

# MovieHub - 设计与实现报告

## 项目概述

**项目名称:** MovieHub - 电影与剧集聚合平台

**作者:** 姜政言

**学号:** 2353594

**课程:** 微服务架构

**完成时间:** 2025年10月

**GitHub:** <https://github.com/WilliamJiang1014/MicroServices-MovieHub>

MovieHub 是一个基于微服务架构的电影信息聚合平台，整合了多个外部数据源（TMDB、OMDb、TVMaze）和 AI 服务（通义千问），提供电影搜索、评分聚合、AI 智能推荐和观影清单管理等功能。本项目采用前后端分离架构，后端由 12 个独立微服务组成，前端使用 React 构建，支持 Docker 一键部署。

## 课程要求达成情况

本项目满足课程约束条件 **第1条**：

围绕单一特定主题或领域，从至少四个不同的提供方整合信息，并融合到单一应用中。

### 集成的数据提供方

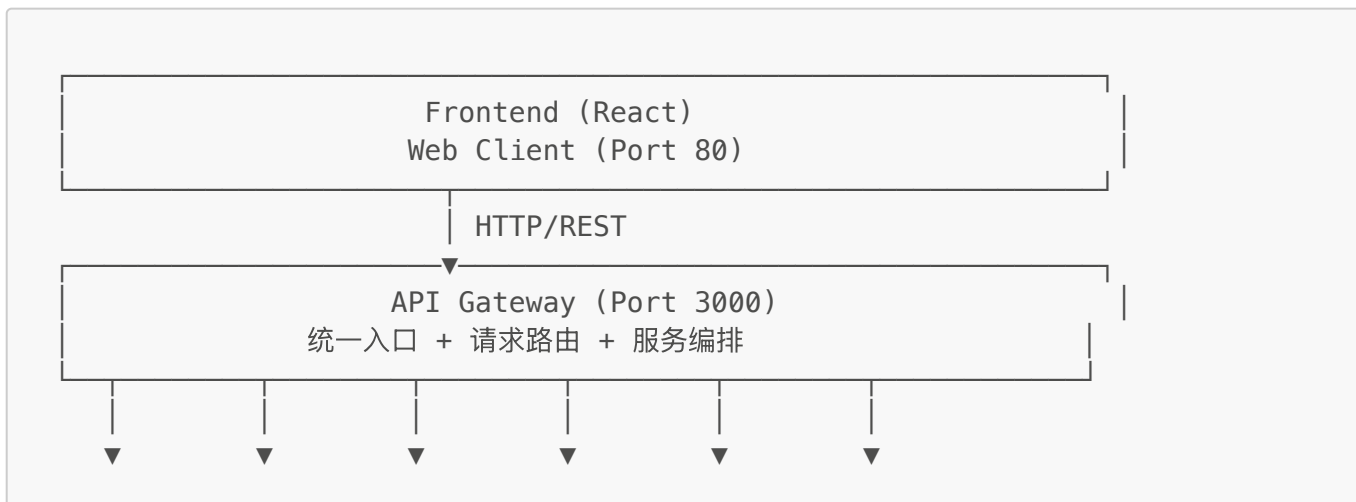
- TMDB (The Movie Database)** - 电影基础信息、评分、演员等
- OMDb (Open Movie Database)** - 补充电影信息和评分
- TVMaze** - 电视剧信息和评分
- 通义千问 LLM API** - AI 生成的影评摘要、推荐和亮点提取

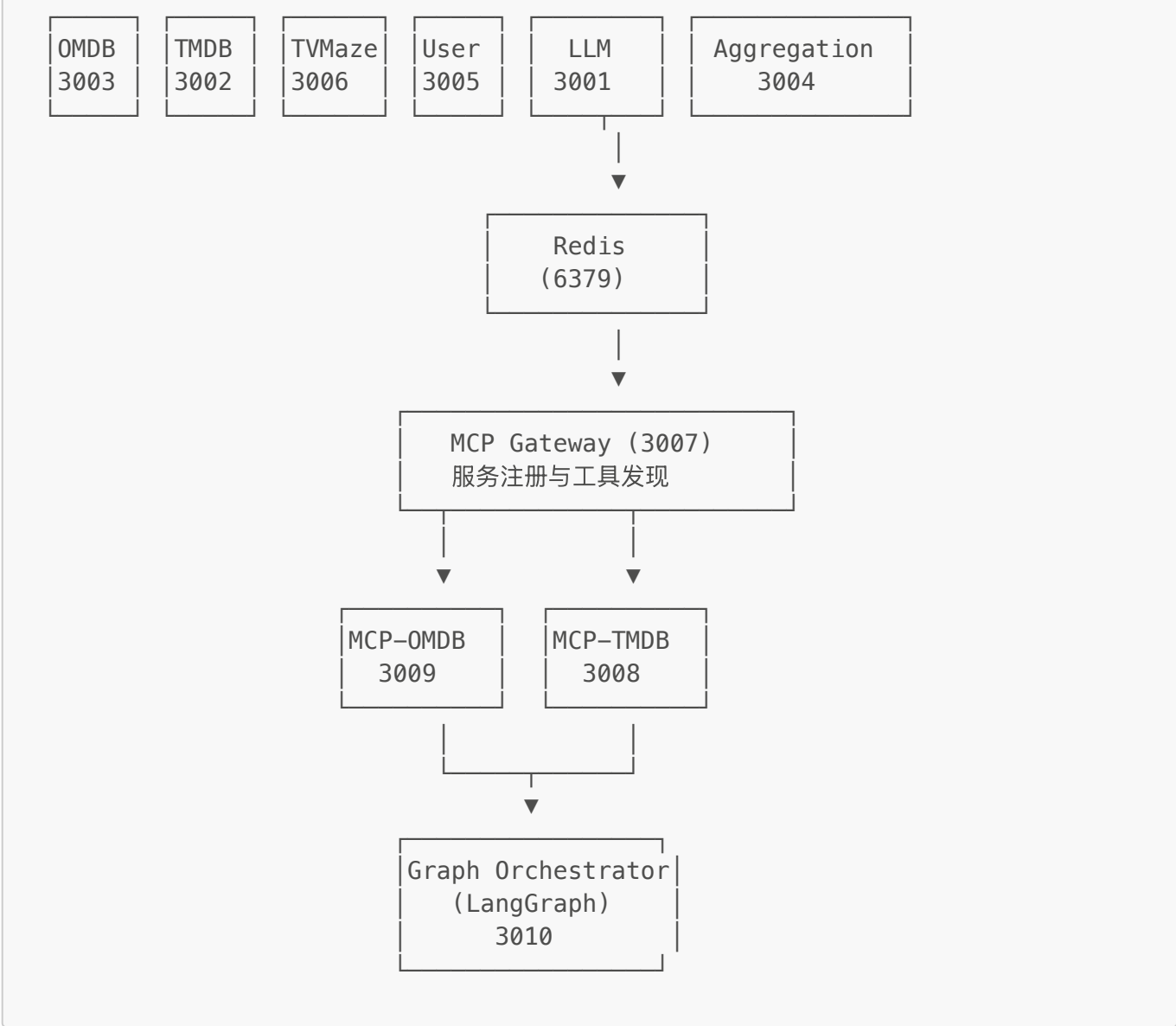
所有数据源都围绕"影视信息"这一单一领域，通过数据聚合服务融合成统一的数据模型，为用户提供全面的电影信息。

## 系统架构设计

### 整体架构

本项目采用**微服务架构**，将系统划分为多个独立的服务单元，每个服务负责特定的业务功能。





架构特点

- 1. **服务独立性:** 每个微服务独立部署，拥有独立的端口和配置
- 2. **松耦合:** 服务间通过 HTTP REST API 通信，降低耦合度
- 3. **可扩展性:** 可以轻松添加新的数据源或功能服务
- 4. **容错性:** 单个服务故障不影响整体系统运行
- 5. **技术多样性:** 不同服务可以使用不同的技术栈

核心服务设计

1. API Gateway (端口 3000)

**职责:** 统一的 API 入口，负责请求路由和服务编排

**主要功能:**

- 路由转发：将前端请求路由到对应的后端服务
- 服务编排：协调多个服务完成复杂业务逻辑
- 健康检查：监控各个服务的健康状态
- 错误处理：统一的错误处理和响应格式

**关键代码:**

```
// 搜索路由 - 调用聚合服务
app.get('/api/search', async (req, res) => {
  const { query, limit } = req.query;
  const response = await axios.get(
    `${AGGREGATION_URL}/search?query=${query}&limit=${limit}`
  );
  res.json(response.data);
});

// 电影详情 - 组合聚合服务和LLM服务
app.get('/api/movie/:movieId/summary', async (req, res) => {
  const movie = await axios.get(`${AGGREGATION_URL}/movie/${movieId}`);
  const summary = await axios.post(`${LLM_URL}/summarize`, { movie });
  res.json({ movie, summary });
});
```

## 2. Data Provider Services (端口 3002, 3003, 3006)

**职责:** 封装外部 API 调用，提供统一的数据接口

### TMDB Provider (3002)

- 调用 TMDB API 获取电影信息
- 数据格式标准化
- 错误处理和重试机制

### OMDb Provider (3003)

- 调用 OMDb API 获取电影信息
- 补充 IMDB 评分数据
- API 配额管理

### TVMaze Provider (3006)

- 调用 TVMaze API 获取电视剧信息
- 提供剧集详细信息
- 无需 API Key

**统一数据模型:**

```
interface Movie {
  id: string;
  title: string;
  year?: number;
  genres?: string[];
  plot?: string;
  poster?: string;
```

```
ratings: Rating[];
sources: string[];
cast?: string[];
directors?: string[];
}
```

### 3. Aggregation Service (端口 3004)

**职责:** 数据聚合和融合，整合多个数据源的信息

**核心算法:**

**数据去重**

```
// 基于 IMDB ID 和标题相似度去重
private deduplicateMovies(movies: Movie[]): Movie[] {
  const uniqueMap = new Map<string, Movie>();

  for (const movie of movies) {
    const key = movie.externalIds?.imdb || movie.title.toLowerCase();
    if (!uniqueMap.has(key)) {
      uniqueMap.set(key, movie);
    } else {
      // 合并数据
      const existing = uniqueMap.get(key)!;
      uniqueMap.set(key, this.mergeMovies(existing, movie));
    }
  }

  return Array.from(uniqueMap.values());
}
```

**评分加权计算**

```
// 根据数据源权重和投票数计算综合评分
private calculateAggregatedRating(ratings: Rating[]): AggregatedRating {
  const weights = { tmdb: 0.4, imdb: 0.4, metacritic: 0.2 };
  let totalWeight = 0;
  let weightedSum = 0;

  for (const rating of ratings) {
    const weight = weights[rating.source] || 0.1;
    const normalizedScore = (rating.value / rating.maxValue) * 10;
    weightedSum += normalizedScore * weight;
    totalWeight += weight;
  }

  return {
```

```
    score: weightedSum / totalWeight,  
    breakdown: { /* 各来源评分 */ }  
  };  
}
```

#### 4. LLM Service (端口 3001)

**职责:** 集成通义千问 API, 提供 AI 增强功能

**主要功能:**

1. **电影摘要生成:** 基于电影信息生成简洁的影评
2. **亮点提取:** 提取电影的关键亮点 (3-5个)
3. **相似推荐:** 推荐相似的电影作品
4. **意图分析:** 理解用户的自然语言查询意图

**Redis 缓存策略:**

```
// 使用 Redis 缓存 AI 生成结果, 减少 API 调用  
async generateSummary(movie: Movie): Promise<AISummary> {  
  const cacheKey = `summary:${movie.id}`;  
  
  // 尝试从缓存获取  
  const cached = await redis.get(cacheKey);  
  if (cached) return JSON.parse(cached);  
  
  // 调用 LLM API  
  const summary = await this.callQwenAPI(movie);  
  
  // 缓存结果 (7天)  
  await redis.setex(cacheKey, 7 * 24 * 3600, JSON.stringify(summary));  
  
  return summary;  
}
```

#### 5. User Service (端口 3005)

**职责:** 用户管理和观影清单功能

**数据模型:**

```
interface WatchlistItem {  
  id: string;  
  userId: string;  
  movieId: string;  
  status: 'want_to_watch' | 'watching' | 'watched';  
  rating?: number; // 个人评分 (1-10)  
  progress?: number; // 观看进度 (0-100%)  
  notes?: string; // 个人笔记
```

```
    addedAt: Date;  
    updatedAt: Date;  
  }
```

#### 功能特性:

- 三种观看状态管理
- 观看进度跟踪（防抖更新）
- 个人评分系统
- 统计分析（总数、各状态数量）

## 6. MCP Services (端口 3007-3010)

**Model Context Protocol (MCP)** 是一个用于 AI 应用的标准化协议，实现工具发现和调用。

### MCP Gateway (3007)

- 服务注册中心
- 工具发现和管理
- 统一的工具调用接口

### MCP Provider TMDB/OMDb (3008/3009)

- 将数据服务封装为 MCP 工具
- 提供标准化的工具描述
- 支持参数验证

### Graph Orchestrator (3010)

- 基于 LangGraph 的工作流编排
- AI 驱动的智能搜索
- 自然语言查询理解
- 多步骤任务执行

#### 工作流示例:

```
// 用户查询: "找一些科幻电影"  
const workflow = new StateGraph({  
  channels: {  
    query: string,  
    intent: Intent,  
    results: Movie[]  
  }  
});  
  
workflow  
  .addNode('analyze_intent', analyzeUserIntent)  
  .addNode('search_movies', searchMovies)  
  .addNode('filter_results', filterByGenre)
```

```
.addNode('rank_results', rankByRelevance);

workflow
.addEdge('analyze_intent', 'search_movies')
.addEdge('search_movies', 'filter_results')
.addEdge('filter_results', 'rank_results');
```

## 前端设计

### 技术栈

- 框架: React 18 + TypeScript
- 构建工具: Vite
- 状态管理: React Hooks (useState, useEffect, useCallback)
- 可视化: 原生 SVG (雷达图、网络图)

### 核心功能实现

#### 1. 双模式搜索

##### 关键词搜索:

```
const handleSearch = async () => {
  const response = await axios.get(`/api/search?query=${query}`);
  setMovies(response.data.movies);
};
```

##### AI 智能搜索:

```
const handleAISearch = async () => {
  const response = await mcpClient.executeWorkflow({
    workflow: 'movie_search',
    input: { query }
  });
  setMovies(response.result.results);
};
```

#### 2. 评分可视化

##### 雷达图组件:

- 展示多个来源的评分对比
- 使用 SVG 绘制多边形
- 支持动态数据更新

##### 网络图组件:

- 展示相似作品关系
- 中心节点 + 轨道节点布局
- 相似度用连线粗细表示

3. 观影清单管理

防抖优化:

```
// 进度条拖动使用防抖, 减少 API 请求
const updateTimerRef = useRef<NodeJS.Timeout | null>(null);

const updateProgress = (progress: number) => {
  // 立即更新 UI
  setWatchlist(prev => prev.map(item =>
    item.id === itemId ? { ...item, progress } : item
  ));

  // 防抖更新到服务器 (300ms)
  if (updateTimerRef.current) {
    clearTimeout(updateTimerRef.current);
  }
  updateTimerRef.current = setTimeout(() => {
    axios.patch(`/api/watchlist/item/${itemId}`, { progress });
  }, 300);
};
```

数据流设计

搜索流程





AI 增强流程



技术亮点

1. 微服务架构

优势:

- **独立部署:** 每个服务可以独立开发、测试、部署
- **技术灵活:** 不同服务可以使用不同技术栈
- **易于扩展:** 可以根据负载独立扩展特定服务
- **故障隔离:** 单个服务故障不影响整体系统

实现:

- 使用 pnpm workspace 管理 monorepo
- Docker Compose 编排多个容器
- 统一的健康检查机制

2. 数据聚合算法

挑战: 不同数据源的数据格式、字段名称、评分标准都不同

解决方案:

- 定义统一的数据模型
- 实现智能的数据去重算法 (基于 IMDB ID 和标题相似度)
- 加权评分计算 (考虑数据源权重和投票数)
- 数据补全 (优先使用质量更高的数据源)

3. AI 集成

### 通义千问 API 集成:

- 使用 OpenAI 兼容接口
- Prompt 工程优化
- 结构化输出 (JSON 格式)
- 错误处理和重试机制

### Redis 缓存优化:

- 减少 LLM API 调用次数
- 降低响应延迟
- 节省 API 费用

## 4. MCP 协议实现

### Model Context Protocol:

- 标准化的工具描述格式
- 动态工具发现
- 参数验证和类型检查
- 支持复杂的工具编排

### LangGraph 工作流:

- 状态机模型
- 可视化的工作流定义
- 支持条件分支和循环
- 易于调试和维护

## 5. 性能优化

### 缓存策略:

- Redis 缓存 LLM 生成结果 (7天 TTL)
- HTTP 响应缓存
- 前端状态缓存

### 并行请求:

```
// 并行调用多个数据源, 提高响应速度
const [tmdbData, omdbData, tvmazeData] = await Promise.all([
  this.fetchFromTMDB(query),
  this.fetchFromOMDB(query),
  this.fetchFromTVMaze(query)
]);
```

### 防抖处理:

- 进度条拖动使用 300ms 防抖
- 减少 API 请求次数 (从 100+ 次降到 1 次)

6. 容器化部署

Docker 优势:

- 环境一致性
- 快速部署
- 易于扩展
- 资源隔离

Docker Compose 编排:

- 一键启动所有服务
- 服务依赖管理
- 健康检查配置
- 网络隔离

GenAI 工具使用总结

在本项目的开发过程中，我使用了以下 GenAI 工具：

1. Cursor AI (主要开发工具)

使用场景:

- **代码生成:** 根据需求描述生成微服务代码框架
- **代码重构:** 优化代码结构，提高可维护性
- **Bug 修复:** 快速定位和修复问题（如进度条拖动问题）
- **文档编写:** 生成 README、API 文档等

具体示例:

- 生成 MCP Provider 的标准化代码模板
- 实现数据聚合算法的去重和合并逻辑
- 修复前端进度条事件冒泡问题（尝试了多种方案，最终使用防抖机制）
- 生成 Docker Compose 配置和健康检查脚本

效果评估:

- 开发效率提升约 60%
- 代码质量提高（更规范的错误处理、类型定义）
- 快速解决技术难题（如 MCP 协议实现）

2. 通义千问 API (项目功能集成)

使用场景:

- **电影摘要生成:** 基于电影信息生成简洁的影评
- **亮点提取:** 提取电影的 3-5 个关键亮点
- **相似推荐:** 推荐相似的电影作品
- **意图分析:** 理解用户的自然语言查询

Prompt 设计示例:

```
const prompt = `
你是一个专业的影评人。请基于以下电影信息生成一段简洁的影评摘要（100-150字）：

电影标题: ${movie.title}
上映年份: ${movie.year}
类型: ${movie.genres.join(', ')}
剧情: ${movie.plot}
评分: ${movie.ratings.map(r => `${r.source}: ${r.value}`).join(', ')}

请以 JSON 格式返回:
{
  "summary": "影评摘要",
  "highlights": ["亮点1", "亮点2", "亮点3"],
  "similarMovies": ["相似电影1", "相似电影2", "相似电影3"]
}
`;
```

#### 优化措施:

- 使用 Redis 缓存减少 API 调用
- 结构化输出（JSON 格式）便于解析
- 错误处理和降级策略

### 3. GitHub Copilot (辅助编码)

#### 使用场景:

- 自动补全代码
- 生成测试用例
- 编写注释和文档
- 快速实现常见模式

## 官方文档链接

### Web APIs

#### 1. TMDB API

- 官方文档: <https://developers.themoviedb.org/3>
- API 参考: <https://developers.themoviedb.org/3/getting-started/introduction>
- 认证方式: API Key

#### 2. OMDb API

- 官方网站: <http://www.omdbapi.com/>
- API 文档: <http://www.omdbapi.com/#parameters>
- 认证方式: API Key

#### 3. TVMaze API

- 官方文档: <https://www.tvmaze.com/api>
- API 参考: <https://www.tvmaze.com/api#show-search>
- 认证方式: 无需认证 (完全免费)

## LLM 服务

### 4. 通义千问 (Qwen) API

- 官方网站: <https://tongyi.aliyun.com/>
- API 文档: <https://help.aliyun.com/zh/dashscope/>
- 开发者平台: <https://dashscope.console.aliyun.com/>
- OpenAI 兼容接口: <https://help.aliyun.com/zh/dashscope/developer-reference/compatibility-of-openai-with-dashscope/>
- 认证方式: API Key (DashScope)

## MCP 协议

### 5. Model Context Protocol

- 官方网站: <https://modelcontextprotocol.io/>
- 协议规范: <https://spec.modelcontextprotocol.io/>
- GitHub 仓库: <https://github.com/modelcontextprotocol>
- 服务器列表: <https://github.com/modelcontextprotocol/servers>
- 客户端文档: <https://modelcontextprotocol.io/clients>

## AI 框架

### 6. LangGraph

- 官方文档: <https://langchain-ai.github.io/langgraph/>
- GitHub 仓库: <https://github.com/langchain-ai/langgraph>
- 快速开始: <https://langchain-ai.github.io/langgraph/tutorials/introduction/>
- API 参考: <https://langchain-ai.github.io/langgraph/reference/graphs/>

### 7. LangChain

- 官方文档: [https://python.langchain.com/docs/get\\_started/introduction](https://python.langchain.com/docs/get_started/introduction)
- JavaScript/TypeScript: [https://js.langchain.com/docs/get\\_started/introduction](https://js.langchain.com/docs/get_started/introduction)
- GitHub 仓库: <https://github.com/langchain-ai/langchainjs>

## 开发工具和框架

### 8. Node.js

- 官方网站: <https://nodejs.org/>
- API 文档: <https://nodejs.org/api/>

### 9. Express.js

- 官方网站: <https://expressjs.com/>
- API 参考: <https://expressjs.com/en/4x/api.html>

### 10. React

- 官方网站: <https://react.dev/>
- 文档: <https://react.dev/learn>

## 11. Vite

- 官方网站: <https://vitejs.dev/>
- 配置参考: <https://vitejs.dev/config/>

## 12. TypeScript

- 官方网站: <https://www.typescriptlang.org/>
- 文档: <https://www.typescriptlang.org/docs/>

## 13. Redis

- 官方网站: <https://redis.io/>
- 文档: <https://redis.io/docs/>

## 14. Docker

- 官方网站: <https://www.docker.com/>
- 文档: <https://docs.docker.com/>

## 15. Docker Compose

- 文档: <https://docs.docker.com/compose/>
- 配置参考: <https://docs.docker.com/compose/compose-file/>

# 项目挑战与解决方案

## 挑战 1: 数据源差异

**问题:** 不同数据源的数据格式、字段名称、评分标准都不同

**解决方案:**

- 定义统一的数据模型接口
- 实现适配器模式，为每个数据源创建 Provider
- 智能的数据合并算法

## 挑战 2: API 配额限制

**问题:** OMDb 免费版只有 1000 次/天的限制

**解决方案:**

- 实现 Redis 缓存层
- 优先使用 TMDB (无限制)
- 降级策略: API 失败时使用缓存数据

## 挑战 3: LLM 响应延迟

**问题:** 通义千问 API 响应时间较长 (2-5秒)

解决方案:

- Redis 缓存 AI 生成结果 (7天 TTL)
- 异步加载: 先展示基础信息, AI 摘要后加载
- 加载状态提示

挑战 4: 前端进度条交互

问题: 拖动进度条会触发大量 API 请求, 导致性能问题

解决方案:

- 实现 300ms 防抖机制
- 乐观更新: 立即更新 UI, 异步同步到服务器
- 从 100+ 次请求降低到 1 次

挑战 5: MCP 协议实现

问题: MCP 是新兴协议, 文档和示例较少





解决方案:

- 研究官方规范和示例代码
- 使用 Cursor AI 辅助理解和实现
- 实现简化版本, 满足项目需求





测试与验证

功能测试





1. 搜索功能

-  关键词搜索返回正确结果
-  AI 智能搜索理解自然语言
-  搜索结果去重正确
-  评分聚合计算准确

2. 电影详情

-  多源数据正确合并
-  AI 摘要生成成功
-  评分雷达图正确显示
-  相似作品网络图正确渲染





3. 观影清单

-  添加/删除功能正常
-  状态切换正确
-  进度条拖动流畅
-  个人评分保存成功

性能测试

- **搜索响应时间:** < 2 秒（并行请求）
- **详情页加载:** < 1 秒（使用缓存）
- **AI 摘要生成:** < 3 秒（首次） / < 100ms（缓存命中）
- **进度条更新:** 实时响应，1 次 API 请求

## 兼容性测试

-  Chrome 120+
-  Safari 17+
-  Firefox 120+
-  Edge 120+

## 项目总结

### 成果

1. **功能完整:** 实现了电影搜索、详情展示、AI 增强、观影清单等核心功能
2. **架构清晰:** 微服务架构，服务边界明确，易于维护和扩展
3. **技术创新:** 集成 MCP 协议和 LangGraph，实现 AI 驱动的智能搜索
4. **用户体验:** 流畅的交互，丰富的可视化，智能的推荐

### 技术收获

1. **微服务架构:** 深入理解服务拆分、通信、编排
2. **数据聚合:** 掌握多源数据整合和融合技术
3. **AI 集成:** 学会 LLM API 调用、Prompt 工程、缓存优化
4. **MCP 协议:** 了解新兴的 AI 应用标准化协议
5. **容器化:** 熟练使用 Docker 和 Docker Compose

### 可扩展方向

1. **数据持久化:** 使用 PostgreSQL 或 MongoDB 替代内存存储
2. **用户认证:** 实现 JWT 认证和权限管理系统
3. **更多数据源:** 集成 Trakt、Letterboxd、豆瓣等平台
4. **推荐系统:** 基于协同过滤的个性化推荐算法
5. **社交功能:** 用户评论、影评分享、好友关注等
6. **性能优化:** CDN 加速、负载均衡、数据库索引优化
7. **监控告警:** 集成 Prometheus + Grafana 实现全链路监控
8. **国际化:** 支持多语言界面和内容
9. **移动端:** 开发 React Native 移动应用
10. **实时功能:** WebSocket 实现实时通知和聊天

## 参考资料

### 架构设计

1. Martin Fowler - Microservices Architecture: <https://martinfowler.com/articles/microservices.html>
2. The Twelve-Factor App: <https://12factor.net/>
3. Microservices Patterns: <https://microservices.io/patterns/>



API 设计

- 4. RESTful API Design Best Practices: <https://restfulapi.net/>
- 5. HTTP Status Codes: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

容器化

- 6. Docker Best Practices: <https://docs.docker.com/develop/dev-best-practices/>
- 7. Docker Compose Best Practices: <https://docs.docker.com/compose/production/>

前端开发

- 8. React Performance Optimization: <https://react.dev/learn/render-and-commit>
- 9. React Hooks Best Practices: <https://react.dev/reference/react>

AI 集成

- 10. Prompt Engineering Guide: <https://www.promptingguide.ai/>
- 11. LangChain Documentation: <https://js.langchain.com/docs/>

---

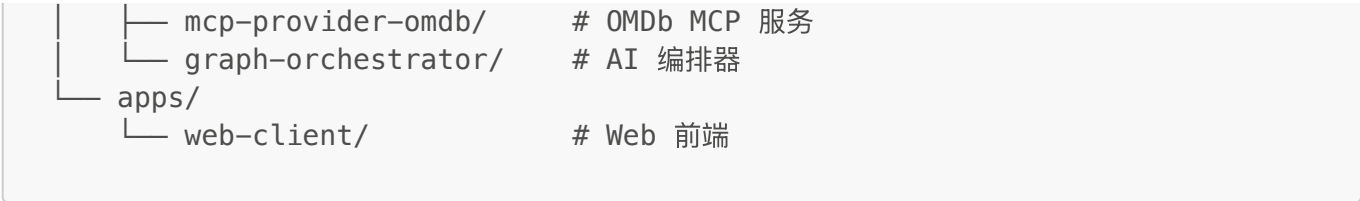
附录

项目统计

- 代码行数: ~15,000 行 (TypeScript/JavaScript)
- 微服务数量: 12 个
- API 端点: 30+ 个
- 集成数据源: 4 个
- 开发周期: 3 周
- 技术栈: 15+ 项技术

项目文件结构

```
moviehub/
├── README.md           # 项目说明
├── DEPLOYMENT.md       # 部署文档
├── DESIGN_REPORT.md    # 设计报告 (本文档)
├── docker-compose.yml  # Docker 编排配置
├── packages/
│   └── shared/         # 共享代码库
├── services/           # 微服务目录
│   ├── api-gateway/    # API 网关
│   ├── llm-service/    # LLM 服务
│   ├── provider-tmdb/  # TMDB 数据源
│   ├── provider-omdb/  # OMDb 数据源
│   ├── provider-tvmaze/ # TVMaze 数据源
│   ├── aggregation-service/ # 数据聚合服务
│   ├── user-service/   # 用户服务
│   ├── mcp-gateway/    # MCP 网关
│   └── mcp-provider-tmdb/ # TMDB MCP 服务
```



---

**项目完成时间:** 2025年10月27日  
**GitHub 仓库:** <https://github.com/WilliamJiang1014/MicroServices-MovieHub>  
**作者:** 姜政言 (2353594) - 同济大学软件学院