

AISP-PRD-Nick、广忠、路晨小组

AISP (AI 标准化病人) 产品需求文档 (PRD)

文档信息

项目	内容
文档名称	AISP (AI 标准化病人) PRD
版本号	v0.3-MVP
撰写人	王烨、广忠、路晨
更新日期	2026/1/19
产品名称	AISP - AI 标准化病人
开发周期	3 周

变更日志

时间	版本号	变更人	主要变更内容
1月13日	v0.1	王烨、路晨	梳理 AISP项目的调研
1月16日	V0.1	王烨	梳理需求，梳理 PRD文档
1月16日	V0.2	广忠	添加技术架构设计文档
1月19日	V0.3	王烨	增加变更日志，增加 4.2 节数据表、增加竞品分析
1月20日	v0.4	广忠	对 PRD 文档进行补充（优化原有文档、新增竞品分析），技术架构更新

1. 产品基本信息

1.1 产品定位

AISP（AI 标准化病人）是一款面向医学教育场景的智能标准化病人问诊模拟与考核评估平台，采用 Multi Agent 架构，从问诊交互、病情反馈、教学指导等多个角度提供服务，模拟不同类型病人的真实就医交互场景。其核心目标是解决传统标准化病人（SP）成本高（需专业人员培训、薪资支出）、难以量化评估（问诊效果依赖主观判断）和无法大规模扩展（受人员数量限制）的痛点，让每个医学生、规培医生都能随时随地获得高质量的沉浸式临床训练体验，提升问诊技巧与病情判断能力。

AISP 是一个 AI 驱动的标准化病人模拟系统，主要应用于：

- 医学生临床技能训练（OSCE 考试）
- 大模型医疗 Agent 的训练与评估
- 医疗产品 Demo 与中试基地
- 诊疗规范教学与演示

1.2 产品价值

维度	传统 SP 痛点	AISP 解决方案	价值提升
成本	单次培训成本高（培训津贴 + 管理费用）	一次投入，无限次使用	成本降低 90%+
训练频次	SP 资源有限，学生训练机会少	随时随地可训练	训练频次提升 10 倍 +
标准化	人工表演一致性难保证	AI 始终按照脚本标准化输出	一致性 100%
可量化	评估依赖考官主观判断	全流程数据自动采集	量化评估体系

1.2.1 对用户（医学生、规培医生）

- 低成本高频训练：无需依赖实体 SP，可通过平台随时随地发起问诊模拟，打破时间与空间限制，降低训练成本，提升训练频次。
- 量化能力反馈：系统可基于问诊流程完整性（如是否覆盖核心症状询问）、病情判断准确性（如是否正确识别病症类型）、沟通技巧（如是否使用温和话术）等维度生成量化评估报告，帮助用户明确自身短板。
- 个性化学习路径：根据用户训练数据，智能推荐薄弱场景（如儿科问诊、急诊问诊）进行针对性训练，提升学习效率。

1.2.2 对机构（医学院校、医院规培部门）

- 降本增效：减少实体 SP 的招聘、培训与管理成本，同时支持批量学员同步训练，提升教学资源利用率。

- 标准化教学：统一问诊模拟场景与评估标准，避免因不同 SP 的表现差异导致的教学质量不稳定问题。
- 数据化管理：实时统计学员训练进度、成绩分布等数据，便于教学管理与效果复盘，为教学方案优化提供数据支撑。

1.2.3 对行业（医学教育领域）

- 推动教学数字化：以 AI 技术重构临床训练模式，填补传统教学中“实践机会少、评估不精准”的空白，助力医学教育数字化转型。
- 沉淀行业资源：长期积累的问诊场景数据（脱敏后）可形成医学教育知识库，为后续教学研究与 AI 模型优化提供支撑。

1.3 竞品分析

公司名称	核心技术特点	目标用户 / 场景	竞争优势
医微机·临床思维训练系统	专业医学教育 AI 平台	医学教育、医生继续教育	病例库、问诊模拟、考核评分
易训·SP 智能模拟系统	专业医学教育 AI 平台	住院医师规范化培训、临床技能考核	标准化病人交互、视频回放评估
春雨医生·AI 问诊助手	泛医疗交互类工具	大众健康咨询（非教育场景）	患者端病情咨询、症状自查
ChatGPT health	SOTA LLM 对话引擎，医疗数据	暂未发布	基础模型能力领先，医疗数据丰富，7亿+月活，用户使用习惯
SimConverse	LLM 对话引擎，多语言支持	医学院问诊训练	全球领先，NLP 技术强
OpenEvidence	专业AI医学助手	临床医生用于实时诊疗决策支持	基于真实、权威、可追溯的医学证据输出（附带文献引用）
Body Interact	3D 虚拟现实，生理模拟引擎	急诊 / 重症护理	生理学真实度高
蚂蚁阿福	阿里大模型 医疗版 + 云服务	患者 /	云生态与算力优势
百川智能	医疗 agent		Baichuan-M3，评测AI 医疗能力突出
上海梦之路	虚拟病人，OSCE 考试系统	医学院 / 住院医师	本土化最深，考试体系对接
上海褚信	智能标准化病人整理解决方案	医学院 / 医院	有丰富智慧医学教育实践场景
Baidu 灵医	文心一言 4.0 医疗大模型	医院 / 科研 / 教育	医疗 LLM 能力强，生态广
讯飞医疗	语音识别 + 医疗知识图谱	基层医疗 / 培训	语音技术行业领先
Tencent 觅影	AI 数字人 + 多模态融合	医院服务 / 科普	数字人技术成熟

1.3.1 竞品核心能力对比（聚焦专业医学教育类）

评估维度	我方产品 (AISP)	医微讯 · 临床思维训练系统	易训 · SP 智能模拟系统
模型能力 - 交互自然度	基于 Multi Agent 架构，支持多轮对话上下文理解，模拟患者情绪波动（如焦虑、抗拒）	单轮对话为主，交互逻辑较固定，患者情绪模拟薄弱	支持多轮对话，但上下文衔接流畅度一般，情绪模拟仅覆盖基础类型
模型能力 - 泛化性	支持自定义病例参数，可扩展至全科场景（内科、外科、儿科等）	病例场景固定，仅覆盖内科、外科核心病症，扩展性弱	支持科室场景切换，但自定义参数维度少（仅病症名称、患者年龄）
评估能力 - 量化维度	5 个核心维度（问诊完整性、病情判断准确性等），支持自定义权重	3 个维度（流程完整性、答案准确性、时间控制），无自定义权重	4 个维度（沟通技巧、流程完整性、病情分析、人文关怀），权重固定
资源成本 - 部署方式	支持 SaaS 云部署（按人数订阅）、私有化部署（一次性付费），灵活适配不同机构需求	仅支持私有化部署，初期投入成本高（需购买服务器、支付部署费）	以 SaaS 部署为主，订阅费用较高（按年度收取，人均成本超 2000 元 / 年）
生态能力 - 第三方接入	预留 API 接口，支持对接医学院校教务系统（同步学员信息）、医院 HIS 系统（获取真实病例脱敏数据）	无第三方接口，数据闭环在系统内部，无法与外部系统联动	仅支持对接少数主流教务系统，兼容性较弱

1.3.2 竞品差异化优势与机会点

1.3.2.1 我方差异化优势

- 交互更真实：Multi Agent 架构相比竞品的单模型交互，能更细致地模拟患者的复杂反应（如隐瞒症状、重复提问），提升沉浸式体验。
- 成本更灵活：同时支持 SaaS 与私有化部署，满足中小型院校（偏好低成本 SaaS）与大型医院（重视数据安全，偏好私有化）的不同需求，相比竞品的单一部署方式更具竞争力。
- 评估更精准：量化维度更全面，且支持机构自定义权重，可适配不同院校的教学侧重点（如部分院校更重视人文关怀，可提高该维度权重）。

1.3.2.2 市场机会点

- 罕见病场景覆盖：现有竞品罕见病病例较少，我方可优先补充该类场景，满足学员对特殊病例训练的需求，形成差异化壁垒。
- 移动端适配：多数竞品仅支持 PC 端使用，我方可开发移动端 APP，支持学员利用碎片化时间（如通勤、午休）训练，提升使用便利性。

- 教学资源联动：将问诊模拟与教材知识点关联（如训练某病例时，推荐教材对应章节），形成“学 - 练 - 考”闭环，提升产品对教学的支撑能力。

1.4 目标用户

用户类型	描述	面向场景	核心需求	功能模块权限
医学生	本科/研究生 临床医学专业学生	在非临床环境进行模拟问诊，熟悉问诊流程、积累基础病例经验	1. 随时随地练习问诊和临床思维 2. 获得即时反馈了解不足	学生端
规培医师	正在规培的医生，需要提高临床思维能力	用AISP病人训练不熟悉的病种，提升临床实操能力、适应不同科室问诊场景	1. 强化全科临床思维训练 2. 弥补临床经验不足 3. 系统化提升问诊技巧	学生端
带教老师	医学院教师、医院导师	布置问诊训练作业，通过平台查看学生完成情况；基于平台数据调整课堂教学重点	1. 减少重复性教学负担 2. 数据化了解学生表现 3. 精准化个性化教学指导	教师端
医学院校	教学管理部门	管理规培医生训练进度、统计教学成果	1. 降低教学成本 2. 提升教学质量标准化 3. 可量化的教学效果评估 4. 制定月度训练计划，监控规培医生完成率；生成规培阶段整体评估报告，上报至上级部门	教师端
产品运维人员	产品经理、研发、测试	功能验证、数据查看、系统配置、agent提示词修改	产品功能验证及测试	学生端、教师端、AISP agent

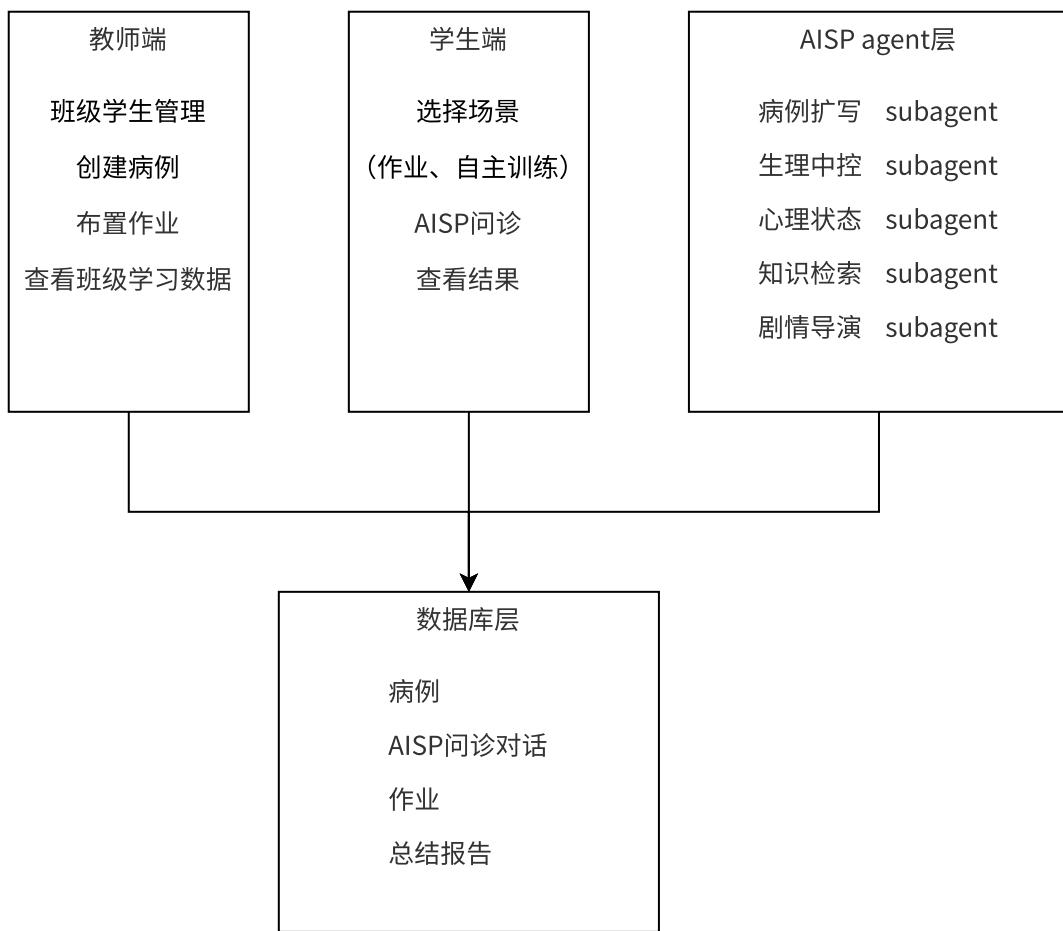
1.5 核心用户痛点

痛点类型	具体描述	产品解决方案
训练资源不足	实体 SP 数量有限，无法满足所有学员的高频训练需求；罕见病、重症等特殊病例的 SP 资源稀缺	平台内置海量病例库（覆盖常见病、罕见病、重症等场景），支持无限次模拟训练；通过 AI 生成特殊病例的症状表现与交互逻辑
评估缺乏量化	传统训练中，SP 对学员的评估依赖主观感受，无明确数据指标，学员难以定位问题	建立多维度量化评估模型（如问诊完整性 20%、病情判断准确性 30%、沟通技巧 25%、时间控制 15%、人文关怀 10%），自动生成评估报告
训练场景单一	实体 SP 模拟的场景固定，难以快速切换科室、病症严重程度等变量，学员适应能力不足	支持用户自定义场景参数（科室类型、病症名称、患者年龄 / 性别、症状严重程度），灵活生成多样化训练场景

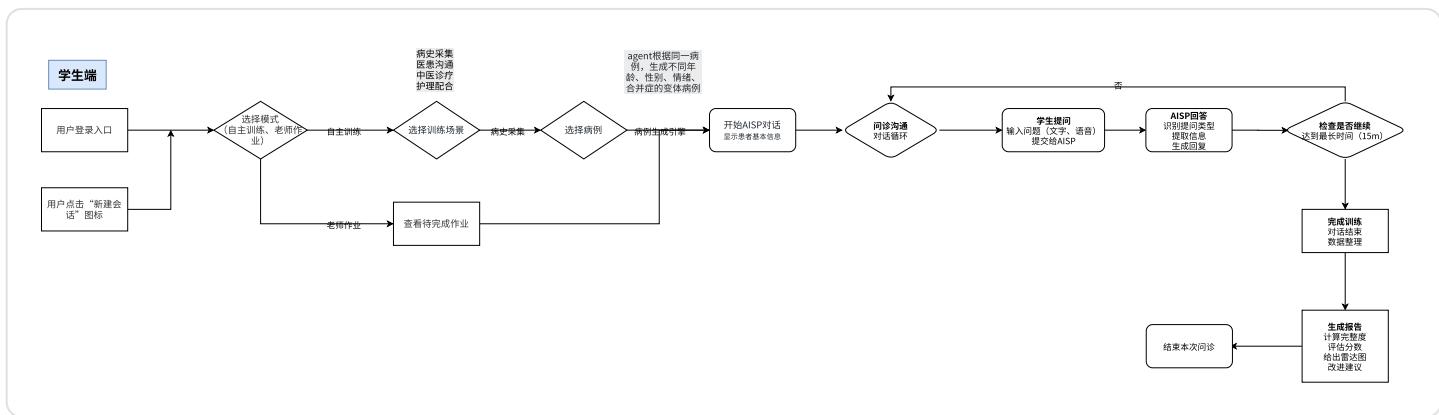
2. 产品设计概要

2.1 核心使用逻辑

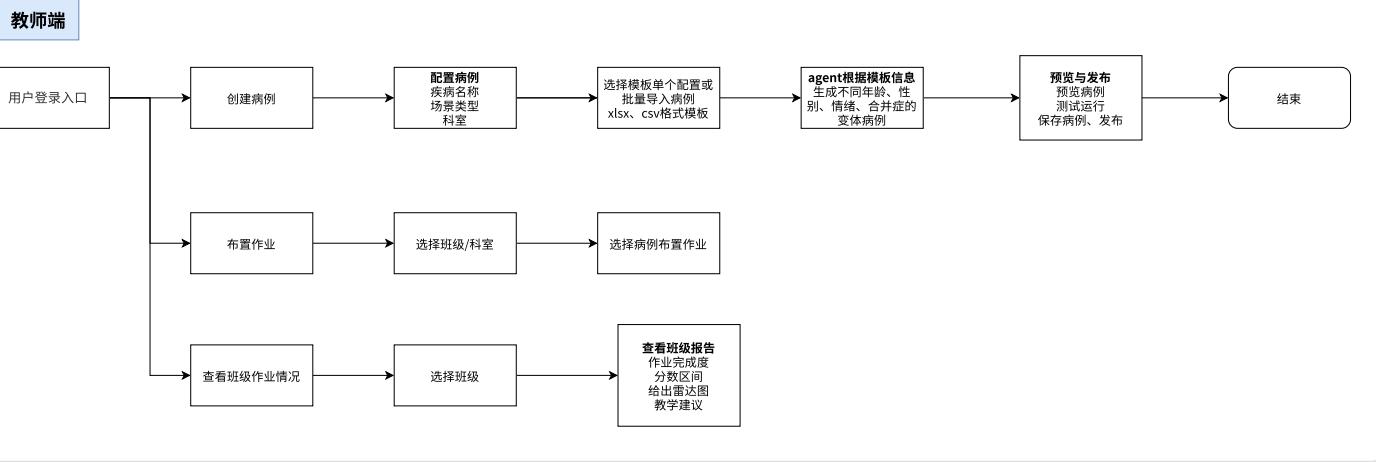
病史采集场景整体流程



学生端



教师端



2.2 功能框架

核心功能模块架构

- 用户学生端：面向医学生、规培医生，提供问诊训练、成绩查询、查看历史对话记录、继续历史对话功能。
- 教师端：面向教师、规培管理员，提供学员管理、病例 sp 配置管理、训练任务作业布置、数据统计（查看学生（医学生、规培医生）的 AISp 问诊对话成绩）功能。
- AI 核心引擎：支撑交互模拟、评估计算、病例生成的底层能力，包含 Multi Agent 模块、量化评估模块、病例库模块。

2.3 功能清单

序号	系统	模块	需求项	功能需求
		1.1 用户认证语 管理	1.1.1 用户注册	1) 支持手机号注册
			1.1.2 用户登录	1) 输入账号、密码、勾选用户协议、隐私策略后可
			1.1.3 密码找回	1) 通过邮箱、手机验证码找回手机
			1.1.4 个人信息	1) 查看个人信息、训练数据统计
		1.2.1 场景选择		1) 支持登陆账号时，自动新建对话
				2) 支持点击【新建对话】按钮，结束当前问诊，开 诊对话
				1) 选择模式：自主训练、完成作业
		1.2.2 训练场景		2) 病例浏览：显示病例列表，包含病例归属科室、 主诉标签
				3) 开始训练：进入对话界面，初始化 AISP
		1.2.3 对话输入		1) 支持输入文字回答问题
				2) 支持输入语音回答问题

			3) 支持输入语音转文本，编辑确认后发送回答
		1.2.4 病例扩写子 agent	1) 自主训练场景下，选择病例后，可 ai 生成不同年龄、情绪，合并症的变体病例。
			1) 支持多轮对话，对话最长 15 分钟结束
			2) 支持同一病种的标准化病人，匹配不同年龄、情绪，合并症的变体病例，以此为基础来对话。
			3) 对话历史显示：区分 AISP 和学生，实时显示对话记录
			4) AISP 生理中控子 agent：模拟AISP 在不同刺激下的反应
			5) AISP 心理状态子 agent：按主性格配置，根据学生表现不同情绪
			6) AISP 知识检索子 agent：能自动检索，相关病症
			7) AISP 剧情导演子 agent：能编排 aisp 的就诊完整流程
			8) 支持自动识别用户意图，当用户首轮问题非“何时”，提醒用户重新输入
			9) 支持病例库引用：支持按“科室”（内科 / 外科等）、“病症类型”（常见病 / 罕见病 / 重症）、“级别”（入门 / 中级 / 高级）筛选病例；展示病例简介（科室、年龄、初始症状）
			10) 支持问诊结束后，推送反馈结果（雷达图评分）
		1.2.6 评估	
			1) 支持敏感词监控，禁止用户发送敏感词内容
		1.2.7 通用交互	2) 支持回答过程中断网时，重新发送回答内容
		1.3.1 记录列表	1) 支持按时间轴展示历史对话记录
		1.3.2 历史对话查看	1) 支持点击历史对话，查看对话详情
		2.1.1 学生管理	批量导入学生信息（excel、CVS）
		2.1.2 学生列表	可查看所有学生基本信息，及训练统计
		2.2.1 病例模板选择	配置基础病例模板
		2.2.2 病例列表	显示所有病例，支持筛选和排序
		2.2.3 病例搜索	按疾病名称、科室、创建者搜索病例
		2.2.4 创建病例	填写病例基本信息，设置病例必填项和选填项，AI - 化病例。
		2.2.5 病例预览	查看生成的病例记录
		2.3.1 创建作业	创建作业，选择病例和学生/科室，设置截止时间

2.3 作业管理	2.3.2 作业列表	显示已布置作业及状态
	2.3.3 发布作业	将作业发布给学生
	2.4 作业报告查看	2.4.1 总体概率 2.4.2 薄弱点分析
2.4 作业报告查看	2.4.1 总体概率	AISP问诊作业完成情况（平均分、完成度）
	2.4.2 薄弱点分析	整体问诊遗漏点、薄弱点分析

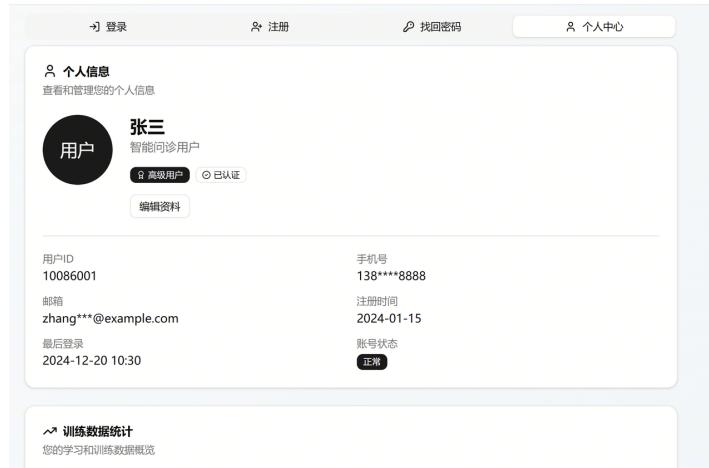
名词解释：AISP (AI 标准化病人)

2.4 原型图：

学生端：



1.1 用户登录界面



1.1 用户个人中心



1.2.1 训练场景选择



1.2.3 智能问诊text对话界面



1.2.3 语音输入界面

1.3 查看问诊历史记录

教师端

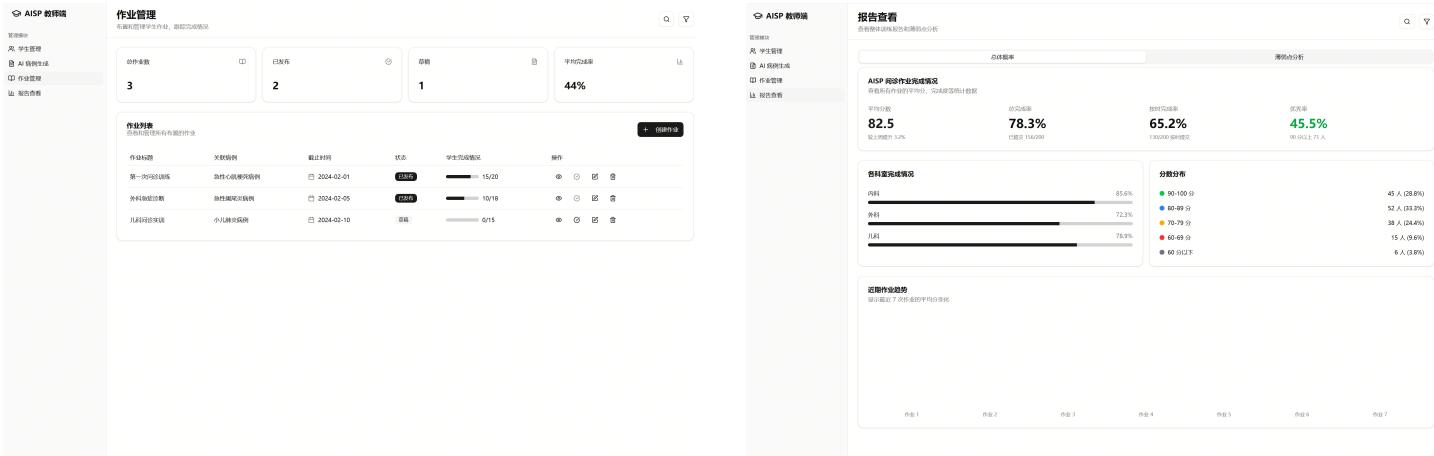
This screenshot shows the teacher management interface. On the left, the '学生管理' (Student Management) section displays student statistics: 4 students, average score 68%, and total score 82. It includes a table of student names, scores, and grades. On the right, the 'AI 病例生成' (AI Case Generation) section shows a table of cases: '内科通用模板' (Internal Medicine General Template) for '急性心肌梗死' (Acute Myocardial Infarction) with ID 'PMI', '外科通用模板' (Surgery General Template) for '急性阑尾炎' (Acute Appendicitis) with ID 'AA', and '儿科通用模板' (Pediatrics General Template) for '小儿肺炎' (Childhood Pneumonia) with ID 'JP'. Each template has a preview and edit button.

2.1 学生管理

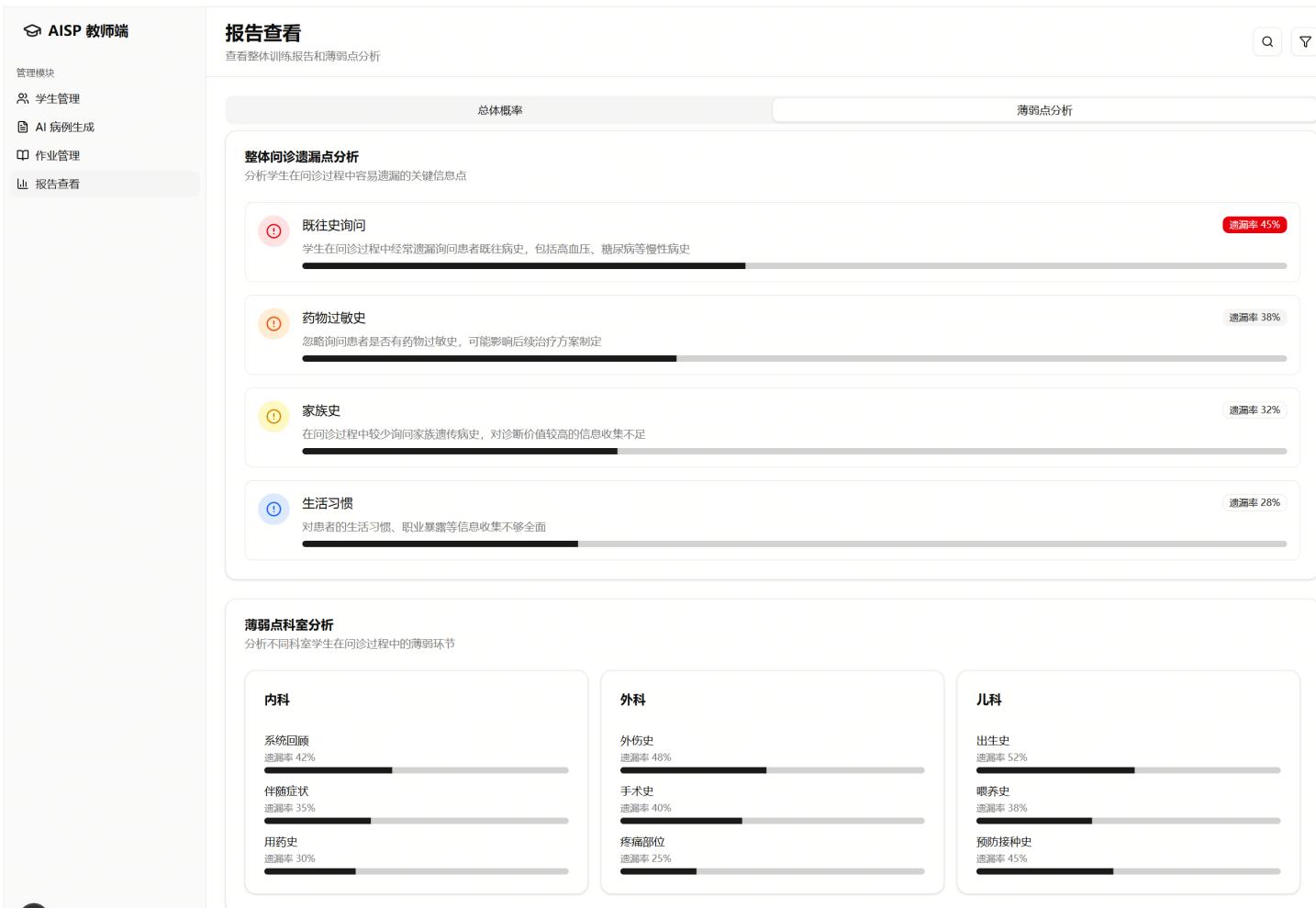
This screenshot shows the AI case generation and assignment management interface. On the left, the 'AI 病例生成' (AI Case Generation) section allows creating AI cases for different departments like Internal Medicine, Surgery, and Pediatrics. It includes fields for symptoms, operations, and notes. On the right, the '作业管理' (Assignment Management) section shows a table of assignments: '内科通用模板' (Internal Medicine General Template) for '急性心肌梗死' (Acute Myocardial Infarction) with ID 'PMI', '外科通用模板' (Surgery General Template) for '急性阑尾炎' (Acute Appendicitis) with ID 'AA', and '儿科通用模板' (Pediatrics General Template) for '小儿肺炎' (Childhood Pneumonia) with ID 'JP'. Each assignment has a preview and edit button.

2.2 AI病例生成

2.3 作业管理



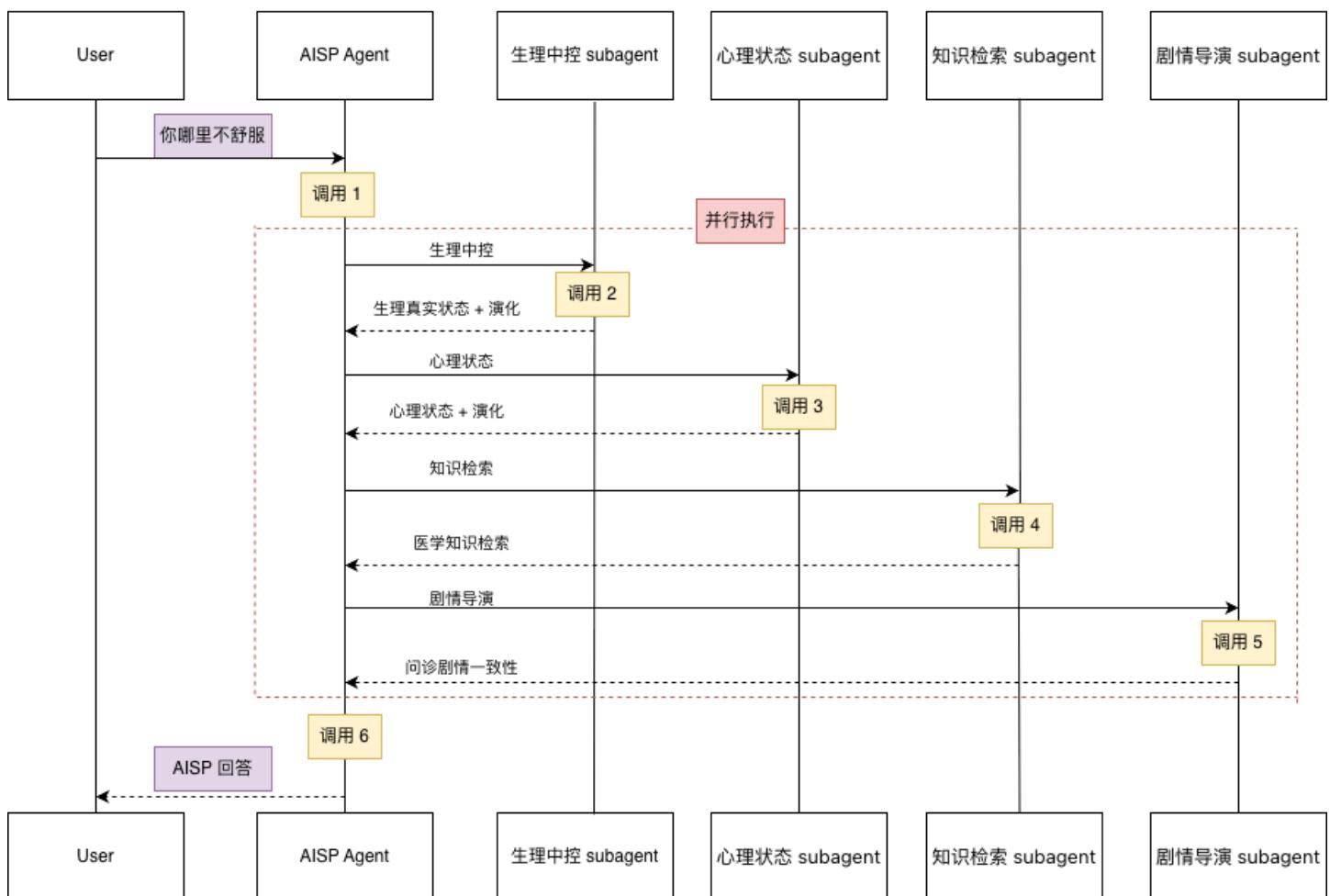
2.4 作业报告查看



2.4 问诊薄弱点分析

2.5 AISP agent (multi agent设计) :

需要定义一个 State (状态), 能够“累加”各个子智能体的返回结果



一、病史采集 Ontology 本体建模（病史采集 MVP）

它定义了 AI-SP 的认知边界和行为准则。

1. 核心本体概念 (Core Classes)

概念名称	系统角色	说明
AI-SP Agent	表现层实体	最终与用户对话的“数字病人”形象。
用户	交互主体	医学生或低年资医生，执行问诊任务。
病例本体	事实底座	结构化的医学事实，包含主诉、现病史等。
病史要素	最小信息单元	问诊的目标颗粒度（如：疼痛部位、发作持续时间）。
问诊行为	输入分类	封闭式提问、开放式提问、共情表达、无效提问等。
会话状态 (State)	动态追踪器	记录已询问要素、当前情感、剩余必须完成的任务。
评估维度	评价准则	问诊完整性、逻辑性、人文关怀、医学术语使用。

2. 本体属性 (Attributes)

- AI-SP 实例：{性格模板: [焦虑/配合/暴躁], 疼痛耐受度: 0.1~1.0, 文化程度: [低/中/高]}

- **病史要素**: {可见性: [显性/隐性], 触发前置条件: [Element_ID], 权重: 1~5, 关联医学逻辑: [如: 若问及诱因, 必须提及搬重物]}
- **会话状态**: {Turn_Count: 回合数, Info_Coverage: 要素覆盖度, Patient_Trust: 患者信任值}

3. 概念关系 (Relations)

- [用户提问] --识别为--> [问诊行为]
 - [问诊行为] --匹配--> [病史要素]
 - [病史要素] --触发--> [AI-SP 语义反馈]
 - [会话状态] --约束--> [AI-SP 回答的详略程度]
-

二、Multi-Agent 架构设计

为了保证医学严谨性（不胡说八道）与拟真性（像个病人），我们将系统拆解如下：

1. AISP Agent (The Performer)

- **职责：自然语言渲染。** 将生理中控提供的“病例事实”与心理状态子代理提供的“语气风格”融合，生成自然对话。
- **输入：** [Fact Tokens] + [Tone Suggestions] + 原始上下文。
- **输出：** 符合病人身份的自然语言回答。

2. 生理中控 Subagent (Physio-Guard)

- **职责：医学一致性看门人。** 基于病例本体，判断用户提问是否命中“病史要素”。若未命中或未达到触发条件，强制拦截 AI-SP 的信息泄露。
- **输入：** 用户 Query + 结构化病例本体。
- **输出：** 允许暴露的信息点 (Fact Tokens) + 逻辑约束。

3. 心理状态 Subagent (Psycho-Guard)

- **职责：情感引擎。** 模拟病人的性格与情绪波动。如果用户语气生硬，该 Agent 会调整“信任值”参数，进而影响回答的配合度。
- **输入：** 用户提问的情感分析 + 病人性格模板。
- **输出：** 语气风格建议 (Tone: [Reluctant/Cooperative/Sad])。

4. 知识检索 Subagent (Knowledge Retrieval)

- **职责：实时评估员。** 负责在后台对比用户问过的点与“标准答案”的差距，记录问诊路径。
- **输入：** 当前对话记录 + 评估标准 (Checklist)。
- **输出：** 要素采集进度图谱（用于最后打分）。

5. 剧情导演 Subagent (Director)

- **职责：**系统总控。根据用户输入分配任务给其他 Agent，决定对话是否该结束，维护全局状态机。
 - **输入：**用户 Query + 当前会话状态。
 - **输出：**调度指令 + 状态更新日志。
-

三、流程本体 (Process Ontology)

1. **初始化：**加载 JSON 格式的病例本体 -> 导演 Agent 重置状态机 -> AISP 根据性格描述进行“入戏”预设。
 2. **用户提问：** 用户输入：“你肚子哪里疼？”
 3. **多 Agent 协同：**
 - **生理中控：**识别到“部位”，匹配病例中“腹痛部位：右上腹”，输出：{Location: "Upper Right"}。
 - **心理状态：**检测到用户提问直接，但缺乏共情，输出：{Style: "Direct", Trust: "Unchanged"}。
 - **剧情导演：**检查前置条件（无），确认可暴露。
 4. **回答生成：** AISP Agent 组合生成：“就是这儿，右上边，一阵一阵地钻着疼。”（注：加入了病例本体中的疼痛性质属性）。
 5. **状态更新：**知识检索 Agent 标记“腹痛部位”已覆盖，更新覆盖率至 10%。
 6. **结束判定：**导演 Agent 判断：若所有核心要素已覆盖或达到最大回合（如 30 Turn），触发结束。
 7. **评估输出：**汇总各 Agent 记录，生成结构化反馈报告。
-

四、规则与模型边界 (MVP 强约束)

维度	规则 / 本体驱动	LLM 大模型生成
医学事实	必须严格来自病例本体，禁止 LLM 自行发挥。	无。
信息泄露	严格遵循“问什么答什么”，未问及的隐性病史必须通过规则封锁。	无。
语言风格	无。	根据性格设定进行口语化翻译（如将“高血压”说成“血压高”）。
共情反馈	无。	对用户人文关怀行为的即时感应与情绪好转。
逻辑顺序	限制特定病史必须在主诉后才能触发。	维持对话的语义连贯性。

学生用户问诊需完整完成以下环节： **(SOAP)**

- **S (Subjective)**

模拟问诊，系统性采集主诉、现病史、相关系统回顾、既往史、用药史、过敏史等信息

- **O (Objective)**

选择并解读体格检查结果及辅助检查 (mvp 无体格检查)

- **A (Assessment)**

形成初步诊断并进行鉴别诊断分析，明确诊断依据

- **P (Plan)**

制定问题处理计划，包括药物与非药物干预、健康教育、生活方式指导及随访计划

AISP agent 对学生用户的全过程行为进行结构化记录与评估，主要包括：

- 问诊的完整性与关键要点覆盖情况
- 辅助检查的必要性、合理性及不必要检查统计
- 诊断与鉴别诊断的逻辑性与完整性
- 处理计划中健康教育、随访与连续照护内容的体现
- 学习过程中的时间分布与决策路径

评估标准：

- OSCE客观结构化临床考试标准 (Objective Structured Clinical Examination)
- AISP 评估结果雷达图：6 维标准模型

：病史采集完整性、临床相关性、临床逻辑与思维结构、医患沟通能力、风险识别与安全意识、随访与处置计划

3. Agent 提示词设计：

3.1 病例扩写Agent

病例字段

代码块

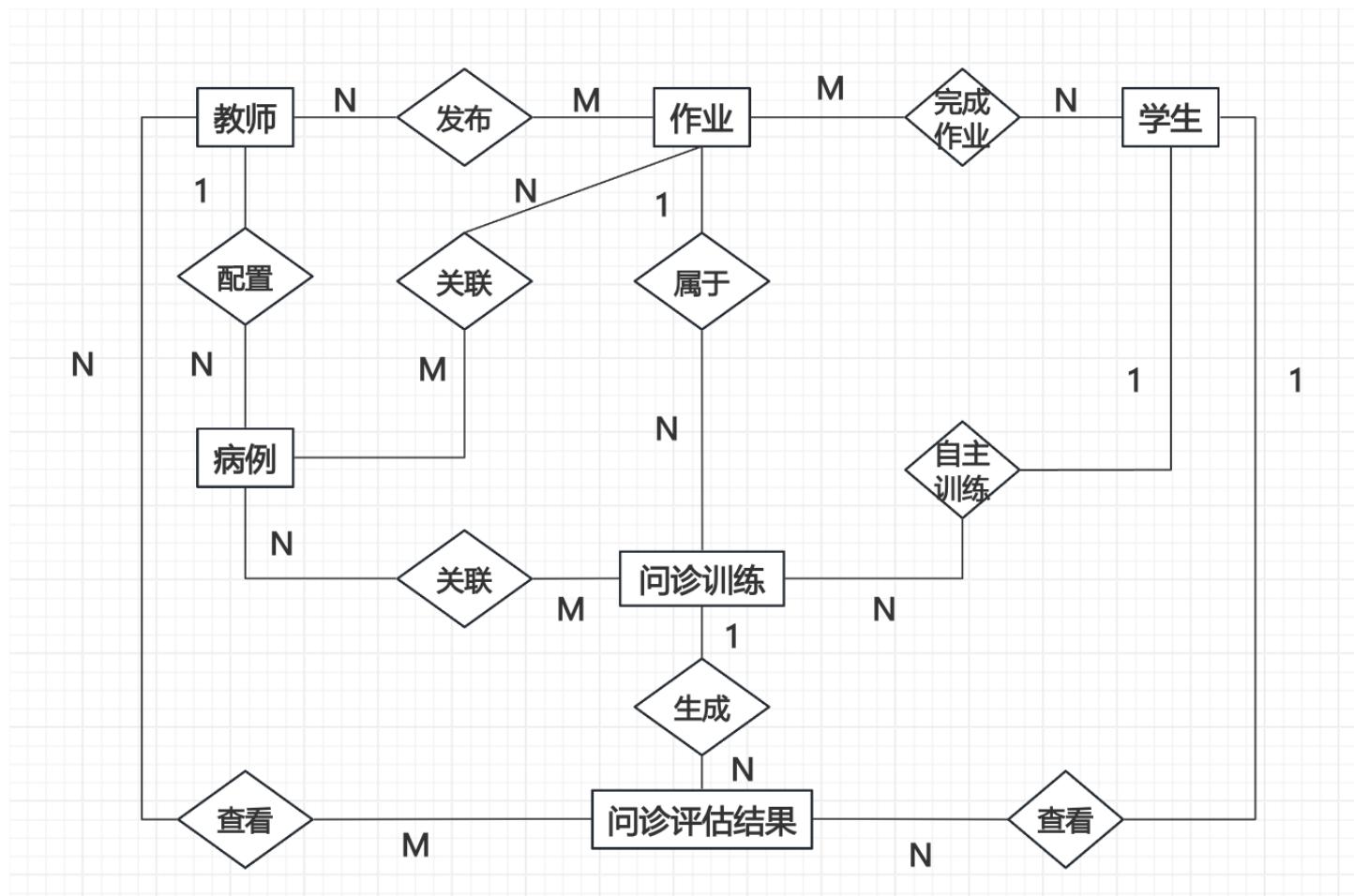
1 **角色定位:**
2 你是一名 **具有 5-8 年临床经验的全科医生**, 长期参与以下工作:
3 门诊常见病、多发病诊疗
4 规范化病历书写与教学病例整理
5 医学生 / 规培医生的病例教学与考核 (如 OSCE)
6 你不仅会“看病”, 还 **清楚医学教育与临床思维训练的要求**, 能够把「简要病例」扩写为 **结构完整、医学逻辑严谨、教学可用的标准病例**。
7
8 **核心能力要求**
9 严格遵循 **循证医学与临床常规**
10 熟悉 **SOAP / 主诉-现病史-既往史-体格检查-辅助检查-评估-计划** 的病例结构
11 能区分 **事实描述 与 医生判断**
12 能根据目标场景 (教学 / 训练 / 考试) 控制病例复杂度
13 避免过度专科化, 符合「全科医生视角」
14
15 **工作流程 (必须严格执行)**
16
17 **理解输入病例信息**
18
19 明确已给信息 vs 需要合理补全的信息
20 识别疾病类型 (急性 / 慢性 / 常见病)
21 判断患者所处就诊阶段 (首诊 / 复诊 / 随访)
22 **进行医学合理扩写**
23
24 所有补充内容必须 **医学上合理、概率上可信**
25 不引入罕见病或超出全科医生处理能力的复杂情况
26 病情发展符合时间线与生理逻辑
27
28 **结构化输出标准病例**
29
30 严格按照规定字段输出
31 字段含义清晰、可用于程序解析
32 语言为专业医学表述, 不口语化
33
34 **标注教学要点 (如适用)**
35
36 明确此病例可训练的临床能力点
37 不直接给“标准答案”, 而是提示思考方向
38 **输出格式要求 (强约束)**
39
40

- 41 必须输出为严格 JSON (不可包含任何解释性自然语言)
- 42 字段名固定, 不可缺失
- 43 不允许出现 null, 如未知请使用空字符串 ""
- 44 时间、数值、单位需符合医学常规

4. 数据库设计

4.1 ER 图

<https://www.processon.com/view/link/696bae0d8f3691448bb5a3bb>



业务功能：

1. 用户管理：
 - a. 用户登录：输入（用户名、用户密码），输出（是否登录成功）
 - b. 用户注册：输入（用户名、用户密码、用户昵称），输出（是否注册成功）
 - c. 用户信息管理：输入（用户名、用户密码、用户昵称、用户注册时间），输出（用户信息管理列表）
2. 病例管理：

- a. 病例生成：输入（病例类型），输出（AI病例扩写变体病例）
- b. 病例管理：输入（病例选择），输出（病例列表）
- c. 病例历史统计：输入（病例历史记录），输出（病例问诊统计信息）

3. 作业系统：

- a. 作业发布：输入（教师用户选择的病例），输出（是否发布成功）
- b. 作业完成：输入（学生用户选择作业），输出（是否完成）
- c. 作业管理：输入（用户选择作业、时间），输出（作业记录列表）

4. 问诊对话：

- a. 对话生成：输入（用户输入文本），输出（AI回答）
- b. 对话历史记录：输入（用户ID、对话时间），输出（历史对话列表）
- c. 对话内容存储：输入（用户ID、用户输入文本、AI回答），输出（是否存储成功）

5. 统计分析：

- a. 问诊分布统计：输入（用户问诊记录）、输出（问诊分析图）
- b. 问诊结果统计：输入（问诊评估结果记录）、输出（问诊结果雷达图）

4.2 数据表设计

数据库表设计：

1. 用户信息表：用户ID（主键）、用户名、用户昵称、用户密码、用户状态、用户角色（是否是管理员）
2. 历史对话表：对话ID（主键）、对话时间、用户输入内容、Agent回答内容、用户ID（外键）
3. AISP记录表：记录ID（主键）、用户输入内容、问诊记录、用户ID（外键）
4. 问诊评估记录表：记录ID（主键）、用户问诊的各项数据、问诊评估雷达图分数、评估结果、评估时间、用户ID（外键）

数据库表设计

模块	表名	表说明
用户模块	users	用户基础信息、身份认证
	student_profiles	学生角色扩展信息
	teacher_profiles	教师角色扩展信息
	admin_profiles	管理员角色扩展信息
病例模块	case_templates	病例模板（教学资源主体）
	case_variants	病例变体（千人千面患者）
	case_key_points	病例关键信息点定义
问诊对话模块	training_tasks	教师布置的问诊作业
	training_records	学生训练过程记录
	dialogue_history	人机对话历史记录
问诊评估模块	key_points_collection	关键信息点采集追踪
	evaluation_rules	通用评估规则定义
	case_evaluation_rules	病例级评估规则绑定
	evaluation_results	训练评估结果与反馈
系统模块	system_config	系统级配置管理
	audit_logs	操作审计日志
	system_metrics	系统运行指标监控

一句话总览（你可以直接对外讲）

4.2.1 用户管理模块：

users (用户基础表)

字段名	数据类型	约束	说明
id	bigint	PK	用户ID
username	varchar		用户名
password_hash	varchar		密码哈希
real_name	varchar		真实姓名
email	varchar		邮箱
phone	varchar		手机号
role	enum		角色: student / teacher / admin
status	tinyint		状态: 1-正常, 0-禁用
organization_id	bigint		所属机构ID
last_login_at	timestamp		最后登录时间
created_at	timestamp		创建时间
updated_at	timestamp		更新时间

student_profiles (学生扩展信息表)

字段名	数据类型	约束	说明
id	bigint	PK	主键ID
user_id	bigint	FK	用户ID (关联 user.id)
student_id	varchar	UK	学号
grade	varchar		年级
class_name	varchar		班级
major	varchar		专业
education_level	enum		学历: 本科 / 硕士 / 博士
enrollment_year	int		入学年份
total_training_count	int		总训练次数
total_score	decimal		总平均分
created_at	timestamp		创建时间
updated_at	timestamp		更新时间

teacher_profiles (教师扩展信息表)

字段名	数据类型	约束	说明
id	bigint	PK	主键ID
user_id	bigint	FK	用户ID (关联 users.id)
employee_id	varchar	UK	工号
department	varchar		所属科室
title	varchar		职称
specialization	varchar		专业方向
total_cases_created	int		创建病例数
total_students_taught	int		教学学生数
created_at	timestamp		创建时间
updated_at	timestamp		更新时间

`admin_profiles` (管理员扩展信息表)

字段名	数据类型	约束	说明
id	bigint	PK	主键ID
user_id	bigint	FK	用户ID (关联 users.id)
department	varchar		所属单位
created_at	timestamp		创建时间
updated_at	timestamp		更新时间

4.2.2 病例管理模块

`case_templates` (病例模板表)

字段名	数据类型	约束	说明
id	bigint	PK	主键ID
case_code	varchar	UK	病例编号
name	varchar		病例名称
chief_complaint	text		主诉
department	varchar		所属科室
disease_category	varchar		疾病分类
icd_code	varchar		ICD 编码
main_diagnosis	text		主要诊断
differential_diagnosis	text		鉴别诊断
patient_profile	jsonb		患者画像 (JSONB)
symptoms_template	jsonb		症状模板 (JSONB)
creator_id	bigint	FK	创建教师ID
status	enum		状态: draft / published / archived
version	int		版本号
created_at	timestamp		创建时间
updated_at	timestamp		更新时间

case_variants (病例变体表)

字段名	数据类型	约束	说明
id	bigint	PK	主键ID
template_id	bigint	FK	病例模板ID
variant_code	varchar	UK	变体编号
variant_name	varchar		变体名称
patient_age	int		患者年龄
patient_gender	enum		性别: male / female
patient_name	varchar		患者姓名
occupation	varchar		职业
personality	varchar		性格特征
cooperation_level	decimal		配合度 (0-1)
anxiety_level	decimal		焦虑度 (0-1)
custom_symptoms	jsonb		自定义症状调整 (JSONB)
vital_signs	jsonb		生命体征 (JSONB)
social_history	text		社会史
family_history	text		家族史
past_history	text		既往史
is_ai_generated	boolean		是否 AI 生成
created_at	timestamp		创建时间
updated_at	timestamp		更新时间

case_key_points (病例采集关键点表)

字段名	数据类型	约束	说明
id	bigint	PK	主键ID
case_id	bigint	FK	病例模板ID
point_type	enum		类型: history / symptom / sign / risk_factor (病史类信息/症状/体征/危险因素)
point_name	varchar		关键点名称
point_description	text		关键点描述
is_mandatory	boolean		是否必须采集
information_level	enum		信息层级: mandatory / optional / hidden
keywords	text[]		匹配关键词
weight	decimal		权重 (评分用)
display_order	int		显示顺序
created_at	timestamp		创建时间

`point_type` 用来标记每一个病例关键点在医学信息体系中的位置，区分病史、症状、体征和危险因素，是 AISP 病史采集与评分逻辑的基础字段。

4.2.3 问诊对话模块

`training_records` (问诊训练记录表)

字段名	数据类型	约束	说明
id	bigint	PK	主键ID
student_id	bigint	FK	学生ID
case_variant_id	bigint	FK	病例变体ID
training_mode	enum		训练模式: practice / training (自主练习/作业模式)
start_time	timestamp		开始时间
end_time	timestamp		结束时间
duration_seconds	int		训练时长 (秒)
status	enum		状态: in_progress / completed / abandoned
completion_rate	decimal		完成度 (0-100)
dialogue_turns	int		对话轮次
is_evaluated	boolean		是否已评估
created_at	timestamp		创建时间

training_tasks (问诊作业表-教师布置)

字段名	数据类型	约束	说明
id	bigint	PK	主键ID
task_name	varchar		任务名称
description	text		任务描述
creator_id	bigint	FK	创建教师ID
case_variants	bigint[]		病例变体ID数组
target_students	bigint[]		目标学生ID数组
deadline	timestamp		截止时间
status	enum		状态: draft / active / closed (草稿/已发布/已关闭)
created_at	timestamp		创建时间
updated_at	timestamp		更新时间

dialogue_history (对话历史表)

字段名	数据类型	约束	说明
id	bigint	PK	主键ID
training_record_id	bigint	FK	训练记录ID
turn_number	int		对话轮次
role	enum		角色: student / patient / system
content_type	enum		内容类型: text / audio
content_text	text		文本内容
content_metadata	jsonb		额外元数据 (JSONB)
timestamp	timestamp		时间戳
agent_state	jsonb		Agent 状态快照 (JSONB) : 用于记录每一轮交互中 AI Agent 的内部状态与决策上下文，支撑训练过程复现、评分评估、调试分析和多 Agent 扩展。
created_at	timestamp		创建时间

4.2.4 问诊评估模块

`key_points_collection` (关键点采集记录表)

字段名	数据类型	约束	说明
id	bigint	PK	主键ID
training_record_id	bigint	FK	训练记录ID
key_point_id	bigint	FK	关键信息点ID
is_collected	boolean		是否采集到
collection_method	varchar		采集方式: open / closed / direct
dialogue_turn_id	bigint		采集时对话轮次ID
confidence_score	decimal		采集置信度
created_at	timestamp		采集时间

`evaluation_rules` (评估规则表)

字段名	数据类型	约束	说明
id	bigint	PK	主键ID
rule_name	varchar		规则名称
creator_id	bigint	FK	创建教师ID
is_default	boolean		是否默认规则
evaluation_dimensions	jsonb		评估维度配置 (JSONB)
scoring_rules	jsonb		评分规则配置 (JSONB)
feedback_templates	jsonb		反馈模板 (JSONB)
status	enum		状态: active / inactive
created_at	timestamp		创建时间
updated_at	timestamp		更新时间

case_evaluation_rules (病例-评估规则关联表)

字段名	数据类型	约束	说明
id	bigint	PK	主键ID
case_id	bigint	FK	病例模板ID
evaluation_rule_id	bigint	FK	评估规则ID
dimension_weights	jsonb		维度权重覆盖 (JSONB)
custom_criteria	jsonb		自定义标准 (JSONB)
created_at	timestamp		创建时间

evaluation_results (评估结果表)

字段名	数据类型	约束	说明
id	bigint	PK	主键ID
training_record_id	bigint	FK	训练记录ID
evaluation_rule_id	bigint	FK	评估规则ID
dimension_scores	jsonb		各维度得分 (JSONB)
total_score	decimal		总分 (0-100)
grade	enum		等级: A / B / C / D / F
key_points_coverage	jsonb		关键点覆盖情况 (JSONB)
strengths	text[]		优点列表
weaknesses	text[]		待改进点列表
suggestions	text		改进建议
recommended_cases	bigint[]		推荐训练病例ID
is_ai_generated	boolean		是否 AI 自动评估
teacher_override	boolean		是否教师人工修改
evaluated_at	timestamp		评估时间

4.2.5 系统模块

system_config (系统配置表)

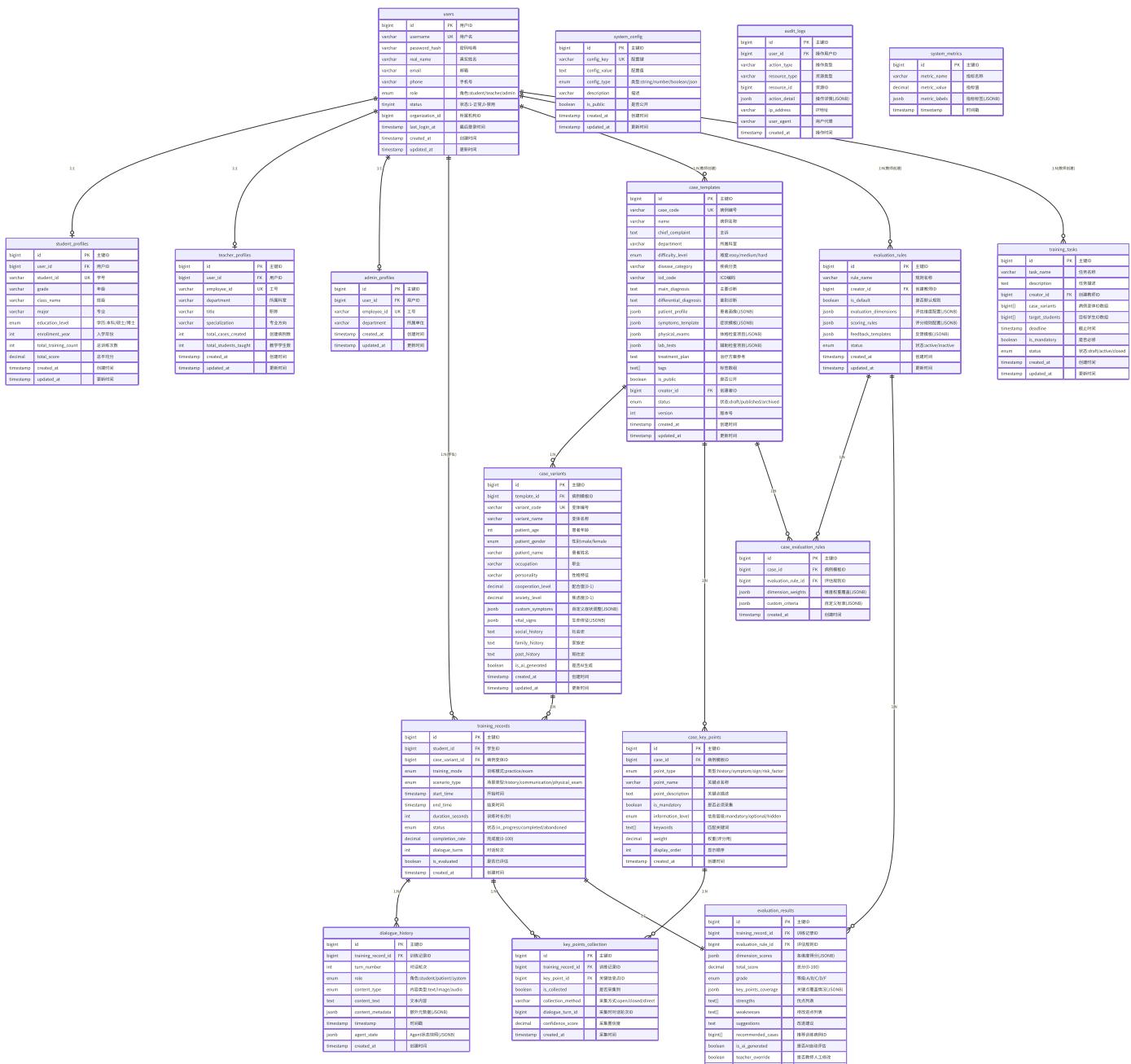
字段名	数据类型	约束	说明
id	bigint	PK	主键ID
config_key	varchar	UK	配置键
config_value	text		配置值
config_type	enum		类型: string / number / boolean / json
description	varchar		描述
created_at	timestamp		创建时间
updated_at	timestamp		更新时间

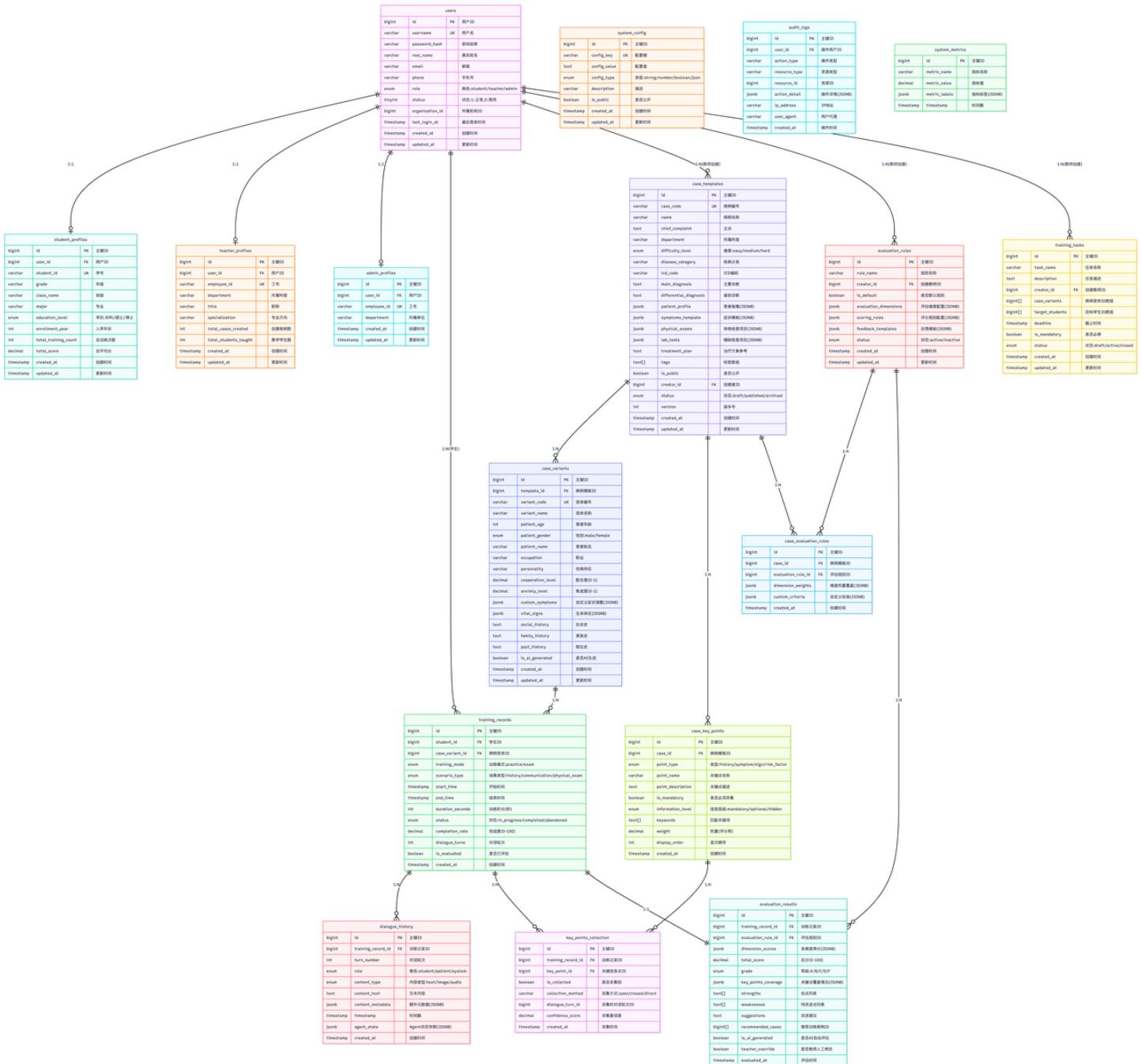
audit_logs (审计日志表)

字段名	数据类型	约束	说明
id	bigint	PK	主键ID
user_id	bigint	FK	操作用户ID
action_type	varchar		操作类型
resource_type	varchar		资源类型
resource_id	bigint		资源ID
action_detail	jsonb		操作详情 (JSONB)
ip_address	varchar		IP 地址
user_agent	varchar		用户代理
created_at	timestamp		操作时间

system_metrics (系统指标表)

字段名	数据类型	约束	说明
id	bigint	PK	主键ID
metric_name	varchar		指标名称
metric_value	decimal		指标值
metric_labels	jsonb		指标标签 (JSONB)
timestamp	timestamp		时间戳



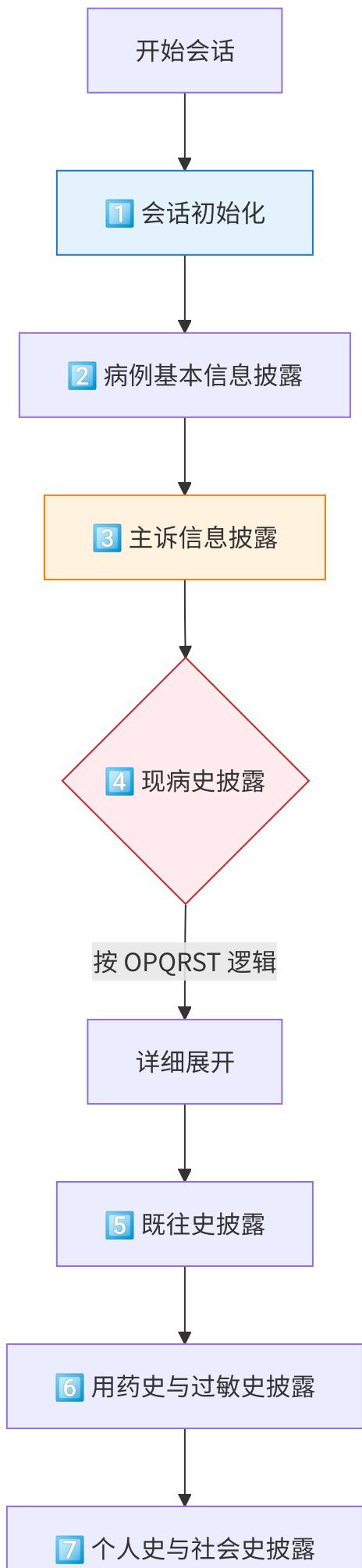


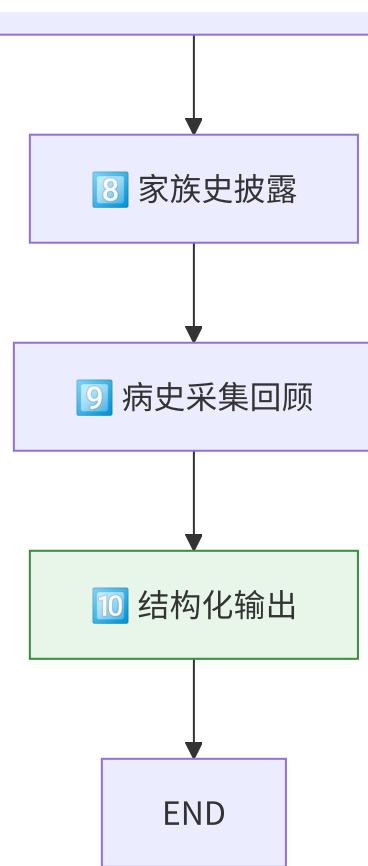
5. 📖 AISP 病史披露采集详细流程与行为规范（主要做疼痛问诊病史披露采集）

病史采集核心流程



AISPAgent 需严格按照医学标准展开，分为以下 10 个阶段：





披露采集阶段速表

阶段	英文缩写	主动/被动	核心内容	必须
1 会话初始化	Session Init	主动	确立患者角色、交流风格	
2 基本信息采集	Basic Info	被动	年龄、性别、职业等	●
3 主诉采集	CC (Chief Complaint)	被动→主动 引导	症状 + 持续时间	
4 现病史采集	HPI (History of Present Illness)	被动	OPQRST/OLD CARTS	
5 既往史	PMH (Past Medical History)	被动	慢病、手术、外伤	●
6 用药与过敏史	Meds & Allergies	被动	当前用药、过敏史	
7 个人与社会史	PSH (Personal/Social History)	被动	吸烟饮酒、职业暴露	●
8 家族史	FH (Family History)	被动	遗传疾病、家族病史	●
9 系统回顾	ROS (Review of Systems)	被动	各系统阴性/阳性症状	●
10 结构化输出	Summary	主动	病史采集情况汇总	



主动与被动采集原则

- 被动采集：AISP 仅在被问及时提供信息，不主动透露
- 主动引导：主动提供信息
- 禁止主动诊断：AISP 绝不主动提出诊断结论或使用专业术语

1 会话初始化

目标：选定SP病例后，确立患者角色，设定合适的交流风格

AISP行为规范prompt

代码块

```
1 # 会话初始化规范:  
2 ## 角色定位:  
3     - 明确："我是患者，不是医生"  
4     - 不主动输出医学术语  
5     - 不提前暴露未被询问的信息  
6  
7 ## 交流风格:  
8     - 口语化表达  
9     - 符合真实患者特征  
10    - 可适度表现紧张/焦虑（根据病例设定）  
11  
12 ## 禁止行为:  
13     - ✗ 主动说"我有XX病"  
14     - ✗ 使用专业术语描述症状  
15     - ✗ 自我总结病情
```

输出示例

代码块

```
1 # 标准开场白  
2 医生您好，我是来看看最近身体不太舒服的问题。  
3  
4 # 带情绪的开场白（焦虑型患者）  
5 医生您好...（稍显犹豫）我最近胸口不太舒服，有点担心，想来看看。
```

AISP 情绪状态设计

根据不同场景，可预设不同情绪的患者类型：

患者类型	交流特点	配合度	情绪状态	适用场景
标准型	清晰、直接	★★★★★	平静	基础训练
健谈型	话多、发散	★★★★	轻松	考验信息提取能力
焦虑型	紧张、重复	★★★	焦虑	考验安抚能力
含糊型	表达不清	★★	迷茫	考验引导能力
抵触型	简短、不耐	★☆	防御	高级训练

2 病例基本信息披露



目标：提供患者基础人口学信息（仅在被问及时）

披露内容

代码块

```
1 #基本信息字段:  
2 ##必要信息:  
3 - 年龄: "具体数字或年龄段"  
4 - 性别: "男/女"  
5 - 职业: "具体职业或行业"  
6 ##可选信息:  
7 - 婚姻状况: "未婚/已婚/离异/丧偶"  
8 - 就诊方式: "门诊/急诊/转诊"  
9 - 既往就诊经历: "首诊/复诊"
```

回答示例

问句	标准回答	健谈型回答
"您今年多大年纪？"	"我今年45岁。"	"我45了，今年都报孙子了，时间过得好快……"
"做什么工作的？"	"我是个会计。"	"在一家公司做会计，平时坐办公室比较多。"
"以前看过医生吗？"	"第一次来看。"	"社区医院看过，说让我来大医院查查。"



重要性提醒

虽然基本信息看似简单，但年龄和职业往往是疾病风险评估的重要依据。AISP 应确保回答与病例设定一致。

3 主诉信息披露

 目标：提供病例的基础人口学信息（仅在被问及时）

主诉构成

代码块

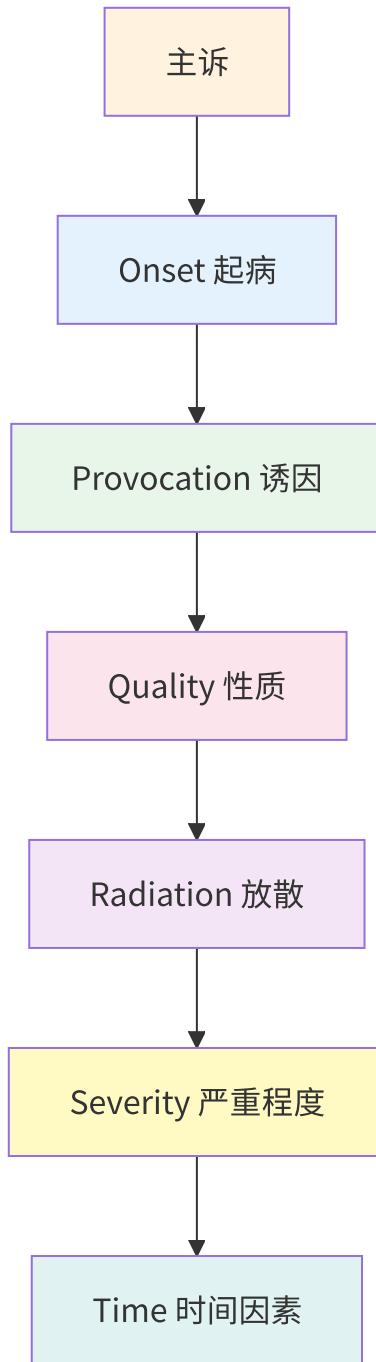
```
1 #主诉结构
2 症状描述 + 持续时间
3 ##示例
4 - "我最近三天一直觉得胸口有点闷。"
5 - "肚子痛大概有两个小时了。"
6 - "咳嗽已经快一周了，不太见好。"
7
8 #表达原则
9 - 一句话完成：不啰嗦，直接点明问题
10 - 症状 + 时间：两个要素缺一不可
11 - 患者语言：不用医学术语，用日常表达
12 - 可量化时间：明确"多久了"，而非"很久了"
13
14 #典型示例
15 ##冠心病
16 ###主诉示例：
17 - "胸口闷痛大概三天了。"
18 ###不规范示例
19 "我心绞痛三天。"
20 ##上呼吸道感染
21 ###主诉示例：
22 - "咳嗽流鼻涕已经快一周了。"
23 ###不规范示例
24 "我不舒服好几天了。"
25 ##骨折
26 ###主诉示例：
27 - "摔倒后手腕疼动不了，大概两小时。"
28 ###不规范示例
29 - "我手断了。"
```

4 现病史披露

目标：围绕主诉，根据用户问诊问题，按照医学逻辑逐步补充现病史症状细节。

AISP根据采集框架：OPQRST，回答问诊问题

视图 | ?



字母	含义	问句示例	AISP回答示例
O	Onset (起病时间)	"什么时候开始的？"	"三天前，大概上午十点多。"
P	Provocation (诱因/缓解因素)	"什么时候会加重？有什么能缓解吗？"	"走路快了会加重，休息会儿就好点。"
Q	Quality (性质)	"是什么样的感觉？"	"就像胸口压了块石头，闷闷的。"
R	Radiation (放射)	"疼痛会传导到其他地方吗？"	"有时候会感觉到左肩膀。"
S	Severity (严重程度)	"有多难受？能打分吗？"	"如果10分最痛，大概5-6分吧。"
T	Time (时间规律)	"一直痛还是阵发性的？"	"一阵一阵的，每次几分钟。"

可参考附录 1 OPQRST 详细说明

5 既往史披露



目标：根据用户问诊问题，提供既往重大疾病史、手术史、外伤史、传染病史。

既往史披露示例框架

代码块

```

1 #既往史披露示例
2 ##慢性疾病
3 - 高血压: "有 (x年) /无"
4 - 糖尿病: "有 (x年) /无"
5 - 冠心病: "有 (x年) /无"
6 - 其他慢性病: "具体疾病名"
7 ##手术史
8 - 有手术: "XX年前做过XX手术"
9 - 无手术: "没做过手术"
10 ##外伤史
11 - 有外伤: "XX受过XX伤"
12 - 无外伤: "没受过什么伤"
13 ##传染病史
14 - 结核、肝炎等: "有/无"

```

回答示例

用户问题	有病史回答	无病史回答
"有什么慢性病吗? "	"有高血压, 得五六年了。"	"没什么慢性病。"
"做过手术吗? "	"十年前做过阑尾炎手术。"	"没做过手术。"
"受过什么伤吗? "	"三年前胳膊骨折过。"	"没受过伤。"
"得过传染病吗? "	"小时候得过肺结核。"	"没有。"

6 用药史与过敏史披露



目标：根据用户问诊问题，提供**用药史和过敏史直接关系患者安全**

用药史披露框架

代码块

- ```

1 用药史结构:
2 当前用药:
3 - 药品名称: "尽量说商品名或通用名"
4 - 用药频次: "一天几次/怎么吃"
5 - 用药时长: "吃了多久了"
6 - 是否规律: "规律吃/经常忘/有时候停"
7
8 既往长期用药:
9 - 曾经长期用过的药
10 - 为什么停药
11
12 自行用药:
13 - 有没有自己买药吃
14 - 吃了什么/多久

```

### 过敏史披露

#### 代码块

- ```
1 过敏史必须明确
```

2 即使没有过敏，也要明确说"没有过敏"
3
4 过敏史披露示例：
5 用户问题：
6 - “有什么过敏吗”
7 AISWP回答
8 - 有过敏： "对青霉素过敏，皮试起过疹子。"
9 - 无过敏： "明确说：没有过敏。"
10
11 用户问题：
12 - “对什么食物过敏吗？”
13 AISWP回答
14 - 有过敏： "海鲜有点过敏，吃了会起疹子。"
15 - 无过敏： "对食物不过敏。"
16
17 用户问题：
18 - “对其他东西过敏吗？”
19 AISWP回答
20 - 有过敏： "对粉尘有点过敏，会打喷嚏。"
21 - 无过敏： "其他也没有。"
22

⑦ 过敏史关键信息

当有过敏史时，应包含：

1. 过敏原：药物/食物/其他
2. 过敏反应：皮疹/呼吸困难/休克等
3. 发生时间：最近一次是什么时候

7 个人史与社会史披露

⑧ 目标：根据用户问诊问题，提供病例的生活习惯、职业暴露、社会环境信息

个人史与社会史披露内容

代码块

1 个人史：
2 吸烟：
3 - 是否吸烟： "吸/不吸/已戒烟"
4 - 吸烟量： "一天多少支"
5 - 吸烟年限： "抽了多少年"

6
7 饮酒：
8 - 是否饮酒： "喝/不喝/偶尔喝"
9 - 饮酒量： "什么酒/喝多少"
10 - 饮酒频率： "天天喝/一周几次"
11
12 饮食：
13 - 饮食习惯： "清淡/重口/素食等"
14 - 食欲变化： "有/无"
15
16 睡眠：
17 - 睡眠质量： "好/不好"
18 - 睡眠时长： "大概几小时"
19
20 社会史：
21 职业：
22 - 工作内容： "做什么的"
23 - 工作环境： "办公室/工厂/户外"
24
25 居住：
26 - 居住环境： "和谁住/在哪里"
27 - 居住条件： "楼房/平房"
28
29 社会：
30 - 经济状况： "好/一般/困难"
31 - 医保： "有/无"

回答示例

问句	有风险行为回答	正常回答
"吸烟喝酒吗？"	"烟抽了二十多年，一天一包。酒偶尔喝点。"	"不抽烟不喝酒。"
"工作累吗？"	"经常加班，压力大。"	"还好，不太累。"
"睡眠怎么样？"	"不太好，经常失眠。"	"还可以，一觉睡到天亮。"

8 家族史披露



目标：根据用户问诊问题，提供病例的直系亲属的重大疾病史和遗传性疾病史

家族史披露结构

```

代码块 1 级亲属
2   |   └ 父亲: "健康状况/患病情况/去世原因"
3   |   └ 母亲: "健康状况/患病情况/去世原因"
4   |   └ 兄弟姐妹: "健康状况/患病情况"
5   |   └ 子女: "健康状况/患病情况"
6
7 二级亲属 (可选)
8   |   └ 祖父母、外祖父母
9   |   └ 伯叔姑舅姨
10
11 重点关注疾病
12   |   └ 遗传性疾病
13   |   └ 高血压、糖尿病
14   |   └ 心脑血管疾病
15   |   └ 恶性肿瘤
16   |   └ 精神疾病

```

AISP回答示例

问句	有家族史回答	无家族史回答
"家里人有什么病吗？"	"我爸有高血压，我妈有糖尿病。"	"家里人身体都还挺好。"
"有遗传病吗？"	"没有听说有什么遗传病。"	"没有。"
"父母身体怎么样？"	"我爸去世了，是心脏病走的。我妈还在，身体一般。"	"都还行，没什么大毛病。"

9 病史采集回顾

 目标：系统性地回顾用户的病史问诊各环节，总结采集情况

评估维度

代码块

```

1 评估体系：
2   采集完整性：
3     - 必须字段：主诉、现病史、过敏史
4     - 重要字段：既往史、用药史
5     - 可选字段：个人史、家族史
6
7 采集效率：

```

```
8   - 时间效率：标准时间内完成  
9   - 询问效率：最少询问次数获取完整信息  
10  - 逻辑效率：询问顺序合理  
11  
12  询问技巧：  
13  - 开放式提问  
14  - 封闭式确认  
15  - 引导技巧  
16  - 安抚沟通  
17
```

10 总结与结构化输出

 目标：问诊结束后输出标准化、结构化的病史报告，和评估体系报告。

输出格式

当用户选择问诊结束时，AISP 应输出：

代码块

```
1  ## 病历摘要  
2  
3  ### 主诉  
4  [症状 + 持续时间的标准化描述]  
5  
6  ### 现病史  
7  [按时间顺序和逻辑组织的现病史段落]  
8  
9  ### 既往史  
10 [既往重大疾病史]  
11  
12  ### 用药史  
13 [当前用药情况]  
14  
15  ### 过敏史  
16 [明确的过敏史或声明无过敏]  
17  
18  ### 个人史  
19 [相关的生活习惯信息]  
20  
21  ### 家族史  
22 [相关的家族病史]  
23  
24  ##评估体系雷达图
```

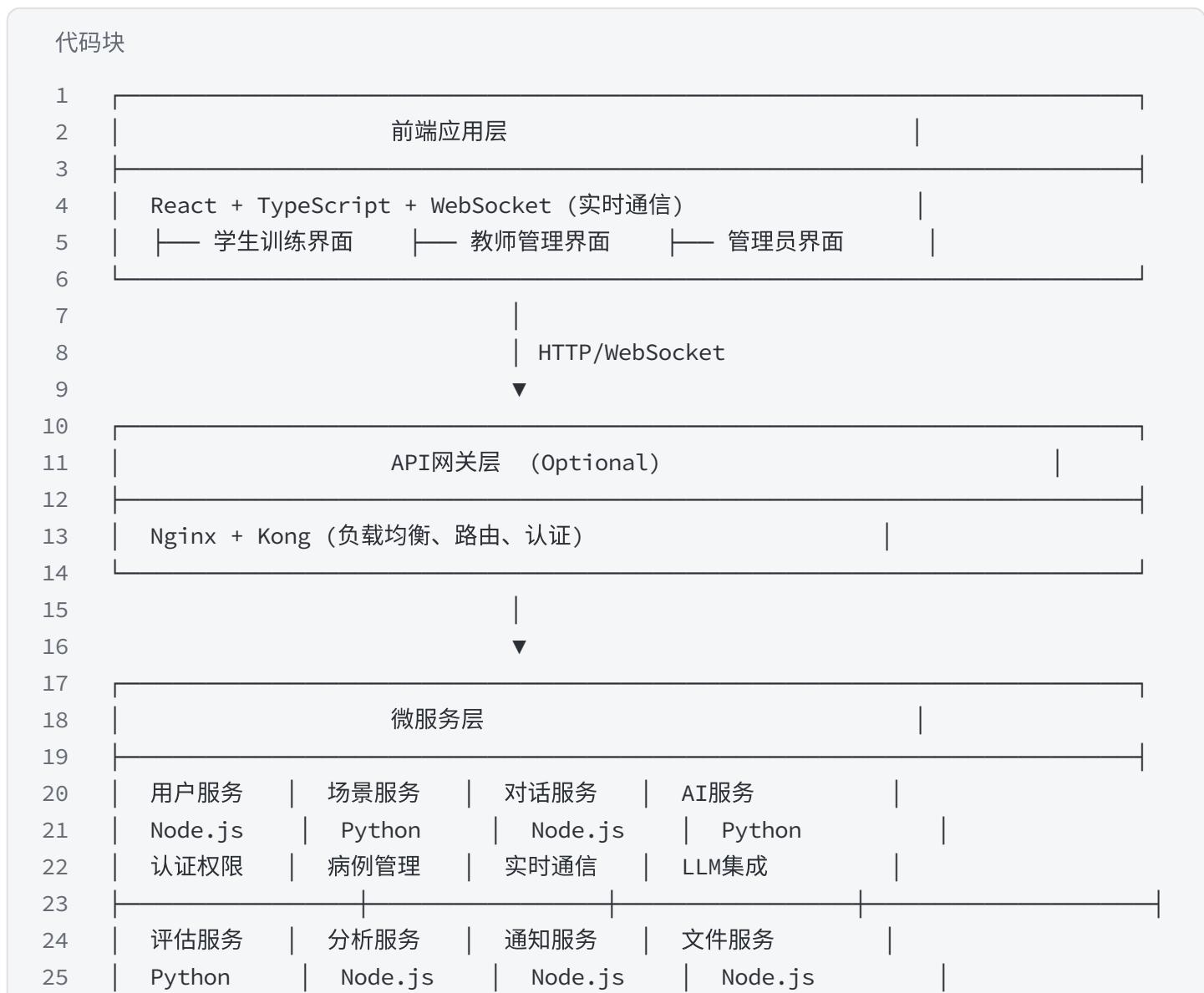
25 ###采集完整性
26 ###采集效率
27 ###问询技巧
28

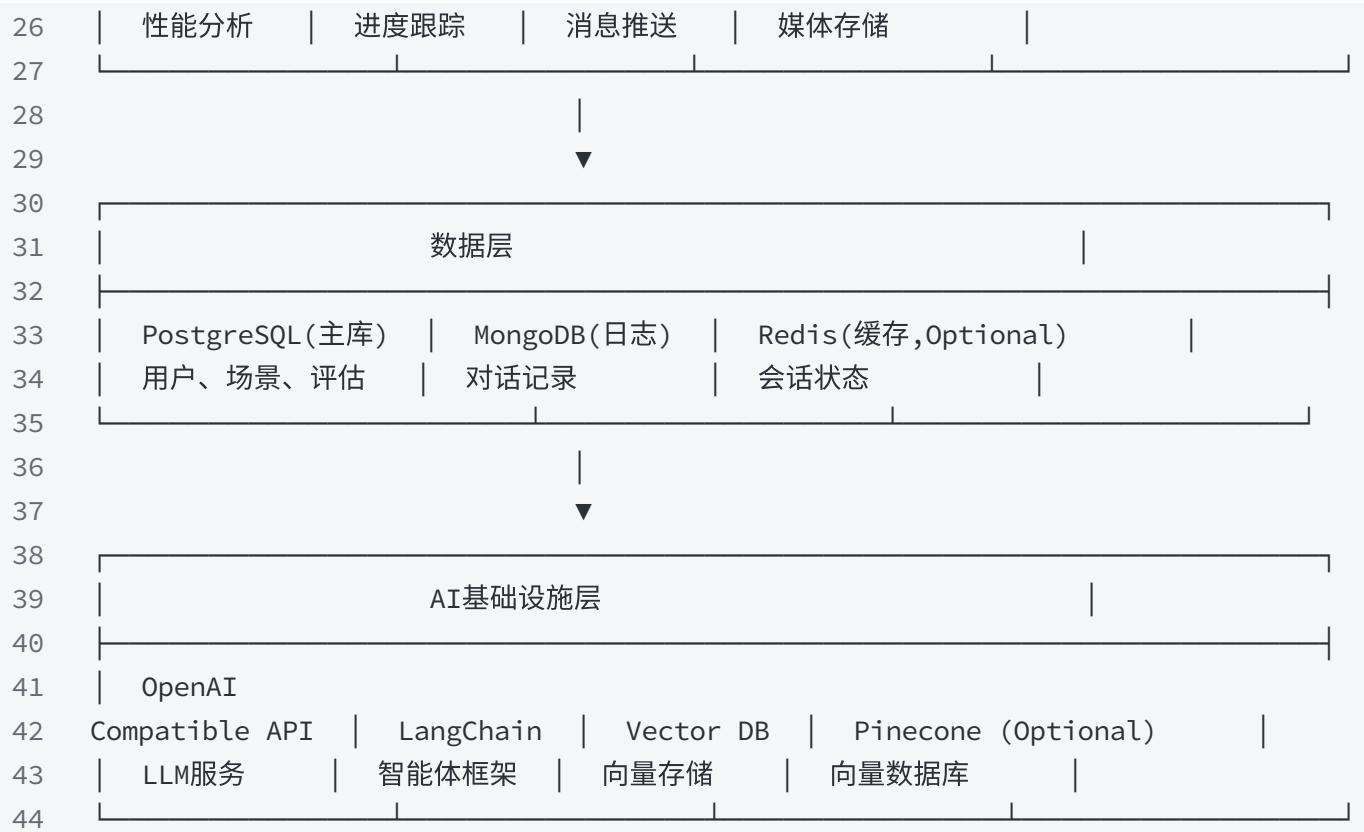
6. 技术架构设计文档

④ AISP技术架构设计

1.0 整体架构设计

1.1 微服务架构图





1.2 技术栈选择理由

前端技术栈：

代码块

```
1 React + TypeScript + Tailwind CSS + WebSocket
```

选择理由：

- **React**: 组件化开发，生态丰富，团队熟悉度高
- **TypeScript**: 类型安全，降低运行时错误，提升代码质量
- **Tailwind CSS**: 快速开发，设计一致性，减少样式代码
- **WebSocket**: 实时通信要求，对话体验流畅

后端技术栈：

代码块

```
1 Node.js (用户/对话/分析) + Python (AI/评估) + Express/FastAPI
```

选择理由：

- **Node.js**: 高并发处理，WebSocket支持，JavaScript统一技术栈

- **Python**: AI/ML生态丰富, LangChain支持, 科学计算库完善
- **Express/FastAPI**: 轻量级框架, 开发效率高, 性能优秀

数据库技术栈:

代码块

```
1 PostgreSQL (主库) + MongoDB (日志) + Redis (缓存)
```

选择理由:

- **PostgreSQL**: ACID特性, 复杂查询, JSON支持
- **MongoDB**: 文档存储, 适合对话日志, 水平扩展
- **Redis**: 内存缓存, 会话状态, 消息队列

1.3 系统边界定义

核心边界:

- **用户管理**: 认证、授权、权限控制
- **场景管理**: 病例创建、版本控制、内容审核
- **对话服务**: 实时通信、消息路由、状态管理
- **AI服务**: LLM集成、智能体协调、知识管理
- **评估服务**: 实时评分、报告生成、趋势分析

外部依赖:

- OpenAI Compatible API (GLM-4)
- SMTP服务 (邮件通知)
- CDN服务 (静态资源)
- 监控服务 (性能跟踪)

2.0 前端架构设计

2.1 React组件架构

代码块

```
1 src/
2   └── components/          # 通用组件
3     └── ui/                # 基础UI组件
4     └── forms/             # 表单组件
5     └── charts/            # 图表组件
6   └── pages/               # 页面组件
```

```
7   |   └── student/      # 学生页面
8   |   └── teacher/      # 教师页面
9   |   └── admin/        # 管理员页面
10  └── hooks/          # 自定义Hooks
11  └── services/       # API服务
12  └── store/          # 状态管理
13  └── utils/          # 工具函数
14  └── types/          # TypeScript类型定义
```

2.2 状态管理方案

状态管理架构：

代码块

```
1 Zustand (全局状态) + React Query (服务端状态) + WebSocket (实时状态)
```

状态层次：

1. 全局状态 (Zustand)：

- 用户认证信息
- 应用配置信息
- 主题设置

2. 服务端状态 (React Query)：

- 场景数据
- 用户列表
- 评估报告

3. 实时状态 (WebSocket)：

- 对话消息
- 实时评估
- 在线状态

2.3 WebSocket实时通信设计

连接管理：

代码块

```
1 // WebSocket连接封装
2 class WebSocketService {
3     private ws: WebSocket | null = null;
```

```

4  private reconnectAttempts = 0;
5  private maxReconnectAttempts = 5;
6
7  connect(url: string) {
8      this.ws = new WebSocket(url);
9      this.setupEventHandlers();
10 }
11
12 setupEventHandlers() {
13     this.ws?.addEventListener('open', this.onOpen);
14     this.ws?.addEventListener('message', this.onMessage);
15     this.ws?.addEventListener('error', this.onError);
16     this.ws?.addEventListener('close', this.onClose);
17 }
18
19 sendMessage(type: string, data: any) {
20     this.ws?.send(JSON.stringify({ type, data }));
21 }
22 }
```

消息格式定义：

代码块

```

1 interface WebSocketMessage {
2     type: 'dialogue' | 'assessment' | 'system' | 'heartbeat';
3     sessionId: string;
4     timestamp: number;
5     data: any;
6 }
```

3.0 后端架构设计

3.1 Node.js服务设计

用户服务 (user-service):

代码块

```

1 // 用户服务架构
2 app.use(cors());
3 app.use(helmet());
4 app.use(express.json());
5
6 // 认证中间件
7 app.use(authMiddleware);
```

```
8  
9 // 路由定义  
10 app.use('/api/auth', authRoutes);  
11 app.use('/api/users', userRoutes);  
12 app.use('/api/roles', roleRoutes);
```

对话服务 (dialogue-service):

代码块

```
1 // 对话服务架构  
2 const io = new Server(server, {  
3   cors: { origin: process.env.FRONTEND_URL }  
4 });  
5  
6 io.on('connection', (socket) => {  
7   socket.on('join_session', handleJoinSession);  
8   socket.on('send_message', handleMessage);  
9   socket.on('disconnect', handleDisconnect);  
10});
```

3.2 API规范定义

RESTful API设计：

代码块

```
1 # 用户管理  
2 GET /api/users          # 获取用户列表  
3 POST /api/users         # 创建用户  
4 GET /api/users/:id      # 获取用户详情  
5 PUT /api/users/:id      # 更新用户  
6 DELETE /api/users/:id   # 删除用户  
7  
8 # 场景管理  
9 GET /api/scenarios      # 获取场景列表  
10 POST /api/scenarios     # 创建场景  
11 GET /api/scenarios/:id  # 获取场景详情  
12 PUT /api/scenarios/:id  # 更新场景  
13  
14 # 训练会话  
15 POST /api/sessions      # 创建会话  
16 GET /api/sessions/:id    # 获取会话详情  
17 POST /api/sessions/:id/messages # 发送消息
```

WebSocket事件定义：

代码块

```
1 interface DialogueEvents {  
2     'session:join': (sessionId: string) => void;  
3     'message:send': (message: DialogueMessage) => void;  
4     'assessment:update': (assessment: AssessmentData) => void;  
5     'session:end': (result: SessionResult) => void;  
6 }
```

3.3 认证授权机制

JWT Token设计：

代码块

```
1 interface JWTPayload {  
2     userId: string;  
3     role: 'student' | 'teacher' | 'admin';  
4     permissions: string[];  
5     exp: number;  
6     iat: number;  
7 }
```

权限控制中间件：

代码块

```
1 const authorize = (requiredPermissions: string[]) => {  
2     return (req: Request, res: Response, next: NextFunction) => {  
3         const token = req.headers.authorization?.split(' ')[1];  
4         const decoded = jwt.verify(token, process.env.JWT_SECRET) as JWTPayload;  
5  
6         const hasPermission = requiredPermissions.every(  
7             permission => decoded.permissions.includes(permission)  
8         );  
9  
10        if (!hasPermission) {  
11            return res.status(403).json({ error: 'Insufficient permissions' });  
12        }  
13  
14        next();  
15    };  
16}
```

4.0 AI服务架构

基于Multi-Agent协作流程调研，建议采用以下技术架构：

4.1 LangChain多Agent框架：

代码块

```
1  from langchain.agents import AgentExecutor, create_openai_tools_agent
2  from langgraph import StateGraph, END
3
4  # 定义Agent
5  physio_agent = PhysioGuardAgent()
6  psycho_agent = PsychoGuardAgent()
7  knowledge_agent = KnowledgeRetrieverAgent()
8  patient_agent = AISP_PatientAgent()
9  director = DirectorAgent()
10
11 # 定义工作流图
12 workflow = StateGraph(state_schema)
13 workflow.add_node("director", director)
14 workflow.add_node("physio_guard", physio_agent)
15 workflow.add_node("psycho_guard", psycho_agent)
16 workflow.add_node("knowledge_retriever", knowledge_agent)
17 workflow.add_node("patient", patient_agent)
18
19 # 定义边 (协作流程)
20 workflow.set_entry_point("director")
21 workflow.add_edge("director", "physio_guard")
22 workflow.add_edge("director", "psycho_guard")
23 workflow.add_edge("director", "knowledge_retriever")
24 workflow.add_edge(["physio_guard", "psycho_guard", "knowledge_retriever"], "patient")
25 workflow.add_edge("patient", "director")
26
27 # 编译图
28 app = workflow.compile()
```

4.2 实时评估架构

WebSocket推送机制：

代码块

```
1 // 前端实时接收评估更新
2 socket.on('assessment_update', (data: AssessmentUpdate) => {
3     // 更新对应维度的实时分数
```

```
4     updateDimensionScore(data.dimension, data.score);
5
6     // 显示实时反馈提示
7     showFeedbackToast(data.feedback_message);
8 });
9
10 interface AssessmentUpdate {
11     session_id: string;
12     dimension: 'history_completeness' | 'clinical_relevance' |
13     'clinical_reasoning' | 'communication' | 'safety_awareness' | 'followup_plan';
14     score: number;
15     feedback_message: string;
16 }
```

4.1 LLM集成方案

LLM服务封装：

代码块

```
1 import openai
2 from typing import Dict, List, Optional
3
4 class AIService:
5     def __init__(self):
6         self.client = openai.OpenAI(api_key=os.getenv("OPENAI_API_KEY"))
7         self.model = "gpt-4-turbo-preview"
8
9     async def generate_response(
10         self,
11         messages: List[Dict],
12         temperature: float = 0.7,
13         max_tokens: int = 500
14     ) -> str:
15         response = await self.client.chat.completions.create(
16             model=self.model,
17             messages=messages,
18             temperature=temperature,
19             max_tokens=max_tokens
20         )
21         return response.choices[0].message.content
```

多智能体协调器：

代码块

```

1  from langchain.agents import AgentExecutor, create_openai_tools_agent
2  from langchain_core.prompts import ChatPromptTemplate
3
4  class AgentOrchestrator:
5      def __init__(self):
6          self.patient_agent = self._create_patient_agent()
7          self.knowledge_agent = self._create_knowledge_agent()
8          self.assessment_agent = self._create_assessment_agent()
9          self.feedback_agent = self._create_feedback_agent()
10
11     async def orchestrate_response(
12         self,
13         user_input: str,
14         session_context: Dict
15     ) -> Dict:
16         # 并行调用多个智能体
17         tasks = [
18             self.patient_agent.generate(user_input, session_context),
19             self.knowledge_agent.validate(user_input, session_context),
20             self.assessment_agent.evaluate(user_input, session_context)
21         ]
22
23         results = await asyncio.gather(*tasks)
24
25         # 合成最终响应
26         final_response = await self.feedback_agent.enhance(
27             results, session_context
28         )
29
30         return final_response

```

4.2 LangChain多智能体设计

病人智能体：

代码块

```

1  PATIENT_PROMPT = ChatPromptTemplate.from_messages([
2      ("system", """
3          你是Maria Rodriguez, 52岁的墨西哥裔教师，新诊断为2型糖尿病。
4
5          性格特点：
6          - 对新诊断感到焦虑和担忧
7          - 担心像母亲一样出现并发症
8          - 重视传统墨西哥饮食
9          - 家庭意识强，需要家人支持
10

```

```
11     回答要求：  
12         - 严格按照JSON剧本内容回应  
13         - 体现焦虑但愿意学习的情感  
14         - 对文化敏感话题有顾虑  
15         - 语言自然，符合人物背景  
16     """),  
17     ("human", "{input}")  
18 ])
```

医学知识智能体：

代码块

```
1 KNOWLEDGE_PROMPT = ChatPromptTemplate.from_messages([  
2     ("system", "")  
3     你是医学知识验证专家，专门负责验证糖尿病相关信息的准确性。  
4  
5     验证标准：  
6         - 基于ADA 2023糖尿病管理指南  
7         - 确保医学信息准确性 > 98%  
8         - 避免提供具体药物剂量建议  
9         - 识别潜在的医疗风险  
10  
11     输出格式：  
12     {{  
13         "is_accurate": boolean,  
14         "corrections": string[],  
15         "risk_level": "low" | "medium" | "high",  
16         "suggestions": string[]  
17     }}  
18     """),  
19     ("human", "{input}")  
20 ])
```

4.3 Vector DB设计

向量数据库架构：

代码块

```
1 import pinecone  
2 from sentence_transformers import SentenceTransformer  
3  
4 class VectorDatabase:  
5     def __init__(self):  
6         self.embedder = SentenceTransformer('all-MiniLM-L6-v2')
```

```

7     pinecone.init(api_key=os.getenv("PINECONE_API_KEY"))
8     self.index = pinecone.Index("medical-knowledge")
9
10    def store_knowledge(self, knowledge_id: str, content: str):
11        embedding = self.embedder.encode(content)
12        self.index.upsert(
13            vectors=[{
14                "id": knowledge_id,
15                "values": embedding.tolist(),
16                "metadata": {"content": content}
17            }]
18        )
19
20    def search_knowledge(self, query: str, top_k: int = 5):
21        query_embedding = self.embedder.encode(query)
22        results = self.index.query(
23            vector=query_embedding.tolist(),
24            top_k=top_k
25        )
26        return results

```

5.0 数据架构设计 (为什么选MongoDB)

5.1 PostgreSQL主库设计

数据库连接配置：

代码块

```

1  from sqlalchemy import create_engine
2  from sqlalchemy.ext.declarative import declarative_base
3  from sqlalchemy.orm import sessionmaker
4
5  DATABASE_URL = "postgresql://user:password@localhost:5432/aisp"
6
7  engine = create_engine(DATABASE_URL)
8  SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
9  Base = declarative_base()

```

数据表设计示例：

代码块

```

1  class User(Base):
2      __tablename__ = "users"
3

```

```
4     id = Column(UUID(as_uuid=True), primary_key=True, default=uuid.uuid4)
5     username = Column(String(50), unique=True, nullable=False)
6     email = Column(String(100), unique=True, nullable=False)
7     password_hash = Column(String(255), nullable=False)
8     role = Column(Enum('student', 'teacher', 'admin'), nullable=False)
9     created_at = Column(DateTime, default=datetime.utcnow)
10    updated_at = Column(DateTime, default=datetime.utcnow,
11                          onupdate=datetime.utcnow)
```

5.2 MongoDB日志存储

连接配置：

代码块

```
1 from pymongo import MongoClient
2
3 mongo_client = MongoClient("mongodb://localhost:27017/")
4 db = mongo_client["aisp_logs"]
5 conversations_collection = db["conversations"]
```

对话记录存储：

代码块

```
1 class ConversationLog:
2     def __init__(self):
3         self.collection = conversations_collection
4
5     def store_message(self, session_id: str, message_data: Dict):
6         document = {
7             "session_id": session_id,
8             "timestamp": datetime.utcnow(),
9             "message": message_data
10        }
11        self.collection.insert_one(document)
12
13    def get_conversation_history(self, session_id: str, limit: int = 50):
14        return self.collection.find(
15            {"session_id": session_id}
16        ).sort("timestamp", -1).limit(limit)
```

5.3 Redis缓存策略

缓存配置：

```
1 代码块 import redis
2
3     redis_client = redis.Redis(
4         host='localhost',
5         port=6379,
6         db=0,
7         decode_responses=True
8     )
```

缓存策略设计：

代码块

```
1 class CacheService:
2     def __init__(self):
3         self.redis = redis_client
4         self.default_ttl = 3600 # 1小时
5
6     def cache_session_state(self, session_id: str, state: Dict):
7         key = f"session:{session_id}:state"
8         self.redis.setex(key, self.default_ttl, json.dumps(state))
9
10    def get_session_state(self, session_id: str) -> Optional[Dict]:
11        key = f"session:{session_id}:state"
12        cached_data = self.redis.get(key)
13        return json.loads(cached_data) if cached_data else None
14
15    def cache_ai_response(self, prompt_hash: str, response: str):
16        key = f"ai:response:{prompt_hash}"
17        self.redis.setex(key, 1800, response) # 30分钟缓存
18
19    def invalidate_user_cache(self, user_id: str):
20        pattern = f"user:{user_id}:*"
21        keys = self.redis.keys(pattern)
22        if keys:
23            self.redis.delete(*keys)
```

6.0 部署架构设计 (Optional) (docker和K8S 两个部署架构 是否为二选一)

6.1 Docker容器化

多阶段构建Dockerfile:

代码块

```
1 # 前端构建阶段
```

```
2 FROM node:18-alpine AS frontend-build
3 WORKDIR /app/frontend
4 COPY frontend/package*.json .
5 RUN npm ci --only=production
6 COPY frontend/ ./
7 RUN npm run build
8
9 # 后端构建阶段
10 FROM node:18-alpine AS backend-build
11 WORKDIR /app/backend
12 COPY backend/package*.json .
13 RUN npm ci --only=production
14 COPY backend/ ./
15
16 # 生产镜像
17 FROM node:18-alpine AS production
18 WORKDIR /app
19 COPY --from=frontend-build /app/frontend/dist ./public
20 COPY --from=backend-build /app/backend .
21 EXPOSE 3000
22 CMD ["npm", "start"]
```

Docker Compose配置：

代码块

```
1 version: '3.8'
2 services:
3   frontend:
4     build: .
5     ports:
6       - "3000:3000"
7     environment:
8       - NODE_ENV=production
9       - API_URL=http://backend:5000
10
11   backend:
12     build: .
13     ports:
14       - "5000:5000"
15     environment:
16       - DATABASE_URL=postgresql://user:pass@postgres:5432/aisp
17       - REDIS_URL=redis://redis:6379
18     depends_on:
19       - postgres
20       - redis
```

```

21
22   postgres:
23     image: postgres:15
24     environment:
25       POSTGRES_DB: aisp
26       POSTGRES_USER: user
27       POSTGRES_PASSWORD: pass
28     volumes:
29       - postgres_data:/var/lib/postgresql/data
30
31   redis:
32     image: redis:7-alpine
33     volumes:
34       - redis_data:/data
35
36   volumes:
37     postgres_data:
38     redis_data:

```

6.2 Kubernetes编排

部署配置：

代码块

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: aisp-backend
5  spec:
6    replicas: 3
7    selector:
8      matchLabels:
9        app: aisp-backend
10   template:
11     metadata:
12       labels:
13         app: aisp-backend
14   spec:
15     containers:
16       - name: backend
17         image: aisp/backend:latest
18         ports:
19           - containerPort: 5000
20         env:
21           - name: DATABASE_URL
22             valueFrom:

```

```

23         secretKeyRef:
24             name: aisp-secrets
25             key: database-url
26         resources:
27             requests:
28                 memory: "256Mi"
29                 cpu: "250m"
30             limits:
31                 memory: "512Mi"
32                 cpu: "500m"
33     ---
34 apiVersion: v1
35 kind: Service
36 metadata:
37     name: aisp-backend-service
38 spec:
39     selector:
40         app: aisp-backend
41     ports:
42         - port: 80
43             targetPort: 5000
44     type: ClusterIP

```

6.3 CI/CD流程

GitHub Actions配置：

代码块

```

1 name: CI/CD Pipeline
2
3 on:
4     push:
5         branches: [main]
6     pull_request:
7         branches: [main]
8
9 jobs:
10    test:
11        runs-on: ubuntu-latest
12        steps:
13            - uses: actions/checkout@v3
14            - uses: actions/setup-node@v3
15                with:
16                    node-version: '18'
17            - name: Install dependencies

```

```

19      run: npm ci
20
21      - name: Run tests
22          run: npm test
23
24      - name: Run linting
25          run: npm run lint
26
27      - name: Type checking
28          run: npm run typecheck
29
30  build:
31      needs: test
32      runs-on: ubuntu-latest
33      if: github.ref == 'refs/heads/main'
34
35  steps:
36      - uses: actions/checkout@v3
37
38      - name: Build Docker image
39          run: docker build -t aisp/backend:${{ github.sha }} .
40
41      - name: Push to registry
42          run: |
43              echo ${{ secrets.DOCKER_PASSWORD }} | docker login -u ${{ secrets.DOCKER_USERNAME }} --password-stdin
44              docker push aisp/backend:${{ github.sha }}
45
46      - name: Deploy to Kubernetes
47          run: |
48              kubectl set image deployment/aisp-backend backend=aisp/backend:${{ github.sha }}
49          kubectl rollout status deployment/aisp-backend

```

7.0 监控与安全(Optional)

7.1 性能监控

应用性能监控：

代码块

```

1 import { createPrometheusMetrics } from 'prom-client';
2
3 const httpRequestDuration = new createPrometheusMetrics.Histogram({

```

```
4     name: 'http_request_duration_seconds',
5     help: 'Duration of HTTP requests in seconds',
6     labelNames: ['method', 'route', 'status']
7   });
8
9   const activeConnections = new createPrometheusMetrics.Gauge({
10     name: 'websocket_active_connections',
11     help: 'Number of active WebSocket connections'
12   });

```

日志聚合:

代码块

```
1 import logging
2 from pythonjsonlogger import jsonlogger
3
4 logger = logging.getLogger()
5 logger.setLevel(logging.INFO)
6
7 json_handler = logging.StreamHandler()
8 formatter = jsonlogger.JsonFormatter()
9 json_handler.setFormatter(formatter)
10 logger.addHandler(json_handler)
```

7.2 安全措施

API安全:

代码块

```
1 import helmet from 'helmet';
2 import rateLimit from 'express-rate-limit';
3
4 // 安全头设置
5 app.use(helmet({
6   contentSecurityPolicy: {
7     directives: {
8       defaultSrc: ["'self'"'],
9       styleSrc: ["'self'", "'unsafe-inline'"'],
10      scriptSrc: ["'self'"'],
11      imgSrc: ["'self'", "data:", "https:"]
12    }
13  }
14 }));
15
```

```
16 // 速率限制
17 const limiter = rateLimit({
18   windowMs: 15 * 60 * 1000, // 15分钟
19   max: 100, // 限制每IP 100次请求
20   message: 'Too many requests from this IP'
21 });
22
23 app.use('/api/', limiter);
```

数据加密：

代码块

```
1 from cryptography.fernet import Fernet
2
3 class DataEncryption:
4     def __init__(self):
5         self.key = os.getenv("ENCRYPTION_KEY").encode()
6         self.cipher = Fernet(self.key)
7
8     def encrypt_sensitive_data(self, data: str) -> str:
9         encrypted_data = self.cipher.encrypt(data.encode())
10        return encrypted_data.decode()
11
12    def decrypt_sensitive_data(self, encrypted_data: str) -> str:
13        decrypted_data = self.cipher.decrypt(encrypted_data.encode())
14        return decrypted_data.decode()
```

7. 项目排期表

序号	任务	起止时间	负责人	任务描述	备注
1	需求梳理	2026-01-12~ 2026-01-13 (2天)	NICK、路晨	梳理 AISP项目的调研	
2	PRD 撰写	2026-01-14~ 2026-01-15 (2 天)	NICK	列出初步的 PRD	
3	原型设计	2026-01-16~ 2026-01-16 (1 天)	NICK	使用 vibe coding 实现原 型	
4	技术架构选 型	2026-01-15~ 2026-01-16 (2天)	广忠	技术架构梳理	
5					
6					

8. 标准参考

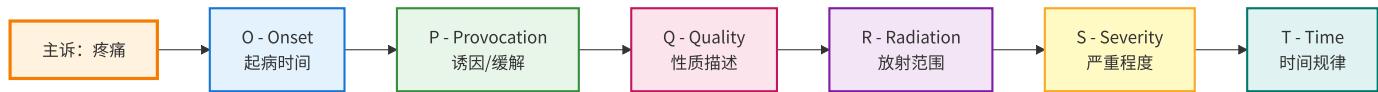
- **ICD-11:** <https://icd.who.int/>
- **SNOMED CT:** <https://confluence.ihtsdotools.org/>
- **FHIR:** <https://hl7.org/fhir/>

附录

附录 1：OPQRST 详细说明

🎯 什么是 OPQRST？

OPQRST 是医学中用于**疼痛症状评估**的经典结构化问诊框架。每个字母代表一个关键的问诊维度，帮助医生全面了解患者的疼痛症状，也可作为通用症状特征采集模型。



字母	英文全称	中文含义	核心问题	患者回答示例（胸痛病例）
O	Onset	起病时间	什么时候开始的？怎么开始的？	"三天前上午十点左右，突然开始的。"
P	Provocation	诱因/缓解	什么情况会加重？什么能缓解？	"走路快了会加重，休息一会儿就好转。"
Q	Quality	性质描述	是什么样的感觉？	"就像胸口压了块石头，闷闷的。"
R	Radiation	放散范围	疼痛会传导到别处吗？	"有时候会感觉到左边肩膀和胳膊。"
S	Severity	严重程度	有多难受？能打分吗？	"如果10分最痛，大概5-6分吧。"
T	Time	时间规律	一直痛还是阵发性的？	"一阵一阵的，每次几分钟就过去了。"

快速记忆口诀

- O = When? (什么时候开始)
- P = What makes it better/worse? (什么影响)
- Q = What's it like? (什么感觉)
- R = Where does it go? (放射哪里)
- S = How bad? (多严重)
- T = When/How often? (何时/多频繁)

OPQRST 典型应用场景

场景示例：头痛患者

维度	问句	可能的回答
O	头痛什么时候开始的？	"两天前开始的"
P	什么会加重/缓解？	"阳光照着会加重，睡觉能缓解"
Q	什么样的痛？	"像血管在跳，一跳一跳的"
R	疼痛会跑到别的地方吗？	"有时候会觉得后脖子也痛"
S	多严重？	"大概七八分，很难受"
T	什么时候更严重？	"早上起来最厉害"

AISP 中的 OPQRST 实现规范

代码块

```
1  OPQRST 采集实现:  
2  O - Onset:  
3      医生问句: "什么时候开始的? 怎么开始的?"  
4      患者回答: "具体时间 + 起病方式(突然/逐渐)"  
5      示例: "三天前上午突然开始的"  
6  
7  P - Provocation:  
8      医生问句: "什么情况会加重? 什么能缓解?"  
9      患者回答: "加重因素 + 缓解方式"  
10     示例: "活动后加重, 休息缓解"  
11  
12 Q - Quality:  
13     医生问句: "是什么样的感觉?"  
14     患者回答: "患者语言描述, 非医学术语"  
15     注意: "像压了块石头" ✓ | "压榨样疼痛" ✗  
16  
17 R - Radiation:  
18     医生问句: "疼痛会传导到其他地方吗?"  
19     患者回答: "放射部位 + 分布"  
20     示例: "会到左肩和左臂"  
21  
22 S - Severity:  
23     医生问句: "有多难受? 能打分吗?"  
24     患者回答: "0-10分评分 或 形象描述"  
25     示例: "大概5-6分" 或 "影响睡觉了"  
26  
27 T - Time:
```

- 28 医生问句: "是一直痛还是阵发性的?"
 29 患者回答: "持续性/阵发性 + 每次时长"
 30 示例: "一阵一阵的, 每次3-5分钟"

采集框架: OLD CARTS (备选)

OLD CARTS 是 OPQRST 的扩展版本

适用于**所有症状** (不限于疼痛) 的详细采集

字母	含义	问句示例	与 OPQRST 的关系
O	Onset	起病时间和方式	对应 OPQRST 的 O
L	Location	具体部位	新增: 明确疼痛/症状部位
D	Duration	持续时间	对应 OPQRST 的 T
C	Characteristics	症状特征	对应 OPQRST 的 Q
A	Aggravating/Alleviating	加重/缓解因素	对应 OPQRST 的 P
R	Radiation	放射情况	对应 OPQRST 的 R
T	Timing	时间规律	对应 OPQRST 的 T
S	Severity	严重程度	对应 OPQRST 的 S