目录

《HIS 医疗管理系统设计说明书（2025 级）》

（需求功能与系统约定）

1　系统概述与设计目标

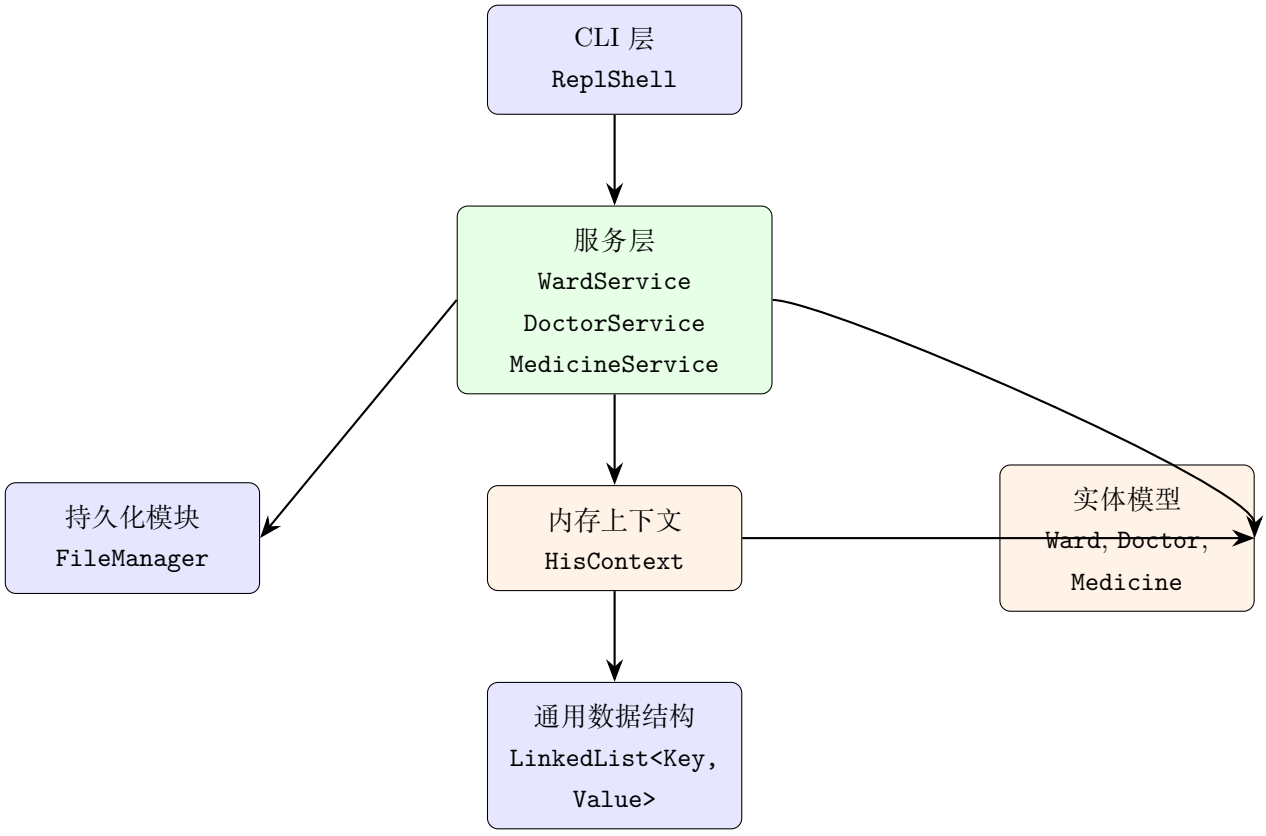
本系统为一个面向小型医院的轻量级医疗信息管理系统（Hospital Information System, HIS），采用 C++ 纯链表实现。通过模块化设计，实现业务逻辑与交互界面的解耦，保障系统的高扩展性与可维护性。

- **业务闭环**：实现完整的挂号、诊疗、检查、住院、开药等医疗生命周期。
- **数据规模**：支持至少 100 名门诊患者、30 名住院患者、20 名医生、5 个科室及 20 类药品。
- **底层要求**：全程基于手写**链表**实现内存数据流转，采用文本文件（JSON 数组）持久化。
- **鲁棒性**：具备极强的异常拦截能力，从容应对测试工程师的极端输入场景（防崩溃设计）。

2　系统架构与项目目录结构

系统采用分层架构设计（CLI 交互层 → Service 逻辑层 → Model 实体层 → Storage 持久层）。

- **CLI 层**：命令解析与结果输出
- **Service 层**：业务规则处理
- **Model 层**：数据结构定义
- **Storage 层**：数据存储与文件持久化



`load` 逻辑：`WardService` 先清空 `HisContext.wards` 再读文件填充，通常不造成累积内存泄漏。

图 1: HIS 系统模块架构图

项目工程文件结构如下：

```
his-project/
  CMakeLists.txt              # 根构建配置
  data/                       # 原始数据目录（评测时展示的文件）
    patients.txt              # 【存储JSON数组的文本文件】至少30条记录
    doctors.txt
    wards.txt                 # 病房信息
    medicines.txt             # 药品库
  include/                    # 头文件目录（.h / .hpp）
    models/                   # 1. 纯数据实体层（Entity）
      patient.h
      doctor.h
      medicine.h
    utils/                    # 2. 底层工具与驱动层
      linked_list.hpp         # 【核心】全手写的泛型链表 template<T>
      file_manager.h          # 文件读写封装
      e2hang_json.h           # 你的 C++20 JSON 解析库头文件
    core/                     # 3. 核心业务层（his-core）
      his_context.h           # 全局上下文（持有所有内存链表的实例）
      patient_service.h       # 处理挂号、看诊等纯逻辑
      report_service.h        # 报表统计算法
    cli/                      # 4. 交互表现层（his-cli）
      repl_shell.h            # 交互式命令行主循环
      format_printer.h        # 负责把链表数据打印成漂亮的 ASCII 表格
  src/                        # 源文件目录（.cpp）=> 源文件目录（实现）
    main.cpp                  # 程序唯一入口：初始化、启动REPL和退出保存
    models/                   # 实体类方法实现
      patient.cpp             # 患者属性操作及 JSON 序列化实现
      doctor.cpp              # 医生属性操作
      ward.cpp                # 病房/床位逻辑实现
      medicine.cpp            # 药品多名称匹配逻辑实现
    core/                     # his-core：核心业务实现（无输入输出）
      his_context.cpp         # 全局单例：管理所有内存链表的加载与保存
      patient_service.cpp     # 实现挂号、诊疗记录、病历生成逻辑
      inpatient_service.cpp   # 实现住院分配、床位流转逻辑
      pharmacy_service.cpp    # 实现处方发药、库存增减、入库逻辑
      report_service.cpp      # 实现多维度报表统计与数据分析算法
    utils/                    # 基础设施实现
      file_manager.cpp        # 处理 .txt 文件的物理读写
      input_sanitizer.cpp     # 【鲁棒性核心】针对测试工程师的输入清洗
      e2hang_json.cpp         # 你的 C++20 JSON 解析库实现
    cli/                      # his-cli：命令行交互实现
      repl_shell.cpp          # 交互式 Shell 主循环（REPL）
      command_parser.cpp      # 指令分发器：将输入映射到 Service 函数
      table_printer.cpp       # 美化工具：将结果格式化为 ASCII 表格
  tests/                      # 测试用例目录
  test_linked_list.cpp    # 必须先确保链表绝对可靠
  test_core_logic.cpp     # 灌入异常数据测试业务层鲁棒性
```

3 数据实体与约定

3.1 通用数据约定

- **唯一性**：所有系统实体均采用**唯一 ID** 作为数据主标识。
- **宽容性**：**允许患者姓名重复**，系统必须在底层支持重名/同名情况下的精准区分。
- **存储规约**：所有内存态数据全程挂载并依托于**自定义泛型链表**进行流转；数据的磁盘持久化统一采用 **JSON 数组**格式存储于对应的 TXT 文件中。

3.2 患者（Patient）

- **实体属性**：PatientID（唯一主键）、姓名、年龄、性别、联系方式、当前状态（门诊 / 住院 / 已出院）。
- **系统约定**：一名患者的生命周期内可对应多条医疗记录；系统查询接口必须支持按患者姓名的**模糊查询**。

3.3 医生（Doctor）

- **实体属性**：DoctorID（唯一主键）、姓名、所属科室 ID、职称（主任 / 副主任 / 主治 / 住院医师）、出诊时间安排。
- **系统约定**：每位医生有且仅隶属于一个明确的科室；为保证医院各科室的接诊流转能力，约定**每个科室至少包含 3 名注册医生**。

3.4 科室（Department）

- **实体属性**：DepartmentID（唯一主键）、科室名称。

3.5 医疗记录（MedicalRecord）【核心】

- **实体属性**：RecordID（唯一主键）、关联的 PatientID、关联的 DoctorID、业务类型（挂号 / 看诊 / 检查 / 住院）、发生时间、详细结构化信息。
- **系统约定**：一条合法的医疗记录**必须同时强关联**当前存在的患者与医生；为保障防篡改性与溯源严肃性，医疗记录**不可直接修改**，必须采用“撤销废除 + 重建新单”的机制。

3.6 病房与床位（Ward & Bed）

- **病房属性**：WardID（唯一主键）、病房类型（普通 / 特殊 / ICU 等）、关联科室 ID、最大床位容量。
- **床位属性**：BedID（唯一主键）、所属病房 ID、当前状态（空闲 / 已占用）、绑定的患者 ID。
- **系统约定**：同一时间段内，一个物理床位**仅能属于一名患者**；系统须支持基于患者住院/出院动作的**床位动态分配与自动释放**。

3.7 药品（Medicine）

- **实体属性**：MedicineID（唯一主键）、通用名、商品名、别名、当前库存数量、所属科室、单价。
- **系统约定**：检索接口必须支持**多名称混合匹配**；在处方发药与出库操作中引入强校验拦截，任何情况下**药品库存均不得为负**。

4 核心功能需求

4.1 数据管理功能

- **（1）增加操作**
  - **操作对象**：添加患者、医生、医疗记录、药品信息。
  - **约束要求**：支持单条命令行输入与 TXT 文件批量导入；自动校验合法性。
- **（2）修改操作**
  - **操作对象**：支持修改患者基本信息、修改药品属性信息。
  - **业务约定**：医疗记录属于核心溯源数据，**不可直接修改**。
- **（3）删除操作**
  - **操作对象**：删除患者档案、删除医疗记录、删除下架药品。
  - **业务约束**：必须进行严谨的**依赖检查**（例如：若患者当前仍在住院，则系统必须拦截并拒绝删除操作）。

4.2 查询功能

系统需支持灵活的多维度数据检索：

- **按患者查询**：查询指定患者完整的历史就诊记录与生命周期。
- **按医生查询**：查询特定医生的挂号列表与诊疗情况。
- **按科室查询**：查询该科室的人员构成与整体运转信息。
- **按药品查询**：根据名称查询药品的实时库存与单价信息。
- **技术要求**：必须支持按姓名/药品名称的**模糊查询**，且支持**排序处理**。

4.3 住院管理功能

- **业务操作**：为患者分配床位、办理出院处理、实时更新床位状态。
- **系统约定**：办理住院时若无空闲床位，系统需自动拒绝；办理出院后系统需自动释放占用的床位资源。

4.4 药房管理功能

- **业务操作**：药品入库（补给）、药品出库、凭医生处方发药。
- **系统约定**：发药需强校验，**库存不足时明确禁止发药**并警报提示；处方开药数量须设上限。

4.5 报表与统计功能

系统需支持以下维度的核心统计报表：

- **医院营业额统计**：综合汇总各项检查费用、药品开销与住院费用。
- **医生工作量统计**：统计各医生/各科室的接诊患者数量与频次。
- **当前住院患者报表**：生成当前全院或指定科室仍在住院的患者清单。
- **床位使用率统计**：计算并展示各病房/科室床位的实时占用百分比。
- **药品库存统计**：一览当前所有药品的库存余量状况及高频消耗趋势。

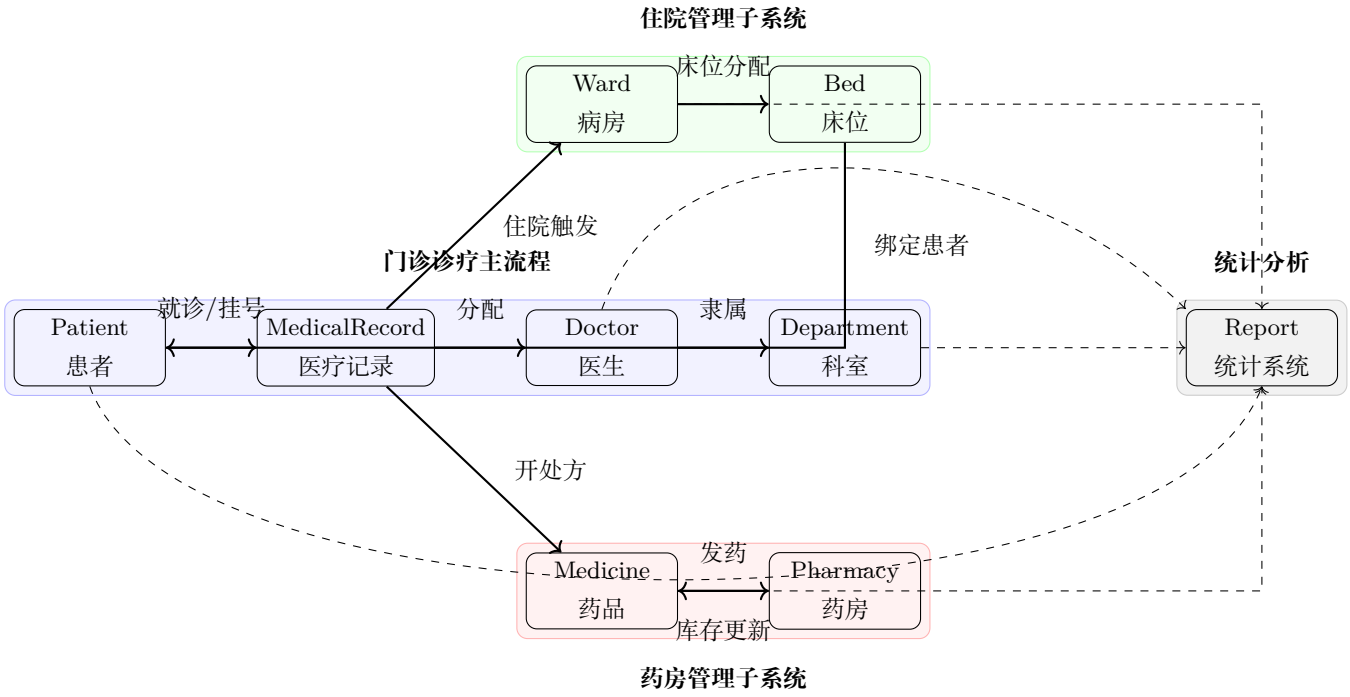
4.6 业务流程管理

- **住院管理**：执行床位检索 → 分配绑定 → 状态更新 → 出院释放。无空床时业务拒绝。
- **药房管理**：处理入库补给、处方强校验发药（库存不足禁止发药）、单次开药限量限制。

4.7 报表与统计（多权限视角）

支持格式化的 ASCII 表格打印输出：医院综合营业额、医生工作量排行、住院患者一览表、全院/科室床位使用率统计、药品库存消耗趋势。

5 系统核心交互流程图（完整业务流）



6 交互层 (CLI) 与鲁棒性设计

6.1 REPL 命令行交互

系统封装了交互式 Shell，支持如 `his > add patient`、`his > admit P001`、`his > report doctor` 等语义化指令，参数输入灵活，错误提示友好。

6.2 极致鲁棒性（防崩溃拦截）

结合 `input_sanitizer.cpp` 提供全方位输入保护：

- **输入异常**：拦截非法字符（防命令注入）、阻断空输入与超长字符串、自动纠正格式错误。
- **业务异常**：妥善处理无空床、库存穿透、非法 ID 关联引用、并发重名等业务死锁状态。

7 数据持久化与扩展机制

系统根目录 `data/` 下维持 `patients.txt` 等存储媒介。采用 JSON 数组格式。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。后期可进一步无缝拓展：历史数据预测分析（如病床流转预测）、医生智能推荐以及多角色权限 (RBAC) 认证系统。

医院信息系统（HIS）
数据结构与系统架构说明文档

代码分析

2026 年 4 月 1 日

摘要

本文档基于 HIS（Hospital Information System）项目源代码详细分析，阐述了系统的核心数据结构、数据模型、存储机制及各个服务模块的功能设计。HIS 是一个医院信息管理系统，用于管理患者、医生、病房、药物及医疗记录等关键信息，采用链表 + 哈希表混合数据结构实现高效的数据操作。

8 系统概述

8.1 项目定义

医院信息系统（HIS, Hospital Information System）是一个综合性医院管理平台，旨在实现对医院各项资源和流程的数字化管理，包括：

- 患者信息管理
- 医生、科室管理
- 病房和床位管理
- 药物库存管理
- 患者医疗记录管理（诊断、处方、住院等）
- 医疗数据分析与报告

8.2 架构设计

系统采用分层架构设计：

- **数据模型层（Models）**：定义各个实体对象
- **数据存储层（Utils）**：提供通用的链表和哈希数据结构
- **业务逻辑层（Core Services）**：实现 CRUD 操作和业务规则
- **表现层（CLI）**：命令行交互界面

9 核心数据结构

9.1 通用存储结构：LinkedList

LinkedList 是系统的基础数据结构，采用双向链表 + 哈希表的混合设计：

9.1.1 结构特点

9.1.2 模板定义

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	通过哈希表快速定位	$O(1)$
删除操作	从链表中移除并更新哈希表	$O(1)$
遍历操作	按链表顺序遍历所有元素	$O(n)$

表 1: LinkedList 操作复杂度

```
template<class Key, class Value>
class LinkedList {
        private:
        ListNode<Key, Value>* head;      // 虚拟头节点
        ListNode<Key, Value>* tail;      // 虚拟尾节点
        std::unordered_map<Key,
        ListNode<Key, Value>*> mp;   // 哈希表索引
        size_t sz;                        // 当前元素数量
};
```

9.2 全局上下文：HisContext

9.2.1 职责

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。

9.2.2 定义

```
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, PatientCase>
                patientCases;
        };
}
```

10 核心数据实体

10.1 患者实体：Patient

10.1.1 用途

存储患者的基本信息，包括个人资料和当前就诊状态。

10.1.2 属性定义

属性名	类型	含义	是否主键
PatientID	string	患者唯一标识	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

10.1.3 就诊状态枚举

```
enum class PatientStatus {
        Outpatient,   // 门诊
        Inpatient,    // 住院
        Discharged    // 已出院
};
```

10.1.4 关键方法

- `updateBasicInfo()` - 更新患者基本信息
- `canBeRemoved()` - 检查患者是否可删除（住院中禁止删除）
- `nameMatches()` - 名字模糊匹配查询
- `toJson()/fromJson()` - JSON 序列化接口

10.2 医生实体：Doctor

10.2.1 用途

记录医生的专业信息，包括科室归属和职务等级。

10.2.2 属性定义

属性名	类型	含义
DoctorID	string	医生唯一标识
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

10.2.3 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
};
```

10.3 病房与床位实体：Ward & Bed

10.3.1 用途

管理医院的住院资源，包括病房的床位分配和患者住院状态。

10.3.2 Bed 结构

属性名	类型	含义
BedID	string	床位唯一标识
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 4: Bed 属性表

10.3.3 Ward 大门结构

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

表 5: Ward 属性表

10.3.4 病房类型枚举

```
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
    };
```

10.3.5 Ward 关键方法

- `getFreeBedID()` - 获取第一个空闲床位
- `admitPatient()` - 患者住院分配床位
- `releaseBed()` - 释放指定床位
- `releasePatient()` - 根据患者 ID 释放床位
- `addBed()/removeBed()` - 床位增减操作
- `freeBedCount()` - 获取空闲床位数
- `occupancyRate()` - 计算床位使用率
  占用率计算公式：

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

10.4 药物实体：Medicine

10.4.1 用途

管理医院的药物库存，支持快速查询和库存管理。

10.4.2 属性定义

属性名	类型	含义
MedicineID	string	药物唯一标识
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 6: Medicine 属性表

10.4.3 关键方法

- `updateBasicInfo()` - 更新药物信息
- `nameMatches()` - 支持通用名、商品名及别名匹配
- `decreaseStock()` - 减少库存
- `addAlias()` - 添加别名

10.5 患者病例实体：PatientCase

10.5.1 用途

患者病例是一个聚合实体，记录患者的所有医疗记录，包括诊断、处方、住院等历史信息。

10.5.2 包含的子记录类型

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 7: DiagnosisRecord 属性

1. 诊断记录 (DiagnosisRecord)

属性	类型	含义
MedicineID	string	药物 ID
MedicineName	string	药物名称
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 8: MedicineRecord 属性

2. 药房记录 (MedicineRecord) 药房费用计算：

$$\text{TotalPrice} = \text{Quantity} \times \text{UnitPrice}$$

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 9: AdmissionRecord 属性

3. 住院记录 (AdmissionRecord)

4. 预约记录 (AppointmentRecord)

10.5.3 PatientCase 顶层属性

10.5.4 关键方法

- `addDiagnosisRecord()` - 添加诊断记录
- `addMedicineRecord()` - 添加药房记录

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	预约备注
Timestamp	time_t	预约创建时间

表 10: AppointmentRecord 属性

属性名	类型	含义
PatientID	string	患者 ID
DiagnosisRecords	vector	诊断记录列表
MedicineRecords	vector	药房记录列表
AdmissionRecords	vector	住院记录列表
AppointmentRecords	vector	预约记录列表
CreatedTime	time_t	病例创建时间
LastModifiedTime	time_t	最后修改时间

表 11: PatientCase 属性表

- `addAdmissionRecord()` - 添加住院记录
- `addAppointmentRecord()` - 添加预约记录
- `dischargeFromLatestAdmission()` - 出院并填写出院小结
- `getCurrentAdmission()` - 获取当前住院记录（如果有）
- `getLatestAdmission()` - 获取最后一次住院记录
- `getTotalMedicineCost()` - 累计药物费用
- `getLatestDiagnosis()` - 获取最新诊断

11 业务逻辑层：服务类

11.1 患者服务：PatientService

11.1.1 职责

- 患者的 CRUD 操作
- 患者状态管理（门诊/住院/出院）
- 患者住院和出院流程
- 多维度查询（按 ID、姓名、联系方式）

11.1.2 关键接口

```
class PatientService {
        public:
        // 查询操作
        const Patient* findPatient(const std::string& patientId) const;
        void findByName(const std::string& keyword, ...);
        void findByContact(const std::string& keyword, ...);
```

```
        // 修改操作
        bool addPatient(const Patient& p);
        bool updatePatient(const std::string& patientId, ...);
        bool removePatient(const std::string& patientId, ...);

        // 住院/出院
        bool admitPatient(const std::string& wardId,
        const std::string& patientId, ...);
        bool releasePatient(const std::string& wardId,
        const std::string& patientId);
};
```

11.2 病房服务：WardService

11.2.1 职责

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

11.2.2 关键接口

```
class WardService {
    public:
    // 查询
    const Ward* findWard(const std::string& wardId) const;

    // 修改
    bool addWard(const Ward& w);
    bool removeWard(const std::string& wardId);
    bool addBed(const std::string& wardId, ...);
    bool removeBed(const std::string& wardId, ...);

    // 持久化
    bool loadFromFile(const std::string& path, ...);
    bool saveToFile(const std::string& path, ...);
};
```

11.3 患者病例服务：PatientCaseService

11.3.1 职责

- 管理患者的完整医疗记录
- 诊断、处方、住院、预约记录的添加和查询
- 患者出院流程处理
- 医疗统计（费用、记录数等）

11.3.2 关键接口

```
class PatientCaseService {
        public:
        // 病例获取
        PatientCase* getOrCreateCase(const std::string& patientId);
        const PatientCase* getCase(const std::string& patientId) const;

        // 记录添加
        bool addDiagnosisRecord(const std::string& patientId, ...);
        bool addMedicineRecord(const std::string& patientId, ...);
        bool addAdmissionRecord(const std::string& patientId, ...);
        bool addAppointmentRecord(const std::string& patientId, ...);

        // 出院处理
        bool dischargePatient(const std::string& patientId,
        const std::string& dischargeSummary);

        // 统计查询
        double getTotalMedicineCost(const std::string& patientId) const;
        size_t getDiagnosisRecordCount(const std::string& patientId) const;
};
```

11.4 医生服务：DoctorService

11.4.1 职责

- 医生信息的 CRUD 操作
- 医生查询和筛选

11.5 药物服务：MedicineService

11.5.1 职责

- 药物库存管理
- 药物查询和匹配
- 库存增减操作

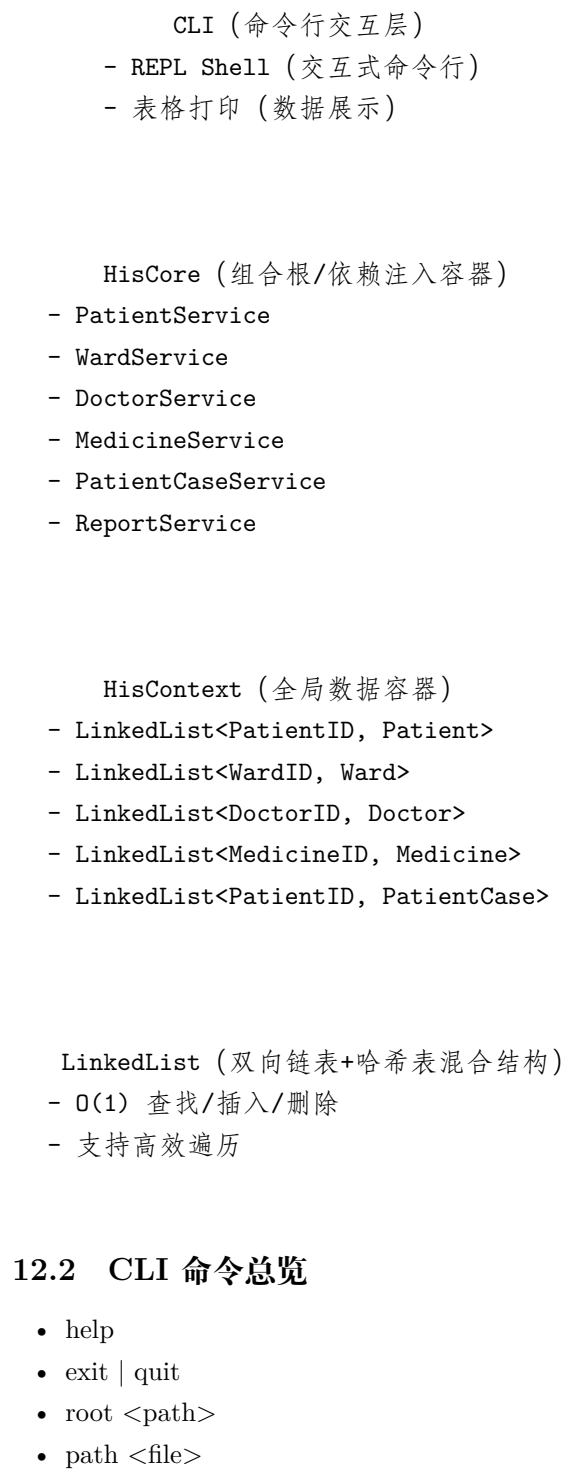
11.6 报告服务：ReportService

11.6.1 职责

- 生成系统统计报表
- 数据分析展示

12 系统架构图

12.1 数据流架构



12.2 CLI 命令总览

- help
- exit | quit
- root <path>
- path <file>
- doctor add <doctorId> <name> <departmentId> <Chief|AssociateChief|Attending|Resident> <schedule>
- doctor rm <doctorId>
- doctor list
- doctor list dept <departmentId>

- doctor search <keyword>
- doctor search name <name_keyword>
- doctor search dept <departmentId>
- doctor search title <title_keyword>
- doctor search schedule <schedule_keyword>
- medicine add <id> <genericName> <brandName> <departmentId> <stock> <unitPrice>
- medicine list
- medicine stock inc <id> <amount>
- medicine stock dec <id> <amount>
- medicine find <keyword>
- medicine load [file_path]
- medicine save [file_path]
- ward add <wardId> <departmentId> <Normal|Special|ICU> <maxBeds>
- ward rm <wardId>
- ward list
- ward show <wardId>
- ward bed add <wardId> <bedId>
- ward bed rm <wardId> <bedId>
- patient add <patientId> <name> <age> <gender> <contact> [Outpatient|Inpatient|Discharged]
- patient update <patientId> <name> <age> <gender> <contact>
- patient rm <patientId>
- patient list
- patient search id <patientId_keyword>
- patient search name <name_keyword>
- patient search phone <contact_keyword>
- patient search <keyword>
- patient load [file_path]
- patient save [file_path]
- patient admit <wardId> <patientId>
- patient release bed <wardId> <bedId>
- patient release patient <wardId> <patientId>
- case view <patientId>
- case diagnosis add <patientId> <doctorId> <diagnosis> [prescription] [remarks]
- case medicine add <patientId> <medicineId> <medicineName> <quantity> <usage> <unitPrice>
- case admission add <patientId> <wardId> <bedId> [reason]
- case appointment add <patientId> <doctorId> <date> [notes]
- case discharge <patientId> [summary]
- case stats <patientId>
- report occupancy <wardId>
- log view [count]
- log clear
- log export <file_path>
- log format set <format_string>
- file create <file_path>
- file delete <file_path>

- file rm-field <file_path> <field>

12.3 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（药房记录）
                1:n  > AdmissionRecord（住院记录）
                1:n  > AppointmentRecord（预约记录）

  M:1  > Ward（病房）
  1:n  > Bed（床位）
  M:1  > Department（科室）

Doctor（医生）
  M:1  > Department（科室）

Medicine（药物）
  M:1  > Department（科室）
```

13 关键业务流程

13.1 患者住院流程

1. 患者通过 PatientService.addPatient() 注册为门诊患者
2. 医生通过 PatientCaseService.addDiagnosisRecord() 添加诊断
3. 需要住院时，调用 PatientService.admitPatient()：
   - 查找 WardService 中的可用病房
   - 从病房获取空闲床位
   - 分配患者到该床位
   - 更新患者状态为 Inpatient
4. 住院期间，通过 PatientCaseService 添加诊断和处方
5. 出院时，调用 PatientCaseService.dischargePatient()：
   - 填写出院小结
   - 释放床位
   - 更新患者状态为 Discharged

13.2 处方和药物管理流程

1. 医生通过 PatientCaseService.addMedicineRecord() 开药
2. 系统记录药物 ID、数量、用法、单价
3. 根据处方自动计算费用（数量 × 单价）
4. 可通过 PatientCaseService.getTotalMedicineCost() 查询患者总费用

13.3 床位资源管理流程

1. 通过 WardService 初始化病房及床位
2. Ward 对象维护床位列表（BedStatus）
3. 患者住院时调用 Ward.admitPatient()：
   - 查找第一个空闲床位
   - 将患者 ID 分配给床位
   - 更改床位状态为 Occupied
4. 患者出院时调用 Ward.releasePatient()：
   - 根据患者 ID 找到占用的床位
   - 释放床位（PatientID 置空）
   - 更改床位状态为 Free

14 数据持久化

14.1 JSON 序列化策略

系统中所有数据模型都支持 JSON 序列化，通过 `toJson()` 和 `fromJson()` 接口实现。

14.2 文件存储

- `wardservice.cpp` 提供 `loadFromFile()` 和 `saveToFile()` 方法
- 支持将 Ward 数据持久化到 JSON 文件
- 系统启动时自动加载已保存的数据

14.3 必需的数据文件

系统需要以下基础数据文件（位于 data/ 目录）：

- `patients.txt` - 患者信息
- `doctors.txt` - 医生信息
- `wards.txt` - 病房信息
- `medicines.txt` - 药物库存

15 设计模式与最佳实践

15.1 采用的设计模式

- **Repository Pattern**: LinkedList + HisContext 作为数据仓储
- **Service Pattern**: 各 Service 类实现业务逻辑的分离
- **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**: 链表的通用操作模板

15.2 数据结构优化

- 双向链表支持高效的插入/删除（$O(1)$）
- 哈希表索引支持高效查询（$O(1)$）

- 混合设计兼顾查询性能和遍历灵活性

15.3 业务规则

- 患者在住院状态中禁止删除（Patient.canBeRemoved()）
- 床位容量受限于病房的 MaxBeds 上限
- 出院记录必须包含出院小结
- 每个患者只能有一个” 当前住院记录”

16 系统使用示例

16.1 创建患者并住院

```
HisCore his;

// 创建患者
Patient p("P001", "张三", 35, "男", "13800000001");
his.patientService.addPatient(p);

// 患者住院
std::string bedId;
his.patientService.admitPatient("W001", "P001", bedId);

// 添加诊断
DiagnosisRecord diag("D001", "感冒", "板蓝根冲剂");
his.patientCaseService.addDiagnosisRecord("P001", diag);

// 添加药房记录
MedicineRecord med("M001", "板蓝根", 2,
"一日两次，饭后服用", 25.0, "D001");
his.patientCaseService.addMedicineRecord("P001", med);

// 患者出院
his.patientCaseService.dischargePatient("P001", "病情好转，出院");
```

17 总结

HIS 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 $O(1)$ 的查找和修改
2. **完整的数据模型**：覆盖患者、医生、病房、药物及医疗记录
3. **清晰的业务流程**：从患者挂号、诊疗到出院的完整流程支持
4. **灵活的扩展性**：Service 层设计使系统易于扩展新功能
5. **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表

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 12: 数据结构性能对比