AwesomeCS 系统架构说明书

前言

本说明书依据 GB/T 45630-2025 架构描述规范，描述 AwesomeCS 后端系统（博客、评论、付费咨询、模拟面试、AI 面试助手等）的系统架构。文档面向系统架构师、后端开发、测试与运维人员，用于设计、实现和运维参考。

引言

- 编写目的：提供系统架构的完整说明，便于开发、测试、部署与运维实现与判断。
- 背景：AwesomeCS 是面向开发者的知识与咨询平台，包含文章发布、评论/点赞、付费咨询、实时咨询会话与 AI 辅助面试模块。
- 参与方：产品经理、后端/前端工程师、测试人员、DBA、运维、安全审计。

目录

---

1 范围

本说明书覆盖系统的逻辑架构、组件划分、运行/部署方式、关键接口、数据架构、非功能需求以及运维与安全要求。涉及 REST API、WebSocket 实时通信、数据库与外部 AI/支付服务的集成要点。

2 规范性引用文件

- GB/T 45630-2025（架构描述标准）
- 项目需求文档
- MySQL 官方手册、Spring Boot 文档、Sa-Token 文档

3 术语和定义

- Service：后端业务层组件
- Controller：REST/WebSocket 入口
- Repository: 数据仓储层
- Mapper: 与数据库交互的入口
- token：Sa-Token 会话/短期 WebSocket 握手凭证

4 符合性

本说明书按照 GB/T 45630-2025 的章节与视图要求组织，架构实现遵循行业通用的 REST、WebSocket 标准及 Spring Boot 与 MyBatis-Plus 的最佳实践。

5 基础概念

- 技术栈：Java 17、Spring Boot、MyBatis-Plus、Sa-Token、WebSocket、MySQL、Gson
- 设计原则：分层架构（Controller/Service/Repository）、接口优先、单一职责、面向接口注入
- 命名规范：数据库 snake_case，Java 驼峰，API 返回字段统一下划线（通过 DTO 控制）

6 架构描述规格说明

6.1 视图集合

- 逻辑视图（模块/组件）
- 运行时视图（序列/交互）
- 部署视图（节点/网络）
- 数据视图（模型/索引）
- 安全与运维视图

6.2 架构元素说明

- Web 层（Controller, WS）
  - 主要文件夹：
    - src/main/java/com/yorozuya/awesomecs/controller
    - src/main/java/com/yorozuya/awesomecs/ws_config
  - 责任：请求解析、鉴权校验（Sa-Token）、参数校验、调用 Service、启动 WebSocket 连接
- 业务层（Service）
  - 主要文件夹：src/main/java/com/yorozuya/awesomecs/service(/impl)
  - 责任：业务编排、事务管理、幂等/并发控制
  - AI服务 （AI）
    - 位置：src/main/java/com/yorozuya/awesomecs/service/ai
    - 责任：DashScope（语音识别）、DeepSeek（文本处理）、MiniMax（TTS）
- 数据访问层（Repository）
  - 数据库访问层（Mapper）
    - 位置：src/main/java/com/yorozuya/awesomecs/repository/mapper 与 src/main/resources/mapper
    - 责任：持久化、复杂 SQL 封装
  - Redis 内存访问层
    - 位置：src/main/java/com/yorozuya/awesomecs/repository/redis
    - 责任：与 Redis 内存数据库交互

- 实体类层 (model)
  - 数据库实体类 (domain)
    - 位置：src/main/java/com/yorozuya/awesomecs/model/domain
    - 责任：负责与数据库关系进行映射
  - 请求响应 DTO 类 (request, response)
    - 位置：src/main/java/com/yorozuya/awesomecs/model/request 和 src/main/java/com/yorozuya/awesomecs/model/response
    - 责任：负责映射 Controller 对应的请求类和响应类
- 常量池 （common）
  - 位置：src/main/java/com/yorozuya/awesomecs/comon
  - 责任：管理着项目的公共部分，比如全局异常处理器，以及返回的错误码等等

6.3 MVC 架构

为了明确架构责任、方便维护与测试，本项目采用经典的 MVC（Model-View-Controller）思想在后端进行分层映射。下面说明三者在工程中的具体对应关系、调用流程与设计建议。

1. Model（模型）

- 对应文件/包：
  - 实体：`src/main/java/com/yorozuya/awesomecs/model/domain`（数据库实体）
  - 持久化：`src/main/java/com/yorozuya/awesomecs/repository/mapper`（MyBatis-Plus Mapper）
  - 数据转换/DTO（部分属于 Model 与 View 之间的桥）：`src/main/java/com/yorozuya/awesomecs/model/request` 与 `.../model/response`
- 责任：表示领域数据结构、负责与数据库交互的最底层代码（Mapper），并提供必要的映射/转换工具。
- 设计要点：实体与数据库字段一一对应；不在实体中放置业务逻辑；复杂查询写在 Mapper 或 XML 中。

2. View（视图 / 表现层）

- 在后端语境下，View 指对外的 API 响应与前端展现：
  - REST JSON 响应：使用 `model/response` 下的 DTO，通过 @JsonProperty 控制下划线字段名
  - WebSocket 消息格式：约定的 JSON 或二进制协议（Interview 模块以二进制音频 + JSON 控制帧）
  - 前端页面/客户端（不在本仓库）负责最终呈现
- 责任：控制对外展示的字段与格式，避免直接暴露数据库实体，保证向前兼容性。
- 设计要点：使用专门的 Response DTO；避免在 Controller 中直接返回 Entity；对敏感或内部字段进行屏蔽或转换。

3. Controller（控制器）

- 对应文件/包：`src/main/java/com/yorozuya/awesomecs/controller`（含 REST controllers 与 WebSocket handlers）
- 责任：接收请求、校验鉴权、参数解析，调用 Service 层执行业务逻辑，并把结果转换为 View（Response DTO）返回。
- 设计要点：保持 Controller 轻量——只做校验与流转，所有业务逻辑、事务与复杂操作都应放在 Service 层；捕获异常并返回统一的 Result 结构。

4. 典型调用序列（MVC 映射）

```
Client -> Controller（校验 token, 参数）-> Service（业务逻辑, 事务）->
Mapper/Repository（DB 操作）

返回 domain/entity -> Service -> Controller -> Response DTO -> Client
```

- WebSocket 场景（咨询/面试聊天）：

```
Client(WebSocket) -> WebSocketHandler（鉴权 token, 会话路由）-> Service（消息
持久化, 业务校验）-> Mapper -> DB
Handler 同时负责将持久化后的消息广播给其他客户端（View 层的实时更新）
```

5. 具体实现

- Controller：
  - 使用注解验证（SaToken 框架提供的 @SaCheckLogin 等注解）与统一异常处理（全局异常处理器）
  - 参数命名采用 snake_case，通过 DTO 的 @JsonProperty 映射到驼峰 Java 字段
- Service：
  - 包含事务边界（@Transactional）并保证方法的幂等性（支付、回调等）
  - 保持可单元测试（尽量通过接口注入 Mapper/依赖）
- Mapper/Repository：
  - 只聚焦持久化操作，复杂 SQL 写在 XML/Mapper
  - 对频繁读写的表考虑乐观锁或在 Service 层使用 Redis 缓存策略
- DTO（Request/Response）：
  - 避免直接使用 Entity 作 API 响应；Response DTO 控制输出字段与格式
  - 对于大型返回（如帖子详情），通过组合 DTO 包含必要的统计字段（like_count、is_liked 等）
- WebSocket Handler：
  - 将其视为一种 Controller，保持逻辑轻量，调用 Service 做持久化与权限判断

7 安全设计与保密

- 鉴权：Sa-Token 用于 REST 与 WebSocket 握手短期 token
- 密码：bcrypt/argon2 存储
- 支付回调：签名校验与 consultation_payments 唯一约束保障幂等

8 非功能需求与质量属性

- 可用性：多实例 + 健康检查 + 负载均衡
- 可扩展性：水平扩展应用层，热点写使用 Redis 聚合
- 性能：阅读/点赞计数使用 Redis
- 可维护性：分层、DTO、统一异常与结果封装（Result）

9 关键设计决策

- Spring Boot + MyBatis-Plus：开发效率高、易维护
- WebSocket 在应用层处理：适合中小规模并发，需扩展时迁移到专用服务

- JSON 存储灵活但需按需规范化；domains/tags 如需高效检索建议生成虚拟列或单独表
- consultation_relation 每用户一条规则既在 Service 层也可在 DB 做唯一约束（user_id）作为二次保障

参考文献

- GB/T 45630-2025