
LABEL ON TABLE ACCTMAST IS '存款账户主表';
```

```
-- 触发器 - 账户余额变动日志
create or replace trigger trg_acctmast_bal_chg
after update of acct_bal on acctmast
referencing new row as n old row as o
for each row
when (n.acct_bal <> o.acct_bal)
begin
    insert into acctbalchglog (
        chg_ts,
        acct_no,
        old_bal,
        new_bal,
        chg_amt,
        user_id
    ) values (
        current_timestamp,
        n.acct_no,
        o.acct_bal,
        n.acct_bal,
        n.acct_bal - o.acct_bal,
        user
    );
end;
```

2.3.3 存款交易处理程序

RPG IV - 存款交易处理:

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('DEPOSIT');
ctl-opt bnddir('DEPOSIT');

// ========================================================================
// 程序: DEPTRANS
// 描述: 存款交易处理核心程序
// 功能: 处理存款、取款、转账等账务交易
// ========================================================================

// 复制书
copy 'DEPCONST'                 // 存款常量定义
copy 'ERRHND'                   // 错误处理
copy 'TRANCTRL'                 // 交易控制
```

```
// 文件定义
dcl-f ACCTMAST usage(*update) keyed;
dcl-f ACCTDTL usage(*output);
dcl-f TRANLOG usage(*output);
dcl-f GLTRANS usage(*output);

// 程序状态数据结构
dcl-ds PgmStat PSDS;
    PgmName     char(10)    pos(1);
    PgmLib      char(10)    pos(81);
    JobName     char(10)    pos(244);
    JobUser     char(10)    pos(254);
    JobNbr      char(6)     pos(264);
    CurrUser    char(10)    pos(358);
end-ds;

// 交易请求结构
dcl-ds TransReq qualified;
    transRefNo      char(32);       // 交易参考号
    transType       char(4);        // 交易类型
    transDt         packed(8);      // 交易日期
    transTm         packed(6);      // 交易时间
    acctNo          char(32);       // 账号
    ccyCd           char(3);        // 币种
    transAmt        packed(19:2);   // 交易金额
    drCrFlag        char(1);        // 借贷标志 D-借 C-贷
    transChnl       char(2);        // 交易渠道
    transBranch     char(10);       // 交易机构
    tellerId        char(10);       // 柜员号
    oppAcctNo       char(32);       // 对方账号(转账用)
    oppBankNo       char(12);       // 对方行号
    transSummary    char(100);      // 交易摘要
    transMemo       varchar(500);   // 交易备注
    clientIP        char(15);       // 客户端IP
    clientDevice    char(100);      // 客户端设备信息
end-ds;

// 交易响应结构
dcl-ds TransResp qualified;
    rtnCode         char(2);
    rtnMsg          char(200);
    transSeq        char(20);       // 交易流水号
    acctBal         packed(19:2);   // 交易后余额
    availBal        packed(19:2);   // 可用余额
    transCertNo     char(50);       // 交易凭证号
```

```
end-ds;

// 账户数据结构
dcl-ds AcctData likeds(ACCTMAST_T) inz;

// 常量
dcl-c TRANS_TYPE_DEPOSIT    '1001';     // 存款
dcl-c TRANS_TYPE_WITHDRAW   '1002';     // 取款
dcl-c TRANS_TYPE_TRANSFER   '1003';     // 转账
dcl-c TRANS_TYPE_INQ        '1099';     // 查询

dcl-c DR_FLAG               'D';
dcl-c CR_FLAG               'C';

// ================================================================
// 主程序入口
// ================================================================

// 导出过程: 存款交易处理
dcl-proc ProcessDeposit export;
    dcl-pi *n likeds(TransResp);
        req likeds(TransReq) const;
    end-pi;

    dcl-ds resp likeds(TransResp) inz;

    // 初始化
    resp.rtnCode = '00';
    resp.rtnMsg = '';

    // 参数校验
    if not ValidateRequest(req : resp);
        return resp;
    endif;

    // 读取并锁定账户
    if not LockAccount(req.acctNo : resp);
        return resp;
    endif;

    // 检查交易限额
    if not CheckTransactionLimit(req : resp);
        unlock acctmast;
        return resp;
    endif;
```

```
    // 执行交易处理
    select;
        when req.transType = TRANS_TYPE_DEPOSIT;
            resp = ProcessCreditTransaction(req);
        when req.transType = TRANS_TYPE_WITHDRAW;
            resp = ProcessDebitTransaction(req);
        when req.transType = TRANS_TYPE_TRANSFER;
            resp = ProcessTransferTransaction(req);
        other;
            resp.rtnCode = '11';
            resp.rtnMsg = '不支持的交易类型: ' + req.transType;
    endsl;

    // 解锁账户
    unlock acctmast;

    return resp;
end-proc;

// ====================================================================
// 参数校验
// ====================================================================
dcl-proc ValidateRequest;
    dcl-pi *n ind;
        req     likeds(TransReq) const;
        resp    likeds(TransResp);
    end-pi;

    // 检查必输字段
    if req.transRefNo = '';
        resp.rtnCode = '01';
        resp.rtnMsg = '交易参考号不能为空';
        return *off;
    endif;

    if req.acctNo = '';
        resp.rtnCode = '02';
        resp.rtnMsg = '账号不能为空';
        return *off;
    endif;

    if req.transAmt <= 0;
        resp.rtnCode = '03';
        resp.rtnMsg = '交易金额必须大于0';
        return *off;
    endif;
```

```
    if req.drCrFlag <> DR_FLAG and req.drCrFlag <> CR_FLAG;
        resp.rtnCode = '04';
        resp.rtnMsg = '无效的借贷标志';
        return *off;
    endif;

    // 检查幂等性 - 防止重复交易
    if CheckDuplicateTrans(req.transRefNo);
        resp.rtnCode = '05';
        resp.rtnMsg = '重复交易: ' + req.transRefNo;
        return *off;
    endif;

    return *on;
end-proc;

// ====================================================================
// 锁定账户
// ====================================================================
dcl-proc LockAccount;
    dcl-pi *n ind;
        acctNo  char(32) const;
        resp    likeds(TransResp);
    end-pi;

    // 读取并锁定账户记录
    chain (acctNo) ACCTMAST;

    if not %found(ACCTMAST);
        resp.rtnCode = '21';
        resp.rtnMsg = '账户不存在: ' + acctNo;
        return *off;
    endif;

    // 检查账户状态
    select;
        when ACCT_STAT = '1';   // 预开户
            resp.rtnCode = '22';
            resp.rtnMsg = '账户未激活';
            return *off;
        when ACCT_STAT = '2';   // 不动户
            resp.rtnCode = '23';
            resp.rtnMsg = '不动户，请先激活';
            return *off;
        when ACCT_STAT = '3';   // 冻结
```

```
            resp.rtnCode = '24';
            resp.rtnMsg = '账户已冻结';
            return *off;
        when ACCT_STAT = '4';   // 止付
            resp.rtnCode = '25';
            resp.rtnMsg = '账户已止付';
            return *off;
        when ACCT_STAT = '5';   // 挂失
            resp.rtnCode = '26';
            resp.rtnMsg = '账户已挂失';
            return *off;
        when ACCT_STAT = '9';   // 销户
            resp.rtnCode = '27';
            resp.rtnMsg = '账户已销户';
            return *off;
    endsl;

    return *on;
end-proc;

// ============================================================
// 检查交易限额
// ============================================================
dcl-proc CheckTransactionLimit;
    dcl-pi *n ind;
        req     likeds(TransReq) const;
        resp    likeds(TransResp);
    end-pi;

    // 取款和转账需要检查限额
    if req.transType = TRANS_TYPE_DEPOSIT;
        return *on;  // 存款无上限限制
    endif;

    // 检查单笔限额
    if SINGLE_DR_LIMIT > 0 and req.transAmt > SINGLE_DR_LIMIT;
        resp.rtnCode = '31';
        resp.rtnMsg = '超过单笔交易限额: ' +
                      %trim(%editc(SINGLE_DR_LIMIT:'J'));
        return *off;
    endif;

    // 检查日累计限额
    if DAILY_DR_LIMIT > 0;
        if (DAILY_DR_AMT + req.transAmt) > DAILY_DR_LIMIT;
            resp.rtnCode = '32';
```

```
            resp.rtnMsg = '超过日累计交易限额: ' +
                          %trim(%editc(DAILY_DR_LIMIT:'J'));
            return *off;
        endif;
    endif;

    // 检查余额(取款和转账)
    if req.drCrFlag = DR_FLAG and req.transAmt > AVAIL_BAL;
        resp.rtnCode = '33';
        resp.rtnMsg = '可用余额不足: ' + %trim(%editc(AVAIL_BAL:'J'));
        return *off;
    endif;

    return *on;
end-proc;

// ====================================================================
// 处理贷记交易(存款)
// ====================================================================
dcl-proc ProcessCreditTransaction;
    dcl-pi *n likeds(TransResp);
        req     likeds(TransReq) const;
    end-pi;

    dcl-ds resp likeds(TransResp) inz;
    dcl-s transSeq char(20);

    // 生成交易流水号
    transSeq = GenerateTransSeq();

    // 更新账户余额
    ACCT_BAL = ACCT_BAL + req.transAmt;
    AVAIL_BAL = AVAIL_BAL + req.transAmt;
    LAST_DR_CR_DT = req.transDt;
    UPDATE_TS = %timestamp();
    UPDATE_USER = CurrUser;
    UPDATE_PRG = PgmName;

    update ACCTMASTR;

    if %error;
        resp.rtnCode = '41';
        resp.rtnMsg = '更新账户余额失败';
        return resp;
    endif;
```

```
    // 写入交易明细
    WriteTransactionDetail(req : transSeq : ACCT_BAL);

    // 更新计息积数
    UpdateInterestBase(req.acctNo : req.transDt);

    // 生成会计分录
    CreateGLEntry(req : transSeq);

    // 组装响应
    resp.rtnCode = '00';
    resp.rtnMsg = '交易成功';
    resp.transSeq = transSeq;
    resp.acctBal = ACCT_BAL;
    resp.availBal = AVAIL_BAL;
    resp.transCertNo = 'DEP' + transSeq;

    return resp;
end-proc;

// ====================================================================
// 处理借记交易(取款)
// ====================================================================
dcl-proc ProcessDebitTransaction;
    dcl-pi *n likeds(TransResp);
        req     likeds(TransReq) const;
    end-pi;

    dcl-ds resp likeds(TransResp) inz;
    dcl-s transSeq char(20);

    // 生成交易流水号
    transSeq = GenerateTransSeq();

    // 更新账户余额
    ACCT_BAL = ACCT_BAL - req.transAmt;
    AVAIL_BAL = AVAIL_BAL - req.transAmt;
    DAILY_DR_AMT = DAILY_DR_AMT + req.transAmt;
    LAST_DR_CR_DT = req.transDt;
    UPDATE_TS = %timestamp();
    UPDATE_USER = CurrUser;
    UPDATE_PRG = PgmName;

    update ACCTMASTR;

    if %error;
```

```
        resp.rtnCode = '42';
        resp.rtnMsg = '更新账户余额失败';
        return resp;
    endif;

    // 写入交易明细
    WriteTransactionDetail(req : transSeq : ACCT_BAL);

    // 更新计息积数
    UpdateInterestBase(req.acctNo : req.transDt);

    // 生成会计分录
    CreateGLEntry(req : transSeq);

    // 组装响应
    resp.rtnCode = '00';
    resp.rtnMsg = '交易成功';
    resp.transSeq = transSeq;
    resp.acctBal = ACCT_BAL;
    resp.availBal = AVAIL_BAL;
    resp.transCertNo = 'WDR' + transSeq;

    return resp;
end-proc;

// ====================================================================
// 处理转账交易
// ====================================================================
dcl-proc ProcessTransferTransaction;
    dcl-pi *n likeds(TransResp);
        req     likeds(TransReq) const;
    end-pi;

    dcl-ds resp likeds(TransResp) inz;
    dcl-s drTransSeq char(20);
    dcl-s crTransSeq char(20);

    // 转账处理 - 先借后贷原则

    // 1. 处理付款方账户(借记)
    drTransSeq = GenerateTransSeq();

    ACCT_BAL = ACCT_BAL - req.transAmt;
    AVAIL_BAL = AVAIL_BAL - req.transAmt;
    DAILY_DR_AMT = DAILY_DR_AMT + req.transAmt;
    LAST_DR_CR_DT = req.transDt;
```

```
    UPDATE_TS = %timestamp();
    UPDATE_USER = CurrUser;
    UPDATE_PRG = PgmName;

    update ACCTMASTR;

    WriteTransactionDetail(req : drTransSeq : ACCT_BAL);
    UpdateInterestBase(req.acctNo : req.transDt);
    CreateGLEntry(req : drTransSeq);

    unlock acctmast;

    // 2. 处理收款方账户(贷记)
    chain (req.oppAcctNo) ACCTMAST;

    if %found(ACCTMAST);
        crTransSeq = GenerateTransSeq();

        ACCT_BAL = ACCT_BAL + req.transAmt;
        AVAIL_BAL = AVAIL_BAL + req.transAmt;
        LAST_DR_CR_DT = req.transDt;
        UPDATE_TS = %timestamp();
        UPDATE_USER = CurrUser;
        UPDATE_PRG = PgmName;

        update ACCTMASTR;

        // 构造反向请求
        dcl-ds oppReq likeds(TransReq);
        oppReq = req;
        oppReq.acctNo = req.oppAcctNo;
        oppReq.drCrFlag = CR_FLAG;
        oppReq.oppAcctNo = req.acctNo;

        WriteTransactionDetail(oppReq : crTransSeq : ACCT_BAL);
        UpdateInterestBase(req.oppAcctNo : req.transDt);
        CreateGLEntry(oppReq : crTransSeq);

        unlock acctmast;
    endif;

    // 组装响应
    resp.rtnCode = '00';
    resp.rtnMsg = '转账成功';
    resp.transSeq = drTransSeq;
    resp.acctBal = ACCT_BAL;
```

```
    resp.availBal = AVAIL_BAL;
    resp.transCertNo = 'TRF' + drTransSeq;

    return resp;
end-proc;

// ================================================================
// 生成交易流水号
// ================================================================
dcl-proc GenerateTransSeq;
    dcl-pi *n char(20) end-pi;

    dcl-s seqNum packed(15);
    dcl-s today packed(8);
    dcl-s time packed(6);
    dcl-s transSeq char(20);

    // 获取序列号
    exec SQL
        SELECT NEXT VALUE FOR TRANS_SEQ INTO :seqNum
        FROM SYSIBM.SYSDUMMY1;

    // 获取当前日期时间
    exec SQL
        SELECT CURRENT DATE, CURRENT TIME
        INTO :today, :time
        FROM SYSIBM.SYSDUMMY1;

    // 组合流水号: 日期(8) + 时间(6) + 序列号后6位
    transSeq = %editc(today : 'X') +
               %editc(time : 'X') +
               %subst(%editc(seqNum : 'X') : 10 : 6);

    return transSeq;
end-proc;

// ================================================================
// 写入交易明细
// ================================================================
dcl-proc WriteTransactionDetail;
    dcl-pi *n;
        req         likeds(TransReq) const;
        transSeq    char(20) const;
        afterBal    packed(19:2) const;
    end-pi;
```

```
    // 映射到输出文件字段
    DTL_SEQ = transSeq;
    DTL_ACCT_NO = req.acctNo;
    DTL_TRANS_TYPE = req.transType;
    DTL_TRANS_DT = req.transDt;
    DTL_TRANS_TM = req.transTm;
    DTL_CCY_CD = req.ccyCd;
    DTL_TRANS_AMT = req.transAmt;
    DTL_DR_CR_FLAG = req.drCrFlag;
    DTL_BEF_BAL = afterBal - req.transAmt;
    if req.drCrFlag = DR_FLAG;
        DTL_BEF_BAL = afterBal + req.transAmt;
    endif;
    DTL_AFT_BAL = afterBal;
    DTL_TRANS_CHNL = req.transChnl;
    DTL_TRANS_BRANCH = req.transBranch;
    DTL_TELLER_ID = req.tellerId;
    DTL_OPP_ACCT_NO = req.oppAcctNo;
    DTL_OPP_BANK_NO = req.oppBankNo;
    DTL_TRANS_SUMMARY = req.transSummary;
    DTL_TRANS_MEMO = req.transMemo;
    DTL_CLIENT_IP = req.clientIP;
    DTL_CLIENT_DEVICE = req.clientDevice;
    DTL_CREATE_TS = %timestamp();

    write ACCTDTLR;

end-proc;

// ============================================================
// 更新计息积数
// ============================================================
dcl-proc UpdateInterestBase;
    dcl-pi *n;
        acctNo      char(32) const;
        transDt     packed(8) const;
    end-pi;

    // 调用计息积数更新服务
    // 积数 = 余额 × 天数
    // 当发生余额变动时，累计当期积数并重新开始计算

    exec SQL
        UPDATE ACCTMAST
        SET INT_BASE_CURR = INT_BASE_CURR +
            (ACCT_BAL * (DAYS(:transDt) - DAYS(LAST_DR_CR_DT))),
```

```
            LAST_DR_CR_DT = :transDt
        WHERE ACCT_NO = :acctNo;

end-proc;

// ====================================================================
// 生成会计分录
// ====================================================================
dcl-proc CreateGLEntry;
    dcl-pi *n;
        req         likeds(TransReq) const;
        transSeq    char(20) const;
    end-pi;

    // 根据交易类型映射会计科目
    // 这里调用总账系统的接口生成会计分录

    // 示例: 存款
    // 借: 现金/清算科目
    // 贷: 客户存款科目

    // 示例: 取款
    // 借: 客户存款科目
    // 贷: 现金/清算科目

end-proc;

// ====================================================================
// 检查重复交易
// ====================================================================
dcl-proc CheckDuplicateTrans;
    dcl-pi *n ind;
        transRefNo  char(32) const;
    end-pi;

    dcl-s cnt packed(5);

    exec SQL
        SELECT COUNT(*) INTO :cnt
        FROM TRANLOG
        WHERE TRANS_REF_NO = :transRefNo
          AND TRANS_DT = CURRENT DATE;

    return (cnt > 0);
end-proc;
```

2.3.4 利息计算引擎

RPG IV - 利息计算程序:

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('INTEREST');

// ============================================================
// 程序: INTCALC
// 描述: 利息计算引擎
// 功能: 计算各类存款利息、利息税
// ============================================================

// 计息方法常量
dcl-c METHOD_ACTUAL_360    'A360';  // 实际天数/360
dcl-c METHOD_ACTUAL_365    'A365';  // 实际天数/365
dcl-c METHOD_30_360        '30360'; // 30/360
dcl-c METHOD_ACTUAL_ACTUAL 'AA';    // 实际天数/实际天数

// 利息计算结果结构
dcl-ds InterestResult qualified template;
    acctNo              char(32);
    intStartDt          packed(8);
    intEndDt            packed(8);
    intDays             packed(5);
    principal           packed(19:2);
    intRate             packed(9:6);
    grossInterest       packed(19:2);
    taxRate             packed(5:2);
    taxAmount           packed(19:2);
    netInterest         packed(19:2);
    calcMethod          char(5);
end-ds;

// ============================================================
// 主程序: 日终批量计息
// ============================================================
dcl-proc DailyInterestAccrual export;
    dcl-pi *n;
        inCalcDt    packed(8) const;
        outRtnCode  char(2);
        outRtnMsg   char(200);
    end-pi;
```

```
    dcl-s processedCnt packed(10);
    dcl-s errorCnt packed(10);

    outRtnCode = '00';
    outRtnMsg = '';
    processedCnt = 0;
    errorCnt = 0;

    // 声明游标遍历所有需要计息的账户
    exec SQL
        DECLARE ACCT_CUR CURSOR FOR
        SELECT ACCT_NO, CCY_CD, ACCT_BAL, INT_RATE, INT_BASE_CURR,
               TAX_RATE, TAX_EXEMPT_FLAG, PROD_TYPE
        FROM ACCTMAST
        WHERE ACCT_STAT = '0'                    // 正常状态账户
          AND INT_RATE > 0                       // 有利率的账户
          AND (LAST_INT_DT IS NULL OR LAST_INT_DT < :inCalcDt)
        FOR UPDATE OF ACCRUED_INT, INT_BASE_CURR, LAST_INT_DT;

    exec SQL OPEN ACCT_CUR;

    dou SQLCODE <> 0;
        dcl-ds acctData qualified;
            acctNo          char(32);
            ccyCd           char(3);
            acctBal         packed(19:2);
            intRate         packed(9:6);
            intBaseCurr     packed(23:2);
            taxRate         packed(5:2);
            taxExemptFlag   char(1);
            prodType        char(2);
        end-ds;

        exec SQL FETCH ACCT_CUR INTO :acctData;

        if SQLCODE = 0;
            // 计算并计提利息
            if CalcAndAccrueInterest(acctData : inCalcDt);
                processedCnt = processedCnt + 1;
            else;
                errorCnt = errorCnt + 1;
            endif;
        endif;
    enddo;

    exec SQL CLOSE ACCT_CUR;
```

```
    // 返回处理结果
    outRtnMsg = '处理完成: 成功=' + %char(processedCnt) +
                ', 失败=' + %char(errorCnt);

    if errorCnt > 0;
        outRtnCode = '01';
    endif;

end-proc;

// ====================================================================
// 计算并计提利息
// ====================================================================
dcl-proc CalcAndAccrueInterest;
    dcl-pi *n ind;
        acctData    likeds(acctData) const;
        calcDt      packed(8) const;
    end-pi;

    dcl-ds result likeds(InterestResult);
    dcl-s lastIntDt packed(8);
    dcl-s accruedInt packed(19:2);

    // 确定上次计息日
    exec SQL
        SELECT LAST_INT_DT, ACCRUED_INT
        INTO :lastIntDt, :accruedInt
        FROM ACCTMAST
        WHERE ACCT_NO = :acctData.acctNo;

    if lastIntDt = 0;
        // 新账户，从开户日起息
        exec SQL
            SELECT OPEN_DT INTO :lastIntDt
            FROM ACCTMAST
            WHERE ACCT_NO = :acctData.acctNo;
    endif;

    // 计算利息
    result = CalculateInterest(
        acctData.acctNo :
        lastIntDt :
        calcDt :
        acctData.acctBal :
        acctData.intRate :
```

```
            acctData.taxRate :
            acctData.taxExemptFlag :
            acctData.prodType
        );

        // 更新账户计提利息
        exec SQL
            UPDATE ACCTMAST
            SET ACCRUED_INT = ACCRUED_INT + :result.grossInterest,
                LAST_INT_DT = :calcDt,
                INT_BASE_CURR = 0,           // 重置当期积数
                INT_BASE_TOTAL = INT_BASE_TOTAL + INT_BASE_CURR,
                UPDATE_TS = CURRENT_TIMESTAMP,
                UPDATE_USER = USER
            WHERE ACCT_NO = :acctData.acctNo;

        // 写入利息计提明细
        exec SQL
            INSERT INTO INTACCRLOG (
                ACCR_DT,
                ACCT_NO,
                FROM_DT,
                TO_DT,
                ACCR_DAYS,
                PRINCIPAL,
                INT_RATE,
                GROSS_INT,
                TAX_RATE,
                TAX_AMT,
                NET_INT,
                CREATE_TS
            ) VALUES (
                :calcDt,
                :acctData.acctNo,
                :result.intStartDt,
                :result.intEndDt,
                :result.intDays,
                :result.principal,
                :result.intRate,
                :result.grossInterest,
                :result.taxRate,
                :result.taxAmount,
                :result.netInterest,
                CURRENT_TIMESTAMP
            );
```

```
    return (SQLCODE = 0);
end-proc;

// ====================================================================
// 利息计算核心算法
// ====================================================================
dcl-proc CalculateInterest export;
    dcl-pi *n likeds(InterestResult);
        inAcctNo        char(32) const;
        inStartDt       packed(8) const;
        inEndDt         packed(8) const;
        inPrincipal     packed(19:2) const;
        inIntRate       packed(9:6) const;
        inTaxRate       packed(5:2) const;
        inTaxExempt     char(1) const;
        inProdType      char(2) const;
    end-pi;

    dcl-ds result likeds(InterestResult);
    dcl-s startDate date;
    dcl-s endDate date;
    dcl-s days packed(5);
    dcl-s yearDays packed(3);
    dcl-s grossInt packed(19:2);
    dcl-s netInt packed(19:2);
    dcl-s taxAmt packed(19:2);

    // 初始化结果
    result.acctNo = inAcctNo;
    result.intStartDt = inStartDt;
    result.intEndDt = inEndDt;
    result.principal = inPrincipal;
    result.intRate = inIntRate;
    result.taxRate = inTaxRate;

    // 转换日期格式
    startDate = %date(inStartDt : *iso);
    endDate = %date(inEndDt : *iso);

    // 计算天数
    days = %diff(endDate : startDate : *days);
    result.intDays = days;

    // 确定计息方法
    select;
        when inProdType = '01';  // 活期
```

```
            result.calcMethod = METHOD_ACTUAL_360;
            yearDays = 360;
        when inProdType = '02';  // 定期
            result.calcMethod = METHOD_ACTUAL_360;
            yearDays = 360;
        when inProdType = '03';  // 通知存款
            result.calcMethod = METHOD_ACTUAL_360;
            yearDays = 360;
        other;
            result.calcMethod = METHOD_ACTUAL_365;
            yearDays = 365;
    endsl;

    // 利息计算公式: 本金 × 利率 × 天数 / 年基数
    // 注意: 利率是年利率，需要转换为日利率
    grossInt = inPrincipal * inIntRate * days / 100 / yearDays;
    result.grossInterest = %dech(grossInt : 19 : 2);

    // 计算利息税
    if inTaxExempt = '1';
        taxAmt = 0;
        result.taxRate = 0;
    else;
        taxAmt = result.grossInterest * inTaxRate / 100;
    endif;
    result.taxAmount = %dech(taxAmt : 19 : 2);

    // 税后利息
    netInt = result.grossInterest - result.taxAmount;
    result.netInterest = %dech(netInt : 19 : 2);

    return result;
end-proc;

// ============================================================
// 定期结息入账
// ============================================================
dcl-proc PostInterest export;
    dcl-pi *n ind;
        inAcctNo        char(32) const;
        inPostDt        packed(8) const;
        outRtnCode      char(2);
        outRtnMsg       char(200);
    end-pi;

    dcl-s accruedInt packed(19:2);
```

```
    dcl-s taxAmt packed(19:2);
    dcl-s netInt packed(19:2);
    dcl-s transSeq char(20);

    outRtnCode = '00';
    outRtnMsg = '';

    // 获取待入账利息
    exec SQL
        SELECT ACCRUED_INT, TAX_RATE
        INTO :accruedInt, :taxRate
        FROM ACCTMAST
        WHERE ACCT_NO = :inAcctNo
          AND ACCT_STAT = '0';

    if SQLCODE <> 0 or accruedInt <= 0;
        outRtnCode = '91';
        outRtnMsg = '无待入账利息';
        return *off;
    endif;

    // 计算利息税
    taxAmt = %dech(accruedInt * taxRate / 100 : 19 : 2);
    netInt = accruedInt - taxAmt;

    // 生成交易流水号
    transSeq = GenerateTransSeq();

    // 更新账户 - 入账税后利息
    exec SQL
        UPDATE ACCTMAST
        SET ACCT_BAL = ACCT_BAL + :netInt,
            AVAIL_BAL = AVAIL_BAL + :netInt,
            ACCRUED_INT = 0,
            NEXT_INT_DT = CASE PROD_TYPE
                WHEN '01' THEN :inPostDt + 1 QUARTER  -- 活期季度结息
                WHEN '02' THEN MATURITY_DT             -- 定期到期结息
                ELSE :inPostDt + 1 YEAR
            END,
            UPDATE_TS = CURRENT_TIMESTAMP
        WHERE ACCT_NO = :inAcctNo;

    // 写入利息入账明细
    exec SQL
        INSERT INTO INTPOSTLOG (
            POST_DT,
```

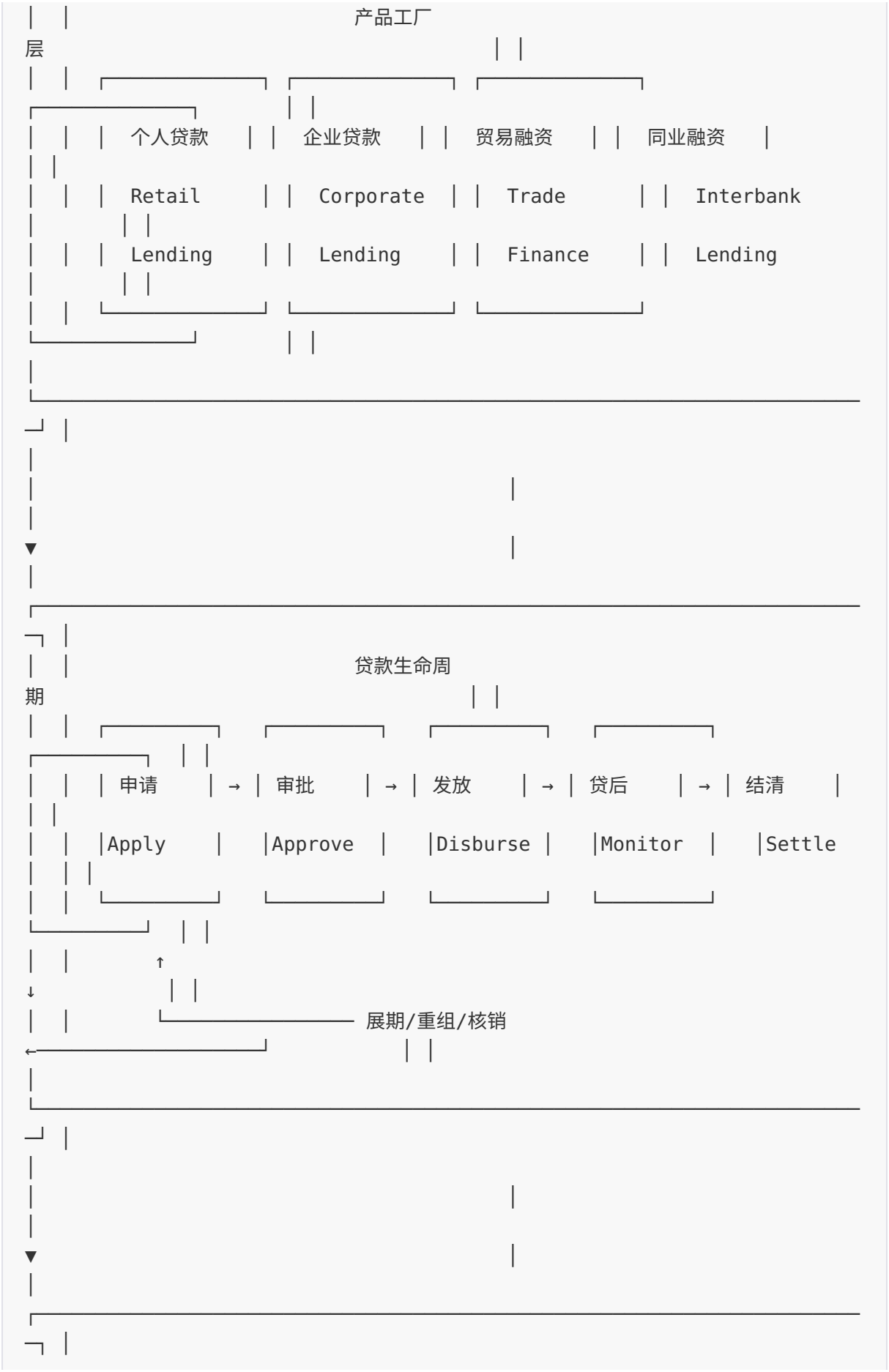
```
            ACCT_NO,
            GROSS_INT,
            TAX_RATE,
            TAX_AMT,
            NET_INT,
            TRANS_SEQ,
            CREATE_TS
        ) VALUES (
            :inPostDt,
            :inAcctNo,
            :accruedInt,
            :taxRate,
            :taxAmt,
            :netInt,
            :transSeq,
            CURRENT_TIMESTAMP
        );

    // 生成会计分录
    // 借: 利息支出科目
    // 贷: 客户存款科目

    outRtnMsg = '利息入账成功: 金额=' + %char(netInt);
    return *on;
end-proc;
```

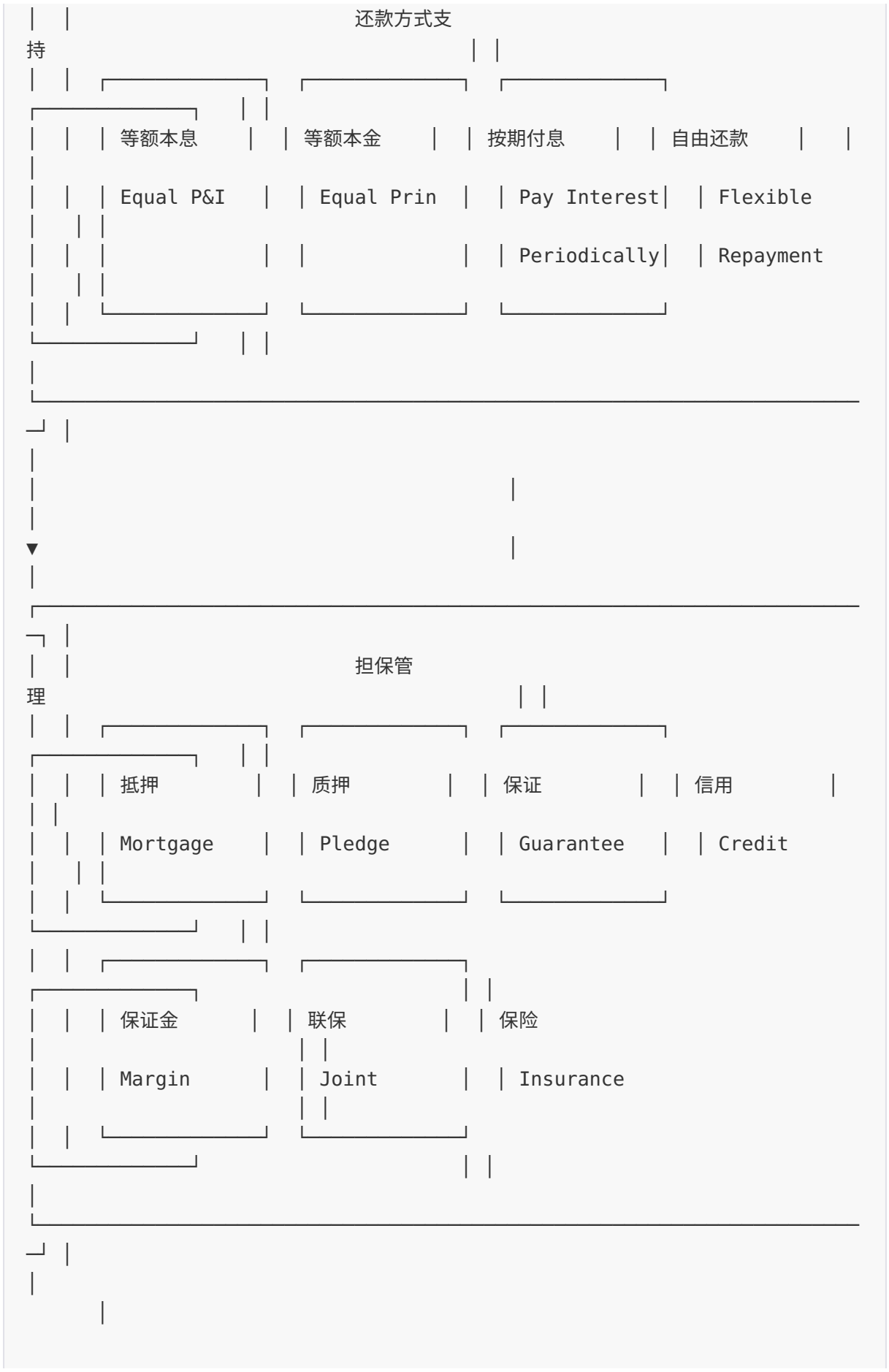
2.4 贷款系统

2.4.1 贷款产品架构

贷款系统是银行的核心资产模块，涵盖个人贷款、企业贷款、贸易融资等多种产品。

```
┌──────────────────────────────────────────────────────────────────────────────
──────┐
│                                 贷款系统产品架
构                                          │
├──────────────────────────────────────────────────────────────────────────────
──────┤
│
      │
│
┌──────────────────────────────────────────────────────────────────────────────
┐ │
```


产品工厂
个人贷款
企业贷款
贸易融资
同业融资
Retail Lending
Corporate Lending
Trade Finance
Interbank Lending
贷款生命周期
申请
审批
发放
贷后
结清
Apply
Approve
Disburse
Monitor
Settle
展期/重组/核销


还款方式支
持
等额本息
Equal P&I
等额本金
Equal Prin
按期付息
Pay Interest
Periodically
自由还款
Flexible
Repayment
担保管
理
抵押
Mortgage
质押
Pledge
保证
Guarantee
信用
Credit
保证金
Margin
联保
Joint
保险
Insurance

```
└──────────────────────────────────────────────────────────────────────────────
──────┘
```

2.4.2 贷款主表设计

```
-- ================================================================
-- 贷款主表 - LOANMAST
-- ================================================================

CREATE TABLE LOANMAST (
    -- 主键
    LOAN_NO             CHAR(20)        NOT NULL,

    -- 客户信息
    CUST_ID             CHAR(20)        NOT NULL,
    CUST_NAME           VARCHAR(100),

    -- 贷款基本信息
    PROD_CD             CHAR(10)        NOT NULL,
    LOAN_TYPE           CHAR(2)         NOT NULL,  -- 01-个人消费 02-个人
经营...
    LOAN_PURPOSE        VARCHAR(200),             -- 贷款用途

    -- 合同金额与期限
    CONTRACT_AMT        DECIMAL(19, 2)  NOT NULL,
    CONTRACT_DT         DECIMAL(8, 0)   NOT NULL,
    MATURITY_DT         DECIMAL(8, 0)   NOT NULL,
    LOAN_TERM_MONTHS    DECIMAL(3, 0),
    LOAN_TERM_DAYS      DECIMAL(5, 0),

    -- 利率信息
    INT_RATE_TYPE       CHAR(1)         DEFAULT '0',  -- 0-固定 1-浮动
    INT_RATE            DECIMAL(9, 6)   NOT NULL,
    INT_RATE_SPREAD     DECIMAL(9, 6)   DEFAULT 0,
    BENCHMARK_RATE      DECIMAL(9, 6),                -- 基准利率(LPR)
    ODR_INT_RATE        DECIMAL(9, 6)   DEFAULT 0,    -- 逾期利率
    ODR_RATE_SPREAD     DECIMAL(5, 2)   DEFAULT 50,   -- 逾期上浮比例%

    -- 还款方式
    REPAY_METHOD        CHAR(2)         NOT NULL,  -- 01-等额本息 02-等额
本金...
    REPAY_FREQ          CHAR(2)         DEFAULT '01', -- 01-月 02-季 03-
半年 04-年
```

```
    REPAY_DAY           DECIMAL(2, 0)   DEFAULT 1,   -- 还款日

    -- 还款账户
    REPAY_ACCT_NO       CHAR(32)        NOT NULL,
    INCOME_ACCT_NO      CHAR(32),                    -- 收入归集账户

    -- 贷款余额
    LOAN_BAL            DECIMAL(19, 2)  DEFAULT 0,  -- 本金余额
    ODR_PRIN_BAL        DECIMAL(19, 2)  DEFAULT 0,  -- 逾期本金
    ODR_INT_BAL         DECIMAL(19, 2)  DEFAULT 0,  -- 应收逾期利息
    ACCRUED_INT         DECIMAL(19, 2)  DEFAULT 0,  -- 已计提利息

    -- 贷款状态
    LOAN_STAT           CHAR(2)         DEFAULT '01', -- 01-申请 02-审批
03-发放...
    STAT_CHG_DT         DECIMAL(8, 0),

    -- 五级分类
    CLASS_LEVEL         CHAR(1)         DEFAULT '1',  -- 1-正常 2-关注
3-次级 4-可疑 5-损失
    CLASS_CHG_DT        DECIMAL(8, 0),
    CLASS_RSN           VARCHAR(200),

    -- 担保信息
    GUAR_TYPE           CHAR(2)         NOT NULL,  -- 01-信用 02-保证
03-抵押 04-质押...
    COLLAT_VALUE        DECIMAL(19, 2),             -- 担保物评估价值

    -- 发放信息
    FIRST_DISB_DT       DECIMAL(8, 0),
    LAST_DISB_DT        DECIMAL(8, 0),
    DISB_AMT            DECIMAL(19, 2)  DEFAULT 0,

    -- 结清信息
    SETTLE_DT           DECIMAL(8, 0),
    SETTLE_TYPE         CHAR(1),                   -- 1-正常结清 2-提前结
清 3-核销...

    -- 客户经理
    CUST_MGR_ID         CHAR(10),
    BRANCH_ID           CHAR(10)        NOT NULL,

    -- 渠道信息
    APPLY_CHNL          CHAR(2)         DEFAULT '01',
    APPLY_DT            DECIMAL(8, 0)   NOT NULL,
```

```
    -- 扩展属性(JSON格式)
    EXT_ATTR            CLOB(10K),

    -- 审计字段
    CREATE_TS           TIMESTAMP       DEFAULT CURRENT_TIMESTAMP,
    CREATE_USER         VARCHAR(20)     DEFAULT USER,
    UPDATE_TS           TIMESTAMP       DEFAULT CURRENT_TIMESTAMP,
    UPDATE_USER         VARCHAR(20)     DEFAULT USER,
    UPDATE_PRG          VARCHAR(20),

    -- 约束
    CONSTRAINT PK_LOANMAST PRIMARY KEY (LOAN_NO),
    CONSTRAINT FK_LOANMAST_CUST FOREIGN KEY (CUST_ID)
        REFERENCES CUSTMAST(CUST_ID)

) RCDFMT LOANMASTR;

-- 还款计划表
CREATE TABLE LOANSCHD (
    LOAN_NO             CHAR(20)        NOT NULL,
    SCHD_SEQ            DECIMAL(5, 0)   NOT NULL,
    PERIOD_NO           DECIMAL(3, 0)   NOT NULL,

    REPAY_DT            DECIMAL(8, 0)   NOT NULL,
    PRIN_AMT            DECIMAL(19, 2)  NOT NULL,  -- 应还本金
    INT_AMT             DECIMAL(19, 2)  NOT NULL,  -- 应还利息
    TOTAL_AMT           DECIMAL(19, 2)  NOT NULL,  -- 应还总额

    PRIN_PAID           DECIMAL(19, 2)  DEFAULT 0, -- 已还本金
    INT_PAID            DECIMAL(19, 2)  DEFAULT 0, -- 已还利息
    ODR_INT_PAID        DECIMAL(19, 2)  DEFAULT 0, -- 已还逾期利息

    PAID_DT             DECIMAL(8, 0),              -- 实际还款日
    PAID_FLAG           CHAR(1)         DEFAULT '0', -- 0-未还 1-已还 2-
部分还

    PRIN_BAL            DECIMAL(19, 2),             -- 还款后本金余额

    CREATE_TS           TIMESTAMP       DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT PK_LOANSCHD PRIMARY KEY (LOAN_NO, SCHD_SEQ),
    CONSTRAINT FK_LOANSCHD_LOAN FOREIGN KEY (LOAN_NO)
        REFERENCES LOANMAST(LOAN_NO) ON DELETE CASCADE
);
```

2.4.3 还款计划生成算法

RPG IV - 等额本息还款计划生成:

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('LOAN');

// ====================================================================
// 程序: GENSCHD
// 描述: 贷款还款计划生成
// 支持: 等额本息、等额本金、按期付息到期还本
// ====================================================================

// 输入参数结构
dcl-ds SchdParms qualified template;
    loanNo          char(20);
    loanAmt         packed(19:2);
    annualRate      packed(9:6);
    totalPeriods    packed(3:0);
    repayMethod     char(2);        // 01-等额本息 02-等额本金 03-按期付息
    repayFreq       char(2);        // 01-月 02-季 03-半年 04-年
    firstRepayDt    packed(8);
    intCalcMethod   char(1);        // 0-360天 1-365天
end-ds;

// 还款计划明细结构
dcl-ds SchdDetail qualified template;
    periodNo        packed(3:0);
    repayDt         packed(8);
    beginBal        packed(19:2);
    prinAmt         packed(19:2);
    intAmt          packed(19:2);
    totalAmt        packed(19:2);
    endBal          packed(19:2);
end-ds;

// ====================================================================
// 生成还款计划主入口
// ====================================================================
dcl-proc GenerateRepaymentSchedule export;
    dcl-pi *n packed(5);  // 返回生成的期数
        parms   likeds(SchdParms) const;
        outRtnCode  char(2);
        outRtnMsg   char(200);
```

```
    end-pi;

    dcl-ds schedule likeds(SchdDetail) dim(360);
    dcl-s periodCnt packed(5);

    outRtnCode = '00';
    outRtnMsg = '';

    // 参数校验
    if parms.loanAmt <= 0;
        outRtnCode = '01';
        outRtnMsg = '贷款金额必须大于0';
        return 0;
    endif;

    if parms.annualRate < 0;
        outRtnCode = '02';
        outRtnMsg = '利率不能为负数';
        return 0;
    endif;

    if parms.totalPeriods <= 0 or parms.totalPeriods > 360;
        outRtnCode = '03';
        outRtnMsg = '还款期数必须在1-360之间';
        return 0;
    endif;

    // 根据还款方式调用不同算法
    select;
        when parms.repayMethod = '01';  // 等额本息
            periodCnt = CalcEqualPandI(parms : schedule);
        when parms.repayMethod = '02';  // 等额本金
            periodCnt = CalcEqualPrincipal(parms : schedule);
        when parms.repayMethod = '03';  // 按期付息到期还本
            periodCnt = CalcPeriodicInterest(parms : schedule);
        other;
            outRtnCode = '04';
            outRtnMsg = '不支持的还款方式';
            return 0;
    endsl;

    // 保存还款计划到数据库
    SaveScheduleToDB(parms.loanNo : schedule : periodCnt);

    return periodCnt;
end-proc;
```

```
// ================================================================
// 等额本息计算
// 公式: 月供 = 本金 × 月利率 × (1+月利率)^期数 / ((1+月利率)^期数 - 1)
// ================================================================
dcl-proc CalcEqualPandI;
    dcl-pi *n packed(5);
        parms       likeds(SchdParms) const;
        outSchedule likeds(SchdDetail) dim(360);
    end-pi;

    dcl-s monthlyRate packed(9:8);
    dcl-s periodRate packed(9:8);
    dcl-s payment packed(19:2);
    dcl-s remainingBal packed(19:2);
    dcl-s periodInt packed(19:2);
    dcl-s periodPrin packed(19:2);
    dcl-s i packed(5);
    dcl-s repayDt date;
    dcl-s daysInPeriod packed(3);

    // 计算周期利率
    periodRate = GetPeriodRate(parms.annualRate : parms.repayFreq);

    // 计算月供/周期供款
    if periodRate = 0;
        payment = parms.loanAmt / parms.totalPeriods;
    else;
        payment = parms.loanAmt * periodRate *
                  (1 + parms.totalPeriods) /
                  ((1 + parms.totalPeriods) - 1);
    endif;

    remainingBal = parms.loanAmt;
    repayDt = %date(parms.firstRepayDt : *iso);

    for i = 1 to parms.totalPeriods;
        // 计算本期利息
        periodInt = %dech(remainingBal * periodRate : 19 : 2);

        // 计算本期本金
        if i = parms.totalPeriods;
            // 最后一期: 本金 = 剩余余额
            periodPrin = remainingBal;
        else;
            periodPrin = payment - periodInt;
```

```
        endif;

        // 更新剩余本金
        remainingBal = remainingBal - periodPrin;

        // 填充计划明细
        outSchedule(i).periodNo = i;
        outSchedule(i).repayDt = %int(%char(repayDt : *iso0));
        outSchedule(i).beginBal = remainingBal + periodPrin;
        outSchedule(i).prinAmt = periodPrin;
        outSchedule(i).intAmt = periodInt;
        outSchedule(i).totalAmt = periodPrin + periodInt;
        outSchedule(i).endBal = remainingBal;

        // 计算下期还款日
        repayDt = AddPeriod(repayDt : parms.repayFreq);
    endfor;

    return parms.totalPeriods;
end-proc;

// ============================================================
// 等额本金计算
// 公式: 每期本金 = 总本金 / 期数
//       每期利息 = 剩余本金 × 周期利率
// ============================================================
dcl-proc CalcEqualPrincipal;
    dcl-pi *n packed(5);
        parms       likeds(SchdParms) const;
        outSchedule likeds(SchdDetail) dim(360);
    end-pi;

    dcl-s periodRate packed(9:8);
    dcl-s fixedPrin packed(19:2);
    dcl-s remainingBal packed(19:2);
    dcl-s periodInt packed(19:2);
    dcl-s i packed(5);
    dcl-s repayDt date;

    // 计算周期利率和固定本金
    periodRate = GetPeriodRate(parms.annualRate : parms.repayFreq);
    fixedPrin = parms.loanAmt / parms.totalPeriods;

    remainingBal = parms.loanAmt;
    repayDt = %date(parms.firstRepayDt : *iso);
```

```
    for i = 1 to parms.totalPeriods;
        // 计算本期利息
        periodInt = %dech(remainingBal * periodRate : 19 : 2);

        // 最后一期调整本金
        if i = parms.totalPeriods;
            fixedPrin = remainingBal;
        endif;

        // 更新剩余本金
        remainingBal = remainingBal - fixedPrin;

        // 填充计划明细
        outSchedule(i).periodNo = i;
        outSchedule(i).repayDt = %int(%char(repayDt : *iso0));
        outSchedule(i).beginBal = remainingBal + fixedPrin;
        outSchedule(i).prinAmt = fixedPrin;
        outSchedule(i).intAmt = periodInt;
        outSchedule(i).totalAmt = fixedPrin + periodInt;
        outSchedule(i).endBal = remainingBal;

        // 计算下期还款日
        repayDt = AddPeriod(repayDt : parms.repayFreq);
    endfor;

    return parms.totalPeriods;
end-proc;

// ================================================================
// 按期付息到期还本计算
// ================================================================
dcl-proc CalcPeriodicInterest;
    dcl-pi *n packed(5);
        parms       likeds(SchdParms) const;
        outSchedule likeds(SchdDetail) dim(360);
    end-pi;

    dcl-s periodRate packed(9:8);
    dcl-s remainingBal packed(19:2);
    dcl-s periodInt packed(19:2);
    dcl-s i packed(5);
    dcl-s repayDt date;

    periodRate = GetPeriodRate(parms.annualRate : parms.repayFreq);
    remainingBal = parms.loanAmt;
    repayDt = %date(parms.firstRepayDt : *iso);
```

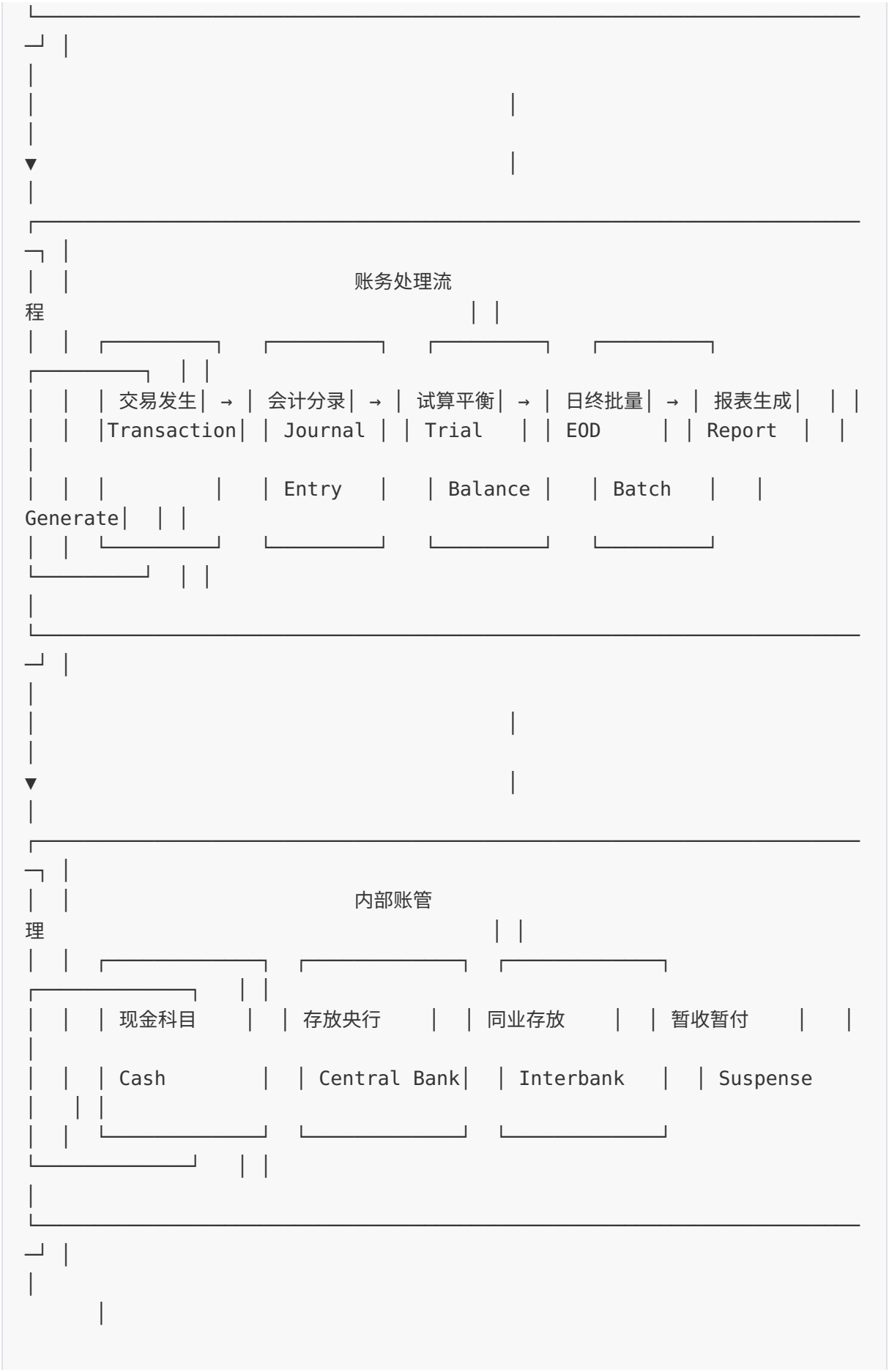
```
    for i = 1 to parms.totalPeriods;
        // 计算本期利息
        periodInt = %dech(remainingBal * periodRate : 19 : 2);

        // 填充计划明细
        outSchedule(i).periodNo = i;
        outSchedule(i).repayDt = %int(%char(repayDt : *iso0));
        outSchedule(i).beginBal = remainingBal;

        if i = parms.totalPeriods;
            // 最后一期还本付息
            outSchedule(i).prinAmt = remainingBal;
            outSchedule(i).endBal = 0;
            remainingBal = 0;
        else;
            // 只付息
            outSchedule(i).prinAmt = 0;
            outSchedule(i).endBal = remainingBal;
        endif;

        outSchedule(i).intAmt = periodInt;
        outSchedule(i).totalAmt = outSchedule(i).prinAmt + periodInt;

        // 计算下期还款日
        repayDt = AddPeriod(repayDt : parms.repayFreq);
    endfor;

    return parms.totalPeriods;
end-proc;

// ================================================================
// 计算周期利率
// ================================================================
dcl-proc GetPeriodRate;
    dcl-pi *n packed(9:8);
        annualRate  packed(9:6) const;
        repayFreq   char(2) const;
    end-pi;

    select;
        when repayFreq = '01';  // 月
            return annualRate / 12 / 100;
        when repayFreq = '02';  // 季
            return annualRate / 4 / 100;
        when repayFreq = '03';  // 半年
```

```
            return annualRate / 2 / 100;
        when repayFreq = '04';  // 年
            return annualRate / 100;
        other;
            return annualRate / 12 / 100;  // 默认按月
    endsl;
end-proc;

// ================================================================
// 增加周期
// ================================================================
dcl-proc AddPeriod;
    dcl-pi *n date;
        inDate      date const;
        repayFreq   char(2) const;
    end-pi;

    select;
        when repayFreq = '01';  // 月
            return inDate + %months(1);
        when repayFreq = '02';  // 季
            return inDate + %months(3);
        when repayFreq = '03';  // 半年
            return inDate + %months(6);
        when repayFreq = '04';  // 年
            return inDate + %years(1);
        other;
            return inDate + %months(1);
    endsl;
end-proc;

// ================================================================
// 保存还款计划到数据库
// ================================================================
dcl-proc SaveScheduleToDB;
    dcl-pi *n;
        loanNo      char(20) const;
        schedule    likeds(SchdDetail) dim(360) const;
        periodCnt   packed(5) const;
    end-pi;

    dcl-s i packed(5);

    // 先删除原有计划
    exec SQL
        DELETE FROM LOANSCHD WHERE LOAN_NO = :loanNo;
```

```
    // 插入新计划
    for i = 1 to periodCnt;
        exec SQL
            INSERT INTO LOANSCHD (
                LOAN_NO, SCHD_SEQ, PERIOD_NO, REPAY_DT,
                PRIN_AMT, INT_AMT, TOTAL_AMT, PRIN_BAL
            ) VALUES (
                :loanNo, :i, :schedule(i).periodNo, :schedule(i).repayD
t,
                :schedule(i).prinAmt, :schedule(i).intAmt,
                :schedule(i).totalAmt, :schedule(i).endBal
            );
    endfor;

end-proc;
```

2.5 总账系统

2.5.1 总账架构设计


账务处理流程
交易发生 Transaction
会计分录 Journal
试算平衡 Trial
日终批量 EOD
报表生成 Report
Generate
Entry
Balance
Batch
内部账管理
现金科目 Cash
存放央行 Central Bank
同业存放 Interbank
暂收暂付 Suspense

```
└──────────────────────────────────────────────────────────────────────────────
──────┘
```

2.5.2 会计分录处理

SQL RPG - 会计分录生成:

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('GL');

// ====================================================================
// 程序: GLENTRY
// 描述: 会计分录处理核心
// 功能: 生成、校验、登记会计分录
// ====================================================================

// 会计分录结构
dcl-ds GLEntry qualified template;
    entryId         char(32);
    entrySeq        packed(5);
    acctDt          packed(8);
    glAcctNo        char(20);
    drCrFlag        char(1);        // D-借 C-贷
    amt             packed(19:2);
    ccyCd           char(3);
    transRefNo      char(32);
    transType       char(4);
    summary         varchar(200);
    branchId        char(10);
    entryTs         timestamp;
end-ds;

// 记账请求结构
dcl-ds PostingRequest qualified template;
    batchId         char(20);
    acctDt          packed(8);
    entryCnt        packed(5);
    entries         likeds(GLEntry) dim(100);
    transRefNo      char(32);
    transType       char(4);
    branchId        char(10);
end-ds;
```

```
// ============================================================
// 登记会计分录主入口
// ============================================================
dcl-proc PostGLEntries export;
    dcl-pi *n likeds(PostingResponse);
        request likeds(PostingRequest) const;
    end-pi;

    dcl-ds response likeds(PostingResponse);
    dcl-s i packed(5);
    dcl-s drTotal packed(19:2);
    dcl-s crTotal packed(19:2);

    response.rtnCode = '00';
    response.rtnMsg = '';
    response.postedEntries = 0;

    // 1. 校验分录平衡
    for i = 1 to request.entryCnt;
        if request.entries(i).drCrFlag = 'D';
            drTotal = drTotal + request.entries(i).amt;
        else;
            crTotal = crTotal + request.entries(i).amt;
        endif;
    endfor;

    if drTotal <> crTotal;
        response.rtnCode = '01';
        response.rtnMsg = '借贷不平衡: 借=' + %char(drTotal) +
                          ' 贷=' + %char(crTotal);
        return response;
    endif;

    // 2. 校验科目有效性
    for i = 1 to request.entryCnt;
        if not ValidateGLAccount(request.entries(i).glAcctNo :
                                 request.entries(i).branchId);
            response.rtnCode = '02';
            response.rtnMsg = '无效会计科目: ' +
request.entries(i).glAcctNo;
            return response;
        endif;
    endfor;

    // 3. 生成分录ID
    dcl-s batchId char(20);
```

```
    batchId = GenerateGLEntryId();

    // 4. 登记分录
    for i = 1 to request.entryCnt;
        if WriteGLEntry(request.entries(i) : batchId : i);
            response.postedEntries = response.postedEntries + 1;
        else;
            // 回滚已登记的分录
            RollbackGLEntries(batchId);
            response.rtnCode = '10';
            response.rtnMsg = '登记分录失败，已回滚';
            return response;
        endif;
    endfor;

    response.batchId = batchId;
    response.rtnMsg = '成功登记 ' + %char(response.postedEntries) + ' 笔
分录';

    return response;
end-proc;

// ====================================================================
// 校验会计科目
// ====================================================================
dcl-proc ValidateGLAccount;
    dcl-pi *n ind;
        glAcctNo    char(20) const;
        branchId    char(10) const;
    end-pi;

    dcl-s cnt packed(5);

    exec SQL
        SELECT COUNT(*) INTO :cnt
        FROM GLMAST
        WHERE GL_ACCT_NO = :glAcctNo
          AND BRANCH_ID = :branchId
          AND ACCT_STAT = '0';   // 正常状态

    return (cnt > 0);
end-proc;

// ====================================================================
// 写入会计分录
// ====================================================================
```

```
dcl-proc WriteGLEntry;
    dcl-pi *n ind;
        entry       likeds(GLEntry) const;
        batchId     char(20) const;
        entrySeq    packed(5) const;
    end-pi;

    dcl-s entryId char(32);
    entryId = batchId + %editc(entrySeq:'X');

    exec SQL
        INSERT INTO GLTRANS (
            ENTRY_ID,
            BATCH_ID,
            ENTRY_SEQ,
            ACCT_DT,
            GL_ACCT_NO,
            DR_CR_FLAG,
            AMT,
            CCY_CD,
            TRANS_REF_NO,
            TRANS_TYPE,
            SUMMARY,
            BRANCH_ID,
            POST_TS,
            CREATE_USER
        ) VALUES (
            :entryId,
            :batchId,
            :entrySeq,
            :entry.acctDt,
            :entry.glAcctNo,
            :entry.drCrFlag,
            :entry.amt,
            :entry.ccyCd,
            :entry.transRefNo,
            :entry.transType,
            :entry.summary,
            :entry.branchId,
            CURRENT_TIMESTAMP,
            USER
        );

    return (SQLCODE = 0);
end-proc;
```

```
// ============================================================
// 更新科目余额
// ============================================================
dcl-proc UpdateGLBalance export;
    dcl-pi *n ind;
        acctDt      packed(8) const;
        glAcctNo    char(20) const;
        drCrFlag    char(1) const;
        amt         packed(19:2) const;
    end-pi;

    dcl-s currentBal packed(19:2);
    dcl-s newBal packed(19:2);

    // 获取当前余额
    exec SQL
        SELECT END_BAL INTO :currentBal
        FROM GLBAL
        WHERE ACCT_DT = :acctDt
          AND GL_ACCT_NO = :glAcctNo;

    if SQLCODE <> 0;
        // 新增余额记录
        currentBal = 0;
        exec SQL
            INSERT INTO GLBAL (ACCT_DT, GL_ACCT_NO, BEGIN_BAL, END_BAL)
            VALUES (:acctDt, :glAcctNo, 0, 0);
    endif;

    // 计算新余额
    if drCrFlag = 'D';
        newBal = currentBal + amt;
    else;
        newBal = currentBal - amt;
    endif;

    // 更新余额
    exec SQL
        UPDATE GLBAL
        SET END_BAL = :newBal,
            DR_AMT = CASE WHEN :drCrFlag = 'D' THEN DR_AMT + :amt ELSE
DR_AMT END,
            CR_AMT = CASE WHEN :drCrFlag = 'C' THEN CR_AMT + :amt ELSE
CR_AMT END,
            UPDATE_TS = CURRENT_TIMESTAMP
        WHERE ACCT_DT = :acctDt
```

```
            AND GL_ACCT_NO = :glAcctNo;

    return (SQLCODE = 0);
end-proc;
```

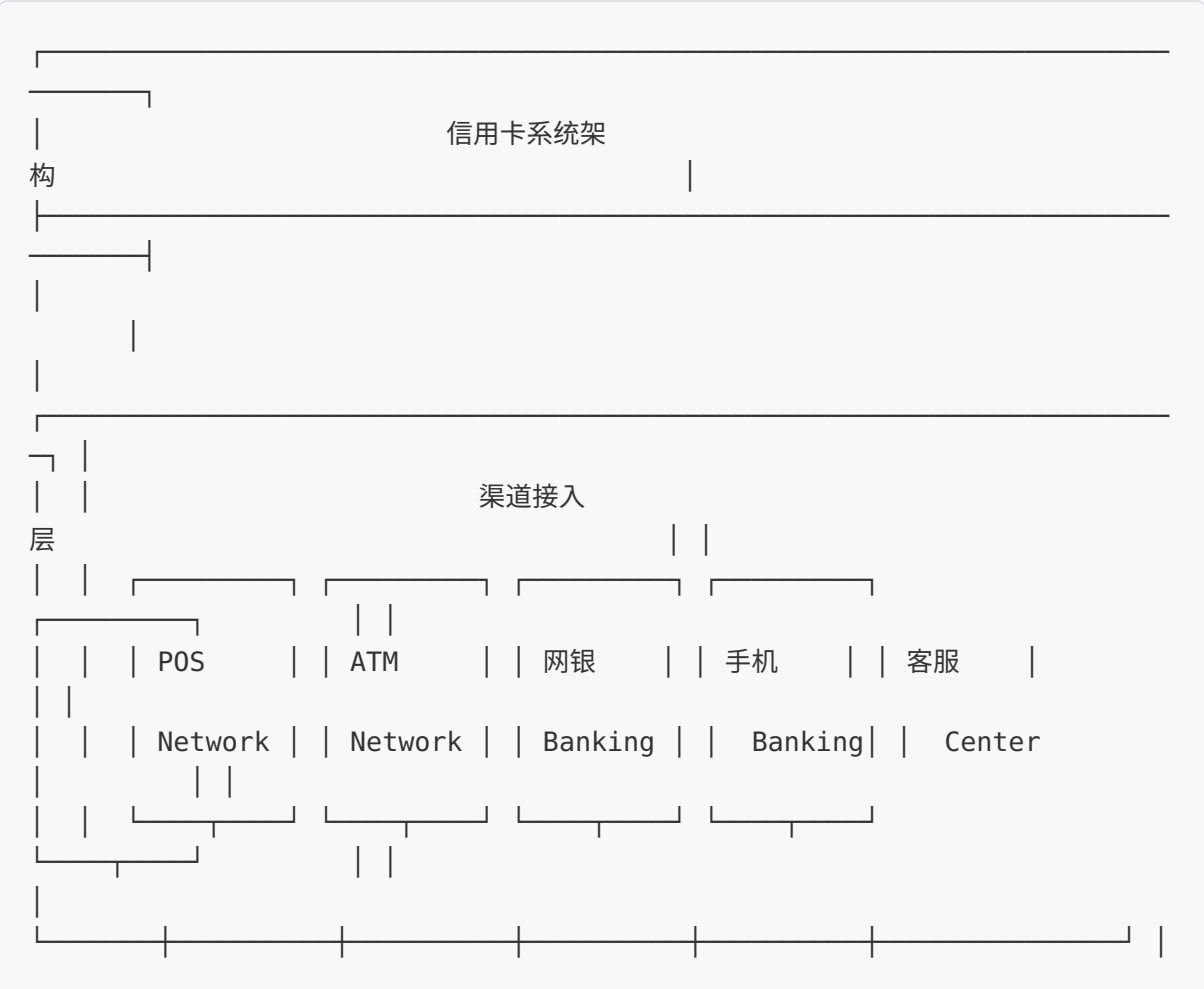
由于文档内容非常庞大（需要超过200,000字符），我将继续追加内容。让我继续写入文档的后续部分。

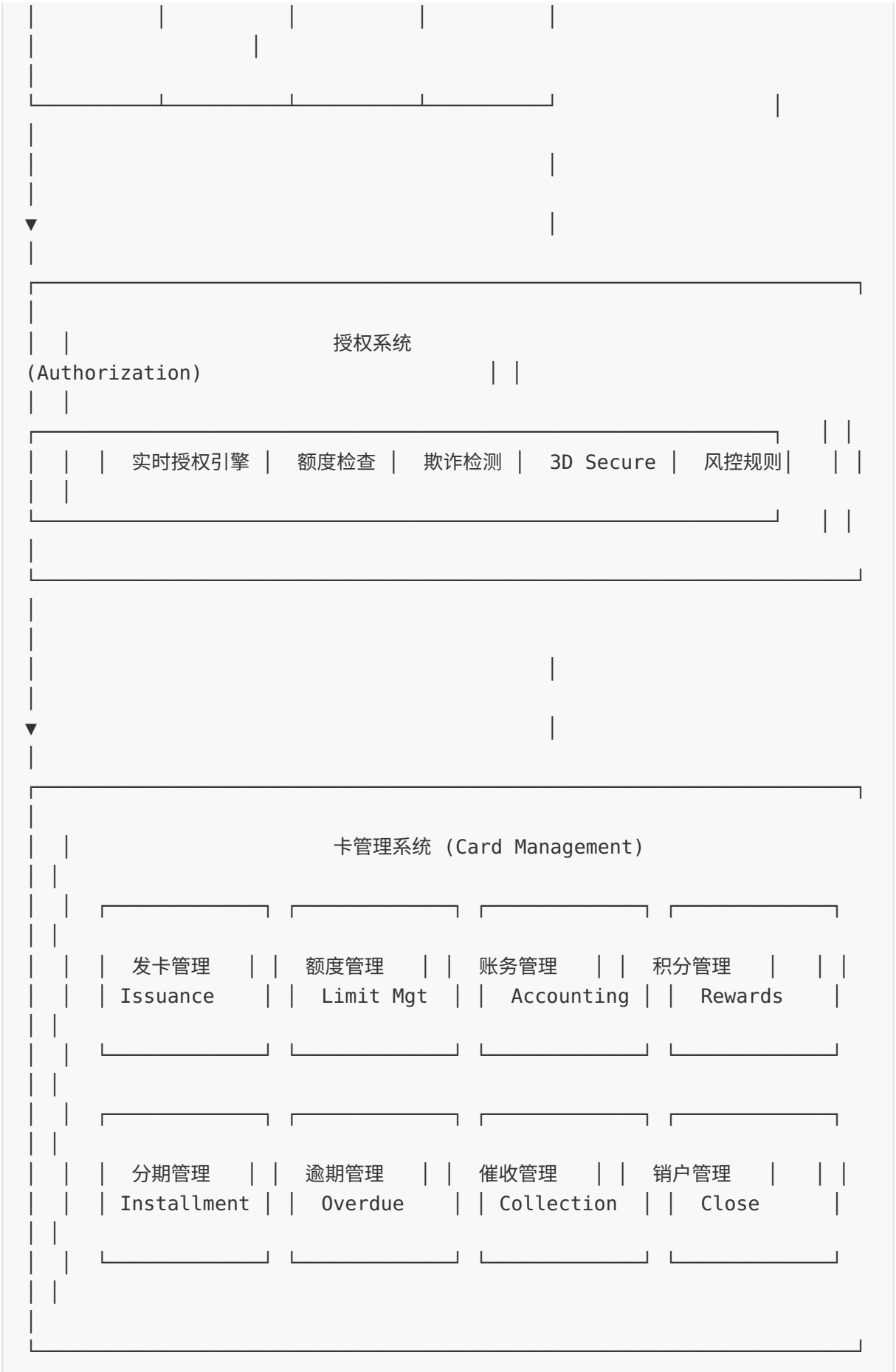
第三章 卡系统与支付

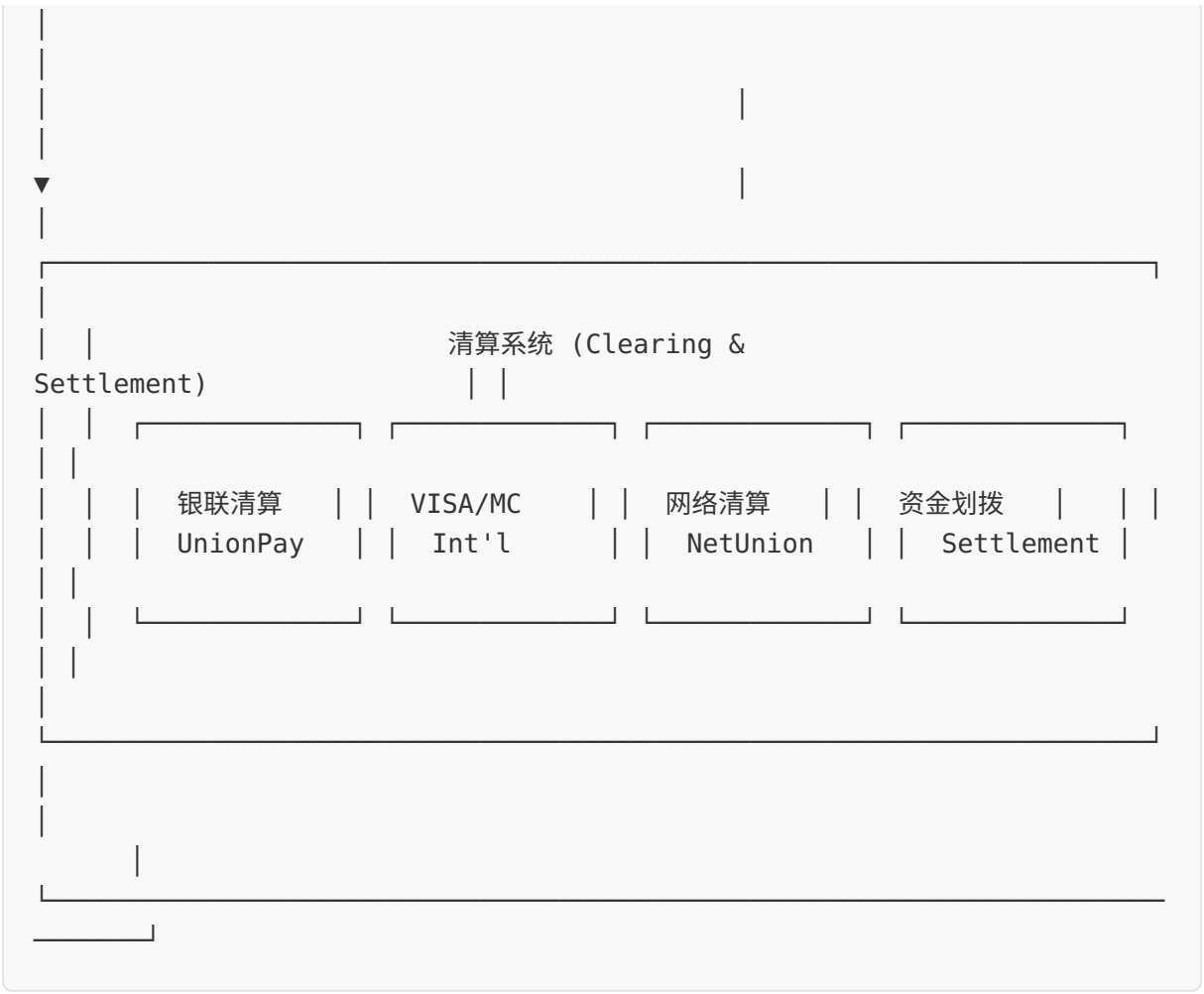
3.1 信用卡处理系统

3.1.1 信用卡系统架构

信用卡系统是银行面向个人客户的重要支付工具，涵盖发卡、授权、清算、催收等全生命周期管理。




授权系统
(Authorization)
实时授权引擎
额度检查
欺诈检测
3D Secure
风控规则
卡管理系统 (Card Management)
发卡管理
Issuance
额度管理
Limit Mgt
账务管理
Accounting
积分管理
Rewards
分期管理
Installment
逾期管理
Overdue
催收管理
Collection
销户管理
Close



3.1.2 信用卡主表设计

```
-- ============================================================
-- 信用卡主表 - CARDMAST
-- ============================================================

CREATE TABLE CARDMAST (
    -- 卡号主键
    CARD_NO             CHAR(19)        NOT NULL,

    -- 账户信息
    ACCT_NO             CHAR(32)        NOT NULL,  -- 信用卡账户号
    CUST_ID             CHAR(20)        NOT NULL,

    -- 卡产品信息
    PROD_CD             CHAR(10)        NOT NULL,
    CARD_TYPE           CHAR(2)         NOT NULL,  -- 01-普卡 02-金卡
03-白金...
    CARD_BRAND          CHAR(2)         NOT NULL,  -- 01-银联 02-VISA
```

```
03-MC 04-Amex

    -- 物理卡信息
    EMV_AID             VARCHAR(50),              -- EMV应用标识
    CHIP_SEQ            DECIMAL(2, 0)   DEFAULT 1, -- 芯片序号
    MAG_STRIPE_DATA     VARCHAR(100),             -- 磁道数据加密存储
    CVV2                CHAR(3),                  -- CVV2(加密)
    ICVV                CHAR(3),                  -- ICVV(加密)

    -- 有效期
    ISSUE_DT            DECIMAL(8, 0)   NOT NULL,
    EXPIRY_DT           DECIMAL(8, 0)   NOT NULL,  -- YYYYMM格式
    VALID_THRU          DECIMAL(8, 0),             -- 卡片有效期截止日期

    -- 卡状态
    CARD_STAT           CHAR(2)         DEFAULT '01', -- 01-正常 02-未激
活...
    STAT_CHG_DT         DECIMAL(8, 0),

    -- 信用额度
    CREDIT_LIMIT        DECIMAL(19, 2)  NOT NULL,
    TEMP_LIMIT          DECIMAL(19, 2),             -- 临时额度
    TEMP_LIMIT_EXP_DT   DECIMAL(8, 0),
    CASH_LIMIT_PCT      DECIMAL(5, 2)   DEFAULT 50, -- 取现额度比例%

    -- 当前欠款
    STMT_BAL            DECIMAL(19, 2)  DEFAULT 0,  -- 账单金额
    CURR_BAL            DECIMAL(19, 2)  DEFAULT 0,  -- 当前总欠款
    AVAILABLE_AMT       DECIMAL(19, 2)  DEFAULT 0,  -- 可用额度

    -- 账单信息
    STMT_DAY            DECIMAL(2, 0)   DEFAULT 1,  -- 账单日
    DUE_DAY             DECIMAL(2, 0)   DEFAULT 25, -- 还款日
    LAST_STMT_DT        DECIMAL(8, 0),
    NEXT_STMT_DT        DECIMAL(8, 0),

    -- 还款信息
    MIN_PAY_PCT         DECIMAL(5, 2)   DEFAULT 10, -- 最低还款比例%
    MIN_PAY_AMT         DECIMAL(19, 2)  DEFAULT 0,
    LAST_PAY_DT         DECIMAL(8, 0),
    LAST_PAY_AMT        DECIMAL(19, 2)  DEFAULT 0,
    CONSEC_PAY_CNT      DECIMAL(3, 0)   DEFAULT 0,  -- 连续还款期数

    -- 逾期信息
    ODR_BAL             DECIMAL(19, 2)  DEFAULT 0,  -- 逾期金额
```

```
    ODR_DAYS            DECIMAL(5, 0)   DEFAULT 0,  -- 逾期天数
    ODR_STAGE           CHAR(1),                    -- 逾期阶段: 1-M1 2-
M2...

    -- 费用利率
    PURCHASE_RATE       DECIMAL(9, 6)   NOT NULL,  -- 消费日利率
    CASH_RATE           DECIMAL(9, 6)   NOT NULL,  -- 取现日利率
    ODR_RATE            DECIMAL(9, 6)   NOT NULL,  -- 逾期日利率
    ANNUAL_FEE          DECIMAL(19, 2)  DEFAULT 0,  -- 年费

    -- 账单地址
    STMT_ADDR_TYPE      CHAR(1)         DEFAULT '1', -- 1-家庭 2-单位 3-
其他
    STMT_EMAIL          VARCHAR(100),
    STMT_SMS_FLAG       CHAR(1)         DEFAULT '1',

    -- 自动还款
    AUTO_PAY_FLAG       CHAR(1)         DEFAULT '0',
    AUTO_PAY_TYPE       CHAR(1),                    -- 1-全额 2-最低
    AUTO_PAY_ACCT       CHAR(32),                   -- 自动还款账户

    -- 交易限制
    DAILY_PURCH_LIMIT   DECIMAL(19, 2),
    DAILY_CASH_LIMIT    DECIMAL(19, 2),
    SINGLE_TRANS_LIMIT  DECIMAL(19, 2),

    -- 密码
    PIN_OFFSET          VARCHAR(100),               -- PIN偏移量(加密)
    PIN_ERR_CNT         DECIMAL(1, 0)   DEFAULT 0,
    PIN_LOCK_DT         DECIMAL(8, 0),

    -- 积分
    POINT_BAL           DECIMAL(10, 0)  DEFAULT 0,
    POINT_EXP_DT        DECIMAL(8, 0),

    -- 挂失/补卡
    LOST_DT             DECIMAL(8, 0),
    LOST_RSN            VARCHAR(200),
    RENEWAL_CNT         DECIMAL(2, 0)   DEFAULT 0,  -- 换卡次数

    -- 营销标签
    CUST_SEG            CHAR(2),                    -- 客户分层
    RISK_LEVEL          CHAR(1)         DEFAULT 'N', -- N-正常 H-高危

    -- 审计字段
```

```
    CREATE_TS           TIMESTAMP       DEFAULT CURRENT_TIMESTAMP,
    CREATE_USER         VARCHAR(20)     DEFAULT USER,
    UPDATE_TS           TIMESTAMP       DEFAULT CURRENT_TIMESTAMP,
    UPDATE_USER         VARCHAR(20)     DEFAULT USER,

    -- 约束
    CONSTRAINT PK_CARDMAST PRIMARY KEY (CARD_NO),
    CONSTRAINT FK_CARDMAST_CUST FOREIGN KEY (CUST_ID)
        REFERENCES CUSTMAST(CUST_ID),
    CONSTRAINT FK_CARDMAST_ACCT FOREIGN KEY (ACCT_NO)
        REFERENCES ACCTMAST(ACCT_NO)

) RCDFMT CARDMASTR;

-- 信用卡交易明细表
CREATE TABLE CARDTRANS (
    TRANS_SEQ           CHAR(32)        NOT NULL,
    CARD_NO             CHAR(19)        NOT NULL,
    ACCT_NO             CHAR(32)        NOT NULL,

    TRANS_DT            DECIMAL(8, 0)   NOT NULL,
    TRANS_TM            DECIMAL(6, 0)   NOT NULL,
    POST_DT             DECIMAL(8, 0),              -- 入账日期

    TRANS_TYPE          CHAR(4)         NOT NULL,  -- 交易类型
    MERCH_TYPE          CHAR(4),                   -- MCC码

    TRANS_AMT           DECIMAL(19, 2)  NOT NULL,
    TRANS_CCY           CHAR(3)         NOT NULL,
    SETTLE_AMT          DECIMAL(19, 2),             -- 清算金额
    SETTLE_CCY          CHAR(3),
    CONV_RATE           DECIMAL(15, 8),             -- 汇率

    AUTH_CODE           CHAR(6),                   -- 授权码
    REF_NO              CHAR(12),                  -- 检索参考号
    BATCH_NO            CHAR(6),
    VOUCHER_NO          CHAR(12),

    MERCH_ID            VARCHAR(15),
    MERCH_NAME          VARCHAR(100),
    MERCH_COUNTRY       CHAR(3),
    MERCH_CITY          VARCHAR(50),

    POS_ENTRY_MODE      CHAR(4),                   -- POS输入方式
    POS_COND_CODE       CHAR(2),                   -- POS条件码
```

```
    -- 渠道信息
    ACQ_INST_ID         CHAR(11),                   -- 收单机构
    FWD_INST_ID         CHAR(11),                   -- 转发机构
    ISS_INST_ID         CHAR(11),                   -- 发卡机构

    -- 3D Secure
    ECI_IND             CHAR(2),                    -- 电子商务标识
    CAVV                VARCHAR(50),                -- 持卡人验证值

    -- 原交易信息(退货/撤销用)
    ORIG_TRANS_SEQ      CHAR(32),
    ORIG_TRANS_DT       DECIMAL(8, 0),
    ORIG_AUTH_CODE      CHAR(6),

    -- 分期信息
    INSTL_FLAG          CHAR(1)         DEFAULT '0',
    INSTL_CNT           DECIMAL(3, 0),
    INSTL_FEE           DECIMAL(19, 2),

    -- 争议标志
    CHARGEBACK_FLAG     CHAR(1)         DEFAULT '0',
    CHARGEBACK_DT       DECIMAL(8, 0),

    TRANS_STAT          CHAR(1)         DEFAULT '0', -- 0-正常 1-撤销 2-
退货...

    CREATE_TS           TIMESTAMP       DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT PK_CARDTRANS PRIMARY KEY (TRANS_SEQ),
    CONSTRAINT FK_CARDTRANS_CARD FOREIGN KEY (CARD_NO)
        REFERENCES CARDMAST(CARD_NO)
);
```

3.1.3 实时授权引擎

RPG IV - 信用卡授权处理:

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('CARDAUTH');

// ============================================================
// 程序: CDAUTH
// 描述: 信用卡实时授权引擎
```

```
// 功能: 处理POS/网上支付授权请求
// 响应时间要求: < 200ms
// ================================================================

// 常量定义
dcl-c AUTH_APPROVE     '00';
dcl-c AUTH_DECLINE     '05';
dcl-c AUTH_PICKUP      '04';
dcl-c AUTH_REFER       '01';
dcl-c AUTH_CALL_AUTH   '03';

dcl-c TRANS_TYPE_PURCH '1000';
dcl-c TRANS_TYPE_CASH  '2000';
dcl-c TRANS_TYPE_PREAUTH '1001';
dcl-c TRANS_TYPE_COMP '1002';

// 授权请求结构
dcl-ds AuthRequest qualified template;
    msgType         char(4);        // 报文类型 0100-授权 0120-撤销
    cardNo          char(19);
    expDate         char(4);        // YYMM
    cvv2            char(4);        // 可能包含CVV2
    pinData         varchar(100);   // PIN数据(加密)

    transType       char(4);        // 交易类型
    transAmt        packed(19:2);
    transCcy        char(3);

    merchType       char(4);        // MCC码
    merchId         varchar(15);
    merchName       varchar(100);
    merchCountry    char(3);

    posEntryMode    char(4);        // POS输入方式
    posCondCode     char(2);

    termId          char(8);
    acqInstId       char(11);
    fwdInstId       char(11);

    traceNo         char(6);
    transDate       packed(8);
    transTime       packed(6);

    // 3DS信息
    eciInd          char(2);
```

```
    cavv            varchar(50);
    xid             varchar(50);
end-ds;

// 授权响应结构
dcl-ds AuthResponse qualified template;
    respCode        char(2);
    respDesc        char(100);
    authCode        char(6);
    refNo           char(12);

    // 风控信息
    riskScore       packed(5:2);
    riskRuleHit     varchar(200);

    // 授权限额
    authLimit       packed(19:2);
    remainLimit     packed(19:2);

    // 建议操作
    suggestAction   char(1);        // A-批准 D-拒绝 R-转人工
end-ds;

// ===================================================================
// 主授权处理入口
// ===================================================================
dcl-proc ProcessAuthRequest export;
    dcl-pi *n likeds(AuthResponse);
        req     likeds(AuthRequest) const;
    end-pi;

    dcl-ds resp likeds(AuthResponse) inz;
    dcl-ds cardData likeds(CARDMAST_T);
    dcl-s validationRtn char(2);

    // 1. 基本参数校验
    if not ValidateAuthRequest(req : resp);
        return resp;
    endif;

    // 2. 读取卡信息(带缓存)
    if not GetCardData(req.cardNo : cardData : resp);
        return resp;
    endif;

    // 3. 卡片有效性检查
```

```
    if not CheckCardValidity(cardData : resp);
        return resp;
    endif;

    // 4. CVV/PIN验证
    if not CheckSecurityCode(req : cardData : resp);
        return resp;
    endif;

    // 5. 额度检查
    if not CheckCreditLimit(req : cardData : resp);
        return resp;
    endif;

    // 6. 欺诈检测(规则引擎)
    if not FraudDetection(req : cardData : resp);
        // 根据风控建议决定是否拒绝
        if resp.suggestAction = 'D';
            resp.respCode = AUTH_DECLINE;
            resp.respDesc = '交易风险较高';
            return resp;
        endif;
    endif;

    // 7. 授权批准
    resp.respCode = AUTH_APPROVE;
    resp.respDesc = '授权成功';
    resp.authCode = GenerateAuthCode();
    resp.refNo = GenerateRefNo(req);
    resp.remainLimit = cardData.AVAILABLE_AMT - req.transAmt;

    // 8. 实时授权登记(异步处理)
    RegisterAuthRecord(req : resp);

    // 9. 预占额度(实时更新)
    ReserveCreditLimit(cardData.ACCT_NO : req.transAmt);

    return resp;
end-proc;

// ============================================================
// 基本参数校验
// ============================================================
dcl-proc ValidateAuthRequest;
    dcl-pi *n ind;
        req     likeds(AuthRequest) const;
```

```
        resp    likeds(AuthResponse);
    end-pi;

    // 检查卡号(Luhn校验)
    if not LuhnCheck(req.cardNo);
        resp.respCode = '14';
        resp.respDesc = '无效卡号';
        return *off;
    endif;

    // 检查金额
    if req.transAmt <= 0;
        resp.respCode = '13';
        resp.respDesc = '无效金额';
        return *off;
    endif;

    // 检查货币代码
    if req.transCcy = '';
        resp.respCode = '12';
        resp.respDesc = '无效交易';
        return *off;
    endif;

    return *on;
end-proc;

// ====================================================================
// Luhn算法校验卡号
// ====================================================================
dcl-proc LuhnCheck;
    dcl-pi *n ind;
        cardNo  char(19) const;
    end-pi;

    dcl-s i packed(3);
    dcl-s len packed(3);
    dcl-s sum packed(5);
    dcl-s digit packed(1);
    dcl-s double packed(2);

    len = %len(%trim(cardNo));
    if len < 13 or len > 19;
        return *off;
    endif;
```

```
    sum = 0;

    for i = len downto 1;
        digit = %int(%subst(cardNo : i : 1));
        if %rem(len - i : 2) = 1;
            double = digit * 2;
            if double > 9;
                double = double - 9;
            endif;
            sum = sum + double;
        else;
            sum = sum + digit;
        endif;
    endfor;

    return (%rem(sum : 10) = 0);
end-proc;

// ====================================================================
// 获取卡数据
// ====================================================================
dcl-proc GetCardData;
    dcl-pi *n ind;
        cardNo      char(19) const;
        cardData    likeds(CARDMAST_T);
        resp        likeds(AuthResponse);
    end-pi;

    // 优先从缓存读取
    if GetCardFromCache(cardNo : cardData);
        return *on;
    endif;

    // 从数据库读取
    exec SQL
        SELECT * INTO :cardData
        FROM CARDMAST
        WHERE CARD_NO = :cardNo;

    if SQLCODE <> 0;
        resp.respCode = '14';
        resp.respDesc = '卡号不存在';
        return *off;
    endif;

    // 写入缓存
```

```
    PutCardToCache(cardNo : cardData);

    return *on;
end-proc;

// ===================================================================
// 检查卡片有效性
// ===================================================================
dcl-proc CheckCardValidity;
    dcl-pi *n ind;
        cardData    likeds(CARDMAST_T) const;
        resp        likeds(AuthResponse);
    end-pi;

    dcl-s today packed(8);

    today = %int(%char(%date() : *iso0));

    // 检查有效期
    if cardData.EXPIRY_DT < %int(%subst(%char(today) : 3 : 4));
        resp.respCode = '54';
        resp.respDesc = '卡片已过期';
        return *off;
    endif;

    // 检查卡状态
    select;
        when cardData.CARD_STAT = '02';  // 未激活
            resp.respCode = '78';
            resp.respDesc = '卡片未激活';
            return *off;
        when cardData.CARD_STAT = '03';  // 已挂失
            resp.respCode = '41';
            resp.respDesc = '卡片已挂失';
            return *off;
        when cardData.CARD_STAT = '04';  // 已冻结
            resp.respCode = '62';
            resp.respDesc = '卡片已冻结';
            return *off;
        when cardData.CARD_STAT = '05';  // 已销户
            resp.respCode = '54';
            resp.respDesc = '卡片已失效';
            return *off;
        when cardData.CARD_STAT = '06';  // 密码锁定
            resp.respCode = '75';
```

```
            resp.respDesc = '密码错误次数超限';
            return *off;
    endsl;

    return *on;
end-proc;

// ====================================================================
// 检查安全码(CVV/PIN)
// ====================================================================
dcl-proc CheckSecurityCode;
    dcl-pi *n ind;
        req         likeds(AuthRequest) const;
        cardData    likeds(CARDMAST_T) const;
        resp        likeds(AuthResponse);
    end-pi;

    // POS联机交易验证PIN
    if %subst(req.posEntryMode : 1 : 2) = '05' or   // 芯片
       %subst(req.posEntryMode : 1 : 2) = '07';      // 接触式芯片
        if not ValidatePIN(req.pinData : cardData.PIN_OFFSET);
            resp.respCode = '55';
            resp.respDesc = '密码错误';
            return *off;
        endif;
    endif;

    // CVV2验证(无卡交易)
    if %subst(req.posEntryMode : 1 : 2) = '01' and  // 手工输入
       req.cvv2 <> '';
        if not ValidateCVV2(req.cvv2 : cardData.CVV2);
            resp.respCode = 'N7';
            resp.respDesc = 'CVV2错误';
            return *off;
        endif;
    endif;

    return *on;
end-proc;

// ====================================================================
// 检查信用额度
// ====================================================================
dcl-proc CheckCreditLimit;
    dcl-pi *n ind;
        req         likeds(AuthRequest) const;
```

```
        cardData    likeds(CARDMAST_T) const;
        resp        likeds(AuthResponse);
    end-pi;

    // 检查可用额度
    if req.transAmt > cardData.AVAILABLE_AMT;
        resp.respCode = '51';
        resp.respDesc = '余额不足';
        return *off;
    endif;

    // 检查取现额度
    if req.transType = TRANS_TYPE_CASH;
        dcl-s cashLimit packed(19:2);
        cashLimit = cardData.CREDIT_LIMIT * cardData.CASH_LIMIT_PCT /
100;
        if req.transAmt > cashLimit;
            resp.respCode = '61';
            resp.respDesc = '超出取现额度';
            return *off;
        endif;
    endif;

    // 检查单笔限额
    if req.transAmt > cardData.SINGLE_TRANS_LIMIT;
        resp.respCode = '61';
        resp.respDesc = '超出单笔限额';
        return *off;
    endif;

    // 检查日累计限额
    dcl-s dailyTotal packed(19:2);
    exec SQL
        SELECT COALESCE(SUM(TRANS_AMT), 0) INTO :dailyTotal
        FROM CARDTRANS
        WHERE CARD_NO = :req.cardNo
          AND TRANS_DT = :req.transDate
          AND TRANS_TYPE = :req.transType
          AND TRANS_STAT = '0';

    if req.transType = TRANS_TYPE_PURCH;
        if (dailyTotal + req.transAmt) > cardData.DAILY_PURCH_LIMIT;
            resp.respCode = '61';
            resp.respDesc = '超出日累计消费限额';
            return *off;
        endif;
```

```
    endif;

    return *on;
end-proc;

// ================================================================
// 欺诈检测
// ================================================================
dcl-proc FraudDetection;
    dcl-pi *n ind;
        req         likeds(AuthRequest) const;
        cardData    likeds(CARDMAST_T) const;
        resp        likeds(AuthResponse);
    end-pi;

    dcl-s riskScore packed(5:2);
    dcl-s ruleHits varchar(200);

    riskScore = 0;
    ruleHits = '';

    // 规则1: 异地交易
    if req.merchCountry <> 'CHN';
        riskScore = riskScore + 20;
        ruleHits = %trim(ruleHits) + '|CROSS_BORDER';
    endif;

    // 规则2: 大额交易
    if req.transAmt > cardData.CREDIT_LIMIT * 0.5;
        riskScore = riskScore + 15;
        ruleHits = %trim(ruleHits) + '|LARGE_AMT';
    endif;

    // 规则3: 高频交易
    dcl-s recentCnt packed(5);
    exec SQL
        SELECT COUNT(*) INTO :recentCnt
        FROM CARDTRANS
        WHERE CARD_NO = :req.cardNo
          AND TRANS_DT = :req.transDate
          AND TRANS_STAT = '0';

    if recentCnt > 5;
        riskScore = riskScore + 10;
        ruleHits = %trim(ruleHits) + '|HIGH_FREQ';
    endif;
```

```
    // 规则4: MCC风险商户
    dcl-s mccRisk char(1);
    exec SQL
        SELECT RISK_FLAG INTO :mccRisk
        FROM MCCMAST
        WHERE MCC_CD = :req.merchType;

    if mccRisk = 'H';  // 高风险MCC
        riskScore = riskScore + 25;
        ruleHits = %trim(ruleHits) + '|RISK_MCC';
    endif;

    // 规则5: 无卡交易
    if %subst(req.posEntryMode : 1 : 2) = '01';  // 手工输入
        riskScore = riskScore + 10;
        ruleHits = %trim(ruleHits) + '|CNP';
    endif;

    // 设置风控结果
    resp.riskScore = riskScore;
    resp.riskRuleHit = ruleHits;

    // 根据分数确定建议操作
    if riskScore >= 80;
        resp.suggestAction = 'D';  // 拒绝
    elseif riskScore >= 50;
        resp.suggestAction = 'R';  // 转人工
    else;
        resp.suggestAction = 'A';  // 批准
    endif;

    return (riskScore < 80);
end-proc;

// ============================================================
// 生成授权码
// ============================================================
dcl-proc GenerateAuthCode;
    dcl-pi *n char(6) end-pi;

    dcl-s authCode char(6);
    dcl-s seqNum packed(10);

    // 获取序列号
    exec SQL
```

```
        SELECT NEXT VALUE FOR AUTH_CODE_SEQ INTO :seqNum
        FROM SYSIBM.SYSDUMMY1;

    // 生成6位授权码
    authCode = %subst(%editc(seqNum : 'X') : 5 : 6);

    return authCode;
end-proc;

// ================================================================
// 预占额度
// ================================================================
dcl-proc ReserveCreditLimit;
    dcl-pi *n;
        acctNo      char(32) const;
        amt         packed(19:2) const;
    end-pi;

    // 使用乐观锁更新额度
    exec SQL
        UPDATE CARDMAST
        SET AVAILABLE_AMT = AVAILABLE_AMT - :amt,
            UPDATE_TS = CURRENT_TIMESTAMP
        WHERE ACCT_NO = :acctNo
          AND AVAILABLE_AMT >= :amt;

end-proc;
```

3.2 借记卡系统

3.2.1 借记卡核心处理

借记卡与存款账户直接关联，交易实时扣减账户余额，是最基础的银行卡产品。

```
-- ================================================================
-- 借记卡主表 - DEBITCARD
-- ================================================================

CREATE TABLE DEBITCARD (
    CARD_NO             CHAR(19)        NOT NULL,
    ACCT_NO             CHAR(32)        NOT NULL,  -- 关联账户

    CARD_TYPE           CHAR(2)         NOT NULL,  -- 01-普卡 02-金卡...
```

```
    CARD_BRAND          CHAR(2)         NOT NULL,  -- 01-银联 02-VISA...

    ISSUE_DT            DECIMAL(8, 0)   NOT NULL,
    EXPIRY_DT           DECIMAL(8, 0)   NOT NULL,

    CARD_STAT           CHAR(2)         DEFAULT '01',

    -- 交易限额
    DAILY_DR_LIMIT      DECIMAL(19, 2),
    DAILY_DR_AMT        DECIMAL(19, 2)  DEFAULT 0,  -- 当日已用额度
    SINGLE_DR_LIMIT     DECIMAL(19, 2),

    -- 密码
    PIN_OFFSET          VARCHAR(100),
    PIN_ERR_CNT         DECIMAL(1, 0)   DEFAULT 0,
    PIN_LOCK_FLAG       CHAR(1)         DEFAULT '0',

    -- 渠道控制
    POS_FLAG            CHAR(1)         DEFAULT '1',
    ATM_FLAG            CHAR(1)         DEFAULT '1',
    ONLINE_FLAG         CHAR(1)         DEFAULT '1',
    OVERSEAS_FLAG       CHAR(1)         DEFAULT '1',

    -- 小额免密
    QRPAY_LIMIT         DECIMAL(19, 2)  DEFAULT 1000,
    QRPAY_FLAG          CHAR(1)         DEFAULT '1',

    CREATE_TS           TIMESTAMP       DEFAULT CURRENT_TIMESTAMP,
    UPDATE_TS           TIMESTAMP       DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT PK_DEBITCARD PRIMARY KEY (CARD_NO),
    CONSTRAINT FK_DEBITCARD_ACCT FOREIGN KEY (ACCT_NO)
        REFERENCES ACCTMAST(ACCT_NO)
);
```

RPG IV - 借记卡交易处理:

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('DBTCARD');

// ============================================================
// 程序: DBTTRANS
// 描述: 借记卡交易处理
// 特点: 实时扣账，与核心账户强关联
```

```
// ============================================================

dcl-f DEBITCARD usage(*update) keyed;
dcl-f ACCTMAST usage(*update) keyed;

// 借记卡交易请求
dcl-ds DBTTransReq qualified template;
    cardNo          char(19);
    transType       char(4);        // 1001-消费 2001-取现
    transAmt        packed(19:2);
    transCcy        char(3);
    pinData         varchar(100);
    posEntryMode    char(4);
    merchId         varchar(15);
    termId          char(8);
end-ds;

// ============================================================
// 主处理程序
// ============================================================
dcl-proc ProcessDebitTrans export;
    dcl-pi *n likeds(TransResponse);
        req     likeds(DBTTransReq) const;
    end-pi;

    dcl-ds resp likeds(TransResponse) inz;
    dcl-ds cardData likeds(DEBITCARD_T);
    dcl-ds acctData likeds(ACCTMAST_T);

    // 读取卡信息
    chain req.cardNo DEBITCARD;
    if not %found(DEBITCARD);
        resp.rtnCode = '14';
        resp.rtnMsg = '卡号不存在';
        return resp;
    endif;

    // 验证卡片状态
    if CARD_STAT <> '01';
        resp.rtnCode = '62';
        resp.rtnMsg = '卡片状态异常';
        return resp;
    endif;

    // 渠道控制检查
    if not CheckChannelLimit(req : resp);
```

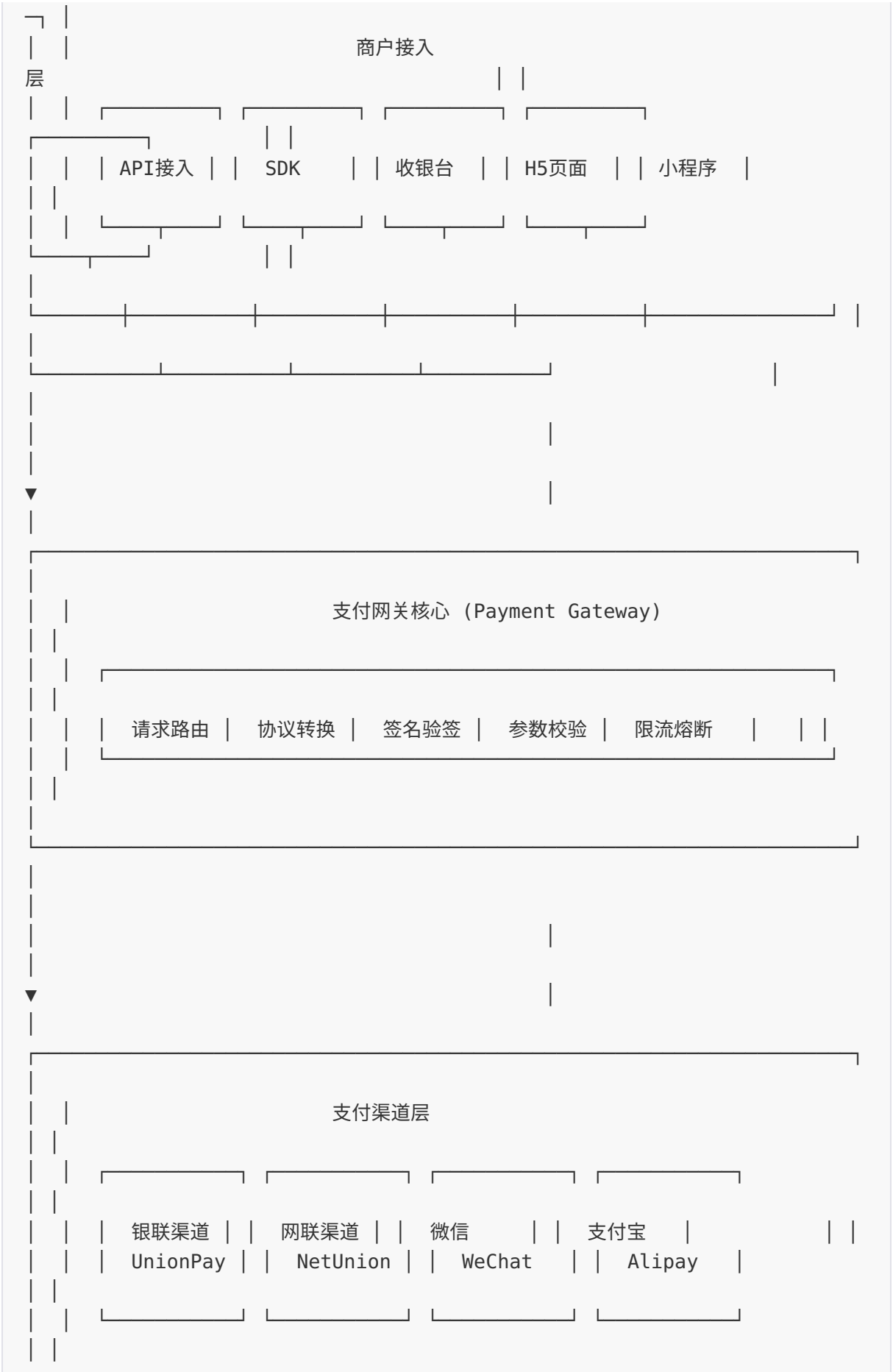
```
        return resp;
    endif;

    // 读取账户信息
    chain ACCT_NO ACCTMAST;
    if not %found(ACCTMAST);
        resp.rtnCode = '14';
        resp.rtnMsg = '账户不存在';
        return resp;
    endif;

    // 余额检查
    if req.transAmt > AVAIL_BAL;
        resp.rtnCode = '51';
        resp.rtnMsg = '余额不足';
        return resp;
    endif;

    // 限额检查
    if req.transAmt > SINGLE_DR_LIMIT;
        resp.rtnCode = '61';
        resp.rtnMsg = '超出单笔限额';
        return resp;
    endif;

    if (DAILY_DR_AMT + req.transAmt) > DAILY_DR_LIMIT;
        resp.rtnCode = '61';
        resp.rtnMsg = '超出日累计限额';
        return resp;
    endif;

    // 扣除账户余额
    ACCT_BAL = ACCT_BAL - req.transAmt;
    AVAIL_BAL = AVAIL_BAL - req.transAmt;
    update ACCTMASTR;

    // 更新卡当日限额
    DAILY_DR_AMT = DAILY_DR_AMT + req.transAmt;
    update DEBITCARDR;

    // 组装成功响应
    resp.rtnCode = '00';
    resp.rtnMsg = '交易成功';
    resp.acctBal = ACCT_BAL;
    resp.availBal = AVAIL_BAL;
```

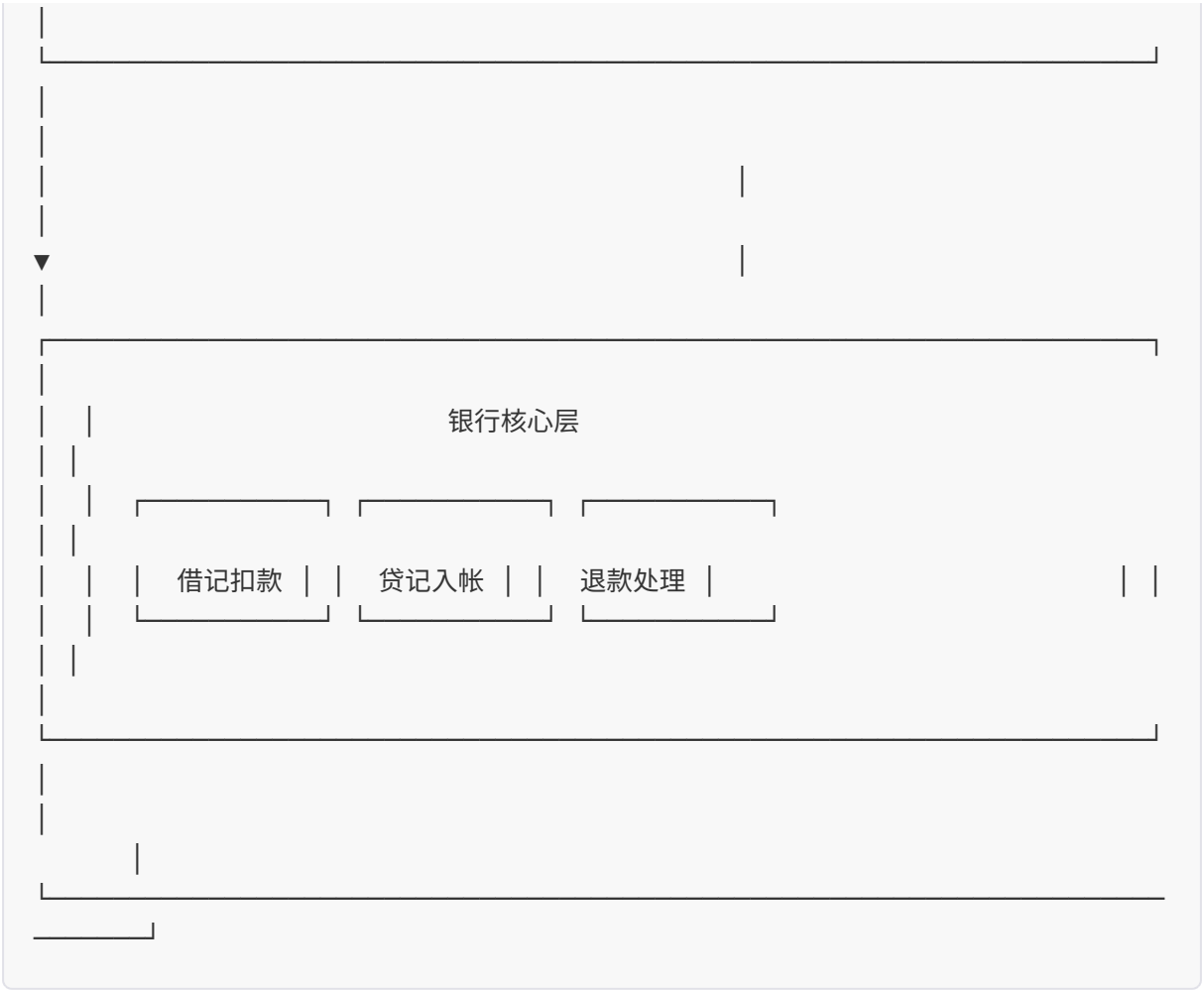
```
    return resp;
end-proc;

// ===================================================================
// 渠道控制检查
// ===================================================================
dcl-proc CheckChannelLimit;
    dcl-pi *n ind;
        req     likeds(DBTTransReq) const;
        resp    likeds(TransResponse);
    end-pi;

    // 检查POS交易
    if req.transType = '1001' and POS_FLAG = '0';
        resp.rtnCode = '57';
        resp.rtnMsg = 'POS交易已禁用';
        return *off;
    endif;

    // 检查ATM取现
    if req.transType = '2001' and ATM_FLAG = '0';
        resp.rtnCode = '57';
        resp.rtnMsg = 'ATM交易已禁用';
        return *off;
    endif;

    return *on;
end-proc;
```

3.3 支付网关

3.3.1 支付网关架构

```
┌──────────────────────────────────────────────────────────────────────────
──────┐
│                                支付网关架
构                                          │
├──────────────────────────────────────────────────────────────────────────
──────┤
│
      │
│
└──────────────────────────────────────────────────────────────────────────
```


商户接入层
API接入
SDK
收银台
H5页面
小程序
支付网关核心 (Payment Gateway)
请求路由
协议转换
签名验签
参数校验
限流熔断
支付渠道层
银联渠道
UnionPay
网联渠道
NetUnion
微信
WeChat
支付宝
Alipay



3.3.2 统一支付接口

RPG IV - 支付网关核心:

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('PAYGATE');

// ============================================================
// 程序: PAYGW
// 描述: 统一支付网关接口
// 功能: 对接多渠道支付，统一协议转换
// ============================================================

// 支付请求
dcl-ds PayRequest qualified template;
    version         char(5);        // 接口版本
    charset         char(10);       // 编码
    signType        char(10);       // 签名类型: RSA2/SM3
```

```
    mchId           char(20);       // 商户号
    mchOrderNo      char(64);       // 商户订单号

    payType         char(10);       // 支付方式
    payProduct      char(10);       // 支付产品

    orderAmt        packed(19:2);   // 订单金额
    orderCcy        char(3);        // 币种

    subject         varchar(256);   // 订单标题
    body            varchar(512);   // 订单描述

    notifyUrl       varchar(256);   // 异步通知地址
    returnUrl       varchar(256);   // 同步跳转地址

    expireTime      packed(14);     // 过期时间YYYYMMDDHHMMSS
    clientIp        char(16);       // 客户端IP
    deviceInfo      varchar(100);   // 设备信息

    userId          char(32);       // 用户标识
    acctNo          char(32);       // 银行账户(协议支付)

    extra           varchar(2000);  // 扩展参数(JSON)
    sign            varchar(512);   // 签名
end-ds;

// 支付响应
dcl-ds PayResponse qualified template;
    rtnCode         char(5);
    rtnMsg          varchar(200);

    mchId           char(20);
    mchOrderNo      char(64);
    orderNo         char(64);       // 平台订单号

    orderAmt        packed(19:2);
    payStatus       char(2);        // 0-待支付 1-支付中 2-成功 3-失败

    payTime         packed(14);     // 支付时间

    credential      varchar(2000);  // 支付凭证(JSON)

    sign            varchar(512);
end-ds;
```

```
// ============================================================
// 统一支付接口
// ============================================================
dcl-proc UnifiedPay export;
    dcl-pi *n likeds(PayResponse);
        req     likeds(PayRequest) const;
    end-pi;

    dcl-ds resp likeds(PayResponse) inz;
    dcl-s orderNo char(64);

    // 1. 参数校验
    if not ValidatePayRequest(req : resp);
        return resp;
    endif;

    // 2. 验签
    if not VerifyRequestSign(req : resp);
        resp.rtnCode = 'SIGN_FAIL';
        resp.rtnMsg = '验签失败';
        return resp;
    endif;

    // 3. 幂等性检查
    if CheckDuplicateOrder(req.mchId : req.mchOrderNo);
        resp.rtnCode = 'DUPLICATE';
        resp.rtnMsg = '重复订单';
        return resp;
    endif;

    // 4. 生成平台订单号
    orderNo = GenerateOrderNo(req.mchId);

    // 5. 保存订单
    SavePayOrder(req : orderNo);

    // 6. 根据支付方式路由到不同渠道
    select;
        when req.payType = 'UNIONPAY';
            resp = RouteToUnionPay(req : orderNo);
        when req.payType = 'WECHAT';
            resp = RouteToWeChat(req : orderNo);
        when req.payType = 'ALIPAY';
            resp = RouteToAlipay(req : orderNo);
        when req.payType = 'QUICK';
```

```
            resp = RouteToQuickPay(req : orderNo);
        other;
            resp.rtnCode = 'UNSUPPORTED';
            resp.rtnMsg = '不支持的支付方式';
            return resp;
    endsl;

    resp.orderNo = orderNo;

    return resp;
end-proc;

// ============================================================
// 查询订单状态
// ============================================================
dcl-proc QueryOrder export;
    dcl-pi *n likeds(QueryResponse);
        mchId       char(20) const;
        mchOrderNo  char(64) const;
        orderNo     char(64) const;
    end-pi;

    dcl-ds resp likeds(QueryResponse) inz;

    exec SQL
        SELECT MCH_ID, MCH_ORDER_NO, ORDER_NO, ORDER_AMT,
               PAY_STATUS, PAY_TIME, CREATE_TS
        INTO :resp.mchId, :resp.mchOrderNo, :resp.orderNo,
             :resp.orderAmt, :resp.payStatus, :resp.payTime, :resp.crea
teTs
        FROM PAY_ORDER
        WHERE (MCH_ID = :mchId AND MCH_ORDER_NO = :mchOrderNo)
           OR ORDER_NO = :orderNo;

    if SQLCODE <> 0;
        resp.rtnCode = 'ORDER_NOT_EXIST';
        resp.rtnMsg = '订单不存在';
    else;
        resp.rtnCode = 'SUCCESS';
        resp.rtnMsg = '查询成功';
    endif;

    return resp;
end-proc;

// ============================================================
```

```
// 退款处理
// ============================================================
dcl-proc RefundOrder export;
    dcl-pi *n likeds(RefundResponse);
        mchId           char(20) const;
        mchOrderNo      char(64) const;
        mchRefundNo     char(64) const;
        refundAmt       packed(19:2) const;
        refundReason    varchar(200) const;
    end-pi;

    dcl-ds resp likeds(RefundResponse) inz;
    dcl-s orglOrderNo char(64);
    dcl-s orglPayStatus char(2);
    dcl-s orglPayAmt packed(19:2);
    dcl-s refundedAmt packed(19:2);

    // 查询原订单
    exec SQL
        SELECT ORDER_NO, PAY_STATUS, ORDER_AMT
        INTO :orglOrderNo, :orglPayStatus, :orglPayAmt
        FROM PAY_ORDER
        WHERE MCH_ID = :mchId AND MCH_ORDER_NO = :mchOrderNo;

    if SQLCODE <> 0;
        resp.rtnCode = 'ORDER_NOT_EXIST';
        resp.rtnMsg = '原订单不存在';
        return resp;
    endif;

    // 检查支付状态
    if orglPayStatus <> '2';
        resp.rtnCode = 'PAY_NOT_SUCCESS';
        resp.rtnMsg = '原订单未支付成功';
        return resp;
    endif;

    // 查询已退款金额
    exec SQL
        SELECT COALESCE(SUM(REFUND_AMT), 0) INTO :refundedAmt
        FROM PAY_REFUND
        WHERE ORDER_NO = :orglOrderNo AND REFUND_STATUS IN ('1', '2');

    // 检查退款金额
    if (refundedAmt + refundAmt) > orglPayAmt;
        resp.rtnCode = 'REFUND_AMT_EXCEED';
```

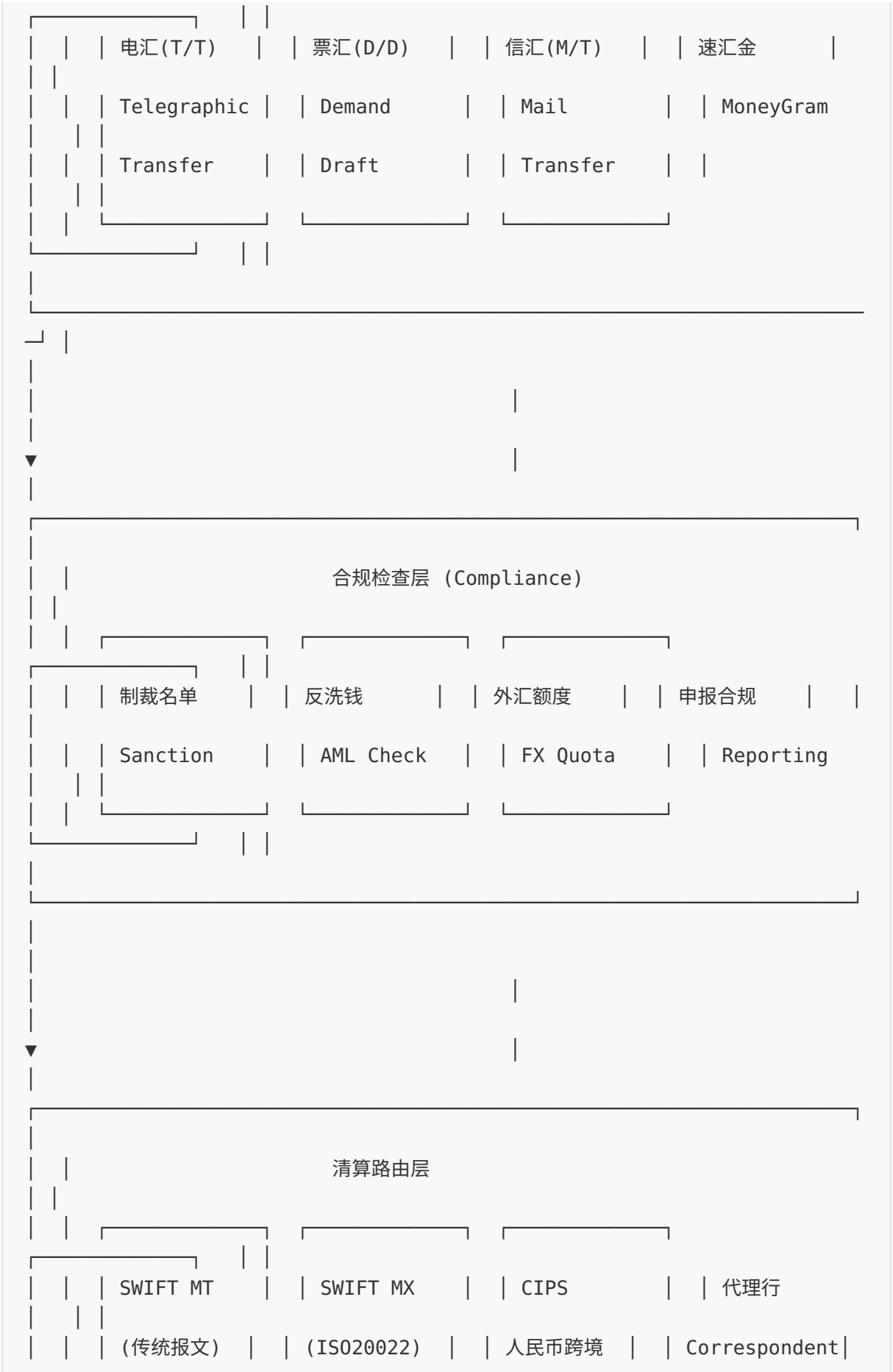
```
        resp.rtnMsg = '退款金额超过原订单金额';
        return resp;
    endif;

    // 调用渠道退款
    // ... 渠道退款逻辑 ...

    // 保存退款记录
    exec SQL
        INSERT INTO PAY_REFUND (
            REFUND_NO, ORDER_NO, MCH_ID, MCH_REFUND_NO,
            REFUND_AMT, REFUND_REASON, REFUND_STATUS, CREATE_TS
        ) VALUES (
            :GenerateRefundNo(), :orglOrderNo, :mchId, :mchRefundNo,
            :refundAmt, :refundReason, '1', CURRENT_TIMESTAMP
        );

    resp.rtnCode = 'SUCCESS';
    resp.rtnMsg = '退款申请已受理';

    return resp;
end-proc;
```

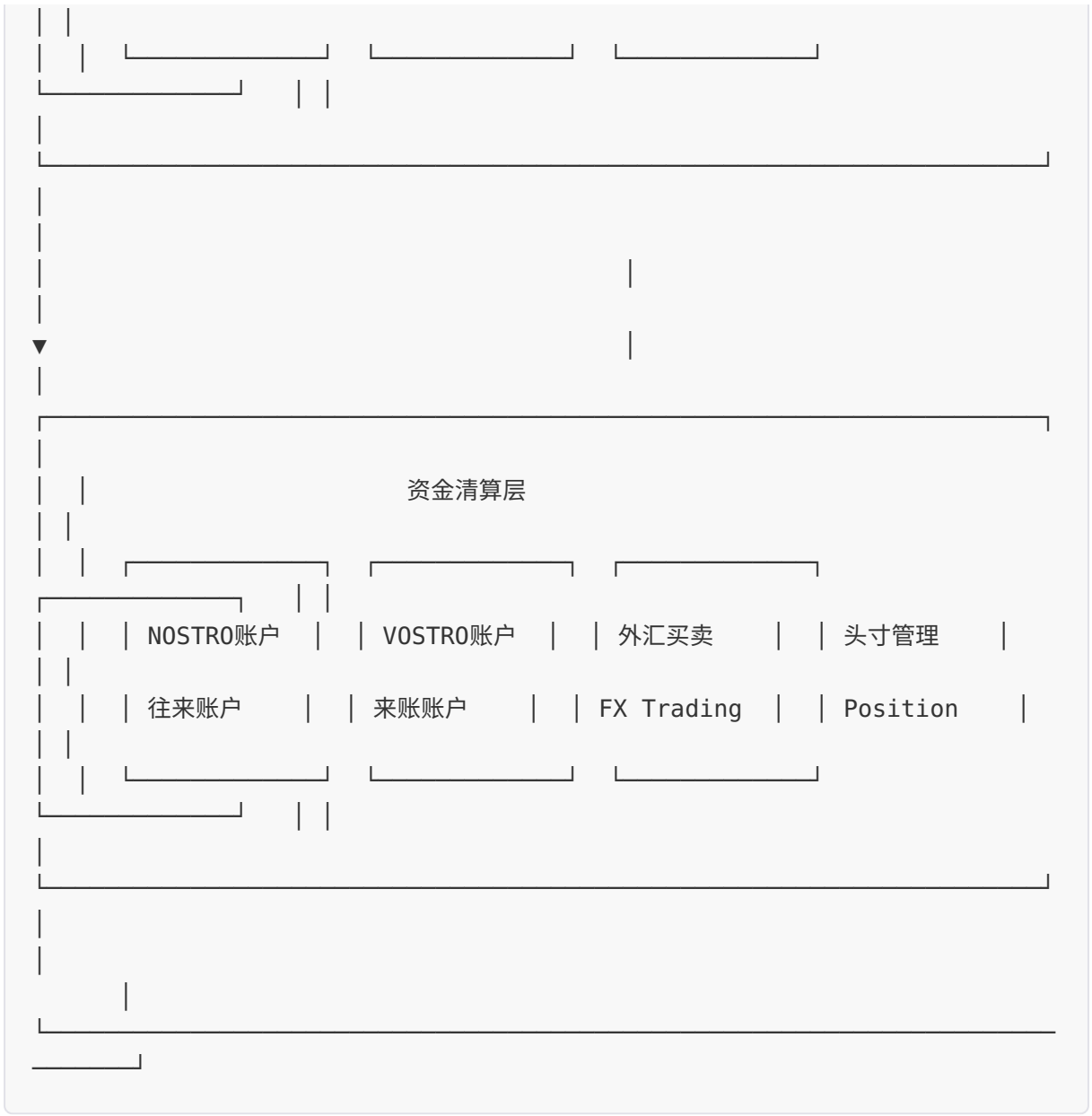
3.4 跨境支付系统

3.4.1 跨境支付架构

跨境支付涉及多币种、多时区、多监管要求，是银行支付体系中最复杂的模块之一。

```
┌──────────────────────────────────────────────────────────────────────────────
──────┐
│                                跨境支付系统架
构                                          │
├──────────────────────────────────────────────────────────────────────────────
──────┤
│
      │
│
┌──────────────────────────────────────────────────────────────────────────────
┐ │
│  │                              汇款发起
层                                          │ │
│  │  ┌──────────────┐   ┌──────────────┐   ┌──────────────┐
```


电汇(T/T)
票汇(D/D)
信汇(M/T)
速汇金
Telegraphic Transfer
Demand Draft
Mail Transfer
MoneyGram
合规检查层 (Compliance)
制裁名单
Sanction
反洗钱
AML Check
外汇额度
FX Quota
申报合规
Reporting
清算路由层
SWIFT MT
(传统报文)
SWIFT MX
(ISO20022)
CIPS
人民币跨境
代理行
Correspondent



3.4.2 跨境汇款处理

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('CROSSBORDER');

// ============================================================
// 程序: CBREMit
// 描述: 跨境汇款处理
// ============================================================

// 汇款请求结构
dcl-ds RemitRequest qualified template;
```

```
    remitRefNo      char(32);       // 汇款参考号
    remitType       char(2);        // 01-个人 02-对公

    // 汇款人信息
    remitCustId     char(20);
    remitName       varchar(100);
    remitAddr       varchar(200);
    remitAcctNo     char(32);

    // 收款人信息
    beneName        varchar(100);
    beneAddr        varchar(200);
    beneAcctNo      char(34);       // IBAN格式
    beneBankName    varchar(100);
    beneBankAddr    varchar(200);
    beneSwiftCode   char(11);
    beneBankCode    char(20);       // 本地清算代码

    // 汇款信息
    remitCcy        char(3);
    remitAmt        packed(19:2);
    feeType         char(1);        // 1-OUR 2-SHA 3-BEN
    feeAmt          packed(19:2);

    // 费用承担
    ourCharge       packed(19:2)    inz(0);  // 汇款人承担
    benCharge       packed(19:2)    inz(0);  // 收款人承担

    // 汇款用途
    purposeCode     char(6);        // 国际收支代码
    purposeDesc     varchar(200);

    // 报文类型
    msgType         char(5);        // MT103/MT202/ISO20022

    // 优先级
    priority        char(1);        // 1-普通 2-加急 3-特急
end-ds;

// ===================================================================
// 跨境汇款主处理
// ===================================================================
dcl-proc ProcessRemittance export;
    dcl-pi *n likeds(RemitResponse);
        req     likeds(RemitRequest) const;
```

```
    end-pi;

    dcl-ds resp likeds(RemitResponse) inz;
    dcl-s swiftMsg varchar(10000);

    // 1. 参数校验
    if not ValidateRemitRequest(req : resp);
        return resp;
    endif;

    // 2. 客户额度检查
    if not CheckFXQuota(req.remitCustId : req.remitAmt : req.remitCcy :
resp);
        return resp;
    endif;

    // 3. 合规检查
    if not ComplianceCheck(req : resp);
        return resp;
    endif;

    // 4. 费率计算
    CalculateRemitFee(req);

    // 5. 扣款处理
    if not DebitRemitterAcct(req : resp);
        return resp;
    endif;

    // 6. 生成SWIFT报文
    select;
        when req.msgType = 'MT103';
            swiftMsg = BuildMT103(req);
        when req.msgType = 'ISO20022';
            swiftMsg = BuildISO20022(req);
        other;
            resp.rtnCode = '01';
            resp.rtnMsg = '不支持的报文类型';
            return resp;
    endsl;

    // 7. 发送报文到SWIFT网络
    if not SendSwiftMessage(swiftMsg : resp);
        // 冲正扣款
        ReverseDebit(req);
        return resp;
```

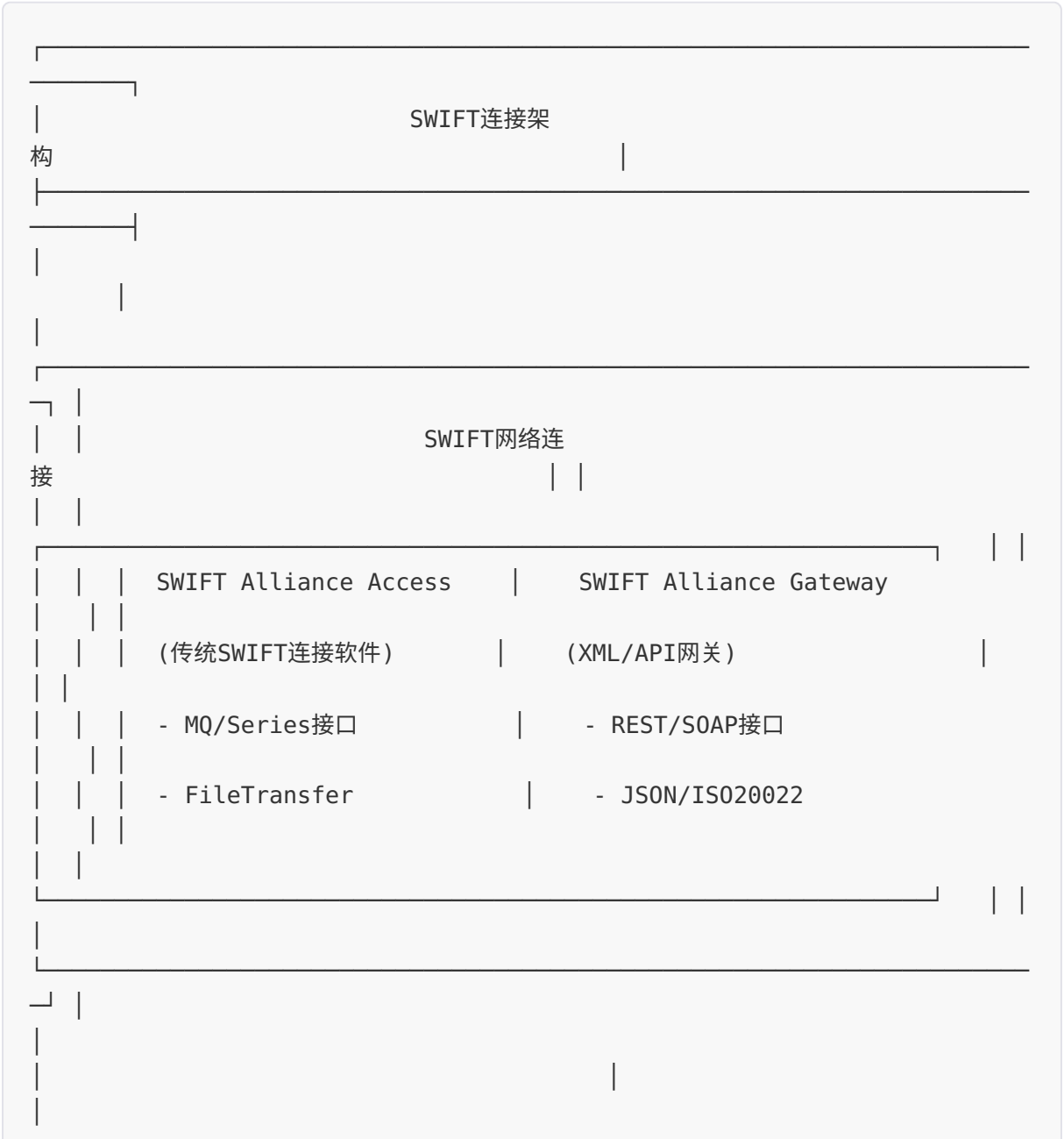
```
    endif;

    // 8. 登记汇款记录
    RegisterRemittance(req : swiftMsg);

    resp.rtnCode = '00';
    resp.rtnMsg = '汇款已发送';
    resp.remitRefNo = req.remitRefNo;

    return resp;
end-proc;

// ============================================================
// 构建MT103报文
// ============================================================
dcl-proc BuildMT103;
    dcl-pi *n varchar(10000);
        req     likeds(RemitRequest) const;
    end-pi;

    dcl-s msg varchar(10000);
    dcl-s block4 varchar(8000);

    // 报文头
    msg = '{1:F01' + GetSwiftBIC() + '0000000000}' +
          '{2:I103' + req.beneSwiftCode + 'N}';

    // Block 4 - 报文正文
    block4 = '{4:' +
             ':20:' + req.remitRefNo +
             ':23B:CRED' +
             ':32A:' + GetValueDate(req.remitCcy) +
                       req.remitCcy +
                       FormatSwiftAmt(req.remitAmt) +
             ':50K:/' + req.remitAcctNo + '\n' +
                        req.remitName + '\n' +
                        req.remitAddr +
             ':59:/' + req.beneAcctNo + '\n' +
                       req.beneName + '\n' +
                       req.beneAddr +
             ':71A:' + GetFeeCode(req.feeType) +
             ':72:/ACC/' + req.purposeDesc;

    // 中间行(如有)
    if req.intermediarySwift <> '';
        block4 = block4 + ':56A:' + req.intermediarySwift;
```

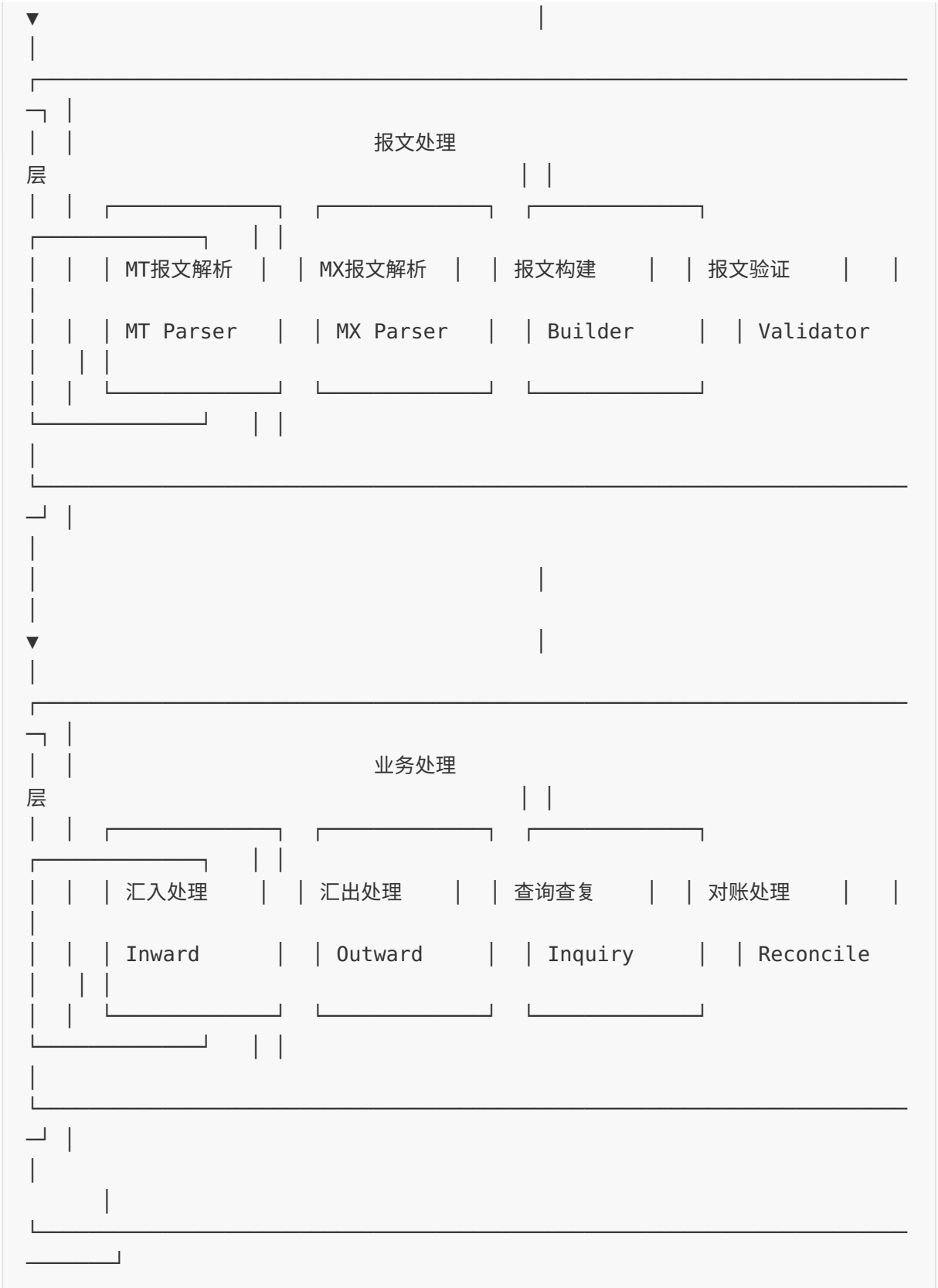
```
    endif;

    // 账户行(如有)
    if req.acctWithSwift <> '';
        block4 = block4 + ':57A:' + req.acctWithSwift;
    else;
        block4 = block4 + ':57D://' + req.beneBankCode + '\n' +
                          req.beneBankName + '\n' +
                          req.beneBankAddr;
    endif;

    block4 = block4 + '-}';

    // 校验和
    msg = msg + block4 + '{5:{CHK:' + CalculateChecksum(block4) + '}}';

    return msg;
end-proc;

// ===================================================================
// 合规检查
// ===================================================================
dcl-proc ComplianceCheck;
    dcl-pi *n ind;
        req     likeds(RemitRequest) const;
        resp    likeds(RemitResponse);
    end-pi;

    // 制裁名单检查
    if CheckSanctionList(req.beneName : req.beneAddr :
req.beneSwiftCode);
        resp.rtnCode = '91';
        resp.rtnMsg = '命中制裁名单';
        return *off;
    endif;

    // AML检查
    if AMLScreening(req);
        resp.rtnCode = '92';
        resp.rtnMsg = '需要AML审核';
        resp.needManualReview = '1';
        return *off;
    endif;

    // 涉及国家检查
    if CheckRestrictedCountry(req.beneAddr);
```

```
        resp.rtnCode = '93';
        resp.rtnMsg = '涉及受制裁国家/地区';
        return *off;
    endif;

    return *on;
end-proc;
```

3.5 SWIFT集成

3.5.1 SWIFT连接架构




报文处理
层
MT报文解析
MT Parser
MX报文解析
MX Parser
报文构建
Builder
报文验证
Validator
业务处理
层
汇入处理
Inward
汇出处理
Outward
查询查复
Inquiry
对账处理
Reconcile

3.5.2 SWIFT报文处理

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('SWIFT');

// ============================================================
// 程序: SWIFTPRC
// 描述: SWIFT报文处理核心
// ============================================================

// SWIFT报文结构
dcl-ds SwiftMessage qualified template;
    msgId           char(32);
    msgType         char(5);        // MT103/MT202/pacs.008...
    msgFormat       char(2);        // MT/MX
    senderBIC       char(11);
    receiverBIC     char(11);
    msgTime         timestamp;
    block1          varchar(100);   // 基本头块
    block2          varchar(100);   // 应用头块
    block3          varchar(500);   // 用户头块
    block4          varchar(8000);  // 正文块
    block5          varchar(500);   // 尾部块
    rawMsg          varchar(10000);
end-ds;

// 报文字段结构
dcl-ds SwiftField qualified template;
    fieldTag        char(4);        // :20:, :32A: 等
    fieldOption     char(1);        // A, B, C, D...
    fieldContent    varchar(1000);
    fieldLines      varchar(100) dim(5);
end-ds;

// ============================================================
// 解析SWIFT报文
// ============================================================
dcl-proc ParseSwiftMessage export;
    dcl-pi *n likeds(SwiftMessage);
        rawMsg      varchar(10000) const;
    end-pi;

    dcl-ds msg likeds(SwiftMessage) inz;
```

```
    dcl-s startPos packed(5);
    dcl-s endPos packed(5);
    dcl-s blockId char(1);
    dcl-s blockContent varchar(8000);

    msg.rawMsg = rawMsg;

    // 解析Block 1 (基本头块)
    startPos = %scan('{1:' : rawMsg);
    if startPos > 0;
        endPos = %scan('}' : rawMsg : startPos);
        msg.block1 = %subst(rawMsg : startPos + 3 : endPos - startPos -
3);
        msg.senderBIC = %subst(msg.block1 : 6 : 8);
    endif;

    // 解析Block 2 (应用头块)
    startPos = %scan('{2:' : rawMsg);
    if startPos > 0;
        endPos = %scan('}' : rawMsg : startPos);
        msg.block2 = %subst(rawMsg : startPos + 3 : endPos - startPos -
3);
        msg.msgType = %subst(msg.block2 : 2 : 3);
        if %subst(msg.block2 : 1 : 1) = 'I';
            msg.msgFormat = 'MT';
            msg.receiverBIC = %subst(msg.block2 : 5 : 12);
        endif;
    endif;

    // 解析Block 4 (正文)
    startPos = %scan('{4:' : rawMsg);
    if startPos > 0;
        endPos = %scan('-}' : rawMsg : startPos);
        msg.block4 = %subst(rawMsg : startPos + 3 : endPos - startPos -
3);
    endif;

    return msg;
end-proc;

// ================================================================
// 提取字段内容
// ================================================================
dcl-proc GetSwiftField export;
    dcl-pi *n likeds(SwiftField);
        msg         likeds(SwiftMessage) const;
```

```
        fieldTag    char(4) const;
    end-pi;

    dcl-ds field likeds(SwiftField) inz;
    dcl-s tagPos packed(5);
    dcl-s nextTagPos packed(5);
    dcl-s content varchar(1000);

    field.fieldTag = fieldTag;

    // 在Block 4中查找字段
    tagPos = %scan(':' + fieldTag + ':' : msg.block4);

    if tagPos > 0;
        // 找到下一个字段标签位置
        nextTagPos = %scan(':' : msg.block4 : tagPos + %len(fieldTag) +
2);

        if nextTagPos = 0;
            nextTagPos = %len(%trim(msg.block4)) + 1;
        endif;

        content = %subst(msg.block4 : tagPos + %len(fieldTag) + 2 :
                         nextTagPos - tagPos - %len(fieldTag) - 2);

        field.fieldContent = content;

        // 解析多行内容
        SplitFieldLines(content : field.fieldLines);
    endif;

    return field;
end-proc;

// ================================================================
// MT103汇入处理
// ================================================================
dcl-proc ProcessMT103Inward export;
    dcl-pi *n likeds(RemitResponse);
        msg         likeds(SwiftMessage) const;
    end-pi;

    dcl-ds resp likeds(RemitResponse) inz;
    dcl-ds field20 likeds(SwiftField);
    dcl-ds field32A likeds(SwiftField);
    dcl-ds field50 likeds(SwiftField);
```

```
    dcl-ds field59 likeds(SwiftField);

    dcl-s senderRef char(16);
    dcl-s valueDate packed(8);
    dcl-s ccy char(3);
    dcl-s amt packed(19:2);
    dcl-s beneAcct char(34);

    // 提取字段
    field20 = GetSwiftField(msg : '20');   // 发报行参考号
    field32A = GetSwiftField(msg : '32A'); // 起息日/币种/金额
    field50 = GetSwiftField(msg : '50');   // 汇款人
    field59 = GetSwiftField(msg : '59');   // 收款人

    // 解析32A字段: YYMMDDCCYAMT
    if field32A.fieldContent <> '';
        valueDate = ParseSwiftDate(%subst(field32A.fieldContent : 1 :
6));
        ccy = %subst(field32A.fieldContent : 7 : 3);
        amt = ParseSwiftAmt(%subst(field32A.fieldContent : 10));
    endif;

    // 解析59字段: /账号\n名称\n地址
    if field59.fieldContent <> '';
        beneAcct = %subst(field59.fieldContent : 2 :
                          %scan('\n' : field59.fieldContent) - 2);
    endif;

    // 查找本行账户
    if not FindLocalAccount(beneAcct);
        // 退汇处理
        GenerateReturnMT199(msg : '受益人账号不存在');
        resp.rtnCode = '01';
        resp.rtnMsg = '账号不存在，已退汇';
        return resp;
    endif;

    // 入账处理
    if CreditBeneficiaryAcct(beneAcct : amt : ccy :
field20.fieldContent);
        // 发送MT199确认
        GenerateConfirmationMT199(msg);
        resp.rtnCode = '00';
        resp.rtnMsg = '汇入款已入账';
    else;
        resp.rtnCode = '99';
```

```
        resp.rtnMsg = '入账失败';
    endif;

    return resp;
end-proc;
```

3.6 ISO 20022实现

3.6.1 ISO 20022报文结构

ISO 20022是SWIFT组织推出的新一代报文标准，采用XML格式，信息更丰富、结构更灵活。

```
<?xml version="1.0" encoding="UTF-8"?>
<Document xmlns="urn:iso:std:iso:20022:tech:xsd:pacs.008.001.08">
    <FIToFICstmrCdtTrf>
        <!-- 报文头 -->
        <GrpHdr>
            <MsgId>MSG202602070001</MsgId>
            <CreDtTm>2026-02-07T10:30:00</CreDtTm>
            <NbOfTxs>1</NbOfTxs>
            <CtrlSum>10000.00</CtrlSum>
            <InstgAgt>
                <FinInstnId>
                    <BICFI>ABOCCNBJXXX</BICFI>
                </FinInstnId>
            </InstgAgt>
            <InstdAgt>
                <FinInstnId>
                    <BICFI>CHASUS33XXX</BICFI>
                </FinInstnId>
            </InstdAgt>
        </GrpHdr>

        <!-- 交易信息 -->
        <CdtTrfTxInf>
            <PmtId>
                <InstrId>INST202602070001</InstrId>
                <EndToEndId>E2E202602070001</EndToEndId>
                <UETR>97a8b6c5-d4e3-4f2g-8h1i-9j0k1l2m3n4o</UETR>
            </PmtId>

            <IntrBkSttlmAmt Ccy="USD">10000.00</IntrBkSttlmAmt>
            <IntrBkSttlmDt>2026-02-07</IntrBkSttlmDt>
```

```
<!-- 手续费信息 -->
<ChrgsInf>
    <Amt Ccy="USD">15.00</Amt>
    <Agt>
        <FinInstnId>
            <BICFI>ABOCCNBJXXX</BICFI>
        </FinInstnId>
    </Agt>
</ChrgsInf>

<!-- 汇款人信息 -->
<Dbtr>
    <Nm>张三</Nm>
    <PstlAdr>
        <AdrLine>北京市朝阳区建国路88号</AdrLine>
    </PstlAdr>
    <Id>
        <PrvtId>
            <Othr>
                <Id>110101199001011234</Id>
                <SchmeNm>
                    <Cd>NIDN</Cd>
                </SchmeNm>
            </Othr>
        </PrvtId>
    </Id>
</Dbtr>

<DbtrAcct>
    <Id>
        <Othr>
            <Id>6222001234567890123</Id>
        </Othr>
    </Id>
</DbtrAcct>

<DbtrAgt>
    <FinInstnId>
        <BICFI>ABOCCNBJXXX</BICFI>
    </FinInstnId>
</DbtrAgt>

<!-- 收款人信息 -->
<CdtrAgt>
    <FinInstnId>
```

```
                    <BICFI>CHASUS33XXX</BICFI>
                </FinInstnId>
            </CdtrAgt>

            <Cdtr>
                <Nm>John Smith</Nm>
                <PstlAdr>
                    <AdrLine>100 Wall Street</AdrLine>
                    <Ctry>US</Ctry>
                </PstlAdr>
            </Cdtr>

            <CdtrAcct>
                <Id>
                    <IBAN>US29NWBK60161331926819</IBAN>
                </Id>
            </CdtrAcct>

            <!-- 汇款用途 -->
            <RmtInf>
                <Ustrd>Education Fee</Ustrd>
            </RmtInf>

            <!-- 制裁筛查信息 -->
            <RgltryRptg>
                <DbtCdtRptgInd>DEBT</DbtCdtRptgInd>
            </RgltryRptg>
        </CdtTrfTxInf>
    </FIToFICstmrCdtTrf>
</Document>
```

3.6.2 ISO 20022报文构建(RPG + SQL XML)

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('ISO20022');

// ============================================================
// 程序: ISO20022GEN
// 描述: ISO 20022 XML报文生成
// ============================================================

// ============================================================
// 生成pacs.008客户汇款报文
```

```
// ============================================================
dcl-proc BuildPacs008 export;
    dcl-pi *n varchar(20000);
        req     likeds(RemitRequest) const;
    end-pi;

    dcl-s xml varchar(20000);
    dcl-s uetr varchar(36);

    // 生成UETR(唯一端对端交易参考)
    uetr = GenerateUETR();

    // 使用SQL XML功能构建报文
    exec SQL
        SELECT XMLSERIALIZE(
            XMLELEMENT(NAME "Document",
                XMLNAMESPACES('urn:iso:std:iso:20022:tech:xsd:pacs.
008.001.08' AS ""),
                XMLELEMENT(NAME "FIToFICstmrCdtTrf",
                    -- Group Header
                    XMLELEMENT(NAME "GrpHdr",
                        XMLELEMENT(NAME "MsgId", :req.remitRefNo),
                        XMLELEMENT(NAME "CreDtTm",
                            TO_CHAR(CURRENT_TIMESTAMP, 'YYYY-MM-
DD"T"HH24:MI:SS')),
                        XMLELEMENT(NAME "NbOfTxs", '1'),
                        XMLELEMENT(NAME "CtrlSum", :req.remitAmt),
                        XMLELEMENT(NAME "InstgAgt",
                            XMLELEMENT(NAME "FinInstnId",
                                XMLELEMENT(NAME
"BICFI", :GetSwiftBIC())
                            )
                        ),
                        XMLELEMENT(NAME "InstdAgt",
                            XMLELEMENT(NAME "FinInstnId",
                                XMLELEMENT(NAME
"BICFI", :req.beneSwiftCode)
                            )
                        )
                    ),
                    -- Credit Transfer Transaction Information
                    XMLELEMENT(NAME "CdtTrfTxInf",
                        XMLELEMENT(NAME "PmtId",
                            XMLELEMENT(NAME
"InstrId", :req.remitRefNo),
                            XMLELEMENT(NAME
```

```
"EndToEndId", :req.remitRefNo),
                            XMLELEMENT(NAME "UETR", :uetr)
                        ),
                        XMLELEMENT(NAME "IntrBkSttlmAmt",
                            XMLATTRIBUTES(:req.remitCcy AS "Ccy"),
                            :req.remitAmt
                        ),
                        XMLELEMENT(NAME "IntrBkSttlmDt",
                            TO_CHAR(CURRENT DATE, 'YYYY-MM-DD')),
                        -- Debtor
                        XMLELEMENT(NAME "Dbtr",
                            XMLELEMENT(NAME "Nm", :req.remitName)
                        ),
                        XMLELEMENT(NAME "DbtrAcct",
                            XMLELEMENT(NAME "Id",
                                XMLELEMENT(NAME "Othr",
                                    XMLELEMENT(NAME
"Id", :req.remitAcctNo)
                                )
                            )
                        ),
                        -- Creditor
                        XMLELEMENT(NAME "Cdtr",
                            XMLELEMENT(NAME "Nm", :req.beneName)
                        ),
                        XMLELEMENT(NAME "CdtrAcct",
                            XMLELEMENT(NAME "Id",
                                XMLELEMENT(NAME
"IBAN", :req.beneAcctNo)
                            )
                        )
                    )
                )
            ) AS CLOB(20000)
        )
        INTO :xml
        FROM SYSIBM.SYSDUMMY1;

    return xml;
end-proc;

// ============================================================
// 解析ISO 20022报文
// ============================================================
dcl-proc ParsePacs008 export;
    dcl-pi *n likeds(Pacs008Data);
```

```
        xmlMsg      varchar(20000) const;
    end-pi;

    dcl-ds data likeds(Pacs008Data);

    // 使用XMLTABLE解析
    exec SQL
        SELECT
            MsgId,
            CreDtTm,
            InstrId,
            UETR,
            SttlmAmt,
            SttlmCcy,
            SttlmDt,
            DbtrNm,
            DbtrAcctId,
            CdtrNm,
            CdtrAcctId
        INTO
            :data.msgId,
            :data.creDtTm,
            :data.instrId,
            :data.uetr,
            :data.sttlmAmt,
            :data.sttlmCcy,
            :data.sttlmDt,
            :data.dbtrNm,
            :data.dbtrAcctId,
            :data.cdtrNm,
            :data.cdtrAcctId
        FROM XMLTABLE(
            XMLNAMESPACES(DEFAULT 'urn:iso:std:iso:20022:tech:xsd:pacs.
008.001.08'),
            '$doc/Document/FIToFICstmrCdtTrf'
            PASSING :xmlMsg AS "doc"
            COLUMNS
                MsgId       VARCHAR(35)     PATH 'GrpHdr/MsgId',
                CreDtTm     TIMESTAMP       PATH 'GrpHdr/CreDtTm',
                InstrId     VARCHAR(35)     PATH 'CdtTrfTxInf/PmtId/
InstrId',
                UETR        VARCHAR(36)     PATH 'CdtTrfTxInf/PmtId/
UETR',
                SttlmAmt    DECIMAL(19,5)   PATH 'CdtTrfTxInf/
IntrBkSttlmAmt',
                SttlmCcy    CHAR(3)         PATH 'CdtTrfTxInf/
```

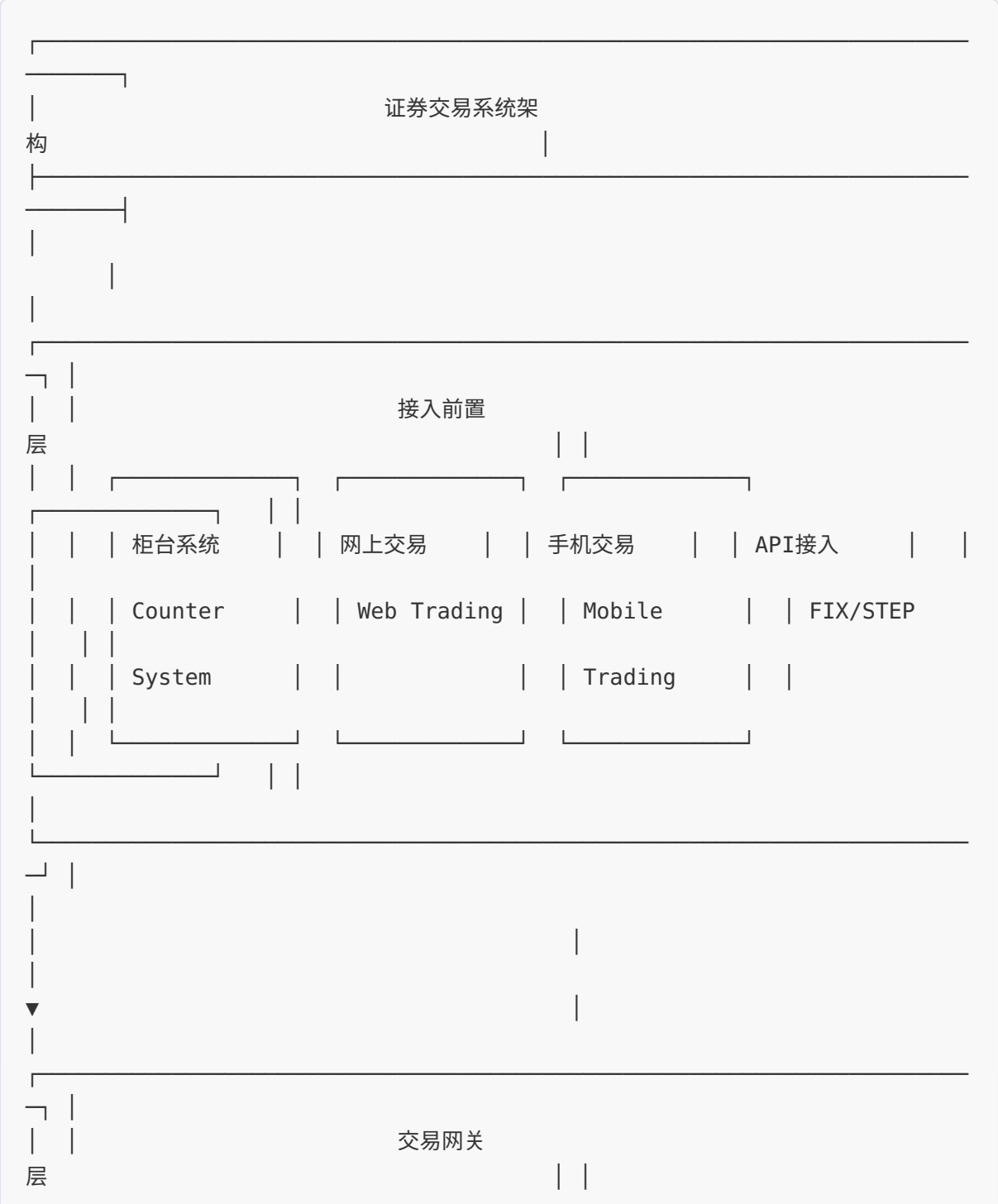
```
IntrBkSttlmAmt/@Ccy',
                SttlmDt     DATE            PATH 'CdtTrfTxInf/
IntrBkSttlmDt',
                DbtrNm      VARCHAR(140)    PATH 'CdtTrfTxInf/Dbtr/Nm',
                DbtrAcctId  VARCHAR(34)     PATH 'CdtTrfTxInf/DbtrAcct/
Id/Othr/Id',
                CdtrNm      VARCHAR(140)    PATH 'CdtTrfTxInf/Cdtr/Nm',
                CdtrAcctId  VARCHAR(34)     PATH 'CdtTrfTxInf/CdtrAcct/
Id/IBAN'
        ) X;

    return data;
end-proc;

// ===============================================================
// 生成UETR
// ===============================================================
dcl-proc GenerateUETR;
    dcl-pi *n varchar(36) end-pi;

    dcl-s uetr varchar(36);
    dcl-s part1 char(8);
    dcl-s part2 char(4);
    dcl-s part3 char(4);
    dcl-s part4 char(4);
    dcl-s part5 char(12);

    // UUID v4格式: xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx
    exec SQL
        SELECT
            LOWER(HEX(RAND())) || '-' ||
            LOWER(HEX(RAND())) || '-4' ||
            SUBSTRING(LOWER(HEX(RAND())), 2, 3) || '-' ||
            CHR(ASCII('8') + MOD(ABS(RAND()*100), 4)) ||
            SUBSTRING(LOWER(HEX(RAND())), 2, 3) || '-' ||
            LOWER(HEX(RAND())) || LOWER(HEX(RAND()))
        INTO :uetr
        FROM SYSIBM.SYSDUMMY1;

    return uetr;
end-proc;
```

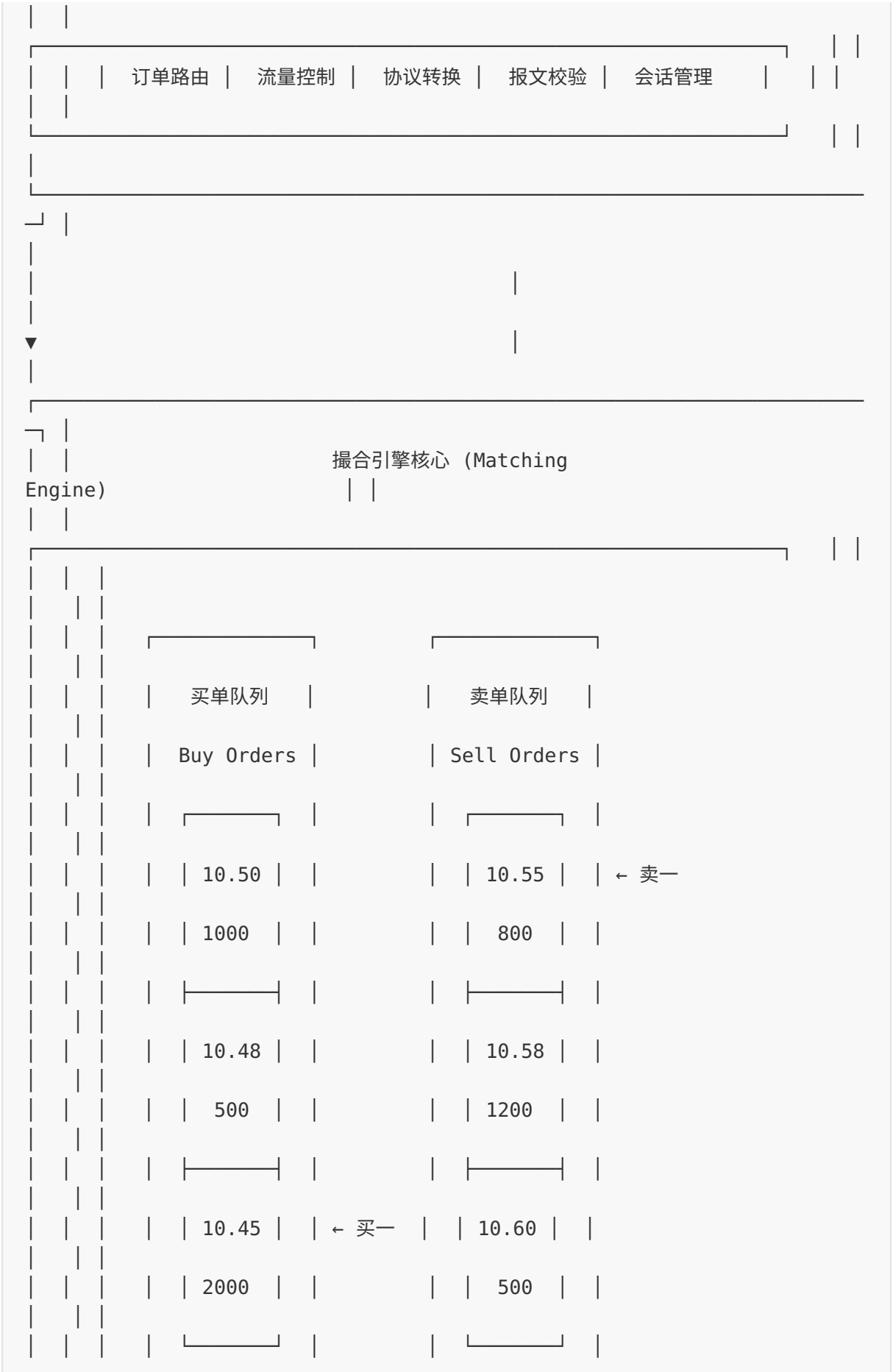
第四章 证券交易系统

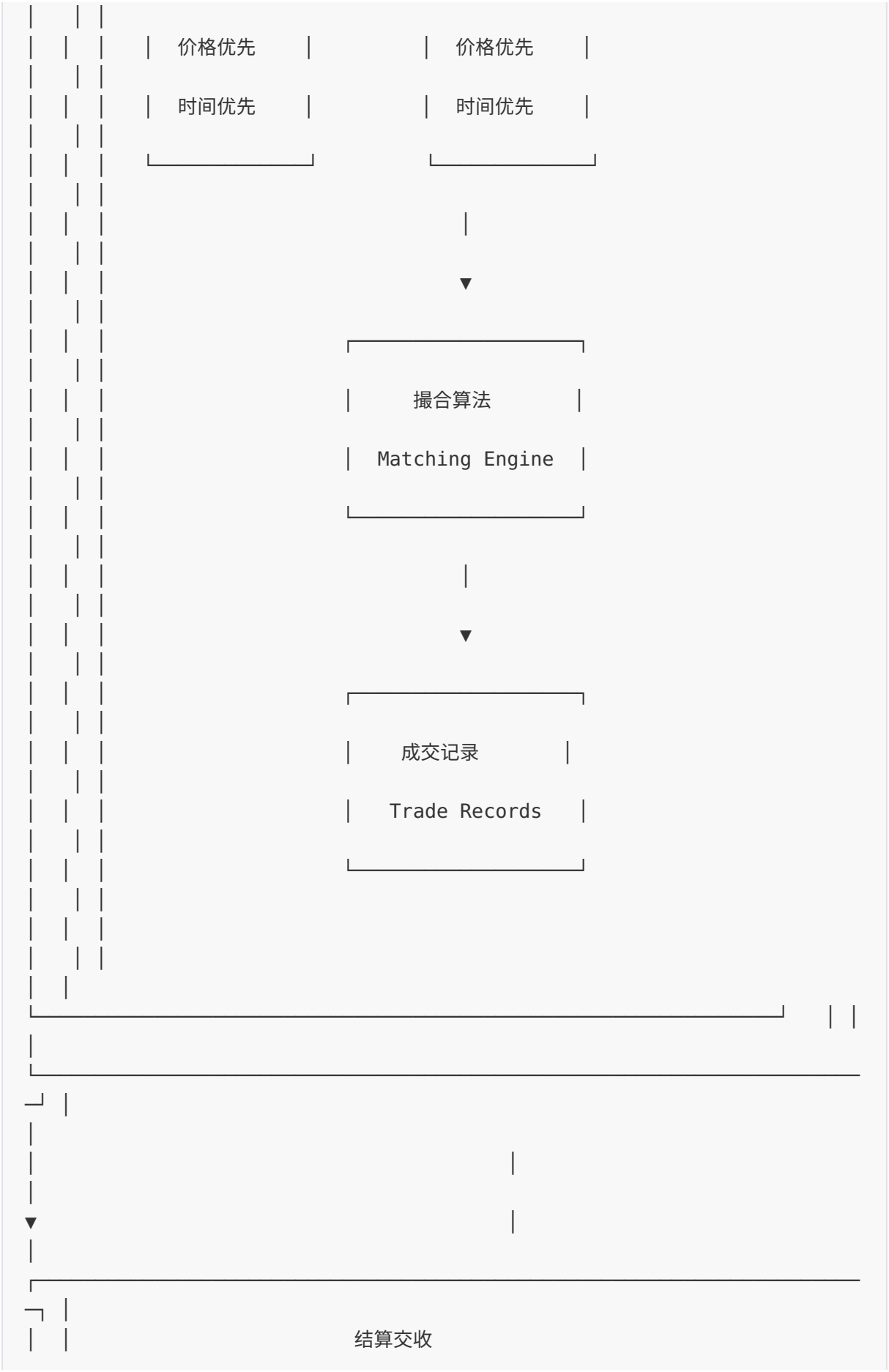
4.1 交易撮合引擎

4.1.1 撮合引擎架构

证券交易系统是金融市场的核心基础设施，撮合引擎决定了交易的公平性和效率。




订单路由
流量控制
协议转换
报文校验
会话管理
撮合引擎核心 (Matching Engine)
买单队列
Buy Orders
卖单队列
Sell Orders
10.50
1000
10.55
800
← 卖一
10.48
500
10.58
1200
10.45
← 买一
10.60
2000
500


价格优先
时间优先
价格优先
时间优先
撮合算法
Matching Engine
成交记录
Trade Records
结算交收

```
层                                          | |
|  |  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐
┌──────────────┐   | |
|  |  | 交易确认    |  | 清算计算    |  | 资金交收    |  | 证券交收    |   |
|
|  |  | Trade       |  | Clearing    |  | Cash        |  | Securities
|   | |
|  |  | Confirm     |  | Calculation |  | Settlement  |  | Settlement
|   | |
|  |  └──────────────┘  └──────────────┘  └──────────────┘
└──────────────┘   | |
|
└──────────────────────────────────────────────────────────────────────────
┘ |
|
      |
└──────────────────────────────────────────────────────────────────────────
──────┘
```

4.1.2 订单薄管理

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('MATCHING');

// ============================================================
// 程序: ORDERBOOK
// 描述: 订单薄管理核心
// ============================================================

// 订单数据结构
dcl-ds Order qualified template;
    orderId         char(20);
    securityId      char(12);       // 证券代码
    orderSide       char(1);        // B-买 S-卖
    orderType       char(1);        // 1-限价 2-市价 3-止损...
    price           packed(15:4);   // 价格
    quantity        packed(15:0);   // 数量

    filledQty       packed(15:0);   // 已成交数量
    remainingQty    packed(15:0);   // 剩余数量

    custId          char(20);
```

```
    accountId       char(20);

    entryTime       timestamp;
    prioritySeq     packed(15:0);   // 时间优先序列号

    orderStatus     char(1);        // 0-待报 1-已报 2-部分成交 3-全部成交
9-已撤
end-ds;

// 成交记录结构
dcl-ds TradeRecord qualified template;
    tradeId         char(20);
    tradeTime       timestamp;
    securityId      char(12);
    tradePrice      packed(15:4);
    tradeQty        packed(15:0);

    buyOrderId      char(20);
    buyCustId       char(20);

    sellOrderId     char(20);
    sellCustId      char(20);
end-ds;

// ================================================================
// 提交订单
// ================================================================
dcl-proc SubmitOrder export;
    dcl-pi *n likeds(OrderResponse);
        newOrder    likeds(Order) const;
    end-pi;

    dcl-ds resp likeds(OrderResponse) inz;
    dcl-ds matchResult likeds(MatchResult);

    // 1. 订单参数校验
    if not ValidateOrder(newOrder : resp);
        return resp;
    endif;

    // 2. 风险检查(价格涨跌幅、持仓、资金)
    if not RiskCheck(newOrder : resp);
        return resp;
    endif;

    // 3. 如果是市价单，转换为限价单(涨跌停价)
```

```
    dcl-ds order likeds(Order);
    order = newOrder;

    if order.orderType = '2';  // 市价单
        order.price = GetMarketOrderPrice(order.securityId :
order.orderSide);
        order.orderType = '1';  // 转为限价
    endif;

    // 4. 分配订单号和时间戳
    order.orderId = GenerateOrderId();
    order.entryTime = %timestamp();
    order.prioritySeq = GetNextPrioritySeq();
    order.orderStatus = '1';

    // 5. 撮合处理
    matchResult = MatchOrder(order);

    // 6. 处理撮合结果
    if matchResult.tradeCnt > 0;
        // 有成交
        ProcessTrades(matchResult);

        order.filledQty = matchResult.totalFilledQty;
        order.remainingQty = order.quantity - order.filledQty;

        if order.remainingQty = 0;
            order.orderStatus = '3';  // 全部成交
        else;
            order.orderStatus = '2';  // 部分成交
            // 剩余订单加入订单薄
            InsertToOrderBook(order);
        endif;
    else;
        // 无成交，直接加入订单薄
        order.remainingQty = order.quantity;
        InsertToOrderBook(order);
    endif;

    // 7. 保存订单
    SaveOrder(order);

    // 8. 组装响应
    resp.rtnCode = '00';
    resp.rtnMsg = '订单已受理';
    resp.orderId = order.orderId;
```

```
    resp.orderStatus = order.orderStatus;
    resp.filledQty = order.filledQty;
    resp.remainingQty = order.remainingQty;

    return resp;
end-proc;

// ============================================================
// 撮合算法核心
// ============================================================
dcl-proc MatchOrder;
    dcl-pi *n likeds(MatchResult);
        inOrder     likeds(Order) const;
    end-pi;

    dcl-ds result likeds(MatchResult);
    dcl-ds oppositeOrder likeds(Order);
    dcl-ds trade likeds(TradeRecord);
    dcl-s continueMatch ind inz(*on);

    // 确定对手方向
    dcl-s oppositeSide char(1);
    if inOrder.orderSide = 'B';
        oppositeSide = 'S';
    else;
        oppositeSide = 'B';
    endif;

    dow continueMatch and result.totalFilledQty < inOrder.quantity;
        // 取最优对手订单
        oppositeOrder = GetBestOppositeOrder(inOrder.securityId :
oppositeSide);

        if oppositeOrder.orderId = '';
            // 无对手订单
            leave;
        endif;

        // 检查价格是否可以成交
        if not CanMatch(inOrder : oppositeOrder);
            leave;
        endif;

        // 计算成交数量
        dcl-s tradeQty packed(15:0);
        tradeQty = %min(inOrder.quantity - result.totalFilledQty :
```

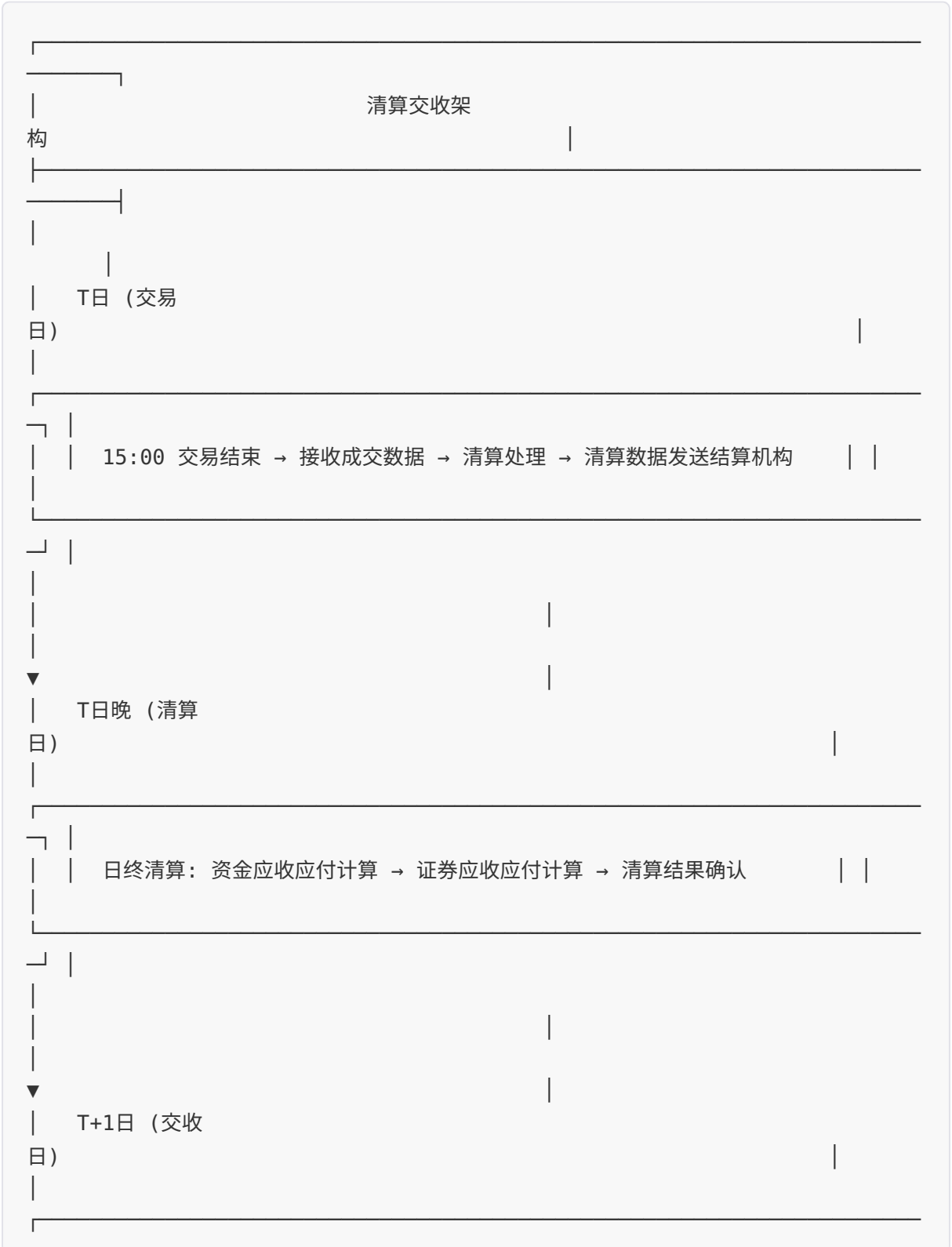
```
                        oppositeOrder.remainingQty);

        // 确定成交价格
        dcl-s tradePrice packed(15:4);
        tradePrice = DetermineTradePrice(inOrder : oppositeOrder);

        // 生成成交记录
        trade.tradeId = GenerateTradeId();
        trade.tradeTime = %timestamp();
        trade.securityId = inOrder.securityId;
        trade.tradePrice = tradePrice;
        trade.tradeQty = tradeQty;

        if inOrder.orderSide = 'B';
            trade.buyOrderId = inOrder.orderId;
            trade.buyCustId = inOrder.custId;
            trade.sellOrderId = oppositeOrder.orderId;
            trade.sellCustId = oppositeOrder.custId;
        else;
            trade.sellOrderId = inOrder.orderId;
            trade.sellCustId = inOrder.custId;
            trade.buyOrderId = oppositeOrder.orderId;
            trade.buyCustId = oppositeOrder.custId;
        endif;

        // 添加到结果
        result.tradeCnt = result.tradeCnt + 1;
        result.trades(result.tradeCnt) = trade;
        result.totalFilledQty = result.totalFilledQty + tradeQty;

        // 更新对手订单
        UpdateOppositeOrder(oppositeOrder.orderId : tradeQty);

        // 如果对手订单已完全成交，从订单薄移除
        if oppositeOrder.remainingQty = tradeQty;
            RemoveFromOrderBook(oppositeOrder.orderId);
        endif;

        // 检查是否需要继续撮合
        if inOrder.orderType = '1' and result.totalFilledQty >=
inOrder.quantity;
            continueMatch = *off;
        endif;
    enddo;

    return result;
```

```
end-proc;

// ====================================================================
// 判断是否可以成交
// ====================================================================
dcl-proc CanMatch;
    dcl-pi *n ind;
        order1      likeds(Order) const;
        order2      likeds(Order) const;
    end-pi;

    // 买单价格 >= 卖单价格时可以成交
    if order1.orderSide = 'B' and order2.orderSide = 'S';
        return (order1.price >= order2.price);
    elseif order1.orderSide = 'S' and order2.orderSide = 'B';
        return (order2.price >= order1.price);
    endif;

    return *off;
end-proc;

// ====================================================================
// 确定成交价格
// ====================================================================
dcl-proc DetermineTradePrice;
    dcl-pi *n packed(15:4);
        buyOrder    likeds(Order) const;
        sellOrder   likeds(Order) const;
    end-pi;

    // 价格优先原则:
    // 如果被动订单(先进入订单薄的订单)是买单，使用卖单价格
    // 如果被动订单是卖单，使用买单价格

    if buyOrder.entryTime < sellOrder.entryTime;
        // 买单先进入，使用被动价格(买价)
        return buyOrder.price;
    else;
        // 卖单先进入，使用被动价格(卖价)
        return sellOrder.price;
    endif;
end-proc;
```

4.2 清算交收系统

4.2.1 清算交收架构

中国A股市场采用T+1交收制度，即交易发生后的下一个交易日完成资金和证券的交收。



```
┐ │
│  │  9:00 资金交收 → 证券交收 → 交收结果反馈 → 投资者账户更新            │ │
│
└──────────────────────────────────────────────────────────────────────────
┘ │
│
      │
└──────────────────────────────────────────────────────────────────────────
──────┘
```

4.2.2 清算处理

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('CLEARING');

// ====================================================================
// 程序: CLEARING
// 描述: 证券清算处理
// ====================================================================

// 成交明细
dcl-ds TradeDetail qualified template;
    tradeId         char(20);
    tradeDate       packed(8);
    tradeTime       packed(6);
    securityId      char(12);

    buyCustId       char(20);
    buyAcctId       char(20);
    buyQty          packed(15:0);

    sellCustId      char(20);
    sellAcctId      char(20);
    sellQty         packed(15:0);

    tradePrice      packed(15:4);
    tradeAmt        packed(19:2);

    tradeFee        packed(19:2);
    stampTax        packed(19:2);
    transferFee     packed(19:2);
    clearingFee     packed(19:2);
end-ds;
```

```
// 清算结果
dcl-ds ClearingResult qualified template;
    custId          char(20);
    acctId          char(20);
    securityId      char(12);

    clearDate       packed(8);

    buyQty          packed(15:0);
    buyAmt          packed(19:2);
    sellQty         packed(15:0);
    sellAmt         packed(19:2);

    netQty          packed(15:0);
    netAmt          packed(19:2);

    receivableAmt   packed(19:2);
    payableAmt      packed(19:2);

    totalFee        packed(19:2);
end-ds;

// ====================================================================
// 日终清算主程序
// ====================================================================
dcl-proc DailyClearing export;
    dcl-pi *n;
        tradeDate   packed(8) const;
        outRtnCode  char(2);
        outRtnMsg   char(200);
    end-pi;

    dcl-s tradeCnt packed(10);
    dcl-s clearCnt packed(10);

    outRtnCode = '00';
    outRtnMsg = '';

    // 1. 检查清算状态
    if CheckClearingStatus(tradeDate);
        outRtnCode = '01';
        outRtnMsg = '当日已清算';
        return;
    endif;
```

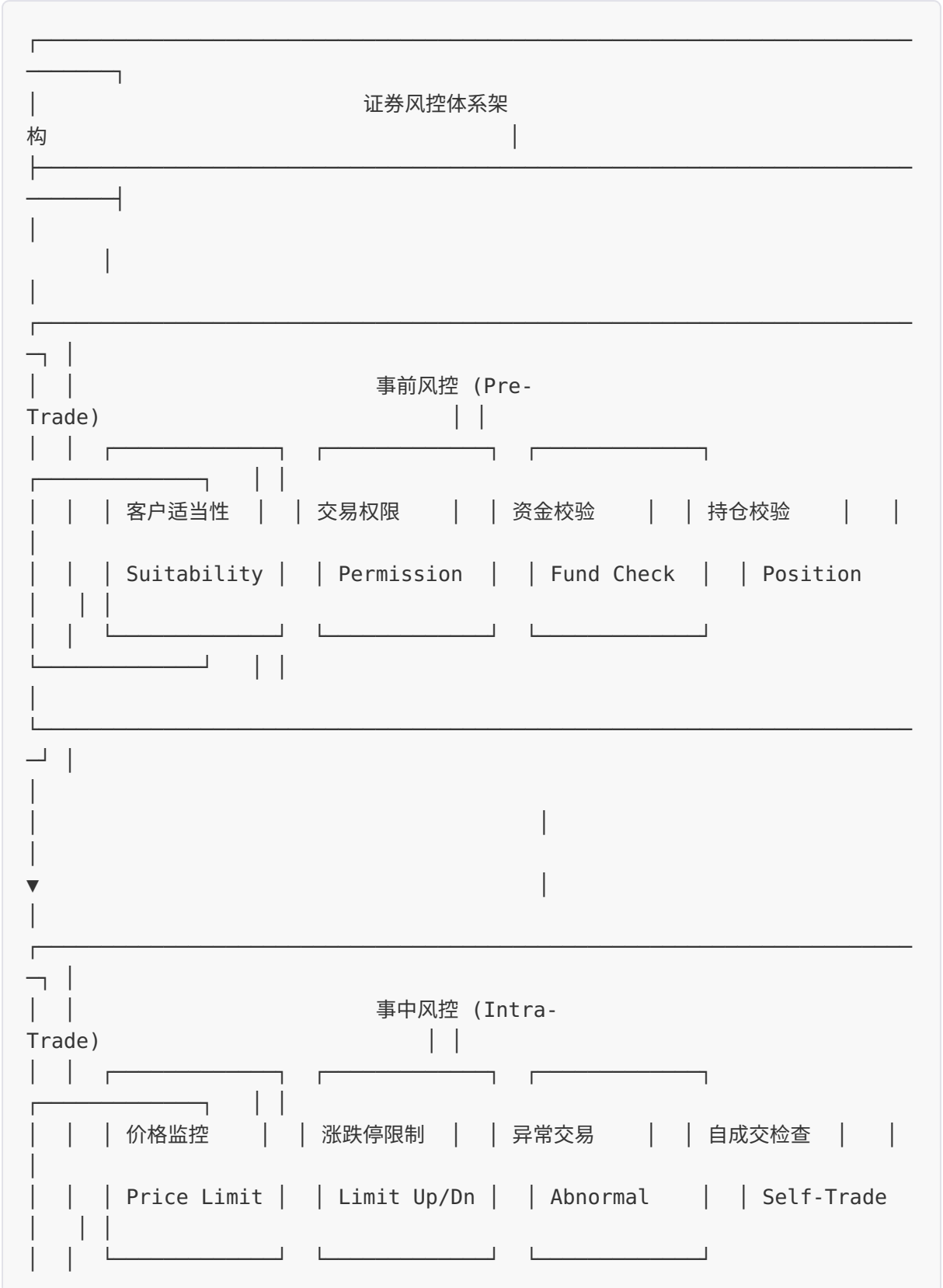
```
    // 2. 读取当日成交数据
    exec SQL
        SELECT COUNT(*) INTO :tradeCnt
        FROM TRADEDETAIL
        WHERE TRADE_DATE = :tradeDate;

    if tradeCnt = 0;
        outRtnCode = '02';
        outRtnMsg = '当日无成交数据';
        return;
    endif;

    // 3. 逐笔计算费用
    CalculateTradeFees(tradeDate);

    // 4. 按客户/证券汇总清算数据
    clearCnt = AggregateClearingData(tradeDate);

    // 5. 生成清算文件发送结算机构
    GenerateClearingFile(tradeDate);

    // 6. 更新清算状态
    UpdateClearingStatus(tradeDate : '1');

    outRtnMsg = '清算完成: 成交笔数=' + %char(tradeCnt) +
                ', 清算笔数=' + %char(clearCnt);

end-proc;

// ====================================================================
// 计算交易费用
// ====================================================================
dcl-proc CalculateTradeFees;
    dcl-pi *n;
        tradeDate   packed(8) const;
    end-pi;

    // 费用标准(示例)
    // 佣金: 不超过成交金额的0.3%，最低5元
    // 印花税: 卖出方缴纳，成交金额的0.1%
    // 过户费: 成交金额的0.001%
    // 清算费: 成交金额的0.005%

    exec SQL
        UPDATE TRADEDETAIL
        SET
```

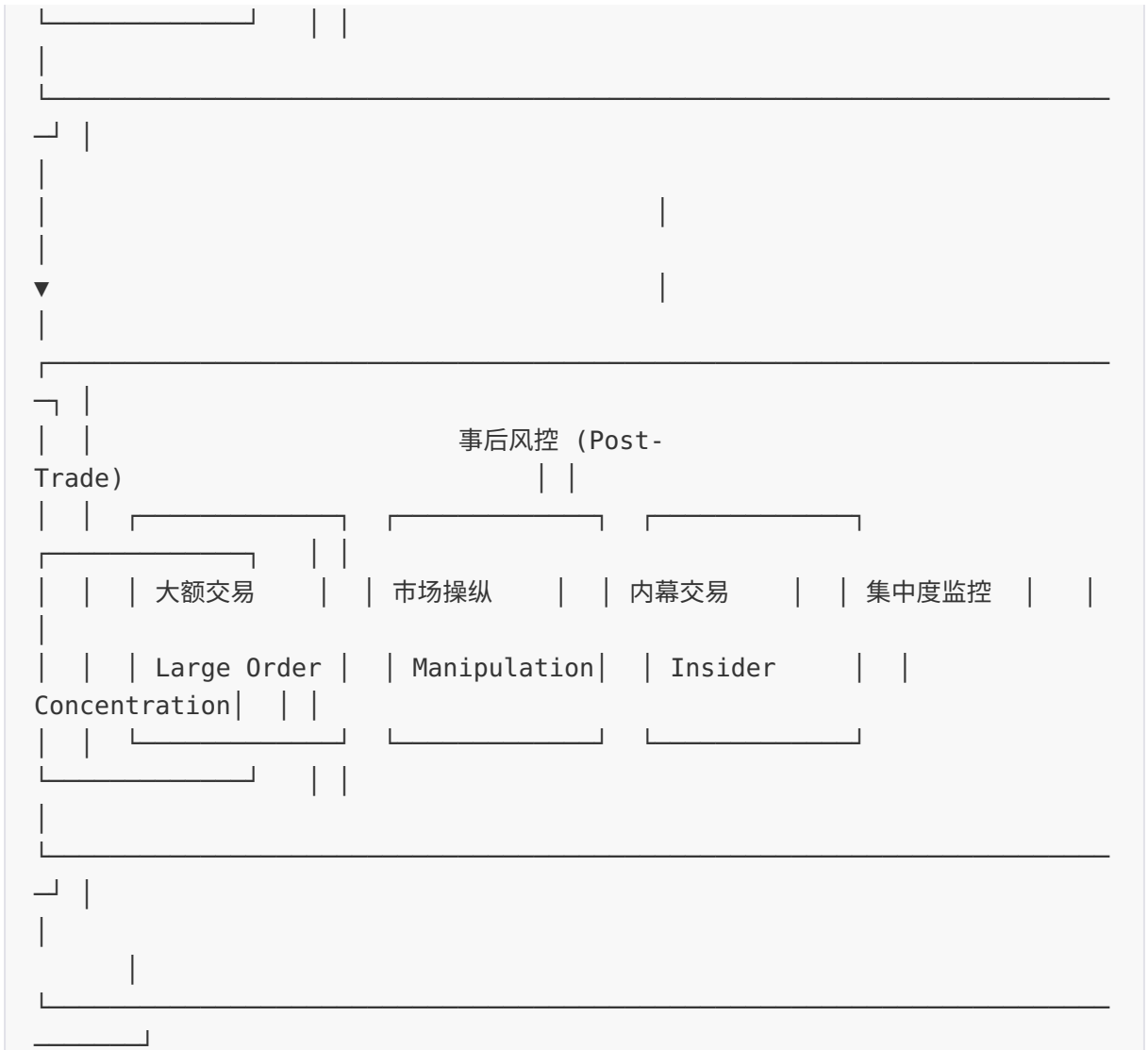
```
            -- 佣金(买卖都收)
            TRADE_FEE = CASE
                WHEN TRADE_AMT * 0.003 < 5 THEN 5
                ELSE TRADE_AMT * 0.003
            END,
            -- 印花税(仅卖出)
            STAMP_TAX = CASE
                WHEN SELL_CUST_ID IS NOT NULL THEN TRADE_AMT * 0.001
                ELSE 0
            END,
            -- 过户费
            TRANSFER_FEE = TRADE_AMT * 0.00001,
            -- 清算费
            CLEARING_FEE = TRADE_AMT * 0.00005
        WHERE TRADE_DATE = :tradeDate;

end-proc;

// ====================================================================
// 汇总清算数据
// ====================================================================
dcl-proc AggregateClearingData;
    dcl-pi *n packed(10);
        tradeDate   packed(8) const;
    end-pi;

    dcl-s clearCnt packed(10);

    // 买入方清算数据
    exec SQL
        INSERT INTO CLEARINGRESULT (
            CUST_ID, ACCT_ID, SECURITY_ID, CLEAR_DATE,
            BUY_QTY, BUY_AMT, SELL_QTY, SELL_AMT,
            NET_QTY, NET_AMT, RECEIVABLE_AMT, PAYABLE_AMT, TOTAL_FEE
        )
        SELECT
            BUY_CUST_ID,
            BUY_ACCT_ID,
            SECURITY_ID,
            :tradeDate,
            SUM(BUY_QTY),
            SUM(TRADE_AMT),
            0,
            0,
            SUM(BUY_QTY),
            SUM(TRADE_AMT + TRADE_FEE + TRANSFER_FEE + CLEARING_FEE),
```

```
            0,
            SUM(TRADE_AMT + TRADE_FEE + TRANSFER_FEE + CLEARING_FEE),
            SUM(TRADE_FEE + TRANSFER_FEE + CLEARING_FEE)
        FROM TRADEDETAIL
        WHERE TRADE_DATE = :tradeDate
        GROUP BY BUY_CUST_ID, BUY_ACCT_ID, SECURITY_ID;

    GET DIAGNOSTICS clearCnt = ROW_COUNT;

    // 卖出方清算数据
    exec SQL
        INSERT INTO CLEARINGRESULT (
            CUST_ID, ACCT_ID, SECURITY_ID, CLEAR_DATE,
            BUY_QTY, BUY_AMT, SELL_QTY, SELL_AMT,
            NET_QTY, NET_AMT, RECEIVABLE_AMT, PAYABLE_AMT, TOTAL_FEE
        )
        SELECT
            SELL_CUST_ID,
            SELL_ACCT_ID,
            SECURITY_ID,
            :tradeDate,
            0,
            0,
            SUM(SELL_QTY),
            SUM(TRADE_AMT),
            -SUM(SELL_QTY),
            -SUM(TRADE_AMT - TRADE_FEE - STAMP_TAX - TRANSFER_FEE -
CLEARING_FEE),
            SUM(TRADE_AMT - TRADE_FEE - STAMP_TAX - TRANSFER_FEE -
CLEARING_FEE),
            0,
            SUM(TRADE_FEE + STAMP_TAX + TRANSFER_FEE + CLEARING_FEE)
        FROM TRADEDETAIL
        WHERE TRADE_DATE = :tradeDate
        GROUP BY SELL_CUST_ID, SELL_ACCT_ID, SECURITY_ID;

    return clearCnt;
end-proc;
```

4.3 风险控制系统

4.3.1 风控体系架构





4.3.2 实时风控检查

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('RISK');

// ============================================================
// 程序: RISKCTRL
// 描述: 交易实时风控检查
// ============================================================

// 风控检查结果
dcl-ds RiskCheckResult qualified template;
    passed          ind;
    riskLevel       char(1);        // L-低 M-中 H-高
```

```
    rejectReason    varchar(200);
    ruleCode        char(10);
    warningMsg      varchar(500);
end-ds;

// ===================================================================
// 综合风控检查
// ===================================================================
dcl-proc ComprehensiveRiskCheck export;
    dcl-pi *n likeds(RiskCheckResult);
        order       likeds(Order) const;
    end-pi;

    dcl-ds result likeds(RiskCheckResult);
    result.passed = *on;
    result.riskLevel = 'L';

    // 1. 客户适当性检查
    result = CheckCustomerSuitability(order);
    if not result.passed;
        return result;
    endif;

    // 2. 交易权限检查
    result = CheckTradingPermission(order);
    if not result.passed;
        return result;
    endif;

    // 3. 资金检查
    result = CheckFundAvailability(order);
    if not result.passed;
        return result;
    endif;

    // 4. 持仓检查
    result = CheckPositionLimit(order);
    if not result.passed;
        return result;
    endif;

    // 5. 价格检查(涨跌停)
    result = CheckPriceLimit(order);
    if not result.passed;
        return result;
    endif;
```

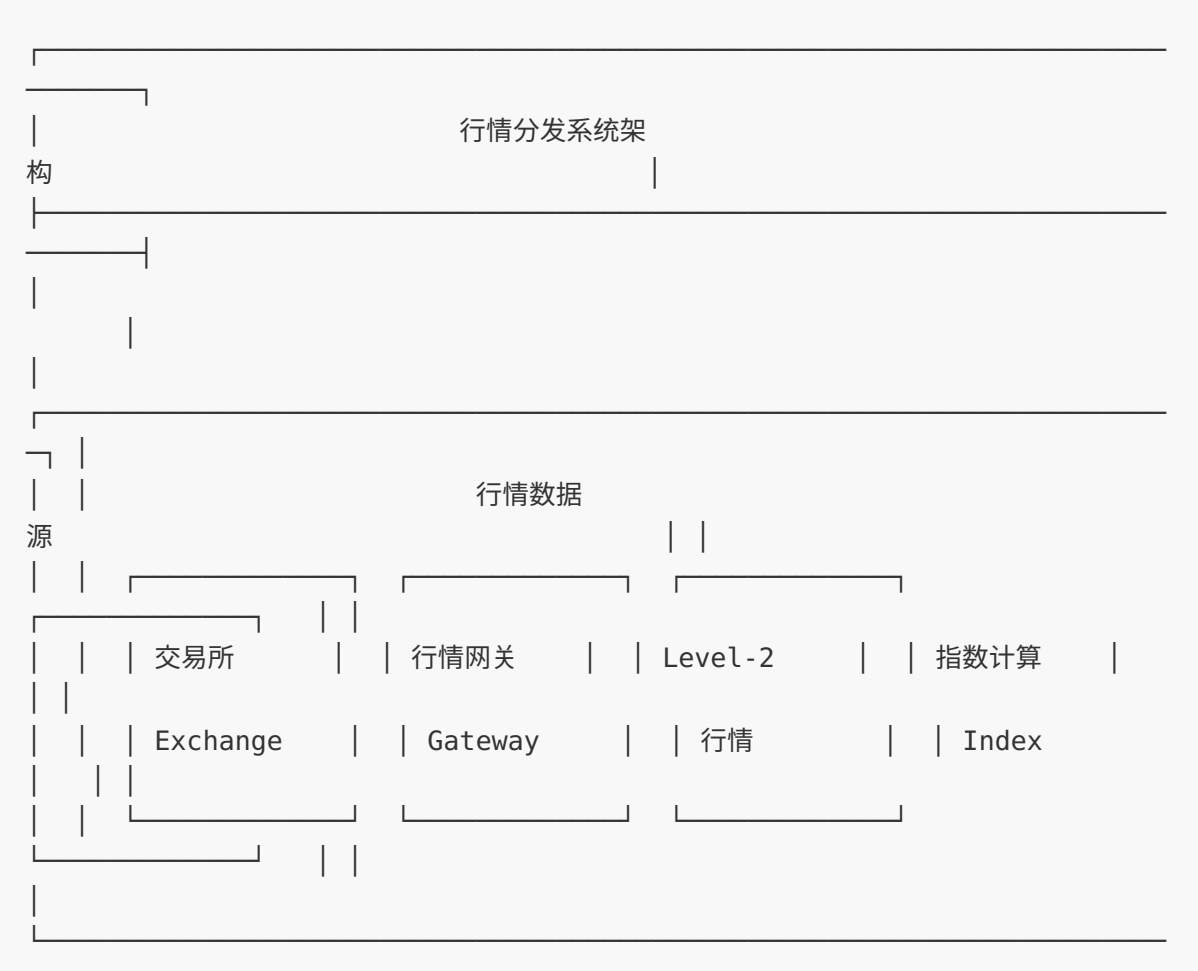
```
    // 6. 异常交易检查
    result = CheckAbnormalTrading(order);
    if result.riskLevel = 'H';
        result.passed = *off;
        return result;
    endif;

    return result;
end-proc;

// ====================================================================
// 价格涨跌停检查
// ====================================================================
dcl-proc CheckPriceLimit;
    dcl-pi *n likeds(RiskCheckResult);
        order       likeds(Order) const;
    end-pi;

    dcl-ds result likeds(RiskCheckResult);
    dcl-s preClosePrice packed(15:4);
    dcl-s upLimitPrice packed(15:4);
    dcl-s downLimitPrice packed(15:4);

    result.passed = *on;

    // 查询昨日收盘价
    exec SQL
        SELECT PRE_CLOSE_PRICE INTO :preClosePrice
        FROM SECURITYINFO
        WHERE SECURITY_ID = :order.securityId;

    if preClosePrice = 0;
        return result;  // 新股上市首日不做限制
    endif;

    // 计算涨跌停价格(普通股票10%，ST股票5%，创业板20%)
    dcl-s limitRate packed(5:2);
    exec SQL
        SELECT CASE SECURITY_TYPE
            WHEN 'ST' THEN 0.05
            WHEN 'GEM' THEN 0.20
            ELSE 0.10
        END INTO :limitRate
        FROM SECURITYINFO
        WHERE SECURITY_ID = :order.securityId;
```

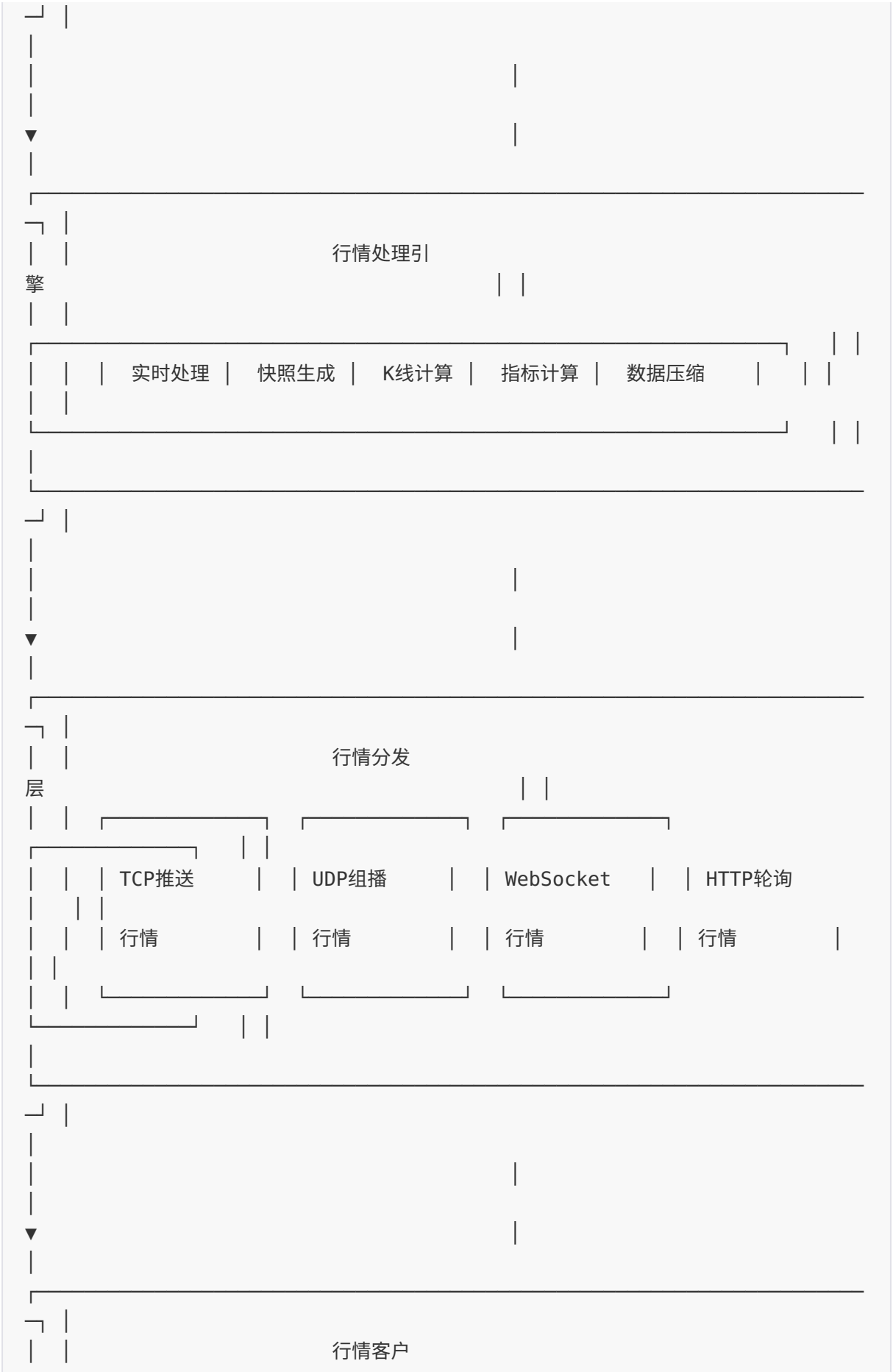
```
    upLimitPrice = preClosePrice * (1 + limitRate);
    downLimitPrice = preClosePrice * (1 - limitRate);

    // 检查价格
    if order.price > upLimitPrice;
        result.passed = *off;
        result.rejectReason = '委托价格超过涨停价';
        result.ruleCode = 'PRICE_UP_LIMIT';
    elseif order.price < downLimitPrice;
        result.passed = *off;
        result.rejectReason = '委托价格低于跌停价';
        result.ruleCode = 'PRICE_DOWN_LIMIT';
    endif;

    return result;
end-proc;

// ============================================================
// 异常交易监控
// ============================================================
dcl-proc CheckAbnormalTrading;
    dcl-pi *n likeds(RiskCheckResult);
        order       likeds(Order) const;
    end-pi;

    dcl-ds result likeds(RiskCheckResult);
    dcl-s recentOrderCnt packed(5);
    dcl-s cancelRate packed(5:2);

    result.passed = *on;
    result.riskLevel = 'L';

    // 检查频繁报撤单
    exec SQL
        SELECT COUNT(*) INTO :recentOrderCnt
        FROM ORDERHIST
        WHERE CUST_ID = :order.custId
          AND SECURITY_ID = :order.securityId
          AND ENTRY_TIME > CURRENT TIMESTAMP - 5 MINUTES;

    if recentOrderCnt > 50;
        result.riskLevel = 'M';
        result.warningMsg = '报撤单频率异常';
    endif;
```

```
    // 检查大额委托
    dcl-s marketCap packed(19:2);
    exec SQL
        SELECT TOTAL_MARKET_VALUE INTO :marketCap
        FROM SECURITYINFO
        WHERE SECURITY_ID = :order.securityId;

    if order.quantity * order.price > marketCap * 0.01;  // 超过流通市值
1%
        result.riskLevel = 'H';
        result.warningMsg = '大额委托，触发监控';
    endif;

    return result;
end-proc;
```

4.4 行情分发系统

4.4.1 行情系统架构




行情处理引
擎
实时处理
快照生成
K线计算
指标计算
数据压缩
行情分发
层
TCP推送
行情
UDP组播
行情
WebSocket
行情
HTTP轮询
行情
行情客户

```
端                                                  | |
|  |  ┌──────────────┐   ┌──────────────┐   ┌──────────────┐
┌──────────────┐    | |
|  |  | 交易终端     |   | 行情软件     |   | 量化系统     |   | 手机APP      |    |
|
|  |  └──────────────┘   └──────────────┘   └──────────────┘
└──────────────┘    | |
|
└──────────────────────────────────────────────────────────────────────────────
┘ |
|
      |
└──────────────────────────────────────────────────────────────────────────────
──────┘
```

4.4.2 行情数据处理

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('MARKETDATA');

// ============================================================
// 程序: MKTDATAPRC
// 描述: 行情数据处理
// ============================================================

// 行情数据结构
dcl-ds MarketData qualified template;
    securityId      char(12);
    dataTime        timestamp;

    preClosePrice   packed(15:4);
    openPrice       packed(15:4);
    highPrice       packed(15:4);
    lowPrice        packed(15:4);
    lastPrice       packed(15:4);

    bidPrice        packed(15:4) dim(5);
    bidVolume       packed(15:0) dim(5);
    askPrice        packed(15:4) dim(5);
    askVolume       packed(15:0) dim(5);

    volume          packed(18:0);
    turnover        packed(19:4);
```

```
    // Level-2特有
    totalBidVolume  packed(18:0);
    totalAskVolume  packed(18:0);
    weightedAvgBid  packed(15:4);
    weightedAvgAsk  packed(15:4);
end-ds;

// K线数据结构
dcl-ds KLineData qualified template;
    securityId      char(12);
    periodType      char(2);        // 01-1分 02-5分 03-15分 11-日 12-
周...
    klineTime       timestamp;

    openPrice       packed(15:4);
    highPrice       packed(15:4);
    lowPrice        packed(15:4);
    closePrice      packed(15:4);
    volume          packed(18:0);
    turnover        packed(19:4);
end-ds;

// ================================================================
// 处理实时行情
// ================================================================
dcl-proc ProcessMarketData export;
    dcl-pi *n;
        mktData     likeds(MarketData) const;
    end-pi;

    // 1. 更新实时行情表
    UpdateRealtimeQuote(mktData);

    // 2. 更新买卖队列
    UpdateOrderQueue(mktData);

    // 3. 更新K线数据
    UpdateKLine(mktData);

    // 4. 计算技术指标
    CalculateIndicators(mktData.securityId);

    // 5. 推送行情到客户端
    BroadcastMarketData(mktData);
```

```
end-proc;

// ============================================================
// 更新K线数据
// ============================================================
dcl-proc UpdateKLine;
    dcl-pi *n;
        mktData     likeds(MarketData) const;
    end-pi;

    dcl-s periodTypes char(2) dim(9);
    dcl-s i packed(3);
    dcl-s klineTime timestamp;

    periodTypes(1) = '01';  // 1分钟
    periodTypes(2) = '02';  // 5分钟
    periodTypes(3) = '03';  // 15分钟
    periodTypes(4) = '04';  // 30分钟
    periodTypes(5) = '05';  // 60分钟
    periodTypes(6) = '11';  // 日线
    periodTypes(7) = '12';  // 周线
    periodTypes(8) = '13';  // 月线
    periodTypes(9) = '14';  // 年线

    for i = 1 to 9;
        // 计算K线时间
        klineTime = CalculateKLineTime(mktData.dataTime :
periodTypes(i));

        // 更新或插入K线
        exec SQL
            MERGE INTO KLINE AS target
            USING (VALUES(
                :mktData.securityId,
                :periodTypes(i),
                :klineTime
            )) AS source (SECURITY_ID, PERIOD_TYPE, KLINE_TIME)
            ON target.SECURITY_ID = source.SECURITY_ID
               AND target.PERIOD_TYPE = source.PERIOD_TYPE
               AND target.KLINE_TIME = source.KLINE_TIME
            WHEN MATCHED THEN
                UPDATE SET
                    HIGH_PRICE = CASE
                        WHEN :mktData.lastPrice > target.HIGH_PRICE
                        THEN :mktData.lastPrice
                        ELSE target.HIGH_PRICE END,
```

```
                    LOW_PRICE = CASE
                        WHEN :mktData.lastPrice < target.LOW_PRICE
                        THEN :mktData.lastPrice
                        ELSE target.LOW_PRICE END,
                    CLOSE_PRICE = :mktData.lastPrice,
                    VOLUME = target.VOLUME + :mktData.volume,
                    TURNOVER = target.TURNOVER + :mktData.turnover
            WHEN NOT MATCHED THEN
                INSERT (SECURITY_ID, PERIOD_TYPE, KLINE_TIME,
                        OPEN_PRICE, HIGH_PRICE, LOW_PRICE, CLOSE_PRICE,
                        VOLUME, TURNOVER)
                VALUES
(:mktData.securityId, :periodTypes(i), :klineTime,
                        :mktData.lastPrice, :mktData.lastPrice,
                        :mktData.lastPrice, :mktData.lastPrice,
                        :mktData.volume, :mktData.turnover);
    endfor;

end-proc;

// ====================================================================
// 计算技术指标
// ====================================================================
dcl-proc CalculateIndicators;
    dcl-pi *n;
        securityId  char(12) const;
    end-pi;

    // 计算MA5, MA10, MA20, MA60
    exec SQL
        UPDATE INDICATOR_MA
        SET MA5 = (SELECT AVG(CLOSE_PRICE) FROM (
            SELECT CLOSE_PRICE FROM KLINE
            WHERE SECURITY_ID = :securityId AND PERIOD_TYPE = '11'
            ORDER BY KLINE_TIME DESC FETCH FIRST 5 ROWS ONLY) T),
            MA10 = (SELECT AVG(CLOSE_PRICE) FROM (
            SELECT CLOSE_PRICE FROM KLINE
            WHERE SECURITY_ID = :securityId AND PERIOD_TYPE = '11'
            ORDER BY KLINE_TIME DESC FETCH FIRST 10 ROWS ONLY) T),
            MA20 = (SELECT AVG(CLOSE_PRICE) FROM (
            SELECT CLOSE_PRICE FROM KLINE
            WHERE SECURITY_ID = :securityId AND PERIOD_TYPE = '11'
            ORDER BY KLINE_TIME DESC FETCH FIRST 20 ROWS ONLY) T)
        WHERE SECURITY_ID = :securityId;

    // 计算MACD, KDJ, RSI等其他指标...
```

```
end-proc;
```

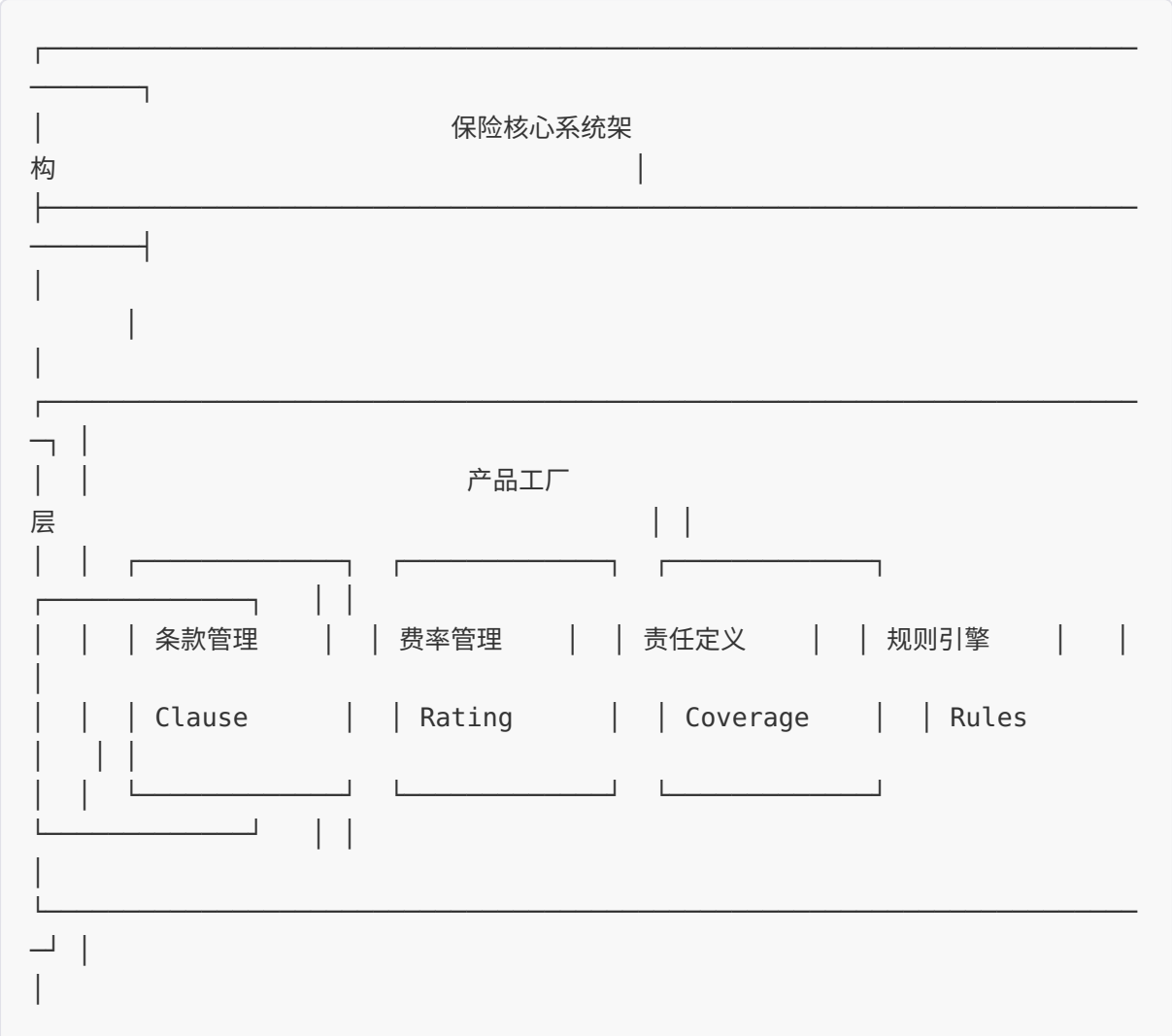
由于篇幅限制，我将继续追加文档的剩余部分。让我继续添加保险系统、风控、安全合规等章节的内容。

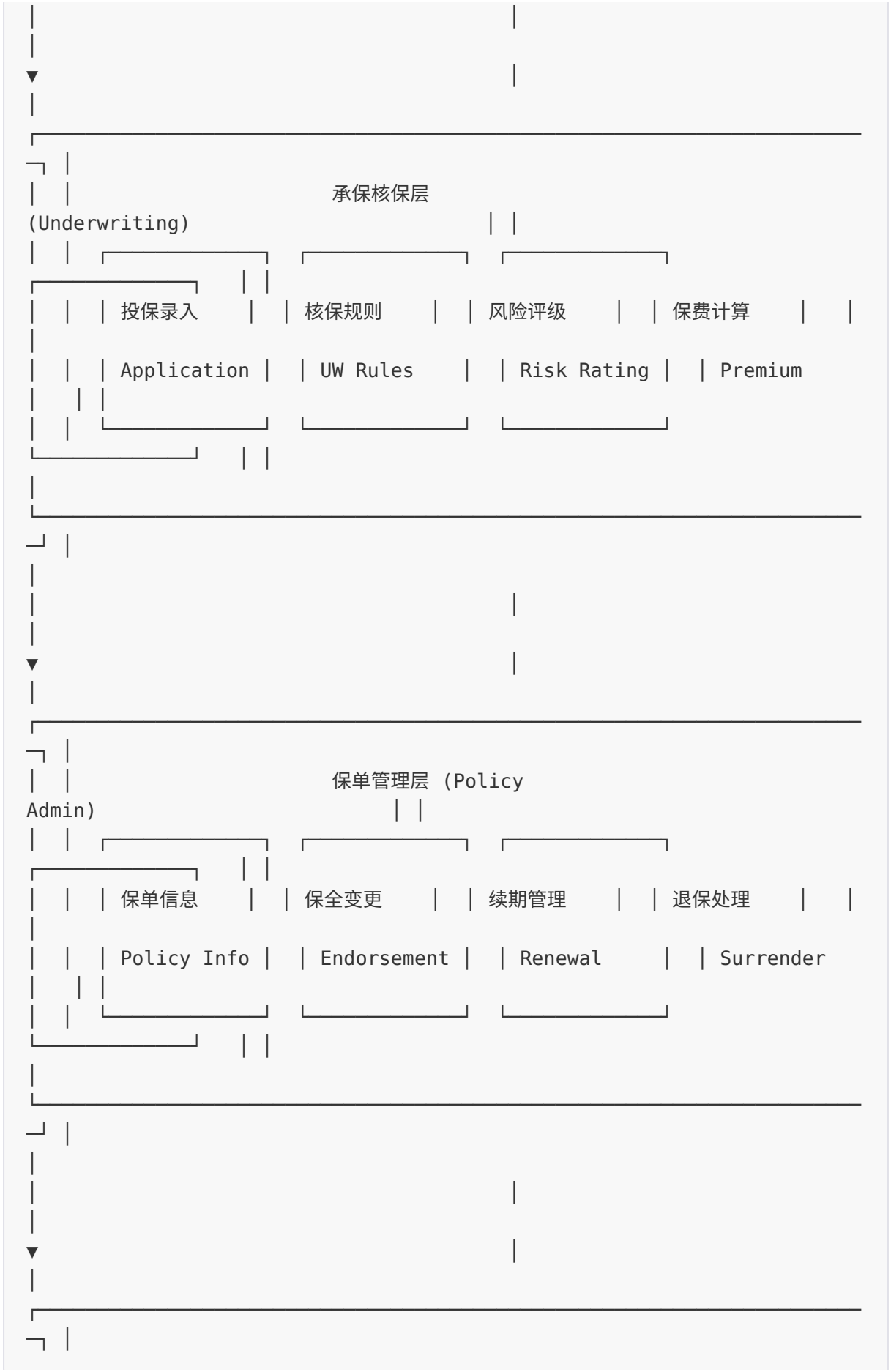
第五章 保险核心系统

5.1 保单管理系统

5.1.1 保单系统架构

保险核心系统是保险公司业务运营的中枢，涵盖产品定义、承保、保全、理赔等全生命周期管理。




承保核保层
(Underwriting)
投保录入
Application
核保规则
UW Rules
风险评级
Risk Rating
保费计算
Premium
保单管理层 (Policy
Admin)
保单信息
Policy Info
保全变更
Endorsement
续期管理
Renewal
退保处理
Surrender

```
|  |                          理赔服务层
(Claims)                                  | |
|  |  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐
┌──────────────┐   | |
|  |  | 报案登记     |  | 查勘定损     |  | 理算核赔     |  | 赔付支付     |   |
|
|  |  | FNOL         |  | Survey       |  | Adjustment   |  | Payment
|   | |
|  |  └──────────────┘  └──────────────┘  └──────────────┘
└──────────────┘   | |
|
└──────────────────────────────────────────────────────────────────────────
┘ |
|
      |
└──────────────────────────────────────────────────────────────────────────
─────┘
```

5.1.2 保单主表设计

```
-- ========================================================================
-- 保单主表 - POLICYMAST
-- ========================================================================

CREATE TABLE POLICYMAST (
    -- 主键
    POLICY_NO           CHAR(20)        NOT NULL,

    -- 产品信息
    PROD_CD             CHAR(10)        NOT NULL,
    PROD_NAME           VARCHAR(100),
    PROD_TYPE           CHAR(2)         NOT NULL,  -- 01-寿险 02-健康险
03-意外险...

    -- 投保人信息
    APPNT_ID            CHAR(20)        NOT NULL,
    APPNT_NAME          VARCHAR(100),
    APPNT_ID_TYPE       CHAR(2),
    APPNT_ID_NO         VARCHAR(50),
    APPNT_BIRTH_DT      DECIMAL(8, 0),
    APPNT_GENDER        CHAR(1),
    APPNT_OCCUP_CD      CHAR(4),
    APPNT_MOBILE        VARCHAR(20),
    APPNT_EMAIL         VARCHAR(100),
```

```
    APPNT_ADDR          VARCHAR(500),

    -- 被保人信息
    INSURED_ID          CHAR(20)        NOT NULL,
    INSURED_NAME        VARCHAR(100),
    INSURED_ID_TYPE     CHAR(2),
    INSURED_ID_NO       VARCHAR(50),
    INSURED_BIRTH_DT    DECIMAL(8, 0),
    INSURED_GENDER      CHAR(1),
    INSURED_OCCUP_CD    CHAR(4),
    INSURED_AGE         DECIMAL(3, 0),
    RELATION_TO_APPNT   CHAR(2),                    -- 01-本人 02-配偶...

    -- 受益人信息
    BENEF_TYPE          CHAR(1)         DEFAULT '1', -- 1-法定 2-指定

    -- 保险期间
    POLICY_TERM         DECIMAL(3, 0),              -- 保险期间(年)
    PREM_TERM           DECIMAL(3, 0),              -- 缴费期间(年)
    INSURED_TERM_TYPE   CHAR(1),                    -- Y-年 M-月 D-日 A-终
身

    -- 保额保费
    SUM_ASSURED         DECIMAL(19, 2)  NOT NULL,
    PREM_MODE           CHAR(1)         NOT NULL,  -- 1-趸交 2-年交 3-半
年...
    BASE_PREM           DECIMAL(19, 2)  NOT NULL,
    EXTRA_PREM          DECIMAL(19, 2)  DEFAULT 0, -- 加费
    TOTAL_PREM          DECIMAL(19, 2),

    -- 保单状态
    POLICY_STAT         CHAR(2)         DEFAULT '10', -- 10-生效 20-中止
30-终止...
    STAT_CHG_DT         DECIMAL(8, 0),

    -- 重要日期
    PROPOSAL_DT         DECIMAL(8, 0)   NOT NULL,  -- 投保日期
    EFFECT_DT           DECIMAL(8, 0)   NOT NULL,  -- 生效日期
    FIRST_PREM_DT       DECIMAL(8, 0),              -- 首期缴费日
    NEXT_PREM_DT        DECIMAL(8, 0),              -- 下次缴费日
    PAID_UP_DT          DECIMAL(8, 0),              -- 缴清日期
    MATURITY_DT         DECIMAL(8, 0),              -- 满期日期

    -- 续期信息
    RENEWAL_CNT         DECIMAL(3, 0)   DEFAULT 0,
```

```
    LAPSE_DT            DECIMAL(8, 0),              -- 失效日期
    REINST_DT           DECIMAL(8, 0),              -- 复效日期

    -- 现金价值
    CASH_VALUE          DECIMAL(19, 2)  DEFAULT 0,
    PAID_PREM           DECIMAL(19, 2)  DEFAULT 0, -- 已缴保费

    -- 渠道信息
    CHNL_TYPE           CHAR(2),                    -- 01-个险 02-银保
03-团险...
    AGENT_ID            CHAR(10),
    BRANCH_ID           CHAR(10)        NOT NULL,

    -- 收付费账户
    PREM_ACCT_NO        CHAR(32),
    PREM_ACCT_NAME      VARCHAR(100),
    PREM_BANK_CD        CHAR(12),

    -- 特别约定
    SPECIAL_PROV        CLOB(32K),

    -- 健康告知
    DISCLOSURE_FLAG     CHAR(1)         DEFAULT '0',

    -- 承保信息
    UW_DECISION         CHAR(1),                    -- A-标准体 S-加费 E-除
外...
    UW_REASON           VARCHAR(500),
    UW_USER_ID          CHAR(10),
    UW_DT               DECIMAL(8, 0),

    -- 审计字段
    CREATE_TS           TIMESTAMP       DEFAULT CURRENT_TIMESTAMP,
    CREATE_USER         VARCHAR(20)     DEFAULT USER,
    UPDATE_TS           TIMESTAMP       DEFAULT CURRENT_TIMESTAMP,
    UPDATE_USER         VARCHAR(20)     DEFAULT USER,

    CONSTRAINT PK_POLICYMAST PRIMARY KEY (POLICY_NO),
    CONSTRAINT FK_POLICY_APPNT FOREIGN KEY (APPNT_ID)
        REFERENCES CUSTMAST(CUST_ID),
    CONSTRAINT FK_POLICY_INSURED FOREIGN KEY (INSURED_ID)
        REFERENCES CUSTMAST(CUST_ID)

) RCDFMT POLICYMASTR;

-- 保单责任表
```

```
CREATE TABLE POLICYCOVER (
    POLICY_NO           CHAR(20)        NOT NULL,
    COVER_SEQ           DECIMAL(3, 0)   NOT NULL,

    COVER_CD            CHAR(10)        NOT NULL,
    COVER_NAME          VARCHAR(100),
    COVER_TYPE          CHAR(2),                    -- 01-主险 02-附加险

    SUM_ASSURED         DECIMAL(19, 2)  NOT NULL,
    PREM                DECIMAL(19, 2)  NOT NULL,

    EFFECT_DT           DECIMAL(8, 0)   NOT NULL,
    EXPIRY_DT           DECIMAL(8, 0),

    WAITING_DAYS        DECIMAL(3, 0),              -- 等待期(天)
    DEDUCTIBLE_AMT      DECIMAL(19, 2),             -- 免赔额
    INDEMNITY_PCT       DECIMAL(5, 2),              -- 给付比例

    COVER_STAT          CHAR(1)         DEFAULT '1', -- 1-有效 0-终止

    CONSTRAINT PK_POLICYCOVER PRIMARY KEY (POLICY_NO, COVER_SEQ),
    CONSTRAINT FK_POLCOVER_POLICY FOREIGN KEY (POLICY_NO)
        REFERENCES POLICYMAST(POLICY_NO)
);
```

5.1.3 保费计算引擎

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('INSURE');

// ============================================================
// 程序: PREMCALC
// 描述: 保费计算引擎
// 支持: 寿险、健康险、意外险保费计算
// ============================================================

// 保费计算请求结构
dcl-ds PremCalcRequest qualified template;
    prodCd          char(10);
    insuredAge      packed(3);
    insuredGender   char(1);
    insuredOccup    char(4);
```

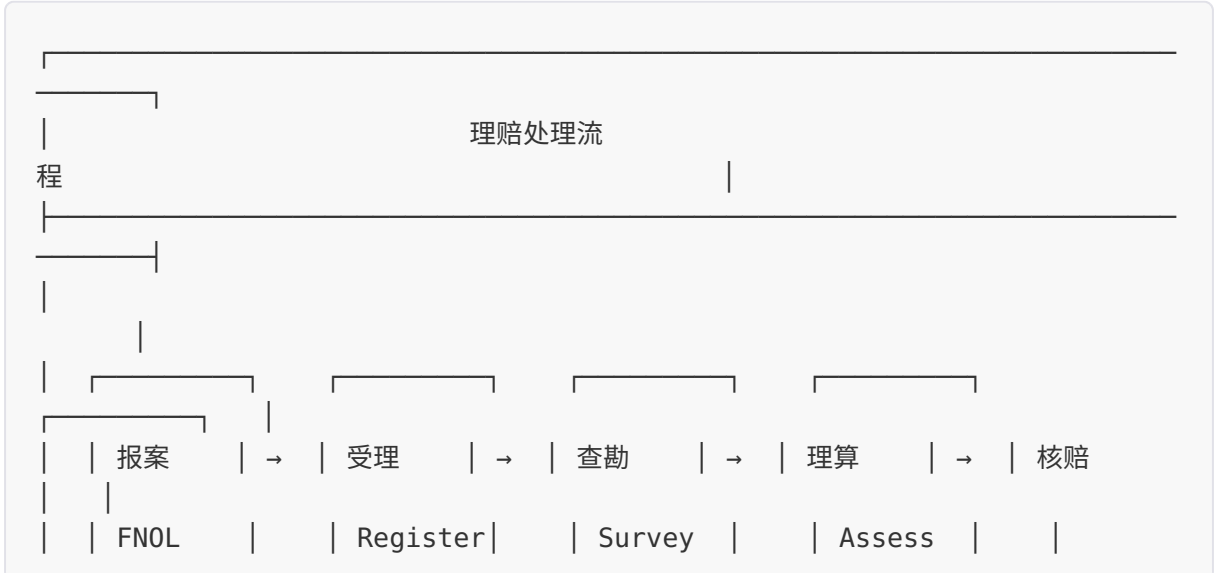
```
    policyTerm      packed(3);
    premTerm        packed(3);
    premMode        char(1);        // 缴费方式

    sumAssured      packed(19:2);

    -- 健康告知
    isSmoker        char(1);
    hasDisease      char(1);
    bmi             packed(5:2);

    -- 职业类别
    occupClass      char(1);        // 1-6类职业
end-ds;

// 保费计算结果结构
dcl-ds PremCalcResult qualified template;
    basePrem        packed(19:2);
    extraPremRate   packed(5:2);    // 加费比例
    extraPrem       packed(19:2);
    totalPrem       packed(19:2);

    riskScore       packed(5:2);
    uwDecision      char(1);        // A-标准 S-加费 E-除外...
    uwSuggestion    varchar(200);
end-ds;

// ===================================================================
// 保费计算主入口
// ===================================================================
dcl-proc CalculatePremium export;
    dcl-pi *n likeds(PremCalcResult);
        req     likeds(PremCalcRequest) const;
    end-pi;

    dcl-ds result likeds(PremCalcResult) inz;
    dcl-ds rateData likeds(RATEMAST_T);

    // 1. 查询基础费率
    rateData = GetBaseRate(req);

    if rateData.rate = 0;
        result.uwDecision = 'D';
        result.uwSuggestion = '无法获取费率数据';
        return result;
    endif;
```

```
    // 2. 计算基础保费
    // 基础保费 = 保额 × 费率 / 缴费方式因子
    result.basePrem = req.sumAssured * rateData.rate / 1000;
    result.basePrem = result.basePrem / GetPayModeFactor(req.premMode);

    // 3. 职业加费
    if req.occupClass > '1';
        result.extraPremRate = (req.occupClass - '1') * 10;  // 每高一类
加10%
        result.extraPrem = result.basePrem * result.extraPremRate /
100;
    endif;

    // 4. 健康加费
    if req.isSmoker = 'Y';
        result.extraPremRate = result.extraPremRate + 20;  // 吸烟加20%
        result.extraPrem = result.basePrem * result.extraPremRate /
100;
    endif;

    if req.bmi < 18.5 or req.bmi > 28;
        result.extraPremRate = result.extraPremRate + 10;  // BMI异常加
10%
        result.extraPrem = result.basePrem * result.extraPremRate /
100;
    endif;

    // 5. 计算总保费
    result.totalPrem = result.basePrem + result.extraPrem;

    // 6. 确定核保结论
    if result.extraPremRate > 50;
        result.uwDecision = 'D';  // 拒保
        result.uwSuggestion = '风险过高，建议拒保';
    elseif result.extraPremRate > 0;
        result.uwDecision = 'S';  // 加费承保
        result.uwSuggestion = '标准体加费' + %char(result.extraPremRate)
+ '%';
    else;
        result.uwDecision = 'A';  // 标准体
        result.uwSuggestion = '标准费率承保';
    endif;

    return result;
```

```
end-proc;

// ====================================================================
// 获取缴费方式因子
// ====================================================================
dcl-proc GetPayModeFactor;
    dcl-pi *n packed(5:3);
        payMode char(1) const;
    end-pi;

    select;
        when payMode = '1';  // 趸交
            return 1;
        when payMode = '2';  // 年交
            return 1;
        when payMode = '3';  // 半年交
            return 0.52;
        when payMode = '4';  // 季交
            return 0.265;
        when payMode = '5';  // 月交
            return 0.09;
        other;
            return 1;
    endsl;
end-proc;
```

5.2 理赔处理系统

5.2.1 理赔流程



```
Approve │   │
│  └──────────┘    └──────────┘    └──────────┘    └──────────┘
└──────────┘   │
│       ↑
│       │
│       └──────────────── 支付 ← 结案 ← 审核
←──────────────────────────┘       │
│
      │
└──────────────────────────────────────────────────────────────────────────
──────┘
```

5.2.2 理赔处理程序

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('CLAIM');

// ====================================================================
// 程序: CLAIMPROC
// 描述: 理赔处理核心
// ====================================================================

// 报案信息结构
dcl-ds ClaimReport qualified template;
    reportNo        char(20);
    reportDt        packed(8);
    reportChnl      char(2);        // 报案渠道

    policyNo        char(20);
    insuredId       char(20);

    claimType       char(2);        // 01-身故 02-重疾 03-医疗 04-伤残 05-
意外...
    accidentDt      packed(8);
    accidentPlace   varchar(200);
    accidentDesc    varchar(2000);

    reporterName    varchar(100);
    reporterPhone   varchar(20);
    reporterRel     char(2);        // 与被保人关系

    hospitalName    varchar(100);
    diagnoseDesc    varchar(500);
```

```
    claimStat       char(2);        // 01-报案 02-受理 03-查勘...
end-ds;

// 理赔金额结构
dcl-ds ClaimAmount qualified template;
    claimNo         char(20);
    policyNo        char(20);
    coverSeq        packed(3);

    applyAmt        packed(19:2);   // 申请金额
    approveAmt      packed(19:2);   // 核赔金额
    deductAmt       packed(19:2);   // 扣减金额

    socInsureAmt    packed(19:2);   // 社保已报销
    otherInsureAmt  packed(19:2);   // 其他保险已赔

    deductReason    varchar(500);
end-ds;

// ============================================================
// 报案登记
// ============================================================
dcl-proc ReportClaim export;
    dcl-pi *n likeds(ClaimResponse);
        report      likeds(ClaimReport) const;
    end-pi;

    dcl-ds resp likeds(ClaimResponse) inz;
    dcl-ds policyData likeds(POLICYMAST_T);

    // 1. 校验保单有效性
    if not ValidatePolicy(report.policyNo : policyData : resp);
        return resp;
    endif;

    // 2. 检查报案时效
    if not CheckReportLimit(report.policyNo : report.accidentDt :
resp);
        return resp;
    endif;

    // 3. 生成报案号
    resp.reportNo = GenerateReportNo();

    // 4. 保存报案信息
```

```
    SaveClaimReport(report : resp.reportNo);

    // 5. 发送短信通知
    SendClaimNotification(policyData.APPNT_MOBILE : resp.reportNo);

    resp.rtnCode = '00';
    resp.rtnMsg = '报案成功，报案号:' + resp.reportNo;

    return resp;
end-proc;

// ====================================================================
// 理赔理算
// ====================================================================
dcl-proc CalculateClaimAmount export;
    dcl-pi *n likeds(ClaimAmount);
        claimNo     char(20) const;
    end-pi;

    dcl-ds result likeds(ClaimAmount);
    dcl-ds coverData likeds(POLICYCOVER_T);
    dcl-ds claimData likeds(CLAIMMAST_T);

    // 读取理赔主信息
    exec SQL
        SELECT * INTO :claimData
        FROM CLAIMMAST
        WHERE CLAIM_NO = :claimNo;

    result.claimNo = claimNo;
    result.policyNo = claimData.POLICY_NO;

    // 读取保单责任
    exec SQL
        SELECT * INTO :coverData
        FROM POLICYCOVER
        WHERE POLICY_NO = :claimData.POLICY_NO
          AND COVER_STAT = '1'
          AND COVER_CD = :claimData.COVER_CD;

    result.coverSeq = coverData.COVER_SEQ;

    // 根据理赔类型计算
    select;
        when claimData.CLAIM_TYPE = '01';  // 身故
            result.approveAmt = coverData.SUM_ASSURED;
```

```
        when claimData.CLAIM_TYPE = '02';  // 重疾
            result.approveAmt = coverData.SUM_ASSURED;
        when claimData.CLAIM_TYPE = '03';  // 医疗
            // 医疗险理算 = min(发票金额 - 社保 - 免赔额, 保额) × 给付比例
            result.approveAmt = CalculateMedicalClaim(claimData :
coverData);
        when claimData.CLAIM_TYPE = '04';  // 伤残
            result.approveAmt = CalculateDisabilityClaim(claimData :
coverData);
    endsl;

    // 扣除其他保险已赔付
    result.approveAmt = result.approveAmt - claimData.OTHER_INSURE_AMT;
    if result.approveAmt < 0;
        result.approveAmt = 0;
    endif;

    // 保存理算结果
    SaveClaimCalculation(result);

    return result;
end-proc;

// ================================================================
// 医疗险理算
// ================================================================
dcl-proc CalculateMedicalClaim;
    dcl-pi *n packed(19:2);
        claimData   likeds(CLAIMMAST_T) const;
        coverData   likeds(POLICYCOVER_T) const;
    end-pi;

    dcl-s invoiceAmt packed(19:2);
    dcl-s socInsureAmt packed(19:2);
    dcl-s deductible packed(19:2);
    dcl-s indemnifyPct packed(5:2);
    dcl-s calcAmt packed(19:2);

    // 获取发票金额
    exec SQL
        SELECT COALESCE(SUM(INVOICE_AMT), 0) INTO :invoiceAmt
        FROM CLAIMINVOICE
        WHERE CLAIM_NO = :claimData.CLAIM_NO;

    // 获取社保报销金额
    exec SQL
```

```
        SELECT COALESCE(SUM(SOC_INSURE_AMT), 0) INTO :socInsureAmt
        FROM CLAIMINVOICE
        WHERE CLAIM_NO = :claimData.CLAIM_NO;

    // 免赔额和给付比例
    deductible = coverData.DEDUCTIBLE_AMT;
    indemnifyPct = coverData.INDEMNITY_PCT;

    // 计算公式
    calcAmt = (invoiceAmt - socInsureAmt - deductible) * indemnifyPct /
100;

    // 不超过保额
    if calcAmt > coverData.SUM_ASSURED;
        calcAmt = coverData.SUM_ASSURED;
    endif;

    return calcAmt;
end-proc;
```

5.3 精算系统

5.3.1 精算模型

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('ACTUARY');

// ============================================================
// 程序: ACTUARY
// 描述: 精算计算引擎
// ============================================================

// 生命表数据结构
dcl-ds LifeTable qualified template;
    age             packed(3);
    qx              packed(9:7);    // 死亡率
    lx              packed(12:0);   // 生存人数
    dx              packed(10:0);   // 死亡人数
    ex              packed(5:2);    // 平均余命
end-ds;

// ============================================================
```

```
// 计算纯保费(净保费)
// 使用公式: P = A / ä
// A: 保险金现值 ä: 年金现值
// ============================================================
dcl-proc CalculateNetPremium export;
    dcl-pi *n packed(19:4);
        age         packed(3) const;        // 投保年龄
        term        packed(3) const;        // 保险期间
        payTerm     packed(3) const;        // 缴费期间
        sumAssured  packed(19:2) const;     // 保额
        intRate     packed(9:7) const;      // 预定利率
        gender      char(1) const;          // M-男 F-女
    end-pi;

    dcl-s axn packed(12:6);     // 生存年金现值
    dcl-s Axn packed(12:6);     // 保险金现值
    dcl-s Pn packed(19:4);      // 纯保费
    dcl-s v packed(9:7);        // 折现因子

    v = 1 / (1 + intRate);

    // 计算生存年金现值 axn
    axn = CalculateAnnuityPV(age : payTerm : v : gender);

    // 计算保险金现值 Axn
    Axn = CalculateBenefitPV(age : term : v : gender);

    // 纯保费 = 保额 × 保险金现值 / 年金现值
    Pn = sumAssured * Axn / axn;

    return Pn;
end-proc;

// ============================================================
// 计算现金价值
// ============================================================
dcl-proc CalculateCashValue export;
    dcl-pi *n packed(19:2);
        policyNo    char(20) const;
        evalDate    packed(8) const;
    end-pi;

    dcl-s cashValue packed(19:2);
    dcl-ds policyData likeds(POLICYMAST_T);
    dcl-s policyYear packed(3);
```

```
    dcl-s futurePrem packed(19:2);
    dcl-s futureBenefit packed(19:2);

    // 读取保单信息
    exec SQL
        SELECT * INTO :policyData
        FROM POLICYMAST
        WHERE POLICY_NO = :policyNo;

    // 计算已过年数
    policyYear = (%date(evalDate : *iso) - %date(policyData.EFFECT_DT :
*iso)) / 365;

    // 未来保费现值
    futurePrem = CalculateFuturePremPV(policyData : policyYear);

    // 未来保险金现值
    futureBenefit = CalculateFutureBenefitPV(policyData : policyYear);

    // 现金价值 = 未来保险金现值 - 未来保费现值
    cashValue = futureBenefit - futurePrem;

    if cashValue < 0;
        cashValue = 0;
    endif;

    return cashValue;
end-proc;

// ====================================================================
// 责任准备金计算
// ====================================================================
dcl-proc CalculatePolicyReserve export;
    dcl-pi *n packed(19:2);
        policyNo    char(20) const;
        evalDate    packed(8) const;
    end-pi;

    dcl-s reserve packed(19:2);
    dcl-ds policyData likeds(POLICYMAST_T);
    dcl-s policyYear packed(3);
    dcl-s netPrem packed(19:4);
    dcl-s paidPrem packed(19:2);
    dcl-s benefitPV packed(19:2);

    // 读取保单
```

```
    exec SQL
        SELECT * INTO :policyData
        FROM POLICYMAST
        WHERE POLICY_NO = :policyNo;

    policyYear = (%date(evalDate : *iso) - %date(policyData.EFFECT_DT :
*iso)) / 365;

    // 计算已缴保费
    paidPrem = policyYear * policyData.BASE_PREM;

    // 未来保险金现值
    benefitPV = CalculateFutureBenefitPV(policyData : policyYear);

    // 责任准备金 = 未来保险金现值 - 未来保费现值
    reserve = benefitPV - CalculateFuturePremPV(policyData :
policyYear);

    return reserve;
end-proc;
```

5.4 再保险系统

5.4.1 再保险处理

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('REINSURANCE');

// ====================================================================
// 程序: REINSURANCE
// 描述: 再保险处理核心
// ====================================================================

// 再保险合约结构
dcl-ds ReinsTreaty qualified template;
    treatyNo        char(20);
    treatyType      char(2);        // 01-成数 02-溢额 03-险位超赔...
    reinsurerCode   char(10);

    cessionRate     packed(5:2);    // 分出比例%
    cessionLimit    packed(19:2);   // 分出限额
    retention       packed(19:2);   // 自留额
```

```
    premCededRate   packed(5:2);    // 分保费比例
    commRate        packed(5:2);    // 分保手续费比例
end-ds;

// ================================================================
// 分出计算
// ================================================================
dcl-proc CalculateCession export;
    dcl-pi *n likeds(CessionResult);
        policyNo    char(20) const;
        coverSeq    packed(3) const;
    end-pi;

    dcl-ds result likeds(CessionResult);
    dcl-ds policyCover likeds(POLICYCOVER_T);
    dcl-ds treaty likeds(ReinsTreaty);
    dcl-s sumAssured packed(19:2);

    // 读取保单责任
    exec SQL
        SELECT * INTO :policyCover
        FROM POLICYCOVER
        WHERE POLICY_NO = :policyNo AND COVER_SEQ = :coverSeq;

    sumAssured = policyCover.SUM_ASSURED;

    // 读取再保险合约
    treaty = FindApplicableTreaty(policyCover.COVER_CD : sumAssured);

    // 根据合约类型计算分出
    select;
        when treaty.treatyType = '01';  // 成数分保
            result.cessionAmt = sumAssured * treaty.cessionRate / 100;
            result.retentionAmt = sumAssured - result.cessionAmt;

        when treaty.treatyType = '02';  // 溢额分保
            if sumAssured <= treaty.retention;
                result.retentionAmt = sumAssured;
                result.cessionAmt = 0;
            else;
                result.retentionAmt = treaty.retention;
                result.cessionAmt = sumAssured - treaty.retention;
                if result.cessionAmt > treaty.cessionLimit;
                    result.cessionAmt = treaty.cessionLimit;
                endif;
```

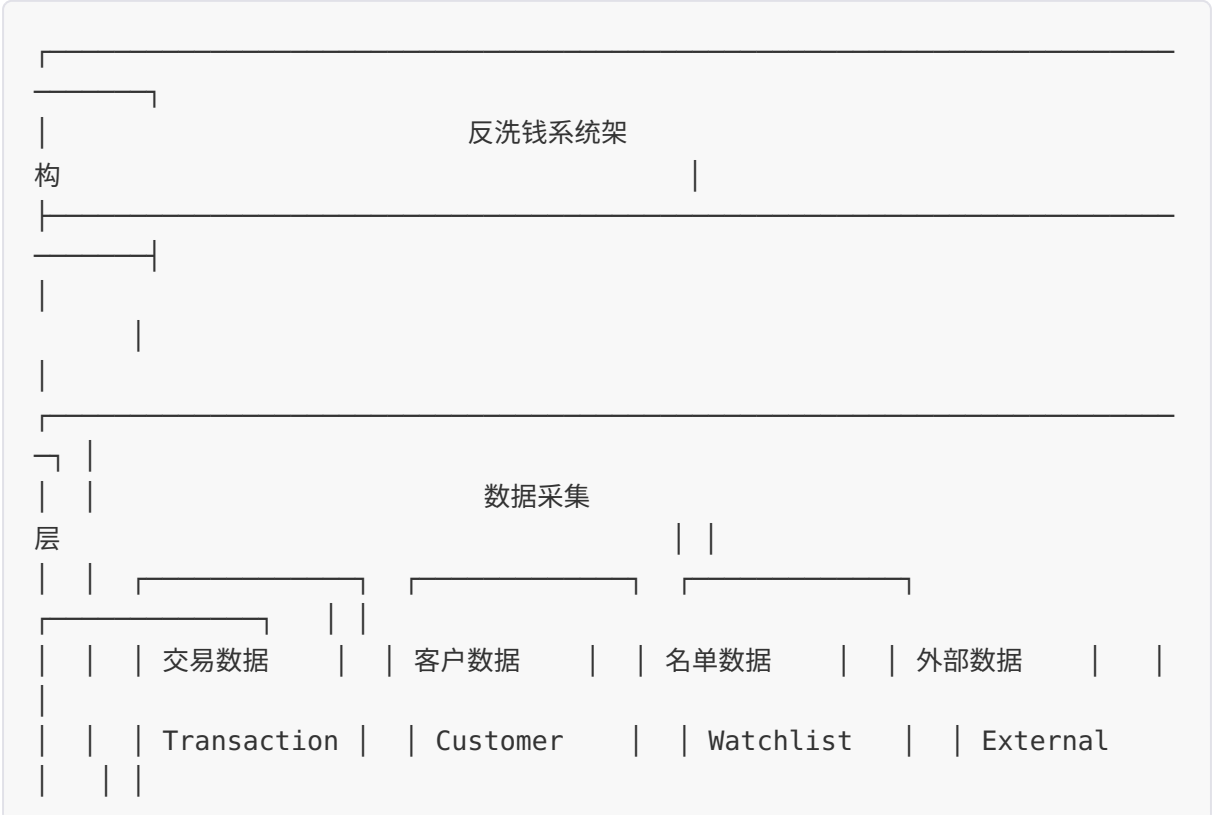
```
            endif;

        when treaty.treatyType = '03';  // 险位超赔
            // 只处理超过自留额的部分
            if sumAssured > treaty.retention;
                result.retentionAmt = treaty.retention;
                result.cessionAmt = sumAssured - treaty.retention;
            else;
                result.retentionAmt = sumAssured;
                result.cessionAmt = 0;
            endif;
    endsl;

    // 计算分出保费和手续费
    result.premCeded = policyCover.PREM * treaty.premCededRate / 100;
    result.commission = result.premCeded * treaty.commRate / 100;

    result.treatyNo = treaty.treatyNo;
    result.reinsurerCode = treaty.reinsurerCode;

    return result;
end-proc;

// ============================================================
// 分保账单生成
// ============================================================
dcl-proc GenerateReinsStatement export;
    dcl-pi *n;
        treatyNo    char(20) const;
        periodStart packed(8) const;
        periodEnd   packed(8) const;
    end-pi;

    dcl-s stmtNo char(20);

    // 生成账单号
    stmtNo = 'RI' + %editc(periodStart:'X') + GenerateSeq();

    // 汇总分保数据
    exec SQL
        INSERT INTO REINS_STMT (
            STMT_NO, TREATY_NO, PERIOD_START, PERIOD_END,
            TOTAL_PREM_CEDED, TOTAL_COMM, TOTAL_CLAIM_CEDED,
            NET_AMOUNT, STMT_DATE
        )
        SELECT
```

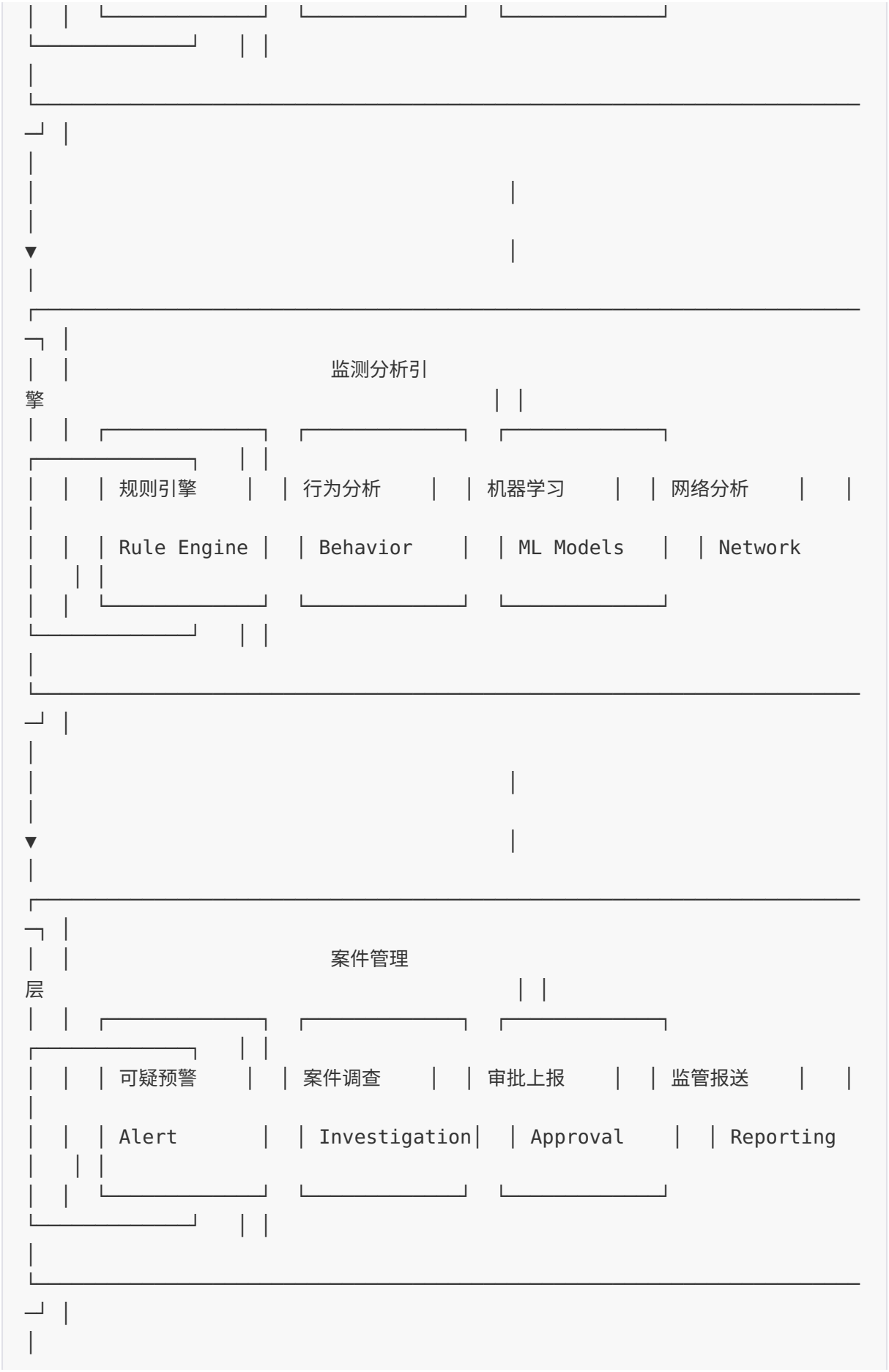
```
            :stmtNo,
            :treatyNo,
            :periodStart,
            :periodEnd,
            SUM(PREM_CEDED),
            SUM(COMMISSION),
            SUM(CLAIM_CEDED),
            SUM(PREM_CEDED) - SUM(COMMISSION) - SUM(CLAIM_CEDED),
            CURRENT DATE
        FROM REINS_CEDE_DTL
        WHERE TREATY_NO = :treatyNo
          AND CEDE_DT BETWEEN :periodStart AND :periodEnd;

end-proc;
```

第六章 金融风控

6.1 反洗钱系统(AML)

6.1.1 AML系统架构




监测分析引
擎
规则引擎
Rule Engine
行为分析
Behavior
机器学习
ML Models
网络分析
Network
案件管理
层
可疑预警
Alert
案件调查
Investigation
审批上报
Approval
监管报送
Reporting

```
      │
└──────────────────────────────────────────────────────────────────────────────
──────┘
```

6.1.2 可疑交易监测规则

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('AML');

// ====================================================================
// 程序: AMLMONITOR
// 描述: 可疑交易监测引擎
// ====================================================================

// 监测规则结构
dcl-ds AMLRule qualified template;
    ruleId          char(10);
    ruleName        varchar(100);
    ruleType        char(2);        // 01-大额 02-可疑 03-异常
    ruleDesc        varchar(500);

    ruleCondition   varchar(2000);  // 规则条件(SQL)
    ruleThreshold   packed(19:2);   // 阈值

    riskLevel       char(1);        // L-低 M-中 H-高
    alertScore      packed(5:0);    // 预警分值
end-ds;

// 预警记录结构
dcl-ds AMLAlert qualified template;
    alertId         char(20);
    alertDt         packed(8);

    custId          char(20);
    acctNo          char(32);

    ruleId          char(10);
    ruleName        varchar(100);
    riskLevel       char(1);
    alertScore      packed(5:0);

    alertDesc       varchar(1000);
    relatedTrans    varchar(2000);  // 关联交易JSON
```

```
    alertStat       char(1);        // 0-待处理 1-已确认 2-已排除...
end-ds;

// ====================================================================
// 可疑交易监测主程序
// ====================================================================
dcl-proc RunAMLMonitoring export;
    dcl-pi *n;
        monitorDate packed(8) const;
        outRtnCode  char(2);
        outRtnMsg   char(200);
    end-pi;

    dcl-ds rules likeds(AMLRule) dim(100);
    dcl-s ruleCnt packed(5);
    dcl-s i packed(5);
    dcl-s totalAlerts packed(10);

    outRtnCode = '00';
    totalAlerts = 0;

    // 1. 加载监测规则
    ruleCnt = LoadAMLRules(rules);

    // 2. 逐条执行规则
    for i = 1 to ruleCnt;
        totalAlerts = totalAlerts + ExecuteRule(rules(i) :
monitorDate);
    endfor;

    // 3. 客户风险评级更新
    UpdateCustomerRiskRating(monitorDate);

    outRtnMsg = '监测完成: 规则数=' + %char(ruleCnt) +
                ', 预警数=' + %char(totalAlerts);

end-proc;

// ====================================================================
// 执行监测规则
// ====================================================================
dcl-proc ExecuteRule;
    dcl-pi *n packed(10);
        rule        likeds(AMLRule) const;
        monitorDate packed(8) const;
```

```
    end-pi;

    dcl-s alertCnt packed(10);

    select;
        when rule.ruleId = 'R001';  // 大额现金交易
            alertCnt = RuleLargeCashTrans(rule : monitorDate);
        when rule.ruleId = 'R002';  // 频繁开户销户
            alertCnt = RuleFreqAccountOpenClose(rule : monitorDate);
        when rule.ruleId = 'R003';  // 短期内资金分散转入集中转出
            alertCnt = RuleStructuring(rule : monitorDate);
        when rule.ruleId = 'R004';  // 跨境异常交易
            alertCnt = RuleAbnormalCrossBorder(rule : monitorDate);
        when rule.ruleId = 'R005';  // 与客户身份不符的大额交易
            alertCnt = RuleUnusualLargeTrans(rule : monitorDate);
        when rule.ruleId = 'R006';  // 休眠账户突然活跃
            alertCnt = RuleDormantAccountActive(rule : monitorDate);
        when rule.ruleId = 'R007';  // 同一IP多账户交易
            alertCnt = RuleMultiAccountSameIP(rule : monitorDate);
    endsl;

    return alertCnt;
end-proc;

// ============================================================
// 规则R001: 大额现金交易监测
// ============================================================
dcl-proc RuleLargeCashTrans;
    dcl-pi *n packed(10);
        rule        likeds(AMLRule) const;
        monitorDate packed(8) const;
    end-pi;

    // 大额标准: 个人≥5万，对公≥20万
    exec SQL
        INSERT INTO AML_ALERT (
            ALERT_ID, ALERT_DT, CUST_ID, ACCT_NO,
            RULE_ID, RULE_NAME, RISK_LEVEL, ALERT_SCORE,
            ALERT_DESC, ALERT_STAT
        )
        SELECT
            'ALT' || CHAR(NEXT VALUE FOR ALERT_SEQ),
            :monitorDate,
            CUST_ID,
            ACCT_NO,
            :rule.ruleId,
```

```
            :rule.ruleName,
            :rule.riskLevel,
            :rule.alertScore,
            '当日单笔现金交易' || CHAR(TRANS_AMT) || '元',
            '0'
        FROM ACCTTRANS
        WHERE TRANS_DT = :monitorDate
          AND TRANS_CHNL = '01'  -- 柜面
          AND TRANS_TYPE IN ('1002', '2002')  -- 现金存取
          AND ((CUST_TYPE = '1' AND TRANS_AMT >= 50000) OR
               (CUST_TYPE = '2' AND TRANS_AMT >= 200000))
          AND NOT EXISTS (
              SELECT 1 FROM AML_ALERT
              WHERE CUST_ID = ACCTTRANS.CUST_ID
                AND ACCT_NO = ACCTTRANS.ACCT_NO
                AND ALERT_DT = :monitorDate
                AND RULE_ID = :rule.ruleId
          );

    GET DIAGNOSTICS alertCnt = ROW_COUNT;

    return alertCnt;
end-proc;

// ============================================================
// 规则R003: 短期内资金分散转入集中转出(疑似拆分)
// ============================================================
dcl-proc RuleStructuring;
    dcl-pi *n packed(10);
        rule        likeds(AMLRule) const;
        monitorDate packed(8) const;
    end-pi;

    // 监测30天内，分散转入(≥5笔)后集中转出(单笔≥80%总额)
    exec SQL
        INSERT INTO AML_ALERT (
            ALERT_ID, ALERT_DT, CUST_ID, ACCT_NO,
            RULE_ID, RULE_NAME, RISK_LEVEL, ALERT_SCORE,
            ALERT_DESC, ALERT_STAT
        )
        WITH INFLOW AS (
            SELECT CUST_ID, ACCT_NO, COUNT(*) AS IN_CNT, SUM(TRANS_AMT)
AS IN_AMT
            FROM ACCTTRANS
            WHERE TRANS_DT BETWEEN :monitorDate - 30 AND :monitorDate
              AND DR_CR_FLAG = 'C'
```

```
              AND TRANS_AMT BETWEEN 10000 AND 50000  -- 疑似拆分金额
            GROUP BY CUST_ID, ACCT_NO
            HAVING COUNT(*) >= 5 AND SUM(TRANS_AMT) >= 200000
        ),
        OUTFLOW AS (
            SELECT CUST_ID, ACCT_NO, MAX(TRANS_AMT) AS MAX_OUT_AMT
            FROM ACCTTRANS
            WHERE TRANS_DT BETWEEN :monitorDate - 7 AND :monitorDate
              AND DR_CR_FLAG = 'D'
            GROUP BY CUST_ID, ACCT_NO
        )
        SELECT
            'ALT' || CHAR(NEXT VALUE FOR ALERT_SEQ),
            :monitorDate,
            I.CUST_ID,
            I.ACCT_NO,
            :rule.ruleId,
            :rule.ruleName,
            :rule.riskLevel,
            :rule.alertScore,
            '30天内' || CHAR(I.IN_CNT) || '笔转入共' || CHAR(I.IN_AMT)
||
            '元，后单笔转出' || CHAR(O.MAX_OUT_AMT) || '元',
            '0'
        FROM INFLOW I
        JOIN OUTFLOW O ON I.CUST_ID = O.CUST_ID AND I.ACCT_NO =
O.ACCT_NO
        WHERE O.MAX_OUT_AMT >= I.IN_AMT * 0.8
          AND NOT EXISTS (
              SELECT 1 FROM AML_ALERT
              WHERE CUST_ID = I.CUST_ID
                AND ALERT_DT = :monitorDate
                AND RULE_ID = :rule.ruleId
          );

    GET DIAGNOSTICS alertCnt = ROW_COUNT;

    return alertCnt;
end-proc;

// ===================================================================
// 制裁名单筛查
// ===================================================================
dcl-proc ScreenSanctionList export;
    dcl-pi *n ind;  // 返回是否命中
        custName    varchar(100) const;
```

```
        idNo        varchar(50) const;
        birthDate   packed(8) const;
        nationCd    char(3) const;
        outHitList  char(200);      // 命中的名单
    end-pi;

    dcl-s hitCnt packed(5);

    outHitList = '';

    // 精确匹配检查
    exec SQL
        SELECT COUNT(*), LISTAGG(LIST_NAME, ';')
        INTO :hitCnt, :outHitList
        FROM SANCTION_LIST
        WHERE (NAME = :custName OR ALT_NAME = :custName)
          AND (ID_NO = :idNo OR :idNo = '')
        GROUP BY NAME;

    if hitCnt > 0;
        return *on;
    endif;

    // 模糊匹配检查(使用相似度算法)
    exec SQL
        SELECT COUNT(*), LISTAGG(LIST_NAME, ';')
        INTO :hitCnt, :outHitList
        FROM SANCTION_LIST
        WHERE DIFFERENCE(NAME, :custName) >= 3  -- Soundex相似度
           OR SIMILARITY(NAME, :custName) >= 0.8;  -- 文本相似度

    return (hitCnt > 0);
end-proc;
```

6.2 KYC/CDD系统

6.2.1 客户尽职调查

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('KYC');

// ============================================================
```

```
// 程序: KYCPROC
// 描述: KYC/CDD处理核心
// ============================================================

// KYC风险评级因素
dcl-ds KYCRiskFactor qualified template;
    customerType    char(1);        // 1-个人 2-企业
    customerSegment char(2);        // 01-普通 02-私人银行...

    // 地理风险
    isHighRiskCountry ind;
    isSanctionedCountry ind;

    // 业务风险
    isCashIntensive ind;
    isComplexProduct ind;
    isHighValue ind;

    // 行为风险
    hasUnusualPattern ind;
    hasNegativeNews ind;

    // PEP状态
    isPEP           ind;
    isPEPFamily     ind;
    isPEPAssociate  ind;
end-ds;

// ============================================================
// 客户风险评级计算
// ============================================================
dcl-proc CalculateCustomerRiskRating export;
    dcl-pi *n char(2);
        custId      char(20) const;
    end-pi;

    dcl-ds factors likeds(KYCRiskFactor);
    dcl-s riskScore packed(5);
    dcl-s riskLevel char(2);

    // 收集风险因素
    factors = CollectRiskFactors(custId);

    // 计算风险评分
    riskScore = 0;
```

```
    // 地理风险(最高30分)
    if factors.isSanctionedCountry;
        riskScore = riskScore + 30;
    elseif factors.isHighRiskCountry;
        riskScore = riskScore + 15;
    endif;

    // PEP风险(最高25分)
    if factors.isPEP;
        riskScore = riskScore + 25;
    elseif factors.isPEPFamily or factors.isPEPAssociate;
        riskScore = riskScore + 15;
    endif;

    // 业务风险(最高20分)
    if factors.isCashIntensive;
        riskScore = riskScore + 10;
    endif;
    if factors.isComplexProduct;
        riskScore = riskScore + 10;
    endif;

    // 行为风险(最高25分)
    if factors.hasUnusualPattern;
        riskScore = riskScore + 15;
    endif;
    if factors.hasNegativeNews;
        riskScore = riskScore + 10;
    endif;

    // 确定风险等级
    if riskScore >= 70;
        riskLevel = 'H1';  // 高风险
    elseif riskScore >= 50;
        riskLevel = 'H2';  // 中高风险
    elseif riskScore >= 30;
        riskLevel = 'M1';  // 中风险
    elseif riskScore >= 10;
        riskLevel = 'M2';  // 中低风险
    else;
        riskLevel = 'L1';  // 低风险
    endif;

    // 更新客户风险等级
    exec SQL
        UPDATE CUSTMAST
```

```
        SET RISK_LEVEL = :riskLevel,
            RISK_RATE_DT = CURRENT DATE,
            UPDATE_TS = CURRENT_TIMESTAMP
        WHERE CUST_ID = :custId;

    return riskLevel;
end-proc;

// ====================================================================
// 定期KYC复审
// ====================================================================
dcl-proc KYCRenewalCheck export;
    dcl-pi *n;
        checkDate   packed(8) const;
        outRtnCode  char(2);
        outRtnMsg   char(200);
    end-pi;

    dcl-s dueCnt packed(10);

    outRtnCode = '00';

    // 根据风险等级确定复审周期:
    // 高风险 - 6个月
    // 中高风险 - 12个月
    // 中风险 - 24个月
    // 低风险 - 36个月

    exec SQL
        SELECT COUNT(*) INTO :dueCnt
        FROM CUSTMAST
        WHERE ACCT_STAT = '0'
          AND KYC_STAT = '2'
          AND (
              (RISK_LEVEL LIKE 'H%' AND KYC_NEXT_DT <= :checkDate) OR
              (RISK_LEVEL = 'M1' AND KYC_NEXT_DT <= :checkDate - 10000)
OR
              (RISK_LEVEL = 'M2' AND KYC_NEXT_DT <= :checkDate - 20000)
OR
              (RISK_LEVEL = 'L1' AND KYC_NEXT_DT <= :checkDate - 30000)
          );

    // 生成KYC复审任务
    exec SQL
        INSERT INTO KYC_RENEWAL_TASK (
            TASK_ID, CUST_ID, DUE_DT, PRIORITY, TASK_STAT, CREATE_DT
```

```
        )
        SELECT
            'KYC' || CHAR(NEXT VALUE FOR KYC_TASK_SEQ),
            CUST_ID,
            KYC_NEXT_DT,
            CASE RISK_LEVEL
                WHEN 'H1' THEN '1'
                WHEN 'H2' THEN '2'
                ELSE '3'
            END,
            '0',  -- 待处理
            :checkDate
        FROM CUSTMAST
        WHERE ACCT_STAT = '0'
          AND KYC_STAT = '2'
          AND KYC_NEXT_DT <= :checkDate;

    outRtnMsg = 'KYC复审任务生成: ' + %char(dueCnt) + '笔';

end-proc;
```

6.3 风险加权资产(RWA)计算

6.3.1 信用风险RWA计算

```
**free
ctl-opt option(*srcstmt : *nodedebugio);
ctl-opt dftactgrp(*no) actgrp('RWA');

// ====================================================================
// 程序: RWACALC
// 描述: 风险加权资产计算
// 依据: 巴塞尔协议III
// ====================================================================

// 风险敞口结构
dcl-ds RiskExposure qualified template;
    exposureId      char(20);
    exposureType    char(2);        // 01-贷款 02-债券 03-表外...

    bookValue       packed(19:2);
    provision       packed(19:2);
    netValue        packed(19:2);
```

```
    counterpartyId  char(20);
    counterpartyType char(2);       // 01-主权 02-银行 03-企业...

    pd              packed(9:7);    // 违约概率
    lgd             packed(5:2);    // 违约损失率
    ead             packed(19:2);   // 违约风险敞口
    maturity        packed(5:2);    // 剩余期限(年)

    collateralValue packed(19:2);
    guaranteeValue  packed(19:2);
end-ds;

// ================================================================
// RWA计算主程序
// ================================================================
dcl-proc CalculateRWA export;
    dcl-pi *n;
        calcDate    packed(8) const;
        outRtnCode  char(2);
        outRtnMsg   char(200);
    end-pi;

    dcl-s creditRWA packed(19:2);
    dcl-s marketRWA packed(19:2);
    dcl-s opRiskRWA packed(19:2);
    dcl-s totalRWA packed(19:2);

    // 1. 计算信用风险RWA
    creditRWA = CalculateCreditRiskRWA(calcDate);

    // 2. 计算市场风险RWA
    marketRWA = CalculateMarketRiskRWA(calcDate);

    // 3. 计算操作风险RWA
    opRiskRWA = CalculateOpRiskRWA(calcDate);

    // 4. 汇总
    totalRWA = creditRWA + marketRWA + opRiskRWA;

    // 5. 保存结果
    exec SQL
        INSERT INTO RWA_RESULT (
            CALC_DATE, CREDIT_RWA, MARKET_RWA, OP_RISK_RWA, TOTAL_RWA
        ) VALUES (
            :calcDate, :creditRWA, :marketRWA, :opRiskRWA, :totalRWA
```

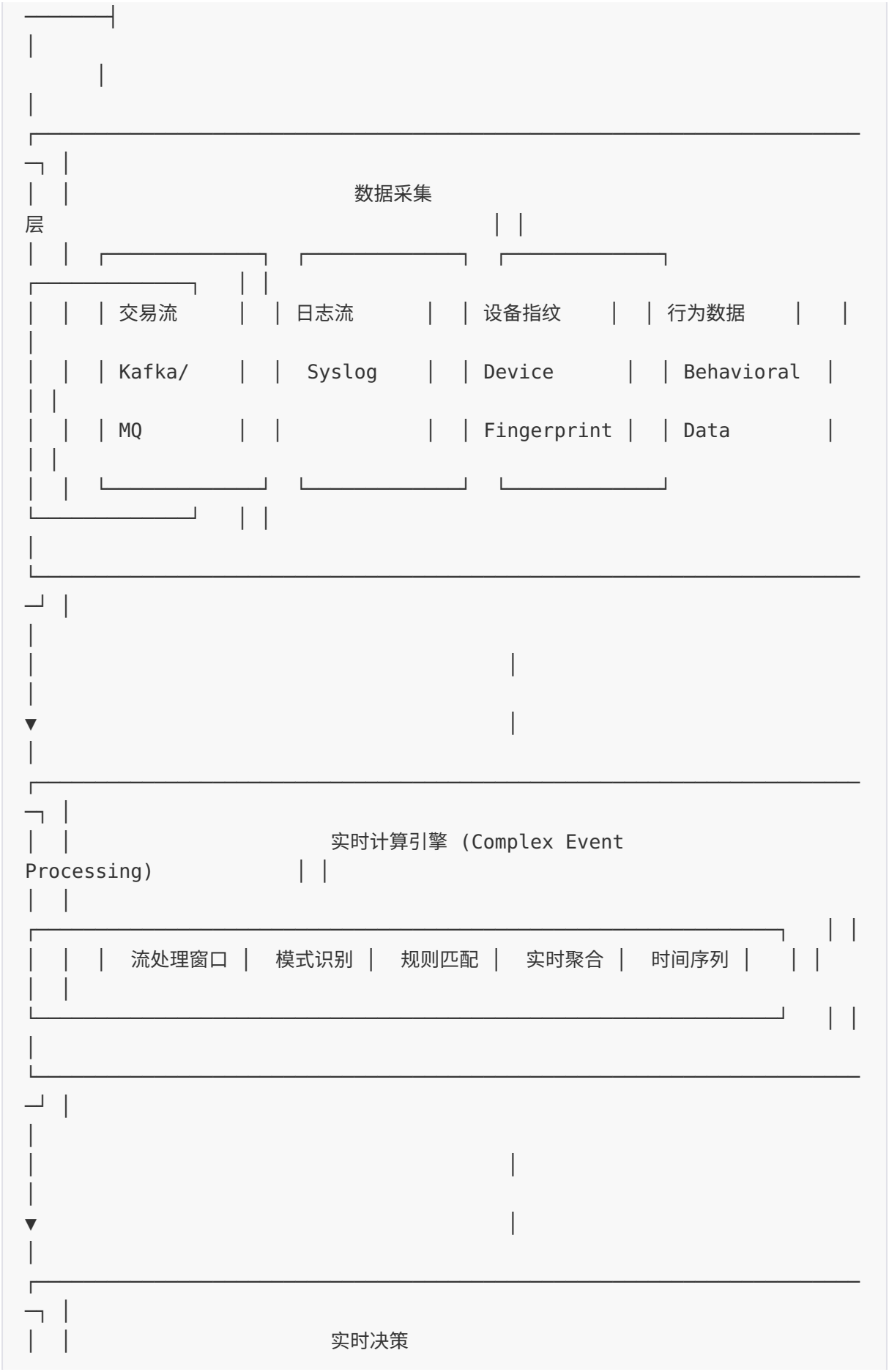
```
        );

    outRtnCode = '00';
    outRtnMsg = 'RWA计算完成: 总RWA=' + %char(totalRWA);

end-proc;

// ================================================================
// 信用风险RWA计算(标准法)
// ================================================================
dcl-proc CalculateCreditRiskRWA;
    dcl-pi *n packed(19:2);
        calcDate    packed(8) const;
    end-pi;

    dcl-s rwa packed(19:2);
    dcl-s riskWeight packed(5:2);

    rwa = 0;

    // 贷款类资产RWA计算
    exec SQL
        SELECT COALESCE(SUM(
            CASE
                WHEN COLLATERAL_VALUE >= BOOK_VALUE * 0.3 THEN
                    (BOOK_VALUE - PROVISION) * 0.35  -- 足值抵押35%权重
                WHEN RISK_RATING = 'AAA' THEN
                    (BOOK_VALUE - PROVISION) * 0.20
                WHEN RISK_RATING = 'AA' THEN
                    (BOOK_VALUE - PROVISION) * 0.50
                WHEN RISK_RATING = 'A' THEN
                    (BOOK_VALUE - PROVISION) * 1.00
                WHEN RISK_RATING = 'BBB' THEN
                    (BOOK_VALUE - PROVISION) * 1.00
                ELSE
                    (BOOK_VALUE - PROVISION) * 1.50
            END
        ), 0)
        INTO :rwa
        FROM LOAN_EXPOSURE
        WHERE CALC_DATE = :calcDate;

    return rwa;
end-proc;

// ================================================================
```

```
// 信用风险RWA计算(内部评级法IRB)
// ================================================================
dcl-proc CalculateIRBRWA;
    dcl-pi *n packed(19:2);
        calcDate    packed(8) const;
    end-pi;

    dcl-s rwa packed(19:2);
    dcl-s correlation packed(9:7);
    dcl-s maturityAdj packed(9:7);
    dcl-s k packed(19:7);       // 资本要求
    dcl-s r packed(9:7);        // 相关系数

    rwa = 0;

    // IRB公式: K = LGD × N[(1-R)^-0.5 × G(PD) + (R/(1-R))^0.5 ×
G(0.999)] - PD × LGD
    // RWA = K × EAD × 12.5

    exec SQL
        SELECT COALESCE(SUM(
            EAD * 12.5 * (
                LGD * SQRT(1 - CORR) * NORMSINV(PD) +
                SQRT(CORR / (1 - CORR)) * NORMSINV(0.999) -
                PD * LGD
            ) * MATURITY_ADJ
        ), 0)
        INTO :rwa
        FROM IRB_EXPOSURE
        WHERE CALC_DATE = :calcDate;

    return rwa;
end-proc;
```

6.4 实时监控系统

6.4.1 实时风控架构

```
┌──────────────────────────────────────────────────────────────────
──────┐
│                              实时风控监控架
构                                    │
├──────────────────────────────────────────────────────────────────
```


数据采集层
交易流
Kafka/ MQ
日志流
Syslog
设备指纹
Device Fingerprint
行为数据
Behavioral Data
实时计算引擎 (Complex Event Processing)
流处理窗口
模式识别
规则匹配
实时聚合
时间序列
实时决策

```
层                                          | |
|  |  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐
┌──────────────┐   | |
|  |  | 通过       |  | 加强认证    |  | 交易阻断    |  | 人工审核    |   |
|
|  |  | Pass      |  | Step-up    |  | Block      |  | Manual      |
| |
|  |  |           |  | Auth       |  |            |  | Review      |
| |
|  |  └──────────────┘  └──────────────┘  └──────────────┘
└──────────────┘   | |
|
└──────────────────────────────────────────────────────────────────────────
┘ |
|
      |
└──────────────────────────────────────────────────────────────────────────
──────┘
```

6.4.2 实时规则引擎

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('REALTIMERISK');

// ============================================================
// 程序: REALTIMERISK
// 描述: 实时风控规则引擎
// ============================================================

// 实时交易事件
dcl-ds TransactionEvent qualified template;
    eventId         char(32);
    eventTime       timestamp;

    custId          char(20);
    acctNo          char(32);
    cardNo          char(19);

    transType       char(4);
    transAmt        packed(19:2);
    transCcy        char(3);

    deviceId        char(32);
```

```
    ipAddress       char(16);
    geoLocation     varchar(100);

    channel         char(2);
    merchantId      varchar(15);
end-ds;

// 实时决策结果
dcl-ds RealTimeDecision qualified template;
    decision        char(1);        // A-接受 R-拒绝 C-挑战
    decisionCode    char(4);
    decisionReason  varchar(200);

    riskScore       packed(5:2);
    ruleHits        varchar(500);

    // 挑战方式
    challengeType   char(2);        // 01-短信 02-密码 03-人脸...
end-ds;

// ===================================================================
// 实时风控决策
// ===================================================================
dcl-proc RealTimeRiskDecision export;
    dcl-pi *n likeds(RealTimeDecision);
        event       likeds(TransactionEvent) const;
    end-pi;

    dcl-ds decision likeds(RealTimeDecision);
    dcl-s score packed(5:2);

    decision.decision = 'A';
    decision.riskScore = 0;

    // 1. 黑名单检查(同步)
    if CheckBlacklist(event);
        decision.decision = 'R';
        decision.decisionCode = '1001';
        decision.decisionReason = '命中黑名单';
        decision.riskScore = 100;
        return decision;
    endif;

    // 2. 设备风险评估
    score = EvaluateDeviceRisk(event.deviceId : event.custId);
    decision.riskScore = decision.riskScore + score;
```

```
    if score > 50;
        decision.decision = 'C';
        decision.challengeType = '01';
    endif;

    // 3. 地理位置风险评估
    score = EvaluateGeoRisk(event.ipAddress : event.custId);
    decision.riskScore = decision.riskScore + score;
    if score > 30;
        decision.decision = 'C';
        decision.challengeType = '01';
    endif;

    // 4. 行为模式分析
    score = EvaluateBehaviorPattern(event);
    decision.riskScore = decision.riskScore + score;

    // 5. 最终决策
    if decision.riskScore >= 80;
        decision.decision = 'R';
        decision.decisionCode = '1002';
        decision.decisionReason = '风险评分过高';
    elseif decision.riskScore >= 50;
        decision.decision = 'C';
        decision.challengeType = '02';
    endif;

    // 6. 记录决策日志(异步)
    LogRiskDecision(event : decision);

    return decision;
end-proc;

// ============================================================
// 设备风险评估
// ============================================================
dcl-proc EvaluateDeviceRisk;
    dcl-pi *n packed(5:2);
        deviceId    char(32) const;
        custId      char(20) const;
    end-pi;

    dcl-s score packed(5:2);
    dcl-s isKnownDevice ind;
    dcl-s deviceTrustLevel char(1);
```

```
    score = 0;

    // 检查是否为已知设备
    exec SQL
        SELECT '1' INTO :isKnownDevice
        FROM CUST_DEVICE
        WHERE CUST_ID = :custId AND DEVICE_ID = :deviceId;

    if not isKnownDevice;
        score = score + 20;  // 新设备风险
    endif;

    // 获取设备信任等级
    exec SQL
        SELECT TRUST_LEVEL INTO :deviceTrustLevel
        FROM DEVICE_RATING
        WHERE DEVICE_ID = :deviceId;

    if deviceTrustLevel = 'L';
        score = score + 30;
    elseif deviceTrustLevel = 'M';
        score = score + 10;
    endif;

    return score;
end-proc;

// ============================================================
// 地理位置风险评估
// ============================================================
dcl-proc EvaluateGeoRisk;
    dcl-pi *n packed(5:2);
        ipAddress   char(16) const;
        custId      char(20) const;
    end-pi;

    dcl-s score packed(5:2);
    dcl-s currCountry char(3);
    dcl-s usualCountry char(3);
    dcl-s isHighRiskCountry ind;

    score = 0;

    // 获取当前位置
    exec SQL
        SELECT COUNTRY_CD INTO :currCountry
```

```
        FROM IP_GEOLOCATION
        WHERE IP_FROM <= :ipAddress AND IP_TO >= :ipAddress;

    // 获取常用位置
    exec SQL
        SELECT COUNTRY_CD INTO :usualCountry
        FROM CUST_USUAL_LOCATION
        WHERE CUST_ID = :custId
        ORDER BY LAST_USED_TS DESC
        FETCH FIRST 1 ROW ONLY;

    // 异地交易风险
    if currCountry <> usualCountry;
        score = score + 25;
    endif;

    // 高风险国家
    exec SQL
        SELECT '1' INTO :isHighRiskCountry
        FROM HIGH_RISK_COUNTRY
        WHERE COUNTRY_CD = :currCountry;

    if isHighRiskCountry;
        score = score + 35;
    endif;

    return score;
end-proc;

// ====================================================================
// 行为模式分析
// ====================================================================
dcl-proc EvaluateBehaviorPattern;
    dcl-pi *n packed(5:2);
        event       likeds(TransactionEvent) const;
    end-pi;

    dcl-s score packed(5:2);
    dcl-s avgAmt packed(19:2);
    dcl-s recentTransCnt packed(5);
    dcl-s hour packed(2);

    score = 0;

    // 交易金额异常
    exec SQL
```

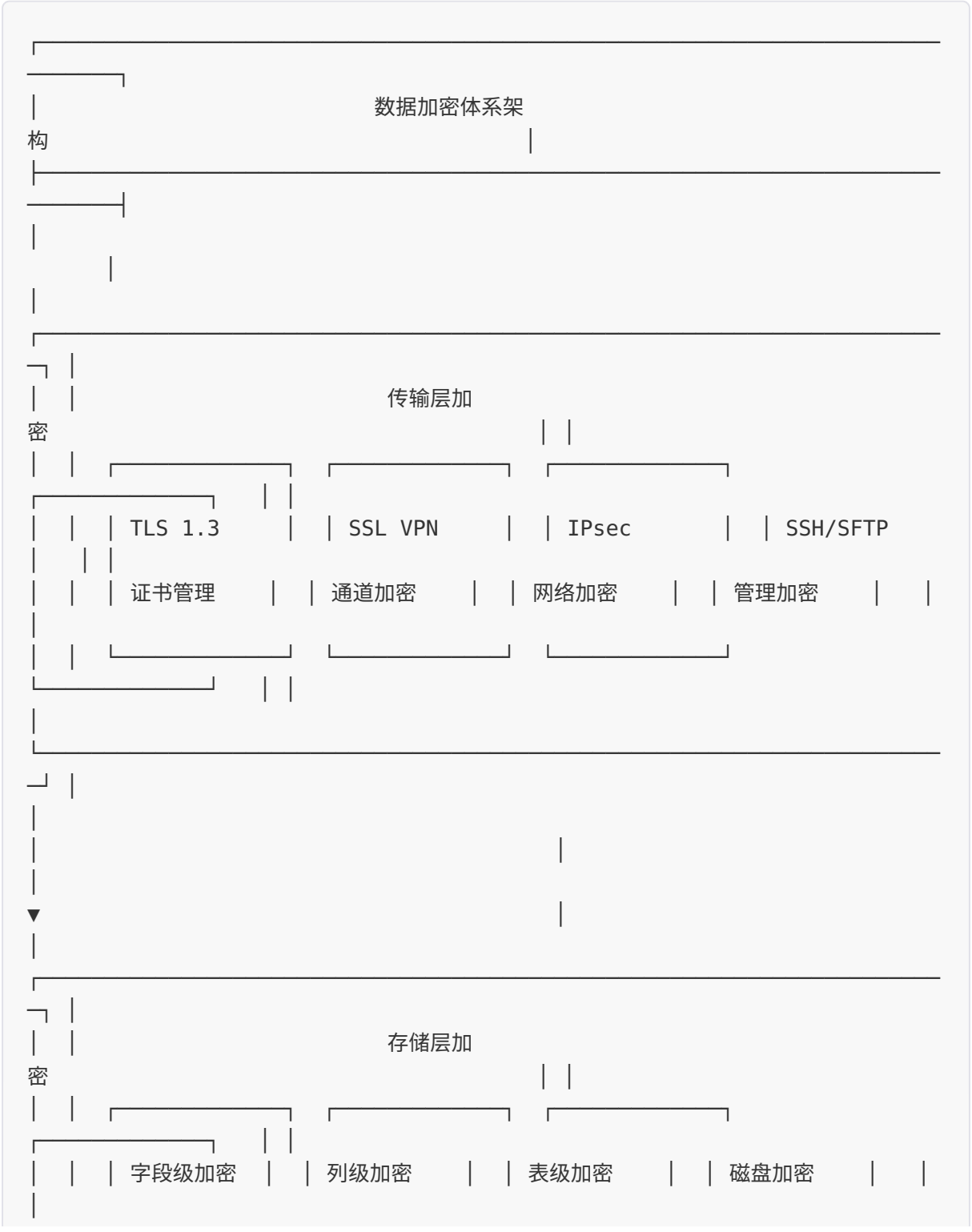
```
        SELECT AVG(TRANS_AMT) INTO :avgAmt
        FROM ACCTTRANS
        WHERE ACCT_NO = :event.acctNo
          AND TRANS_DT > CURRENT DATE - 90 DAYS;

    if event.transAmt > avgAmt * 5;
        score = score + 25;
    elseif event.transAmt > avgAmt * 3;
        score = score + 15;
    endif;

    // 交易频率异常
    exec SQL
        SELECT COUNT(*) INTO :recentTransCnt
        FROM ACCTTRANS
        WHERE ACCT_NO = :event.acctNo
          AND ENTRY_TS > CURRENT TIMESTAMP - 1 HOUR;

    if recentTransCnt > 10;
        score = score + 20;
    endif;

    // 异常时段交易
    hour = %subdt(event.eventTime : *hours);
    if hour < 6 or hour > 23;
        score = score + 10;
    endif;

    return score;
end-proc;
```

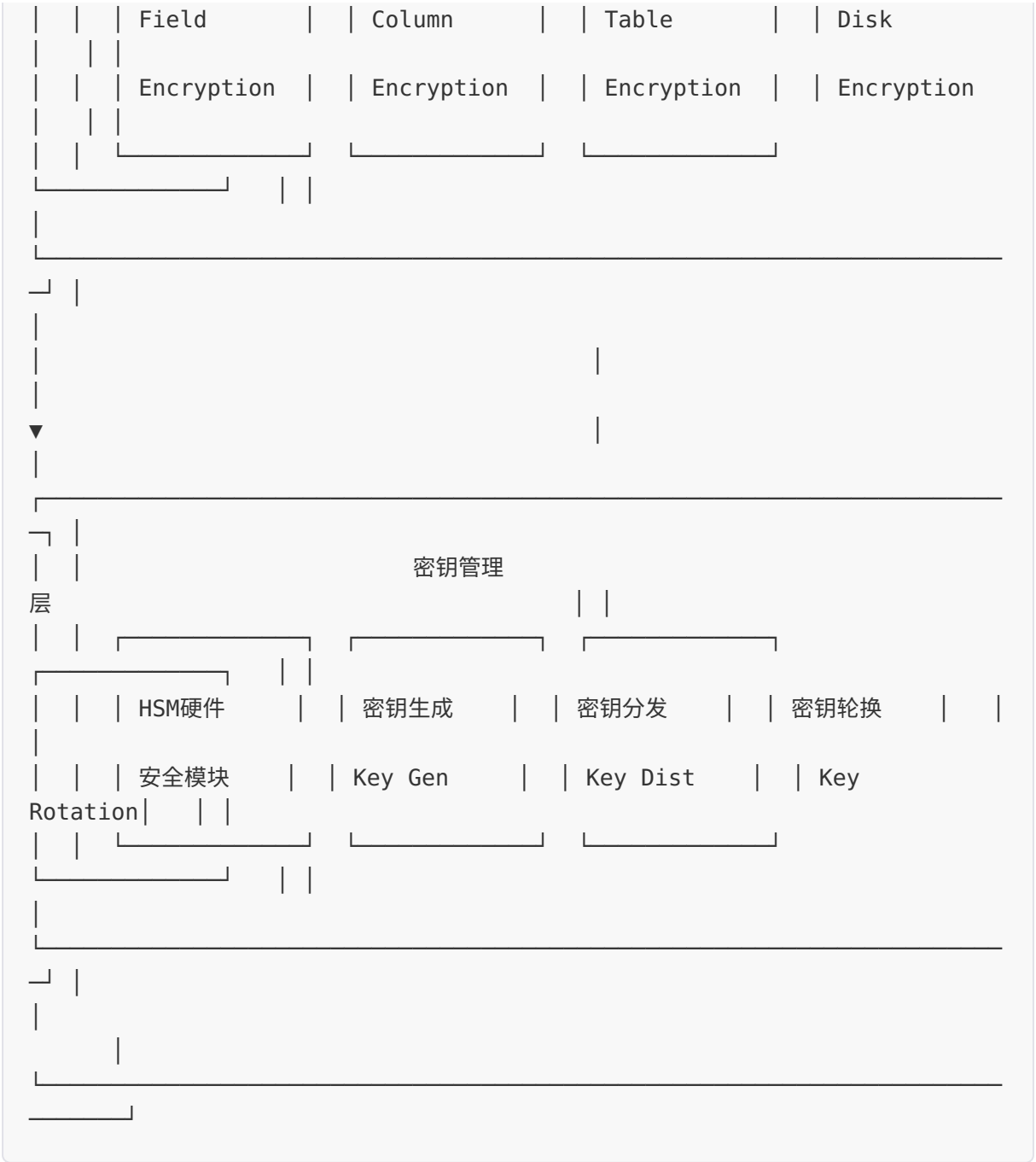
由于篇幅仍然不足，我将继续追加更多章节内容，包括安全合规、性能优化、附录等。

第七章 安全合规

7.1 数据加密

7.1.1 加密体系架构





7.1.2 字段级加密实现

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('ENCRYPTION');

// ============================================================
// 程序: FIELDENC
// 描述: 字段级加密服务
// 算法: AES-256-GCM / SM4
```

```
// ============================================================

// 加密数据结构
dcl-ds EncryptedField qualified template;
    cipherText      varchar(1000);  // 密文(Base64)
    authTag         char(32);       // 认证标签
    nonce           char(24);       // 随机向量
    keyVersion      char(4);        // 密钥版本
    algorithm       char(10);       // AES256/SM4
end-ds;

// 密钥结构
dcl-ds CryptoKey qualified template;
    keyId           char(20);
    keyVersion      char(4);
    keyValue        varchar(256);   // 加密存储
    keyType         char(2);        // DE-数据加密 KE-密钥加密
    createdTs       timestamp;
    expiryTs        timestamp;
end-ds;

// ============================================================
// 加密服务主入口
// ============================================================
dcl-proc EncryptField export;
    dcl-pi *n likeds(EncryptedField);
        plainText   varchar(500) const;
        keyId       char(20) const;
        algorithm   char(10) const options(*nopass);
    end-pi;

    dcl-ds result likeds(EncryptedField);
    dcl-ds key likeds(CryptoKey);
    dcl-s alg char(10);

    // 默认算法
    if %parms >= 3;
        alg = algorithm;
    else;
        alg = 'AES256';
    endif;

    // 获取当前有效密钥
    key = GetActiveKey(keyId);
    if key.keyId = '';
        // 密钥不存在
```

```
        return result;
    endif;

    // 调用加密API
    select;
        when alg = 'AES256';
            result = EncryptAES256(plainText : key);
        when alg = 'SM4';
            result = EncryptSM4(plainText : key);
        when alg = 'RSA';
            result = EncryptRSA(plainText : key);
    endsl;

    result.keyVersion = key.keyVersion;
    result.algorithm = alg;

    return result;
end-proc;

// ============================================================
// 解密服务
// ============================================================
dcl-proc DecryptField export;
    dcl-pi *n varchar(500);
        encData     likeds(EncryptedField) const;
    end-pi;

    dcl-s plainText varchar(500);
    dcl-ds key likeds(CryptoKey);

    // 获取对应版本密钥
    key = GetKeyByVersion(encData.keyId : encData.keyVersion);

    select;
        when encData.algorithm = 'AES256';
            plainText = DecryptAES256(encData : key);
        when encData.algorithm = 'SM4';
            plainText = DecryptSM4(encData : key);
    endsl;

    return plainText;
end-proc;

// ============================================================
// AES-256-GCM加密
// ============================================================
```

```
dcl-proc EncryptAES256;
    dcl-pi *n likeds(EncryptedField);
        plainText   varchar(500) const;
        key         likeds(CryptoKey) const;
    end-pi;

    dcl-ds result likeds(EncryptedField);
    dcl-s plainBlob blob(1000);
    dcl-s cipherBlob blob(1000);
    dcl-s nonce char(12);
    dcl-s authTag char(16);
    dcl-s keyBlob blob(32);

    // 生成随机nonce
    exec SQL
        SELECT HEX(RANDOM()) INTO :nonce
        FROM SYSIBM.SYSDUMMY1;

    // 转换明文为BLOB
    plainBlob = %blob(plainText);
    keyBlob = %blob(key.keyValue);

    // 调用IBM i加密API (QC3ENCDT / Qc3EncryptData)
    // 或使用SQL加密函数
    exec SQL
        SET :cipherBlob = ENCRYPT_AES256(:plainBlob, :keyBlob, :nonce);

    // Base64编码
    result.cipherText = Base64Encode(cipherBlob);
    result.nonce = nonce;
    result.authTag = authTag;

    return result;
end-proc;

// ============================================================
// 敏感数据脱敏
// ============================================================
dcl-proc MaskSensitiveData export;
    dcl-pi *n varchar(100);
        data        varchar(100) const;
        dataType    char(2) const;      // ID-身份证 MO-手机 AC-账号...
    end-pi;

    dcl-s masked varchar(100);
    dcl-s len packed(3);
```

```
    len = %len(%trim(data));

    select;
        when dataType = 'ID';  // 身份证号
            // 显示前6位和后4位
            masked = %subst(data : 1 : 6) + '**********' +
                     %subst(data : len - 3 : 4);
        when dataType = 'MO';  // 手机号
            // 显示前3位和后4位
            masked = %subst(data : 1 : 3) + '****' +
                     %subst(data : len - 3 : 4);
        when dataType = 'AC';  // 银行账号
            // 显示后4位
            masked = '************' + %subst(data : len - 3 : 4);
        when dataType = 'NA';  // 姓名
            // 显示姓，名用*代替
            masked = %subst(data : 1 : 1) + '**';
        when dataType = 'CA';  // 银行卡号
            // 显示前6位和后4位
            masked = %subst(data : 1 : 6) + '********' +
                     %subst(data : len - 3 : 4);
        other;
            masked = '********';
    endsl;

    return masked;
end-proc;

// ============================================================
// SQL字段加密函数封装
// ============================================================
// 创建SQL加密函数
// CREATE FUNCTION ENCRYPT_FIELD(
//     PLAIN_TEXT VARCHAR(500),
//     KEY_ID CHAR(20)
// )
// RETURNS VARCHAR(1000)
// LANGUAGE RPGLE
// PARAMETER STYLE SQL
// EXTERNAL NAME 'FIELDENC(EncryptFieldSQL)'
//;

// 创建SQL解密函数
// CREATE FUNCTION DECRYPT_FIELD(
//     CIPHER_TEXT VARCHAR(1000),
```

```
//     KEY_ID CHAR(20)
// )
// RETURNS VARCHAR(500)
// LANGUAGE RPGLE
// PARAMETER STYLE SQL
// EXTERNAL NAME 'FIELDENC(DecryptFieldSQL)'
//;
```

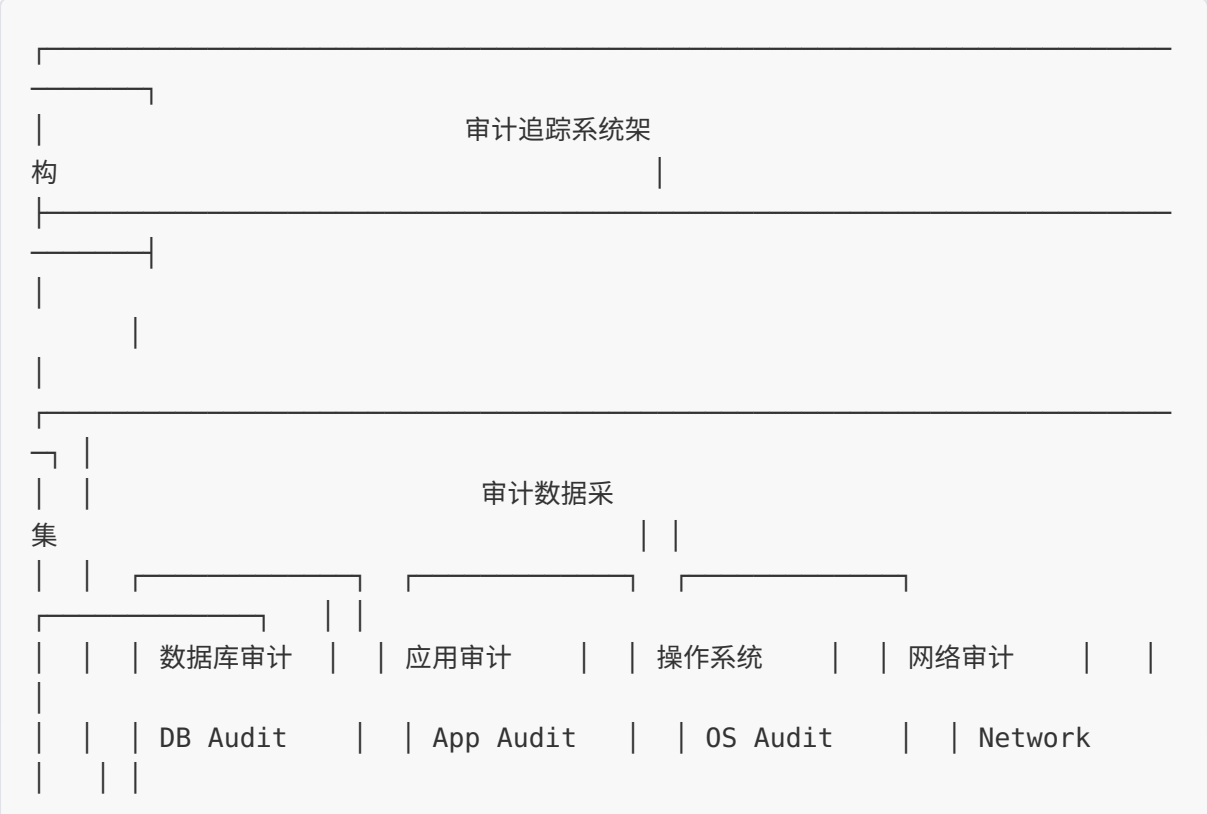
7.1.3 数据库透明加密(TDE)

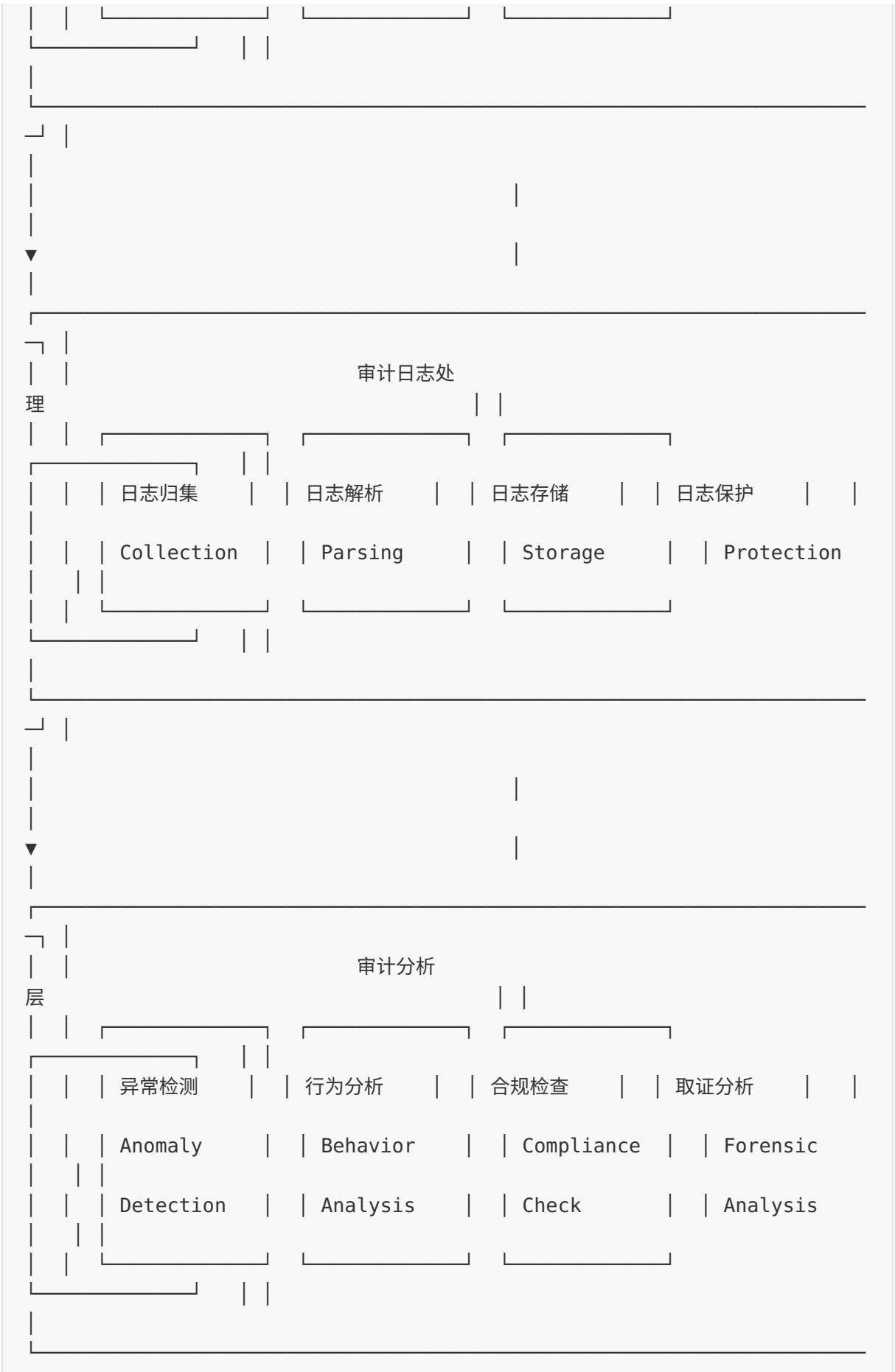
```
-- ================================================================
-- IBM i 数据库透明加密配置
-- ================================================================

-- 1. 创建加密密钥库
CREATE ENCRYPTION KEYSTORE MY_KEYSTORE
    TYPE *SYSTEM
    DESCRIPTION '金融系统数据加密密钥库';

-- 2. 创建主密钥
CREATE MASTER KEY MY_MASTER_KEY
    KEYSTORE MY_KEYSTORE
    ALGORITHM AES256
    LABEL 'Master Encryption Key';

-- 3. 创建表空间加密密钥
CREATE TABLESPACE ENCRYPTION KEY ACCT_TS_KEY
    MASTER KEY MY_MASTER_KEY
    ALGORITHM AES256
    LABEL 'Account Tablespace Key';

-- 4. 创建加密表空间
CREATE TABLESPACE ACCT_ENCRYPTED
    PAGESIZE 32K
    ENCRYPTION ENABLED
    ENCRYPTION KEY ACCT_TS_KEY;

-- 5. 创建加密表
CREATE TABLE ACCTMAST_ENCRYPTED (
    ACCT_NO         CHAR(32)        NOT NULL,
    CUST_ID         CHAR(20)        NOT NULL,
    -- 自动加密字段
    ACCT_BAL        DECIMAL(19, 2)  ENCRYPTED,
    -- 显式加密字段
```

```
    SENSITIVE_DATA  VARCHAR(500)    ENCRYPTED WITH COLUMN ENCRYPTION
KEY,

    CONSTRAINT PK_ACCTENCRYPT PRIMARY KEY (ACCT_NO)
) IN ACCT_ENCRYPTED;

-- 6. 列级加密密钥管理
CREATE COLUMN ENCRYPTION KEY CUST_INFO_KEY
    MASTER KEY MY_MASTER_KEY
    ALGORITHM AES256;

-- 7. 应用列级加密
ALTER TABLE CUSTMAST
    ALTER COLUMN ID_NO
    SET ENCRYPTION KEY CUST_INFO_KEY;

ALTER TABLE CUSTMAST
    ALTER COLUMN MOBILE_NO
    SET ENCRYPTION KEY CUST_INFO_KEY;
```

7.2 审计追踪

7.2.1 审计架构




审计日志处
理
日志归集
Collection
日志解析
Parsing
日志存储
Storage
日志保护
Protection
审计分析
层
异常检测
Anomaly
Detection
行为分析
Behavior
Analysis
合规检查
Compliance
Check
取证分析
Forensic
Analysis

```
┘ │
│
      │
└──────────────────────────────────────────────────────────────────────────────
──────┘
```

7.2.2 审计日志表设计

```
-- ================================================================
-- 审计日志主表
-- ================================================================

CREATE TABLE AUDIT_LOG (
    AUDIT_ID        CHAR(32)        NOT NULL,
    AUDIT_TS        TIMESTAMP       NOT NULL DEFAULT CURRENT_TIMESTAMP,

    -- 审计事件类型
    AUDIT_TYPE      CHAR(2)         NOT NULL,  -- 01-登录 02-交易 03-查
询...
    EVENT_TYPE      VARCHAR(50)     NOT NULL,  -- 具体事件代码
    EVENT_DESC      VARCHAR(200),

    -- 操作者信息
    USER_ID         VARCHAR(20)     NOT NULL,
    USER_NAME       VARCHAR(100),
    USER_IP         VARCHAR(40),
    USER_DEVICE     VARCHAR(200),

    -- 操作对象
    OBJ_TYPE        CHAR(2),                   -- 01-表 02-记录 03-文
件...
    OBJ_NAME        VARCHAR(100),              -- 对象名称
    OBJ_KEY         VARCHAR(200),              -- 对象主键值

    -- 操作详情
    ACTION_TYPE     CHAR(1),                   -- I-插入 U-更新 D-删除 S-查
询
    BEFORE_IMG      CLOB(32K),                 -- 修改前数据(JSON)
    AFTER_IMG       CLOB(32K),                 -- 修改后数据(JSON)

    -- 交易信息
    TRANS_REF_NO    VARCHAR(50),
    BUSINESS_TYPE   VARCHAR(20),
```

```
    -- 结果
    ACTION_RESULT   CHAR(1),                    -- 0-成功 1-失败
    ERROR_CODE      VARCHAR(10),
    ERROR_MSG       VARCHAR(500),

    -- 完整性保护
    HASH_VALUE      VARCHAR(64),                -- SHA-256哈希

    CONSTRAINT PK_AUDIT_LOG PRIMARY KEY (AUDIT_ID)
);

-- 分区表(按月分区)
CREATE INDEX IDX_AUDIT_LOG_TS ON AUDIT_LOG(AUDIT_TS);
CREATE INDEX IDX_AUDIT_LOG_USER ON AUDIT_LOG(USER_ID, AUDIT_TS);
CREATE INDEX IDX_AUDIT_LOG_OBJ ON AUDIT_LOG(OBJ_NAME, OBJ_KEY);
CREATE INDEX IDX_AUDIT_LOG_TYPE ON AUDIT_LOG(AUDIT_TYPE, AUDIT_TS);

-- 创建触发器自动生成审计日志
CREATE TRIGGER TRG_CUSTMAST_AUDIT
AFTER INSERT OR UPDATE OR DELETE ON CUSTMAST
REFERENCING NEW ROW AS N OLD ROW AS O
FOR EACH ROW
BEGIN
    DECLARE AUDIT_ID CHAR(32);
    DECLARE ACTION_TYPE CHAR(1);
    DECLARE BEFORE_IMG CLOB(32K);
    DECLARE AFTER_IMG CLOB(32K);

    SET AUDIT_ID = 'AUD' || HEX(GENERATE_UNIQUE());

    IF INSERTING THEN
        SET ACTION_TYPE = 'I';
        SET BEFORE_IMG = NULL;
        SET AFTER_IMG = JSON_OBJECT(
            'CUST_ID' VALUE N.CUST_ID,
            'CUST_NAME' VALUE N.CUST_NAME,
            'ID_NO' VALUE N.ID_NO
        );
    ELSEIF UPDATING THEN
        SET ACTION_TYPE = 'U';
        SET BEFORE_IMG = JSON_OBJECT(
            'CUST_ID' VALUE O.CUST_ID,
            'CUST_NAME' VALUE O.CUST_NAME,
            'ID_NO' VALUE O.ID_NO
        );
```

```
        SET AFTER_IMG = JSON_OBJECT(
            'CUST_ID' VALUE N.CUST_ID,
            'CUST_NAME' VALUE N.CUST_NAME,
            'ID_NO' VALUE N.ID_NO
        );
    ELSEIF DELETING THEN
        SET ACTION_TYPE = 'D';
        SET BEFORE_IMG = JSON_OBJECT(
            'CUST_ID' VALUE O.CUST_ID,
            'CUST_NAME' VALUE O.CUST_NAME,
            'ID_NO' VALUE O.ID_NO
        );
        SET AFTER_IMG = NULL;
    END IF;

    INSERT INTO AUDIT_LOG (
        AUDIT_ID, AUDIT_TYPE, EVENT_TYPE, EVENT_DESC,
        USER_ID, USER_IP,
        OBJ_TYPE, OBJ_NAME, OBJ_KEY,
        ACTION_TYPE, BEFORE_IMG, AFTER_IMG,
        ACTION_RESULT
    ) VALUES (
        AUDIT_ID, '03', 'CUST_UPDATE', '客户信息变更',
        USER, CLIENT_IP(),
        '02', 'CUSTMAST', COALESCE(N.CUST_ID, O.CUST_ID),
        ACTION_TYPE, BEFORE_IMG, AFTER_IMG,
        '0'
    );
END;
```

7.2.3 审计日志保护

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('AUDIT');

// ====================================================================
// 程序: AUDITPROTECT
// 描述: 审计日志完整性保护
// ====================================================================

// ====================================================================
// 计算审计日志哈希链
// ====================================================================
```

```
dcl-proc CalculateAuditHash export;
    dcl-pi *n char(64);
        auditId     char(32) const;
        auditTs     timestamp const;
        userId      varchar(20) const;
        eventType   varchar(50) const;
        beforeImg   clob(32K) const;
        afterImg    clob(32K) const;
        prevHash    char(64) const;
    end-pi;

    dcl-s hashInput varchar(50000);
    dcl-s hashValue char(64);

    // 构建哈希输入
    hashInput = auditId +
                %char(auditTs) +
                userId +
                eventType +
                %subst(%str(beforeImg) : 1 : %len(%str(beforeImg))) +
                %subst(%str(afterImg) : 1 : %len(%str(afterImg))) +
                prevHash;

    // 计算SHA-256哈希
    exec SQL
        SELECT LOWER(HEX(HASH(:hashInput, 256)))
        INTO :hashValue
        FROM SYSIBM.SYSDUMMY1;

    return hashValue;
end-proc;

// ============================================================
// 验证审计日志完整性
// ============================================================
dcl-proc VerifyAuditIntegrity export;
    dcl-pi *n ind;
        startDate   packed(8) const;
        endDate     packed(8) const;
        outResult   char(200);
    end-pi;

    dcl-s verifyInd ind inz(*on);
    dcl-s prevHash char(64);

    // 声明游标遍历审计日志
```

```
exec SQL
    DECLARE AUDIT_CUR CURSOR FOR
    SELECT AUDIT_ID, AUDIT_TS, USER_ID, EVENT_TYPE,
           BEFORE_IMG, AFTER_IMG, HASH_VALUE
    FROM AUDIT_LOG
    WHERE DATE(AUDIT_TS) BETWEEN :startDate AND :endDate
    ORDER BY AUDIT_TS
    FOR READ ONLY;

exec SQL OPEN AUDIT_CUR;

dou SQLCODE <> 0;
    dcl-ds auditRec qualified;
        auditId     char(32);
        auditTs     timestamp;
        userId      varchar(20);
        eventType   varchar(50);
        beforeImg   clob(32K);
        afterImg    clob(32K);
        hashValue   char(64);
    end-ds;

    exec SQL FETCH AUDIT_CUR INTO :auditRec;

    if SQLCODE = 0;
        // 验证当前记录哈希
        dcl-s calcHash char(64);
        calcHash = CalculateAuditHash(
            auditRec.auditId :
            auditRec.auditTs :
            auditRec.userId :
            auditRec.eventType :
            auditRec.beforeImg :
            auditRec.afterImg :
            prevHash
        );

        if calcHash <> auditRec.hashValue;
            verifyInd = *off;
            outResult = '哈希验证失败: ' + auditRec.auditId;
            leave;
        endif;

        prevHash = auditRec.hashValue;
    endif;
enddo;
```

```
    exec SQL CLOSE AUDIT_CUR;

    if verifyInd;
        outResult = '审计日志完整性验证通过';
    endif;

    return verifyInd;
end-proc;
```

7.3 监管报送

7.3.1 报送数据架构

```
-- ============================================================
-- 监管报送数据表
-- ============================================================

-- 一、EAST 5.0报送
-- 1. 个人信贷业务借据表
CREATE TABLE S_CUST_LOAN (
    DATA_ID         CHAR(32)        NOT NULL,
    REPORT_DATE     DECIMAL(8, 0)   NOT NULL,

    -- 借据信息
    DUE_BILL_NO     VARCHAR(100)    NOT NULL,
    CONTRACT_NO     VARCHAR(100),

    -- 客户信息
    CUST_ID         CHAR(20),
    CUST_NAME       VARCHAR(200),
    CERT_TYPE       VARCHAR(10),
    CERT_NO         VARCHAR(100),

    -- 借据信息
    LOAN_AMT        DECIMAL(19, 4),
    BALANCE         DECIMAL(19, 4),
    LOAN_RATE       DECIMAL(9, 6),
    LOAN_RATE_TYPE  VARCHAR(10),
    LOAN_DATE       DECIMAL(8, 0),
    MATURITY_DATE   DECIMAL(8, 0),

    -- 五级分类
```

```
    CLASS_5         VARCHAR(10),

    -- 报送状态
    REPORT_STATUS   CHAR(1)         DEFAULT '0',
    CHECK_RESULT    VARCHAR(500),

    CREATE_TS       TIMESTAMP       DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT PK_S_CUST_LOAN PRIMARY KEY (DATA_ID)
);

-- 2. 对公信贷业务借据表
CREATE TABLE S_CORP_LOAN (
    DATA_ID         CHAR(32)        NOT NULL,
    REPORT_DATE     DECIMAL(8, 0)   NOT NULL,

    DUE_BILL_NO     VARCHAR(100)    NOT NULL,
    CONTRACT_NO     VARCHAR(100),

    CUST_ID         CHAR(20),
    CUST_NAME       VARCHAR(200),
    ORG_CODE        VARCHAR(50),

    LOAN_AMT        DECIMAL(19, 4),
    BALANCE         DECIMAL(19, 4),
    LOAN_RATE       DECIMAL(9, 6),

    CLASS_5         VARCHAR(10),

    REPORT_STATUS   CHAR(1)         DEFAULT '0',

    CREATE_TS       TIMESTAMP       DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT PK_S_CORP_LOAN PRIMARY KEY (DATA_ID)
);

-- 二、人行大集中报送
CREATE TABLE S_PBOC_REPORT (
    REPORT_ID       CHAR(32)        NOT NULL,
    REPORT_DATE     DECIMAL(8, 0)   NOT NULL,
    REPORT_TYPE     CHAR(4)         NOT NULL,  -- A1411-资产负债表等

    ITEM_CODE       VARCHAR(20)     NOT NULL,
    ITEM_NAME       VARCHAR(100),
    CURRENCY_CD     CHAR(3),
```

```
    BALANCE         DECIMAL(19, 2),

    REPORT_STATUS   CHAR(1)         DEFAULT '0',

    CONSTRAINT PK_S_PBOC_REPORT PRIMARY KEY (REPORT_ID)
);

-- 三、外管局报送
CREATE TABLE S_SAFE_REPORT (
    REPORT_ID       CHAR(32)        NOT NULL,
    REPORT_DATE     DECIMAL(8, 0)   NOT NULL,

    REPORT_TYPE     CHAR(4),                    -- 2101-账户信息 2102-收支
余...
    BUSINESS_NO     VARCHAR(50),

    CUST_NAME       VARCHAR(200),
    ORG_CODE        VARCHAR(50),

    FOREIGN_PAY_AMT DECIMAL(19, 2),
    FOREIGN_RECV_AMT DECIMAL(19, 2),

    COUNTRY_CD      CHAR(3),
    TXN_CODE        VARCHAR(10),

    REPORT_STATUS   CHAR(1)         DEFAULT '0',

    CONSTRAINT PK_S_SAFE_REPORT PRIMARY KEY (REPORT_ID)
);
```

7.3.2 报送数据生成程序

```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('REPORT');

// ============================================================
// 程序: GENREPORT
// 描述: 监管报送数据生成
// ============================================================

// ============================================================
// 生成EAST个人信贷报送数据
// ============================================================
```

```
dcl-proc GenerateEASTCustLoan export;
    dcl-pi *n;
        reportDate  packed(8) const;
        outRtnCode  char(2);
        outRtnMsg   char(200);
    end-pi;

    dcl-s genCnt packed(10);

    outRtnCode = '00';

    // 清空当日数据
    exec SQL
        DELETE FROM S_CUST_LOAN WHERE REPORT_DATE = :reportDate;

    // 生成报送数据
    exec SQL
        INSERT INTO S_CUST_LOAN (
            DATA_ID, REPORT_DATE, DUE_BILL_NO, CONTRACT_NO,
            CUST_ID, CUST_NAME, CERT_TYPE, CERT_NO,
            LOAN_AMT, BALANCE, LOAN_RATE, LOAN_RATE_TYPE,
            LOAN_DATE, MATURITY_DATE, CLASS_5
        )
        SELECT
            'ECL' || CHAR(NEXT VALUE FOR EAST_DATA_SEQ),
            :reportDate,
            L.LOAN_NO,
            L.CONTRACT_NO,
            L.CUST_ID,
            C.CUST_NAME,
            C.ID_TYPE,
            C.ID_NO,
            L.CONTRACT_AMT,
            L.LOAN_BAL,
            L.INT_RATE,
            L.INT_RATE_TYPE,
            L.FIRST_DISB_DT,
            L.MATURITY_DT,
            CASE L.CLASS_LEVEL
                WHEN '1' THEN '正常'
                WHEN '2' THEN '关注'
                WHEN '3' THEN '次级'
                WHEN '4' THEN '可疑'
                WHEN '5' THEN '损失'
            END
        FROM LOANMAST L
```

```
        JOIN CUSTMAST C ON L.CUST_ID = C.CUST_ID
        WHERE L.LOAN_STAT NOT IN ('01', '02', '99')  -- 排除申请、审批、已
删除
          AND L.CUST_TYPE = '1';  -- 个人客户

    GET DIAGNOSTICS genCnt = ROW_COUNT;

    // 数据校验
    exec SQL
        UPDATE S_CUST_LOAN
        SET CHECK_RESULT = CHECK_RESULT || '金额为空;'
        WHERE REPORT_DATE = :reportDate
          AND (LOAN_AMT IS NULL OR LOAN_AMT = 0);

    exec SQL
        UPDATE S_CUST_LOAN
        SET CHECK_RESULT = CHECK_RESULT || '证件号码为空;'
        WHERE REPORT_DATE = :reportDate
          AND (CERT_NO IS NULL OR CERT_NO = '');

    // 更新报送状态
    exec SQL
        UPDATE S_CUST_LOAN
        SET REPORT_STATUS = '1'  -- 待报送
        WHERE REPORT_DATE = :reportDate
          AND (CHECK_RESULT IS NULL OR CHECK_RESULT = '');

    outRtnMsg = 'EAST个人信贷报送数据生成: ' + %char(genCnt) + '条';

end-proc;

// ====================================================================
// 生成人行大集中报表
// ====================================================================
dcl-proc GeneratePBOCReport export;
    dcl-pi *n;
        reportDate  packed(8) const;
        reportType  char(4) const;      -- A1411, A1412, A1464...
        outRtnCode  char(2);
        outRtnMsg   char(200);
    end-pi;

    outRtnCode = '00';

    // 根据报表类型调用不同生成逻辑
    select;
```

```
        when reportType = 'A1411';  // 资产负债项目统计表
            GenerateA1411(reportDate);
        when reportType = 'A1412';  // 贷款分行业统计表
            GenerateA1412(reportDate);
        when reportType = 'A1464';  // 贷款五级分类统计表
            GenerateA1464(reportDate);
        other;
            outRtnCode = '01';
            outRtnMsg = '未知报表类型: ' + reportType;
            return;
    endsl;

    outRtnMsg = '人行报表' + reportType + '生成完成';

end-proc;

// ====================================================================
// A1411报表: 资产负债项目统计表
// ====================================================================
dcl-proc GenerateA1411;
    dcl-pi *n;
        reportDate  packed(8) const;
    end-pi;

    // 61 各项贷款
    exec SQL
        INSERT INTO S_PBOC_REPORT
        SELECT
            'PBC' || CHAR(NEXT VALUE FOR PBOC_REPORT_SEQ),
            :reportDate,
            'A1411',
            '61',
            '各项贷款',
            'CNY',
            SUM(LOAN_BAL),
            '0'
        FROM LOANMAST
        WHERE ACCT_STAT = '0'
          AND LOAN_STAT = '04';  // 发放状态

    // 62 企业贷款
    exec SQL
        INSERT INTO S_PBOC_REPORT
        SELECT
            'PBC' || CHAR(NEXT VALUE FOR PBOC_REPORT_SEQ),
            :reportDate,
```

```
            'A1411',
            '62',
            '其中：企业贷款',
            'CNY',
            SUM(LOAN_BAL),
            '0'
        FROM LOANMAST
        WHERE ACCT_STAT = '0'
          AND LOAN_STAT = '04'
          AND CUST_TYPE = '2';  // 企业客户

    // 71 各项存款
    exec SQL
        INSERT INTO S_PBOC_REPORT
        SELECT
            'PBC' || CHAR(NEXT VALUE FOR PBOC_REPORT_SEQ),
            :reportDate,
            'A1411',
            '71',
            '各项存款',
            'CNY',
            SUM(ACCT_BAL),
            '0'
        FROM ACCTMAST
        WHERE ACCT_STAT = '0'
          AND PROD_TYPE IN ('01', '02', '03');  // 存款类

end-proc;
```

7.4 PCI DSS合规

7.4.1 PCI DSS要求实现

```
┌──────────────────────────────────────────────────────────────────────────────
──────┐
│                           PCI DSS合规框
架                                        │
├──────────────────────────────────────────────────────────────────────────────
──────┤
│
      │
│  要求1: 安装并维护防火墙配置以保护持卡人数据
│
```

```
│
┌──────────────────────────────────────────────────────────────────────────────┐
  │
│   │  - 网络边界防火墙隔离卡支付环境(CDE)
│    │
│   │  - 内部防火墙分隔生产与开发/测试环境
│    │
│   │  - 防火墙规则定期审查与更新
│    │
│
└──────────────────────────────────────────────────────────────────────────────┘
  │
│
      │
│  要求3: 保护存储的持卡人数据
│
│
┌──────────────────────────────────────────────────────────────────────────────┐
  │
│   │  - PAN数据使用强加密存储(AES-256)
│    │
│   │  - 卡号显示时仅显示前6位和后4位
│    │
│   │  - 加密密钥使用HSM保护
│    │
│   │  - 密钥定期轮换(至少每年一次)
│    │
│
└──────────────────────────────────────────────────────────────────────────────┘
  │
│
      │
│  要求4: 加密传输中的持卡人数据
│
│
┌──────────────────────────────────────────────────────────────────────────────┐
  │
│   │  - 使用TLS 1.2或更高版本
│    │
│   │  - 禁用弱加密算法(SSLv3, TLS 1.0/1.1)
│    │
│   │  - 证书有效性验
证                                                          │   │
│
└──────────────────────────────────────────────────────────────────────────────┘
  │
```

```
|
      |
|  要求8: 识别并验证对系统组件的访问
|
|
┌──────────────────────────────────────────────────────────────────────────────┐
  |
|  |  - 唯一用户
ID                                                              |   |
|  |  - 强密码策略(最小长度12位，复杂度要求)
|   |
|  |  - 多因素认证
(MFA)                                                           |   |
|  |  - 会话超时控制(15分钟无操作)
|   |
|
└──────────────────────────────────────────────────────────────────────────────┘
  |
|
      |
|  要求10: 跟踪并监控所有对网络资源和持卡人数据的访问                          |
|
┌──────────────────────────────────────────────────────────────────────────────┐
  |
|  |  - 全面的审计日
志                                                          |   |
|  |  - 日志完整性保护(哈希链)
|   |
|  |  - 日志保留至少12个月
|   |
|  |  - 实时监控与告
警                                                          |   |
|
└──────────────────────────────────────────────────────────────────────────────┘
  |
|
      |
└──────────────────────────────────────────────────────────────────────────────
──────┘
```

7.4.2 卡号安全处理

```
**free
ctl-opt option(*srcstmt : *nodebugio);
```

```
ctl-opt dftactgrp(*no) actgrp('PCI');

// ============================================================
// 程序: PCIDSS
// 描述: PCI DSS卡号安全处理
// ============================================================

// ============================================================
// 验证PAN有效性(Luhn校验)
// ============================================================
dcl-proc ValidatePAN export;
    dcl-pi *n ind;
        pan         char(19) const;
    end-pi;

    // 使用前文定义的LuhnCheck
    return LuhnCheck(pan);
end-proc;

// ============================================================
// 显示PAN(脱敏)
// 仅显示前6位和后4位
// ============================================================
dcl-proc DisplayPAN export;
    dcl-pi *n varchar(19);
        pan         char(19) const;
    end-pi;

    dcl-s len packed(2);
    dcl-s display varchar(19);

    len = %len(%trim(pan));

    if len < 10;
        return 'INVALID';
    endif;

    // 格式: 前6位 + * + 后4位
    display = %subst(pan : 1 : 6) +
              %subst('******************' : 1 : len - 10) +
              %subst(pan : len - 3 : 4);

    return display;
end-proc;

// ============================================================
```

```
// 安全存储PAN(加密)
// ================================================================
dcl-proc SecureStorePAN export;
    dcl-pi *n varchar(1000);
        pan         char(19) const;
    end-pi;

    dcl-ds encData likeds(EncryptedField);
    dcl-s encryptedJSON varchar(1000);

    // 使用HSM加密
    encData = EncryptField(pan : 'PCI_HSM_KEY' : 'AES256');

    // 转换为JSON存储
    encryptedJSON = '{"ct":"' + encData.cipherText +
                    '","nonce":"' + encData.nonce +
                    '","kv":"' + encData.keyVersion + '"}';

    return encryptedJSON;
end-proc;

// ================================================================
// 令牌化(Tokenization)
// 用随机令牌替代真实PAN
// ================================================================
dcl-proc TokenizePAN export;
    dcl-pi *n char(19);
        pan         char(19) const;
    end-pi;

    dcl-s token char(19);
    dcl-s tokenSeq packed(15);

    // 生成唯一令牌序列
    exec SQL
        SELECT NEXT VALUE FOR PAN_TOKEN_SEQ INTO :tokenSeq
        FROM SYSIBM.SYSDUMMY1;

    // 构建令牌(保持最后一位校验位相同)
    token = '4' + %editc(tokenSeq : 'X') + %subst(pan : 19 : 1);

    // 保存PAN与令牌映射
    exec SQL
        INSERT INTO PAN_TOKEN_MAP (
            TOKEN, PAN, TOKEN_TS, EXPIRY_TS
        ) VALUES (
```

```
            :token,
            :SecureStorePAN(pan),
            CURRENT_TIMESTAMP,
            CURRENT TIMESTAMP + 7 YEARS
        );

    return token;
end-proc;

// ===================================================================
// 解令牌化(仅授权系统可调用)
// ===================================================================
dcl-proc DetokenizePAN export;
    dcl-pi *n char(19);
        token       char(19) const;
        authToken   char(32) const;     // 授权令牌
    end-pi;

    dcl-s pan char(19);
    dcl-s encryptedPAN varchar(1000);

    // 验证授权
    if not VerifyPCIAccess(authToken : 'DETOKENIZE');
        // 记录安全事件
        LogSecurityEvent('UNAUTHORIZED_DETOKENIZE_ATTEMPT' : token);
        return '';
    endif;

    // 查询加密PAN
    exec SQL
        SELECT PAN INTO :encryptedPAN
        FROM PAN_TOKEN_MAP
        WHERE TOKEN = :token
          AND EXPIRY_TS > CURRENT_TIMESTAMP;

    if SQLCODE <> 0;
        return '';
    endif;

    // 解密PAN
    dcl-ds encData likeds(EncryptedField);
    // 解析JSON并解密...

    return pan;
end-proc;
```

第八章 性能优化与高可用

8.1 性能优化策略

8.1.1 数据库优化

```
-- ================================================================
-- 数据库性能优化最佳实践
-- ================================================================

-- 1. 索引优化
-- 为频繁查询的列创建索引
CREATE INDEX IDX_ACCTMAST_BALANCE ON ACCTMAST(ACCT_BAL)
    WHERE ACCT_BAL > 0;

-- 复合索引设计原则：等值查询列在前，范围查询列在后
CREATE INDEX IDX_TRANS_QUERY ON ACCTTRANS(TRANS_DT, ACCT_NO,
TRANS_TYPE);

-- 2. 分区表
-- 按时间分区的大表
CREATE TABLE ACCTTRANS_2026 (
    LIKE ACCTTRANS INCLUDING COLUMN DEFAULTS
) PARTITION BY RANGE(TRANS_DT) (
    PARTITION P202601 STARTING(20260101) ENDING(20260131),
    PARTITION P202602 STARTING(20260201) ENDING(20260229),
    PARTITION P202603 STARTING(20260301) ENDING(20260331),
    PARTITION P_MAX STARTING(20260401)
);

-- 3. 物化视图(汇总查询加速)
CREATE TABLE DAILY_BALANCE_SUMMARY AS (
    SELECT
        TRANS_DT,
        ACCT_NO,
        SUM(CASE WHEN DR_CR_FLAG = 'D' THEN TRANS_AMT ELSE 0 END) AS
DR_AMT,
        SUM(CASE WHEN DR_CR_FLAG = 'C' THEN TRANS_AMT ELSE 0 END) AS
CR_AMT,
        COUNT(*) AS TRANS_CNT
    FROM ACCTTRANS
```

```
    GROUP BY TRANS_DT, ACCT_NO
) DATA INITIALLY DEFERRED REFRESH DEFERRED;

-- 4. 查询优化提示
-- 使用优化器提示
SELECT /*+ USE INDEX(ACCTMAST, IDX_ACCTMAST_CUST) */
    ACCT_NO, ACCT_BAL
FROM ACCTMAST
WHERE CUST_ID = '1234567890';

-- 5. 统计信息更新
-- 定期更新统计信息
CALL SYSPROC.ADMIN_CMD('RUNSTATS ON TABLE ACCTMAST WITH DISTRIBUTION');
```

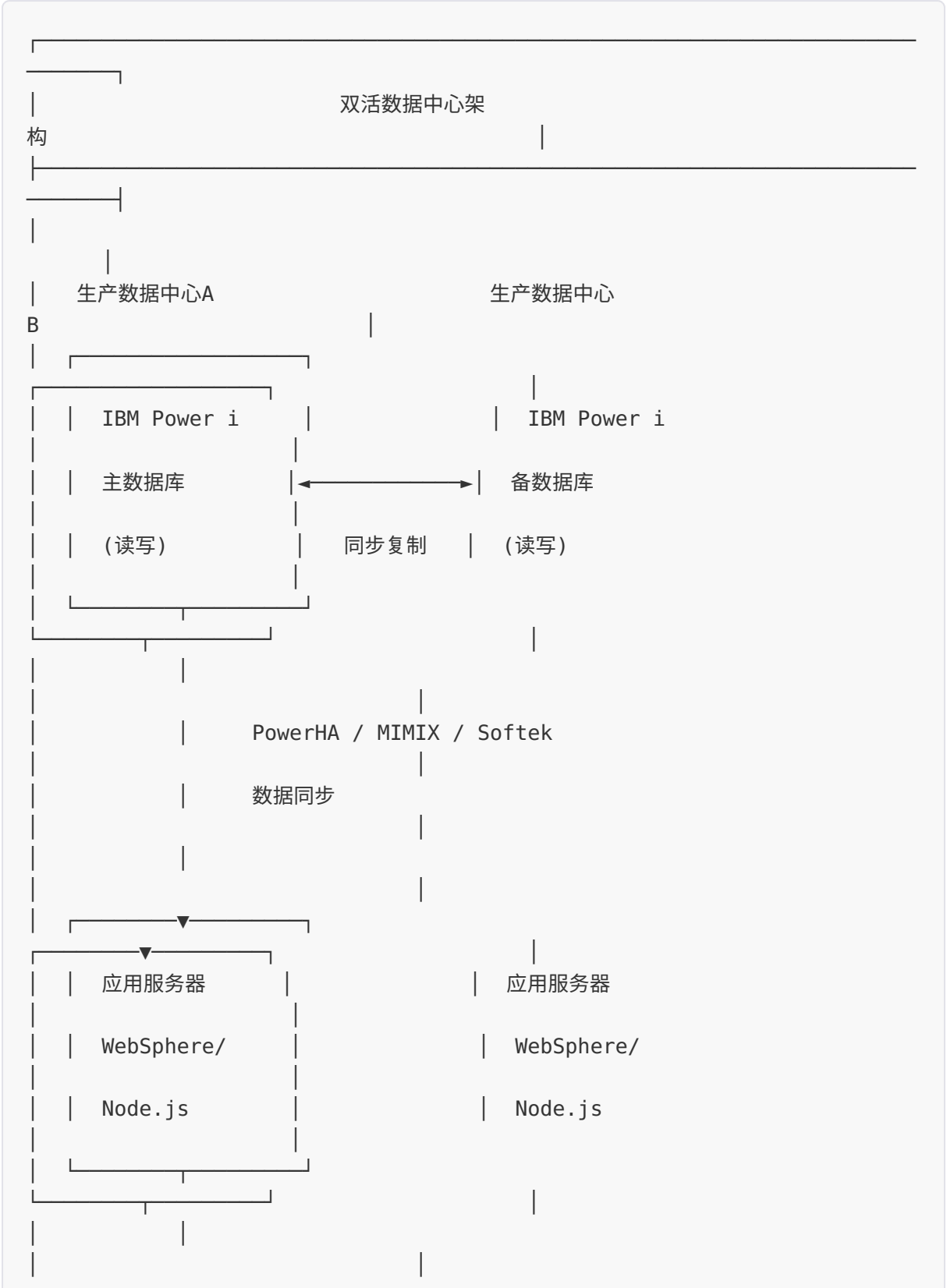
8.1.2 RPG程序优化

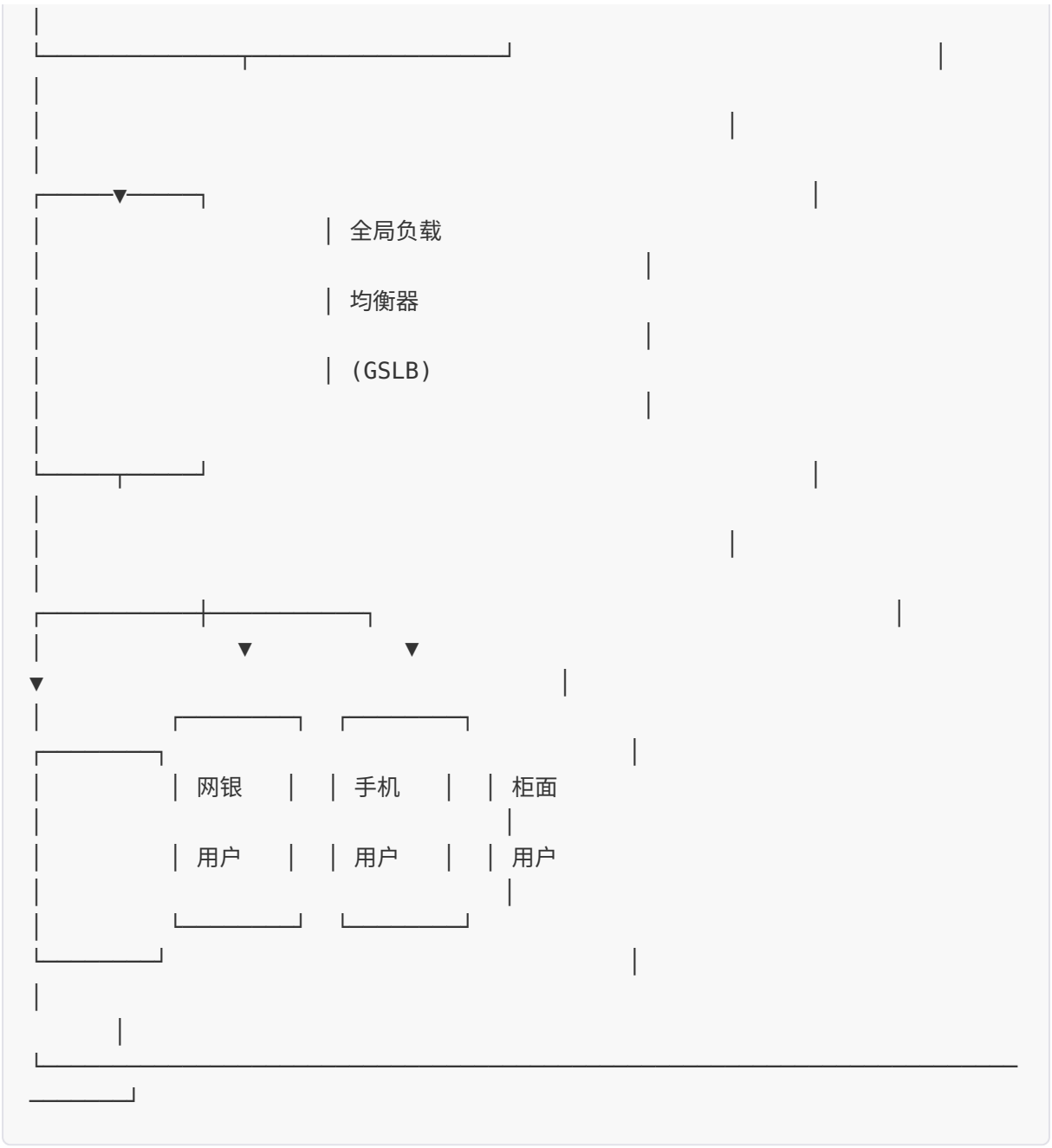
```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('OPTIMIZE');
ctl-opt nomain;  // 模块化编译

// ========================================================================
// RPG性能优化最佳实践
// ========================================================================

// 1. 使用更高效的文件访问方式
// 避免使用READ/WRITE，优先使用CHAIN/UPDATE

// 低效方式:
// read ACCTMAST;
// dow not %eof;
//   if ACCT_NO = searchAcct;
//     ...
//   endif;
//   read ACCTMAST;
// enddo;

// 高效方式:
chain searchAcct ACCTMAST;
if %found;
    // 处理
endif;

// 2. 使用SQL替代逻辑读取
```

```
// 低效: 使用RPG循环处理大量记录
// 高效: 使用SQL集合操作

// 3. 批量处理减少I/O
// 使用%occur处理数组批量操作
dcl-ds transArray likeds(TransData) dim(1000);
dcl-s arrayIdx packed(10);

// 批量读取
exec SQL
    DECLARE TRANS_CUR CURSOR FOR
    SELECT * FROM ACCTTRANS WHERE TRANS_DT = :today;

exec SQL OPEN TRANS_CUR;

arrayIdx = 0;
dou SQLCODE <> 0;
    arrayIdx = arrayIdx + 1;
    exec SQL FETCH TRANS_CUR INTO :transArray(arrayIdx);
    if arrayIdx = 1000 or SQLCODE <> 0;
        // 批量处理
        ProcessBatch(transArray : arrayIdx);
        arrayIdx = 0;
    endif;
enddo;

exec SQL CLOSE TRANS_CUR;

// 4. 内存管理优化
// 使用动态内存分配
dcl-s ptr pointer;
dcl-s dataLen packed(10);

ptr = %alloc(10000);
// 使用内存...
dealloc(n) ptr;

// 5. 字符串处理优化
// 使用%len而不是%trimr循环
// 使用%scan而不是手动查找

// 6. 避免不必要的数据类型转换
// 保持数据类型一致
```

8.2 高可用架构

8.2.1 双活数据中心架构





8.2.2 高可用配置

```
-- ================================================================
-- IBM i 高可用配置
-- ================================================================

-- 1. 日志镜像配置
-- 启用日志镜像确保日志写入多个磁盘
CHGJRN JRN(QSYS/QDBJRN) JRNRCV(*GEN) MIRROR(*YES);

-- 2. 磁盘保护配置
```

```
-- RAID 6配置(可承受2块磁盘故障)
-- SST (System Service Tools)配置

-- 3. 备份配置
-- 使用BRMS (Backup Recovery and Media Services)
-- 在线备份策略

-- 4. 日志审计点
-- 创建日志审计点用于快速恢复

-- 5. 闪存拷贝(FlashCopy)
-- 使用存储级快照实现快速备份
```

第九章 现代化改造

9.1 API化改造

9.1.1 REST API设计

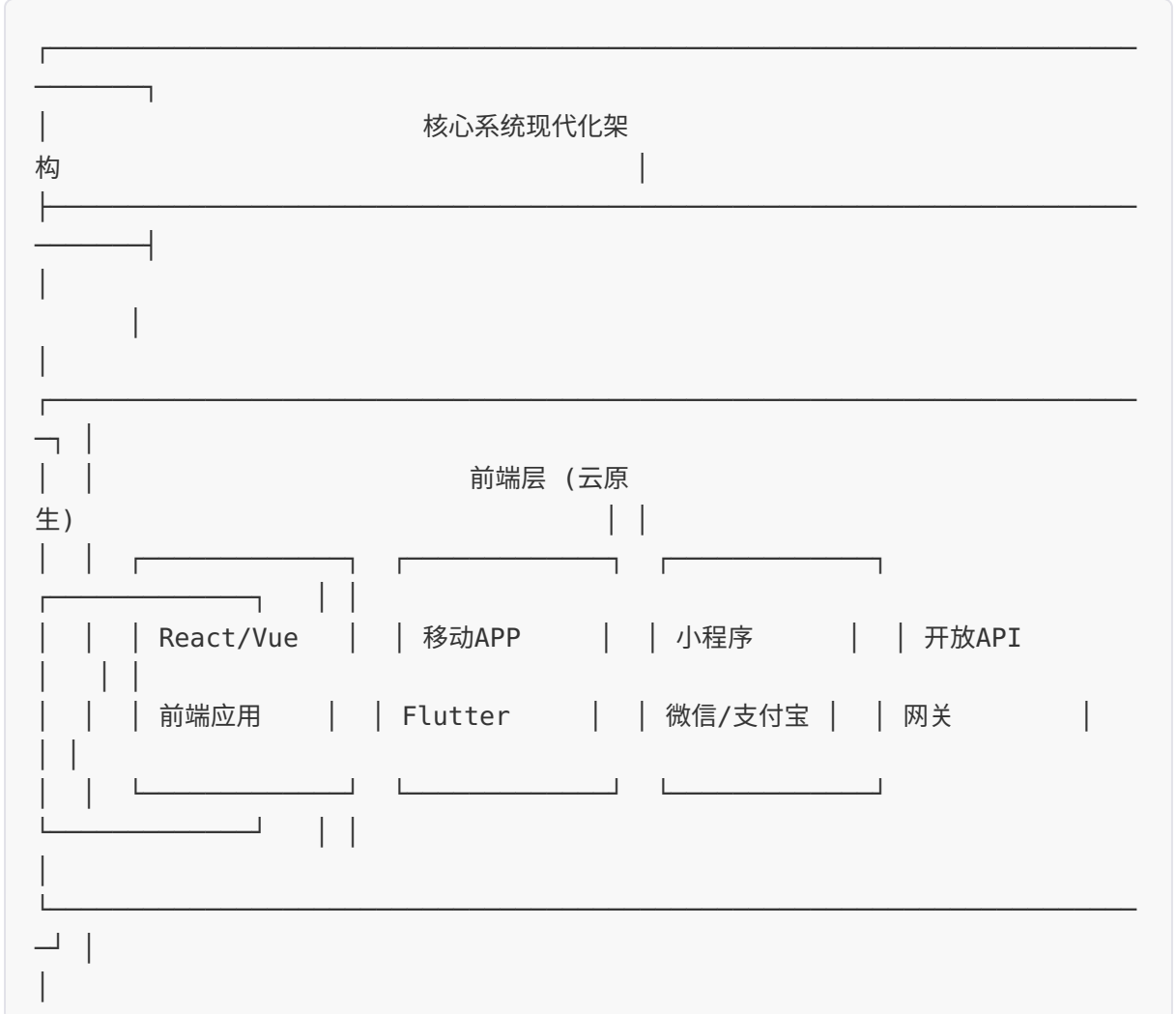
```
**free
ctl-opt option(*srcstmt : *nodebugio);
ctl-opt dftactgrp(*no) actgrp('API');

// ============================================================
// 程序: ACCOUNTAPI
// 描述: 账户服务REST API (使用ILEastic框架)
// ============================================================

/copy ileastic/qrpglesrc,ileastic
/copy jsonxml/qrpglesrc,jsonparser

// ============================================================
// 获取账户余额 API
// GET /api/v1/accounts/{accountNo}/balance
// ============================================================
dcl-proc getAccountBalance export;
    dcl-pi *n;
        request     likeds(REQUESTDS);
        response    likeds(RESPONSEDS);
    end-pi;
```

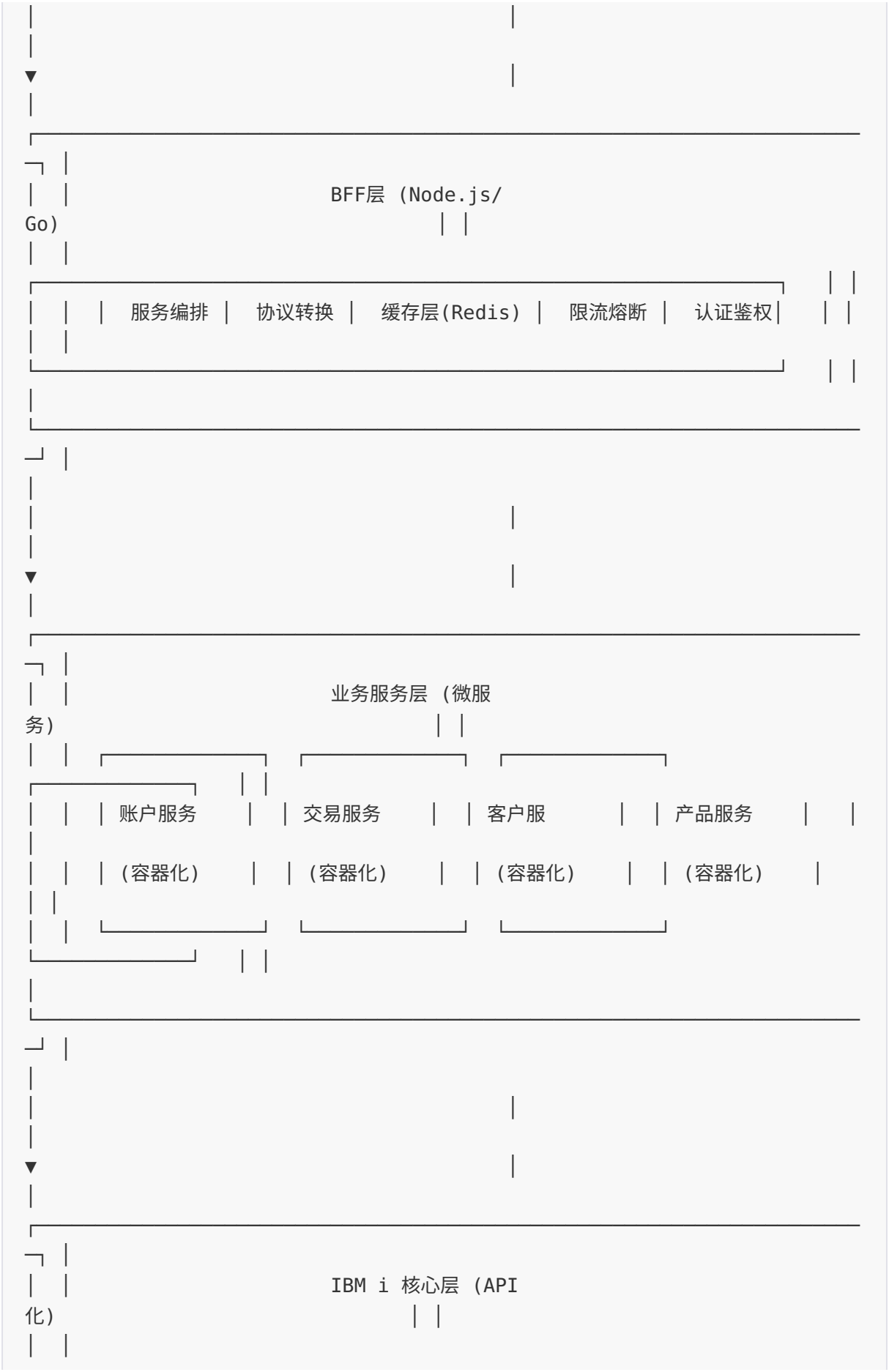
```
    dcl-s acctNo char(32);
    dcl-ds acctData likeds(ACCTMAST_T);
    dcl-s jsonResponse varchar(2000);

    // 从URL路径获取账号
    acctNo = il_getPathParameter(request : 'accountNo');

    // 查询数据库
    exec SQL
        SELECT ACCT_NO, CUST_ID, ACCT_BAL, AVAIL_BAL, ACCT_STAT
        INTO :acctData
        FROM ACCTMAST
        WHERE ACCT_NO = :acctNo;

    if SQLCODE <> 0;
        // 账号不存在
        response.status = 404;
        jsonResponse = '{"error":"Account not found"}';
        il_responseWrite(response : jsonResponse);
        return;
    endif;

    // 构建JSON响应
    jsonResponse = '{"accountNo":"' + %trim(acctData.ACCT_NO) + '"' +
                   ',"balance":' + %char(acctData.ACCT_BAL) +
                   ',"availableBalance":' + %char(acctData.AVAIL_BAL) +
                   ',"status":"' + acctData.ACCT_STAT + '"' +
                   ',"currency":"CNY"}';

    response.contentType = 'application/json';
    il_responseWrite(response : jsonResponse);

end-proc;

// ============================================================
// 账户转账 API
// POST /api/v1/accounts/transfer
// Request Body: {"fromAccount":"xxx","toAccount":"yyy","amount":
100.00}
// ============================================================
dcl-proc postAccountTransfer export;
    dcl-pi *n;
        request     likeds(REQUESTDS);
        response    likeds(RESPONSEDS);
    end-pi;
```

```
dcl-s requestBody varchar(2000);
dcl-ds transferReq qualified;
    fromAcct    varchar(32);
    toAcct      varchar(32);
    amount      packed(19:2);
    currency    varchar(3);
    memo        varchar(200);
end-ds;
dcl-ds transReq likeds(TransReq);
dcl-ds transResp likeds(TransResp);
dcl-s jsonResponse varchar(2000);

// 读取请求体
requestBody = il_requestGetBody(request);

// 解析JSON
ParseTransferRequest(requestBody : transferReq);

// 参数校验
if transferReq.fromAcct = '' or transferReq.toAcct = ''
   or transferReq.amount <= 0;
    response.status = 400;
    jsonResponse = '{"error":"Invalid request parameters"}';
    il_responseWrite(response : jsonResponse);
    return;
endif;

// 构建交易请求
transReq.transRefNo = GenerateTransRef();
transReq.transType = '1003';  // 转账
transReq.acctNo = transferReq.fromAcct;
transReq.oppAcctNo = transferReq.toAcct;
transReq.transAmt = transferReq.amount;
transReq.ccyCd = transferReq.currency;
transReq.drCrFlag = 'D';
transReq.transChnl = '99';  // API渠道
transReq.transSummary = transferReq.memo;

// 调用核心交易处理
transResp = ProcessDeposit(transReq);

// 构建响应
if transResp.rtnCode = '00';
    response.status = 200;
    jsonResponse = '{"success":true' +
```

```
                        ","transactionId":"' + transResp.transSeq + '"'
+
                        ","referenceNo":"' + transReq.transRefNo + '"' +
                        ","fromBalance":' + %char(transResp.acctBal) +
                        ","timestamp":"' + %char(%timestamp()) + '"}';
    else;
        response.status = 422;
        jsonResponse = '{"success":false' +
                        ","errorCode":"' + transResp.rtnCode + '"' +
                        ","errorMessage":"' + transResp.rtnMsg + '"}';
    endif;

    response.contentType = 'application/json';
    il_responseWrite(response : jsonResponse);

end-proc;
```

9.2 微服务集成




BFF层 (Node.js/ Go)
服务编排
协议转换
缓存层(Redis)
限流熔断
认证鉴权
业务服务层 (微服务)
账户服务
(容器化)
交易服务
(容器化)
客户服
(容器化)
产品服务
(容器化)
IBM i 核心层 (API化)

```
┌─────────────────────────────────────────────────────────────────────┐  │ │
│  │  RPG IV 业务逻辑  │  SQL 数据访问  │  Db2 for i 数据库        │  │  │
│                                                                     │  │
│  │  REST API 封装    │  事务管理      │  单级存储架构            │  │  │
│                                                                     │  │
│  │                                                                  │  │ │
└─────────────────────────────────────────────────────────────────────┘  │ │
                                                                         │
└─────────────────────────────────────────────────────────────────────────
┘ │
                                                                         │
      │
└─────────────────────────────────────────────────────────────────────────
──────┘
```

附录

附录A: RPG IV编程规范

A.1 代码风格指南

```
**free
// ============================================================
// 程序: EXAMPLE
// 描述: 示例程序展示RPG IV最佳实践
// 作者: [作者名]
// 日期: [创建日期]
// 修改历史:
//   YYYY-MM-DD  修改人 修改内容描述
// ============================================================

// 控制选项
dcl-opt option(*srcstmt : *nodebugio);
dcl-opt dftactgrp(*no) actgrp('EXAMPLE');
dcl-opt thread(*concurrent);
dcl-opt nomain;  // 对于模块化处理

// ============================================================
// 全局常量定义
// ============================================================
```

```
dcl-c MAX_RECORDS      1000;
dcl-c STATUS_ACTIVE    'A';
dcl-c STATUS_INACTIVE  'I';
dcl-c COMPANY_CODE     'ABC';

// ================================================================
// 全局变量定义
// ================================================================
dcl-s g_currentDate    date;
dcl-s g_currentTime    time;
dcl-s g_userId         char(10) static;

// ================================================================
// 数据结构定义
// ================================================================

// 外部数据结构(来自物理文件)
dcl-ds customerData extname('CUSTMAST') qualified end-ds;

// 内部数据结构
dcl-ds addressData qualified template;
    street          varchar(100);
    city            varchar(50);
    state           char(2);
    zipCode         char(10);
    country         char(3);
end-ds;

// 程序状态数据结构
dcl-ds pgmStatus psds;
    pgmName         char(10)  pos(1);
    statusCode      char(10)  pos(11);
    pgmLibrary      char(10)  pos(81);
    jobName         char(10)  pos(244);
    jobUser         char(10)  pos(254);
    jobNumber       char(6)   pos(264);
end-ds;

// ================================================================
// 导出过程定义
// ================================================================

// 主入口过程
dcl-proc main export;
    // 过程实现
end-proc;
```

```
// 辅助过程
dcl-proc processCustomer;
    dcl-pi *n ind;  // 返回布尔值
        customerId  char(20) const;
        outResult   likeds(customerData);
    end-pi;

    // 过程实现
    return *on;
end-proc;
```

A.2 命名规范

类型	命名规则	示例
程序	功能缩写 + 动作	CUSTINQ, ACCTUPD, DEPTRANS
文件	描述性名称	CUSTMAST, ACCTDTL, TRANLOG
字段	驼峰式/下划线	CustName, ACCT_BAL
常量	全大写	MAX_RECORDS, STATUS_ACTIVE
过程	动词+名词	GetCustomer, UpdateBalance
变量	有意义的名称	currentBalance, customerCount

附录B: SQL优化清单

B.1 索引优化检查点

- [ ] 是否为WHERE子句中的列创建索引
- [ ] 是否为JOIN条件中的列创建索引
- [ ] 是否为ORDER BY列创建索引
- [ ] 复合索引的列顺序是否正确
- [ ] 是否存在冗余索引
- [ ] 是否定期更新统计信息

B.2 查询优化检查点

- [ ] 是否避免SELECT *
- [ ] 是否使用绑定变量
- [ ] 是否避免在索引列上使用函数
- [ ] 是否使用EXISTS替代IN(大数据量时)
- [ ] 是否避免隐式数据类型转换
- [ ] 是否使用分页查询(避免大数据量返回)

附录C: 常用API参考

C.1 IBM i系统API

```
// 获取系统信息
QUSRJOBI - 获取作业信息
QWCRDTAA - 检索数据区
QUSROBJD - 检索对象描述
QUSLJOB - 列出作业

// 日期时间处理
CEELOCT - 获取本地时间
CEEDATM - 格式化日期时间
CEEDAYS - 计算天数差

// 消息处理
QMHSNDPM - 发送程序消息
QMHRCVPM - 接收程序消息
QMHSNDM - 发送消息

// 用户空间操作
QUSCRTUS - 创建用户空间
QUSDLTUS - 删除用户空间
QUSPTRUS - 获取用户空间指针
```

C.2 加密API

```
// IBM i加密API
QC3ENCDT / Qc3EncryptData - 加密数据
```

```
QC3DECDT / Qc3DecryptData - 解密数据
QC3CALHA / Qc3CalculateHash - 计算哈希
QC3GENRN / Qc3GenerateRandomNumber - 生成随机数
```

附录D: 故障排查指南

D.1 常见错误代码

错误代码	描述	解决方案
CPF0001	通用错误	查看详细错误消息
SQL0100	记录未找到	检查查询条件
SQL0802	数据转换错误	检查数据类型匹配
SQL0901	SQL系统错误	检查SQL语句语法
RNX0100	数组索引错误	检查数组边界
RNX1211	除零错误	检查除数是否为0

D.2 调试技巧

```
// 使用DUMP操作码
if debugMode;
    dump(a);  // 输出所有变量
endif;

// 使用STRDBG命令
// STRDBG PGM(MYLIB/MYPGM) UPDPROD(*NO)

// 使用%STATUS内置函数
if %status <> 0;
    // 处理错误
endif;

// 使用%ERROR和%FOUND
chain key file;
if %error;
    // 文件错误
```

```
elseif not %found;
    // 记录未找到
endif;
```

附录E: 性能监控工具

E.1 IBM i性能工具

- **Collection Services** - 系统性能数据收集
- **Performance Explorer** - 性能数据详细分析
- **Job Watcher** - 作业级别性能分析
- **Disk Watcher** - 磁盘I/O性能分析
- **SQL Performance Center** - SQL性能监控

E.2 关键性能指标

指标	健康阈值	监控方法
CPU使用率	< 80%	WRKACTJOB
磁盘空间	< 85%	WRKDSKSTS
响应时间	< 2秒	SQL Performance Monitor
锁等待	< 5%	WRKOBJLCK
内存使用	< 90%	WRKSYSSTS

附录F: 版本控制与部署

F.1 源代码管理

```
# 使用Git管理RPG源代码
# .gitattributes配置
*.rpgle text eol=crlf
*.sql text eol=crlf
*.cl text eol=crlf
*.clle text eol=crlf
```

```
# .gitignore配置
*.evfevent
*.dtaara
*.lib
```

F.2 CI/CD流程

```
# Azure DevOps Pipeline示例
trigger:
  - main

pool:
  name: 'IBM i Agent Pool'

steps:
  - script: |
      # 编译RPG程序
      CRTRPGMOD MODULE($(library)/$(module)) SRCSTMF('src/$
(module).rpgle')
    displayName: 'Compile RPG Module'

  - script: |
      # 运行单元测试
      CALL PGM(TESTLIB/TESTSUITE) PARM('$(module)')
    displayName: 'Run Unit Tests'

  - script: |
      # 部署到测试环境
      CRTBNDRPG PGM(TESTLIB/$(module)) SRCSTMF('src/$(module).rpgle')
    displayName: 'Deploy to Test'
    condition: succeeded()
```

附录G: 参考资料

G.1 IBM官方文档

- IBM i Knowledge Center
- RPG IV Reference
- Db2 for i SQL Reference

- IBM i Security Reference

G.2 行业标准

- PCI DSS 4.0
- ISO 27001
- 巴塞尔协议III/IV
- 中国人民银行金融科技标准

G.3 推荐书籍

- 《Modernizing IBM i Applications》
- 《Programming in ILE RPG》
- 《IBM i Security Guide》
- 《SQL for IBM i》

总结

本文档全面介绍了IBM iSeries金融系统开发的最佳实践，涵盖以下核心内容:

主要章节回顾

1. **银行核心系统**: 包括客户信息管理、存款系统、贷款系统、总账系统和利息计算引擎的详细设计和代码实现。
2. **卡系统与支付**: 涵盖信用卡授权处理、借记卡交易、支付网关、跨境支付、SWIFT集成和ISO 20022实现。
3. **证券交易系统**: 包括交易撮合引擎、清算交收、风险控制和行情分发系统。
4. **保险核心系统**: 涵盖保单管理、理赔处理、精算系统和再保险。
5. **金融风控**: 包括反洗钱(AML)、KYC/CDD、风险加权资产(RWA)计算和实时监控系统。
6. **安全合规**: 涵盖数据加密、审计追踪、监管报送和PCI DSS合规。
7. **性能优化与高可用**: 包括数据库优化、RPG程序优化和高可用架构设计。
8. **现代化改造**: API化改造和微服务集成方案。

核心技术要点

- **RPG IV现代化**: 使用自由格式、模块化设计和SQL RPG
- **数据库优化**: 索引设计、分区表、物化视图
- **安全架构**: 多层加密、审计追踪、合规报送
- **高可用设计**: 双活数据中心、数据同步、故障切换

文档编写说明

本文档基于以下原则编写:
- 实用性强，提供可直接使用的代码示例
- 架构清晰，使用ASCII图表展示系统结构
- 覆盖全面，从业务分析到技术实现
- 符合监管要求，满足金融行业合规标准

文档版本: 1.0
最后更新: 2026年2月7日
文档维护: 金融科技架构团队

补充章节: 金融行业案例研究

案例一: 大型国有银行核心系统现代化改造

项目背景

某国有大型商业银行核心系统始建于上世纪九十年代，经过二十多年的运行，系统积累了超过二十亿客户账户和数十亿历史交易记录。随着金融科技的发展和客户需求的变化，原系统面临以下挑战:

1. **技术架构陈旧**: 采用传统的单体架构，系统耦合度高，新功能上线周期长
2. **性能瓶颈凸显**: 日终批处理时间过长，高峰期交易响应延迟严重
3. **运维成本高昂**: 老旧系统维护困难，对技术人员依赖度高
4. **创新能力不足**: 难以快速响应互联网银行、移动支付等新兴业务需求

改造方案

该银行采用"保留核心、外围解耦"的渐进式改造策略:

第一阶段: 核心加固与标准化(6个月)

在这一阶段，技术团队对现有的RPG程序进行了全面的代码审查和重构。通过引入自由格式RPG和现代开发规范，将原有的固定格式代码逐步迁移。同时，建立了完整的版本控制体系，使用Git管理所有源代码，实现了代码的追溯和审计。数据库层面，优化了核心业务表的索引设计，将频繁访问的数据表按照业务类型进行了合理分区。日志系统也得到了强化，引入了全链路跟踪机制，为后续的问题定位奠定了基础。

第二阶段: API化改造(12个月)

这是整个改造过程中最具挑战性的阶段。技术团队使用ILEastic框架将核心业务逻辑封装为RESTful API，同时保持原有绿屏界面的兼容性。为了确保改造过程的平滑过渡，采用了"绞杀者模式"，即新业务逐步通过API访问，旧业务仍保持原有通道。这一阶段还建立了完善的服务治理体系，包括API网关、服务注册发现、熔断限流等机制。通过引入契约测试，确保了前后端接口的兼容性。

第三阶段: 渠道层云化(18个月)

在完成核心API化后，银行将渠道层应用逐步迁移至私有云平台。手机银行、网上银行、微信银行等渠道应用采用微服务架构重新构建，实现了弹性伸缩和故障隔离。通过引入Kubernetes容器编排平台，应用的部署效率提升了十倍以上。同时，建立了完整的DevOps流水线，实现了从代码提交到生产部署的自动化。

第四阶段: 数据智能化(持续进行)

利用Db2 for i的数据仓库功能，构建了企业级数据湖。将历史交易数据、客户行为数据、风险数据等整合至统一平台，为数据分析和人工智能应用提供支撑。通过引入机器学习模型，实现了智能风控、精准营销等创新应用。

实施成效

经过三年的分阶段实施，该银行核心系统现代化改造取得了显著成效:

- **交易处理能力提升300%**: 核心交易响应时间从平均800毫秒降低至200毫秒以内
- **日终批处理时间缩短70%**: 从原来的6小时缩短至1.8小时
- **新功能上线周期缩短80%**: 从原来的3-6个月缩短至2-4周
- **系统可用性达到99.99%**: 全年计划外停机时间不超过1小时
- **运维人员减少40%**: 自动化程度大幅提升，人工干预显著减少

经验总结

该项目的成功实施为金融行业核心系统现代化提供了宝贵经验:

1. **循序渐进**: 避免"大爆炸"式重构，采用渐进式改造降低风险
2. **业务连续性优先**: 确保改造过程中业务不受影响，新旧系统平滑过渡
3. **人才培养**: 重视现有技术团队的技能提升，培养既懂传统技术又掌握新技术的复合型人才
4. **治理先行**: 建立完善的架构治理体系，确保改造过程中的技术一致性

案例二: 股份制银行支付系统灾备建设

项目背景

某全国性股份制银行支付系统承载着日均千万级交易量的处理任务，系统可用性要求达到99.999%。为应对日益严峻的网络安全形势和自然灾害风险，银行决定建设异地双活数据中心，实现支付系统的容灾备份。

技术架构

同城双中心架构

在相同城市内建设两个数据中心，相距约30公里，通过专用光纤网络互联。两个数据中心采用Active-Active模式同时对外提供服务，通过全局负载均衡器实现流量的智能分发。单个数据中心故障时，另一个数据中心可在秒级接管全部业务。

异地灾备中心

在距离主中心500公里以上的异地建设灾备中心，采用Active-Standby模式。通过数据异步复制保持与主中心的数据一致性。当同城双中心同时故障时，异地灾备中心可在分钟级完成业务接管。

数据复制方案

采用IBM　PowerHA配合MIMIX数据复制软件实现核心数据的实时同步。关键业务数据采用同步复制模式，确保RPO(恢复点目标)为零。非关键数据采用异步复制，减少对生产系统性能的影响。通过压缩和加密技术，确保数据传输的安全性和效率。

关键技术实现

网络层双活

通过BGP　Anycast技术实现IP地址在两个数据中心的同时发布。当某个数据中心网络故障时，路由自动切换到可用中心，用户无感知。DNS智能解析根据用户地理位置和网络质量，自动选择最优数据中心。

数据库双活

Db2 for i采用逻辑复制技术，在两端数据库同时保持可写状态。通过分布式事务协调器确保跨数据中心事务的一致性。采用冲突检测和自动解决机制，处理极少数情况下的数据冲突。

应用层双活

应用系统设计为无状态架构，会话信息集中存储在Redis集群中。任意应用实例均可处理任意用户请求，实现真正的负载均衡。通过消息队列实现跨中心的异步解耦，确保业务连续性。

灾备演练与验证

定期演练机制

建立季度演练制度，每季度进行一次不同场景的灾备演练。演练场景包括:单点故障、单中心故障、双中心故障、网络分区、数据库故障等。通过真实业务流量的切换验证，确保灾备方案的有效性。

自动化演练工具

开发自动化演练平台，可一键触发灾备切换流程。平台自动执行故障注入、流量切换、健康检查、业务验证等步骤，全程无需人工干预。演练完成后自动生成报告，评估各项指标的达成情况。

实施成效

- **RTO(恢复时间目标)**: 同城切换小于30秒，异地切换小于5分钟
- **RPO(恢复点目标)**: 同城为零，异地小于5分钟
- **业务连续性**: 全年可用性达到99.999%，计划外停机时间为零
- **合规达标**: 满足银保监会关于银行业信息系统灾难恢复等级5级要求

案例三: 城商行信用卡系统风控升级

项目背景

某城市商业银行信用卡业务近年来快速发展，发卡量突破百万张。然而，随着业务规模扩大，欺诈交易和信用风险事件也有所增加。原有的风控系统基于规则引擎，难以应对日益复杂的欺诈手段，误报率和漏报率均居高不下。

问题分析

通过深入分析，发现原有风控系统存在以下问题:

1. **规则僵化**: 基于专家经验的规则难以覆盖所有风险场景，新类型欺诈无法及时识别

2. **响应滞后**: 事后分析模式导致风险发现滞后，资金损失难以挽回
3. **误报严重**: 过度保守的规则策略导致大量正常交易被拦截，影响客户体验
4. **数据孤岛**: 风控数据分散在不同系统，缺乏统一视图和关联分析能力

解决方案

实时风控引擎重构

采用复杂事件处理(CEP)技术重构风控引擎，实现毫秒级风险决策。引入行为生物特征分析，通过设备指纹、操作习惯、交易模式等多维度特征识别异常行为。建立实时图数据库，实现关联网络分析，快速识别团伙欺诈。

机器学习模型应用

与专业AI公司合作，基于历史数据训练反欺诈模型。采用XGBoost、深度学习等算法，构建交易欺诈预测模型、申请评分模型、行为异常检测模型。模型每日自动训练和更新，持续优化预测准确率。通过A/B测试验证，机器学习模型相较规则引擎，欺诈识别率提升40%，误报率降低60%。

智能决策平台

构建统一的智能风控决策平台，整合规则引擎、机器学习模型、知识图谱等多种决策能力。支持策略的灵活配置和灰度发布，风控人员可通过可视化界面快速调整策略参数。建立决策追踪机制，完整记录每笔交易的决策路径，支持事后审计和分析。

技术创新点

联邦学习应用

在保护数据隐私的前提下，与同业机构开展联邦学习合作。各方在不共享原始数据的情况下，共同训练反欺诈模型，显著提升了新类型欺诈的识别能力。该做法符合数据安全监管要求，为行业协作提供了新思路。

图神经网络(GNN)

引入图神经网络技术，挖掘客户关联网络中的隐藏风险。通过分析担保关系、交易关系、设备共用关系等，识别隐性关联风险和团伙欺诈模式。相比传统关联分析，GNN能够发现更深层次的关联特征。

实施成效

- **欺诈损失降低65%**: 年度欺诈损失金额从改造前的数千万降至千万以下
- **审批效率提升300%**: 自动化审批比例从40%提升至85%，人工审核工作量大幅减少
- **客户体验改善**: 误拦截率从15%降至3%，客户投诉量下降70%

- **模型准确率**: 欺诈识别准确率达到95%以上，处于行业领先水平

案例四: 农商行核心系统云化迁移

项目背景

某农村商业银行原核心系统运行在老旧的小型机上，设备老化、维护成本高、扩展性差。为降低IT投入、提升系统灵活性，银行决定将核心系统迁移至混合云架构，保留关键业务在本地IBM i平台，将渠道和分析类应用迁移至公有云。

迁移策略

业务分级分类

将全行应用系统按照业务关键性和数据敏感性进行分级:
- **核心层(保留本地)**: 账户系统、总账系统、支付清算等核心交易
- **服务层(混合部署)**: 客户信息、产品工厂、定价引擎等共享服务
- **渠道层(全面云化)**: 手机银行、网上银行、微信银行等客户触点
- **分析层(全面云化)**: 数据仓库、风险管理、监管报送等分析应用

技术架构设计

采用"云-边-端"三层架构:
- **云端**: 部署渠道应用、数据分析、开发测试环境，利用云的弹性伸缩能力
- **边端(本地)**: 保留IBM i核心系统，通过API网关与云端互联
- **终端**: 网点、ATM、POS等接入设备

云边之间通过专线+VPN双链路互联，确保通信可靠性。采用SD-WAN技术实现智能选路，优化网络质量。

数据同步架构

设计实时+批量相结合的数据同步方案:
- **实时同步**: 关键交易数据通过消息队列实时同步至云端，支持实时风控和客户服务
- **批量同步**: 全量数据每日批量同步，支持数据分析和报表生成
- **反向同步**: 云端配置数据、产品参数等反向同步至本地核心系统

实施过程

第一阶段: 基础设施准备(3个月)

完成公有云账号开通、网络架构设计、安全策略配置。建立云运维体系，培训技术团队掌握云原生技术。完成本地数据中心网络改造，部署云边互联设备。

第二阶段: 非核心系统迁移(6个月)

优先迁移开发测试环境和办公系统，积累云化经验。随后迁移数据分析和监管报送系统，验证大数据量在云端的处理能力。最后迁移渠道类应用，采用蓝绿部署方式实现平滑切换。

第三阶段: 核心系统API化(12个月)

对本地IBM i核心系统进行API化改造，确保云端应用能够安全、高效地访问核心服务。建立统一的API管理平台，实现接口的全生命周期管理。完善监控告警体系，实现云边一体化运维。

实施成效

- **IT成本降低40%**: 通过云资源的弹性使用，避免了传统IT的大量前期投入
- **系统扩展能力提升**: 渠道应用可根据业务量自动扩缩容，从容应对业务高峰
- **灾备能力增强**: 利用云的灾备服务，以较低成本实现异地灾备
- **创新速度加快**: 云原生技术支持快速迭代，新产品上线周期缩短60%

经验与启示

该案例表明，中小银行完全可以通过混合云架构实现核心系统的现代化升级。关键在于:

1. **明确边界**: 核心账务保留在安全的本地平台，渠道和分析充分利用云的灵活性
2. **渐进迁移**: 从外围系统开始，逐步向核心业务延伸
3. **安全优先**: 云边之间采用多重安全机制，确保数据传输和访问安全
4. **人才培养**: 重视云技术人才的培养和引进，建立云原生运维能力

补充章节: 金融科技前沿技术

分布式账本技术在金融领域的应用

技术概述

分布式账本技术(DLT)，特别是区块链技术，正在重塑金融基础设施。与传统中心化账本不同，DLT通过密码学和共识机制实现多方账本的同步维护，具有去中心化、不可篡改、透明可追溯等特点。

金融应用场景

跨境支付与清算

传统跨境支付依赖代理行网络，流程长、成本高、透明度低。基于DLT的跨境支付平台可实现点对点资金转移，将结算时间从天级缩短至秒级，成本降低70%以上。多家国际银行已参与或发起了此类项目，如JPM Coin、汇丰银行的FX Everywhere等。

贸易融资

贸易融资业务涉及多个参与方和大量纸质单据，效率低下且容易造假。DLT可将贸易单据数字化并上链，实现单据的可信流转和自动核验。智能合约可根据预设条件自动执行付款，大幅降低操作风险和欺诈风险。

供应链金融

核心企业的信用可以通过DLT传递给多级供应商，解决中小企业融资难问题。应收账款上链后，供应商可快速获得融资，且资金流向全程可追溯，降低金融机构的风险敞口。

央行数字货币(CBDC)

多国央行正在研究基于DLT的央行数字货币。中国数字人民币(e-CNY)采用"央行-商业银行"双层运营架构，既保持了货币主权的集中管理，又充分发挥了商业银行的渠道优势。

技术挑战与应对

性能瓶颈

公有链的TPS(每秒交易数)难以满足金融级要求。联盟链通过精简共识节点、采用高性能共识算法(如PBFT、Raft)，可将TPS提升至数千甚至上万。结合分片、侧链等技术，性能还可进一步提升。

隐私保护

金融交易数据高度敏感。零知识证明(ZKP)、同态加密、可信执行环境(TEE)等技术的应用，使得在保护隐私的前提下实现数据验证成为可能。如Zcash使用的zk-SNARKs技术，可隐藏交易金额和地址。

监管合规

DLT的匿名性与反洗钱监管要求存在冲突。解决方案包括:链上身份认证、交易监控节点、监管沙盒等。设计之初就考虑监管要求，是金融DLT项目成功的关键。

人工智能在金融领域的深度应用

智能客服

技术实现

基于大语言模型(LLM)的智能客服系统，能够理解自然语言、进行多轮对话、处理复杂咨询。通过RAG(检索增强生成)技术，将银行知识库与语言模型结合，确保回答的准确性和时效性。

应用效果

- 问题解决率从传统机器人的60%提升至90%
- 客户满意度显著提升
- 人工客服工作量减少50%
- 7×24小时服务能力，无等待时间

智能投研

技术架构

整合多源异构数据(财报、新闻、社交媒体、宏观数据)，利用NLP技术自动提取关键信息。结合知识图谱，构建企业关联关系网络。通过时间序列预测、情感分析等模型，生成投资建议。

应用场景

- **量化投资**: 基于AI模型的量化策略，实现高频交易
- **风险预警**: 通过舆情监控和关联分析，提前识别投资风险
- **ESG评估**: 自动收集和分析企业的ESG相关数据，辅助投资决策

智能风控

反欺诈

采用深度学习和图神经网络，识别复杂的欺诈模式。实时分析交易流数据，毫秒级响应。联邦学习技术支持跨机构协作，共同提升风控能力而不泄露敏感数据。

信用评估

传统信用评估主要依赖征信报告和财务数据。AI技术可以整合替代数据(如支付行为、社交网络、设备信息)，为缺乏征信记录的人群建立信用画像。机器学习模型可处理高维稀疏数据，发现人工难以察觉的风险因子。

市场风险管理

利用强化学习优化投资组合，在风险约束下追求收益最大化。通过蒙特卡洛模拟和压力测试，评估极端市场情况下的风险敞口。实时监测市场风险指标，触发预警时自动调整持仓。

隐私计算技术

技术类型

多方安全计算(MPC)

允许多个参与方在不泄露各自输入的情况下，共同计算某个函数。在联合风控场景中，多家金融机构可在不共享客户数据的前提下，共同评估客户的信用风险。

联邦学习

模型训练过程分布在各个参与方本地，仅交换模型参数或梯度信息，原始数据不出本地。适用于客户画像、精准营销等对数据隐私要求高的场景。

可信执行环境(TEE)

基于硬件的安全技术(如Intel SGX、ARM TrustZone)，在处理器内部创建安全飞地。敏感数据和计算逻辑在TEE内执行，外部无法窥探。适合需要高性能且安全性要求极高的场景。

金融应用

联合反洗钱

多家银行通过隐私计算技术共享可疑交易模式，共同提升洗钱识别能力。单家银行的可疑交易样本有限，联合建模可显著提升模型效果。

精准营销

银行与电商、运营商等跨界合作，在不泄露各自客户数据的前提下，实现跨行业的客户画像和精准推荐。

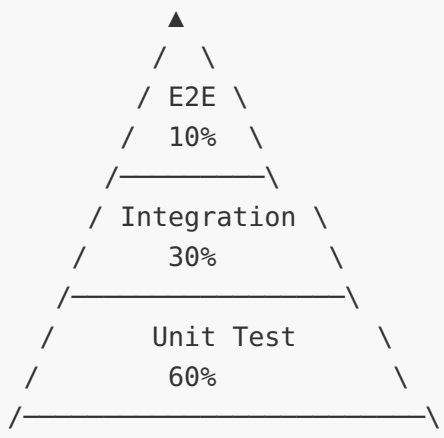
医疗金融

保险公司与医疗机构合作，基于加密的患者数据进行健康险定价和理赔审核，既保护了患者隐私，又提升了业务效率。

补充章节: 金融系统测试最佳实践

测试策略体系

测试金字塔



单元测试是测试金字塔的基石，应当覆盖核心业务逻辑。集成测试验证模块间的协作。端到端(E2E)测试模拟真实用户场景。

测试类型详解

单元测试

使用RPGUnit框架对RPG程序进行单元测试。测试用例应当覆盖正常路径、边界条件和异常场景。

```
**free
dcl-proc TestCalculateInterest export;
    dcl-pi *n extproc(*dclcase);
    end-pi;

    // 测试正常场景
    assert(CalculateInterest(10000 : 0.05 : 365) = 500 :
           '正常利息计算失败');

    // 测试边界条件: 本金为0
    assert(CalculateInterest(0 : 0.05 : 365) = 0 :
           '本金为0时计算失败');

    // 测试边界条件: 利率为0
```

```
    assert(CalculateInterest(10000 : 0 : 365) = 0 :
           '利率为0时计算失败');

    // 测试大数值
    assert(CalculateInterest(999999999 : 0.05 : 365) > 0 :
           '大数值计算失败');
end-proc;
```

集成测试

验证模块间的接口和数据流。使用测试数据库，测试完成后自动回滚，确保测试隔离性。

性能测试

使用Apache JMeter或LoadRunner模拟高并发场景。关键指标包括:
- 响应时间: 95%交易响应时间在可接受范围内
- 吞吐量: 系统能够支撑的业务量峰值
- 资源利用率: CPU、内存、磁盘、网络使用是否合理
- 并发用户数: 系统能够同时支持的用户数量

安全测试

- 渗透测试: 模拟黑客攻击，发现系统漏洞
- 代码审计: 检查代码中的安全缺陷
- 漏洞扫描: 使用自动化工具扫描已知漏洞
- 配置审查: 检查系统配置是否符合安全基线

混沌工程

主动在生产环境注入故障，验证系统的韧性。如随机杀死容器、模拟网络延迟、关闭数据库连接等。

测试数据管理

数据脱敏

生产数据用于测试前必须经过脱敏处理，确保敏感信息(姓名、身份证号、银行卡号、手机号等)被替换为模拟数据。脱敏应当保持数据的格式和业务规则，确保测试有效性。

数据子集

全量生产数据量过大，不适合测试环境。采用抽样技术提取具有代表性的数据子集，既能覆盖各种业务场景，又能控制测试数据规模。

数据生成

使用数据生成工具创建完全模拟的测试数据。优点是完全可控，不会涉及真实客户信息。现代数据生成工具可以学习生产数据的分布特征，生成统计学上相似的数据。

持续测试

将测试集成到CI/CD流水线，实现代码提交后的自动测试。测试失败的代码无法合并至主干，确保代码质量。

补充章节: 金融系统运维管理

运维体系架构

ITIL实践

事件管理

建立7×24小时运维值班制度，确保系统故障能够第一时间响应。事件分级标准:
- P1(紧急): 核心系统全面故障，影响全行业务
- P2(高): 重要功能故障，影响部分业务或大量客户
- P3(中): 一般功能故障，影响有限
- P4(低): 轻微问题，不影响业务

问题管理

对反复出现的事件进行根因分析，从根本上解决问题。建立知识库，积累解决方案，提高问题处理效率。

变更管理

所有生产变更必须经过评审、测试、审批流程。变更窗口安排在低峰时段，变更前做好回滚准备。变更后验证业务功能，发现问题及时回滚。

配置管理

建立配置管理数据库(CMDB)，记录所有IT资产及其关系。配置项包括:服务器、网络设备、操作系统、中间件、应用程序、配置文件等。

监控体系

基础设施监控

- CPU、内存、磁盘、网络使用率
- 磁盘空间、IO性能
- 进程状态、服务可用性

应用监控

- 交易响应时间
- 交易成功率
- 并发用户数
- 异常错误率

业务监控

- 业务量趋势
- 渠道访问分布
- 客户行为分析
- 风险指标监测

智能告警

基于机器学习建立动态基线，识别异常模式。关联分析多个指标，减少误报。告警分级推送，确保关键信息及时送达。

应急响应

应急预案

针对各类可能的故障场景(数据库故障、网络中断、应用崩溃、自然灾害等)，制定详细的应急预案。预案包括:
- 故障识别和分级标准
- 应急处理流程
- 责任人联系方式
- 技术处理步骤
- 业务连续性措施

灾备切换

定期演练灾备切换流程，确保在真实灾难发生时能够迅速恢复业务。切换决策应当由业务、技术、风险等多方共同决定。

文档结束