设计报告-计科2203

1 引言

1.1 研究背景

近年来，全球数字人文与文化遗产研究蓬勃发展，海外博物馆不断开放藏品元数据及多媒体资源。与此同时，大量中国文物散落世界各地，其时代特征、艺术价值与背后历史仍鲜有系统化、可交互的知识服务平台。传统线下展陈和静态网页难以满足公众对深层语义检索、情境化叙事与移动端沉浸式体验的需求。建设一套**“海外中国文物知识管理与服务平台”**，不仅有助于推动中华文化的全球传播，也能为文物保护、跨博物馆研究与教育推广提供统一的信息基础。

1.2 项目目标

根据课程综合实验要求，本项目需完成“五位一体”的总体功能：

1. **知识图谱构建子系统**：自动化爬取指定海外博物馆（≥5家）的中国文物信息，完成数据清洗、三元组建模与图数据库入库；
2. **海外文物知识服务子系统**：提供多维筛选、图谱可视化及时间轴浏览等 Web 端服务；
3. **知识问答子系统**：基于图谱数据支持至少 10 种单实体-单属性问答；
4. **掌上博物馆（移动端）**：围绕单类文物（如瓷器）构建轻量级 App，实现展示、交互及以图搜图；
5. **后台管理子系统**：实现用户、内容、数据与日志的全链路运营维护。

1.3 研究意义

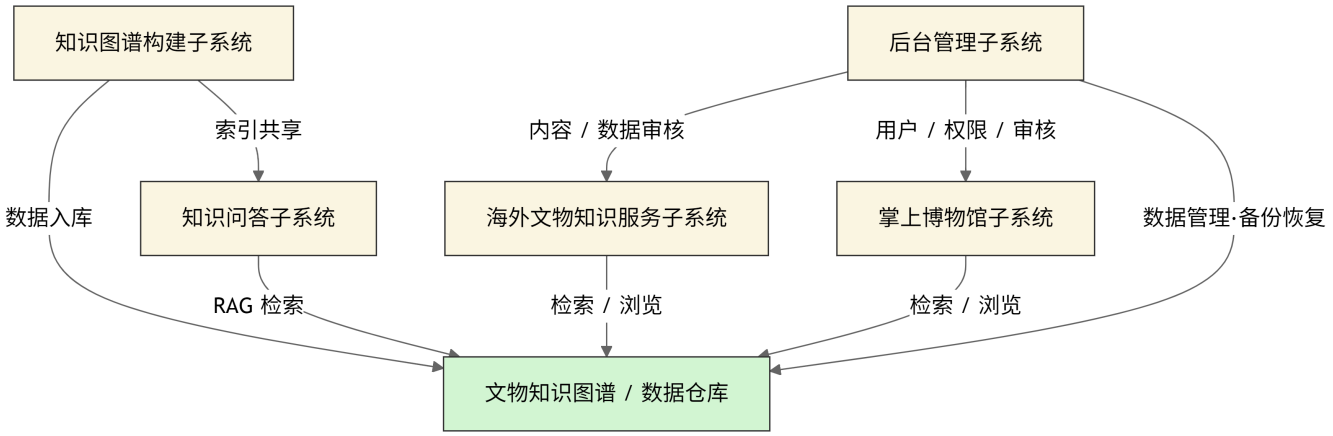
- **学术价值**：统一异构数据，实现海外文物的语义融合与关联推理，为数字人文研究提供可复用的开放知识基座。
- **文化传播**：以图谱＋可视化方式呈现文物脉络，让公众获得“看得到关系、读得懂故事”的交互体验。
- **工程示范**：全栈式实现从数据抓取、知识工程到多端应用的完整闭环，为今后同类系统的快速迭代奠定范式。

2 总系统设计概述

2.1 总体设计目标

- **一体化**：贯通“数据采集 → 知识增强 → 多端服务 → 运维管理”全流程，避免信息孤岛。
- **高可扩展**：五大子系统通过统一 **API 网关** 解耦，支持独立演进与弹性扩容。
- **高可用**：核心服务 7 × 24 小时在线，故障转移 ≤ 30 min，日增量 + 周全量双备份策略。
- **安全合规**：OAuth 2.0／JWT、多因素认证、RBAC、SQL·XSS·CSRF 全链路防护。

2.2 总体架构



2.3 子系统职责与关键接口

子系统	核心功能	典型接口	关键技术
知识图谱构建	多源爬虫、ETL、三元组建模、数据补充、入库	CSV 批量导入；Cypher/SPARQL	Scrapy + Selenium、CIDOC-CRM、Neo4j/Virtuoso
海外文物知识服务	Web 展示、筛选、检索、图谱可视化、以图搜图	REST/GraphQL	Spring Boot + Vue 3; D3.js; JWT
知识问答	单实体/复杂查询生成、上下文多轮、RAG	GraphQL; /qa	FastAPI + LangChain; Neo4j-Driver
掌上博物馆	HarmonyOS App：文物浏览、以图搜图、社区互动	REST; HMS ML Kit	ArkTS MVVM; AGC 云函数
后台管理	用户/权限/评论&帖子审核；文物数据管理；备份恢复	Dash 回调; /admin	Plotly Dash; MySQL; 任务调度

2.4 数据与控制流

1. **采集流**：知识图谱构建子系统定时推送增量三元组 → 文物知识图谱/数据仓库。
2. **检索流**：Web 服务、掌上博物馆通过 API 网关直接检索知识库；热点查询先命中 Redis 缓存。
3. **问答流**：知识问答子系统利用 RAG 机制向知识库查询实体/关系后生成答案。
4. **运维/管理流**：后台管理对子系统进行用户管理、内容审核、数据库备份恢复并写入审计日志。

2.5 技术选型与部署

层面	组件	说明
前端	Vue 3 + Element-Plus · ArkTS (HarmonyOS)	响应式 UI，美观易用。
后端	Spring Boot · FastAPI · Django	微服务架构，接口统一。
数据	Neo4j/Virtuoso · MySQL 8 · Redis	图 + 关系 + 缓存。
DevOps	Docker Compose · Kubernetes · Prometheus/Grafana	CI/CD、自动扩缩容、监控告警

2.6 非功能需求映射

属性	设计对策	需求来源
性能	≤ 3 s 页面加载；支持 1 000 QPS	
可靠性	双活备份、自动快照、故障转移 ≤ 30 min	
安全性	BCrypt 密码、JWT、RBAC、OWASP Top-10 防护	
可维护性	模块化、接口文档化、单元/集成测试	

3 子系统设计概述

3.1知识图谱构建子系统

3.1.1设计目标

1. 字段选择目标:

以全面记录中国文物基本属性、出处信息及媒体信息为目标，确保后续支持数据可视化、检索与知识图谱构建。分别选取三个博物馆网站的部分文物，提取出关键信息之后，再统一命名格式。

2. 三元组建模目标:

将结构化数据映射为文物编号与实体之间的语义关系，便于构建Neo4j等图数据库中的知识图谱。

3.1.2字段设计

1. 字段设计原则:

- 尽可能映射网站结构，做到字段完整、规整。
- 命名统一、含义明确，便于程序处理和团队协作。
- 考虑未来图谱构建对实体关系的要求，字段要具备实体/属性特征。

2. 原始字段格式

通过爬虫，我们初步得到三个博物馆的字段分布如下:

- Smithosian :

Title	Artist	Place of Origin	Date	Material
标题	艺术家	原产地	日期	材料
On View	inscribed/Inscripions	Description	save_path	Image URL
是否展出	铭文/题字	描述	图片保存路径	图片链接
Credit Line	Style	Type	Dimensions	Medium
藏品来源说明	风格	类型	尺寸	媒介

- (2) Asian Art Museum:

Img_path	Credit Line	Portfolio	Materials	Artist	Markings	Place of Origin
图片路径	藏品来源说明	系列/作品集	材料	艺术家	标记/印记	原产地
Publisher	On View	Department	School	Date	Inscribed	Title
出版者	是否展出	部门	流派/学院	日期	铭文	标题
Signed	Classifications	Printer	Dynasty	More Information	Img_url	Dimensions
签名	分类	印刷者	朝代	更多信息	图片链接	尺寸
Period	Culture	Provenance Statement	Object number	Composer		
时期	文化	来源说明	藏品编号	作曲者		

- (3) Chicago Museum

title	artist_display	date_display	place_of_origin	category_titles
标题	艺术家展示名	日期展示	原产地	分类标签
description	medium_display	dimensions	provenance_text	exhibition_history
描述	媒介展示	尺寸	来源说明	展览历史
publication_history	credit_line	department_title	style_title	artwork_type_title
出版历史	藏品来源说明	所属部门	风格名称	艺术品类型名称
technique_titles	subject_titles	image_url	image_path	
技术标签	主题标签	图片链接	图片路径	

3.1.3数据格式设计

MySQL数据格式

（1）对三个爬虫网站的数据进行整合，对各个字段进行全面的检查和对比。我们发现部分字段在不同的爬虫数据当中表示的是相同或者相似的信息，比如 `Artist`（创作者）在各个数据源当中有类似的表述，但是在字段名称上面存在差异，有 `Artist` 与 `artist_display` 两种命名格式，将其统一为 `Artist` ；

（2）兼顾完整性与结构化。将三个博物馆网站当中的关键属性提取出来,既保留了详细信息，也实现规模化管理，遵循实体关系设计原则；

（3）聚焦于文物的核心属性（比如艺术品、朝代等）和的定义（如艺术品所属朝代、材料等）框架。

最终确定保留与合并以下属性名（同时对创作者信息进行分割，分为仅姓名和信息丰富的两种格式）：

id	Title	Artist	OnlyArtist	Dynasty	Classifications
编号	标题	创作者(包括籍贯和生卒年)	创作者	朝代	分类
Dimensions	**Materials**	**Description**	**Museum**	**PlaceOri**	**Inscribed**
尺寸	材料	描述	博物馆	原产地	铭文/题字
ImgPath	**ImgUrl**	**Medium**	**CreditLine**		
图片路径	图片链接	媒介	藏品来源说明		

考虑到Web组需要做时间轴可视化处理，而这里的朝代信息较复杂，杂糅了主时间段和次时间段，因此用正则表达式进一步处理朝代信息，同时分割成下面字段：

Dynasty	MainDynasty	StartYear	EndYear	SubDynasty	SubStartYear	SubEndYear
朝代	主时间段朝代	起始时间	结束时间	次时间段朝代	起始时间	结束时间
这种设计易于筛选，并且按照时间排序，实现了时间粒度的细化，无论数据具体细节如何变动，都能统一处理。						

最后以下面的格式存进MySQL数据库：

列名	释义	Type	列名	释义	Type
id	**唯一编号**	**int**	Museum	现藏博物馆	varchar
Title	标题	varchar	PlaceOri	来源地	varchar
Artist	创作者	varchar	Classifications	分类	varchar
Dynasty	年代	varchar	Medium	媒介	varchar
Credit	来源说明	varchar	*OnlyArtist*		
Dimensions	尺寸	varchar	main_start	主时代开始年份	int
ImgPath	本地路径	varchar	main_end	主时代结束年份	int
ImgUrl	网络链接	varchar	sub_start	次时代开始年份	int
Materials	材料	varchar	sub_end	次时代结束年份	int
Description	描述	text	*Periods*		
Inscribed	铭文/题词	text			
其中 `id` 为主键，不可为空。					

知识图谱数据格式

1. 主要实体 `Artifacts` （文物）属性：

id	Title	Artist	Dynasty	CreditLine	Dimensions	Materials
编号	标题	创作者	年代	来源说明	尺寸	材料
Inscribed	**Museum**	**PlaceOri**	**Classifications**	**Medium**	**Description**	
铭文/题词	博物馆	来源地	分类	媒介	分类	

2. 其他实体名：

用文物的 `id` 属性和其他实体的唯一属性名相联系生成三元组：

Artist	Dynasty	Dimension	Material	Museum	PlaceOri	Classification	Medium
创作者是	创造年代是	尺寸是	材质是	现藏博物馆是	来源地是	类型是	媒介是

知识图谱中实体关系清晰，便于后续的推理、搜索和可视化。

3.2 海外文物知识服务子系统详细设计

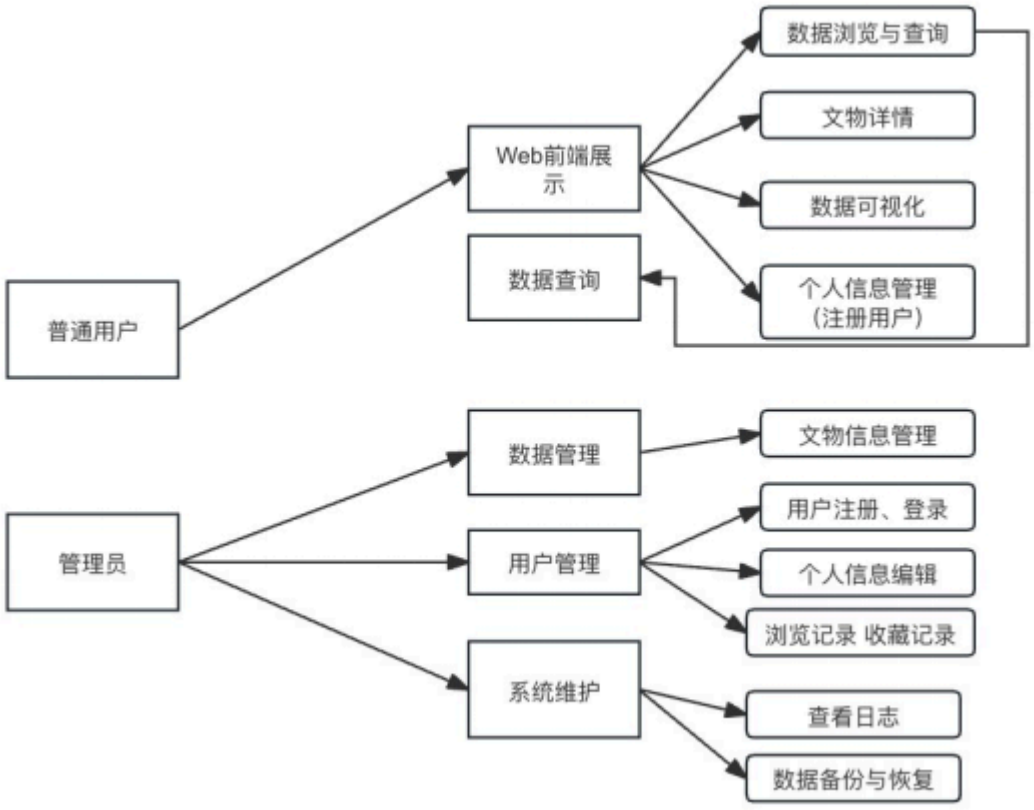
3.2.1 系统功能介绍

（1）数据浏览：支持多种形式展示的浏览功能。①提供基本的筛选、排序功能，可按照文物类型、文物年代、博物馆等多种基础信息进行索引、筛选、排序方式浏览文物信息以方便用户的使用。②提供查看文物详情功能，显示文物的详细数据，如文本、图像等信息，点击文物图片，可以进行放大缩小。③相关文物推荐功能，在该文物页面显示相关文物，相关规则自定，如相似主题、同一作者、图像内容相似等。

（2）数据查询：支持文物的简单查询功能和高级查询功能。简单查询根据输入的关键字，如文物名称、博物馆名称、文物年代等进行查询。高级查询可以对文物的多个字段进行限定查询。

（3）数据可视化显示：支持两种可视化。①包含结点、边的力导向图知识图谱展示。②文物时间轴：按照时间轴的方式、展示各个时段的文物信息、时间等信息。

3.2.2 体系结构设计



3.2.3 接口设计

3.2.3.1 *认证相关接口（AuthController):*

/login - 用户登录

/regest - 用户注册

/forget - 忘记密码

/getCode - 获取邮箱验证码

3.2.3.2**文物相关接口（ArtController）:

/relic/relic_types - 获取所有文物类别

/relic/relic_times - 获取所有朝代信息

/relic/relic_museums - 获取博物馆列表

/relic/timeline - 获取时间轴数据

/relic/timelineinfo - 获取指定时间范围的文物信息

/relic/relic_search - 文物搜索接口，支持多条件筛选

3.2.3.3**用户相关接口（UserController）:

/user/upgradeAvatar - 更新用户头像

/user/sendInfo - 更新用户信息

/user/myfav - 获取用户收藏

/user/disfav - 取消收藏

/user/browser - 获取浏览历史

/user/delbrowser - 删除浏览历史

/user/mycomts - 获取用户评论

/user/discomts - 删除用户评论

3.2.3.4**评论相关接口（CommentController）:

/comments/search - 搜索评论

/comments/add - 添加评论

3.2.3.5**认证文物相关接口（AuthedArtController）:

/AuthRelic/relic_detail - 获取文物详情

/AuthRelic/relic_isfav - 收藏/取消收藏文物

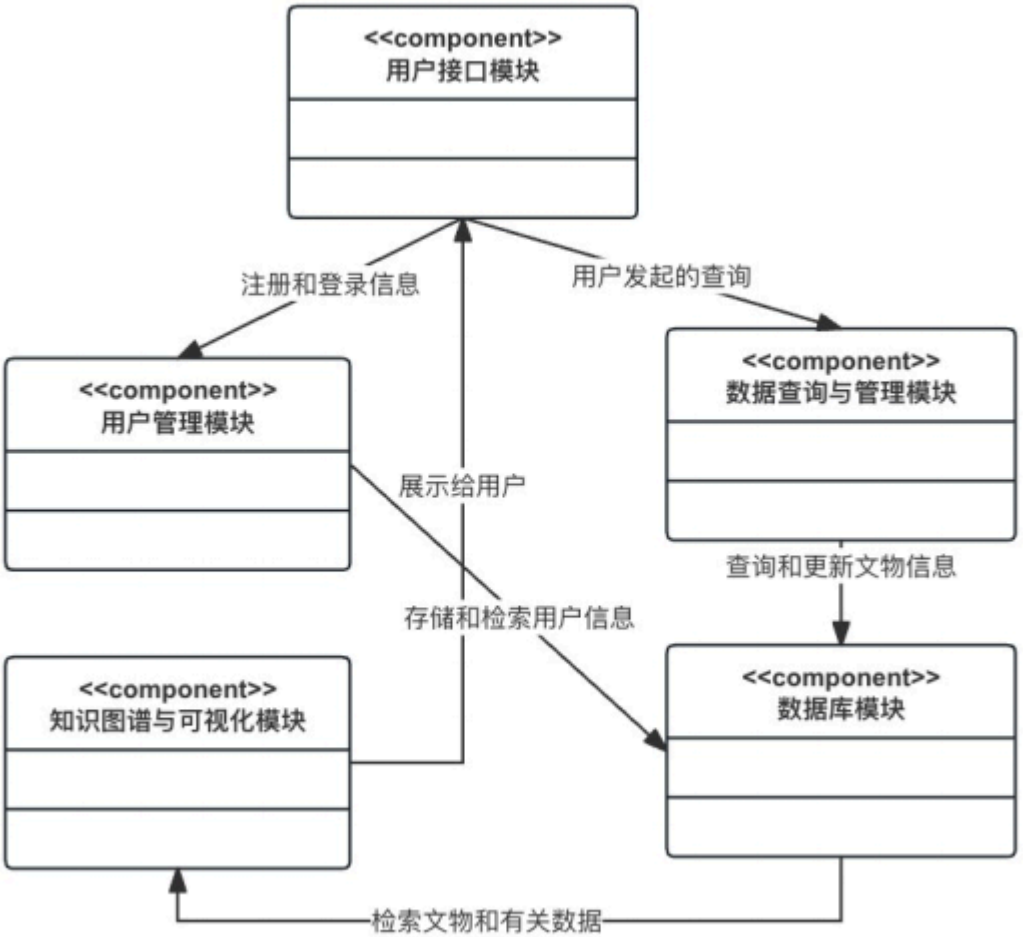
/AuthRelic/rehistory - 添加浏览历史

3.2.4 数据设计

Mysql 数据库的数据设计和后台管理子系统的设计相同。

neo4j 数据库的数据设计与知识图谱子系统的设计相同。

3.2.5 模块设计



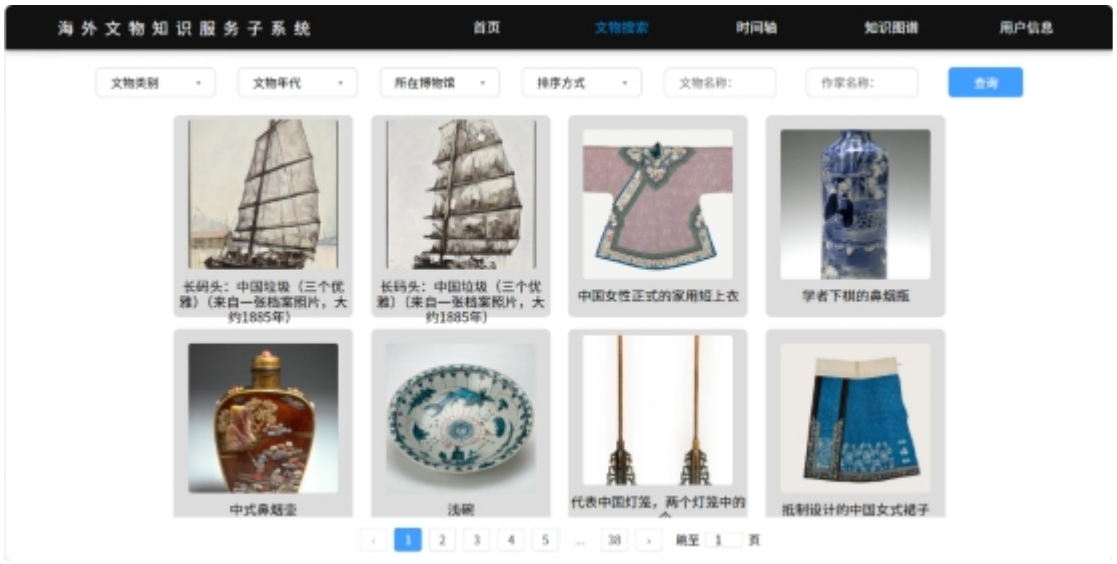
3.2.6 界面设计

主要为 web 端各种界面的设计。主要有：首页、文物浏览页面、知识图谱页面、文物时间轴页面、用户信息页面.

3.2.6.1 首页

首页下方共有五张高清文物图片，可任意选择切换文物图片。

3.2.6.2 文物浏览页面



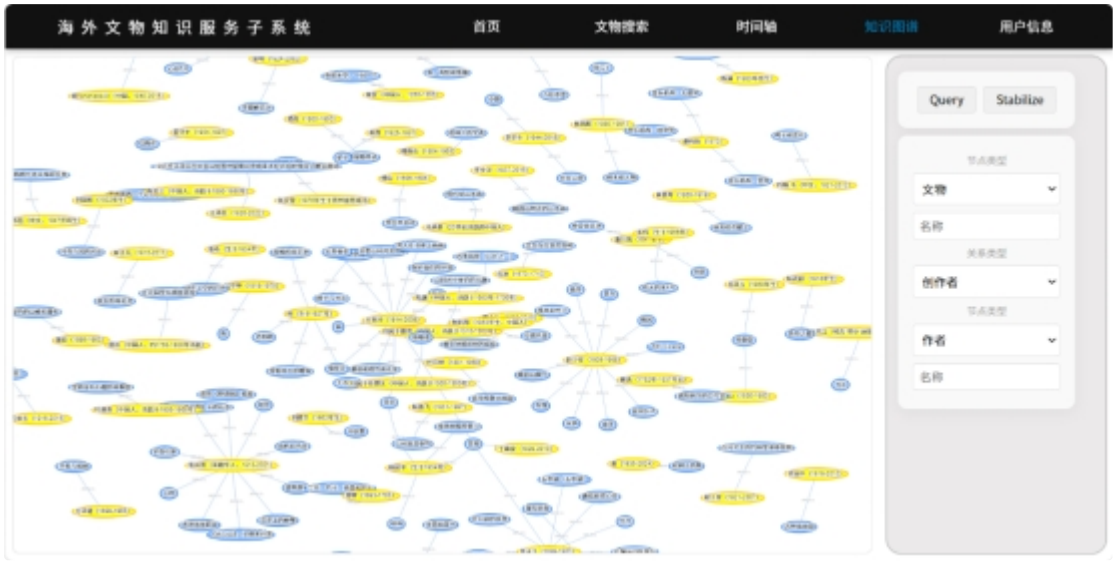
页面主体部分展示所有文物。文物展示页面如上图，对于每件文物，上面为文图图片，下面为文物的名称。

页面上方为文物浏览提供基本的筛选、排序功能，可按照文物类型、文物年代、博物馆等多种基础信息进行索引、筛选、也可按文物名称或文物ID或文物年代进行排序。

点击文物图片，可以进行文物具体信息的浏览，跳转到文物详情介绍页面。

可进行分页查询，优化大数据量的展示。

3.2.6.3 知识图谱页面



有多种节点类型和关系类型可选择（下图以文物的创作者为例，可以清晰看到每个创作者创作的文物）

知识图谱形成过程是动态的，有Stablize功能使其静止。

3.2.6.4 文物时间轴页面

按时间线展示文物，可梳理文物的历史发展轨迹。点击右边箭头即可滑动时间轴。

任意点击一个时间节点，可以展示出这个时间段的文物信息

*3.2.6.**5***　*文物详情页面*

左侧为文图图片，右侧为文物详细介绍，包括文物图片、名称、作者、类型、介绍。

具有文物评论和收藏功能。

页面底端有相似文物推荐，显示相似文物的名称。

*3.2.6.**5***　*用户信息**页面***

用户登录页面

可进行用户注册，找回密码。其中找回密码使用邮箱验证，登录方式可以选择账号或邮箱

个人信息管理：

修改个人信息功能：可上传头像，修改密码和电子邮箱。

我的收藏：

查看收藏的文物功能：展示了收藏过的所有文物及收藏的时间，可以选择查看文物具体信息或删除收藏记录。

我的评论：

查看我的评论功能：展示评论过的所有文物和评论内容和评论的时间，可以选择查看文物具体信息或删除评论记录。

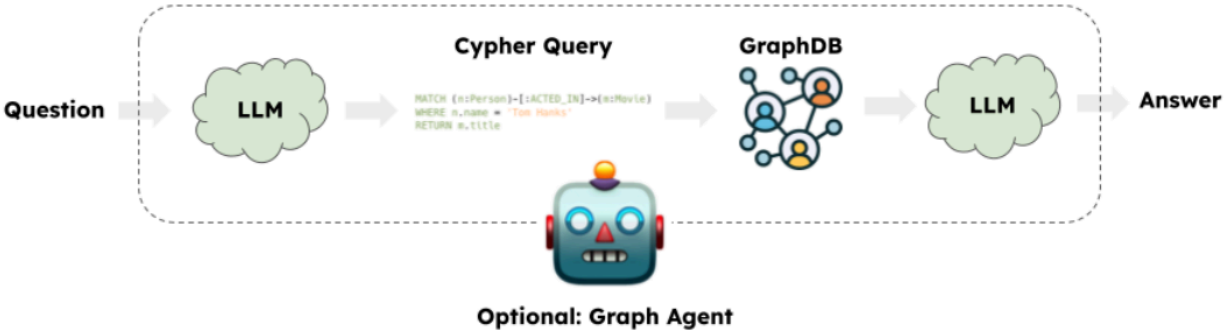
浏览历史：

查看浏览记录功能：展示浏览过的所有文物，从晚到早按时间排序，可以选择查看文物具体信息或删除浏览记录。

3.3 知识问答子系统

3.3.1 系统功能简介

- ***核心功能***：
  1. ***精准问答***：支持用户输入自然语言问题（如“龙造型鼻烟壶的年代？”），系统从知识图谱中检索结构化答案（如“1986年”）。
  2. ***多轮对话***：基于上下文继承能力处理连续提问（如“它位于哪里？”），维持对话连贯性。
  3. ***复杂语义解析***：解析多实体/多属性问题（如“葫芦形和双龙鼻烟壶的材质”），返回分项结果。
  4. ***闲聊***：简单回答非文物问题（如“今天天气？”），友好引导用户回归文物主题。
- ***技术目标***：
  1. 实现 ***LangChain-RAG框架*** 与 ***Neo4j知识图谱*** 的高效协同。



1. 通过 ***Django后端*** 动态处理查询，***Vue3前端*** 提供即时交互体验。

3.3.2 体系结构设计

- ***系统交互架构*** ：
- ***关键连接*** ：
  1. ***知识图谱子系统*** ：
     - 接收知识图谱数据。
     - 数据格式：***JSON*** （含文物名称、描述、类别等字段）。

3.3.3 接口设计

- ***核心接口*** ：

接口名称	***输入***	***输出***	***功能说明***
/**back*/****view/chat****	***{ "**message****": "..." }****	***{ "**final_answer****": "..." }****	处理聊天请求
/**back*/****view/set_username****	***{ *“**message****” }****	***{ "**message****" }****	处理用户逻辑，保存userid，讲历史记录返回前端
/**langchain_module/chatHistory/stream_response*	***{ *“**content****” }****	***{ "**response****" }****	将问题作为输入，之后在此接口调用知识图谱查询相关操作，最后将查询结果结合上下文返回最终结果
/langchain_module/chatArtifact/process_question	***{**“****query****” ****}****	***{**“****final_cypher****” ****}****	问题处理主函数，获取LLM生成的Cypher，并调用函数execute_cypher与refine_cypher，并最终返回Cypher查询语句与查询结果
/langchain_module/chatArtifact/get_artifact_answer	***{**“****question****” ****,****”****history****” ****}****	***{**“****answer****” ****}****	核心调用函数，调用process_question和format_result，串联项目总流程

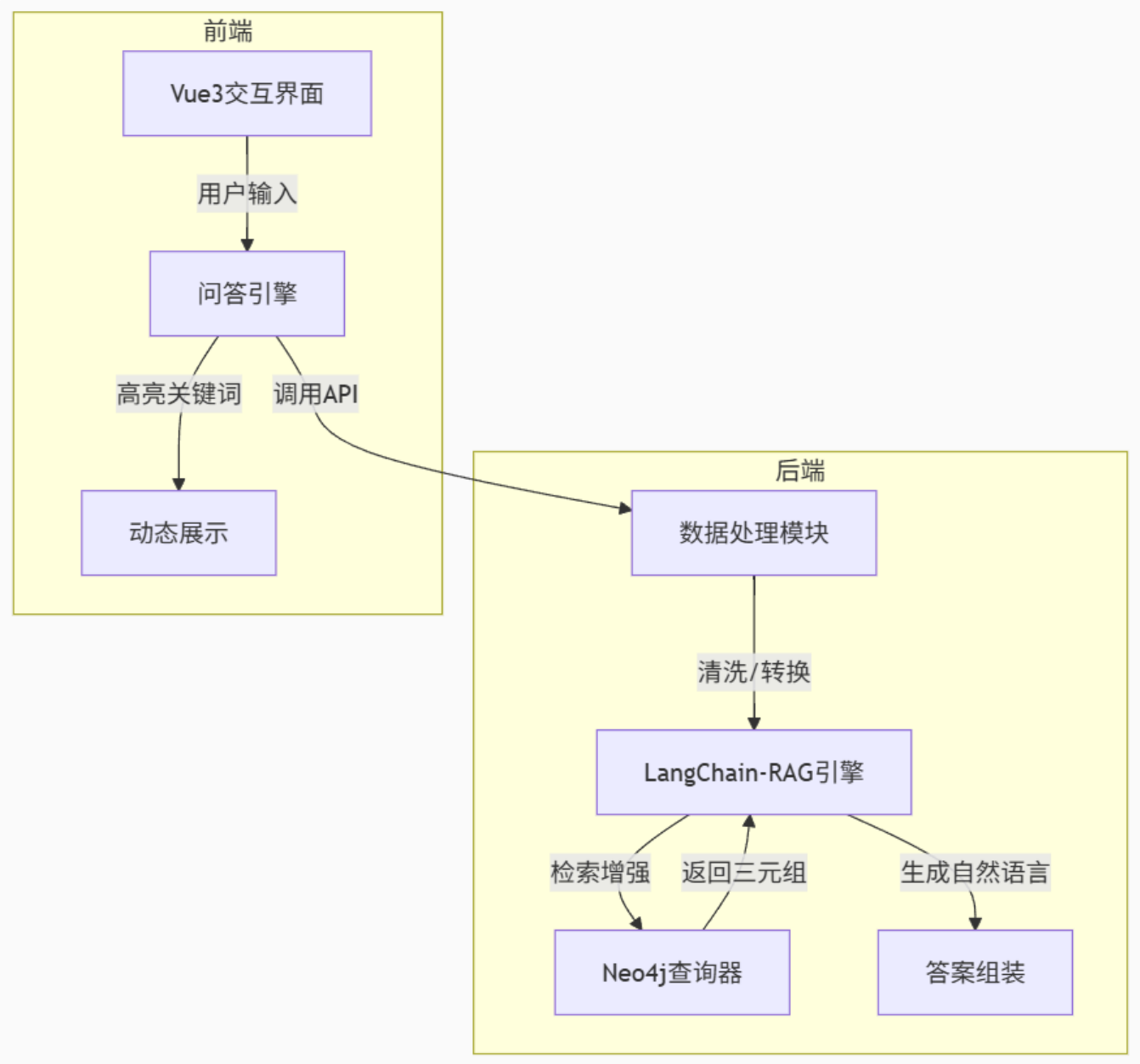
- ***实现方式*** ：
  - 前端通过 ***JavaScript*** 调用接口，动态解析返回的 JSON 数据。
  - 后端使用 ***Django*** 构建轻量级 API。

3.3.4 数据设计

- ***数据源***：
  - ***Neo4j 知识图谱***：
    - 节点类型：***文物***、***朝代***、***作者***、***材质***（继承知识图谱子系统设计）。
    - 关系：***制造于***、***创作者***、***藏品材质***等。
  - 查询示例：

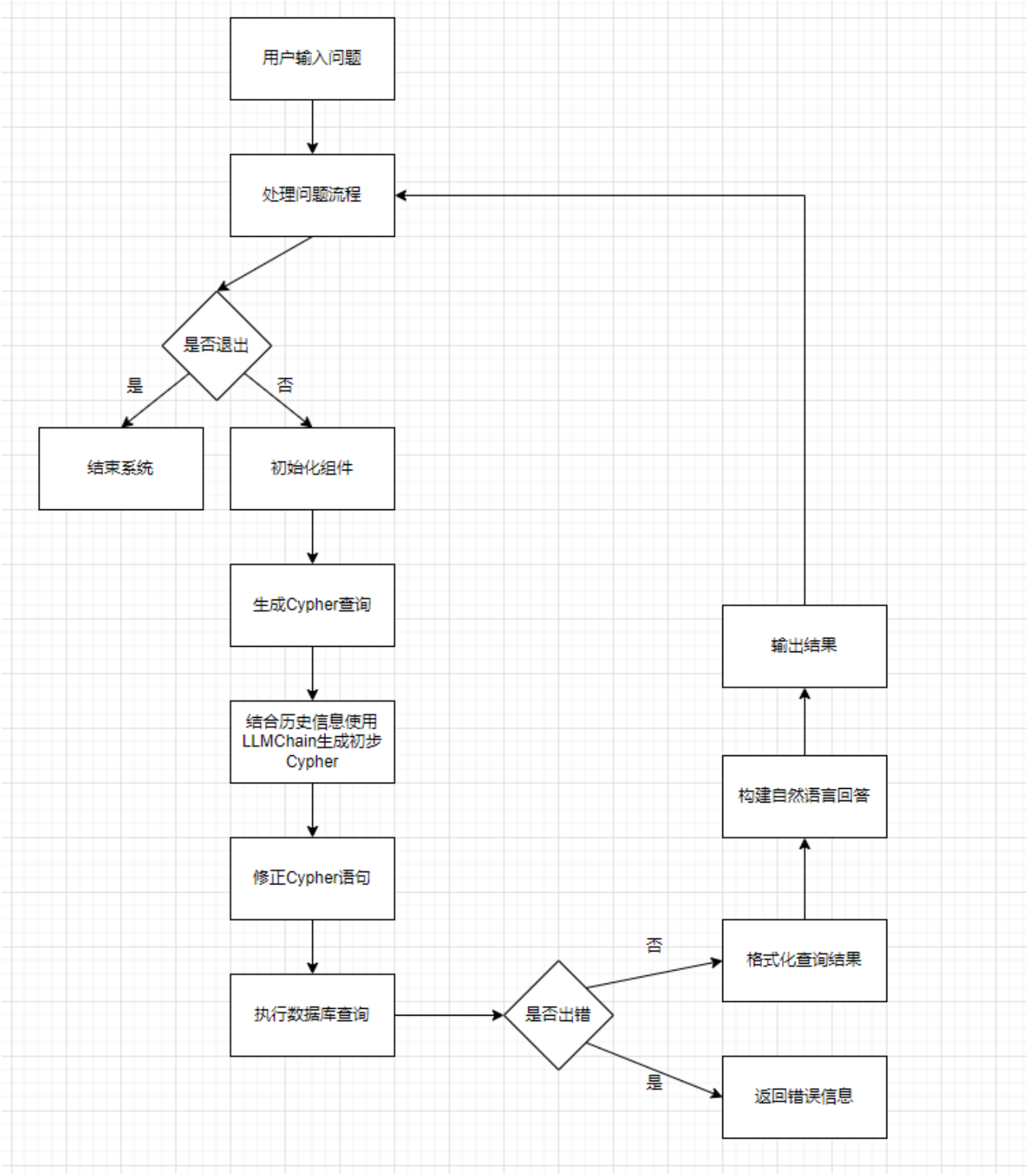
3.3.5 模块设计

-



- ***模块分工***：
  1. ***Cypher生成**模块*** （LangChain）：
     - 清洗生成Cypher查询，使查询语句更准确·。
  2. ***LangChain-RAG 引擎***：

- 结合用户问题生成 Cypher 查询，检索知识图谱。



1. ***前端问答引擎*** （JavaScript）：
    - 解析用户输入，支持即时关键词匹配与结果预览。
2. ***动态展示模块*** ：
    - 使用 Vue设计前端页面实现答案的展示。

3.3.6 界面设计

- ***核心界面*** ：

- ***功能区域*** ：
  1. ***搜索框*** ：
     - 占位符提示：“请输入您的问题 ...”。
  2. ***对话历史区*** ：
     - 按会话流展示用户问题与系统答案。
     - 支持点击历史问题重新触发查询。
  3. ***用户登录区*** ：
     - 显示用户ID，设置用户名
- ***设计原则*** ：
  1. ***简洁性*** ：减少视觉干扰，聚焦问答内容。

3.4掌上博物馆子系统

3.4.1项目概述

3.4.1.1 项目背景与目标

掌上博物馆是一款基于HarmonyOS平台开发的移动应用，旨在为用户提供便捷的博物馆文物浏览、搜索和互动体验。该项目通过数字技术将博物馆文物资源进行数字化呈现，让用户无需亲临现场即可欣赏珍贵文物，同时提供丰富的交互功能和个性化服务。

主要目标包括：

- 提供文物高清图像和3D模型浏览功能
- 实现文物信息的智能搜索与图像识别搜索
- 支持用户评论、收藏等社交互动功能

- 提供个性化推荐和用户管理
- 确保系统在多用户并发访问下的稳定性能

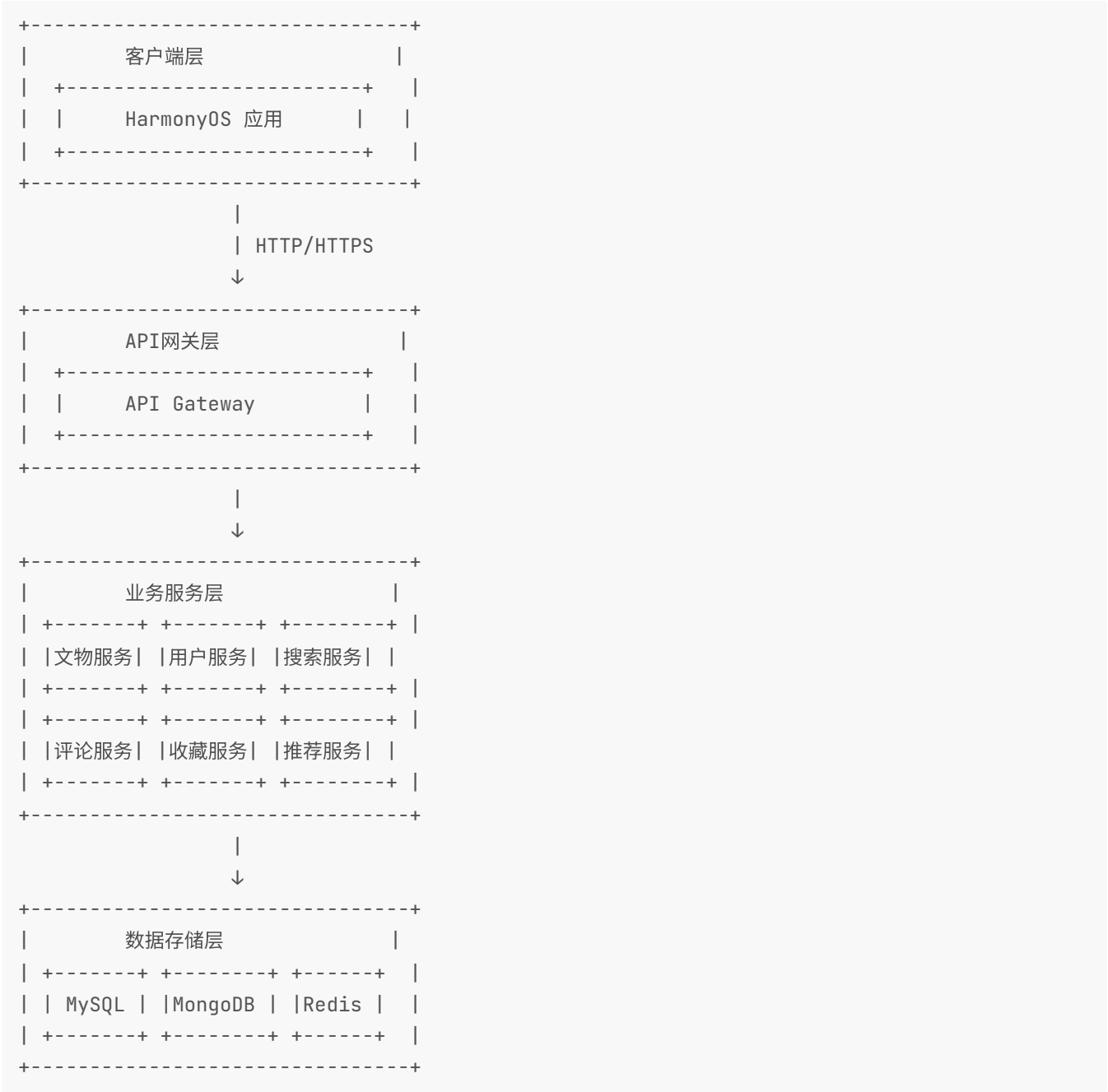
3.4.1.2 适用范围

本设计报告适用于掌上博物馆项目的开发团队、测试团队和维护人员，包括前端开发人员、后端开发人员、UI设计师、测试工程师等。

3.4.2 系统架构设计

3.4.2.1 总体架构

掌上博物馆采用前后端分离的架构，前端使用HarmonyOS原生应用开发框架，后端采用微服务架构。系统整体架构如下图所示：



3.4.2.2 技术架构

3.4.2.2.1 前端架构

前端采用HarmonyOS开发框架，主要技术栈包括：

- 开发语言：ArkTS
- UI框架：HarmonyOS原生UI组件
- 架构模式：MVVM（Model-View-ViewModel）
- 状态管理：AppStorage
- 网络请求：HTTP请求模块
- 图像处理：HMS图像分析服务

3.4.2.2.2 后端架构

后端采用微服务架构，主要技术栈包括：

- 开发语言：Node.js
- Web框架：Express
- 数据库：MySQL（关系型数据）+ MongoDB（非关系型数据）
- 缓存：Redis
- 搜索引擎：Elasticsearch
- 图像识别：TensorFlow模型
- API文档：Swagger

3.4.3 功能模块设计

3.4.3.1 功能模块划分

根据业务需求，掌上博物馆系统划分为以下功能模块：

1. **首页模块**：展示推荐文物、热门展览等内容
2. **文物浏览模块**：按照分类、朝代、材质等维度浏览文物
3. **文物详情模块**：展示文物详细信息、高清图像和3D模型
4. **搜索模块**：支持文本搜索和语义搜索
5. **图像搜索模块**：通过上传图片识别相似文物
6. **用户管理模块**：用户注册、登录、个人信息管理
7. **评论模块**：文物评论和点赞
8. **收藏模块**：用户收藏文物管理
9. **个人中心模块**：用户个性化内容和历史记录

3.4.3.2 核心功能流程

3.4.3.2.1 文物浏览流程

```
用户 → 选择分类 → 浏览文物列表 → 查看文物详情
                → 筛选条件 → 浏览筛选结果 → 查看文物详情
```

3.4.3.2.2 图像搜索流程

```
用户 → 上传图片 → 系统处理图像 → 特征提取 → 相似度匹配 → 返回相似文物列表 → 用户查看结果
```

3.4.4 数据库设计

3.4.4.1 关系模型设计

文物表(artifacts)

字段名	类型	说明
id	INT	主键，自增
name	VARCHAR(100)	文物名称
dynasty	VARCHAR(50)	朝代
category	VARCHAR(50)	分类
material	VARCHAR(50)	材质
description	TEXT	文物描述
era	VARCHAR(100)	年代
unearthed_site	VARCHAR(200)	出土地点
museum	VARCHAR(100)	所属博物馆
collection_no	VARCHAR(50)	馆藏编号
create_time	TIMESTAMP	创建时间
update_time	TIMESTAMP	更新时间

文物图像表(artifact_images)

字段名	类型	说明
id	INT	主键，自增
artifact_id	INT	外键，关联文物表
image_url	VARCHAR(255)	图像URL
image_type	TINYINT	图像类型(1:封面,2:高清图,3:细节图)
sequence	INT	图像顺序
create_time	TIMESTAMP	创建时间

用户表(users)

字段名	类型	说明
id	INT	主键，自增
username	VARCHAR(50)	用户名
password	VARCHAR(100)	密码(加密存储)
nickname	VARCHAR(50)	昵称
avatar_url	VARCHAR(255)	头像URL
phone	VARCHAR(20)	手机号
email	VARCHAR(100)	邮箱
status	TINYINT	状态(0:禁用,1:正常)
register_time	TIMESTAMP	注册时间
last_login_time	TIMESTAMP	最后登录时间

评论表(comments)

字段名	类型	说明
id	INT	主键，自增
artifact_id	INT	外键，关联文物表
user_id	INT	外键，关联用户表
content	TEXT	评论内容
like_count	INT	点赞数
create_time	TIMESTAMP	创建时间
update_time	TIMESTAMP	更新时间

收藏表(collections)

字段名	类型	说明
id	INT	主键，自增
user_id	INT	外键，关联用户表
artifact_id	INT	外键，关联文物表
create_time	TIMESTAMP	收藏时间

3.4.4.2 文档模型设计(MongoDB)

文物详情(artifact_details)

JSON

```
{
```

```
  "_id": "ObjectId",
  "artifact_id": 123,
  "name": "青花瓷瓶",
  "detailed_info": "详细的文物背景和历史信息...",
  "specifications": {
    "height": 30.5,
    "width": 15.2,
    "weight": 1.5,
    "volume": 2.0
  },
  "cultural_significance": "文化价值和历史意义...",
  "preservation_status": "保存状态描述...",
  "research_findings": "相关研究发现...",
  "3d_model_url": "https://example.com/models/123.glb",
  "related_artifacts": [124, 125, 126],
  "tags": ["瓷器", "明代", "宫廷用品"],
  "metadata": {
    "created_at": "ISODate",
    "updated_at": "ISODate"
  }
}
```

3.4.5 接口设计

3.4.5.1 API接口规范

掌上博物馆系统API遵循RESTful设计风格，主要接口规范如下：

- 使用HTTP动词表示操作：GET(查询)、POST(新建)、PUT(更新)、DELETE(删除)
- 接口路径遵循资源层次结构：/api/v1/{resource}/{id}
- 统一返回格式：

  JSON

  ```
  {
    "code": 200,      // 状态码
    "message": "操作成功", // 状态信息
    "data": {}        // 返回数据
  }
  ```

- 状态码说明：
  - 200：成功
  - 400：请求参数错误
  - 401：未授权
  - 403：权限不足
  - 404：资源不存在
  - 500：服务器内部错误

3.4.5.2 核心接口定义

文物相关接口

接口名称	请求方式	接口路径	说明
获取文物列表	GET	/api/v1/artifacts	分页获取文物列表
获取文物详情	GET	/api/v1/artifacts/{id}	获取指定文物详情
按条件筛选文物	GET	/api/v1/artifacts/filter	按条件筛选文物
获取文物评论	GET	/api/v1/artifacts/{id}/comments	获取文物评论
图像搜索文物	POST	/api/v1/artifacts/image-search	以图搜图

用户相关接口

接口名称	请求方式	接口路径	说明
用户注册	POST	/api/v1/users/register	用户注册
用户登录	POST	/api/v1/users/login	用户登录
获取用户信息	GET	/api/v1/users/{id}	获取用户信息
更新用户信息	PUT	/api/v1/users/{id}	更新用户信息
获取用户收藏	GET	/api/v1/users/{id}/collections	获取用户收藏列表

3.4.6 UI设计

3.4.6.1 整体风格

掌上博物馆应用整体采用清新、典雅的设计风格，以米色、青色作为主色调，突出文物的历史感和艺术性。界面设计遵循HarmonyOS设计规范，强调简洁性和一致性。

3.4.6.2 主要界面设计

1. 首页
   :
   - 顶部搜索栏
   - 轮播图展示精选文物
   - 分类导航
   - 推荐文物瀑布流
2. 文物详情页
   :
   - 高清图片轮播
   - 3D模型交互区域
   - 文物基本信息卡片
   - 详细描述区域
   - 相关文物推荐

    - 评论区域
3. 图像搜索页：
    - 图片上传区域
    - 相机拍摄入口
    - 搜索结果列表
4. 个人中心：
    - 用户信息概览
    - 收藏列表
    - 浏览历史
    - 系统设置

3.4.7 安全设计

3.4.7.1 身份认证与授权

- 采用JWT(JSON Web Token)进行用户认证
- 实现基于角色的访问控制(RBAC)
- 敏感API接口加入权限验证
- 密码加密存储(使用bcrypt算法)

3.4.7.2 数据安全

- 所有通信采用HTTPS加密传输
- 敏感信息(如用户密码)单向加密存储
- 定期数据备份机制
- 数据访问审计日志

3.4.7.3 应用安全

- 防SQL注入措施：使用参数化查询
- 防XSS攻击：输入输出过滤
- 防CSRF攻击：使用CSRF Token
- 限流/防DDoS机制：API网关层实现请求限制

3.4.8 性能设计

3.4.8.1 性能目标

- 首页加载时间：<2秒
- 文物详情页加载：<3秒
- 图像搜索响应时间：<5秒
- 系统支持并发用户数：>100
- API接口响应时间：<1秒

3.4.8.2 性能优化策略

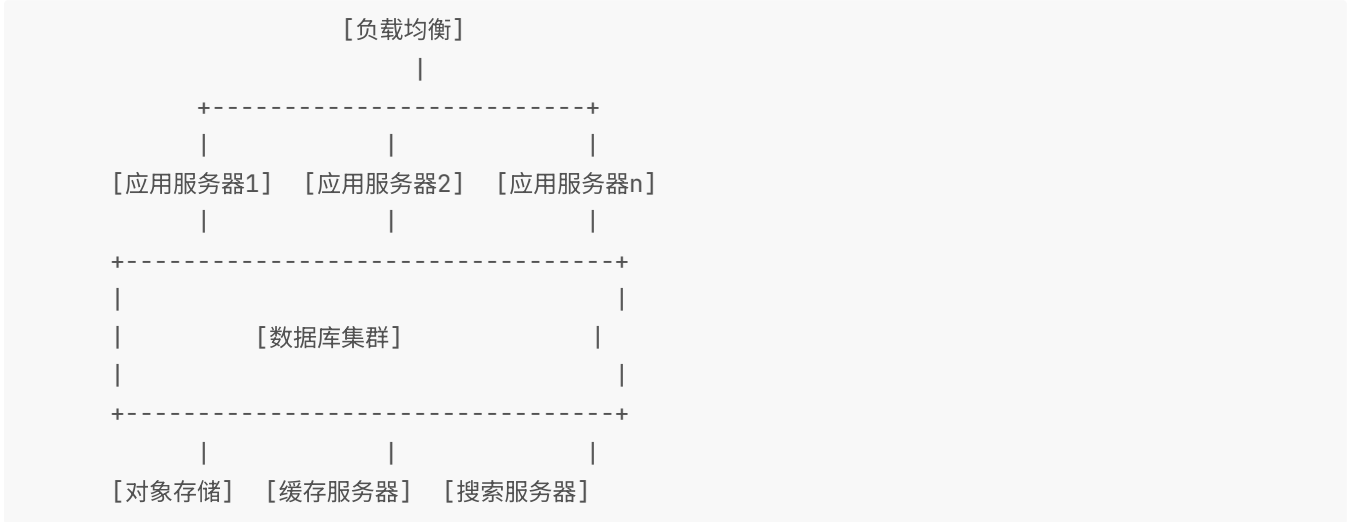
- CDN加速静态资源(图片、3D模型等)
- Redis缓存热门文物数据
- 数据库索引优化
- 延迟加载(懒加载)大图片和3D模型
- API响应数据压缩
- 前端资源压缩与合并

3.4.9 部署方案

3.4.9.1 环境配置

- 开发环境
  :
  - 前端: DevEco Studio 3.1.0
  - 后端: Node.js v16.x
  - 数据库: MySQL 8.0, MongoDB 5.0
- 测试环境
  :
  - 服务器: 腾讯云 2核4G
  - 操作系统: CentOS 8
  - 容器化: Docker, Docker Compose
- 生产环境
  :
  - 服务器集群: 阿里云ECS
  - 负载均衡: 阿里云SLB
  - 对象存储: 阿里云OSS

3.4.9.2 部署架构



3.4.10 项目规划

3.4.10.1 开发周期

总计划周期：3个月

阶段	时间	主要任务
需求分析	2周	用户需求调研、功能需求分析、系统需求规格说明书编写
概要设计	2周	系统架构设计、数据库设计、接口设计、原型设计
详细设计	2周	UI设计、详细技术方案制定
开发实现	6周	前端开发、后端开发、数据库实现、接口联调
测试	2周	单元测试、集成测试、系统测试、性能测试
部署上线	1周	环境部署、试运行、正式上线

3.4.10.2 里程碑计划

1. **M1**：需求分析完成，确定功能范围（T+2周）
2. **M2**：系统原型设计完成（T+4周）
3. **M3**：核心功能开发完成（T+8周）
4. **M4**：系统测试完成（T+10周）
5. **M5**：系统正式上线（T+12周）

3.4.11 风险评估与对策

3.4.11.1 主要风险

风险	概率	影响	应对策略
需求变更频繁	中	高	采用敏捷开发方法，增量式迭代开发
文物图像数据量大	高	中	采用分布式存储和CDN加速策略
图像识别精度不足	中	高	引入多种识别算法，持续优化模型
系统性能瓶颈	中	高	前期进行架构评审，定期性能测试
用户体验不佳	低	高	进行用户测试，收集反馈持续改进

3.4.11.2 应对措施

- 建立畅通的沟通渠道，及时响应需求变更
- 制定详细的测试计划，覆盖功能和性能测试
- 采用可扩展的架构设计，预留系统扩展空间
- 定期进行代码审查和重构，保证代码质量
- 制定应急预案，确保系统出现问题时能快速响应

3.4.12 附录

3.4.12.1 术语表

术语	描述
ArkTS	华为鸿蒙系统的应用开发语言
HMS	Harmony Mobile Services，华为移动服务
MVVM	Model-View-ViewModel，一种软件架构模式
RESTful	一种软件架构风格，用于设计网络应用程序API
JWT	JSON Web Token，一种开放标准，用于在各方之间安全地传输信息

3.4.12.2 参考文献

1. 《HarmonyOS应用开发指南》，华为开发者联盟，2023
2. 《RESTful Web Services》，Leonard Richardson，Sam Ruby著，2007
3. 《MongoDB: The Definitive Guide》，Kristina Chodorow著，2019
4. 《Mobile Design Pattern Gallery》，Theresa Neil著，2014

3.5 后台管理子系统

3.5.1 系统功能简介

海外文物后台管理系统是基于Plotly Dash框架开发的综合性管理平台，为海外文物展示平台提供全面的后台管理服务。系统面向博物馆管理员、文物研究人员和系统运维人员，提供文物数字化管理、用户权限控制、内容审核管理等核心功能。

3.5.2 主要功能模块

3.5.2.1 用户管理系统

- **Web端用户管理**：基于 `SysUser` 模型管理后台管理员账户，支持用户信息的增删改查
- **APP端用户管理**：基于 `MyAppUser` 模型管理移动端注册用户，监控用户活跃度和行为数据
- **权限管理**：基于RBAC模型的角色权限分配，通过 `SysRole` 和 `SysRoleAccessMeta` 实现细粒度权限控制
- **用户组管理**：通过 `SysGroup` 按部门或职能划分用户组，便于批量权限管理

3.5.2.2 文物信息管理

- **文物入库管理**：基于 `SysAntique` 模型支持文物基本信息录入，包括图片、描述、分类等
- **库存管理**：实现文物的查找、查阅，支持图片预览功能
- **分类管理**：根据文物种类进行分类管理，便于检索和统计
- **文物展示**：支持文物信息的展示和状态管理

3.5.2.3 评论审核系统

- **评论审核**：通过评论管理模块对用户针对文物的评论进行人工审核和通过操作
- **帖子管理**：通过帖子管理模块对APP端用户发布的帖子进行审核和撤销
- **内容过滤**：支持关键词过滤和自动化审核机制
- **审核记录**：完整的审核日志，支持审核决策追溯

3.5.2.4 数据可视化看板

- **访问统计**：通过工作台模块展示浏览量、评论量、访问量等关键指标
- **数据图表**：通过Feffery Antd Charts组件以折线图、柱状图、饼图等形式展示数据趋势
- **实时监控**：实时更新系统运行状态和用户活跃度
- **统计分析**：支持评论状态分布、访问趋势分析等功能

3.5.2.5 任务调度系统

- **任务管理**：基于任务管理模块支持定时任务、周期任务和监听任务
- **数据采集**：支持通过任务系统进行数据采集和抽取
- **消息推送**：集成多种通知渠道，支持任务执行状态通知
- **脚本执行**：支持Shell、Bat、Python等多种脚本类型

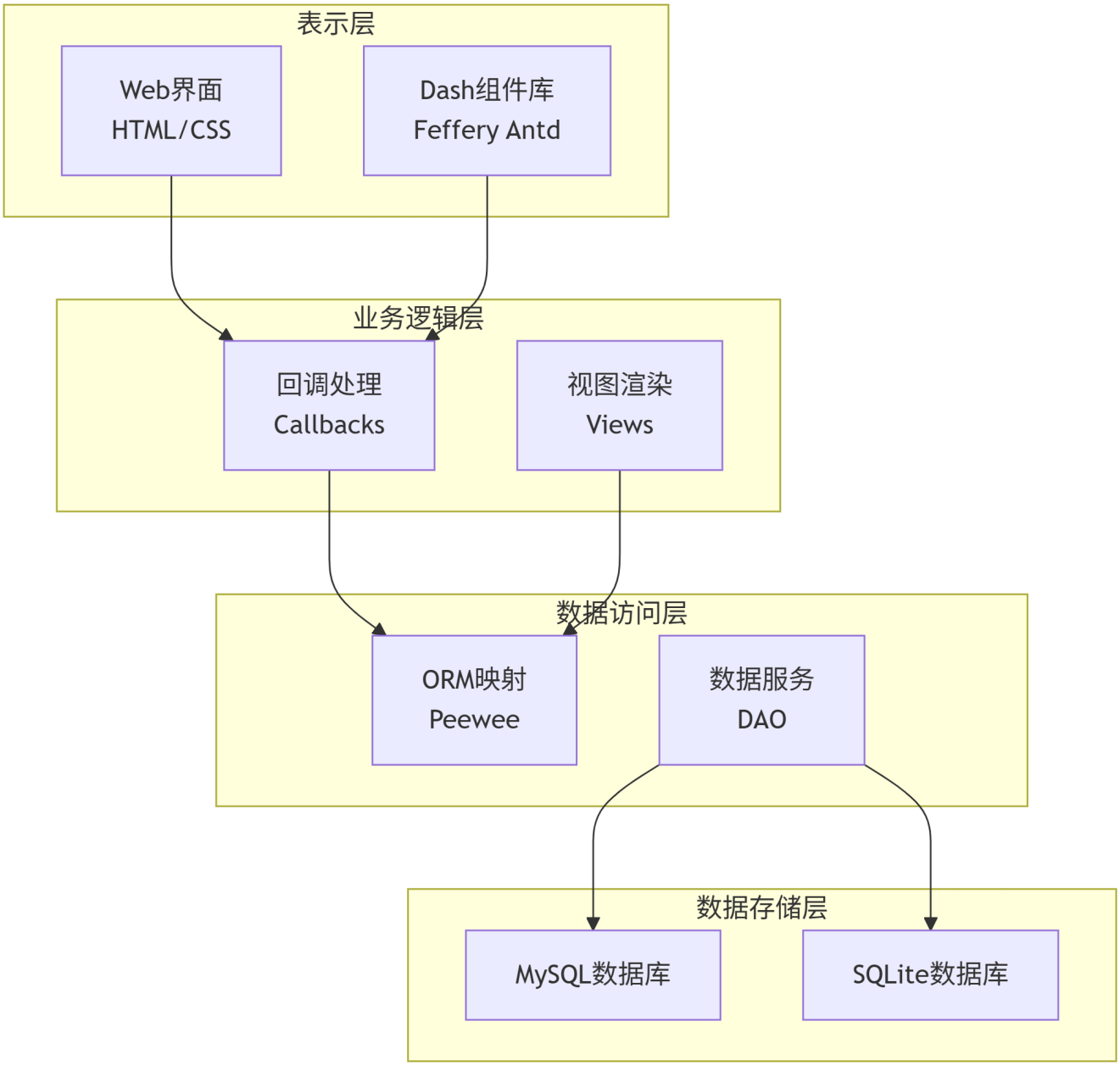
3.5.2.6 系统管理

- **数据库维护**：通过数据库备份模块支持一键数据备份和恢复操作
- **日志管理**：系统操作日志记录和查询
- **配置管理**：通过配置文件进行系统参数配置和环境设置

3.5.3 体系结构设计

3.5.3.1 整体架构

系统采用基于Plotly Dash的类MVC架构模式，通过回调机制实现前后端交互。整体架构分为表示层、业务逻辑层、数据访问层和数据存储层。



3.5.3.2 技术架构

前端技术栈

- **Plotly Dash**：基于React.js和Flask的Python Web框架
- **Feffery Antd Components**：基于Ant Design的UI组件库
- **Feffery Utils Components**：工具类组件库
- **Feffery Antd Charts**：数据可视化图表组件库
- **HTML5/CSS3**：现代Web标准技术

后端技术栈

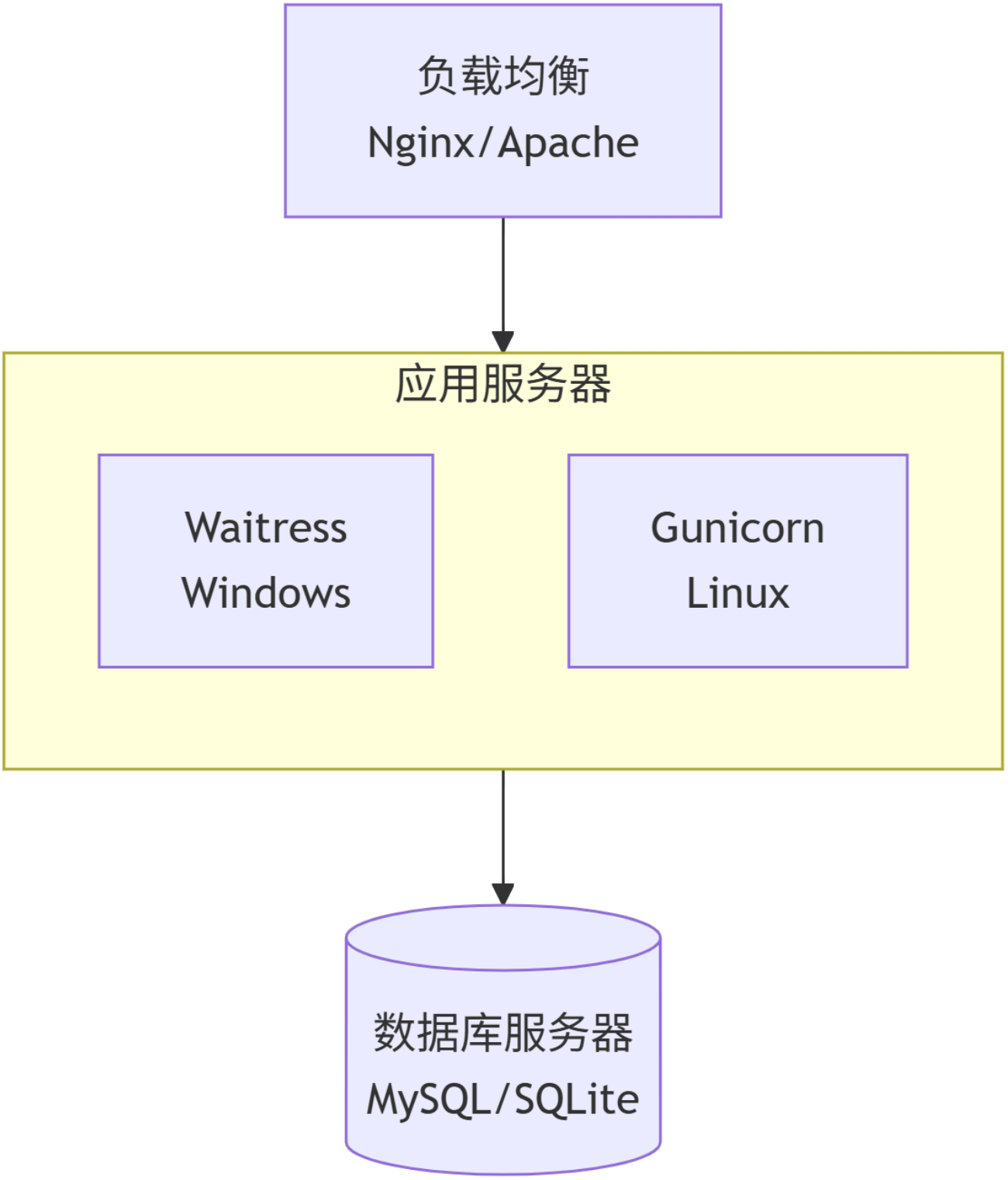
- **Python 3.8+**：主要开发语言
- **Flask**：轻量级Web框架（Dash底层）
- **Peewee ORM**：轻量级数据库操作框架
- **APScheduler**：任务调度框架

- **Waitress/Gunicorn**：WSGI服务器

数据存储

- **MySQL 5.7+**：主要关系型数据库
- **SQLite 3.0+**：轻量级数据库选项
- **文件系统**：静态资源和备份文件存储

3.5.3.3 部署架构



3.5.4 接口设计

3.5.4.1 内部接口设计

Dash回调接口

系统基于Dash的回调机制进行前后端交互，主要回调接口基于代码库实现：

```
# 文物表格初始化回调（基于antique_order_c.py）
@app.callback(
    Output('antique-table-container', 'children'),
    [Input('antique-init-timeout', 'timeoutCount'),
     Input('antique-status-filter', 'value')],
    prevent_initial_call=True,
)
def init_table(timeoutCount, status_filter):
    """页面加载时初始化渲染文物信息表格"""

# 评论审核回调（基于post_comment_c.py）
@app.callback(
    Output('comment-table-container', 'children'),
    [Input('comment-init-timeout', 'timeoutCount'),
     Input('comment-status-filter', 'value')],
    prevent_initial_call=True,
)
def init_table(timeoutCount, status_filter):
    """页面加载时初始化渲染表格"""

# 任务管理回调（基于task_mgmt_c.py）
@app.callback(
    Output('task-mgmt-table-container', 'children'),
    Input('task-mgmt-init-timeout', 'timeoutCount'),
    prevent_initial_call=True,
)
def init_table(timeoutCount):
    """页面加载时初始化渲染任务管理表格"""
```

数据库接口

基于Peewee ORM的数据库操作接口，参考现有DAO模式：

```
# 文物管理接口（基于dao_antique_order.py）
def insert_antique(title, artist, dynasty, classification, credit, description, materials,
dimensions, imgurl):
    """新增文物信息"""

def get_antique_info():
    """获取文物信息列表"""

# APP用户管理接口（基于dao_user.py）
def get_app_user_info(exclude_frozen=True):
    """获取APP用户信息"""

def update_app_user_status(account, is_frozen):
    """更新APP用户状态"""
```

3.5.4.2 外部接口设计

任务调度接口

基于APScheduler的任务调度系统接口：

```
# 任务执行接口
POST /task/execute
Content-Type: application/json
{
    "task_id": "string",
    "script_content": "string",
    "task_type": "interval|cron|listen"
}

# 任务状态查询接口
GET /task/status/{task_id}

# 任务日志接口
GET /task/logs/{task_id}
```

数据备份接口

支持数据库备份和恢复的接口：

```
# 备份创建接口
POST /backup/create
Content-Type: application/json
{
    "backup_type": "manual|auto"
}

# 备份恢复接口
POST /backup/restore
Content-Type: application/json
{
    "backup_file": "string"
}
```

3.5.5 数据设计

3.5.5.1 现有数据库表结构

系统基于现有数据库表结构，主要包括以下核心表：

用户相关表（基于table_user.py）

```
-- 后台用户表
CREATE TABLE sys_user (
    user_id INT PRIMARY KEY AUTO_INCREMENT,
    user_name VARCHAR(255) NOT NULL,
    user_nick_name VARCHAR(255),
    user_avatar VARCHAR(255),
    user_password VARCHAR(255) NOT NULL,
    user_email VARCHAR(255),
    is_frozen TINYINT DEFAULT 0,
    is_admin TINYINT DEFAULT 0
);

-- APP端用户表
CREATE TABLE my_app_user (
    account VARCHAR(255) PRIMARY KEY,
    password VARCHAR(255),
    email VARCHAR(255),
    is_frozen TINYINT DEFAULT 0,
    is_admin TINYINT DEFAULT 0
);

-- 管理员表
CREATE TABLE admin_info (
    admin_id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT,
    admin_nick_name VARCHAR(255)
);
```

权限管理表

```
-- 角色表
CREATE TABLE sys_role (
    role_id INT PRIMARY KEY AUTO_INCREMENT,
    role_name VARCHAR(255) NOT NULL,
    role_key VARCHAR(255),
    sort_num INT,
    role_status TINYINT DEFAULT 1
);

-- 权限元数据表
CREATE TABLE sys_access_meta (
    access_meta_id INT PRIMARY KEY AUTO_INCREMENT,
    access_meta_name VARCHAR(255) NOT NULL,
    access_meta_key VARCHAR(255)
);

-- 角色权限关联表
CREATE TABLE sys_role_access_meta (
    role_access_meta_id INT PRIMARY KEY AUTO_INCREMENT,
    role_id INT,
    access_meta_id INT,
    FOREIGN KEY (role_id) REFERENCES sys_role(role_id),
    FOREIGN KEY (access_meta_id) REFERENCES sys_access_meta(access_meta_id)
);
```

文物相关表（基于现有业务）

```
-- 文物表（基于SysAntique模型）
CREATE TABLE sys_antique (
    id INT PRIMARY KEY AUTO_INCREMENT,
    Title VARCHAR(255),
    Artist VARCHAR(255),
    Dynasty VARCHAR(255),
    Classifications VARCHAR(255),
    Credit VARCHAR(255),
    Description TEXT,
    Materials VARCHAR(255),
    Dimensions VARCHAR(255),
    ImgUrl VARCHAR(255),
    ImgPath VARCHAR(255)
);
```

系统操作日志表

```
-- 系统操作日志表
CREATE TABLE sys_operation_log (
    log_id BIGINT PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    user_name VARCHAR(255),
```

```
    operation_type VARCHAR(100) NOT NULL COMMENT '操作类型:
CREATE/UPDATE/DELETE/LOGIN/LOGOUT',
    operation_module VARCHAR(100) NOT NULL COMMENT '操作模块: USER/ANTIQUE/COMMENT/TASK等',
    operation_desc VARCHAR(500) COMMENT '操作描述',
    target_id VARCHAR(100) COMMENT '操作目标ID',
    target_type VARCHAR(100) COMMENT '操作目标类型',
    operation_data JSON COMMENT '操作数据快照',
    operation_result VARCHAR(50) DEFAULT 'SUCCESS' COMMENT '操作结果: SUCCESS/FAILED',
    error_msg TEXT COMMENT '错误信息',
    ip_address VARCHAR(45) COMMENT 'IP地址',
    user_agent TEXT COMMENT '用户代理',
    operation_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP COMMENT '操作时间',
    duration_ms INT COMMENT '操作耗时(毫秒)',
    INDEX idx_user_id (user_id),
    INDEX idx_operation_type (operation_type),
    INDEX idx_operation_module (operation_module),
    INDEX idx_operation_time (operation_time)
) COMMENT='系统操作日志表';
```

审核操作记录表

```
-- 审核操作记录表
CREATE TABLE sys_review_log (
    review_log_id BIGINT PRIMARY KEY AUTO_INCREMENT,
    reviewer_id INT NOT NULL COMMENT '审核员ID',
    reviewer_name VARCHAR(255) COMMENT '审核员姓名',
    review_type VARCHAR(50) NOT NULL COMMENT '审核类型: COMMENT/POST/ANTIQUE',
    target_id INT NOT NULL COMMENT '审核目标ID',
    target_content TEXT COMMENT '审核内容快照',
    review_action VARCHAR(50) NOT NULL COMMENT '审核动作: APPROVE/REJECT/DELETE',
    review_reason VARCHAR(500) COMMENT '审核理由',
    review_note TEXT COMMENT '审核备注',
    old_status VARCHAR(50) COMMENT '原状态',
    new_status VARCHAR(50) COMMENT '新状态',
    review_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP COMMENT '审核时间',
    auto_review TINYINT DEFAULT 0 COMMENT '是否自动审核',
    review_score DECIMAL(3,2) COMMENT '审核评分',
    INDEX idx_reviewer_id (reviewer_id),
    INDEX idx_review_type (review_type),
    INDEX idx_target_id (target_id),
    INDEX idx_review_time (review_time)
) COMMENT='审核操作记录表';
```

数据变更日志表

```
-- 数据变更日志表
CREATE TABLE sys_data_change_log (
    change_log_id BIGINT PRIMARY KEY AUTO_INCREMENT,
    table_name VARCHAR(100) NOT NULL COMMENT '表名',
    record_id VARCHAR(100) NOT NULL COMMENT '记录ID',
    change_type VARCHAR(50) NOT NULL COMMENT '变更类型: INSERT/UPDATE/DELETE',
```

```
    field_changes JSON COMMENT '字段变更详情',
    old_values JSON COMMENT '变更前数据',
    new_values JSON COMMENT '变更后数据',
    operator_id INT COMMENT '操作人ID',
    operator_name VARCHAR(255) COMMENT '操作人姓名',
    change_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP COMMENT '变更时间',
    change_reason VARCHAR(500) COMMENT '变更原因',
    business_id VARCHAR(100) COMMENT '业务ID',
    INDEX idx_table_name (table_name),
    INDEX idx_record_id (record_id),
    INDEX idx_change_type (change_type),
    INDEX idx_operator_id (operator_id),
    INDEX idx_change_time (change_time)
) COMMENT='数据变更日志表';
```

系统访问日志表

```
-- 系统访问日志表
CREATE TABLE sys_access_log (
    access_log_id BIGINT PRIMARY KEY AUTO_INCREMENT,
    user_id INT COMMENT '用户ID',
    user_name VARCHAR(255) COMMENT '用户名',
    session_id VARCHAR(255) COMMENT '会话ID',
    access_url VARCHAR(500) COMMENT '访问URL',
    access_method VARCHAR(20) COMMENT '请求方法',
    access_params TEXT COMMENT '请求参数',
    response_status INT COMMENT '响应状态码',
    response_time INT COMMENT '响应时间(毫秒)',
    ip_address VARCHAR(45) COMMENT 'IP地址',
    user_agent TEXT COMMENT '用户代理',
    referer VARCHAR(500) COMMENT '来源页面',
    access_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP COMMENT '访问时间',
    INDEX idx_user_id (user_id),
    INDEX idx_ip_address (ip_address),
    INDEX idx_access_time (access_time),
    INDEX idx_response_status (response_status)
) COMMENT='系统访问日志表';
```

3.5.6 模块设计

3.5.6.1 核心模块结构(基于实际代码库)

```
src/
├── app.py                          # 主应用入口
├── app_apscheduler.py              # 任务调度服务入口
├── config/                         # 配置模块
│   ├── __init__.py
│   └── dashgo.ini                  # 配置文件
├── database/                       # 数据库模块
│   └── sql_db/                     # SQL数据库
│       ├── conn.py                 # 数据库连接
```

```
│       ├── dao/                        # 数据访问对象
│       │   ├── dao_antique_order.py # 文物数据访问
│       │   ├── dao_post_comment.py  # 评论数据访问
│       │   └── dao_user.py          # 用户数据访问
│       └── entity/                     # 数据实体
│           └── table_user.py        # 用户表实体
├── dash_view/                          # 视图模块
│   └── application/
│       ├── access_/                    # 权限管理视图
│       │   └── app_mgmt.py          # APP用户管理
│       ├── dashboard_/                 # 仪表板视图
│       │   └── workbench.py         # 工作台
│       ├── message_/                   # 消息管理视图
│       └── task_/                      # 任务管理视图
│           └── db_backup_mgmt.py    # 数据库备份管理
├── dash_callback/                      # 回调模块
│   └── application/
│       ├── access_/                    # 权限管理回调
│       │   └── app_mgmt_c.py        # APP用户管理回调
│       ├── message_/                   # 消息管理回调
│       │   ├── antique_order_c.py   # 文物管理回调
│       │   ├── post_comment_c.py    # 评论管理回调
│       │   └── post_info_c.py       # 帖子管理回调
│       └── task_/                      # 任务管理回调
│           ├── task_mgmt_c.py       # 任务管理回调
│           └── db_backup_mgmt_c.py  # 数据库备份回调
└── common/                             # 通用模块
    └── utilities/                      # 工具类
        └── backup_utils.py          # 备份工具
```

3.5.6.2 主要模块功能

视图模块（dash_view）

基于实际代码结构，负责构建用户界面：

```python
# 工作台视图模块（workbench.py）
def render_content(menu_access: MenuAccess, **kwargs):
    """渲染工作台内容，包含统计卡片和图表"""
    # 统计卡片展示
    # 数据可视化图表
    # 实时数据监控

# APP用户管理视图（app_mgmt.py）
def render_content(menu_access: MenuAccess, **kwargs):
    """渲染APP用户管理界面"""
    # 用户表格展示
    # 操作按钮和模态框
    # 权限控制
```

回调模块（dash_callback）

处理用户交互和业务逻辑：

```
# 文物管理回调（antique_order_c.py）
@app.callback(
    Output('antique-table', 'data', allow_duplicate=True),
    Input('antique-table-add-modal', 'okCounts'),
    # 处理文物新增逻辑
)

# 任务管理回调（task_mgmt_c.py）
@app.callback(
    Output('task-mgmt-table-add-modal', 'children'),
    Input('task-mgmt-table-add-modal', 'visible'),
    # 处理任务管理模态框
)
```

数据访问模块（dao）

封装数据库操作逻辑：

```
# 文物数据访问（dao_antique_order.py）
def insert_antique(title, artist, dynasty, classification, credit, description, materials,
dimensions, imgurl):
    """插入文物信息"""

def get_antique_info():
    """获取文物信息列表"""

# 用户数据访问（dao_user.py）
def get_app_user_info(exclude_frozen=True):
    """获取APP用户信息"""
```

工具模块（utilities）

提供通用的工具函数：

```
# 备份工具（backup_utils.py）
def create_backup():
    """创建数据库备份"""

def restore_backup(backup_file):
    """恢复数据库备份"""
```

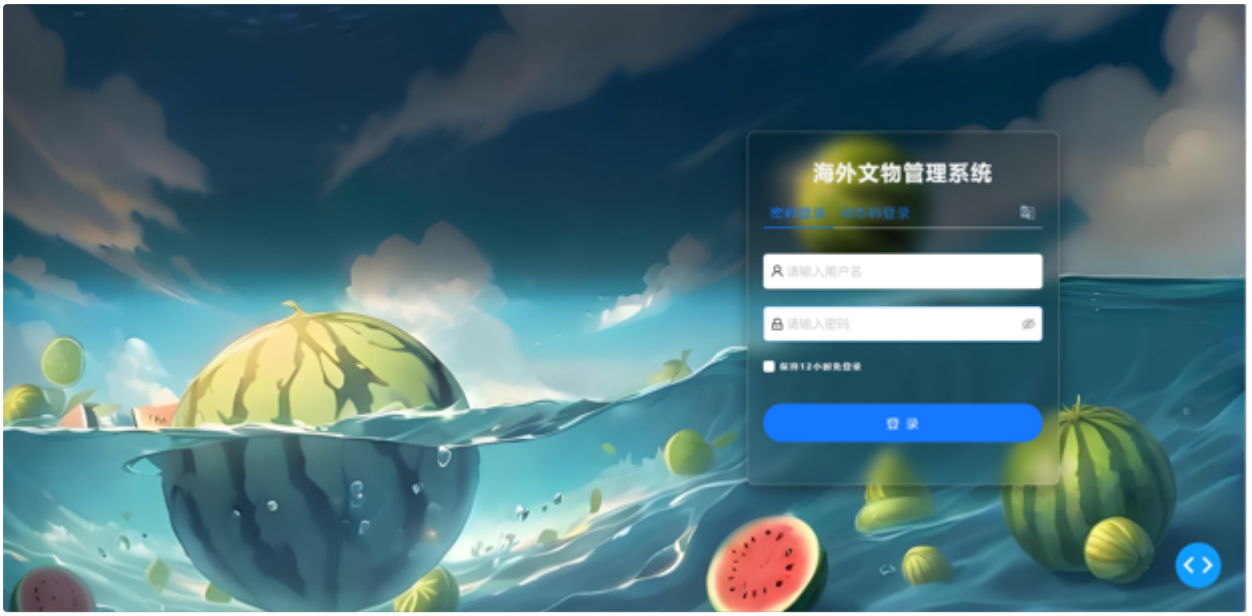
3.5.7 界面设计

3.5.7.1 界面设计原则

1. **一致性原则**：统一的视觉风格和交互模式
2. **易用性原则**：简洁明了的界面布局和操作流程
3. **响应式设计**：适配不同屏幕尺寸和设备
4. **可访问性**：支持键盘导航和屏幕阅读器

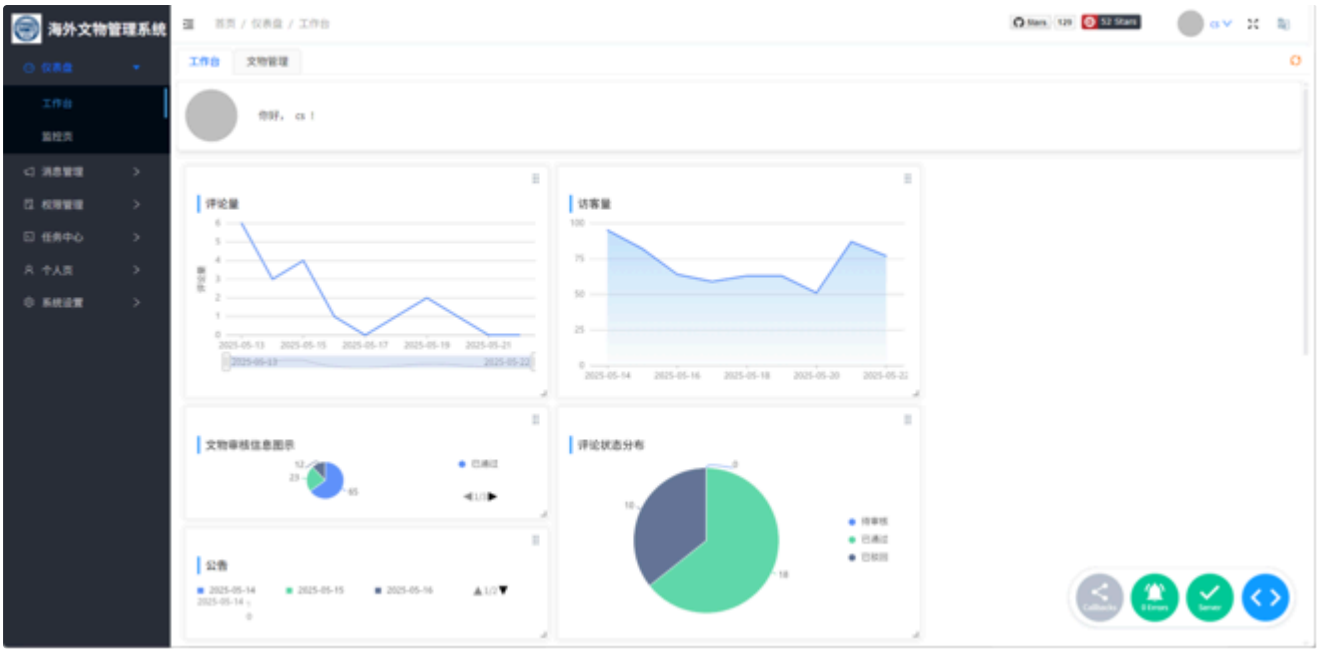
3.5.7.2 主要界面设计

登录界面



- **设计目标**：简洁、安全、易用
- **主要元素**：登录表单、记住密码选项
- **视觉风格**：现代化卡片式设计，品牌色彩搭配

数据看板界面



- **设计目标**：信息概览、数据可视化
- **主要元素**：统计卡片、图表组件、快捷操作
- **布局结构**：栅格布局，响应式设计
- **图表类型**：饼图、柱状图、双轴图等多种图表展示

用户管理界面

- **设计目标**：高效的用户管理操作
- **主要元素**：搜索栏、数据表格、操作按钮、模态框
- **交互设计**：批量操作、行内编辑、分页浏览
- **权限控制**：基于角色的功能访问控制

文物管理界面

- **设计目标**：直观的文物信息展示和管理
- **主要元素**：图片预览、详情面板、分类筛选
- **特色功能**：图片缩放、状态筛选、文物信息录入

审核管理界面

- **设计目标**：高效的内容审核流程
- **主要元素**：内容预览、审核操作、批量处理
- **状态指示**：不同审核状态的视觉区分
- **筛选功能**：按状态、时间等条件筛选

数据库备份界面

- **设计目标**：简单易用的备份管理
- **主要元素**：备份列表、操作按钮、状态指示
- **安全特性**：备份文件管理、一键恢复功能

3.5.7.3 UI组件库选择

系统采用Feffery生态组件库，基于Ant Design设计语言：

主要组件使用

- **布局组件**：fac.AntdLayout、fac.AntdSider、fac.AntdContent
- **导航组件**：fac.AntdMenu、fac.AntdBreadcrumb
- **数据展示**：fac.AntdTable、fac.AntdCard、fac.AntdStatistic
- **表单组件**：fac.AntdInput、fac.AntdSelect、fac.AntdDatePicker
- **反馈组件**：fac.AntdMessage、fac.AntdModal、fac.AntdNotification
- **图表组件**：fact.AntdPie、fact.AntdColumn、fact.AntdDualAxes

样式定制

```
/* 主题色彩定义 */
:root {
    --primary-color: #1890ff;
    --success-color: #52c41a;
    --warning-color: #faad14;
    --error-color: #f5222d;
}

/* 布局样式 */
.main-layout {
    min-height: 100vh;
}

.content-area {
    padding: 24px;
    background: #fff;
    margin: 24px;
    border-radius: 6px;
```

```
}
```

3.5.7.4 响应式设计

断点设置

- **超小屏幕**：< 576px（手机）
- **小屏幕**：576px - 768px（平板竖屏）
- **中等屏幕**：768px - 992px（平板横屏）
- **大屏幕**：992px - 1200px（桌面）
- **超大屏幕**：> 1200px（大桌面）

响应式策略

- **导航菜单**：小屏幕下折叠为抽屉式菜单
- **数据表格**：支持横向滚动和响应式列隐藏
- **卡片布局**：自适应栅格系统，动态调整列数
- **表单布局**：小屏幕下单列显示，大屏幕下多列显示

文档版本：v2.0
最后更新：2025年6月
维护团队：Team5海外文物后台管理系统开发组