

目录

1	系统概述与设计目标	2
2	系统架构与项目目录结构	2
3	通用数据约定	5
4	核心数据结构: LinkedList	5
4.1	结构特点	5
4.2	模板定义	5
5	核心数据实体	7
5.1	患者实体: Patient	7
5.1.1	属性定义	7
5.1.2	就诊状态枚举	7
5.1.3	关键方法	7
5.2	医生实体: Doctor	7
5.2.1	属性定义	7
5.2.2	医学职称枚举	8
5.3	科室实体: Department	8
5.3.1	属性定义	8
5.3.2	关键方法	8
5.4	药品实体: Medicine	8
5.4.1	属性定义	8
5.4.2	关键方法	9
5.5	检查项目实体: Check	9
5.5.1	属性定义	9
5.6	病房与床位实体: Ward & Bed	9
5.6.1	Bed 结构	9
5.6.2	Ward 结构	9
5.6.3	枚举定义	9
5.6.4	Ward 关键方法	10
6	病例与记录模型	11
6.1	患者病例: PatientCase	11
6.1.1	顶层属性	11
6.2	诊断记录: DiagnosisRecord	11
6.3	用药记录: MedicineRecord	11
6.4	检查记录: CheckRecord	12
6.5	住院记录: AdmissionRecord	12
6.6	预约记录: AppointmentRecord	12
7	支付与结算模型	13
7.1	支付记录: Payment	13
7.1.1	属性定义	13

7.1.2 支付方式枚举	13
7.2 结算单: Settlement	13
7.2.1 结算明细项: SettlementItem	13
7.2.2 Settlement 属性	13
8 全局上下文与组合根	14
8.1 HisContext (全局数据容器)	14
8.2 HisCore (组合根)	14
9 业务逻辑层: 服务类	16
9.1 患者服务: PatientService	16
9.2 医生服务: DoctorService	16
9.3 药品服务: MedicineService	16
9.4 病房服务: WardService	16
9.5 检查项目服务: CheckService	16
9.6 科室服务: DepartmentService	16
9.7 患者病例服务: PatientCaseService	16
9.8 支付服务: PaymentService	17
9.9 结算服务: SettlementService	17
9.10 支付管理服务: PaymentManagementService	17
9.11 报表服务: ReportService	17
10 工具层	18
10.1 Logger (日志系统)	18
10.2 FileManager (文件管理)	18
10.3 UUID (唯一标识符生成器)	18
10.4 E2hangJson (自定义 JSON 库)	18
11 用户界面层	19
11.1 CLI 命令行界面 (ReplShell)	19
11.1.1 患者管理	19
11.1.2 医生管理	19
11.1.3 药品管理	19
11.1.4 病房管理	19
11.1.5 检查项目管理	19
11.1.6 科室管理	19
11.1.7 病例管理	19
11.1.8 支付与结算	19
11.1.9 日志与系统	20
11.2 GUI 图形界面 (MainWindow)	20
11.2.1 视角枚举 (ViewRole)	20
11.2.2 主要标签页	20
11.2.3 对话框组件	20
12 系统核心交互流程图	21

13 关键业务流程	22
13.1 患者完整就诊流程	22
13.2 住院管理流程	22
13.3 处方与药物管理流程	22
13.4 支付与结算流程	22
14 数据持久化	23
14.1 JSON 序列化策略	23
14.2 数据文件清单	23
14.3 加载与保存策略	23
15 编译与运行	23
15.1 环境要求	23
15.2 编译步骤	23
15.3 运行方式	24
16 实体关系图	25
17 设计模式与最佳实践	25
17.1 采用的设计模式	25
17.2 业务规则	25
18 总结	26
A 数据结构时间复杂度对比	26

《HIS 医院信息管理系统》

(数据结构、功能架构与使用说明)

1 系统概述与设计目标

本系统为一个面向中小型医院的完整医疗信息管理系统 (Hospital Information System, HIS), 采用 C++20 开发, 结合 Qt6 框架构建图形界面。系统通过模块化分层设计, 实现业务逻辑与交互界面的解耦, 保障高扩展性与可维护性。

- **业务闭环**: 实现完整的挂号、诊疗、检查、处方、住院、出院、支付、结算等医疗生命周期管理。
- **双界面支持**: 同时提供 CLI 命令行交互界面和 Qt6 GUI 图形界面, 适配不同使用场景。
- **数据规模**: 支持大量患者、医生、药品、病房、检查项目、科室及支付结算记录的管理。
- **底层核心**: 全程基于手写泛型双向链表 + 哈希索引 (LinkedList) 实现内存数据流转, 采用 JSON 文本文件持久化。
- **多角色视角**: GUI 支持管理视角、病人视角和医护视角三种权限视图。
- **支付结算**: 支持现金、信用卡、移动支付、医保等多种支付方式, 自动生成结算单与财务报表。
- **鲁棒性**: 完善的错误处理、输入校验和业务规则拦截 (如住院中禁止删除患者、库存不足禁止发药等)。

2 系统架构与项目目录结构

系统采用四层分层架构设计 (UI 表现层 → Core 业务层 → Model 实体层 → Utils 工具层 → 持久化层)。

- **UI 表现层**: CLI (ReplShell) 命令行交互 / GUI (Qt6 MainWindow) 图形界面
- **Core 业务层**: HisCore 组合根 + 11 个 Service 服务类
- **Model 实体层**: 9 个核心数据模型 + 5 种记录结构体
- **Utils 工具层**: LinkedList、Logger、FileManager、UUID、E2hangJson
- **持久化层**: JSON 格式文本文件 (data/ 目录)

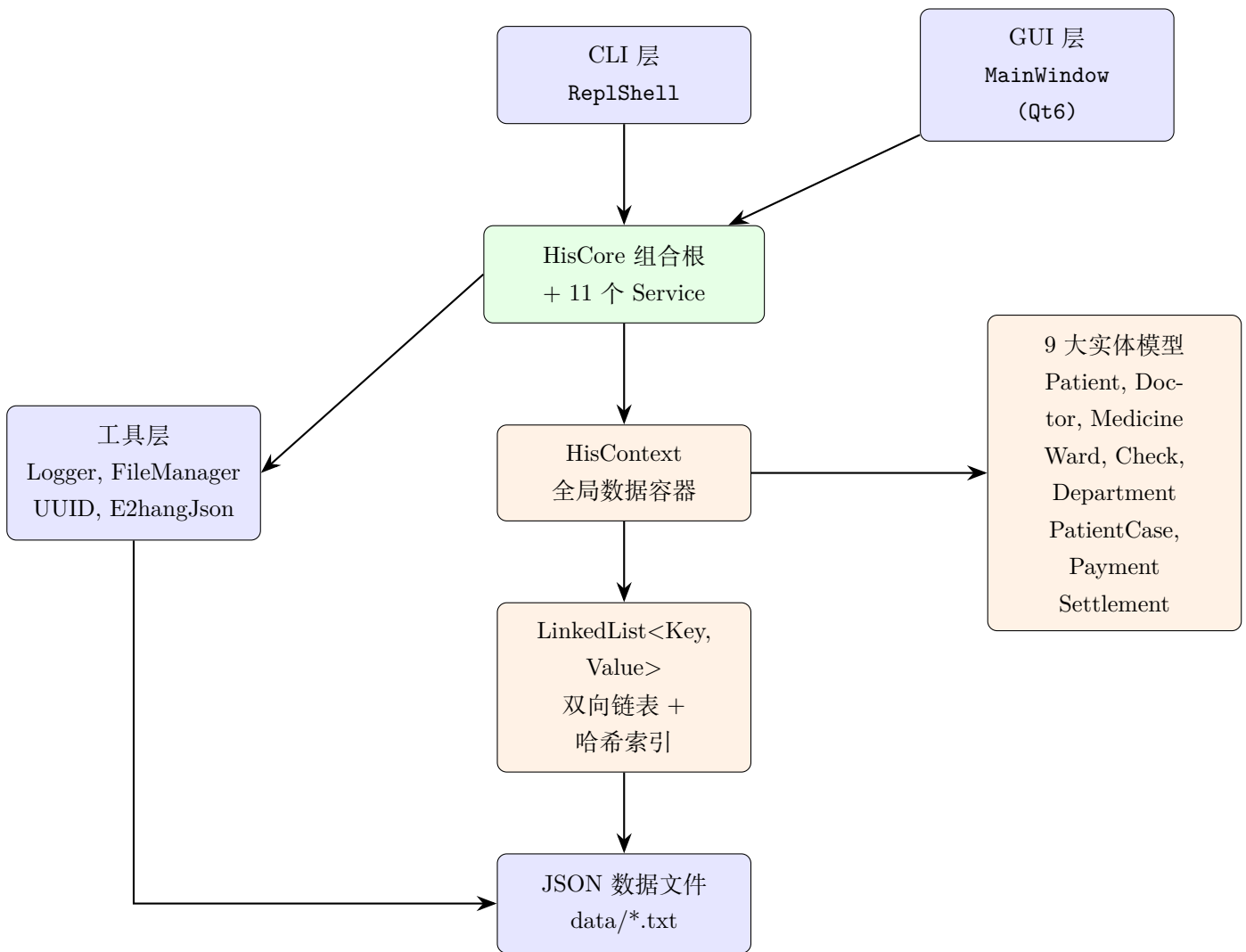


图 1: HIS 系统四层架构图

项目工程文件结构如下：

```
HIS-GUI/
CMakeLists.txt          # CMake 构建配置 (Qt6 + C++20)
README.md               # 项目简要说明
include/                # 头文件目录
  cli/                  # CLI 交互层
    repl_shell.h        # 交互式命令行主循环
    table_printer.h     # ASCII 表格打印工具
  core/                 # 核心业务服务层 (11个Service)
    his_core.h          # 组合根 (Composition Root)
    his_context.h       # 全局数据上下文 (HisContext)
    patient_service.h   # 患者服务
    doctor_service.h    # 医生服务
    medicine_service.h  # 药品服务
    ward_service.h      # 病房服务
    check_service.h     # 检查项目服务
    department_service.h # 科室服务
    patient_case_service.h # 患者病例服务
    payment_service.h   # 支付服务
    settlement_service.h # 结算服务
    payment_management_service.h # 支付管理服务
    report_service.h    # 报表服务
  models/               # 数据模型层 (9个实体 + 5种记录)
    patient.h           # 患者模型
    doctor.h            # 医生模型
    medicine.h          # 药品模型
    ward.h              # 病房与床位模型
    check.h             # 检查项目模型
    department.h        # 科室模型
    patient_case.h      # 患者病例模型
    payment.h           # 支付记录模型
    settlement.h        # 结算单模型
  utils/                # 工具层
    linkedlist.hpp      # 泛型双向链表 + 哈希索引
    logger.h            # 日志系统
    file_manager.h      # 文件读写与持久化
    uuid.h              # UUID v4 生成器
    json/               # 自定义 JSON 库
      JsonValue.h
      JsonParse.h
      JsonSerializer.h
      JsonError.h
  src/                  # 源文件实现
    main_shell.cpp      # CLI 入口
    main_noshell.cpp    # 无 Shell 模式入口
    cli/                # CLI 实现
    core/               # 11 个 Service 实现
    models/             # 9 个 Model 实现
    utils/              # 工具类实现
```

```

gui/                                # Qt6 GUI 相关
main_gui.cpp                        # GUI 入口
mainwindow.h/cpp                    # 主窗口（多标签页 + 角色切换）
payment_dialog.h/cpp                # 支付对话框
payment_management_dialog.h/cpp     # 支付管理对话框
settlement_dialog.h/cpp             # 结算对话框
dialogs/                            # 子对话框
    patient_dialog.h/cpp
    department_detail_dialog.h/cpp
data/                                # 数据文件目录（JSON 格式）
    patients.txt                    # 患者数据
    doctors.txt                     # 医生数据
    medicines.txt                   # 药品数据
    wards.txt                       # 病房数据
    checks.txt                      # 检查项目数据
    departments.txt                 # 科室数据
    patient_cases.txt               # 患者病例数据
    payments.txt                    # 支付记录数据
    settlements.txt                 # 结算单数据
tests/                              # 测试用例
docs/                                # 文档
build/                              # 构建输出

```

3 通用数据约定

- **唯一性**：所有系统实体均采用 **UUID** 作为唯一主键标识，通过 `UUID::generate()` 生成符合 RFC 4122 标准的 v4 UUID。
- **宽容性**：允许患者姓名重复，系统通过唯一 ID 精准区分同名患者。
- **存储规约**：内存态数据全程挂载于自定义泛型链表 **LinkedList** 进行流转；磁盘持久化统一采用 **JSON 数组** 格式存储于 `data/` 目录下的 TXT 文件中。
- **序列化**：所有模型均实现 `toJson()` 和 `fromJson()` 方法，与自定义 `E2hangJson` 库配合完成数据序列化/反序列化。

4 核心数据结构：LinkedList

4.1 结构特点

`LinkedList` 是系统的基础泛型数据结构，采用**双向链表** + **哈希表**的混合设计：

4.2 模板定义

```

template<class Key, class Value>
class LinkedList {
private:
    ListNode<Key, Value>* head; // 哨兵头节点
    ListNode<Key, Value>* tail; // 哨兵尾节点

```

特性	说明	复杂度
插入操作	在链表头部插入新元素	$O(1)$
查找操作	通过 <code>unordered_map</code> 哈希索引定位	$O(1)$
删除操作	从链表中移除并更新哈希表	$O(1)$
遍历操作	按链表顺序从头到尾遍历	$O(n)$
移到头部	LRU 风格，将节点移到链表头部	$O(1)$

表 1: LinkedList 操作复杂度

```
std::unordered_map<Key, ListNode<Key, Value>>*> mp; // 哈希索引
size_t sz; // 元素数量
public:
    void insert_front(Key, Value);
    void remove(Key);
    ListNode* find(Key);
    void move_to_front(Key);
    void remove_tail();
    size_t size();
    void for_each(visitor);
};
```


5 核心数据实体

5.1 患者实体：Patient

5.1.1 属性定义

属性名	类型	含义	是否主键
PatientID	string	患者唯一标识 (UUID)	是
Name	string	患者姓名	否
Age	int	患者年龄	否
Gender	string	性别	否
Contact	string	联系电话	否
Status	enum	就诊状态	否

表 2: Patient 属性表

5.1.2 就诊状态枚举

```
enum class PatientStatus {
    Outpatient,    // 门诊患者
    Inpatient,     // 住院患者
    Discharged,    // 已出院
    Visited        // 已就诊（门诊已完成诊断和用药）
};
```

5.1.3 关键方法

- updateBasicInfo() - 更新患者基本信息
- canBeRemoved() - 检查是否可删除（住院中禁止删除）
- nameMatches() - 姓名模糊匹配查询
- generateUniqueId() - 生成 UUID
- toJson()/fromJson() - JSON 序列化接口

5.2 医生实体：Doctor

5.2.1 属性定义

属性名	类型	含义
DoctorID	string	医生唯一标识 (UUID)
Name	string	医生姓名
DepartmentID	string	所属科室 ID
Title	enum	医学职称
Schedule	string	出诊时间安排

表 3: Doctor 属性表

5.2.2 医学职称枚举

```
enum class DoctorTitle {
    Chief,           // 主任医师
    AssociateChief, // 副主任医师
    Attending,       // 主治医师
    Resident         // 住院医师
};
```

5.3 科室实体: Department

5.3.1 属性定义

属性名	类型	含义
DepartmentID	string	科室唯一标识 (UUID)
Name	string	科室名称
Description	string	科室描述

表 4: Department 属性表

5.3.2 关键方法

- `getDoctorsByDepartment()` - 获取科室下所有医生
- `getMedicinesByDepartment()` - 获取科室下所有药品
- `canDeleteDepartment()` - 检查是否可删除 (有医生或药品时不能删)
- `getDoctorCountInDepartment()` - 获取科室下医生数量
- `getMedicineCountInDepartment()` - 获取科室下药品数量
- `getCheckCountInDepartment()` - 获取科室下检查项目数量

5.4 药品实体: Medicine

5.4.1 属性定义

属性名	类型	含义
MedicineID	string	药品唯一标识 (UUID)
GenericName	string	通用名
BrandName	string	商品名
Aliases	vector<string>	别名列表
StockQuantity	int	当前库存数量
DepartmentID	string	所属科室 ID
UnitPrice	double	单价

表 5: Medicine 属性表

5.4.2 关键方法

- `updateBasicInfo()` - 更新药品信息
- `nameMatches()` - 支持通用名、商品名及别名模糊匹配
- `decreaseStock()` - 减少库存 (库存不足时失败)
- `increaseStock()` - 增加库存
- `addAlias()` - 添加别名

5.5 检查项目实体: *Check*

5.5.1 属性定义

属性名	类型	含义
CheckID	string	检查项目唯一标识 (UUID)
Name	string	检查名称
DepartmentID	string	所属科室 ID
Price	double	检查价格

表 6: *Check* 属性表

5.6 病房与床位实体: *Ward & Bed*

5.6.1 *Bed* 结构

属性名	类型	含义
BedID	string	床位唯一标识 (UUID)
WardID	string	所属病房 ID
Status	enum	床位状态
PatientID	string	当前患者 ID (空表示空闲)

表 7: *Bed* 属性表

5.6.2 *Ward* 结构

属性名	类型	含义
WardID	string	病房唯一标识 (UUID)
DepartmentID	string	所属科室 ID
Type	enum	病房类型
MaxBeds	int	最大床位数
Beds	vector<Bed>	床位列表

表 8: *Ward* 属性表

5.6.3 枚举定义

```
enum class WardType {
    Normal,    // 普通病房
    Special,   // 特殊病房
    ICU        // 重症监护室
};

enum class BedStatus {
    Free,      // 空闲
    Occupied   // 被占用
};
```

5.6.4 Ward 关键方法

- `getFreeBedID()` - 获取第一个空闲床位
- `admitPatient()` - 为患者分配床位
- `releaseBed()` / `releasePatient()` - 释放床位
- `addBed()` / `removeBed()` - 增减床位
- `freeBedCount()` / `occupiedBedCount()` - 床位统计
- `occupancyRate()` - 计算床位使用率

占用率计算公式:

$$\text{OccupancyRate} = \frac{\text{OccupiedBeds}}{\text{MaxBeds}}$$

6 病例与记录模型

6.1 患者病例：PatientCase

PatientCase 是聚合实体，记录患者的完整医疗历史。

6.1.1 顶层属性

属性名	类型	含义
PatientID	string	患者 ID
DiagnosisRecords	vector	诊断记录列表
MedicineRecords	vector	用药记录列表
CheckRecords	vector	检查记录列表
AdmissionRecords	vector	住院记录列表
AppointmentRecords	vector	预约记录列表
CreatedTime	time_t	创建时间
LastModifiedTime	time_t	最后修改时间

表 9: PatientCase 属性表

6.2 诊断记录：DiagnosisRecord

属性	类型	含义
DoctorID	string	诊断医生 ID
Diagnosis	string	诊断内容
Prescription	string	处方（可选）
Remarks	string	备注
Timestamp	time_t	诊断时间戳

表 10: DiagnosisRecord 属性

6.3 用药记录：MedicineRecord

属性	类型	含义
MedicineID	string	药品 ID
MedicineName	string	药品名称
Quantity	int	用药数量
Usage	string	用法说明
UnitPrice	double	单价
DoctorID	string	开药医生 ID
Timestamp	time_t	开药时间戳

表 11: MedicineRecord 属性

费用计算：TotalPrice = Quantity × UnitPrice

6.4 检查记录: *CheckRecord*

属性	类型	含义
CheckID	string	检查 ID
CheckName	string	检查名称
DepartmentID	string	归属科室 ID
Price	double	检查价格
DoctorID	string	开单医生 ID
Timestamp	time_t	开单时间戳

表 12: *CheckRecord* 属性

6.5 住院记录: *AdmissionRecord*

属性	类型	含义
WardID	string	病房 ID
BedID	string	床位 ID
AdmissionTime	time_t	入院时间
DischargeTime	time_t	出院时间 (0 = 未出院)
Reason	string	住院原因
DischargeSummary	string	出院小结

表 13: *AdmissionRecord* 属性

判断当前是否住院: `isCurrentlyAdmitted()` 返回 `DischargeTime == 0`。

6.6 预约记录: *AppointmentRecord*

属性	类型	含义
DoctorID	string	医生 ID
PatientID	string	患者 ID
AppointmentDate	string	预约日期 (YYYY-MM-DD)
Notes	string	备注
Timestamp	time_t	创建时间

表 14: *AppointmentRecord* 属性

7 支付与结算模型

7.1 支付记录：Payment

7.1.1 属性定义

属性名	类型	含义
PaymentID	string	支付唯一标识 (UUID)
PatientID	string	患者 ID
Amount	double	支付金额
Method	enum	支付方式
PaymentType	string	支付类型 (挂号/检查/用药等)
OperationID	string	关联的操作 ID
PaymentTime	time_t	支付时间
Status	string	状态 (Pending/Completed/Failed/Refunded)
Description	string	支付描述

表 15: Payment 属性表

7.1.2 支付方式枚举

```
enum class PaymentMethod {
    Cash,           // 现金
    CreditCard,     // 信用卡
    MobilePayment,  // 移动支付 (微信/支付宝)
    HealthInsurance // 医保
};
```

7.2 结算单：Settlement

7.2.1 结算明细项：SettlementItem

属性名	类型	含义
ItemName	string	项目名称
ItemType	string	项目类型
OperationID	string	关联操作 ID
Quantity	double	数量
UnitPrice	double	单价
Amount	double	金额
PaymentMethod	string	支付方式

表 16: SettlementItem 属性表

7.2.2 Settlement 属性

属性名	类型	含义
SettlementID	string	结算单唯一标识 (UUID)
PatientID	string	患者 ID
PatientName	string	患者姓名
DischargeDate	string	出院日期
SettlementTime	time_t	结算时间
Items	vector<SettlementItem>	费用明细列表
TotalAmount	double	医疗总费用
InsurancePaid	double	医保支付金额
PatientPaid	double	患者自付金额
Status	string	状态 (Pending/Completed/Settled)
Notes	string	备注

表 17: Settlement 属性表

8 全局上下文与组合根

8.1 HisContext (全局数据容器)

HisContext 是系统的数据容器，持有所有实体的 LinkedList，作为各服务类的共享数据源。

```
namespace core {
class HisContext {
public:
    LinkedList<std::string, Ward> wards;
    LinkedList<std::string, Patient> patients;
    LinkedList<std::string, Doctor> doctors;
    LinkedList<std::string, Medicine> medicines;
    LinkedList<std::string, Check> checks;
    LinkedList<std::string, PatientCase> patientCases;
    LinkedList<std::string, Department> departments;
    LinkedList<std::string, Payment> payments;
    LinkedList<std::string, Settlement> settlements;
};
}
```

8.2 HisCore (组合根)

HisCore 集中组装所有服务和上下文，是系统的入口点。

```
namespace core {
class HisCore {
public:
    HisCore();
    bool loadDataFromFolder(const std::string& dataFolder, std::string& outError);
    void fixDepartmentReferences();

    HisContext ctx_;
};
}
```



```
WardService wardService;  
PatientService patientService;  
PatientCaseService patientCaseService;  
ReportService reportService;  
DoctorService doctorService;  
MedicineService medicineService;  
CheckService checkService;  
DepartmentService departmentService;  
PaymentService paymentService;  
SettlementService settlementService;  
PaymentManagementService paymentManagementService;  
};  
}
```

9 业务逻辑层：服务类

9.1 患者服务：PatientService

- 患者的 CRUD 操作（增删改查）
- 患者状态管理（门诊/住院/已出院/已就诊）
- 住院分配与出院释放床位流程
- 多维度查询：按 ID、姓名、联系方式模糊搜索

9.2 医生服务：DoctorService

- 医生信息的 CRUD 操作
- 按科室遍历医生
- 模糊搜索（姓名/科室/职称/排班）

9.3 药品服务：MedicineService

- 药品库存管理（CRUD）
- 名称模糊匹配（通用名/商品名/别名）
- 入库增加库存 / 出库减少库存（库存不足时失败）

9.4 病房服务：WardService

- 病房和床位的 CRUD 操作
- 病房数据的持久化（JSON 文件 load/save）
- 床位分配和回收管理

9.5 检查项目服务：CheckService

- 检查项目的 CRUD 操作
- 按名称模糊搜索
- 创建时自动生成 UUID

9.6 科室服务：DepartmentService

- 科室的 CRUD 操作
- 获取科室下的医生、药品、检查项目列表
- 删除前检查依赖（有医生或药品时不能删除）
- 统计科室下各类资源数量

9.7 患者病例服务：PatientCaseService

- 管理患者的完整医疗记录（诊断/用药/检查/住院/预约）
- 患者出院流程处理
- 医疗统计（费用计算、记录数统计）

9.8 支付服务: *PaymentService*

- 创建支付记录 (支持多种支付方式)
- 查询支付记录 (按患者、按类型)
- 更新支付状态 (完成/退款)
- 获取患者总支付金额

9.9 结算服务: *SettlementService*

- 为患者生成结算单 (出院时自动生成)
- 添加费用项到结算单
- 计算结算总额 (总费用/医保/自付)
- 完成结算、生成结算单报告

9.10 支付管理服务: *PaymentManagementService*

- 支付记录管理 (按状态/日期范围过滤)
- 支付统计 (总收入/已完成/待支付/已退款)
- 支付方式统计 (按 Cash/CreditCard/MobilePayment/HealthInsurance 分类)
- 今日/本月支付统计
- 结算单管理 (按状态/患者过滤、搜索)
- 报表生成 (日报/月报/收入报表/患者账单)
- 收入趋势数据 (每日/每月/按类型)
- 异常支付检测与退款处理

9.11 报表服务: *ReportService*

- 病房床位使用率计算

10 工具层

10.1 Logger (日志系统)

- 支持 12 种日志类型 (Shell 命令/患者操作/医生操作/药品操作/病房操作/诊断/用药/住院/出院/检查/系统事件等)
- 可配置日志格式 (占位符: {time}, {type}, {command}, {details}, {objectId})
- 支持控制台输出和文件输出
- 提供专用日志方法: logShellCommand(), logPatientOperation(), logDiagnosisRecord() 等
- 支持导出日志到文件、清空日志缓冲区

10.2 FileManager (文件管理)

为所有实体类型提供统一的文件读写操作:

- 基础文件操作: readTextFile(), writeTextFile(), createFile(), deleteFile()
- JSON 字段删除: deleteJsonField()
- 所有 9 种实体的 loadXxxListFromFile() 和 saveXxxListToFile() 方法

10.3 UUID (唯一标识符生成器)

生成符合 RFC 4122 标准的 UUID v4:

- generate() - 标准格式: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
- generateShort() - 无连字符格式
- isValid() - 格式校验
- toShort() / fromShort() - 格式转换

10.4 E2hangJson (自定义 JSON 库)

- JsonValue: JSON 值表示 (对象、数组、字符串、数字、布尔值)
- JsonParse: JSON 解析器
- JsonSerializer: JSON 序列化器
- JsonError: JSON 错误处理

11 用户界面层

11.1 CLI 命令行界面 (ReplShell)

基于命令行的交互式 Shell，支持所有核心操作。主要命令分类如下：

11.1.1 患者管理

- patient add / update / rm / list / search
- patient load / save
- patient admit / release bed / release patient

11.1.2 医生管理

- doctor add / rm / list / search
- doctor list dept / search name / search dept / search title
- doctor load / save

11.1.3 药品管理

- medicine add / list / find
- medicine stock inc / dec
- medicine load / save

11.1.4 病房管理

- ward add / rm / list / show
- ward bed add / rm

11.1.5 检查项目管理

- check add / list / find
- check load / save

11.1.6 科室管理

- department add / rm / list
- department load / save

11.1.7 病例管理

- case diagnosis add / case medicine add / case check add
- case admission add / case discharge / case appointment add
- case view / case stats

11.1.8 支付与结算

- payment create / list / complete / refund / search
- settlement generate / list / view / complete

11.1.9 日志与系统

- log view / export / clear
- help / exit / save / load

11.2 GUI 图形界面 (MainWindow)

基于 Qt6 的图形界面，提供多标签页和角色视角切换。

11.2.1 视角枚举 (ViewRole)

视角	说明	可见功能
Admin	管理视角	所有功能
Patient	病人视角	只能查看医生信息
Medical	医护视角	患者、医生、药品、检查、病例管理

表 18: GUI 视角枚举

11.2.2 主要标签页

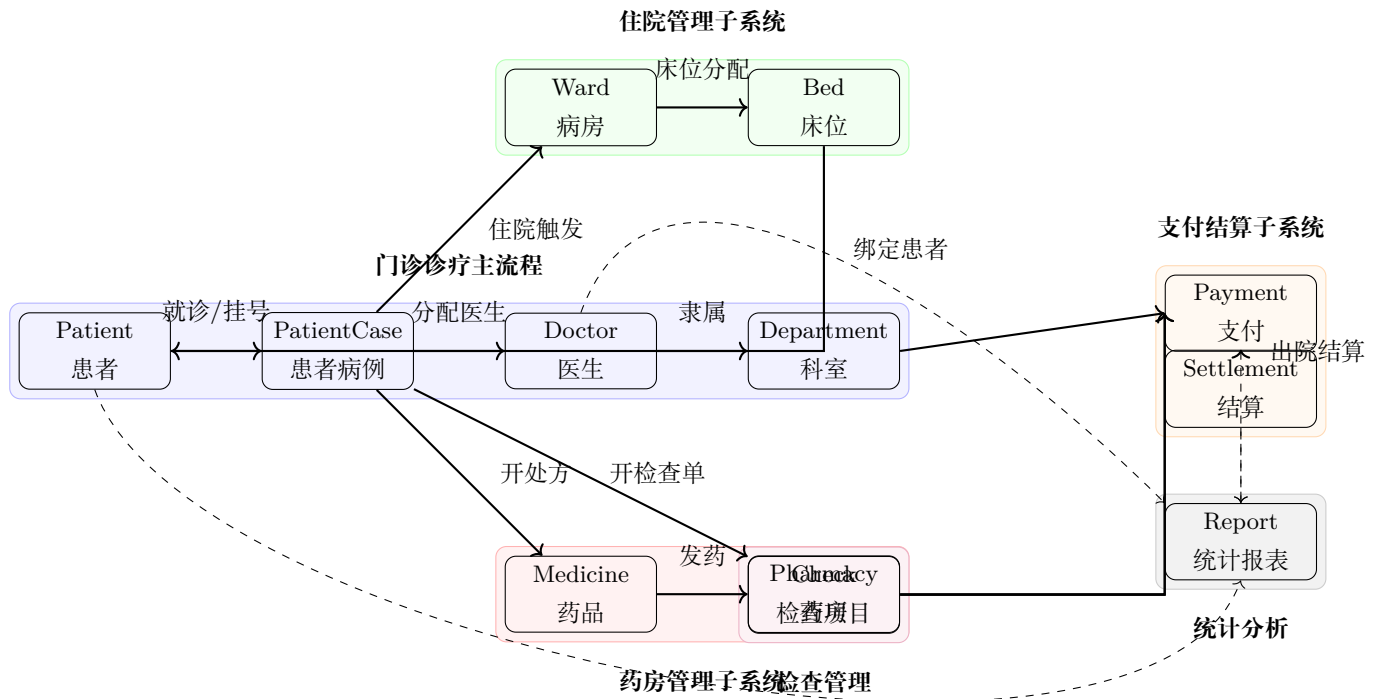
标签页	主要组件	功能
仪表盘	summaryLabel	系统统计信息
病房管理	wardTreeView, wardTable	树形浏览、床位管理
患者管理	patientTable	列表、搜索、增删改查
医生管理	doctorTable	列表、搜索、增删
药品管理	medicineTable	列表、搜索、库存管理
检查管理	checkTable	列表、搜索、增删改
科室管理	departmentTable	列表、搜索、详情查看
日志管理	logDisplay	日志查看、导出、清除

表 19: GUI 标签页组件

11.2.3 对话框组件

- PatientDialog - 患者信息编辑对话框
- DepartmentDetailDialog - 科室详情（显示科室下医生、药品、检查项目）
- PaymentDialog - 支付对话框
- PaymentManagementDialog - 支付管理（统计、报表、图表）
- SettlementDialog - 结算对话框

12 系统核心交互流程图



13 关键业务流程

13.1 患者完整就诊流程

1. 患者注册: `PatientService.addPatient()` 或 GUI 添加, 状态为 `Outpatient`
2. 挂号预约: `PatientCaseService.addAppointmentRecord()` 创建预约记录
3. 医生诊断: `PatientCaseService.addDiagnosisRecord()` 添加诊断记录
4. 开药: `PatientCaseService.addMedicineRecord()` 添加用药记录, 同时 `MedicineService.decreaseStock()` 扣减库存
5. 开检查: `PatientCaseService.addCheckRecord()` 添加检查记录
6. 支付: `PaymentService.createPayment()` 创建支付记录
7. 如需住院: `PatientService.admitPatient()` 分配床位, 状态改为 `Inpatient`
8. 出院: `PatientCaseService.dischargePatient()` 填写出院小结, `SettlementService.generateSettlement()` 生成结算单
9. 结算完成: `SettlementService.completeSettlement()` 完成结算, 患者状态改为 `Discharged`

13.2 住院管理流程

1. 查找可用病房和空闲床位: `Ward.getFreeBedID()`
2. 分配床位: `Ward.admitPatient()` 绑定患者到床位
3. 更新患者状态为 `Inpatient`
4. 住院期间可继续添加诊断、用药、检查记录
5. 出院时: 释放床位 `Ward.releasePatient()`, 更新患者状态为 `Discharged`

13.3 处方与药物管理流程

1. 医生开具处方: `PatientCaseService.addMedicineRecord()`
2. 系统记录药品 ID、数量、用法、单价
3. 自动扣减库存: `MedicineService.decreaseStock()` (库存不足时拒绝)
4. 计算费用: `Quantity * UnitPrice`
5. 查询患者总药费: `PatientCase.getTotalMedicineCost()`

13.4 支付与结算流程

1. 创建支付: `PaymentService.createPayment()` 记录支付方式、金额、类型
2. 完成支付: `PaymentService.completePayment()` 更新状态为 `Completed`
3. 出院时自动生成结算单: `SettlementService.generateSettlement()`
4. 结算单汇总所有费用 (用药/检查/住院等), 区分医保支付和患者自付
5. 完成结算: `SettlementService.completeSettlement()`
6. 管理端可查看统计报表: `PaymentManagementService.getPaymentStatistics()`

14 数据持久化

14.1 JSON 序列化策略

所有数据模型均支持 JSON 序列化，通过 `toJson()` 和 `fromJson()` 接口实现。数据以 JSON 数组格式存储于 TXT 文件中。

14.2 数据文件清单

文件路径	存储内容
data/patients.txt	患者数据
data/doctors.txt	医生数据
data/medicines.txt	药品数据
data/wards.txt	病房数据
data/checks.txt	检查项目数据
data/departments.txt	科室数据
data/patient_cases.txt	患者病例数据
data/payments.txt	支付记录数据
data/settlements.txt	结算单数据

表 20: 数据文件清单

14.3 加载与保存策略

遵循”启动时全量挂载，执行关键操作或退出时序列化落盘”的策略。

- CLI 模式：load / save 命令加载/保存所有数据；也可单独加载/保存某类数据
- GUI 模式：启动时自动从 data/ 文件夹加载；点击保存按钮保存所有数据

15 编译与运行

15.1 环境要求

- C++20 兼容编译器（GCC 10+ / Clang 12+ / MSVC 2019+）
- CMake 3.16+
- Qt6（Widgets, Charts 组件）

15.2 编译步骤

```
mkdir -p build
cd build
cmake ..
make -j4
```

15.3 运行方式

```
# 交互式 CLI Shell
./his

# 无 Shell 模式 (仅加载数据)
./his --noshell

# GUI 图形界面
./his_gui

# 批处理模式 (管道输入)
printf "patient_load_./data/patients.txt\ncase_view_p1001\nexit\n" | ./his
```

16 实体关系图

Patient (患者)

```

1:1 > PatientCase (患者病例)
    1:n > DiagnosisRecord (诊断记录)
    1:n > MedicineRecord (用药记录)
    1:n > CheckRecord (检查记录)
    1:n > AdmissionRecord (住院记录)
    1:n > AppointmentRecord (预约记录)

M:1 > Ward (病房) > 1:n > Bed (床位)
1:n > Payment (支付记录)
1:n > Settlement (结算单) > 1:n > SettlementItem (结算明细)

```

Doctor (医生)

```

M:1 > Department (科室)

```

Medicine (药品)

```

M:1 > Department (科室)

```

Check (检查项目)

```

M:1 > Department (科室)

```

Ward (病房)

```

M:1 > Department (科室)

```

17 设计模式与最佳实践

17.1 采用的设计模式

- **Repository Pattern**: LinkedList + HisContext 作为数据仓储
- **Service Pattern**: 各 Service 类实现业务逻辑分离
- **Composition Root**: HisCore 集中管理依赖注入
- **DTO Pattern**: 通过 toJson()/fromJson() 实现数据传输对象
- **Template Method**: 链表的泛型操作模板
- **Observer Pattern**: Logger 记录所有关键操作

17.2 业务规则

- 患者在住院状态中禁止删除 (Patient.canBeRemoved())
- 药品库存不得为负 (MedicineService.decreaseStock() 库存不足时返回 false)
- 同一物理床位同一时间仅能属于一名患者
- 办理住院时无空闲床位则业务拒绝
- 科室下有医生或药品时不能删除该科室
- 出院记录必须包含出院小结
- 所有实体使用 UUID 作为唯一标识

18 总结

HIS 医院信息管理系统通过精心设计的数据结构和分层架构，实现了医院信息管理的核心功能：

- 1. **高效的数据操作**：通过 LinkedList 混合结构提供 $O(1)$ 的查找、插入和删除
- 2. **完整的数据模型**：覆盖患者、医生、药品、病房、检查项目、科室、病例、支付、结算 9 大实体
- 3. **清晰的业务流程**：从患者挂号、诊疗、检查、开药到住院、出院、支付、结算的完整流程
- 4. **双界面支持**：CLI 命令行 + Qt6 GUI 图形界面，适配不同使用场景
- 5. **多角色视角**：GUI 支持管理/病人/医护三种权限视图
- 6. **完善的支付结算**：支持多种支付方式、医保结算、财务报表和统计图表
- 7. **灵活的扩展性**：Service 层设计使系统易于扩展新功能
- 8. **数据持久化**：JSON 序列化支持数据的保存和恢复

系统的核心优势在于数据结构设计的平衡性——通过链表 + 哈希表的混合使用，既保证了查询效率，又支持了灵活的数据遍历和修改操作。

A 数据结构时间复杂度对比

操作	LinkedList	普通 Vector	HashMap
查找	$O(1)$	$O(n)$	$O(1)$
插入	$O(1)$	$O(n)$	$O(1)$
删除	$O(1)$	$O(n)$	$O(1)$
遍历	$O(n)$	$O(n)$	$O(n)$

表 21: 数据结构性能对比