

乐拍视界项目文档

项目小组 第 2 小组

小组成员 柏翔 刘金林 兰寅银 朱灿银 周俊

联系方式 17748752006

重庆师范大学软件工程系

摘要

“乐拍视界”项目旨在解决年轻群体创作音乐短视频时操作繁琐、社交孤立的痛点，通过打造一个集智能拍摄、海量配乐、一键美化与沉浸式社交于一体的移动平台，开创短音乐视频新赛道。项目计划以 6-12 个月完成开发，核心是整合音乐库、智能算法与推荐系统，旨在通过极致流畅的一体化体验，快速吸引用户并构建活跃社区，最终成为引领年轻文化的领先平台。

日期	修改	描述	作者
2025 年 11 月 4 日	0.1	完成了立项和愿景	柏翔 刘金林 兰寅银 朱灿 银 周俊
2025 年 11 月 23 日	0.2	完成了用况建模	柏翔 刘金林 兰寅银 朱灿 银 周俊
2025 年 12 月 14 日	0.3	完成了需求分析	柏翔 刘金林 兰寅银 朱灿 银 周俊
2026 年 1 月 11 日	0.4	完成了架构设计	刘金林 兰寅 银 周俊
206 年 1 月 16 日	0.5	完成了详细设计	刘金林 兰寅 银 朱灿银

目录

摘要.....	2
第 1 章 立项.....	8
1.1. 项目起源与提案.....	8
1. 发现问题.....	8
2. 提案构想.....	8
1.2. Business Case.....	8
1. 摘要.....	8
2. 市场机遇.....	9
3. 目标市场与客户细分.....	9
4. 竞争优势.....	9
5. 市场营销与用户获取策略.....	9
6. 风险与应对.....	9
7. 成本估算.....	10
8. 项目目标.....	10
第 2 章 愿景.....	11
2.1. 问题陈述.....	11
1. 问题一.....	11
2. 问题二.....	11
3. 问题三.....	12
2.2. 涉众与用户.....	12
1. 涉众.....	12
2. 用户.....	13
2.3. 关键涉众和用户的需要.....	14
2.4. 产品概述.....	16
1. 产品定位陈述.....	16
2. 完整的产品概述.....	16
2.5. 产品特性.....	18
2.6. 特性优先级.....	21
2.7. 其他产品需求.....	23
第 3 章 用况建模.....	25
3.1. 术语表.....	25
3.2. 参与者清单.....	26
1. 识别主参与者.....	26
2. 参与者简要描述.....	26
2.1. 网民.....	26
2.2. 内容审核员.....	26
2.3. 客服.....	26
2.4. 运营人员.....	26
3.3. 乐拍视界的主要用况.....	26
1.1. 视频社交.....	27
1.2. 审核视频.....	27

1.3. 解答网民问题.....	27
3.4. 视频社交用况的描述.....	27
1. 完全描述.....	27
1.1. 前置条件.....	27
1.2. 基本流.....	27
1.3. 备选流.....	29
第4章 需求分析.....	30
4.1. 健壮性分析.....	30
1. 视频社交用况.....	30
1.1. 获取要播放的视频：	30
1.2. 点赞视频.....	30
1.3. 关注其他网民.....	31
1.4. 浏览被关注的网民的视频.....	31
1.5. 拍摄视频.....	31
1.6. 编辑视频.....	32
1.7. 上传视频.....	32
4.2. 交互建模.....	33
1. 视频社交.....	33
1.1. 顺序图.....	33
4.3. 总类图：	37
第5章 架构设计.....	38
5.1. 设定权衡优先级.....	38
1. 核心设计目标的优先级排序.....	38
2. 具体的架构权衡决策.....	38
2.1. 空间换时间.....	38
2.2. 可用性优于一致性.....	39
2.3. 功能分级与快速迭代.....	39
5.2. 估算系统性能.....	39
1. 基础假设与模型参数.....	39
2. 计算吞吐量.....	40
2.1. 视频浏览.....	40
2.2. 视频上传.....	40
2.3. 点赞视频.....	40
3. 估算存储需求.....	40
4. 估算网络带宽.....	41
5. 性能估算总结与架构调整.....	41
5.3. 选择架构风格.....	41
1. 整体系统架构： 分布式客户/服务器风格.....	41
2. 客户端架构： 混合架构.....	42
2.1. 整体风格： MVVM (Model-View-ViewModel).....	42
2.2. 核心流媒体架构： MVD (Model-View-Delegate).....	42
3. 服务器端架构： 分层与事件驱动架构.....	42
4. 视频处理子系统： 基于消息队列的异步任务架构.....	42
5. 数据存储架构： 主从复制与读写分离.....	43

5.4. 将系统划分成子系统.....	43
1. 客户端子系统的划分.....	43
1.1. 媒体采集与预处理子系统.....	43
1.2. 播放与互动子系统.....	43
1.3. 网络通信与配置子系统.....	44
2. 服务端子系统的划分.....	44
2.1. API 网关与业务子系统.....	44
2.2. 异步媒体处理子系统.....	44
2.3. 数据与存储基础设施.....	44
3. 子系统间的依赖与通信关系.....	45
5.5. 确定问题内部的并发性.....	45
1. 移动客户端的并发设计.....	45
2. 服务端 API 服务的并发设计 (Reactor 模型).....	45
3. 后台转码服务的并发设计.....	46
4. 对象状态与互斥管理.....	46
5.6. 配置子系统的硬件.....	46
1. 物理节点分类与配置.....	46
1.1. 移动终端节点.....	46
1.2. 服务终端节点.....	46
2. 网络连接.....	47
3. 硬件冗余与故障转移.....	47
5.7. 管理数据存储.....	47
1. 非结构化数据: MinIO 分布式对象存储.....	47
2. 结构化数据: PostgreSQL (应用层读写分离).....	48
3. 缓存与消息队列: Redis.....	48
5.8. 处理全局资源.....	48
1. 全局唯一标识符 (UUID).....	48
2. 身份认证与访问控制.....	48
3. 视频处理控制策略 (管道与过滤器).....	49
4. 异常处理与边界条件.....	49
5.9. 选择软件控制策略.....	49
1. 移动客户端: 事件驱动型控制策略.....	49
2. 服务端 API 服务: 异步回调与 Reactor 控制策略.....	49
3. 视频后台处理: 轮询调度与进程隔离策略.....	49
4. 异常与中断处理策略.....	50
5.10. 处理边界条件.....	50
1. 初始化.....	50
2. 终止.....	50
3. 故障与失效.....	51
5.11. 制订复用计划.....	52
1. 框架级复用.....	52
2. 外部程序与类库复用.....	52
3. 内部组件复用与构建.....	52
4. 设计模式复用.....	53

5.12. 乐拍视界软件架构.....	53
第 6 章 详细设计.....	55
6.1. 设计规范与标准.....	55
1. 代码命名规范.....	55
1.1. C++ 类与文件.....	55
1.2. 成员变量与局部变量.....	55
1.3. 成员函数（操作）.....	55
1.4. QML 组件与属性.....	55
1.5. 数据库对象.....	55
2. 可见性与封装标准.....	55
2.1. C++ 访问修饰符标准.....	55
2.2. QML 交互接口标准.....	56
3. 数据类型映射标准.....	56
3.1. 基础数据类型映射.....	56
3.2. 值对象定义.....	56
3.3. 空值与可选值处理.....	56
4. 操作签名标准.....	56
4.1. 参数列表.....	57
4.2. 返回值.....	57
4.3. 异常规范.....	57
4.4. Const 正确性.....	57
6.2. 类的详细设计与精化.....	57
1. 客户端类设计.....	57
2. 服务端类设计（api_service）.....	58
3. 服务端类设计（video_worker）.....	61
6.3. 操作与算法设计.....	61
1. 操作签名定义.....	61
1.1. 服务端：视频服务.....	61
1.2. 客户端：视频上传管理器.....	62
2. 对象状态建模.....	62
2.1. 视频生命周期状态机.....	62
2.2. 播放器控制器状态机.....	63
6.4. 关联与结构设计.....	65
6.5. 设计优化与模式应用.....	65
1. 设计模式应用.....	66
1.1. 单例模式.....	66
1.2. 工厂模式.....	66
1.3. 观察者模式.....	66
1.4. 代理模式.....	66
2. 性能与资源优化.....	67
2.1. 冗余数据设计.....	67
2.2. 路径优化与预计算.....	67
2.3. 对象池技术.....	67
2.4. 异步并发与写缓冲.....	68

6.6. 数据存储详细设计.....	68
1. 关系型数据库模式 (PostgreSQL).....	68
1.1. ER 图转物理表结构定义.....	68
1.2. 索引设计.....	69
2. 非结构化存储设计 (MinIO).....	70
2.1. 存储桶 (Bucket) 规划.....	70
2.2. 目录层级结构.....	70
3. 缓存数据结构 (Redis).....	70
3.1. Key-Value 命名规范.....	70
3.2. 核心缓存结构定义.....	71
6.7. 接口详细说明.....	71
1. 客户端-服务端交互接口.....	72
1.1. 通用协议规范.....	72
1.2. 核心接口定义.....	72
1.3. 错误码定义表.....	73
2. QML-C++ 交互接口.....	73
2.1. 上下文属性.....	74
2.2. 数据模型接口.....	74
3. 子系统内部接口.....	74
3.1. 推荐引擎调用接口.....	74
3.2. 视频转码服务回调接口.....	75

第1章 立项

1.1. 项目起源与提案

1. 发现问题

在当前的移动互联网环境下，我们观察到青少年群体中兴起了一种自发的、富有创造力的内容创作模式：他们使用手机或数码相机录制生活中的精彩片段（如舞蹈、滑板、搞笑短剧等），然后借助另一台设备（如电脑、MP3播放器或另一部手机）播放背景音乐，通过后期剪辑或直接跟拍的方式，将画面与音乐结合，最终将作品上传到论坛、博客或早期视频网站进行分享。这个过程虽然充满热情，但存在明显的技术断层和体验割裂：

操作繁琐：用户需要在不同设备间切换，涉及文件传输、音画对齐等复杂步骤，创作门槛高。

即时性差：无法实现“即想即拍、即拍即享”，灵感与创作冲动在繁琐的流程中被消耗。

社交孤立：分享渠道分散，缺乏一个专注于此类短音乐视频的平台，创作者难以找到同好，无法形成有效的互动和反馈闭环。

2. 提案构想

因此，我们提议开发“乐拍视界”——一款真正实现视频拍摄、智能配乐、一键美化、无缝社交一体化的移动应用程序。

对于视频拍摄者：我们将提供海量正版热门音乐库，用户可以在拍摄前或拍摄后轻松选择配乐；应用内置智能节拍识别功能，能自动将视频画面与音乐高潮点对齐；提供多种电影级的滤镜和转场特效，让零基础的网民也能在几分钟内创作出酷炫的、富有感染力的音乐短视频。

对于内容消费者与社交分享者，我们致力于打造一个以音乐为纽带、以视频为载体的沉浸式社交平台。平台将用户从传统的“搜索—观看”单向模式中解放出来，转向“发现—互动—再创作”的闭环体验，构建一个真正“懂你”的音乐视频互动社区：

在内容层面，我们通过强大的个性化推荐算法，实现从“人找内容”到“内容找人”的转变。系统会根据用户的观看偏好、互动行为与音乐口味，智能推送感兴趣的视频，降低选择成本，提升沉浸感。

在社交层面，我们重视人与人之间的连接与互动。用户可通过“同款音乐”功能进行创意比拼，通过点赞、评论、分享和关注等行为，与其他创作者建立联系。形成一个紧密的、充满活力的创意社群。这些互动能够促进内容的流动与传播。

1.2. Business Case

1. 摘要

该项目瞄准15-30岁的年轻群体，旨在解决其制作高质量、富有表现力的音乐短视频时面临的“操作复杂、社交低效”的核心痛点。通过将专业的视频拍摄、庞大的音乐库、智能的剪辑工具与强大的社交功能无缝整合，“乐拍视界”将开创“移动短音乐视频社交”这一全新赛道。我们预期通过快速获取用户、构建内容生态，并最终通过广告、虚拟商品、直播

和品牌合作等多种方式实现盈利，占据市场领导地位。

2. 市场机遇

移动设备普及： 智能手机性能不断提升，网络开始普及，为移动端视频的拍摄、上传和播放提供了硬件基础。

用户行为变迁： 年轻一代是“视觉系”原住民，他们更倾向于用图像和视频而非文字进行表达和社交。

音乐需求旺盛： 音乐是年轻人表达情绪、彰显个性的重要载体，与视频结合拥有巨大的想象空间。

市场空白： 现有社交平台（如微博、QQ空间）或视频平台（如优酷、Youtube）均未提供专为“短音乐视频”设计的、便捷的观看，创作与社交一体化体验。

3. 目标市场与客户细分

核心用户： 15-25 岁的学生和年轻白领。他们追求潮流、乐于表达、渴望认同、拥有强烈的社交需求。

次要用户： 25-30 岁的都市青年，以及有记录和分享孩子成长瞬间需求的年轻父母。

潜在用户： 寻求与年轻消费者建立连接的品牌方、广告主，以及希望通过内容创作获得影响力的创作者/KOL。

4. 竞争优势

产品体验优势： 打开即播放，单列信息流，上下滑动切换的模式让观看视频简单轻松。提供低门槛的创作体验，将海量正版音乐库、一键式美颜滤镜、丰富的特效模板整合进 App。用户不需要专业技巧，就能轻松创作出看起来“很酷”的视频。一体化产品体验，构建了从发现、音乐选择、特效拍摄、一键发布到互动分享的极致流畅闭环，其无缝体验显著优于功能割裂的传统音视频工具。

技术与算法优势： 该产品采用基于深度学习的推荐算法，能通过用户极短时间的观看行为（完播率、点赞、评论、转发、停留时长等），精准地建模用户兴趣，并实时调整推荐内容。并且可以给用户在拍摄时提供滤镜、美颜、贴纸、背景、时间特效（如慢动作、快动作）等功能。

5. 市场营销与用户获取策略

冷启动： 从艺术院校、舞蹈社团、街舞爱好者等垂直社群切入，邀请种子用户，生产高质量内容。

校园推广： 与全国高校合作，举办“校园音乐视频大赛”，快速在目标人群中建立知名度。

线上营销： 在微博、贴吧、QQ空间等年轻人聚集的社交平台进行内容投放和话题炒作。

6. 风险与应对

竞争风险： 其他软件的模仿与跟进。对策：以快速迭代产品，不断更新技术，提升用户体验为基础，花重金提高产品知名度，拓展商业合作，最终实现市场上的稳固地位。

版权风险：音乐版权纠纷。对策：初期与音乐版权代理商建立正规合作，后期建立自己的版权库。

内容风险：用户上传违规内容。对策：建立“AI 算法+人工审核”的内容审核机制，确保内容健康。

盈利风险：无法高效盈利。对策：谨慎探索多元化的商业模式，优先保障用户体验。

7. 成本估算

一次性开发成本：

开发团队 5 人，打造一个基础版本约需 6-12 个月，按平均月薪 1 万计算，1 个月约 5 万。初期用户量少，采用云服务（如阿里云、腾讯云）。视频存储和带宽是主要开销，初期每月约 1-3 万。最低开发成本最低为 36 万，最高为 96 万。

持续运营成本：

团队扩充至 30-40 人，新增运营、市场、算法、审核等岗位，年度人力成本约 800 万-1200 万。

带宽与服务器成本（随用户量指数增长）：若日活达到百万级，月度带宽成本可能达到 50 万-200 万，年度 600 万-2400 万。

市场营销费用：用户获取与品牌建设，预估 500 万-1000 万。

音乐版权与内容审核：预估 1 年 200 万。

首年持续运营总成本预估：2100 万-4800 万人民币。

8. 项目目标

短期目标（6-12 个月）：成功上线 Android 版本，积累首批核心种子用户，建立活跃的创作者社区，实现每日用户创作视频量过万。

中期目标（1-2 年）：成为国内青少年群体中最受欢迎的短音乐视频社交平台，形成独特的社区文化，探索初步的商业化模式。

长期目标（3-5 年）：构建以“乐拍视界”为核心的短视频内容生态，成为引领年轻文化潮流的重要阵地，并拓展至海外市场。

第2章 愿景

2.1. 问题陈述

1. 问题一

要素	描述
问题	网民的创作过程割裂且技术门槛高，缺乏一个集成化工具，能够让他们在移动端轻松、快速地将视频与热门音乐结合，并添加专业特效的创作。现有流程依赖多个割裂的设备和复杂软件，操作繁琐。
影响	创作者
结果	网民旺盛的创作热情和灵感被技术难题所压制。繁琐的流程导致网民从“灵感”到“作品”的转化率极低，大量创意被浪费。平台内容生态依赖少数技术娴熟的创作者，内容风格单一，数量增长缓慢。绝大多数潜在创作者保持沉默，无法形成百花齐放的社区氛围。
优点	<p>该产品提供了一体化移动端创作工具，内嵌海量正版音乐库、智能剪辑功能和丰富特效。</p> <p>实现“所想即所得”，网民只需选择音乐和片段，算法自动完成音画对齐与节奏匹配，将创作门槛降至最低。</p> <p>激发创作欲望，让每个人都能轻松制作出酷炫的音乐短视频，从而极大地丰富了平台内容的多样性和数量。</p>

2. 问题二

要素	描述
问题	内容发现效率低下，网民留存困难。传统平台依赖用户主动搜索和订阅，这是一种“人找内容”的低效模式，无法根据网民的潜在兴趣进行精准内容推荐，导致网民陷入选择疲劳。
影响	平台，网民
结果	<p>网民难以持续发现感兴趣的内容，浏览体验枯燥。平台无法为用户创造“上瘾”的沉浸感，导致网民使用时长短、流失率高。</p> <p>平台活跃度增长陷入瓶颈，无法让网民稳定留在平台。即使有优质内容，也无法被对的人看到，内容价值无法最大化。</p>
优点	<p>引入基于 AI 的个性化推荐引擎，通过分析用户行为，精准推送其可能感兴趣的内容，从“人找内容”变为“内容找人”。</p> <p>打造全屏沉浸式信息流，提供无缝、零思考的浏览体验，最大化用户的专注度和停留时长。</p>

3. 问题三

要素	描述
问题	社交互动薄弱，从观看到创作的转化路径断裂。现有平台的社交互动停留在浅层的点赞和评论，且观看与创作是两个完全割裂的场景。用户即使被视频激发起创作欲望，也缺乏无缝、低成本的创作方式。
影响	用户
结果	<p>用户仅是被动的内容消费者，难以深度融入社区。创作灵感因复杂的转化路径而转瞬即逝，平台无法将高涨的观看情绪有效转化为创作行为。</p> <p>平台里没有热闹的交流感，更像用户各自刷内容的“冷清空间”，用户之间没形成关联。内容生态缺乏自生长的动力，需要持续的外部刺激来维持内容产出，运营成本高。</p>
优点	<p>构建以“同款音乐”和“挑战赛”为核心的互动机制，用户可一键使用同款模板参与创作，实现“看了就想拍”。</p> <p>深度整合社交功能，如好友动态、合拍等，强化用户之间的创意互动与联系。</p> <p>形成“观看-灵感-创作-互动”的完美闭环，驱动内容的自我繁荣。</p>

2.2. 涉众与用户

1. 涉众

涉众	涉众类型	简要描述
项目经理	开发团队	制定项目计划，管理进度、风险和资源，保证项目按期交付
测试人员	开发团队	测试系统的功能、性能、数据安全
需求分析师	开发团队	与用户沟通，获取需求和想法，将这些需求转化为详细的需求规格说明书。
编码员	开发团队	负责编码实现系统
审核人员	平台维护人员	审核用户发布的作品，删除违规的作品并提醒作者,处理违规用户
客服	平台维护人员	解答用户的问题
官方账号运营团队	平台维护人员	运营平台的官方账号，通过发布内容、策划线上活动来激发社区活力，提升用户粘性
国家互联网信息办公室	监管机构	负责互联网信息内容的管理，落实互联网信息传播的方针政策，指导、协调和督促有关部门加强网络内容管理，并依法查处违法违规网站和应用。

工业和信息化部	监管机构	负责电信与互联网行业的管理，监管范围包括网络基础设施、信息服务业务许可、网络与信息安全技术标准等。
国家版权局	监管机构	负责监管平台上的版权问题，处理用户上传内容可能涉及的音乐、视频、文字等版权侵权纠纷。
公安部	监管机构	负责网络安全保卫工作，打击利用平台进行的网络诈骗、传播违法信息、侵犯公民个人信息等违法犯罪活动。
网民	用户	观看或发布音视频
音乐版权方	版权方	提供音乐的版权，防止音乐作品在未经授权的情况下被平台用户使用
应用商店	合作伙伴	审核应用资质，确保应用上架符合相关法律法规和平台规定
电信运营商	合作伙伴	提供网络连接
云服务商	合作伙伴	提供云服务
品牌方	合作伙伴	提供钱让平台发布广告，他们是平台未来实现流量变现、进行商业合作的关键对象。
投资者	发起人	资金提供者，关注投资回报

2. 用户

用户	用户类型	简要描述
观看者	内容消费者	寻求轻松、有趣的娱乐方式，打发碎片时间；通过浏览内容获得放松。 或为获取知识、技能、资讯，主动搜索或关注科普、技能、资讯类内容，追求内容的专业性、实用性和可学习性。
官方机构	内容创作者	宣传活动，发布热点视频吸引流量代表平台运营官方账号，通过策划宣传活动、发布热点视频、输出平台动态等方式吸引流量、传递品牌价值、维护用户关系。
博主	内容创作者	渴望进行自我表达，获得关注和认同；需要低门槛、高效的创作工具；希望通过分享日常、

		生活技巧等内容，与同好交流，积累粉丝。
运营团队	管理类	负责内容生态搭建、创作者扶持与管理，统筹内容审核、流量运营及用户维护，保障平台有序运转与持续增长。
审核人员	业务处理类	对平台内容、用户行为进行合规性审核，依据平台规则筛查违规信息，保障平台内容生态的健康、安全与合规。
客服	业务处理类	及时解答用户在平台使用过程中的疑问、投诉与建议，处理用户诉求，提升用户使用体验与满意度。

2.3. 关键涉众和用户的需要

关键涉众	需要
投资者	<p>1. 商业回报：清晰的盈利路径（如广告、虚拟商品、电商）和投资回报率。</p> <p>2. 增长潜力：高速的用户增长、市场占有率及平台网络效应的形成。</p> <p>3. 竞争壁垒：产品相较于潜在竞争者的独特优势和可持续性（如技术、社区文化）。</p>
音乐版权方	<p>1. 版权保护与收益：其音乐作品被合法使用，并能获得公平的版权分成。</p> <p>2. 作品推广：平台能成为其新歌、新艺人推广的有效渠道，扩大音乐影响力。</p> <p>3. 数据洞察：获得其音乐在平台上的使用数据（如播放量、热门视频等），以指导宣发。</p>
政府监管机构	<p>1. 内容合规：平台内容符合国家安全、社会公序良俗及青少年保护法规。</p> <p>2. 数据安全：用户数据（尤其是未成年人数据）的收集、使用符合相关法律要求。</p> <p>3. 协同治理：平台能积极配合监管要求，建立有效的自查与清理机制。</p>

关键用户	需要
观看者	<p>1. 个性化娱乐：平台能“懂我”，通过精准算法持续提供我感兴趣的内容，带来轻松愉悦的“杀时间”体验。</p> <p>2. 社交归属感：能关注喜欢的创作者，与同好互动（点赞、评论），感觉自己身处一个有趣的社区。</p>

	<p>3. 内容新鲜度：总能发现新的潮流、热门话题和创意形式，保持对平台的新鲜感和期待。</p> <p>4. 精准获取内容：通过关键词搜索、标签分类快速找到目标内容，过滤无效信息，高效获得想要的内容。</p> <p>5. 内容权威性：内容来源可信、逻辑清晰，有实操步骤或权威依据，能真正了解世界，扩充自己。</p> <p>6. 视频体系化：支持内容收藏、合集查看，方便按主题连贯观看。</p>
博主	<p>1. 低门槛表达：提供简单易用、功能强大的创作工具（音乐库、特效、模板），让创意能轻松实现。</p> <p>2. 获得认可与影响力：获得粉丝、点赞、评论等社交正反馈，积累个人影响力，满足表达欲和成就感。</p> <p>3. 创作灵感启发：能方便地追踪热门挑战和流行趋势，获得持续创作的灵感和动力。</p> <p>4. 提高粉丝留存度：增加账户热度。</p> <p>5. 保证粉丝活跃度：扩大账户知名度。</p>
运营团队	<p>1. 生态健康运转：通过数据工具监控内容质量、用户行为，及时调整运营策略，维持正向生态。</p> <p>2. 创作者扶持：搭建分层扶持体系，提供流量倾斜、工具特权等资源，激励优质创作者留存。</p> <p>3. 增长与转化：策划热点活动、优化流量分发机制，提升用户活跃度、留存率及商业转化效率。</p>
官方机构	<p>1. 高效宣传：借助平台流量优势，让宣传内容精准触达目标受众，提升活动或品牌曝光度。</p> <p>2. 权威形象传递：通过官方认证标识、合规内容审核支持，保障信息发布的权威性和可信度。</p> <p>3. 互动与反馈：搭建与用户的沟通渠道，收集公众意见，提升品牌好感度和用户粘性。</p>
审核人员	<p>1. 高效审核工具：提供智能筛查、关键词识别等辅助工具，提升违规内容识别效率，降低工作负荷。</p> <p>2. 明确审核标准：有清晰、统一的规则手册和更新机制，避免审核判断偏差。</p> <p>3. 风险预警支持：对高风险内容、敏感话题提前预警，保障审核工作的准确性和及时性。</p>

客服	<p>1. 高效响应工具：整合用户咨询渠道，提供快捷回复模板、问题分类标签，快速处理用户诉求。</p> <p>2. 问题解决支持：获取平台规则、功能说明等权威资料，能准确解答用户疑问或协调处理复杂问题。</p> <p>3. 用户反馈同步：建立反馈机制，将用户高频问题、建议同步给相关团队，优化服务体验。</p>
----	---

2.4. 产品概述

1. 产品定位陈述

For	渴望表达自我的年轻一代与寻求轻松、愉悦内容消费的移动互联网用户
Who	他们需要一种简单、有趣的方式来创作和分享生活，并渴望获得即时的社交互动与认同。但现有工具创作门槛高，平台内容分发效率低下，观看与创作行为割裂。
The	乐拍视界
That	是一款集音乐短视频创作、智能推荐与沉浸式社交于一体的移动平台。我们通过一体化创作工具和精准的推荐算法，为用户提供零门槛的创意表达和高度个性化的内容消费体验，让每个人都能轻松记录和分享生活中的乐趣。
Ulike	不同于功能复杂、以长视频和搜索为核心的 YouTube，也不同于功能单一、缺乏社交生态的 Dubsmash。
Our product	我们提供了从创意激发、极简创作到精准分发与互动的完整闭环，构建了一个充满活力的创意社区。

2. 完整的产品概述

一、能力概述

乐拍视界作为移动端短音乐视频社交应用，核心向用户提供三大核心能力。创作方面，通过内置海量正版音乐库、智能剪辑工具、美颜滤镜、动态贴纸等功能，以及“一键出片”模板，实现低门槛视频创作与美化；消费方面，以全屏上下滑动的沉浸式信息流为载体，通过个性化推荐算法，为用户推送定制化内容流；社交方面，借助点赞、评论、关注、分享、私信等功能。构建以音乐为核心的创作互动与灵感交流闭环，覆盖“创作-消费-社交”全场景需求。

二、客户效益（Benefits）

涉众类型	核心效益	对应产品特性
创作者	降低创意表达门槛，快速实现创作想法	智能音乐匹配、简易拍摄美化、创意特效库、“一键出片”模板
	获得社交认同与灵感启发，提升创作动力	“拍同款”功能、同款音乐聚合页

涉众类型	核心效益	对应产品特性
消费者	高效获取感兴趣的内容，获得沉浸娱乐体验	个性化推荐算法、全屏沉浸式信息流
	发现潮流内容与同好，增强社区归属感	互动功能
品牌 / 运营方	实现品牌宣传与商业收益转化	直播带货、广告投放系统

三、假设和依赖

核心假设

- 用户对短音乐视频的创作需求持续存在，且倾向于 “低成本、高创意呈现” 的创作模式。
- 个性化推荐算法能精准捕捉用户兴趣，持续提升用户留存与使用时长。
- 以音乐为核心的社交互动模式，能有效激发用户参与度，形成稳定的社区生态。

关键依赖

- 版权依赖：需持续与主流音乐版权方合作，保障曲库的合法性、丰富度与时效性。
- 技术依赖：依赖智能节拍识别、推荐算法、实时特效渲染等技术的稳定迭代与优化。
- 环境依赖：适配主流移动端操作系统（Android），需兼容不同品牌、型号的手机硬件与系统版本。
- 生态依赖：需吸引足够数量的创作者产出优质内容，同时积累初始用户群体，形成 “创作 - 消费” 的正向循环。

四、取舍和竞争

核心取舍

- 功能取舍：优先聚焦 “音乐 + 短视频” 核心场景，简化复杂编辑功能，暂时放弃长视频、多场景综合剪辑等非核心能力，确保创作门槛足够低。
- 内容取舍：侧重年轻化、潮流化、娱乐化内容生态，暂时弱化专业知识。

竞争对比

竞争产品	优势	劣势	本产品差异化优势
快手	下沉市场渗透深、社区氛围浓厚、真实感强	内容精致度不足、潮流属性弱、推荐精准度待提升	信息流更沉浸、创作工具更侧重音乐适配，潮流化内容导向更明确
YouTube	内容生态丰富、长短视频全覆盖、搜	功能复杂、操作门槛高、社交互动性弱	聚焦移动端短视频，简化操作流程，强化 “创作 - 社交” 闭环，

竞争产品	优势	劣势	本产品差异化优势
	索功能强		更贴合碎片化使用场景。

2.5. 产品特性

内容消费者：

（1）观看者

核心需要	对应系统特性
获得个性化娱乐体验，高效“杀时间”	精准推荐算法：基于用户兴趣与行为，持续推送感兴趣的短视频。
	全屏沉浸体验：上下滑动无缝切换视频，最大化娱乐沉浸感。
融入有趣社区，获得社交归属感	多元互动工具：提供关注、点赞、评论、私信等功能，构建互动社区。
	粉丝团体系：支持用户加入专属粉丝团，增强与创作者的联结。
持续接触新鲜潮流，保持平台新鲜感	热点聚合页面：实时更新热门话题与挑战，一站式追踪全网潮流。
	创意模板库：提供海量同款音乐、特效和拍摄模板，降低参与门槛。
高效获取实用知识或技能，解决实际问题	垂直知识库：按用途、技能等分类聚合深度内容。
	精准搜索筛选：支持关键词搜索，并可按最新、最热等维度筛选。
	“干货”标识系统：对知识密度高的视频进行标记，便于用户识别。
视频体系化，避免碎片化	合集/列表功能：创作者可将系列内容整理成合集，支持连续观看。
	视频进度跟踪：自动记录在合集中的观看进度，支持断点续看。
	笔记与收藏功能：支持在看视频时记录要点，并分类收藏内容。
辨别信息真伪，获取可靠内容	创作者认证体系：对教育、医学等领域的专业人士进行身份认证。
	事实核查机制：与权威机构合作，对热门技能、知识类视频进行事实标注。

	优质创作者推荐：在相关领域优先推荐经过认证的用户。
激发兴趣，探索未知领域	知识科普话题：设立如“科普一下”等官方话题，降低认知门槛。
	跨领域推荐：在用户原有兴趣基础上，智能推荐关联领域的入门内容。

内容创作者：

核心需要	对应系统特性
低门槛实现创意表达，快速完成作品制作	海量正版音乐库：提供分类清晰、一键使用的正版音乐与音效。
	剪辑工具集：添加字幕、添加背景音乐、添加特效、裁剪视频等辅助工具。
	实时美颜与滤镜：提供多档美颜调节与风格化滤镜，提升画面质感。
	“一键出片”模板：提供海量创意模板，替换素材即可快速生成优质视频。
获得社交正反馈，积累粉丝与影响力	实时数据看板：清晰展示作品点赞、评论、转发、涨粉等核心数据。
	粉丝分层管理：支持对粉丝进行分组、标记，实现精细化社群运营。
	私信互动管理：提供高效的私信收发与批量管理功能。
获取创作灵感，紧跟行业趋势	热门挑战榜单：实时展示平台最热门的挑战与话题。
	潮流内容推荐：根据创作者领域个性化推送热门内容与同行佳作。
	同款音乐聚合页：热门 BGM 及使用该音乐的高赞作品集中展示。
提高粉丝留存度与保证粉丝活跃度	直播功能：支持与粉丝实时互动。
	特殊标识回复评论：博主回复粉丝评论带有特殊标识。

管理类：

核心需要	对应系统特性
搭建健康内容生态，保障平台	审核任务管理后台：支持任务批量分配、优先级设置与审核

核心需要	对应系统特性
合规运转	员绩效统计。
扶持优质创作者，提升创作者留存	创作数据分析工具：为创作者提供作品、粉丝、收益等多维度数据分析。
提升用户活跃度与平台增长	热点活动策划工具：支持快速创建、发布和推广线上挑战赛等运营活动。
	用户分层运营模块：基于用户行为标签，实现精准的 Push 与消息触达。
	流量分发调控中心：可对推荐算法策略进行人工干预与权重调整。
优化商业化转化效率	广告投放管理后台：管理广告位库存，监控填充率等核心指标。
	电商转化数据监测：实时跟踪从内容曝光到商品成交的全链路数据。
	品牌合作管理系统：管理合作项目流程，并评估项目 ROI。

业务处理类：

（1）审核人员

核心需要	对应系统特性
高效完成合规审核，降低工作负荷	智能预审与分发：系统自动预筛并标记高风险内容，提升审核效率。
	批量处理功能：支持对同类低风险内容进行一键通过操作和一键拒绝操作。
	审核 workflow 引擎：根据内容类型和风险等级自动分配任务队列。
确保审核标准统一，减少判断偏差	实时规则查询手册：提供在线、可搜索的详细审核规则与案例库。
	审核结果抽样复核：系统自动对审核结果进行抽样，由质检员进行复核。
及时预警高风险内容，保障审核准确性	敏感话题实时预警：对突发热点事件及相关内容进行自动标记与提权。

（2）客服

核心需要	对应系统特性
快速响应用户诉求，提升问题处理效率	智能快捷回复模板：针对常见问题提供标准化回复模板，一键发送。
同步用户反馈，助力产品优化	用户反馈标签化收集：支持为每一条用户反馈打上问题类型与优先级标签。
	反馈数据看板：统计高频问题、用户满意度趋势，并生成周期性报告。

2.6. 特性优先级

编号	特性	优先级
1	精准推荐算法：基于用户兴趣与行为，持续推送感兴趣的短视频。	Must
2	全屏沉浸体验：上下滑动无缝切换视频。	Must
3	多元互动工具：提供关注、点赞、评论、私信等功能	Must
4	粉丝团体系：支持用户加入专属粉丝团。	Should
5	热点聚合页面：实时展示平台最热门的挑战与话题。	Should
6	创意模板库：提供海量同款音乐、特效和拍摄模板，降低参与门槛。	Must
7	垂直知识库：按用途、技能等分类聚合深度内容。	Should
8	精准搜索筛选：支持关键词搜索，并可按最新、最热等维度筛选。	Must
9	“干货”标识系统：对知识密度高的视频进行标记，便于用户识别。	Must
10	合集/列表功能：创作者可将系列内容整理成合集，支持连续观看。	Must
11	观看进度跟踪：自动记录在合集中的观看进度，支持断点续看。	Must
12	收藏功能：收藏重要或喜欢的内容，并可分类收藏内容。	Must
13	笔记功能：支持在看视频时记录重要时间戳，为该视频添加对应文本内容。	Could
14	创作者认证体系：对教育、医学等领域的专业人士进行身份认证。	Must
15	事实核查机制：与权威机构合作，对热门知识类视频进行事实标注。	Won't
16	优质创作者推荐：在相关领域优先推荐经过认证的用户。	Could

17	提供正版音乐库：提供分类清晰、一键使用的正版音乐与音效。	Must
18	剪辑工具集： 添加字幕、添加背景音乐、添加特效、裁剪视频等辅助工具。	Must
19	实时美颜与滤镜： 提供多档美颜调节与风格化滤镜，提升画面质感。	Must
20	“一键出片”模板： 提供海量创意模板，替换素材即可快速生成优质视频。	Could
21	实时数据看板： 清晰展示作品点赞、评论、转发、涨粉等核心数据。	Must
22	粉丝分层管理： 支持对粉丝进行分组、标记，实现精细化社群运营。	Must
23	私信互动管理： 提供高效的私信收发与批量管理功能。	Must
24	同款音乐聚合页： 热门 BGM 及使用该音乐的高赞作品集中展示。	Must
25	商品挂载功能： 支持在视频和直播中挂载商品，直接引导销售。	Must
26	直播带货工具集： 提供直播间商品橱窗、优惠券、抽奖等营销工具。	Must
27	品牌合作接单平台： 为创作者与品牌方提供官方、安全的合作对接渠道。	Must
28	广告分成的接口： 创作者参与流量分成，从平台广告收入中获益。	Could
29	智能审核系统： 应用 AI 模型对文本、图像、视频进行违规内容预筛查。	Could
30	审核任务管理后台： 支持任务批量分配、优先级设置与审核员绩效统计。	Must
31	创作者成长体系： 设计等级与权益，对应不同的流量扶持与工具特权。	Should
32	创作数据分析工具： 为创作者提供作品、粉丝、收益等多维度数据分析。	Must
33	流量分发调控中心： 可对推荐算法策略进行人工干预与权重调整。	Must
34	用户分层运营模块： 基于用户行为标签，实现精准的 Push 与消息触达。	Could
35	广告投放管理后台： 管理广告位库存，监控填充率等核心指标。	Should

36	电商转化数据监测： 实时跟踪从内容曝光到商品成交的全链路数据。	Should
37	品牌合作管理系统： 管理合作项目流程，并评估项目投资回报率。	Won't
38	批量处理功能： 支持对同类低风险内容进行一键通过操作和一键拒绝操作。	Should
39	审核工作流引擎： 根据内容类型和内容标签自动分配任务队列。	Should
40	实时规则查询手册： 提供在线、可搜索的详细审核规则与案例库。	Must
41	审核结果抽样复核： 系统自动对审核结果进行抽样，由质检员进行复核。	Should
42	敏感话题实时预警： 对突发热点事件及相关内容进行标记，并上报。	Could
43	智能快捷回复模板： 针对常见问题提供标准化回复模板，一键发送。	Must
44	用户反馈标签化收集： 支持为每一条用户反馈打上问题类型与优先级标签。	Should
45	反馈数据看板： 统计高频问题、用户满意度趋势，并生成周期性报告。	Should

2.7. 其他产品需求

类别	需求描述
性能需求	1. 响应速度： 95%的视频请求应在 1 秒内开始播放。 2. 流畅性： App 主界面滑动及视频播放帧率应稳定在 60fps。 3. 并发能力： 系统需支持百万级日活用户的并发访问与内容上传。
可用性需求	1. 直观易用： 新用户无需教程即可完成首次视频发布。 2. 一致性： 全平台保持统一的 UI/UX 设计语言和交互逻辑。 3. 无障碍： 支持系统级字体大小调整，关键元素有足够的对比度。
可靠性需求	1. 系统稳定性： App 崩溃率低于 0.1%。 2. 数据持久性： 用户数据与创作内容需有 99.9%的可靠性保障。
安全性需求	1. 数据安全： 用户密码加密存储，通信链路全程加密。 2. 内容安全： 具备反垃圾、反作弊、反爬虫机制。 3. 隐私保护： 严格遵循隐私法规，提供隐私设置开关，明确告知数据用途。
兼容性需求	1. 系统版本： 支持 Android 7.0 及以上版本。 2. 机型适配： 适配市场主流品牌及全面屏、刘海屏等特殊屏幕。

第3章 用况建模

3.1. 术语表

名称	定义
粉丝记录	该网民的粉丝。
关注记录	该网民关注的网民的集合。
标签页	应用内按不同规则筛选与展示内容的导航页面。

3.2. 参与者清单

1. 识别主参与者

用户类型	参与者
内容消费者、内容创作者	网民
业务处理者	内容审核员，客服
管理类	运营人员

2. 参与者简要描述

2.1. 网民

网民可以在“乐拍视界”浏览、社交，创作和发布视频。

2.2. 内容审核员

内容审核员负责审核所有标准用户在“乐拍视界”上发布的视频内容和评论，对违规内容进行处理。

2.3. 客服

当网民在使用系统时出现的问题，将发送消息给客服。客服负责通过软件接收消息，并与网民沟通。

2.4. 运营人员

负责内容生态搭建、创作者扶持与管理，统筹内容审核、流量运营及用户维护，保障平台有序运转与持续增长。

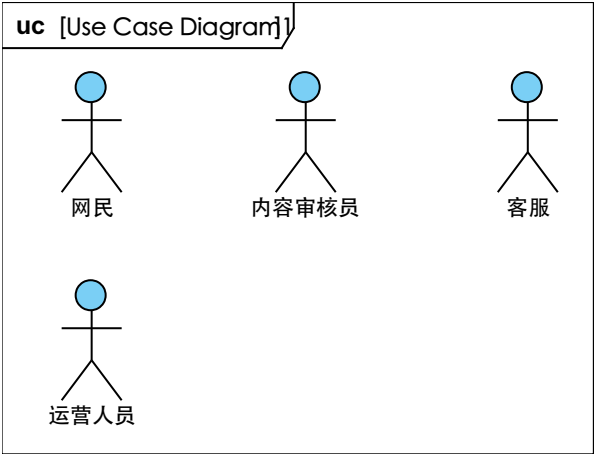


图 3-2“乐拍视界”中的参与者

3.3. 乐拍视界的主要用况

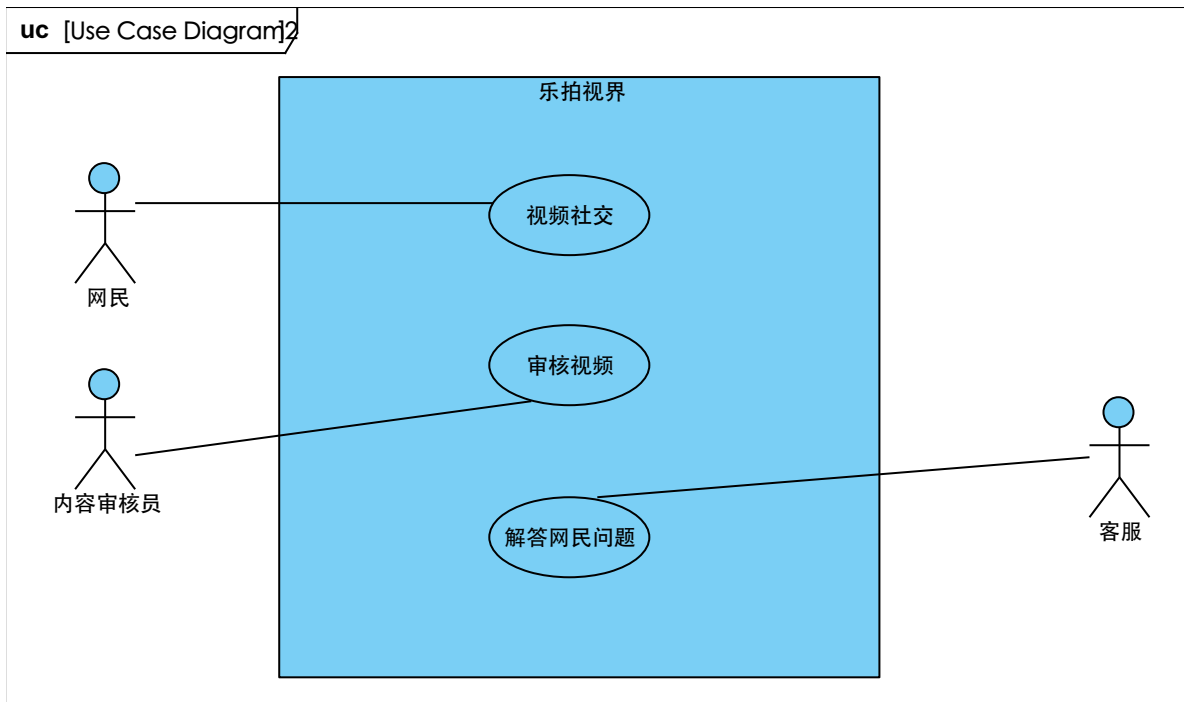


图 3-3“乐拍视界”中的主要用况

1.1. 视频社交

本用况描述了网民如何通过“乐拍视界”平台，以音视频为核心媒介，进行一系列的社交活动。这包括浏览，创作，上传视频内容、通过视频引发社交互动如点赞。

1.2. 审核视频

本用况描述了内容审核员如何通过“乐拍视界”系统，对网民上传的视频内容进行审核。包括驳回视频内容，通过审核。

1.3. 解答网民问题

本用况描述了客服如何通过“乐拍视界”系统，对网民发送的问题进行解答。

3.4. 视频社交用况的描述

1. 完全描述

1.1. 前置条件

- 1.参与者网民的设备成功连接到服务器。
- 2.网民成功登录系统。

1.2. 基本流

{启动用况}

- 1.参与者网民打开软件，进入推荐视频页，用况开始。

{网民选择操作}

- 2.系统播放推荐的视频。

3.网民可选择以下操作：

- a.滑动切换视频，执行步骤 4。
- b.点赞视频，执行步骤 5。
- c.关注其他网民，执行步骤 6。
- d.选择视频**标签页**，执行步骤 7。
- e.发布视频，执行步骤 8。
- f.网民选择退出程序，系统用况结束。

{切换视频}

4.若网民向上/下滑动视频，系统播放上/下一条播放的视频,事件流回到步骤 3。

{点赞视频}

5.网民在浏览视频界面，点赞视频，系统更新该视频的点赞数,事件流回到步骤 3。

{关注其他网民}

6.网民在浏览视频界面，关注当前视频的网民，系统更新该网民的**关注记录**，被关注网民的**粉丝记录**,事件流回到步骤 3。

{浏览被关注网民的视频}

7.网民选择：

a.推荐视频**标签页**：

a1：如果当前在推荐视频**标签页**，则系统刷新推荐视频。

a2：如果当前在其他视频**标签页**，则系统切换到推荐视频页，并加载和播放推荐视频。网民可继续执行步骤 3。

b.关注视频**标签页**：

b1：如果当前在关注视频**标签页**，则系统刷新当前页面视频。

b2：如果当前在其他视频**标签页**，则系统切换到关注视频页，并加载和播放该网民关注的网民的视频。网民可继续执行步骤 3。

8.网民切换到视频拍摄界面：

- a.网民点击开始拍摄，执行步骤 9。
- b.网民选择加载本地视频，执行步骤 10。

{拍摄视频}

9.网民完成拍摄视频，系统留存视频。

{编辑视频}

10.网民进入视频编辑界面，选择裁剪视频，添加背景音乐。

11.系统处理视频并播放处理后的视频。

{填写视频信息}

12.网民选择发布视频

13.系统显示填写信息框。

14.网民填写视频描述。

{确认发布视频}

15.网民选择确认发布。

16.系统上传视频数据与视频描述。

17.事件流回到步骤 3。

1.3. 备选流

A1.加载视频失败

在**{切换视频}**处，如果视频加载失败，则

1.系统的视频浏览页面显示是否重新加载视频

2.网民选择

a.浏览下一个视频，执行步骤 3。

b.重新加载视频，执行步骤 4。

3.系统加载下一个视频并播放。事件流回到基本流步骤 3。

4.系统重新加载当前视频并播放。事件流回到基本流步骤 3。

第4章 需求分析

4.1. 健壮性分析

1. 视频社交用况

用况描述见第三章 3.4.4 小节

1.1. 获取要播放的视频：

(1) 通信图：

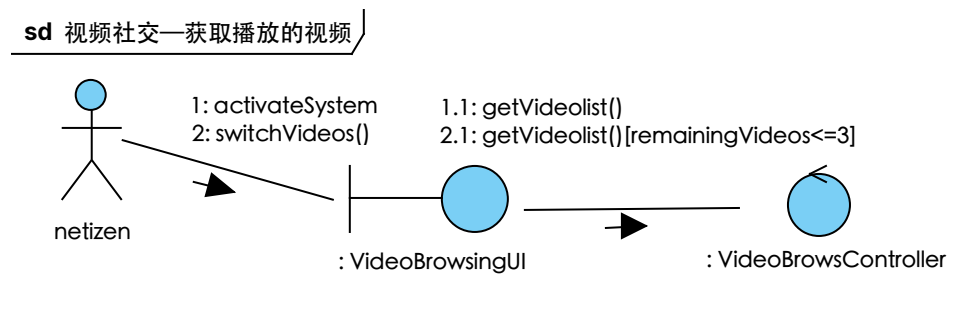


图 4-1 浏览视频通信图

(2) 从图中确认类：

边界类：VideoBrowsingUI

控制类：VideoBrowsController

1.2. 点赞视频

(1) 通信图：

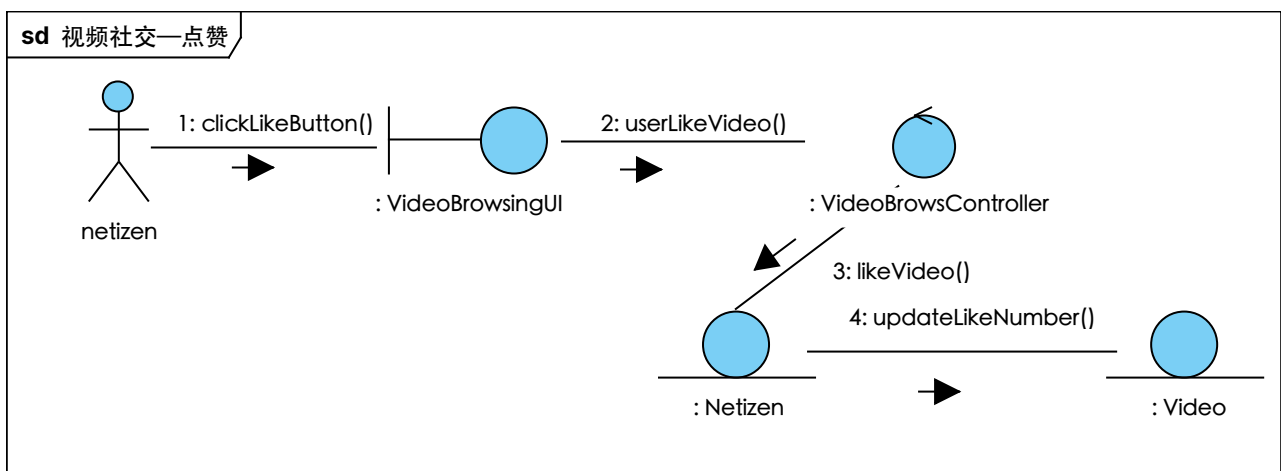


图 4-2 点赞视频通信图

(2) 从图中确认类：

实体类：Video, Netizen

边界类：VideoBrowsingUI

控制类：VideoBrowsController

1.3. 关注其他网民

(1) 通信图：

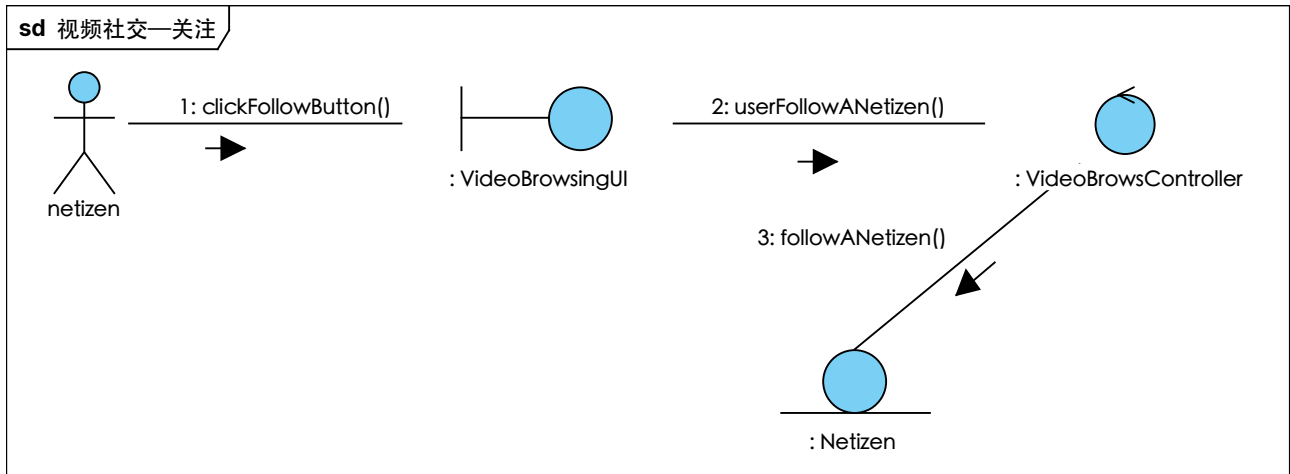


图 4-3 关注其他用户通信图

(2) 从图中确认类：

实体类：Netizen

边界类：VideoBrowsingUI

控制类：VideoBrowsController

1.4. 浏览被关注的网民的视频

(1) 通信图：

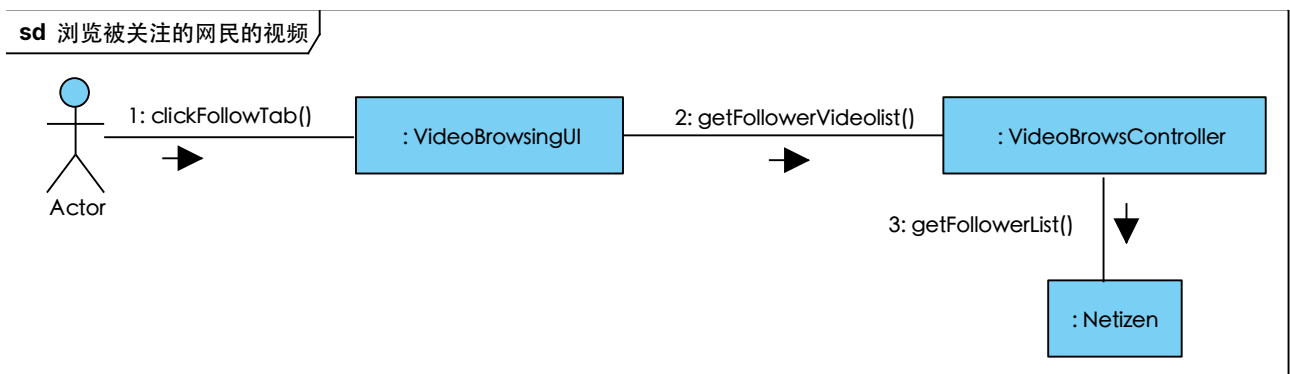


图 4-4 关注其他用户通信图

(2) 从图中确认类：

实体类：Netizen

边界类：VideoBrowsingUI

控制类：VideoBrowsController

1.5. 拍摄视频

(1) 通信图：

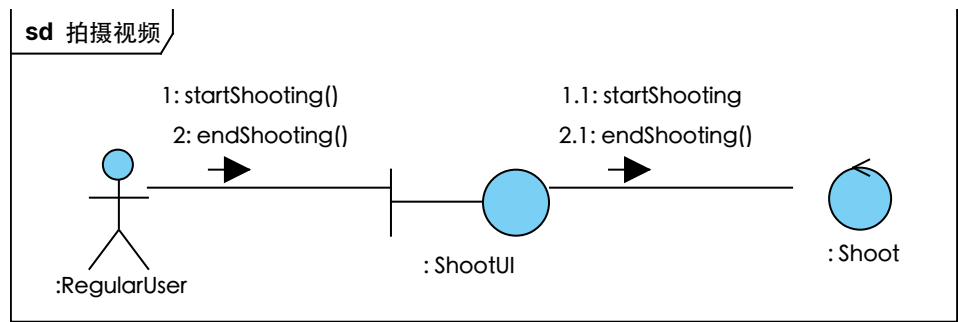


图 4-5 拍摄视频通信图

(2) 从图中确认类：

边界类：ShootUI

控制类：ShootController

1.6. 编辑视频

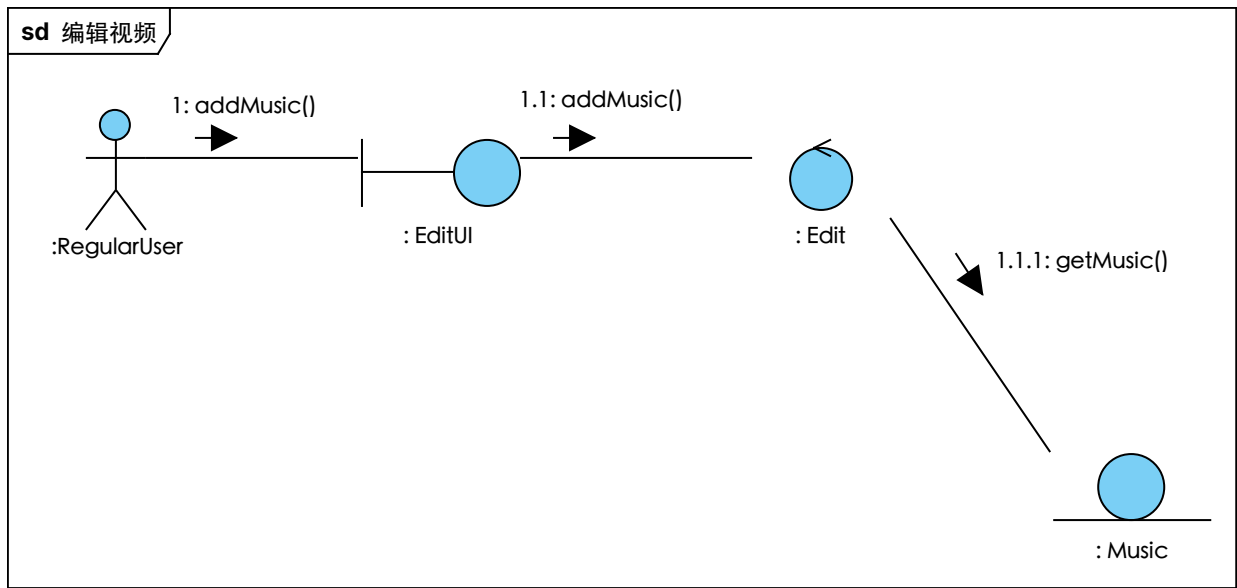


图 4-6 编辑视频通信图

从图中确认类：

实体类：Music

边界类：EditUI

控制类：EditController

1.7. 上传视频

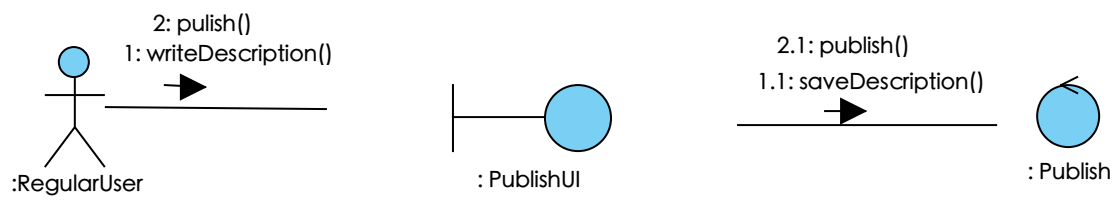


图 4-7 上传视频通信图

从图中确认类：

边界类：PublishUI

控制类：PublishController

4.2. 交互建模

1. 视频社交

1.1. 顺序图

（1）视频社交总顺序图

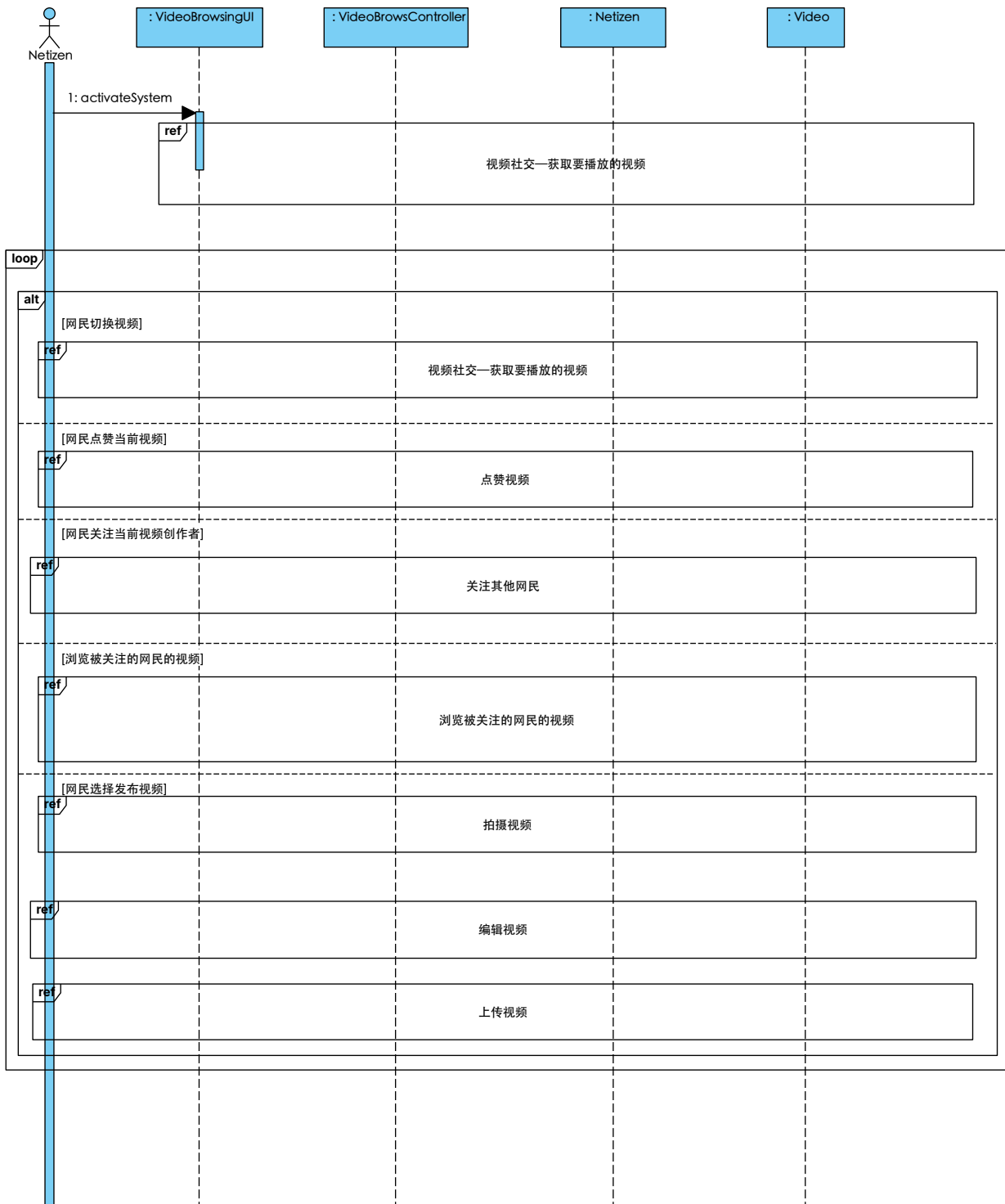


图 4-8 视频社交基本流顺序图

(2) 视频社交子流——更新视频播放列表

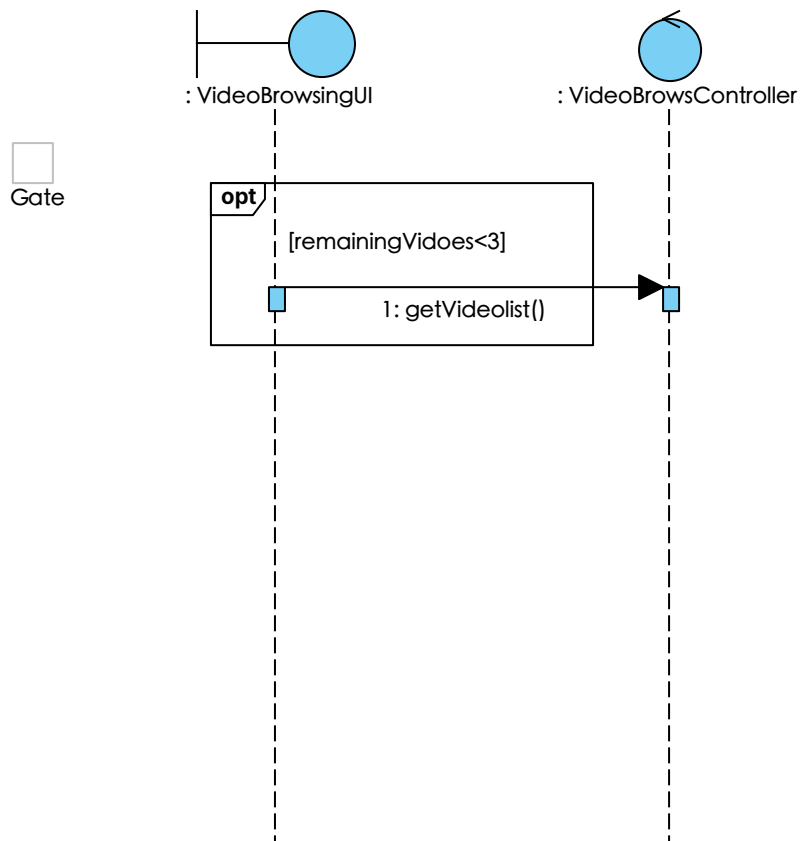


图 4-9 视频社交基本流子流—更新视频播放列表顺序图

(3) 视频社交备选流——点赞视频

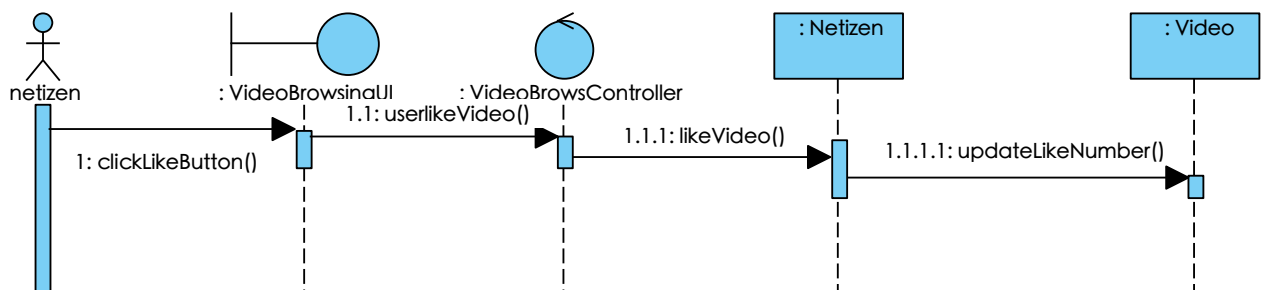


图 4-10 视频社交—点赞视频顺序图

(4) 视频社交——关注其他用户

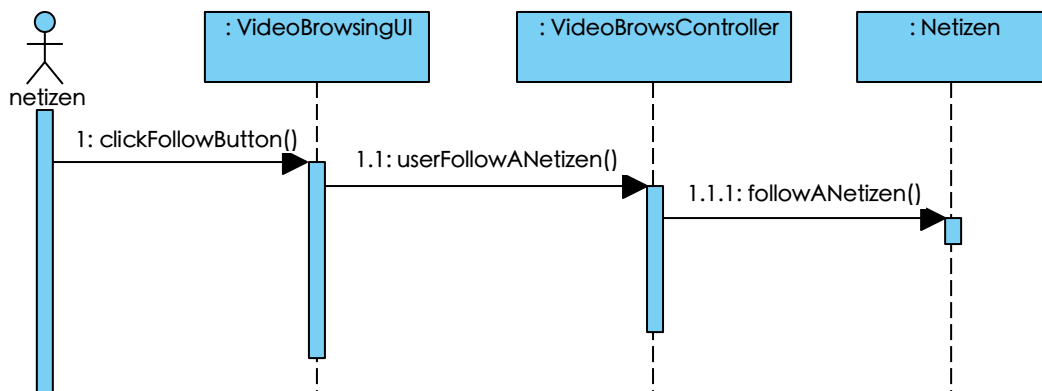


图 4-11 视频社交—关注其他用户顺序图

(1) 拍摄视频

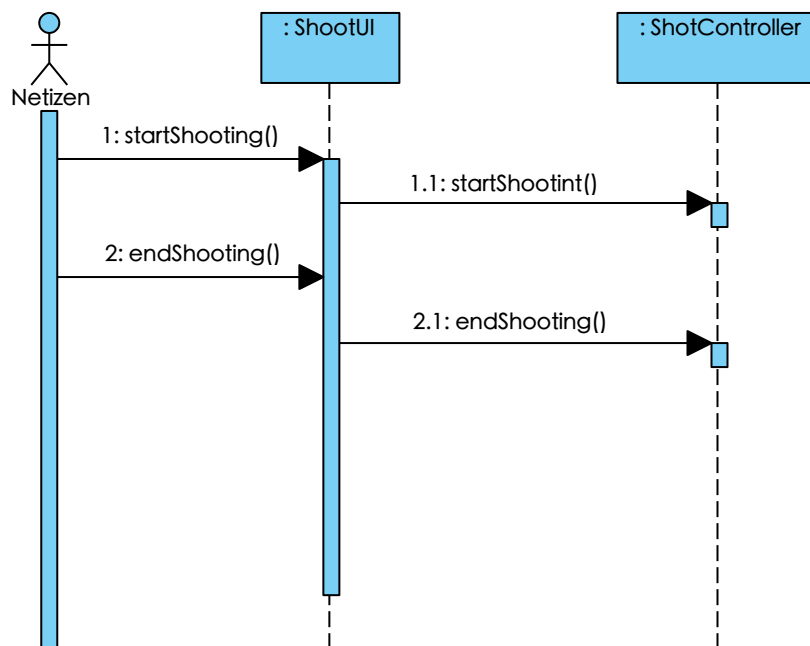


图 4-12 拍摄视频的顺序图

(2) 编辑视频

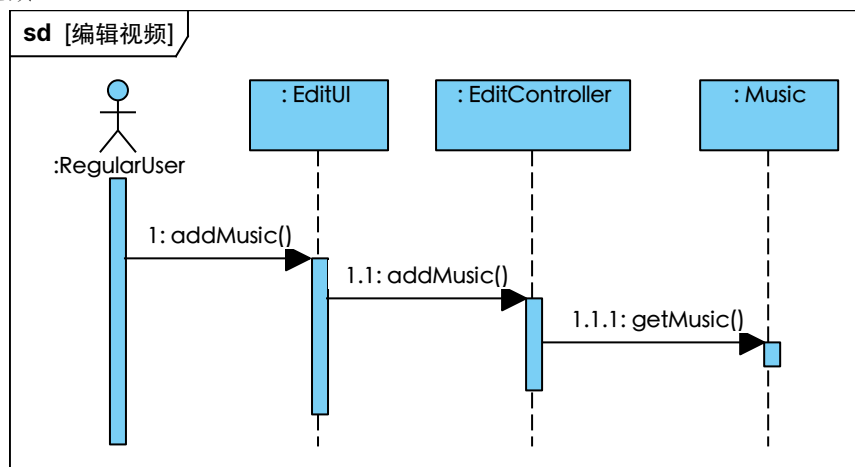


图 4-13 编辑视频的顺序图

(3) 上传视频

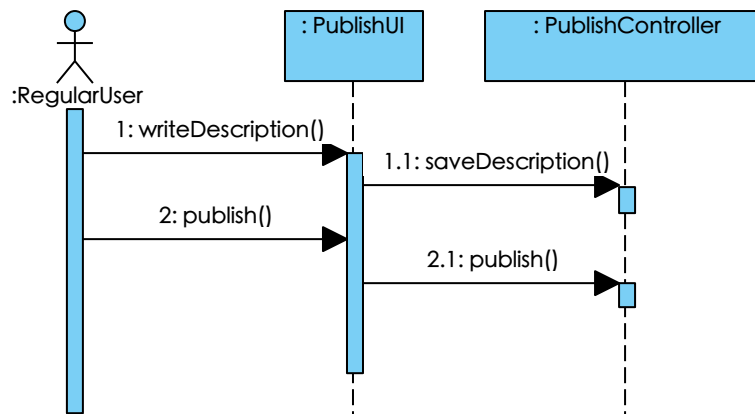


图 4-14 上传视频的顺序图

4.3. 总类图:

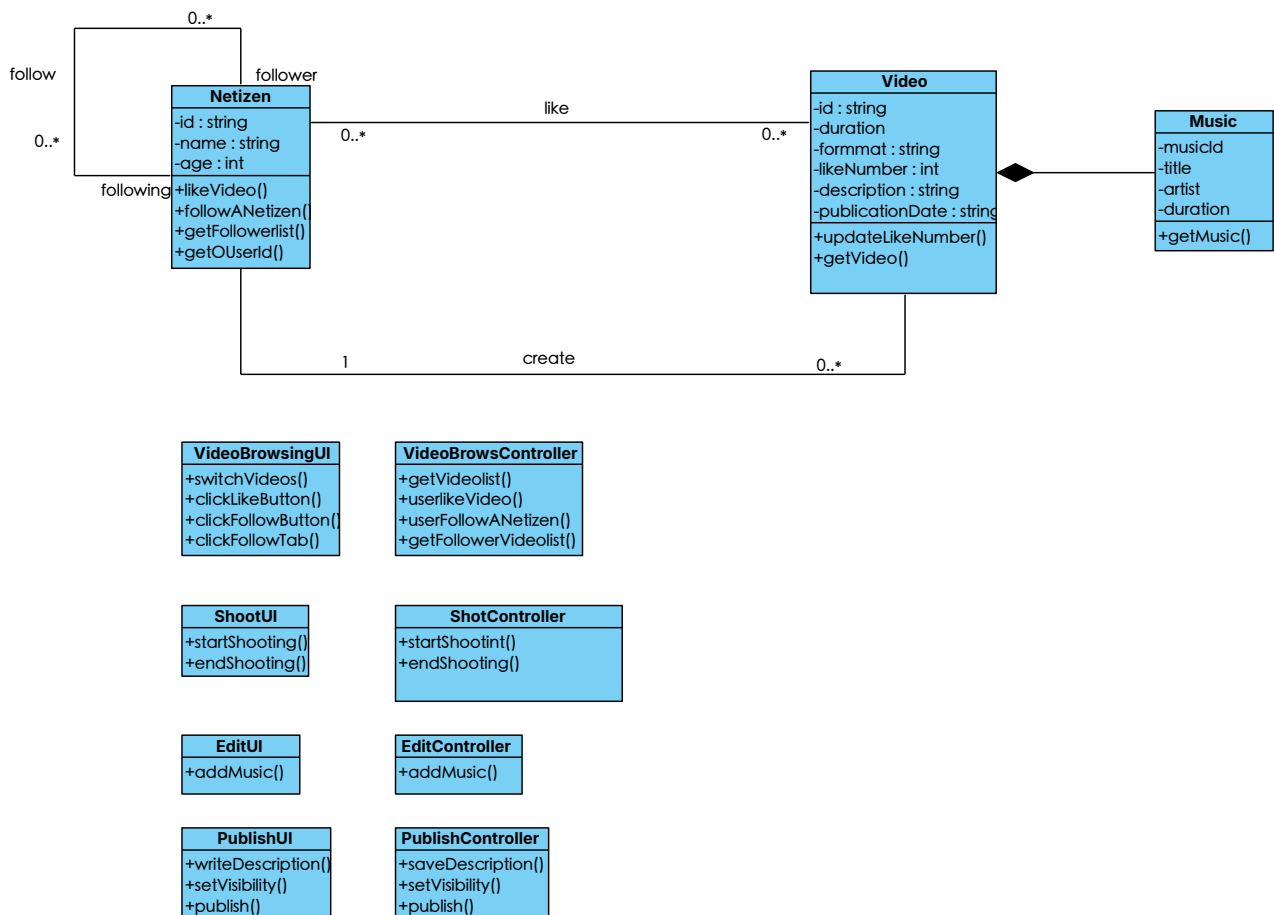


图 4-15 总类图

第5章 架构设计

5.1. 设定权衡优先级

为了确保系统架构能够最大限度地支持产品的核心价值——即“极致流畅的视听体验”与“活跃的社区互动”，我们基于商业目标和非功能性需求，制定了以下设计目标的优先级指导原则。

1. 核心设计目标的优先级排序

我们确立了“性能与可用性优先，可扩展性次之，适度牺牲数据强一致性与初期成本”的总体架构策略。具体优先级排序如下：

第一优先级（最高权重）：性能与可用性

定义：系统必须对用户的操作做出即时响应。视频播放必须达到“秒开”标准，滑动切换无卡顿；视频拍摄与特效渲染必须实时完成，且界面交互必须直观、流畅。

理由：“乐拍视界”面向的是耐心极低、追求感官刺激的年轻用户群体。任何显著的延迟或交互上的阻碍都会直接导致用户流失。

第二优先级（高权重）：可扩展性与安全性

定义：架构必须支持从初期的种子用户快速平滑扩展至百万级日活用户；同时必须保障用户隐私数据安全及内容合规。

理由：短视频产品具有爆发性增长的特征，系统必须能够通过增加硬件资源来线性提升处理能力（水平扩展），以应对突发流量。同时，内容合规是平台长期运营的红线。

第三优先级（中权重）：可维护性与上市时间

定义：代码结构应清晰，便于后续迭代；系统需在6-12个月内完成核心功能上线。

理由：虽然长期维护很重要，但在激烈的市场竞争初期，抢占市场窗口期更为关键。我们可以接受在初期引入少量的“技术债务”，以换取核心功能的快速交付。

第四优先级（低权重）：数据强一致性与运营成本

定义：各个节点的数据必须在同一时刻保持绝对一致；硬件和带宽资源的节省。

理由：在社交场景下，用户对点赞数、浏览量的实时精确性不敏感，因此可以采用“最终一致性”模型。此外，为了保障第一优先级的“性能”，我们愿意在初期投入较高的带宽和CDN成本。

2. 具体的架构权衡决策

基于上述优先级排序，我们在系统设计的关键领域做出了以下具体的权衡决策：

2.1. 空间换时间

决策描述：为了满足高性能的视频播放需求，我们选择消耗更多的存储空间和内存资源。

具体措施：

多版本转码存储：服务器端不仅存储视频原片，还预先转码生成多种分辨率、多种码率的视频副本。

激进的缓存策略：在移动端本地和 CDN 节点大量使用缓存技术。

2.2. 可用性优于一致性

决策描述：根据 CAP 理论，在分布式系统中，面对网络分区时，我们优先保障可用性。

具体措施：

最终一致性设计：对于点赞、评论计数、粉丝数等高频并发数据，系统不保证所有用户在同一毫秒看到的数据是完全一致的。我们允许数据在短时间内存在差异，通过异步消息队列进行后端处理，确保数据在最终状态下是一致的。

降级策略：当推荐算法服务不可用或响应超时，系统将自动降级为加载本地缓存或通用的热门列表。

2.3. 功能分级与快速迭代

决策描述：为了满足上市时间的要求，我们对功能进行严格分级，牺牲部分非核心功能的上线时间。

具体措施：

MVP（最小可行性产品）策略：首个版本集中资源打磨“拍摄-剪辑-发布-浏览”的核心闭环。对于复杂的“即时通讯（IM）高级功能”、“多级分销系统”等，推迟到后续版本迭代。

使用现成而非自研：对于非核心竞争力的功能模块（如美颜算法库、即时通讯底层通道），优先使用成熟的第三方服务，而不是投入团队自研，以缩短开发周期。

5.2. 估算系统性能

在系统架构设计的早期阶段，我们基于“乐拍视界”的中期业务目标——即日活跃用户（DAU）达到 100 万的规模，对系统的核心性能指标进行了粗略估算。考虑到互联网流量的突发性，我们在计算结果的基础上增加了 2 至 3 倍的冗余量，以确保架构足以应对高峰负载和营销活动带来的流量冲击。

1. 基础假设与模型参数

为了进行有效的估算，我们设定了以下基础业务模型参数：

日活跃用户（DAU）：1,000,000 人。

用户平均在线时长：30 分钟。

平均观看视频数：假设每位用户每天观看 30 个短视频（含完整观看与快速划过）。

内容生产比例：假设 1% 的用户每天上传 1 个视频（即每天新增 10,000 个视频）。

互动率：假设用户对观看的视频中有 5% 进行点赞操作。

峰值时段：假设全天 80% 的流量集中在 4 小时的晚间高峰期（20:00 - 24:00）。

2. 计算吞吐量

2.1. 视频浏览

日总请求量： $1,000,000 \text{ 用户} \times 30 \text{ 视频} = 30,000,000 \text{ 次请求/天}$ 。

平均 QPS： $30,000,000 / (24 \times 3600) \approx 347 \text{ QPS}$ 。

峰值 QPS 估算：

根据二八原则与峰值时段假设： $(30,000,000 \times 0.8) / (4 \times 3600) \approx 1,666 \text{ QPS}$ 。

架构设计目标 QPS：考虑到突发热点，乘以 3 倍安全系数，系统需支持 5,000 QPS 的视频流获取请求。

结论：单台应用服务器难以承担，必须采用负载均衡集群，且核心推荐接口需具备毫秒级响应能力。

2.2. 视频上传

日总上传量： $10,000 \text{ 个视频/天}$ 。

峰值上传 TPS：假设上传也集中在高峰期， $(10,000 \times 0.8) / (4 \times 3600) \approx 0.6 \text{ TPS}$ 。

架构设计目标 TPS：即使考虑到活动期间上传量翻倍，设计目标定为 5-10 TPS。

结论：虽然 TPS 数值不高，但单次上传耗时长、占用带宽大且触发后续转码流程。架构上必须采用异步处理，即上传完成后立即响应用户，转码在后台排队进行。

2.3. 点赞视频

日总互动量： $30,000,000 \text{ 浏览} \times 5\% = 1,500,000 \text{ 次互动/天}$ 。

峰值 TPS： $(1,500,000 \times 0.8) / 14,400 \approx 83 \text{ TPS}$ 。

架构设计目标 TPS：乘以 3 倍冗余，设计目标约为 250 - 300 TPS。

结论：写入压力尚可，但考虑到点赞需要实时反馈给前端，使用 Redis 等高性能缓存数据库来处理计数，随后异步持久化到关系型数据库。

3. 估算存储需求

单个视频大小：假设原视频平均 20MB，经过系统转码生成不同清晰度版本后，总存储占用约为 30MB。

日新增存储量： $10,000 \text{ 视频/天} \times 30\text{MB} = 300,000 \text{ MB} \approx 300 \text{ GB/天}$ 。

年存储需求： $300 \text{ GB} \times 365 \approx 109.5 \text{ TB/年}$ 。

结论：

存储增长迅速，无法使用单机硬盘存储。

架构必须采用分布式对象存储服务，支持无限水平扩展。

需要设计冷热数据分离策略，1 年以上的旧视频可归档至低成本存储。

4. 估算网络带宽

平均视频码率：假设流畅播放需要的平均下行流量为 5MB/视频。

峰值流量消耗：

峰值时刻每秒有 1,666 个视频被请求播放。

每秒所需带宽流量 = $1,666 \times 5\text{MB} \approx 8,330\text{ MB/s}$ 。

换算为带宽速率 = $8,330\text{ MB/s} \times 8\text{ bits} \approx 66\text{ Gbps}$ 。

结论：

66 Gbps 的瞬时流量对于自建机房是巨大的压力且成本极高。

架构决策：引入内容分发网络（CDN）。核心服务器仅负责调度和业务逻辑，95% 以上的视频流流量应由 CDN 边缘节点承担。

5. 性能估算总结与架构调整

基于上述粗略计算，我们对系统架构做出以下针对性确认：

硬件资源配置：数据库服务器不需要极其昂贵的顶配硬件，因为读写 QPS 在 RDBMS 的处理范围内，但推荐系统计算节点需要高性能 CPU/GPU 支持。

瓶颈预判：系统的主要瓶颈不在于应用服务器的并发数，而在于网络带宽和存储 I/O。

架构修正：

读写分离：鉴于读（浏览）远大于写（上传），数据库应设计为主从复制架构，多台从库负责读取。

静态资源剥离：所有视频、图片等静态资源强制走 CDN，严禁直接从应用服务器读取。

缓存策略：由于 80% 的播放集中在 20% 的热门视频上，必须在各级（客户端、CDN、服务端）建立激进的缓存策略以降低回源流量。

5.3. 选择架构风格

1. 整体系统架构：分布式客户/服务器风格

在宏观层面，系统采用经典的三层分布式架构，明确划分前端（移动设备）、应用服务层（API 网关与业务逻辑）与数据基础设施层。

客户端（胖客户端策略）：移动端采用“胖客户端”模式。利用 Qt/QML 跨平台框架构建，客户端不仅负责 UI 渲染，还承担媒体采集、本地预处理（如视频拼接）及缓存管理职责。

服务端（无状态服务集群）：服务端采用无状态（Stateless）设计。通过 Redis 集中管理 Token。

通信机制：客户端与服务端之间采用标准的 RESTful API (HTTP/1.1) 进行通信，数据交

换格式为 JSON。

2. 客户端架构：混合架构

2.1. 整体风格：MVVM (Model-View-ViewModel)

采用 数据驱动 UI 的模式。

Model (C++): 负责封装核心业务数据与网络请求逻辑。

View (QML): 负责界面的声明式布局与动画效果。

ViewModel (C++ Adapter): 作为中间适配层，通过 Qt 的信号与槽机制，将后端数据变化自动映射到前端 UI，实现逻辑与视图的彻底解耦。

2.2. 核心流媒体架构：MVD (Model-View-Delegate)

针对核心的“视频信息流”功能，采用 Qt 的 MVD 架构。

Model: QAbstractListModel 的子类，高效管理视频列表数据。

View: ListView，负责滚动与布局。

Delegate: 负责单个视频项的实例化与渲染。

优势: 利用 UI 虚拟化技术，仅渲染当前屏幕可见的视频节点 (Player)，确保在包含成千上万条视频的列表中滑动依然流畅。

3. 服务器端架构：分层与事件驱动架构

服务端基于 Drogon C++ Web 框架构建，采用高性能的 Reactor 模式。

并发模型: 异步非阻塞 I/O

采用 Event Loop 机制。少量的 I/O 线程即可处理成千上万的并发连接，数据库查询与 Redis 操作均采用异步回调方式，避免线程阻塞。

逻辑分层:

控制层 (Controller): 处理 HTTP 请求路由与参数解析。

服务层 (Service): 编排业务逻辑。

数据访问层 (Repository): 封装 SQL 语句与 ORM 操作。

4. 视频处理子系统：基于消息队列的异步任务架构

针对高耗时的视频转码与发布流程，采用 生产者-消费者模式，实现业务逻辑与计算密集型任务的彻底解耦。

生产者 (API Service): 当用户上传视频后，API 服务仅将任务 ID 推送至 Redis 消息队列 (video_queue)，并立即响应客户端，不阻塞用户界面。

消费者 (Video Worker): 独立的后台守护进程，持续轮询 Redis 队列。一旦获取任务，即启动处理流程。

执行策略：外部进程调用 (Process Invocation)

Worker 进程通过 `execvp/fork` 系统调用启动独立的 FFmpeg 子进程进行转码、切片 (HLS) 和截图。

优势：这种“进程隔离”策略极大地提高系统的稳定性——即使某个“脏”视频文件导致 FFmpeg 崩溃，也不会影响主 Worker 进程的运行。

5. 数据存储架构：主从复制与读写分离

鉴于短视频社区“读多写少”的特性（浏览量远大于发布量），存储架构设计如下：

关系型数据库 (PostgreSQL)：采用 主从复制 架构。

Master：处理所有写入操作（注册、发布、点赞）。

Slave：处理高频的读取操作（推荐流获取、用户信息查询）。

对象存储 (MinIO)：采用兼容 AWS S3 协议的分布式对象存储。

Temp Bucket：用于客户端直传原始文件。

Public Bucket：用于存储处理后的 HLS 切片与封面图，并对外提供 CDN 访问。

5.4. 将系统划分成子系统

根据物理部署边界将系统划分为两大顶级子系统：移动客户端子系统与服务端服务子系统。服务端进一步拆分为在线业务集群与离线计算集群，以实现资源隔离。

1. 客户端子系统的划分

移动客户端采用分层与模块化设计，向下屏蔽硬件差异，向上提供流畅交互。

1.1. 媒体采集与预处理子系统

职责：负责调用手机硬件进行视音频录制，以及对生成的媒体文件进行本地预处理。

核心功能：

拍摄控制：基于 `CaptureSession` 管理摄像头与麦克风，支持前后置切换与实时预览。

基础合成：采用 进程调用策略。通过 `QProcess` 调用移动端内置的 FFmpeg 可执行文件，执行视频裁剪与音视频轨道合并操作。

1.2. 播放与互动子系统

职责：负责沉浸式“无限流”视频的渲染与用户交互。

核心功能：

流媒体播放：基于 `MediaPlayer` 组件，支持 HLS 流的缓冲与播放，通过 `VideoOutput` 进行画面渲染。

列表管理：利用 `MVD` 架构管理视频列表数据，处理分页加载与滑动事件。

交互反馈：处理点赞、关注等用户指令，实时更新本地 UI 状态（乐观更新），无

需等待服务器返回。

1.3. 网络通信与配置子系统

职责：作为客户端与服务端的唯一通信桥梁，管理应用生命周期数据。

核心功能：

统一网络层：封装 HTTP 请求类，负责 Token 注入、请求签名及错误处理。

配置管理：ConfigManager 与 AuthManager 负责本地持久化存储，管理用户 Token、CDN 节点配置及上传断点信息。

2. 服务端子系统的划分

后端采用微服务化的拆分思想，将 I/O 密集型业务与 CPU 密集型业务在物理上分离。

2.1. API 网关与业务子系统

物理组件：Nginx (Gateway) + api_service (C++ Binary)。

职责：处理高并发的客户端 HTTP 请求，执行核心业务逻辑。

核心功能：

用户模块：处理注册、登录、鉴权及用户信息管理。

社交模块：处理点赞、关注逻辑，以及 Feed 流的分发。

发布调度：接收视频发布请求，写入数据库元数据，并将转码任务推送到 Redis 消息队列，不执行具体的转码操作。

2.2. 异步媒体处理子系统

物理组件：video_worker (C++ Daemon) + FFmpeg (Executable)。

职责：作为后台守护进程，消费计算密集型的视频处理任务。

核心功能：

任务消费：持续轮询 Redis 的 video_queue 队列，获取待处理视频 ID。

进程隔离执行：通过系统调用 (fork/execvp) 启动独立的 FFmpeg 子进程，执行视频切片 (HLS)、封面截取和元数据提取。

状态回调：处理完成后，将成品上传至对象存储，并更新数据库中的视频状态为“已发布”。

2.3. 数据与存储基础设施

职责：提供持久化存储与高速缓存服务。

组件划分：

核心存储：PostgreSQL 集群（主从架构），存储用户关系与视频元数据。

对象存储：MinIO 集群，物理存储视频文件、切片与图片。

高速缓存/队列：Redis，用于 Token 会话存储、热门视频缓存及转码任务队列。

3. 子系统间的依赖与通信关系

客户端 -> 服务端：仅通过 HTTP/RESTful API 进行同步通信。

API 服务 -> 媒体 Worker：通过 Redis 消息队列 进行单向、异步通信，实现完全解耦。
API 服务只负责“生产任务”，Worker 只负责“消费任务”。

服务端 -> 基础设施：

API 服务通过 TCP 连接池访问 PostgreSQL 和 Redis。

Worker 服务通过 HTTP 协议（S3 API）访问 MinIO 对象存储。

5.5. 确定问题内部的并发性

为了在保证用户体验流畅性的同时，最大化服务器吞吐量和系统稳定性，系统在不同层级采用针对性的并发设计策略。

1. 移动客户端的并发设计

客户端严格遵循“主线程负责渲染，后台处理耗时任务”的原则，确保界面（UI）永远不会冻结。

UI 主线程：

职责：仅负责 QML 界面的绘制、场景图渲染以及响应用户的点击与滑动事件。

约束：严禁在该线程执行任何阻塞性 I/O 操作。

异步网络通信：

机制：利用 Qt 的信号与槽机制实现异步网络请求。发出请求后立即返回，不阻塞调用者；当服务端响应到达时，通过信号回调触发数据更新。

媒体处理并发：

机制：音视频合并与处理采用 外部进程异步调用 策略。客户端启动独立的 ffmpeg 子进程执行任务，并通过非阻塞管道监听进程状态，避免在应用进程内进行高负载编解码导致的崩溃风险。

2. 服务端 API 服务的并发设计 (Reactor 模型)

API 服务端采用高性能的 Reactor 事件驱动模型。

I/O 事件循环：

系统维护少量的 I/O 线程。所有 HTTP 请求的接入、解析以及数据库/Redis 的读写操作均以 非阻塞方式注册到事件循环中。

异步回调链：

业务逻辑通过 Lambda 回调函数 串联。当数据库查询或网络 I/O 完成时，事件循环自动触发回调继续执行后续逻辑。

3. 后台转码服务的并发设计

针对计算密集型的视频转码任务，采用 轮询调度与进程级并发 相结合的策略。

任务获取：

Worker 服务通过定时器以低频率轮询 **Redis** 任务队列，实现“拉模式”的任务获取，防止突发流量压垮转码节点。

并发控制：

Worker 内部维护一个受控的并发计数器。每个转码任务在独立的 子线程 中管理，而实际的编解码计算则下发给 独立的系统进程 (FFmpeg) 执行。

4. 对象状态与互斥管理

在并发环境下，系统通过以下策略保证数据的一致性与完整性：

高频计数器：

对于视频点赞数等高频并发写操作，不直接锁数据库行，而是利用 **Redis** 的 原子递增 (INCR/DECR) 操作在内存中处理，随后通过定时任务异步同步至数据库，消除数据库写入瓶颈。

关键业务事务：

对于“关注/取消关注”等涉及多表数据一致性的操作，在数据库层面开启 事务，确保操作的原子性，要么全部成功，要么全部回滚。

本地资源锁：

在 **Worker** 多线程处理任务队列时，使用互斥锁保护本地的任务队列状态，防止多线程同时抢占同一个上传任务。

5.6. 配置子系统的硬件

系统采用 3 台物理机 模拟真实的互联网分布式拓扑。

1. 物理节点分类与配置

1.1. 移动终端节点

部署子系统： 媒体采集与处理子系统、播放与互动子系统、本地数据管理子系统。

硬件特征：

设备类型： 用户持有的移动智能手机。

关键资源： 高清摄像头与麦克风阵列（用于采集）、GPU/NPU、本地闪存（用于草稿箱存储）。

配置要求： 系统向下兼容至 **Android 7.0** 及主流中低端机型，但在高端机型上开启高帧率（60fps）录制与高清播放模式。

1.2. 服务终端节点

节点别名	角色	部署组件	职责描述
PC-1 (The Vault)	数据中心	PostgreSQL(Master/Slave), Redis, MinIO	负责所有数据的持久化存储，IO 密集型节点。
PC-2 (The Brain)	应用服务	Nginx(Gateway), API Service	负责核心业务逻辑运算，CPU 密集型节点。
PC-3 (The Muscle)	边缘计算	FFmpeg Worker, Nginx(CDN)	负责视频转码重活及流媒体分发，兼具计算与带宽压力。

2. 网络连接

为了保障数据传输的效率与安全性，硬件节点之间的连接遵循以下拓扑规则：
外部接入网络：

移动终端节点通过 4G/5G/Wi-Fi 网络连接互联网。

服务终端所有节点处于同一局域网段，通过 Docker Network (Bridge 模式) 或端口映射进行通信。API 服务通过 IP 直连数据库与缓存节点。

安全组策略： 数据存储节点不分配公网 IP，仅允许应用服务节点通过特定端口访问，物理隔离外部攻击。

3. 硬件冗余与故障转移

为了满足高可用性需求，硬件配置实施了冗余策略：

主从热备： 数据库节点采用“一主多从”配置，主节点故障时，监控系统自动将从节点提升为主节点。

5.7. 管理数据存储

系统根据数据的类型和访问特征，选用三种异构存储组件进行组合管理。

1. 非结构化数据：MinIO 分布式对象存储

采用兼容 AWS S3 协议的 MinIO 集群存储所有媒体文件，通过分桶策略隔离原始素材与发布内容。

临时存储桶 (temp):

用途：用于客户端直传拍摄的原始视频文件。

策略：设置为允许公开上传。该桶作为“中转站”，文件在转码任务完成后会被 Worker 自动清理，生命周期短暂。

公共发布桶 (public):

用途：存储经过转码后的 HLS 视频切片（.ts/.m3u8）、视频封面图及用户头像。

策略：设置为只读（Public Read），作为 Nginx/CDN 的回源站。所有对外分发的媒体资源 URL 均指向此桶。

2. 结构化数据：PostgreSQL (应用层读写分离)

核心业务数据存储采用 PostgreSQL 的主从复制架构，并在代码层面实现读写分离路由。

主库 (Master):

职责：处理所有修改数据的操作。

场景：用户注册、视频元数据写入、点赞/关注关系的创建。

从库 (Slave):

职责：处理高频的只读查询操作。

场景：首页视频流拉取、用户个人资料查询。应用服务通过配置独立的只读客户端句柄连接从库，分担主库压力。

3. 缓存与消息队列：Redis

Redis 在架构中承担高速缓存、任务调度与数据缓冲的三重角色。

会话管理：存储用户登录 Token 与 UserID 的映射关系，设置自动过期时间（TTL），实现无状态服务的鉴权。

异步任务队列：使用 List 数据结构 (video_queue) 构建生产者-消费者管道，解耦 API 服务与视频转码服务。

高并发计数缓冲：

机制：针对点赞数等高频更新，使用 Redis 原子计数器 (INCR) 进行实时处理。

同步策略：维护一个“脏数据集” (dirty_videos)，通过后台调度器定期将 Redis 中的最新计数批量写回 PostgreSQL 数据库，实现最终一致性，避免数据库因高频更新而锁死。

5.8. 处理全局资源

1. 全局唯一标识符 (UUID)

放弃复杂的 Snowflake 算法，采用 UUID (v4) 生成策略。

应用场景：用户 ID、视频 ID、Token。

优势：C++ drogon::utils::getUuid() 原生支持，无中心化发号器依赖，适合当前集群规模。

2. 身份认证与访问控制

机制：基于 Redis 的有状态会话 (Stateful Session)。

流程：登录成功后生成 UUID Token 存入 Redis 并设置 TTL（如 30 天）。客户端请求携带 token，服务端通过 Redis 校验有效性。

3. 视频处理控制策略 (管道与过滤器)

异步处理：用户发布视频后，仅在数据库标记状态为“处理中”，立即返回响应，不阻塞客户端。

处理流：Worker 进程后台获取任务 -> 下载原片 -> FFmpeg 转码/截图 -> 上传成品 -> 回调修改数据库状态。

4. 异常处理与边界条件

数据库重连：Drogon ORM 客户端配置自动重连机制。

转码失败：Worker 捕获 FFmpeg 异常，若处理失败将数据库视频状态标记为 2 (Failed)，并清理临时文件。

冷启动：当 Redis 缓存未命中 (Cache Miss) 时，自动回源查询 PostgreSQL 从库并重建缓存。

5.9. 选择软件控制策略

系统各个子系统根据其负载特征和响应性要求，选择不同的控制流策略。

1. 移动客户端：事件驱动型控制策略

客户端的控制流完全由外部事件（用户操作、网络响应）驱动，核心机制依赖于 Qt 框架的信号与槽。

控制机制：主线程运行一个无限的事件循环。

具体应用：

UI 交互：用户的点击和滑动手势被封装为“信号”，触发对应的业务逻辑“槽函数”。

网络通信：发起 HTTP 请求后立即返回控制权给主循环，不阻塞界面。当服务端响应到达时，底层 Socket 触发 finished 信号，异步回调执行数据解析和界面更新逻辑。

2. 服务端 API 服务：异步回调与 Reactor 控制策略

为了在高并发下保持低资源消耗，API 服务端采用了 异步非阻塞 控制策略。

控制机制：基于 Reactor 模式的 I/O 事件分发。

具体应用：

请求处理：当遇到数据库查询或 Redis 操作时，控制流会注册一个 Lambda 回调函数并立即挂起当前逻辑，释放线程去处理其他请求。

流转逻辑：当 I/O 操作完成（如数据库返回结果），事件循环唤醒回调函数，恢复上下文并继续执行后续业务逻辑。

3. 视频后台处理：轮询调度与进程隔离策略

针对视频转码这种长耗时、高风险的任务，采用了主动拉取与外部执行的控制策略。

控制机制：基于 定时器的轮询调度。

具体应用：

任务获取：**Worker** 进程通过内置定时器主动去 **Redis** 队列检查是否有新任务。

执行控制：获取任务后，**Worker** 通过系统调用启动独立的 **FFmpeg** 子进程。控制流转变为等待子进程的退出状态码，以此判断任务成功或失败。

4. 异常与中断处理策略

异步错误回调：在客户端和服务端的网络交互中，错误处理通过专门的 错误回调路径来处理，确保在异步环境下的错误也能被 **UI** 正确捕获。

状态标记恢复：对于后台转码任务，若发生进程崩溃或超时中断，系统通过数据库的状态字段来标记错误边界。

5.10. 处理边界条件

系统架构定义以下三类边界条件的处理策略：初始化（**Initialization**）、终止（**Termination**）以及故障（**Failure**）。

1. 初始化

移动客户端启动流程：

配置加载与完整性检查：**App** 启动时，首先加载本地配置文件，并校验核心组件（如 **ffmpeg** 动态库）的完整性。若检测到文件损坏或版本不兼容，自动触发热修复补丁下载。

权限申请与硬件预热：在进入拍摄模块前，系统检查摄像头、麦克风及存储权限。若权限齐备，后台线程预初始化 **Camera** 硬件对象，以减少用户点击“拍摄”按钮时的等待延迟。

网络探测与会话恢复：系统自动检测当前网络环境（**Wi-Fi/4G/无网**）。同时，读取本地加密存储的 **Token** 尝试自动登录。若 **Token** 有效，立即建立 **WebSocket** 长连接以接收推送消息；若失效，则跳转至登录页。

UI 骨架屏与缓存加载：为避免启动白屏，优先从本地 **SQLite** 数据库加载上次缓存的首页视频列表和图片占位符，待网络请求返回最新数据后进行无缝替换。

服务端服务启动流程：

配置拉取：服务容器启动时，首先连接分布式配置中心，拉取最新的业务参数（如限流阈值、推荐算法权重）。

资源池预热：初始化数据库连接池和 **Redis** 连接池，建立最小空闲连接数，避免首批请求因建立 **TCP** 连接而超时。

服务注册：待所有内部组件初始化完毕且健康检查通过后，向服务注册中心注册自身 **IP** 和端口，正式对外接收流量。

2. 终止

当用户退出应用或服务器需要停机维护时，系统必须执行清理操作以释放资源并保存上

下文。

移动客户端终止：

硬件资源释放： 无论是用户主动杀掉进程还是系统因内存不足回收后台进程，**LifeCycleObserver** 均强制执行摄像头与音频驱动的释放操作，防止硬件被占用导致其他应用无法使用。

草稿自动保存： 在拍摄或编辑过程中，若用户触发退出（或来电中断），系统捕获 **onPause/onStop** 事件，将当前的视频片段、编辑进度序列化写入本地文件系统，确保用户下次打开时能恢复现场。

后台任务管理： 对于正在进行的视频上传任务，记录上传进度断点。应用退出时暂停上传，待下次启动时自动根据断点续传。

服务端停机：

流量截断： 接收到停机信号（**SIGTERM**）时，服务首先从注册中心注销，停止接收新的外部请求。

在途请求处理： 设置 **N** 秒的缓冲期，等待当前线程池中已接收的请求处理完毕。

数据落盘： 强制将内存缓冲区中的日志、埋点数据、未提交的事务进行刷盘或提交，关闭数据库与消息队列连接，最后销毁进程。

3. 故障与失效

网络异常边界：

弱网与断网： 当客户端检测到网络不可用时，**UI** 自动切换至“离线模式”，展示本地缓存的已浏览视频，并禁用点赞等在线交互功能。

上传中断： 视频上传过程中若发生网络中断，系统自动触发指数退避（**Exponential Backoff**）重试机制。若多次重试失败，标记任务为“失败”，提示用户手动重试，而不是无限循环消耗电量。

服务不可用边界（雪崩保护）：

熔断与降级： 当非核心服务（如个性化推荐引擎）响应超时或错误率超过阈值时，客户端自动触发降级策略，转为请求静态的“全站热门视频”列表，确保核心的视频播放功能不受影响。

数据库故障： 若主数据库宕机，系统自动切换至只读模式，允许用户浏览视频，但暂时屏蔽发布、点赞等写入操作，并对用户进行友好提示。

数据边界与异常值：

空状态处理（**Cold Start**）： 对于没有任何行为记录的新注册用户，推荐系统自动切换至“冷启动策略”，基于用户注册时选择的性别、年龄标签推送泛人群喜好的高热视频。

超限防护： 对于超出系统处理能力的异常输入（如上传 **10GB** 的视频文件），**API** 网关层直接进行拦截并返回明确的错误码，防止后端服务崩溃。

5.11. 制订复用计划

为了降低开发成本并提高系统稳定性，项目在框架、外部组件和内部代码三个层面制定了严格的复用策略。

1. 框架级复用

利用成熟框架提供的基础设施，避免重复造轮子。

客户端框架 (Qt 6):

控制流复用：直接复用 Qt 的事件循环和信号槽机制作为应用的核心控制中枢，实现 UI 与业务逻辑的松耦合。

渲染复用：复用 Qt Quick 的 Scene Graph 渲染引擎处理高性能 UI，通过 QML 声明式语法构建界面。

网络复用：复用 QNetworkAccessManager 模块内部的线程池和连接复用机制，处理所有 HTTP 通信。

服务端框架 (Drogon):

并发模型复用：复用 Drogon 框架底层的 Reactor 模型和非阻塞 I/O 事件循环，无需手写复杂的 Socket 通信和线程池管理代码。

ORM 复用：复用框架内置的 ORM（对象关系映射）功能。

2. 外部程序与类库复用

多媒体处理核心：FFmpeg (Executable)

复用方式：外部进程调用 (Process Invocation)。

策略：客户端和服务端 Worker 不直接链接 FFmpeg 的 C/C++ 静态库，而是通过系统调用启动独立的 ffmpeg 进程。

优势：这种“黑盒复用”极大降低集成难度，且 FFmpeg 进程崩溃不会导致主程序闪退。

加密算法库：OpenSSL

复用内容：复用其 HMAC-SHA256 和 SHA256 算法实现。

场景：用于服务端 MinioClient 计算对象存储上传接口的签名（AWS Signature V4）。

JSON 解析库：

服务端复用 JsonCpp（Drogon 内置依赖）进行高性能 JSON 序列化与反序列化。

客户端复用 QJsonDocument（Qt Core）处理 API 数据交互。

3. 内部组件复用与构建

通过提取公共模块，确保不同子系统间逻辑的一致性。

服务端公共静态库 (lepai_common)

定义：将 API 服务与 Video Worker 服务共用的逻辑抽取为独立的 CMake 静态库。

客户端网络单例 (NetworkClient)

定义：封装 HTTP 请求的公共逻辑。

复用点：统一处理 Token 注入、请求头构造、错误码解析和基础 URL 配置，供所有 UI 模块调用。

QML 通用组件

定义：将通用的 UI 元素封装为独立 QML 文件，在不同页面间实现复用。

4. 设计模式复用

单例模式 (Singleton)：确保全局配置和网络连接池的唯一性。

生产者-消费者 (Producer-Consumer)：复用 Redis 的 List 数据结构实现转码任务队列，解耦业务逻辑与后台处理。

5.12. 乐拍视界软件架构

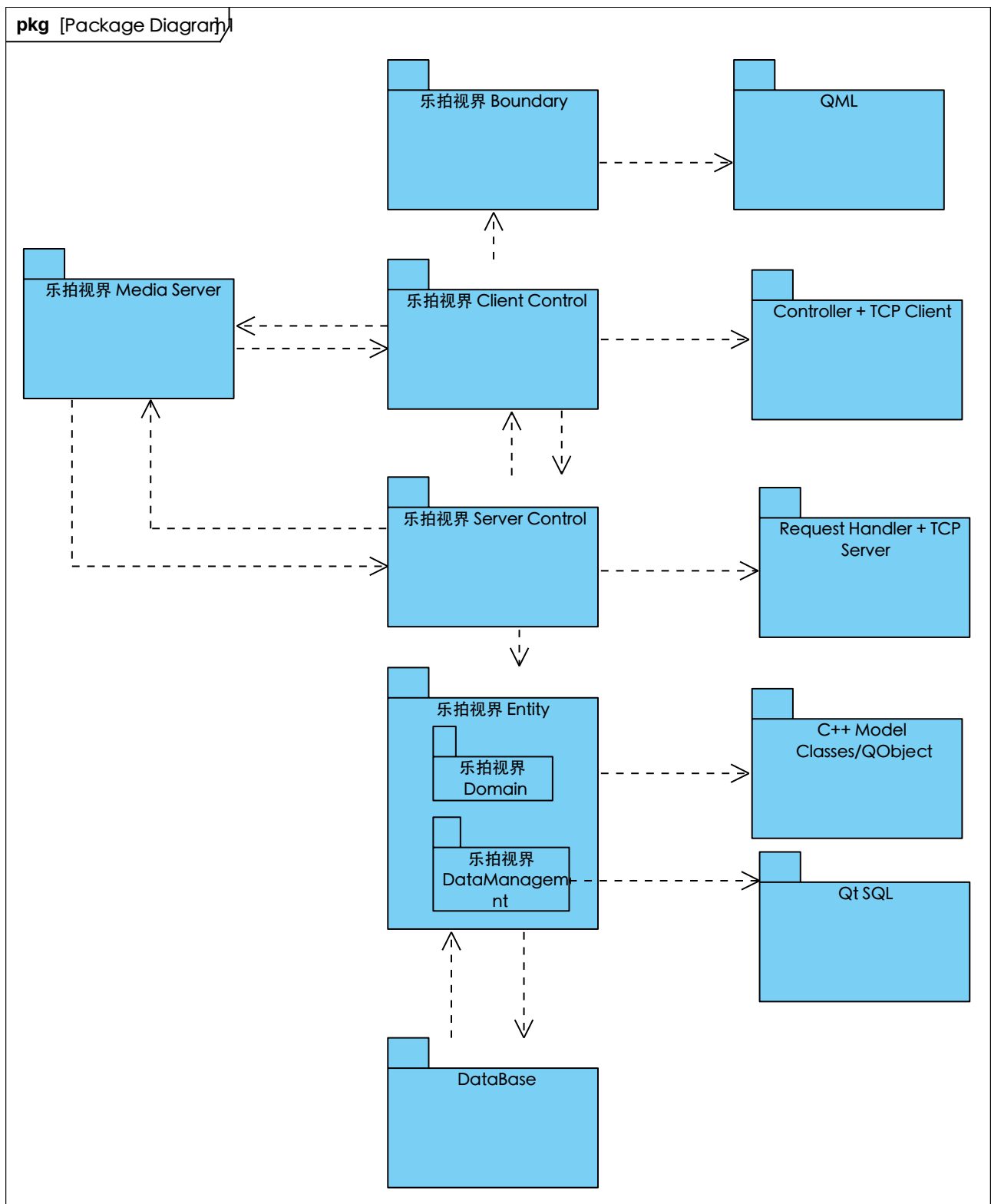


图 5-1 乐拍视界的包图

第6章 详细设计

6.1. 设计规范与标准

1. 代码命名规范

1.1. C++ 类与文件

类名：采用帕斯卡命名法（PascalCase），例如 VideoManager、UserAuthService。

源文件/头文件：采用全小写字母，例如 videomanager.cpp、userauth.h。

接口类：纯虚类或接口以 I 为前缀，例如 IVideoEncoder。

1.2. 成员变量与局部变量

私有/保护成员变量：采用 m_ 前缀加驼峰命名法（camelCase），例如 m_videoList、m_currentUserId。

局部变量与参数：采用驼峰命名法，例如 targetIndex、videoUrl。

静态成员变量：采用 s_ 前缀，例如 s_instanceCount。

1.3. 成员函数（操作）

采用驼峰命名法，动词或动宾结构，例如 calculateRecommendation、uploadFile。

获取器与设置器：严格遵循 Qt 风格，获取器直接使用属性名（如 userName()），设置器使用 set 前缀（如 setUsername()）。

1.4. QML 组件与属性

QML 文件名：帕斯卡命名法，必须与组件名一致，例如 VideoFeedView.qml。

组件 ID：驼峰命名法，建议添加组件类型后缀以提高可读性，例如 loginButton、videoListView。

自定义属性：驼峰命名法，例如 property bool isRecording。

1.5. 数据库对象

表名：全小写，复数形式，下划线分隔，例如 users、video_posts。

字段名：全小写，下划线分隔，例如 created_at、video_duration。

2. 可见性与封装标准

2.1. C++ 访问修饰符标准

Private (-)：所有非静态成员变量必须设为 private。外部对数据的访问必须通过公共接口进行。

Protected (#)：仅在设计模板方法模式或确需子类访问父类内部状态时使用。

Public (+)：仅暴露对外的业务操作接口。析构函数若非虚函数，通常应为 public（值对

象除外）。

2.2. QML 交互接口标准

Q_PROPERTY: 用于将 C++ 成员变量暴露给 QML 前端。对于只读属性，必须使用 **CONSTANT** 或仅定义 **READ** 方法，不提供 **WRITE** 接口；对于需动态更新的属性，必须定义 **NOTIFY** 信号以支持数据绑定。

Q_INVOKABLE: 用于将 C++ 业务方法暴露给 QML 调用。此类方法必须保证是非阻塞的；若涉及耗时操作，应通过开启新线程处理，并在完成后发射信号通知前端，严禁阻塞 UI 线程。

Public Slots: 用于响应 UI 事件或网络回调，可见性等同于 **public** 方法。

3. 数据类型映射标准

3.1. 基础数据类型映射

逻辑类型	C++ 实现类型	QML /JS 类型	数据库类型 (PostgreSQL)	说明
ID/标识符	qint64 (对应 int64_t)	string	BIGINT	基于雪花算法生 成的 64 位 ID
字符串	QString	string	VARCHAR/ TEXT	支持 UTF-8 多语 言字符集
时间/日期	QDateTime 或 qint64(时间戳)	Date 或 var	TIMESTAMPTZ	服务端统一存储 UTC 时间
大数值	double / qreal	double	DECIMAL	用于货币、精确 统计
二进制流	QByteArray	var	BYTEA	仅用于小文件， 大文件存 MinIo 链接
列表/数组	Qvector<T> / Qlist<T>	var / ListModel	JSONB (非关系 型数据)	列表数据通过 Model-View 机制 传输

3.2. 值对象定义

对于具有特定业务含义的数据组合，必须封装为结构体或类，而不是散乱的基本类型。

3.3. 空值与可选值处理

C++ 端使用 `std::optional<T>` 或 Qt 的 `QVariant::isNull()` 来明确表示“无值”状态，严禁使用魔术数字（如 -1 或 0）代表空值，以维护数据的完整性约束。

4. 操作签名标准

4.1. 参数列表

输入参数尽量使用 `const T&`（常量引用）以减少拷贝开销（基础类型除外）。

参数顺序应遵循“输入参数在前，输出参数在后”或“必填参数在前，可选参数在后”的原则。

4.2. 返回值

避免使用 `void` 进行可能失败的业务操作，应返回 `bool`（配合输出参数）或自定义的 `Result<T>` 类型（包含错误码和数据），以便调用方处理异常流。

4.3. 异常规范

明确标识可能抛出异常的方法。对于 QML 调用的 `Q_INVOKABLE` 方法，严禁直接抛出 C++ 异常，必须捕获并转换为错误码或信号通知前端。

4.4. Const 正确性

不修改对象内部状态的方法必须声明为 `const`，以支持在常量对象上调用，并增强接口语义的清晰度。

6.2. 类的详细设计与精化

1. 客户端类设计

1. 实体类

UserModel（用户模型类）

- 职责：封装用户核心业务数据，为 UI 提供数据绑定支持
- 关键属性：`id`(用户 ID)、`username`(用户名)、`avatarUrl`(头像 URL)、`followingCount`(关注数)、`followerCount`(粉丝数)、`isFollowed`(是否关注)

VideoModel（视频模型类）

- 职责：封装视频业务数据，作为纯值对象提供数据访问
- 关键属性：`id`(视频 ID)、`title`(标题)、`url`(视频地址)、`coverUrl`(封面)、`duration`(时长)、`likeCount`(点赞数)、`authorName`(作者名)、`isLiked`(是否点赞)、`isFollowed`(是否关注作者)

2. 边界类（Boundary）

BrowseVideosModelView（视频浏览视图模型）

- 职责：连接 QML 界面与业务逻辑，实现 MVVM 数据绑定
- 关键属性：`videoMap`(视频缓存)、`isLoading`(加载状态)、`nextOffset`(分页偏移)
- 主要方法：`requestVideos`(请求视频)、`likeVideo`(点赞)、`followUser`(关注)

3. 控制类（Control）

AuthManager（认证管理器）

- 职责：统一管理用户认证流程，包括登录、注册、登出、Token 管理
- 关键属性：token(会话令牌)、currentUser(当前用户)、wasLogin(登录状态)
- 特点：单例模式，全局唯一认证状态管理

VideoPublisher（视频发布管理器）

- 职责：处理视频上传和发布的完整流程，状态机管理
- 关键属性：isUploading(上传状态)、uploadProgress(上传进度)、statusMessage(状态消息)

VideoAudioMerger（音视频合并控制器）

- 职责：调用 FFmpeg 进行音视频合成，进程管理
- 关键属性：videoFile(视频文件)、audioFile(音频文件)、durationMs(合成时长)
- 特点：外部进程调用，避免主进程崩溃

4. 辅助工具类（Utility）

NetworkClient（网络通信中间件）

- 职责：统一处理所有 HTTP 网络请求，提供异步 API
- 关键属性：networkManager(网络管理器)、apiBaseUrl(API 地址)、uploadEndpoint(上传端点)
- 特点：单例模式，支持回调函数

FileUtils（文件操作工具）

- 职责：封装跨平台文件操作，简化文件处理
- 主要方法：createVideoTempDir(创建临时目录)、getMergedVideoPath(获取合并路径)、fileExists(文件检查)

ConfigManager（配置管理器）

- 职责：管理应用配置，提供配置项访问接口
- 关键属性：config(配置数据)
- 特点：单例模式，支持配置文件加载

2. 服务端类设计（api_service）

1. 实体类（Entity）

User（用户实体）

- 职责：封装用户数据模型，定义数据库映射

- 关键属性：id(用户 ID)、username(用户名)、passwordHash(密码哈希)、avatarUrl(头像)、followingCount(关注数)、followerCount(粉丝数)

Video (视频实体)

- 职责：封装视频数据模型，包含个性化状态
- 关键属性：id(视频 ID)、title(标题)、url(视频地址)、coverUrl(封面)、duration(时长)、likeCount(点赞数)、authorName(作者名)、isLiked(是否点赞)

2. 数据持久化存储类 (Repository)

UserRepository (用户数据仓储)

- 职责：封装用户数据访问，实现 CRUD 操作和社交关系管理
- 主要方法：findByUsername(按用户名查询)、updateFollowStatus(更新关注状态)、getFollowingIds(批量查询关注)
- 特点：支持读写分离，批量操作优化

VideoRepository (视频数据仓储)

- 职责：封装视频数据访问，支持多种视频流查询
- 主要方法：getGlobalFeed(全局视频流)、getFollowingFeed(关注视频流)、getLikedFeed(点赞视频流)
- 特点：复杂 SQL 联表查询，性能优化

SessionRepository (会话仓储)

- 职责：管理 Redis 中的会话数据，支持互斥登录
- 主要方法：saveSession(保存会话)、removeSession(移除会话)
- 特点：Token-UserID 双向映射，自动过期清理

3. 业务类 (Service)

UserService (用户业务服务)

- 职责：实现用户核心业务逻辑，协调仓储操作
- 主要方法：registerUser(用户注册)、login(用户登录)、followUser(关注用户)
- 特点：业务规则封装，错误处理统一

VideoService (视频业务服务)

- 职责：实现视频核心业务逻辑，包含异步处理
- 主要方法：publishVideo(发布视频)、toggleLike(点赞/取消)
- 特点：乐观锁设计，最终一致性保证

RecommendationService (推荐服务)

- 职责：生成多种视频推荐流，个性化数据处理
- 主要方法：getDiscoveryFeed(发现页)、getFollowingFeed(关注页)、enrichUserData(数据丰富化)
- 特点：多种推荐策略，个性化状态计算

4. 控制器类 (Controller)

UserController (用户 HTTP 控制器)

- 职责：处理用户相关 HTTP 请求，参数验证和响应格式化
- 路由示例：POST /api/user/login、POST /api/user/follow
- 特点：统一响应格式，错误码标准化

FeedController (视频流控制器)

- 职责：处理视频流相关 HTTP 请求，分页和身份处理
- 路由示例：GET /api/feed/discovery、GET /api/feed/following
- 特点：支持游客模式，分页逻辑统一

VideoController (视频控制器)

- 职责：处理视频相关 HTTP 请求，业务参数验证
- 路由示例：POST /api/video/publish、POST /api/video/like
- 特点：乐观更新支持，事务一致性保证

5. 过滤器与中间件

LoginFilter (登录认证过滤器)

- 职责：验证 Token 有效性，保护需要登录的接口
- 处理流程：提取 Token → Redis 验证 → 注入用户 ID → 放行/拒绝
- 特点：横切关注点，统一认证逻辑

6. 调度器

SyncScheduler (数据同步调度器)

- 职责：定期同步 Redis 缓存数据到数据库，实现最终一致性
- 主要任务：syncLikesToDB(点赞计数同步)
- 特点：后台定时任务，批量处理优化

3. 服务端类设计 (video_worker)

视频处理工作业务类

1. FFmpegHelper (FFmpeg 工具辅助类)

- 职责：封装 FFmpeg/FFprobe 命令行工具调用
- 关键方法：getVideoDuration(获取时长)、generateThumbnail(生成封面)、transcodeToHls(转码为 HLS)

2. TaskQueue (任务队列管理类)

- 职责：管理 Redis 中的异步任务队列
- 关键方法：popTask(弹出任务)
- 特点：生产者-消费者模式，解耦视频发布与处理

3. VideoStatusRepository (视频状态仓储类)

- 职责：管理视频处理状态在数据库中的持久化
- 关键方法：markAsPublished(标记发布)、markAsFailed(标记失败)
- 特点：状态转换管理，错误处理

4. TranscodeService (视频转码服务类)

- 职责：视频处理管道的核心协调器
- 关键属性：queue(任务队列)、repository(状态仓储)、storage(存储客户端)
- 主要流程：队列获取 → 元数据提取 → 封面生成 → HLS 转码 → 并发上传 → 状态更新
- 特点：多线程并发处理，完整的错误恢复机制

6.3. 操作与算法设计

1. 操作签名定义

为了保证接口的清晰度与安全性，核心业务操作与其辅助操作分离。以下列举客户端与服务端最关键的几个类的方法签名。

1.1. 服务端：视频服务

职责：处理视频发布、点赞及元数据管理。

发布视频

签名：

```
void publishVideo(  
    const std::string& userId,
```

```

        const std::string& title,

        const std::string& tempFileUrl,

        std::function<void(Result<std::string>)>> callback

    );

```

描述：接收客户端上传至临时桶的文件地址，创建数据库记录，并推送到 Redis 转码队列。

异常：若数据库写入失败，通过 `callback` 返回错误码 500。

切换点赞状态

签名：

```

void toggleLike(

    const std::string& userId,

    const std::string& videoId,

    bool isLike,

    std::function<void(Result<long long>)>> callback

);

```

描述：原子性地更新点赞关系表，并同步更新 Redis 计数器。返回最新的点赞数。

1.2. 客户端：视频上传管理器

职责：管理分片上传、断点续传及进度通知。

开始上传

签名：

```

Q_INVOKABLE void startUpload(const QString& filePath, const QString& title);

```

描述：供 QML 调用。启动后台线程，计算文件 MD5，申请上传凭证，并开始分块传输。

计算文件哈希

签名：

```

QString calculateFileHash(const QString& filePath) const;

```

描述：读取文件前 1MB 和后 1MB 进行快速哈希计算，用于秒传检测。

2. 对象状态建模

针对系统中生命周期复杂的对象，采用有限状态机（FSM）进行建模，以确保状态流转的可控性。

2.1. 视频生命周期状态机

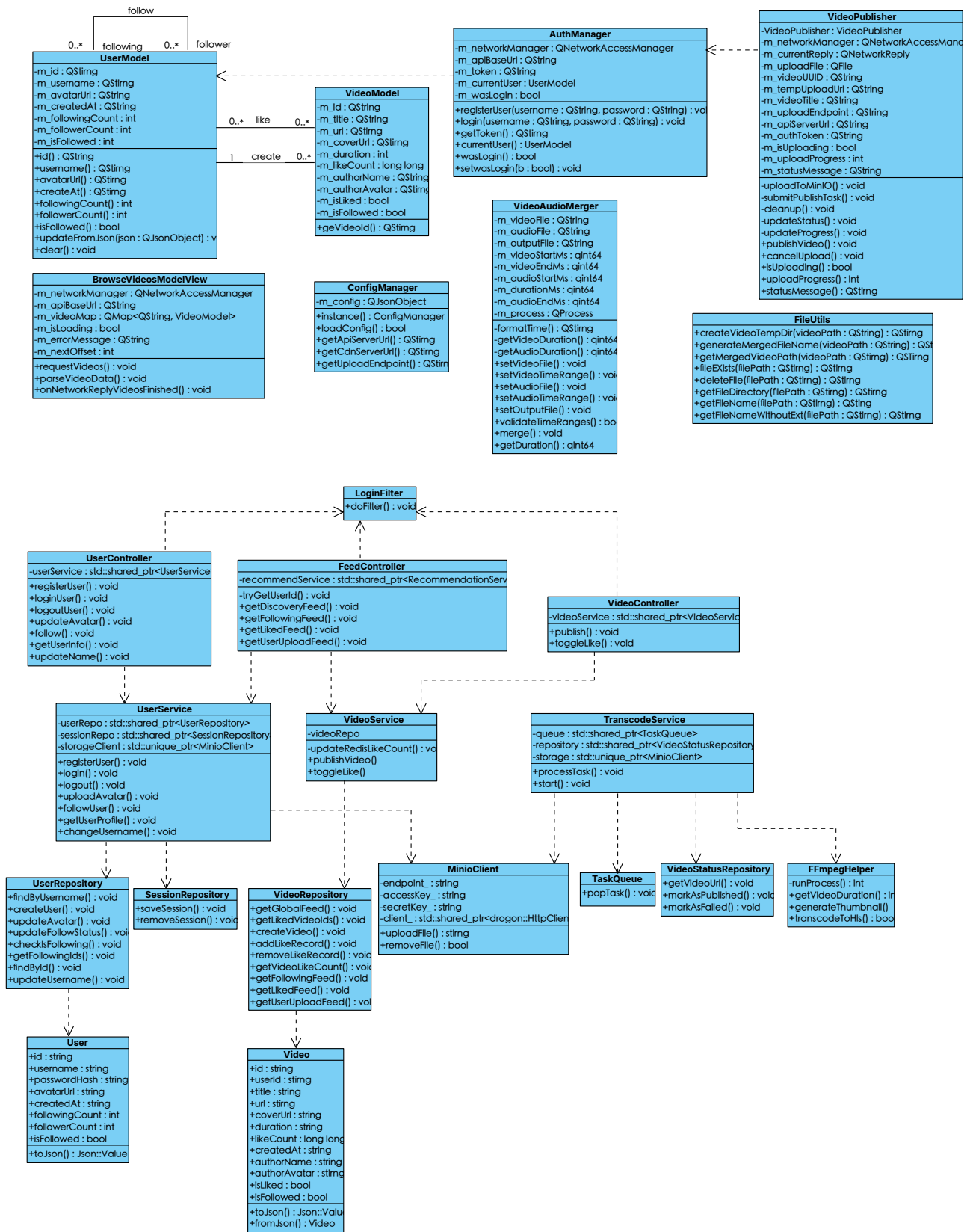
当前状态	触发事件	转换后状态	动作
Draft	客户端开始上传	Uploading	在 temp 桶创建对象，记录上传进度
Uploading	客户端上传完成	Processing	触发 Redis 消息队列，Worker 开始下载
Uploading	上传超时/网络中断	Failed	清理 temp 桶残留文件
Processing	转码 & 审核通过	Published	移动文件至 public 桶，生成 HLS，更新 CDN
Processing	转码失败/审核驳回	Failed	记录错误日志，通知用户
Published	用户删除/管理员下架	Deleted	标记 status=-1，不再分发

2.2. 播放器控制器状态机

当前状态	触发事件	转换后状态	动作
Idle	设置 mediaSource	Loading	初始化解码器，请求网络资源
Loading	缓冲数据足够	Playing	开始渲染帧，启动进度条计时器
Loading	网络请求超时	Error	显示重试按钮
Playing	用户点击暂停	Paused	暂停渲染，保持当前帧
Playing	缓存枯竭	Buffering	暂停播放，显示加载动画 (Loading Spinner)
Playing	播放结束	Ended	重置进度为 0 (若循环则跳回 Playing)
Buffering	数据补充完毕	Playing	隐藏加载动画，恢复播放

d	Pause	用户点击播放	g	Playin	恢复渲染
	Error	用户点击重试		Loadi	重新发起网络请求

6.4. 关联与结构设计



6.5. 设计优化与模式应用

1. 设计模式应用

为了提高代码的可维护性、扩展性及解耦各模块，本项目在关键环节应用了以下设计模式：

1.1. 单例模式

应用场景：全局配置管理与资源池管理。

具体实现：

GlobalConfig 类：负责加载并持有本地 `config.json` 及服务端下发的配置参数。整个应用程序生命周期内仅需一份配置副本。

DatabaseConnectionPool (服务端)：维护 PostgreSQL 的连接池。数据库连接是昂贵资源，通过单例模式确保连接池的统一管理和复用，避免频繁创建/销毁连接带来的开销。

收益：确保全局访问点的唯一性，严格控制共享资源的实例化数量。

1.2. 工厂模式

应用场景：多格式媒体处理器的创建。

具体实现：

定义抽象基类 **IMediaDemuxer**（解封装器）。

创建 **DemuxerFactory** 类，提供 `createDemuxer(const QString& filePath)` 方法。

根据输入文件的后缀名（`.mp4`, `.mov`, `.flv`），工厂方法动态返回对应的 **MP4Demuxer** 或 **FLVDemuxer** 实例。

收益：符合开闭原则（OCP），新增支持的视频格式时，只需新增子类 and 修改工厂逻辑，无需修改业务调用代码。

1.3. 观察者模式

应用场景：跨层通信与界面实时更新。

具体实现：

机制复用：深度利用 Qt 的 **Signal & Slot** 机制，这是观察者模式的高级实现。

场景：当 C++ 层的 **VideoUploadManager** (被观察者) 的上传进度发生变化时，发射 `progressChanged(int)` 信号。前端 QML 层的 **ProgressBar** 组件 (观察者) 连接此信号，自动更新 UI 进度条，无需轮询。

收益：实现了业务逻辑层（C++）与视图层（QML）的完全解耦。

1.4. 代理模式

应用场景：列表滑动时的图片懒加载。

具体实现：

在视频列表（**ListView**）中，使用 **ImageProxy** 替代直接加载高清封面图。

ImageProxy 在初始化时仅显示极小的占位色块或本地缓存的低清缩略图。

当该列表项滚动进入屏幕可视区域（**Viewport**）时，代理对象才发起真实的网络请求下载高清图，并进行替换。

收益：大幅降低快速滑动列表时的内存峰值和网络带宽压力，提升列表流畅度。

2. 性能与资源优化

为了达成“秒开”和“60fps 流畅滑动”的性能目标，我们在数据结构和架构层面进行了深度优化。

2.1. 冗余数据设计

问题：在视频信息流（**Feed**）查询中，如果每次都通过 **JOIN** 操作去关联 **User** 表获取作者头像、昵称，在大数据量下会导致数据库 **CPU** 飙升。

优化方案：空间换时间。

在 **Video** 表中冗余存储 **author_name** 和 **author_avatar_url** 字段。

在 **Video** 表中增加 **like_count** 字段，而不是每次通过 **SELECT COUNT(*) FROM likes WHERE video_id=...** 实时计算。

代价管理：当用户修改头像或昵称时，通过消息队列异步更新其发布的所有视频的冗余字段（最终一致性）。

2.2. 路径优化与预计算

问题：客户端播放视频需经过“请求 API -> 解析 URL -> 请求 CDN -> 302 重定向 -> 获取流”的漫长过程。

优化方案：最短访问路径。

服务端预签名：API 返回的视频 **URL** 不再是应用服务器的代理地址，而是直接指向 **MinIO** 的 **CDN** 分发地址。

预计算播放列表：对于“推荐页”，系统后台预先计算好用户的推荐列表并存入 **Redis List**。用户刷新时，直接从 **Redis** 弹出前 10 条视频数据，无需实时运行复杂的推荐 **SQL**，将接口响应时间控制在 50ms 以内。

2.3. 对象池技术

问题：**QML** 中 **VideoOutput** 和解码器对象的创建与销毁非常耗时，导致滑动切换视频时出现掉帧。

优化方案：复用播放器实例。

在全屏滑动组件中，仅维护 3 个 播放器实例（**Current**, **Previous**, **Next**）。

当用户向下滑动时，将移出屏幕的“**Previous**”实例回收，重置其状态，并将其复用为新的“**Next**”实例加载预加载数据。

避免了反复申请和释放 **GPU** 显存和解码器上下文。

2.4. 异步并发与写缓冲

问题：热门视频可能在短时间内遭受数万次点赞请求，直接写入数据库会导致行锁冲突，甚至拖垮数据库。

优化方案：Redis 缓冲 + 异步落库。

读：点赞数直接从 Redis 读取。

写：用户点赞操作仅在 Redis 中执行 INCR 原子操作，并写入消息队列。

落库：后台 Worker 进程批量消费队列，每隔 N 秒或积累 M 条数据后，合并 SQL 批量写入 PostgreSQL

6.6. 数据存储详细设计

本系统采用混合存储架构：关系型数据库（PostgreSQL）存储核心业务数据，对象存储（MinIO）存储音视频非结构化数据，内存数据库（Redis）用于缓存热点数据和消息队列。

1. 关系型数据库模式 (PostgreSQL)

1.1. ER 图转物理表结构定义

用户表 (users)

描述：存储用户基本信息及统计数据的冗余字段。

DDL 定义：

```
CREATE TABLE users (  
    id VARCHAR(64) PRIMARY KEY,      -- UUID  
    username VARCHAR(50) NOT NULL,   -- 用户名  
    password_hash VARCHAR(256) NOT NULL, -- SHA256 加密存储  
    avatar_url VARCHAR(255),          -- 头像 CDN 地址  
    following_count INT DEFAULT 0,    -- 关注数  
    follower_count INT DEFAULT 0,    -- 粉丝数  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

视频表 (videos)

描述：存储视频元数据及发布状态。

DDL 定义：

```
CREATE TABLE videos (  
    id VARCHAR(64) PRIMARY KEY,      -- UUID
```

```

user_id VARCHAR(64) NOT NULL REFERENCES users(id), -- 作者 ID
title VARCHAR(200), -- 视频描述/标题
url VARCHAR(255) NOT NULL, -- HLS 索引文件(.m3u8)的 CDN 地址
cover_url VARCHAR(255), -- 封面图地址
duration INT, -- 时长(秒)
like_count BIGINT DEFAULT 0, -- 点赞数
status SMALLINT DEFAULT 0, -- 0:处理中, 1:已发布, 2:审核失败, -1:已删除
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

```

点赞关系表 (video_likes)

描述：记录用户与视频的点赞关系。

DDL 定义：

```

CREATE TABLE video_likes (
    user_id VARCHAR(64) NOT NULL REFERENCES users(id),
    video_id VARCHAR(64) NOT NULL REFERENCES videos(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (user_id, video_id) -- 复合主键，防止重复点赞
);

```

关注关系表 (user_follows)

描述：记录用户之间的关注关系。

DDL 定义：

```

CREATE TABLE user_follows (
    follower_id VARCHAR(64) NOT NULL REFERENCES users(id), -- 粉丝
    following_id VARCHAR(64) NOT NULL REFERENCES users(id), -- 被关注者
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (follower_id, following_id)
);

```

1.2. 索引设计

表名	索引	字段	用途说明
----	----	----	------

	类型		
users	UNI QUE	username	保证用户名全局唯一，加速登录查询
videos	BTR EE	user_id	加速“个人主页-作品”列表查询
videos	BTR EE	created_at DESC	加速“首页-最新”信息流的分页查询
video_likes	BTR EE	video_id	虽然有主键，但单独对 video_id 建索引可加速“谁赞过我”查询
user_follows	BTR EE	following_id	加速“粉丝列表”查询

2. 非结构化存储设计 (MinIO)

2.1. 存储桶 (Bucket) 规划

桶名称 (Bucket)	访问权限	用途	生命周期策略
lepai-temp	Private	存放客户端上传的原始视频文件	设置 24 小时自动过期删除，节省空间
lepai-public	Public Read	存放转码后的 HLS 切片、封面图、头像	永久存储，配合 CDN 分发

2.2. 目录层级结构

为了避免单目录下文件过多导致性能下降，采用基于业务 ID 的打散策略：

头像：avatars/{user_id}.jpg

视频封面：covers/{date}/{video_id}.jpg

视频流媒体：videos/{video_id}/

结构：

index.m3u8 (HLS 索引文件)

seg_000.ts (视频切片 0)

seg_001.ts (视频切片 1)

...

3. 缓存数据结构 (Redis)

3.1. Key-Value 命名规范

所有 Key 采用 项目名:业务模块:ID 的格式

3.2. 核心缓存结构定义

用户会话 (Session)

Key: session:token:{token_string}

Type: String

Value: {user_id}

TTL: 30 天 (每次活跃自动续期)

视频点赞计数器

Key: video:likes:{video_id}

Type: String (作为 Integer 使用)

Value: 点赞数值

操作: INCR, DECR

说明: 高频点赞先在 Redis 操作, 后台定时同步回 PostgreSQL。

视频转码队列

Key: queue:transcode

Type: List

Value: {video_id}

操作: LPUSH (生产), BRPOP (消费)

热门视频排行榜

Key: rank:video:daily

Type: ZSet (Sorted Set)

Member: {video_id}

Score: 推荐算法计算出的热度分值

说明: 用于快速拉取首页推荐的 Top N 内容。

关注列表

Key: user:following:{user_id}

Type: Set

Value: {target_user_id}

说明: 用于快速判断“我是否关注了作者”, 在渲染视频流时进行 $O(1)$ 查找, 避免查库。

6.7. 接口详细说明

1. 客户端-服务端交互接口

采用 RESTful API 风格，通过 HTTP/HTTPS 协议通信。请求与响应体统一采用 JSON 格式。

1.1. 通用协议规范

基础路径 (Base URL): `https://api.lepai.com/v1`

字符集: UTF-8

鉴权方式: HTTP Header 中携带 JWT Token

Authorization: Bearer <token_string>

统一响应结构:

所有接口（除文件流下载外）均返回以下 JSON 结构：

```
{
  "code": 200,      // 业务状态码，200 表示成功
  "message": "success", // 提示信息，用于调试或前端展示
  "data": { ... }   // 业务数据负载，可为 Object 或 Array
}
```

1.2. 核心接口定义

获取推荐视频流

Method: GET /feed/recommend

Params:

limit: (int, optional) 单次拉取数量，默认 10。

offset: (string, optional) 分页游标。

Response (Data):

```
{
  "next_cursor": "eyJid...",
  "items": [
    {
      "video_id": "v89757",
      "title": "街舞挑战",
      "play_url": "https://cdn.lepai.com/videos/v89757/index.m3u8",
      "cover_url": "https://cdn.lepai.com/covers/v89757.jpg",
      "author": { "id": "u123", "name": "DanceKing", "avatar": "..."}
    }
  ]
}
```

```

        "stats": { "likes": 1024, "comments": 50 }
    }
]
}

```

发布视频

Method: POST /video/publish

Body:

```

{
    "title": "我的第一个作品",
    "temp_file_key": "temp/u123_random.mp4", // 客户端直传 MinIO 后的临时 Key
    "duration": 15,
    "music_id": "m_1001"
}

```

Response (Data): { "video_id": "v99999", "status": "processing" }

点赞/取消点赞

Method: POST /video/like

Body: { "video_id": "v89757", "action": true } // true=点赞, false=取消

Response (Data): { "current_likes": 1025 } // 返回最新点赞数

1.3. 错误码定义表

错误码	含义	处理建议
200	成功	正常处理业务逻辑
401	未授权/Token 失效	跳转至登录页或静默刷新 Token
403	权限不足	提示用户无权操作
404	资源未找到	提示内容已被删除
4001	视频格式不支持	提示用户重新选择视频
5000	服务端内部错误	提示“网络开小差了”，稍后重试

2. QML-C++ 交互接口

客户端采用 Qt 框架，逻辑层使用 C++，界面层使用 QML。两者的交互主要通过属性绑定和信号槽机制实现。

2.1. 上下文属性

C++ 将核心控制类实例化后，注册为 QML 的全局对象，使得前端可以直接调用。

对象名: AuthService (类: UserAuthManager)

Q_PROPERTY(bool isLoggedIn READ isLoggedIn NOTIFY loginStateChanged): 用于 UI 根据登录状态切换界面。

Q_INVOKABLE void login(QString username, QString password): 登录操作。

对象名: VideoUploader (类: VideoUploadManager)

Q_PROPERTY(double progress READ getProgress NOTIFY progressChanged): 绑定到进度条组件。

Q_PROPERTY(int status READ getStatus NOTIFY statusChanged): 0=闲置, 1=上传中, 2=完成。

2.2. 数据模型接口

用于展示长列表，C++ 实现 QAbstractListModel。

类名: VideoFeedModel

暴露给 QML 的角色 (Roles):

TitleRole (对应 title): 视频标题。

UrlRole (对应 play_url): 播放地址，绑定到 VideoOutput。

CoverRole (对应 cover_url): 封面图。

AuthorRole (对应 author_name): 作者昵称。

槽函数 (Slots):

void refresh(): 下拉刷新时调用。

void loadMore(): 滑动到底部时调用。

3. 子系统内部接口

3.1. 推荐引擎调用接口

调用方: API 服务 (Feed 模块)

提供方: 推荐子系统

协议: gRPC / Internal HTTP

接口描述:

输入: { "user_id": "u123", "scene": "discovery", "count": 10 }

输出: { "video_ids": ["v1", "v2", "v3", ...] }

说明: API 服务仅索取 ID 列表, 随后自行查询数据库组装详细信息。

3.2. 视频转码服务回调接口

调用方: 转码 Worker (FFmpeg 节点)

提供方: API 服务 (Video 模块)

机制: 消息队列 (Redis/Kafka) 或 HTTP Webhook

接口描述:

转码完成后, Worker 向 queue:transcode:result 队列写入消息:

Payload:

```
{  
  "video_id": "v99999",  
  "status": "success",  
  "hls_url": "videos/v99999/index.m3u8",  
  "duration": 15  
}
```

API 服务监听此队列, 更新数据库状态为 “已发布 (Published)”