

肇慶學院

毕 业 论 文

题 目	基于 Spring Boot 的新媒体企业 信息化系统设计与实现
学 院	计算机科学与软件学院
专 业	软件工程
年 级	2016 级
学 号	201624133225
学生姓名	何承翰
指导教师	宋强、梁展鹏（企业）
完成时间	2020 年 4 月

肇庆学院教务处制

学术诚信声明

本人所呈交的毕业论文，是在指导教师的指导下独立完成。研究工作所取得的成果、数据、图片资料均真实可靠。除文中已注明引用的内容外，不包含任何其他人或集体已经发表或撰写过的作品或成果。对本论文的研究做出重要贡献的个人和集体，均已在文中以明确的方式标明。本毕业论文的知识产权归属于培养单位。本人完全意识到本声明的法律结果由本人承担。

本人签名（手写）：

日期：2020 年 4 月 6 日

目 录

摘要与关键词.....	1
1 概述	1
1.1 研究背景	1
1.2 技术现状	2
1.3 改进的可行性研究	3
2 需求分析	4
2.1 公司需求	4
2.1.1 招商引流	4
2.1.2 企业管理	5
2.1.3 活动直播与互动	5
2.1.4 内容审核	5
2.1.5 内容分发	5
2.2 客户需求	5
2.2.1 对外宣发	6
2.2.2 引流推广	6
2.2.3 品牌建设	6
3 系统设计	6
3.1 系统框架	6
3.1.1 前端	6
3.1.2 后端	7
3.2 设计模式	8
3.2.1 前后分离	8
3.2.2 分布式	8
3.3 功能模块	8
3.3.1 客户前端（官网）	9
3.3.2 管理前端	10
3.3.3 小程序前端	12
3.3.4 内容分发后端	12
3.3.5 数据综控后端	13
3.4 数据库设计	13
4 系统实现	14
4.1 Spring Boot 后端	14

4.1.1 控制层 (Controller)	15
4.1.2 服务层 (Service)	17
4.1.3 数据访问对象 (Data Access Object, DAO)	18
4.1.4 持久对象 (Persistent Object, PO)	19
4.1.5 表现对象 (Value Object, VO)	19
4.1.6 数据传输对象 (Data Transfer Object, DTO)	20
4.1.7 简单无规则 JAVA 对象 (Plain Ordinary Java Object, POJO)	20
4.1.8 JAVA 持久化 API (Java Persistence API)	21
4.2 Vue.js 前端	21
4.2.1 视图组件 (Component)	21
4.2.2 页面容器 (View)	22
4.2.3 路由 (Route)	25
4.2.4 数据通信 (Data Communication)	26
4.2.5 状态管理库 (Vuex)	27
4.2.6 工具 (Utility)	28
4.3 小程序前端	28
4.3.1 用户授权 (Authorize)	28
4.3.2 视图组件 (Component)	29
4.3.3 数据通信 (Data Communication)	29
4.3.4 页面容器 (View)	29
4.3.5 工具 (Utility)	29
4.3.6 全局变量 (Global)	30
4.3.7 路由 (Route)	30
4.4 Python 后端	30
4.4.1 消费者模块 (消息队列)	30
4.4.2 自动化模块 (内容分发)	31
4.4.3 数据库操作模块 (信息回调)	31
4.5 RabbitMQ 消息队列中间件	32
4.5.1 基本对象	32
4.5.2 处理逻辑	32
4.6 项目部署	33
4.6.1 后端部署	33
4.6.2 前端部署	33
4.6.3 小程序部署	34
4.6.4 服务器部署	34

4.6.5 代码托管	35
5 系统测试	35
5.1 压力测试	35
5.1.1 模拟高并发	35
5.1.2 模拟高负载	36
5.2 逻辑测试	36
5.3 漏洞检测	37
6 结论	37
参考文献:	39
Abstract and Keywords	41
致 谢	42

基于 Spring Boot 的新媒体企业信息化系统设计与实现

摘 要: 本论文应从改善传统媒体企业中的痛点与不足入手, 论述构建立体、全面、便捷的新媒体信息化工程体系的意义, 结合习近平总书记在新的形势下提出“建设全媒体, 推动媒体融合向纵深发展”这一重要论题, 比较新旧传媒行业的异同, 通过实践具体分析行业需求, 初步探讨大环境下 IT 技术如何切实服务地方媒体企业, 与如何实现私营性质的新媒体企业的快速过渡与转型, 并详细描述已被构建完成的新媒体企业信息化系统, 论述未来可能会出现业界变革。

关键词: 新媒体; 信息化工程; 分布式; Spring Boot; Vue.js; Python 自动化; 消息队列; 媒体分发; 小程序

1 概述

1.1 研究背景

时至今日, 在“万物互联”的 IT 浪潮下, “自媒体”、“融媒体”、“全媒体”等以网络为主体的新媒体形式不断升温, 智能手机、平板电脑等便携设备在短短的几年时间内就改变了绝大多数人获取信息的习惯; 纸媒、广播、书籍等传统的媒体形式渐渐转变为了公众号推文、短视频、问答评论; 越来越多的年轻人热衷于分享自己的生活, 除了在朋友圈秀出自己的日常生活外, 他们也会在各大媒体平台创造属于自己的“用户生产内容 (User-generated Content, UGC)”……逆水行舟, 不进则退, 越来越多的传统媒体开始数字化转型为新媒体, 并开始生产“专业生产内容 (Professionally-generated Content, PGC)”和“职业生产内容 (Occupationally-generated Content, OGC)”, 其中包括各大党媒、省市级电视台、报社、广播电台、地方性媒体等; 各类内容生产媒体之间的关系如图 1-1 所示。

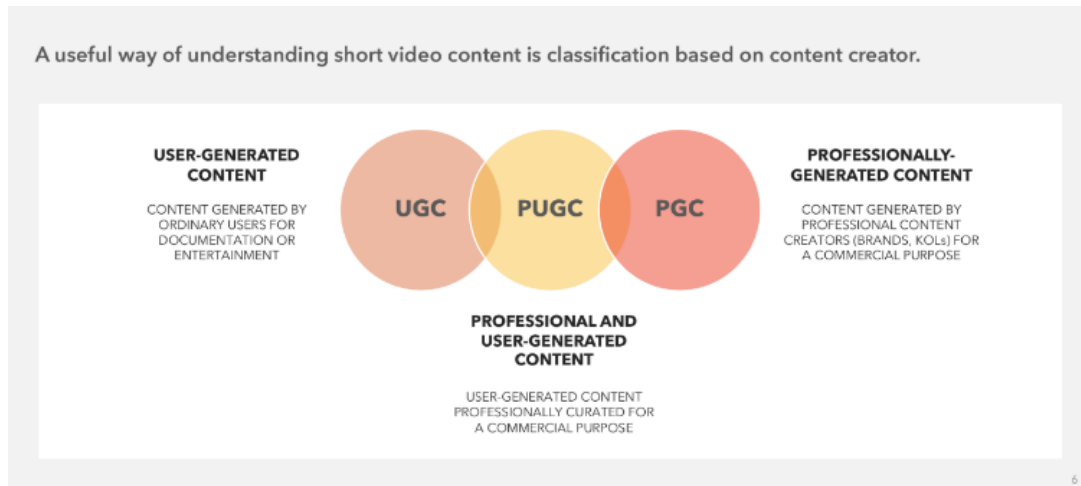


图 1-1 各类内容生产媒体一览

如此庞大的转型需求在技术高速迭代的推动下，催生出了一个巨大的 IT 商机与潜在的媒体管理服务市场：大量的媒体机构亟需令自己采编的新闻通过网络传递给更多的受众，而通过互联网人们开始收获源源不断的信息，如何以最短的时间将最具价值的信息发布给最多的人？这便是媒体行业在工业革命 4.0 时代最大的需求，也是这个信息化时代的最终出口^[1]。

1.2 技术现状

据了解，国内大量新媒体公司或传统媒体企业或机构的新媒体编辑部仍然沿用具有普适性、功能单一的信息化管理系统，这些系统仅帮助公司进行考勤、工作汇报、审批等公司日常事务的处理，与媒体相关的业务也和传统媒体业务紧密相连，譬如部门之间的文稿协同修改与多级审核、内部稿件的多角色打分与闭环分发（仅在授权机构内共享）。若要发布至其他网络媒体平台，仍然要使用外部的其他工具。

在内容分发上，国内外皆有一定数量的多平台账号分发管理与分析系统（Communications as a Service, CaaS），同时也拥有部分专门针对媒体单位开发的工作流管理系统，它们用于改进内容项目组件和/或生成新的内容项目，可以利用各种发布规则来确定系统何时准备发布内容项并将该项分发给客户端设备^[2]。却并没有现成的集招商引资、企业管理、活动直播、内容分发、流程管理等多项媒体服务为一体的定制化系统，如果存在这样直接针对新媒体行业的特殊性，解决特定的问题的系统，新媒体企业的工作效率将得到极大的提升；各大数媒 IT 企业中常见的内容分发流程如图 1-2 所示。

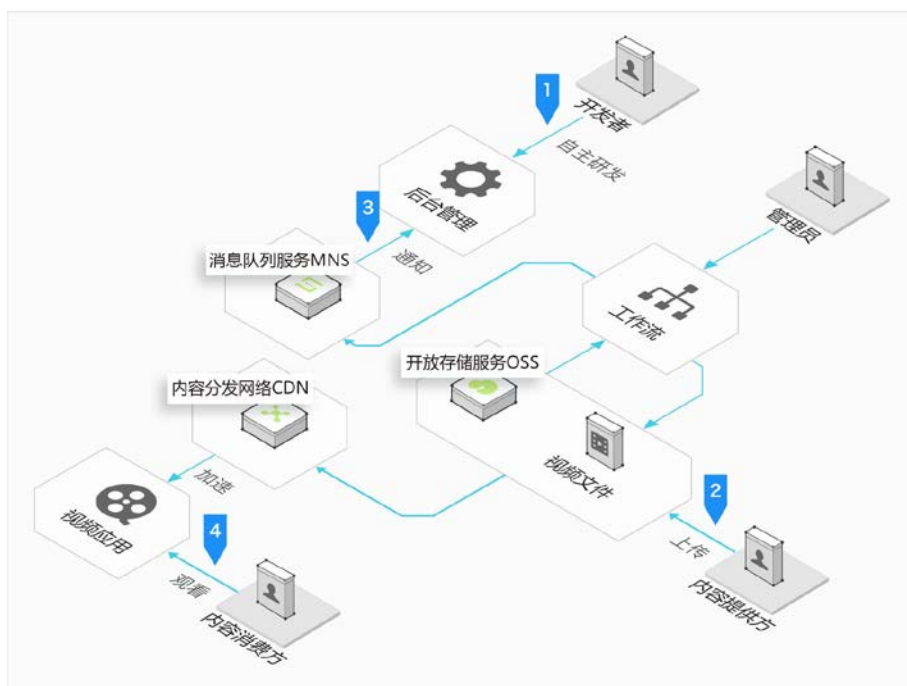


图 1-2 传统意义上的内容分发流程

1.3 改进的可行性研究

本论文选题来自个人当下实习的单位「腾讯·大粤网（肇庆）」，通过为期 4 个多月的实习，本人对新媒体公司的各项业务有了很深刻的认知。其中，由于新媒体运营讲求「短、平、快」，一个活动、视频或宣发项目的体量较小，同时制作周期相对紧张，员工们大多潜心于实现对口单位的需求，而往往腾不出时间进行内容的纵向宣发。这就导致了两样潜在的隐患：其一是影响力有限，作为我们服务方的大多都是事政单位，仅凭我们在腾讯新闻 APP 地方性页卡的展示和他们在各自的专业平台进行宣发，很多有价值、花了心思采编的新闻并没有通过脍炙人口的网络媒体平台使更多的人知晓，又由于没有一个专门的平台整合展示公司曾经接手的项目，这使得我们扩展业务仅仅凭借“腾讯”的口碑，从而陷入被动；其二是这样的结果会大大打击员工们的积极性，同时也会让我们的甲方失去信心。

以小见大，这样的问题并非我们一家新媒体企业独有，而是许许多多同类型公司的通病。媒体业务的精髓在于团队的高效合作，其中也包括了采编之外的宣发。而回到主题，IT 恰恰是为了解放人类劳动力而生的工具，为何不能通过它来改变这种现状，提供我们的生产力呢？

从采编到审核，再到分发，以及资料归档和展示，公司业务流还有很多地方可以改进，可以通过技术手段提高工作效率，提升企业的生命力，故我们开始着手设计自家公司的新媒体矩阵，详细架构如图 1-3 所示。

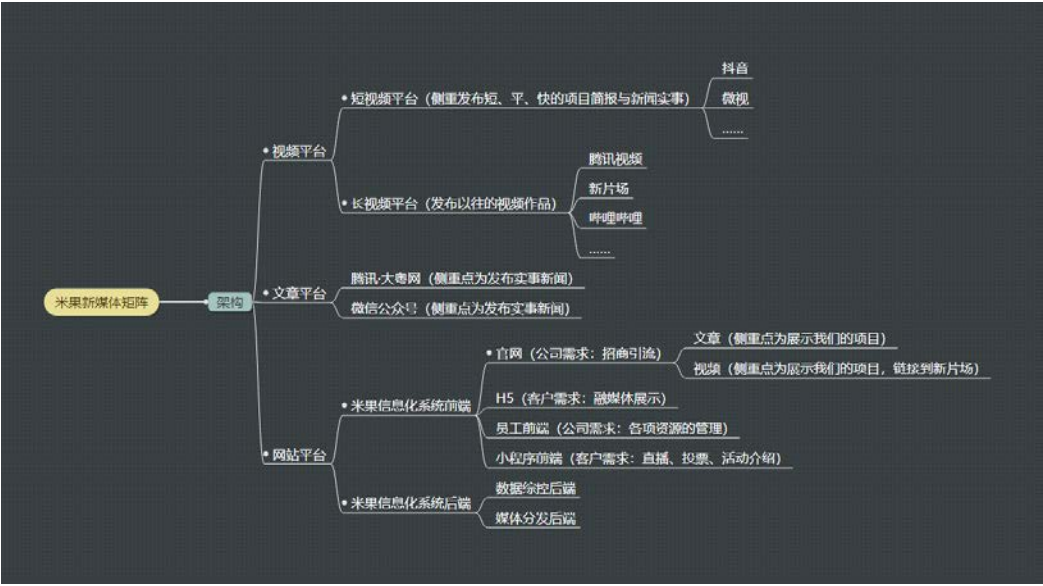


图 1-3 公司新媒体矩阵架构图

2 需求分析

2.1 公司需求

本人目前实习的公司为广东米果传媒有限公司，为腾讯·大粤网在肇庆地区的城市合作伙伴。公司主要负责当地新闻的采编与发布，提供企业和政府品牌建设服务、企业和政府宣传片摄制、大型活动策划与组织。为了更好地满足客户的需求，同时也为了公司的茁壮成长和员工之间的高效配合，以下需求被归类为公司自身所需要的服务。

表 2-1 2019 年公司业务量总览

业务量	企业单位	事政单位	合作伙伴	合计
新闻采编	12	34	65	111
视频制作	15	16	15	46
活动策划	5	4	8	17
合计	32	54	88	174

2.1.1 招商引流

为了接更多的商单，挖掘更多的潜在客户，同时也为了今后向粤西地区扩张业务，公司需要一个官方网站以展示其实力，其中包括经手项目、采编内容的分类归档，公司基本信息展示等。

2.1.2 企业管理

对于公司的日常事务，需要一个系统实现员工的工作管理，例如日常打卡、外出报备、日志周报、报销出纳、器材使用申请等。

2.1.3 活动直播与互动

公司的主营业务之一就是大型活动的策划与直播，其中直播业务即为编导组、拍摄组、后勤组相互配合采集活动会场实况并转播到腾讯新闻 APP 特定的直播页卡，相比于以往功能局限的直播业务，我们决定在与观众的互动层面上实施改进，需要新增公司直播小程序，要求集成投票、公告、弹幕互动、赞助商介绍等多项业务于一体，同时能够实现不同活动之间的后端无缝一键切换。

2.1.4 内容审核

对于公司内部采编团队，为了使员工编辑的内容更具严谨性，需要加入上级审核功能，媒体编辑员工提交审核后，管理员即可开始审核，这能够免去传统工作流中通过第三方社交软件沟通交流的过程，使得整体作业更加高效。

2.1.5 内容分发

针对上述员工工作侧重点并非内容分发的问题，需要提供一套精简分发流程的解决方案，要求通过技术手段实现媒体的一键分发，包括公司官网+网络媒体平台的内容分发，即「一次分发，多平台同步」。

2.2 客户需求

除去公司自身的各项需求外，由于我们新媒体企业的特殊性，还需要设计一系列针对服务方（甲方）的业务解决方案；由图 2-1 可知，新闻采编和视频制作为公司主营业务，但利润最高的却是业务量最小的活动策划，如何继续发展该项业务，是公司未来重要的商业筹码。



图 2-1 2019 年公司收支总览

2.2.1 对外宣发

为了协助事业单位类型的甲方宣发政策和普及知识，除了在对应的官方平台上发布政策解读等材料外，还应当根据地域性针对特定人群进行精准投放，此时就需要借助多方网络媒体平台的帮助。得益于公司商务部的协助，有多家媒体平台相继与我们达成了地区合作伙伴关系，如何将具有时效性的新闻内容快速上传到对应的合作平台？这也需要上文中提到的“精简分发流程的解决方案”。

2.2.2 引流推广

如何帮助企业类的甲方更好地面向社会招商，同时提升公司自身的影响力？这需要通过某种方式更广地传播我们帮助甲方策划与制作的宣发物料，首先是通过母公司“腾讯·大粤网”自身巨大的影响力，其次需要其他平台协助推广给对口的人群，这就需要将制作的媒体内容发布到更多的网络平台上，潜在客户可以通过搜索、系统推送、竞价排名等方式了解到甲方的业务。

2.2.3 品牌建设

无论是何种形式的组织单位，品牌建设必定是发展途中不可缺少的一环。若欲可持续发展，必须得有一个既定的良好印象留存给对口客户。这就需要一个承载单位核心竞争力的特定的展示页作为对外窗口，如何设计官网以达到给人留下深刻印象这一目的？网页创意与布局至关重要。

3 系统设计

3.1 系统框架

3.1.1 前端

对于目前的业务需求，需要选用一种布局美观、业务逻辑清晰、易于迭代与维护的前端方案，所以本项目最终选用以下前端框架进行开发：

(1)Vue.js。

作为渐进式 JavaScript 框架中的翘楚，注重视图层的 Vue.js 有足够的灵活性来适应不同的需求^[3]。它的独特之处在于脱离了以往庞大的后端依赖，仅凭借配置接口即可无视后端状态，甚至可进行跨域 WebSocket 数据请求。本项目采用其作为官网前端和管理前端的前端框架，响应式工作逻辑如图 3-1 所示。

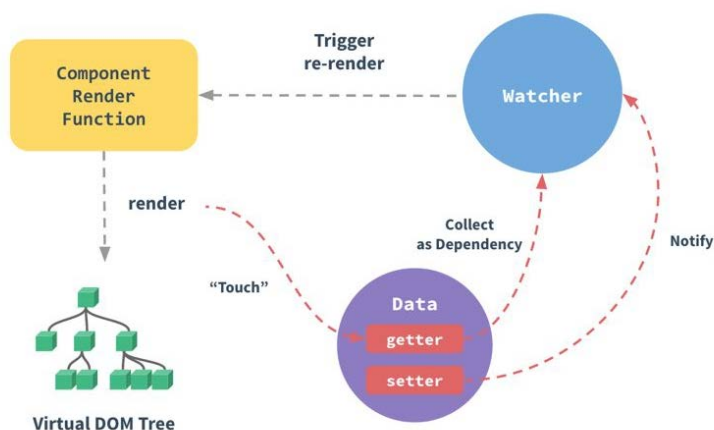


图 3-1 Vue. js 响应式框架图

(2) 小程序。

这是一种无需下载安装，只需扫一扫即可在线使用，使用完毕也无需卸载，非常方便快捷的微服务框架^[3]。相较于传统的原生 APP，微信小程序通过高度集成与优化，已经完成了对各类型手机的优化，从而大大减轻了开发者的调试成本；而其简洁至臻的开发文档，则又一次降低了开发者的学习门槛。本项目使其作为“活动直播与互动”这一需求的前端载体，其后端仍然需要 Spring Boot 的支撑。

3.1.2 后端

由于业务涉及多元接口的互相通信，为实现复杂的分布式逻辑交互需求，本人最终选用以下四项技术（框架）：

(1) Spring Boot。作为目前非常流行的微服务框架，由于其清晰的架构、高效的开发与运行效率，它深受互联网企业的青睐，几乎成为了微服务中间件事实上的标准^[4]。本项目采用其构建官网、小程序、自动分发、消息队列的数据综控后端。

(2) Python。被冠以「胶水语言」的 Python 可以帮助我们轻松模拟普通人的操作，突破网站拦截限制，实现对各大平台的自动化执行流程。使用 Python 开发程序，许多功能不必从零编写，直接调用现成封装好的模块的即可^[5]。本项目采用其模拟人工操作，实现媒体内容的自动分发。

(3) MySQL。这是如今开发中小型项目最常用的一种关系型数据库，此类数据库将数据保存于不同的表中，由后端直接对口操作，兼容多种语言，普适性强。本项目采用其作为后端数据库存储业务数据。

(4) RabbitMQ。在介绍 RabbitMQ 前，首先简单讲解下消息队列（亦称消息中间件）。高级消息队列协议（Advanced Message Queuing Protocol, AMQP）是面向消息的中间件设计的一个应用层协议开放标准^[6]。说回 RabbitMQ，它是一个实现了高级消息队列协议的开源

消息代理组件（亦称面向消息的中间件），多用于分布式系统消息的存储与转发。本项目采用其作为后端媒体分发业务中 Spring Boot 与 Python 进行数据交互的中间件。

3.2 设计模式

3.2.1 前后分离

前后端分离开发是时下主流的互联网项目开发方式，在本项目中最典型的体现即为 Spring Boot + Vue.js 设计方式。这种开发模式的优点在于：

(1) 可演进与可迭代。与以往将 MVC 架构集成于某一个特定的框架之中不同，它将视图层（View）划分出来，使各模块各司其职，高内聚低耦合。在系统生命周期的中后期，需要维护与增加新功能的时候能够迅速做出响应，快速完成变更。

(2) 业务解耦与复用。在这种开发模式下，由于模型层（Model）和控制层（Controller）被分离了出来，视图层（View）皆以调用接口的方式获取数据，故接口的重复使用成为了可能，无需如传统的系统那样针对视图层开发接口，造成服务冗余。

(3) 部署独立。配置前后端分离项目的时候，它们无需同步更新，共同发布。这也可以算是它的缺点，当项目前期规划不明确而导致前后端业务黏连时，后期部署就会产生很大的问题，包括跨域、接口分配混乱、请求方法不匹配等。但如果项目前期规划得当，这些缺点将不复存在，反而会成为优点，即分别部署，层次清晰，易于维护。

3.2.2 分布式

分布式作为业界主流之一，其本质即为将一个系统的各个组件（如前端、后端、数据库等）分布在网络中的各台主机之上，并且各组件之间仅通过指定端口交互数据来通信以实现负载均衡。本项目中媒体自动化分发组件需要通过分布式架构来实现。

3.3 功能模块

以下内容该系统各个功能模块的详细介绍，本人将围绕项目本身的业务展开描述，系统间各模块的联系如图 3-2 所示。



图 3-2 系统各模块间的联系

3.3.1 客户前端（官网）

对于中小型新媒体企业而言，一个展示公司自身实力的官网是必备项。如何让有对口需求的客户快速精确地找到我们，且了解我们能提供的服务，是这个官网的主要职责。根据上述需求，我们设计了以下包含电脑端和移动端的栏目展示区：

(1) 文章展示。

此栏目将收录与展示公司采编过的各类文章，包括代理公众号的推文、发布于腾讯页卡的特类新闻、公司经手项目的项目记录等；其中分三种方式展示，其一为官网主页的不规则文章展示框，文章配图、文章标题、文章导语将会以一种符合大众审美的方式在此处进行呈现，如图 3-3 所示；其二为文章列表页，所有发布上架过的文章都会被归档在这个目录之下，用户可以通过搜索功能检索出标题与简介包含特定关键字的文章；其三为文章详情页，该页面除了展示指定文章的作者、发布时间和详细的正文内容外，其右侧还会出现推荐阅读栏，其中将根据用户的浏览喜好来推送特定的推荐文章。



图 3-3 官网主页不规则文章展示框

(2) 视频展示。

该栏目会展示公司制作过的各类视频，包括企业宣传片、政府单位宣传片、活动流程直播录像等；与上述文章展示栏目一样，也分几种呈现形式，其一为官网主页的伪轮播视频展示组件，视频封面、视频标题将以一种清晰明了的方式在此处呈现，如图 3-4 所示；其二为官网主页的大型聚焦区，管理员将定义某条公司主推的视频置于此处进行展示，让

用户一进入官网就能被吸引眼球，通过这个视频了解到我们的核心竞争力。其三为视频列表页，所有发布上架过的视频都会被归档在这个目录之下，用户可以通过搜索功能检索出标题与简介包含特定关键字的视频；其三为视频详情页，该页面除了展示指定视频的作者、发布时间和详细的正文内容外，其右侧还会出现推荐观看栏，其中将根据用户的浏览喜好来推送特定的推荐视频。



图 3-4 官网主页（移动端）伪轮播视频组件

(3) 轮播图展示。

该组件置于官网各页面的头条，滚动播放与公司业务相关的介绍图片。在向客户表达公司文化的同时，也增加了网站的整体美观。

(4) 公司信息展示。

该页面陈列了公司的主营业务、公司文化、对外平台、企业文化等资料，让客户更深入地了解公司。今后还会再度更新优秀员工、获奖项目等模块，进一步打造公司的对外形象。

3.3.2 管理前端

除了最重要的客户前端（官网），管理前端也是不可缺少的。它负责管理官网，以及小程序乃至媒体分发的各项数据，是整个系统的控制核心。

(1) 员工功能。管理系统的员工需要对上述各项内容进行增删查改（Create & Retrieve & Update & Delete, CRUD），不过仅限于修改部分数据，各条目最终是否展示或分发取决于管理员的审核。图 3-5 展示了该页面的大致设计与布局，其中员工可进行如下操作：

- ① 官网的文章、视频、轮播图的增删查改；
- ② 小程序的活动、投票对象、赞助商、公告信息、投票记录、轮播图的增删查改；
- ③ 内容分发的文章、视频的增删查改；
- ④ 个人信息的修改；



图 3-5 管理前端的员工文章管理界面

(2) 管理员功能。管理员除了具备所有普通员工已有的权限外，还具有审核员工发布的内容的功能，以及对员工的管理，图 3-6 展示了该页面的大致设计与布局，详情如下：

- ① 员工的增删，其中删除为软删除，后期可恢复；
- ② 用户的分权，可以定义用户为普通员工或管理员；
- ③ 网站内容的审核，对员工发布的文章、视频、轮播图的审核，可驳回、待定或通过。
- ④ 网站内容的上架，对已审核文章、视频进行上架或下架操作。
- ⑤ 分发内容的审核，对员工发布的文章、视频进行审核，可驳回、待定或通过。

其中当分发内容通过审核后，将由分发后端推送至各大众平台，各平台过审后返回视频源，同时更新数据至网站媒体表，员工需对返回的条目作二次修改，且管理员需二次审核才能发布到官网。

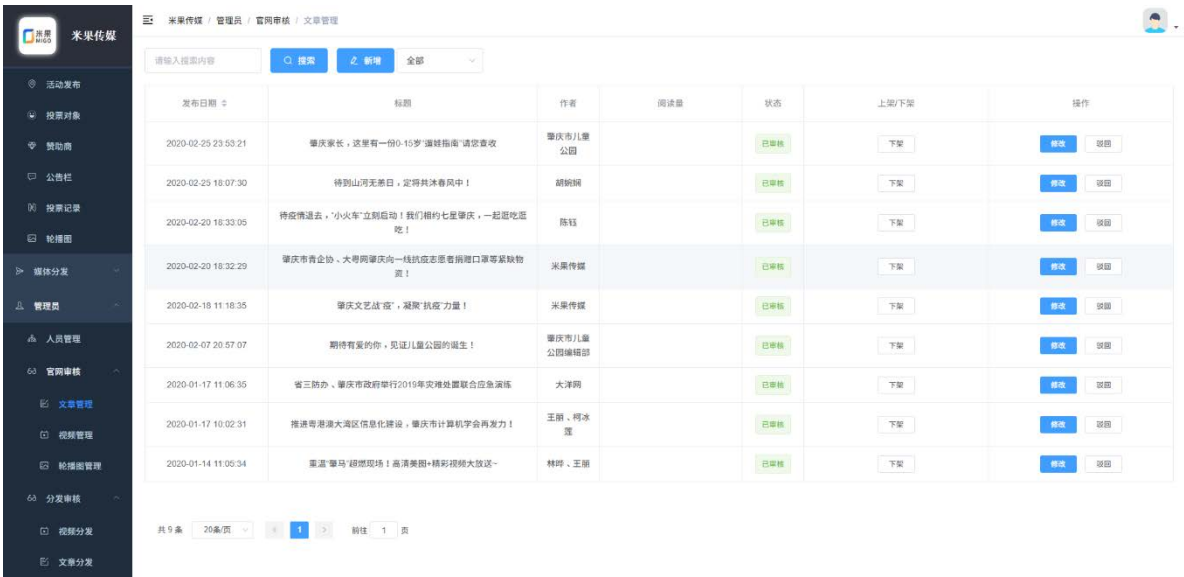


图 3-6 管理前端的管理人员文章审核界面

(3) 登陆功能。每位用户都需要通过该功能获取权限来进入系统，若填写的登陆信息（账号和密码）与数据库里保存的不一致，系统会提示用户重新输入；不同用户组仅可访问自己权限下的内容。

3.3.3 小程序前端

针对上述活动直播与互动相关的需求，我们最终决定通过设计一个集成各项互动交流业务于一身的小程序来实现，图 3-7 展示了小程序的大致设计与布局，具体功能模块如下：

(1) 投票模块。用户通过直播浏览活动中的参演单位，为自己喜爱的对象投票。其中用户可以点击某一特定对象进入详情页面，更深入地了解当前对象的背景信息；当用户决定为该项目投票时，点击对应投票按钮，若此为该用户第一次投票，即为相应对象增加一票；若非首次投票，则拒绝用户的重复投票。此外，还设计该模块具备防止恶意刷票的功能。

(2) 公告展示。根据活动流程按时间显示当下环节的相关信息，展示该环节的名称和简介，历史公告将按时间逆序排序。支持到点自动刷新，实时更新公告内容。

(3) 活动直播。通过桥接腾讯新闻 APP 后端的内部推流接口，为用户提供稳定的直播视频流，未来版本迭代中将会添加发送弹幕的功能。

(4) 赞助商展示。为活动赞助商留下的一席之地，精简展示赞助商推广信息，并通过将观众引导至对应微信公众号主页的方式间接实现招商引流。



图 3-7 活动直播小程序各界面一览

3.3.4 内容分发后端

此后端包含消息队列模块和 Python 自动化模块，上继 Spring Boot 数据综控后端，下

承 MySQL 数据库，通过消息队列 + 自动化脚本实现内容的多平台分发，用于分发文章或视频至多个全媒体平台，目前程序可进行今日头条和哔哩哔哩这两个第三方平台的分发。

(1) 文章分发。

员工通过指定页面创建新文章，经由管理员审核通过，系统自动将对应条目的信息（文章标题、正文内容、文章头图、作者名称）推送至消息队列（生产者），作为消费者的 Python 自动化模块监听到指定队列有数据积攒时，自动拉取对应条目，将其中的内容向多个第三方平台的分发接口进行数据输出，输出完后接收平台的返回值，若发布成功，将该返回值写入对应条目的指定属性中，然后拷贝该条目至网站媒体内容下，等待员工的二次修改与审核以发布至官网；若发布失败，返回失败原因至管理前端。

(2) 视频分发。

员工通过指定页面创建新视频和与之对应的视频信息，经由管理员审核通过，系统自动将对应条目的信息（视频标题、视频片段、视频简介、视频封面、创作团队、视频标签）推送至消息队列（生产者），作为消费者的 Python 自动化模块监听到指定队列有数据积攒时，自动拉取对应条目，将其中的内容向多个第三方平台的分发接口进行数据输出，输出完后接收平台的返回值，若发布成功，将该返回值写入对应条目的指定属性中，然后拷贝该条目至网站媒体内容下，等待员工的二次修改与审核以发布至官网；需要注意的是，由于视频通常而言所占存储空间较大，为不造成系统空间冗余，每次发布成功后将会自动删除已上传的视频片段以节省空间；若发布失败，返回失败原因至管理前端。

3.3.5 数据综控后端

此后端相当于实现管理前端中用户操作的抽象类，各项功能在前文均有提及，即直接与数据库通信，根据用户的操作完成对数据库各项数据条目各种形式的增删查改，处理前端布置的定时任务，控制数据分发后端，并根据实际情况返回对应返回值。

3.4 数据库设计

为完整地容纳整个系统所需的数据，本系统的数据库设计为包含以下业务需求的各项数据表；其中值得注意的是，每个表都必然包含这六个字段：{编号（数据表主键），创建时间，创建人，更新时间，更新人，是否为非激活}，下边详细的介绍会省略上述字段。系统数据库总体 E-R 图如图 3-8 所示。

(1) 官网相关。与客户前端（官网）相关的数据表，此类数据表用于存储与公司相关的各类展示信息，每个实体定义的字段如下：

① 轮播图（官网）信息表：{轮播图存储地址，轮播图名称，轮播图状态，轮播图审核人}

② 文章信息表：{文章正文，文章标题，文章作者，审稿人，阅读量，文章状态，文章头图}

③ 视频信息表：{视频标题，视频简介，视频链接，视频 aid，视频 bvid，视频竖版封面，视频横版封面，视频创作日期，视频状态，视频创作团队，视频审核人}

④ 员工信息表：{员工姓名，员工昵称，登陆密码，员工权限}

(2) 小程序相关

① 活动信息表：{活动名称，活动简介，活动是否激活，活动直播链接}

② 轮播图（小程序）信息表：{轮播图存储地址，轮播图名称}

③ 活动公告信息表：{活动编号（外键），公告名称，公告正文，展示开始时间}

④ 投票记录信息表：{投票对象编号（外键），用户昵称，用户 OPENID，投票时间}

⑤ 投票对象信息表：{活动编号（外键），对象组名，对象简介，对象介绍图，对象介绍缩略图，对象获票数}

⑥ 赞助商信息表：{活动编号（外键），赞助商名称，赞助商信息，赞助商图标}

(3) 分发相关。

① 分发文章信息表：{分发文章正文，分发文章标题，分发文章作者，分发审稿人，分发文章状态，分发文章头图}

② 分发视频信息表：{分发视频标题，分发视频简介，分发视频链接，分发视频 aid，分发视频 bvid，分发视频横版封面，分发视频创作日期，分发视频状态，分发视频创作团队，分发视频审核人}

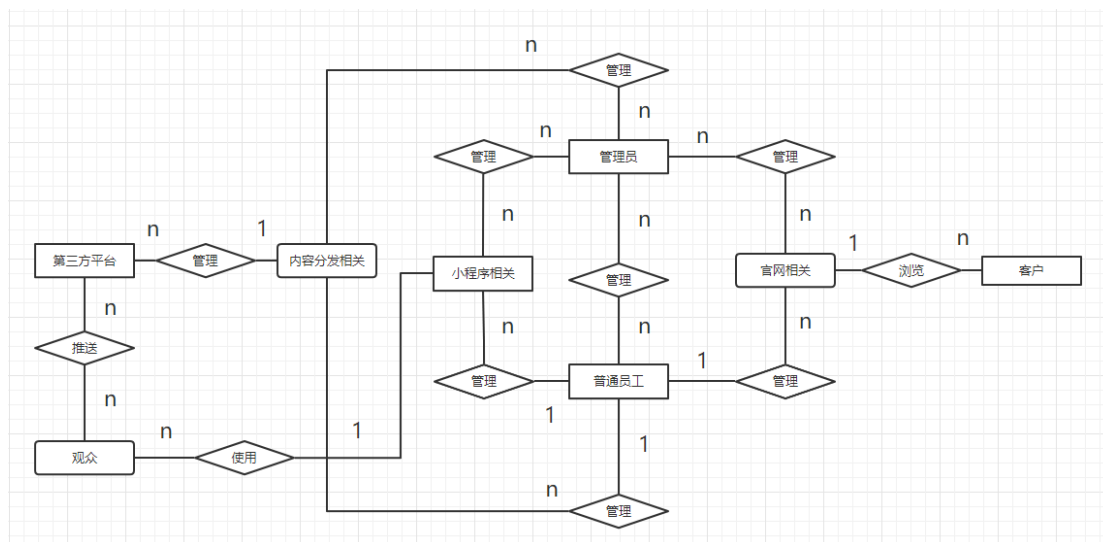


图 3-8 系统数据库总体 E-R 图

4 系统实现

4.1 Spring Boot 后端

用作快速开发的 Spring Boot 框架主要依赖于注解驱动设计模式，它将各类业务以结构

单元的模式抽象出来后提供给开发者^[4]，而作为开发者的我们，通过注释驱动的方式将需要加入的功能或模块注入 Bean 即可完成自动装配，它主要由以下多个可执行结构组成：

4.1.1 控制层（Controller）

控制层即为校验与包装数据和转发业务的逻辑层，它负责监听前端发来的各类请求、定义数据 URL 接口、包装返回数据的格式、将数据请求提供给服务层、调用其他服务模块（例如消息队列）等业务，典型代码案例如图 4-1 所示，以下将结合各项功能详细说明。

```
@ApiOperation("分页查找所有标题或者内容包含该关键字的小程序帮助晒（录入活动页面用，「keywords」为空时返回所有）")
@PostMapping("/mp_merchant/find_all_by_keywords")
public Result<PageResult<MerchantVo>> mpMerchantFindAll(@RequestBody SearchDto searchDto) {
    Result<PageResult<MerchantVo>> result = new Result<>();
    Page<MerchantVo> page;
    try {
        if (searchDto.getKeywords() == null || "".equals(searchDto.getKeywords())) {
            page = merchantService.findMerchantByKeywords( keywords: "", searchDto.getPage(), searchDto.getSize(), searchDto.getDirection());
        } else {
            page = merchantService.findMerchantByKeywords(searchDto.getKeywords(), searchDto.getPage(), searchDto.getSize(), searchDto.getDirection());
        }
        PageResult<MerchantVo> pageResult = new PageResult<>();
        pageResult.setSize(searchDto.getSize()).setPage(searchDto.getPage()).setData(page.getContent()).setTotal(page.getTotalElements());
        result.setMessage("success").setCode(HttpStatus.OK).setData(pageResult);
    } catch (Exception e) {
        System.out.print(e);
        result.setMessage("fail").setCode(HttpStatus.OK).setData(null);
    }
    return result;
}
```

图 4-1 控制层典型代码案例截图

(1) 请求方式。

常见的请求方式有 POST/GET/PUT/... 等，对应上述各项请求的注释分别为 @PostMapping（接收前端发来的自定义类型数据）、@GetMapping（接收前端发来的请求类型数据）、@PutMapping（接收前端发来的已定义类型数据），他们都是与 @RequestMapping 结合的组合型注解。

① @PostMapping 的应用场景：结构较为复杂的数据的包装与传输，例如查找表内符合某查询条件的全部条目、上传一组包含用户信息的自定义格式的数据以执行登陆验证、增加与修改某一个条目各字段的信息等。

② @GetMapping 的应用场景：接收前端发来的请求类数据，返回值不限，例如以分页的方式查找一组媒体内容（文章或视频），根据唯一索引查找单个数据条目、不设任何接收值，仅需请求接口即返回对应查询信息等。

③ @PutMapping 的应用场景：在数据类型与前端约定好的情况下进行传输，常用于数据的变更，即对表内某条数据的修改或对符合条件的某一批条目批量更新。

(2) 定义数据 URL 接口。

与请求方式相对应的，还有对接口的自定义，上述各注解后皆应加路径（例如：@PostMapping("/account/update")），同时，若需要不带请求方式地定义父接口，可以通过在

对应父类上添加注入方法“@RequestMapping + 路径”来实现。

(3) 包装返回数据的格式。

在复杂的业务中，后端返回的值应该不仅仅包含特定业务需求的数据种类，而应当加入响应码、状态描述这些能够精确描述后端实时状态的参数，将他们和返回值通过包装成泛型参数的方式一并返回；若业务涉及分页返回，还需多重包装，将请求时发来的分页信息一并装入返回。

(4) 将请求下发给服务层（Service）。

控制层与服务层区别最大的地方就在于它会综合地处理某项业务，打个比方，整个程序是一个巨大的工厂，那么服务层是生产者，数据访问层为其提供原料；而控制层是消费者，消费从服务层获取的资料，然后再把包装处理好的数据返回给前端；而它在收到某项请求的时候，会将请求下发给服务层以获取资料，而服务层返回的信息通常是不携带除去需求以外的任何元素，或仅经过简单转换过的元数据。

(5) 第三方模块的调用。

在该系统中，共有两处需要控制层调用第三方模块：

① 微信小程序的静默授权。

此过程是微信官方为保护用户隐私与信息安全而专门设计的，首先小程序通过用户的进入，会自动获取一个临时登录凭证“code”，通过将该凭证发送至开发者服务器（后端），然后由后端将该凭证与另外两个值（APPID 和 AppSecret）包装好后，直接请求微信服务接口以获取“session_key”、“OPENID”和“UNIONID”这三个（也有可能只有两个）和用户直接相关的字段的值。接收小程序发来的临时登陆凭证，并发往开发者服务器获取用户隐私值，然后进一步处理获取的信息，这一较为繁琐的任务就交给系统里负责小程序用户验证的控制层来处理了。

首先通过构造访问网址的方式拼接出请求地址，然后交由 restTemplate 类向验证服务器发送 HTTP 请求，接着用 mapper 类中的 readValue 方法将获取到的值与数据传输对象（Data Transfer Object, DTO）OpenIdJson.class（其中包含上述用户信息的多个属性）一一配对，最后向前端返回这个传输对象。

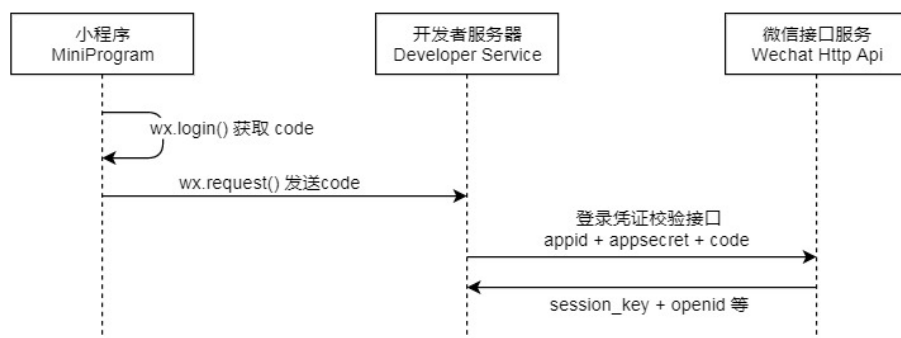


图 4-2 微信小程序静默授权流程时序

② 作为消息队列的生产者。

在内容分发模块中，当管理员通过了某项媒体内容的审核时，系统立刻就会开始执行相关内容的分发流程。负责接收分发数据的控制层收到分发指令后，首先会构造一个 `HashMap`，即“基于哈希表的 `Map` 接口的实现^[4]”，将与之相关的数据通过 `put` 方法统统存入其中，此时该数据也可称作“消息”，接着将消息绑定在路由键“`videoSendingRouting`”并发送到交换机“`videoSendingExchange`”上（有关消息队列的处理逻辑请参见本文章节 4.5，此处不作展开），此时队列“`videoSendingQueue`”即可接收并将此条消息压入队列，供消费者使用，`Spring Boot` 中生产者配置如图 4-3 所示。

```
@Bean
public RabbitTemplate rabbitTemplate(final ConnectionFactory connectionFactory) {
    final RabbitTemplate rabbitTemplate = new RabbitTemplate(connectionFactory);
    rabbitTemplate.setMessageConverter(producerJackson2MessageConverter());
    return rabbitTemplate;
}

@Bean
public Queue videoSendingQueue() { return new Queue( name: "videoSendingQueue", durable: true); }

DirectExchange videoSendingExchange() { return new DirectExchange( name: "videoSendingExchange"); }

@Bean
Binding bindingVideoSending(){
    return BindingBuilder.bind(videoSendingQueue()).to(videoSendingExchange()).with( routingKey: "videoSendingRouting");
}

@Bean
public Jackson2JsonMessageConverter producerJackson2MessageConverter() {
    return new Jackson2JsonMessageConverter();
}
```

图 4-3 `Spring Boot` 生产者配置代码

4.1.2 服务层 (Service)

接收由上述控制层发来的数据请求，实现具体的业务逻辑。在本项目中，它负责接收并转换由数据访问层发来的数据、为各对象的多种属性赋值；其中，请求数据访问层的方式可以有两种：其一是直接使用已经被封装好的 `JAVA` 持久化 API（`Java Persistence API`, `JPA`），其二是直接构造出具体的 `SQL` 语句，通过 `Hibernate` 连接并发送语句操作数据库。典型代码案例如图 4-4 所示。

```
public Merchant findOneById(String id) { return merchantRepository.findMerchantById(id); }

public void update(Merchant merchant){
    Merchant merchantTemp=this.findOneById(merchant.getId());
    BeanUtils.copyProperties(merchant, merchantTemp);
    merchantTemp.setUpdateBy((String) session.getAttribute( s: "user"));
    merchantTemp.setUpdateAt(new Date());
    merchantRepository.saveAndFlush(merchantTemp);
}
```

图 4-4 服务层典型代码案例截图

(1) 数据类型的转换。

以系统中的一项业务举例，在管理员的审核模块中有一个按“上架与否”条件分类媒体的选项，此时后端将接收前端发来的代表上架（True）或下架（False）的布尔型数据，该数据原始状态为 String 类型，须通过 Boolean.parseBoolean 方法转型，此步骤在对应审核查询方法的服务层进行。

(2) 为对象的属性赋值。

在对表的更新操作中，为了以最精简、效率最高的方式实现对象属性的赋值，我们使用 BeanUtils.copyProperties 方法来完成，且为符合开闭原则，防止接口冗余，该方法被限定仅在服务层中使用。这一方法接收两个属性完全相同，或后者包含前者所有属性（ $A \subseteq B$ ）的对象，它会将前者属性携带的所有值覆盖在后者之上，若有属性前者不包含，则保留原样而非置为 NULL。

(3) 请求数据访问层。

这是服务层最核心的功能，如上文所言，有两种方式可以完成请求：

① 直接使用已被封装完毕的 JPA，如 JpaRepository.class、CrudRepository.class 等，这些类中已经封装好了一系列操作数据库的方法可供开发者使用，若所涉及的业务仅使用这类相对简单的方法可以完成，则无需自建与服务层相对应的数据访问对象。

② 建立与实体表相对应的数据访问对象，通过 @Query 注解实现自定义查询方法，支持命名参数及索引参数的使用。其中查询语句可以是 MySQL 原生语句，也可以是针对实体类的变体语句，由参数 nativeQuery 定义。

4.1.3 数据访问对象（Data Access Object, DAO）

数据访问对象是服务层中业务逻辑的具体实现，该对象使用 Hibernate 直接连接并操作数据库，并将返回的数据根据持久层或表现层的定义来包装。其中，通过冒号、井号、问号等缩略占位符来绑定查询函数中的参数，典型代码案例如图 4-5 所示。

```
/**
 * 通过id找激活的投票对象
 * @param id
 * @return
 */
@Query(value = "select * from vote_merchants WHERE activity_id = :#{id}", nativeQuery = true)
List<Merchant> findActiveMerchant(String id);
```

图 4-5 通过活动 id 在赞助商列表中找到对应条目的查询语句

若相应对象为表现层，则需通过创建对应实例的方式将参数与实体结合起来，典型代码案例如图 4-6 所示。


```

/**
 * 分页查找所有标题或者内容包含该关键字且未被软删除的活动（录入活动页面用，「keywords」为空时返回所有）
 * @param keywords
 * @param pageable
 * @return
 */
@Query("SELECT new com.miguo.matrix.vo.miniprogram.MerchantVo(" +
    "g.id," +
    "g.createBy," +
    "g.updateBy," +
    "g.createAt," +
    "g.updateAt," +
    "g.isDel," +
    "g.activityId," +
    "g.merchantName," +
    "g.merchantInfo," +
    "g.merchantLogo," +
    "a.activityName)" +
    "FROM Merchant g " +
    "LEFT JOIN Activity a ON g.activityId=a.id WHERE g.merchantName LIKE %:#{keywords}% OR g.merchantInfo LIKE %:#{keywords}%")
Page<MerchantVo> findMerchantByKeywords(String keywords, Pageable pageable);

```

图 4-6 根据赞助商关键字找与其绑定的活动

4.1.4 持久对象（Persistent Object, PO）

该对象作为实体类之一，与数据表中的字段逐个对应，且定义他们的数据格式，亦可以初始化部分需要提前赋值的属性。在这种对象的包装下，实体类脱离了传统意义上需要逐个定义 set 方法和 get 方法的繁琐取赋动作，简化了开发流程。典型代码案例如图 4-7 所示。

```

@EqualsAndHashCode(callSuper = true)
@Data
@Slf4j
@Entity
@Accessors(chain = true)
@ApiModel("小程序赞助商实体")
@TableName(name = "vote_merchants")
public class Merchant extends BaseEntity {

    @Column(name = "activity_id")
    @ApiModelProperty("活动id")
    private String activityId;

    @Column(name = "merchant_name")
    @ApiModelProperty("赞助商名称")
    private String merchantName;

    @Column(name = "merchant_info")
    @ApiModelProperty("赞助商信息")
    private String merchantInfo;

    @Column(name = "merchant_logo")
    @ApiModelProperty("赞助商图标（链接地址）")
    private String merchantLogo;
}

```

图 4-7 小程序赞助商实体表持久对象

4.1.5 表现对象（Value Object, VO）

它是建立在上述持久对象之上的一项针对系统业务而设立的虚拟表现对象，表现对象

只包含业务需要用到的数据，而其他数据则根据系统安全性原理和数据解耦规则实行隔离。和持久对象一样，它也包含定义 set 方法和 get 方法的隐藏属性。典型代码案例如图 4-8 所示。

```
@Data
@Slf4j
@NoArgsConstructor
@AllArgsConstructor
@Accessors(chain = true)
@ApiModel("文章Vo实体")
public class ArticleVo{

    private String id;

    private String article;

    private String title;

    private String pic;

    private String author;

    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss", timezone = "GMT+8")
    private Date createdAt;
```

图 4-8 官网文章缩略展示业务的表现对象

4.1.6 数据传输对象 (Data Transfer Object, DTO)

这是一种较为独特的数据封装对象，它可脱离于数据库字段，适用于封装那些频繁传输于前后端的数据，类似于表现对象，典型代码案例如图 4-9 所示。

```
/**
 * @author Hocassian
 */
@Data
public class SearchDto {
    private int page;
    private int size;
    private String keywords;
    private Sort.Direction direction;
}
```

图 4-9 进行普通查询的数据传输对象

4.1.7 简单无规则 JAVA 对象 (Plain Ordinary Java Object, POJO)

实际上这就是普通的 JavaBeans，即独立的、实现某种特定功能的工具类或接口，在该系统中，具体表现为时间戳转换、生成随机数（雪花 id）、定时任务、微信小程序登陆校验等功能。

4.1.8 JAVA 持久化 API (Java Persistence API)

与数据访问对象不同, JPA 封装了一系列数据库的操作方法, 部分简单的 CRUD 业务可以直接使用, 缺点在于集成度较高, 不适用于自定义的查询。

4.2 Vue.js 前端

这是一套用于构建用户界面的渐进式 JavaScript 框架, 与其它大型框架不同的是, Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层, 方便与第三方库或既有项目整合^[3]。值得一提的是, 生产与开发环境的快速衔接, 方便快捷的打包部署与完备的混淆加密机制令无数互联网开发者获得了更高的开发效率和知识产权的保护, 这些优点让它成为了近些年来最受国内外各类 IT 企业青睐的前端框架。

4.2.1 视图组件 (Component)

该组件分两大部分, 全局组件和局部组件, 前者通过路由组件注入的方式, 装配为指定页的导航栏、页头、页尾等; 后者通过特定页面的脚本来引用, 而后通过数据绑定实现组件间传值, 图 4-10 展示了管理前端中轮播图管理页各组件的引用情况。



图 4-10 管理前端轮播图管理页各组件的引用一览

将功能组件化意味着解耦, 而优质的解耦则需要具备良好的可维护性和可重用性。对于父子组件而言, 他们的交流方法为: 父组件向子组件传递属性, 而子组件调用父组件的各类事件, 通过事件包含数据间接传值。图 4-11 举例了图片上传组件中的传值与对事件的调用。

```
<my-upload
  method="POST"
  field="file"
  v-model="cropperShow"
  :width=1920
  :height=450
  :url="this.$store.state.settings.uploadUrl"
  lang-type='zh'
  img-format='jpg'
  img-bgc='#FFF'
  :no-circle=true
  @crop-upload-success="cropUploadSuccess">
</my-upload>
```

图 4-11 图片上传组件的传值与事件的调用

4.2.2 页面容器（View）

Vue.js 的模板建立在普通 HTML5 语法的基础之上，它允许开发者声明式地将 DOM 绑定至底层 Vue 实例的数据，而后将模板编译成虚拟 DOM 渲染函数^[3]。典型的 Vue 页面分为三大模块：模板语言（template）、运行脚本（script）、布局样式（style）。

① 模板语言。

相当于一个页面的表示层，通过 Mustache 语法（双大括号）绑定数据，使用 slot 插槽引入上述视图组件，以及运用以”v-”开头的各类事件处理器处理模板内的各类逻辑，还有子组件注册索引关键字”ref”等。此处引入一个具体的案例来解释模板语言的实现逻辑：

```
<el-table-editabled v-model="tableData" :columns="[ 'groupName', 'groupProfile', 'groupPicHd', 'activityName', 'activityId']" ref="editTable">
  <el-table
    :key="tableKey"
    v-loading="listLoading"
    :data="tableData"
    border
    fit
    highlight-current-row
    style="..."
    @sort-change="sortChange"
    @selection-change="handleSelectionChange"
  >
    <el-table-column type="selection" width="50px" align="center"...>
    <el-table-column label="发布日期" prop="date" sortable="custom" width="200px" align="center">
      <template slot-scope="{row}">
        <span>{{ row.updateAt }}</span>
      </template>
    </el-table-column>
    <el-table-column label="小组名称" min-width="80px" align="center"...>
    <el-table-column label="小组简介" min-width="200px" align="center"...>
    <el-table-column label="小组获赞数" min-width="100px" align="center"...>
    <el-table-column label="小组图片" min-width="100px" align="center"...>
    <el-table-column label="小组对应活动" min-width="100px" align="center"...>
    <el-table-column label="操作" align="center" width="300px" class-name="small-padding fixed-width"...>
  </el-table>
</el-table-editabled>
```

图 4-12 小程序投票对象录入的实现模板

如图 4-12 所示，这是小程序投票对象录入的实现模板，最外层<el-table-editabled>标签表示我们使用的是表格内可编辑组件，v-model=”tableData”表示双向绑定与之相对的表单数据，:columns 数组中的元素表示可编辑的属性；内一层的<el-table>即为该组件的子组件，

包含普通表单的各类参数；其子项<el-table-column>可通过 slot-scope 定义作用域插槽，绑定对应 row 的属性。

最终实现效果如图 4-13 所示：

<input type="checkbox"/>	2020-02-07 19:02:18	直升机快速转运伤员	现场指挥官向空中救援组下达直升机转运伤员的命令。空中救援组根据人员与地面指挥组联络，将直升机到达现场上空，并安全降落。在停机点后，救援人员跑步到安全区域，在地面救援人员协助下，用担架将重伤员抬上飞机。救援直升机起飞，转运伤员到演练场外另外一个停机点。	12	<button>更新照片</button> <button>预览</button>	省三防办 肇庆市灾难应急演练	<button>编辑</button> <button>删除</button>
<input type="checkbox"/>	2020-02-07 19:02:11	无人机空中灾情评估	指挥组专家通过无人机传回来的画面与演练区、肇庆新城区	92	<button>更新照片</button> <button>预览</button>	省三防办 肇庆市灾难应急演练!	<button>更新</button> <button>删除</button>
<input type="checkbox"/>	2020-02-07 19:01:05	灾后保障	台风、暴雨、雷电造成大面积停电和通信设施受损，供电、供水、电信部门组织抢修。供电、供水、电信部门出动应急发电车、电力抢修车、龙吸水，通信抢修车等专业大型抢险设备到现场抢险。“环境监测组运用环境监测和检测设备，对水源指标进行监测。环境监测组长向现场指挥组报告监测情况。”疾控中心人员穿戴防护服开展消杀药剂配备、喷洒，并开展“突发”流行性传染病调查。	113	<button>更新照片</button> <button>预览</button>	省三防办 肇庆市灾难应急演练	<button>编辑</button> <button>删除</button>

图 4-13 小程序投票对象录入的最终效果

② 运行脚本。

在<script>标签内包含各种组成脚本的元素，比如通过 import 引入某个 js 模块，可以当做普通参变量引入模板的 computed 计算属性，负责监听模板的 watched 监听属性，模板中的各种参变量的集合 data 函数，包含各类函数的普通方法属性 Methods 等。此处引入具体案例“前端登陆的逻辑脚本”来解释运行脚本的各组成元素。

a. 模块引入与 data 函数。对于用户登陆逻辑，需要引入包含用户信息接口调用的 js 模块和状态管理库 vuex 中的全局变量（以 mapGetters 辅助函数的形式取值），data 函数除去包含用户名和密码这些必要属性外，还包含 loginRules 校验方法，规范用户的行为模式，如图 4-14 所示：

```
import LoginApi from '@api/website/LoginApi'
import { mapGetters } from 'vuex'
export default {
  name: "Login",
  data() {
    return {
      loginForm: {
        nickname: '',
        password: ''
      },
      loginRules: {
        nickname: [{ required: true, trigger: 'blur', message: '请输入账号!' }],
        password: [{ required: true, trigger: 'blur', message: '请输入密码!' }],
      },
      loading: false,
      passwordType: 'password',
      redirect: undefined,
      user: []
    }
  },
}
```

图 4-14 前端登陆脚本中的模块引入与 data 函数

b. 计算属性与监听属性。如图 4-15 所示，computed 中的...mapGetters 即为取出仓库中全局变量”roles”的值，watch 中的\$route 即为监听用户登录后路由跳转的路径。

```
computed: {
  ...mapGetters([
    'roles',
  ])
},
watch: {
  $route: {
    handler: function(route) {
      this.redirect = route.query && route.query.redirect
    },
    immediate: true
  }
},
}
```

图 4-15 前端登陆脚本中的计算属性与监听属性

c. 方法属性。如图 4-16 所示，methods 中包含多个函数实例，其中 submitForm() 为登陆校验函数，其逻辑为先回传用户输入值到后端判断，将返回的角色值载入浏览器缓存 sessionStorage 中，并通过仓库 \$store 中的调度方法 dispatch 为全局变量中的角色属性赋值，最后由 \$router.push 实现跳转。

```
methods: {
  showPwd() {...},
  submitForm() {
    this.$refs["loginForm"].validate((valid) => {
      if(!valid){...} else {
        this.loading = true
        LoginApi.login(this.loginForm).then(data => {
          if (data === 'admin' || data === 'staff') {
            this.user.push(data)
            // console.log(this.user)
            sessionStorage.setItem('user', data)
            this.$store.dispatch('user/getInfo', this.user)
            this.$router.push({ path: this.redirect || '/' })
          }
          else {...}
          this.loading = false
        }).catch(err => {...})
      }
    })
  }
}
```

图 4-16 前端登陆脚本中的方法属性

③ 布局样式。

和其他框架一样，Vue.js 支持层叠样式表（Cascading Style Sheets, CSS）、加强语法样式表（Syntactically Awesome Style Sheets, SASS）和语法层叠样式表（Sassy Cascading Style Sheets, SCSS）。通过三种样式表实现红色点击效果的代码如图 4-17 所示：

SASS	SCSS	CSS
<pre>\$color: red \$color2: lime a color: \$color &:hover color: \$color2</pre>	<pre>\$color: #f00; \$color2: #0f0; a { color: \$color; &:hover { color: \$color2; } }</pre>	<pre>a { color: red; } a:hover { color: lime; }</pre>

图 4-17 SASS、SCSS 和 CSS 分别实现红色点击效果示例

值得一提的是，使用 SCSS 可以实现网页的动画效果，例如在管理前端的导航栏中切换不同选项时会有有一个从左至右逐渐淡入淡出的效果，它通过图 4-18 中的代码实现：

```
/* fade-transform */
.fade-transform-leave-active,
.fade-transform-enter-active {
  transition: all .5s;
}

.fade-transform-enter {
  opacity: 0;
  transform: translateX(-30px);
}

.fade-transform-leave-to {
  opacity: 0;
  transform: translateX(30px);
}
```

图 4-18 通过 SCSS 实现通过导航切换页面时的动画效果

4.2.3 路由（Route）

Vue.js 的路由分为狭义和广义两种概念，前者集成在一个 js 脚本（route.js）上，每个页面对应一个路由对象，包含页面名称、用户访问的路径、页面在项目中实际存放的路径等属性，支持父子页面的嵌套和分类。在管理前端中，基于其特性还增加了用户分权、动态页面渲染等功能，这就不得不提路由器卫士的作用，也就是广义下的路由功能支持了。

路由器卫士会在用户的每一次访问页面时执行，它将会判断用户是否已经登录，是否对当前页面持有访问权限，这需要结合上述 route.js 中的 meta 属性来实现。以该系统的管理前端为例，路由器卫士的具体流程如下图 4-19 所示。

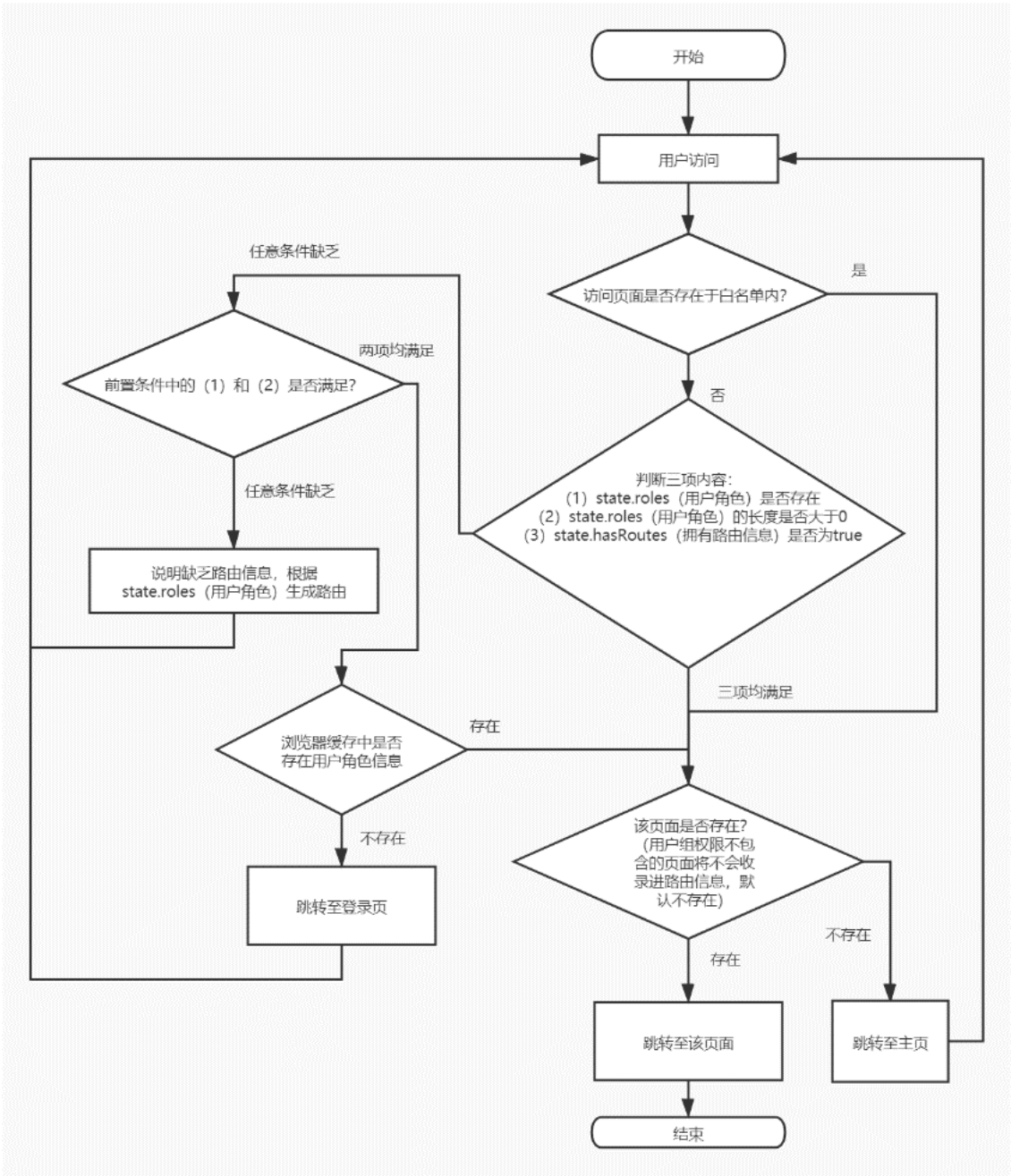


图 4-19 用户访问网站任意网页时路由器卫士的执行流程图

4.2.4 数据通信 (Data Communication)

由于 Vue.js 本身并不包含此项内容，它是根据这一框架衍生出来的封装方法，即实际开发中的上层建筑。故在展开解释之前，首先介绍经常用于 vue 中请求数据的方法 promise。

promise 是 web 开发中常用的异步请求方法，它可以将异步操作队列化，按照期望的顺序执行，返回符合预期的结果^[7]。在 Vue.js 中我们常常需要用到异步请求，因为“请求后端→后端返回数据”这一流程消耗的时间远远长于前端内部处理数据所需的时间。传统的 ajax 标准因不包含响应函数，故无法在它们改变状态时再处理后续步骤，而 promise 中

同级的 `then()` 方法即可实现这一点，当状态经由回调值确定为 `resolve`（处理完成）或 `reject`（处理出错），`then()` 方法内包含的响应函数才开始执行，这有效解决了传统开发中回调地狱（代码嵌套与执行自下而上递归）的问题。

了解了 `promise` 的基本原理后，我们会发现，若在每种接口的请求方法中都进行添加，会造成大量的代码冗余。有没有办法继续精简回调函数的代码呢？方法就是结合 `Axios` 进行函数封装。

`Axios` 是一个基于 `promise` 的 `HTTP` 库，针对原生 `promise` 方法进行了多元的优化，可用于浏览器和 `node.js` 中^[7]。在实际开发中，我们针对每一种请求方法都构造了出了与之对应的 `promise` 异步包装请求函数，其核心请求由 `axios` 结合 `promise` 发出，并将他们集成在同一个工具类（`http-kit.js`）上，除此之外还附加了各类 `Http Request` 拦截器（例如访问接口时判断是否登陆超时等）。

每次需要进行数据通信时，当前页面中的脚本会执行如图 4-20 中的逻辑流程。

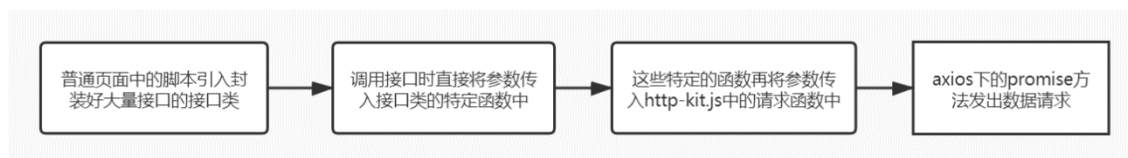


图 4-20 用户访问任意网页时前端接口的数据通信流程

4.2.5 状态管理库（Vuex）

它可以看作是 `Vue.js` 开发框架下的一个可移动单例，作为 `vue` 的全局变量，它包含 `store`（仓库）、`state`（状态）、`getters`（仓库调用）、`mutations`（状态变更）、`actions`（异步状态变更）、`modules`（仓库分区）等多种执行单元，通过它们共享核心单元 `store` 中的数据，完成跨组件间的信息交互^[3]。其中需要注意的是，该仓库中的数据会在浏览器刷新后重置。

如上文管理中管理前端登陆逻辑的例子，`"this.$store.dispatch('user/getInfo', this.user)"` 这条语句调用的正是 `user` 仓库中的 `getInfo` 异步状态变更函数，然后再由状态变更函数 `SET_ROLES` 设定用户身份，如图 4-21 所示。

```

const state = {
  roles: []
}

const mutations = {
  SET_ROLES: (state, roles) => {
    state.roles = roles
  }
}

const actions = {
  getInfo({ commit }, roles) {
    commit('SET_ROLES', roles)
  },
  userLogout({ commit }) {
    commit('SET_ROLES', [])
    resetRouter()
  }
}
  
```

图 4-21 用户登陆功能中通过 `vuex` 获取身份信息的内部逻辑

4.2.6 工具 (Utility)

和 Spring Boot 后端中的简单无规则 JAVA 对象类似，这里的工具指前端一系列独立的、实现某种特定功能的工具类或函数，如在上文数据通信段落中介绍过的请求封装脚本 http-kit.js，时间戳转工具 timeformat.js，根据用户下拉页面的程度来加载网页资源的方法 scroll-to.js 等等。

4.3 小程序前端

这是一种无需下载安装，只需扫一扫即可在线使用，使用完毕也无需卸载，非常方便快捷的微服务框架。相较于传统的原生 APP，微信小程序通过高度集成与优化，已经完成了对各类型手机的优化，从而大大减轻了开发者的调试成本；而其简洁至臻的开发文档，则又一次降低了开发者的学习门槛。本项目使其作为“活动直播与互动”这一需求的前端载体，其后端仍然需要 Spring Boot 的支撑。

4.3.1 用户授权 (Authorize)

如上文 4.1.1 所言，用户初次进入小程序时，会自动获取一个临时登录凭证“code”，通过将该凭证发送至开发者服务器（后端），然后由后端将该凭证与另外两个值（APPID 和 AppSecret）包装好后，直接请求微信服务接口以获取“session_key”、“OPENID”和“UNIONID”这三个（也有可能只有两个）和用户直接相关的字段的值。这一过程便称为“用户授权”，它可以是静默的（仅获取用户 OPENID）也可以是手动的（获取用户昵称、头像、所在地等完整信息）。

在此项目中，由于投票仅需校验用户的唯一性，故使用静默授权，在获取到用户信息后将它们载入缓存，如图 4-22 所示。

```
// 登录
wx.login({
  success: res => {
    // 发送 res.code 到后台换取 openId, sessionKey, unionId
    if (res.code) {
      wx.setStorageSync('wxCode', res.code);
      var _url = this.globalData.serverAddr
      wx.request({
        url: _url + 'user/login',
        method: 'POST',
        data: {
          code: res.code
        },
        header: {
          'content-type': 'application/x-www-form-urlencoded'
        },
        success(res) {
          if (res.data.openid != "" || res.data.data.openid != null) {
            // 登录成功
            wx.setStorageSync("openid", res.data.data.openid); // 将用户id保存到缓存中
            wx.setStorageSync("session_key", res.data.data.session_key); // 将session_key保存到缓存中
            console.log('获取授权成功');
          }
        }
      })
    }
  })
})
```

图 4-22 小程序登陆后静默授权流程

4.3.2 视图组件 (Component)

和 vue 一样, 视图组件亦分为全局组件和局部组件, 前者多为底部导航栏, 在 app.json 内定义其名称与图标; 后者通过引入特定标签和数据绑定实现组件间传值, 子组件通过 this.triggerEvent 来给父组件传递动作信息, 图 4-23 中的投票组件即为一个简单但典型的局部组件例子。

```
<w-vote  
  class="src"  
  votesList="{{votes}}"  
  nickName="{{nickName}}"  
  bind:eventListener="handleEventListener"  
></w-vote>
```

图 4-23 小程序首页引入投票组件

4.3.3 数据通信 (Data Communication)

与 vue 类似, 不过小程序内封装的请求工具类更加精简, 即实现了请求方法的自定义, 每个页面在使用该封装函数的时候, 需传入请求路径、请求方法、请求数据这三个值。如下图 4-24 中所示的获取排名就是一个典型的异步请求方法。

```
getRank() {  
  let that = this;  
  var activityId = app.globalData.activityId  
  var data = {  
    id: activityId,  
  }  
  util.getAPI("GET", data, 'group/find_rank').then((res) => {  
    if (res.data.code === 200) {  
      let dataList = res.data.data;  
      // console.log(dataList)  
      Object.keys(dataList).forEach(key => (dataList[key].groupPicHd = app.globalData.mpCallbackUrl + dataList[key].groupPicHd))  
      that.setData({  
        rankList: dataList  
      })  
    }  
  })  
},
```

图 4-24 小程序投票页获取排名的请求函数

4.3.4 页面容器 (View)

小程序作为一种尤其注重规范的开发方式, 它的各页面由四个基本单元组成: WXML 模板、WXSS 样式、JSON 配置、JS 脚本^[8]。和 Vue.js 框架类比的话, 相当于将一个 vue 文件里的各部分拆出来, 作为一项项单独的个体, template 对应 WXML、style 对应 WXSS、script 对应 JS。由于大部分内容和 vue 异途同归, 故在此不展开赘述。

4.3.5 工具 (Utility)

和 Spring Boot 后端中的简单无规则 JAVA 对象类似, 这里的工具指前端一系列独立的、实现某种特定功能的工具类或函数, 如在上文数据通信段落中介绍过的请求封装脚本, 时间戳转工具, 根据用户下拉页面的程度来加载网页资源的方法等等。

4.3.6 全局变量 (Global)

和 vue 需要引入 vuex 不同的是, 小程序自带全局状态管理库 app.js, 它可以保存当前活动信息、接口配置信息、用户授权信息等, 且刷新时不会失效。如下图 4-25 所示

```
globalData: {
  userInfo: null,
  activityId: '',
  activityName: '',
  activityProfile: '',
  callbackUrl: 'https://migotimes.com/file/',
  mpCallbackUrl: 'https://migotimes.com/file/thumbnails/mp',
  serverAddr: 'https://migotimes.com/client_miniprogram/',
},
/**
 * 校验缓存
 */
checkCache: function () {
  if (wx.getStorageSync("session_key") && wx.getStorageSync("openid") && this.globalData.userInfo) {
    return true
  } else {
    return false
  }
},
},
```

图 4-25 小程序的全局变量 (部分)

4.3.7 路由 (Route)

由于不包含用户分权需求, 故此处保持最简单的功能, 即动态页面的渲染和静态页面的定位。小程序的每个路由都是独立的视图, 可以类比为 vue 中的多个 view, 而能够缓存全局数据是由于小程序实现脚本逻辑的线程是单线程, 而渲染视图则是多线程。

4.4 Python 后端

通过 Python 自动化模拟真人操作, 使用者可以突破网站拦截限制, 实现对各大平台的内容一键分发, 且脱离综控后端, 易于在第三方平台接口更新的时候进行维护。使用 Python 开发程序, 许多功能不必从零编写, 直接调用现成封装好的模块的即可^[5]。

4.4.1 消费者模块 (消息队列)

分发前的第一步就是准备好即将发出的数据, 在启动该后端时, 程序会自动连接服务器中 docker 容器里的 RabbitMQ, 与相对应的队列和路由实行绑定后, 开启监听。只要有消息进栈 (被生产), 程序就会自动令它们逐个出栈并执行 (被消费)。Python 中的消费者配置代码如下图 4-26 所示。

```

# 创建临时队列，队列名传空字符，consumer关闭后，队列自动删除
channel.queue_declare(queue='videoSendingQueue', durable = True)

channel.exchange_declare(exchange = 'videoSendingExchange', durable = True, exchange_type='direct')
# 绑定exchange和队列 exchange 使我们能够确切地指定消息应该到哪个队列去

channel.queue_bind(exchange = 'videoSendingExchange', queue='videoSendingQueue', routing_key='videoSendingRouting')

# 设置成 False，在调用callback函数时，未收到确认标识，消息会重回队列。True，无论调用callback成功与否，消息都被消费掉
channel.basic_consume('videoSendingQueue',self.callback, auto_ack = False)
channel.start_consuming()

```

图 4-26 Python 消费者配置代码

4.4.2 自动化模块（内容分发）

从队列中拿出所需数据后，程序经过以下几个步骤实现分发：模拟安卓端登录→将视频分片上传至平台接口→把视频信息打包为 json 格式上传至平台接口→接收返回信息。如图 4-27 所示，所有信息打包好后一并以 post 的方式推送给第三方服务器。

```

def add():
    r = self.session.post('https://member.bilibili.com/x/vu/web/add?csrf=' + self.csrf,
                          json={
                              "copyright" : copyright,
                              "source"    : source,
                              "title"     : title,
                              "tid"       : tid,
                              "tag"       : ','.join(tag),
                              "no_reprint": int(no_reprint),
                              "desc"      : desc,
                              "cover"     : cover,
                              "mission_id": 0,
                              "order_id"  : 0,
                              "videos"    : videos,
                              "dtime"     : dtime,
                              "open_elec" : int(open_elec),
                              "dynamic"   : dynamic,
                              "subtitle"  : {
                                  "lan" : "",
                                  "open": int(open_subtitle),
                              },
                          },
    )
    return r

```

图 4-27 视频信息打包为 json 格式并 post 给第三方服务器

4.4.3 数据库操作模块（信息回调）

如上图所示，向服务器发完 post 请求后会得到一个回调值“r”，其包含视频地址等信息。获取后再向对应的表中写入它们。需要注意的是，为减少重复操作，默认情况下媒体分发成功后，其内容会拷贝至官网内容中，若该内容需要在官网展示，则需管理员进行二次审核。

4.5 RabbitMQ 消息队列中间件

消息队列中间件在近两年中大型互联网项目的开发中占据十分重要的位置，它负责利用高效可靠的消息传递机制进行与平台无关的数据交流，并基于数据通信来进行分布式系统的集成。通过提供消息传递和消息排队模型，它可以在分布式环境下进行进程之间的通信^[6]。之所以需要 RabbitMQ 消息队列，是因为内容的分发需要时间，尤其是视频的上传，用户上传至公司服务器需要时间，而公司服务器上传至第三方服务器也需要时间，若能将这些步骤并发执行，即同时接收同时发布，工作效率将大大提高。

4.5.1 基本对象

(1) 消息交换机 (Exchange)。它指定消息按什么样的规则，路由到具体哪个队列。该项目中我们选用的是直连交换机 (Direct)，路由到"videoSendingQueue"队列。

(2) 消息队列 (Queue)。作为消息的载体，控制消息的入栈和出栈。只有当消费者完整消费完一个消息，并发出完成指令后，消息队列才会推送下一个消息。

(3) 绑定 (Binding)。它将消息交换机和队列按照路由规则绑定起来，每条消息只会根据路由关键字在绑定好的这条路线上移动，互不干扰。

(4) 路由关键字 (Routing Key)。消息交换机根据这个关键字进行消息的投递，通常用来标注绑定，引导消息的传输。

(5) 虚拟主机 (Virtual Host)。开设多个虚拟主机以实现多层独立的消息处理，例如系统中投递文章和投递视频是两个分离的虚拟主机，它们各自独立互不干扰。

(6) 消息生产者 (Producer)。指投递消息的程序，在本系统中 Spring Boot 后端包含这一角色。

(7) 消息消费者 (Consumer)。指消费消息的程序，在本系统中 Python 后端包含这一角色。

(8) 消息通道 (Channel)。类似 sql 中的建立连接，在客户端的每个连接里可建立多个消息通道以用于数据传输。

4.5.2 处理逻辑

生产者 (Spring Boot) 里生产一条消息，然后通过 TCP 指定端口连接到 RabbitMQ 容器，容器内再建立一条消息通道连接到直连交换机，交换机再根据它定义的绑定规则和消息包含的路由关键字将消息转发到消息队列，如图 4-28 所示，分发信息包装为消息后通过交换机进入队列，然后排队等待取出。

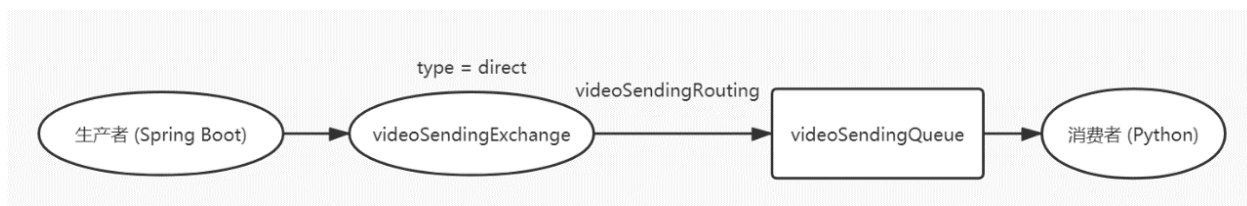


图 4-28 视频自动分发业务流程图

4.6 项目部署

4.6.1 后端部署

(1) 项目打包与发布。首先在配置中更改部署环境为生产环境 (prod)，通过 maven 的开发周期中的 clean 清除完运行冗余后，再使用打包 (package) 生成 jar 运行包，将 jar 包上传至 linux 服务器，新建一个窗口 (screen) 并运行 "java -jar *.jar" 指令 (其中 * 代表包名，若项目部署在 443 端口，还需使用 sudo 提权前缀命令)，然后关闭该窗口即可。此处不建议使用 "&" 后缀令项目后台运行，这会导致关闭项目需要另行撰写停用脚本，不利于项目初期的调试，而且 screen 易于开启，可时刻打开查看运行记录。

(2) 配置 SSL 证书。当开发者需要流经后端的数据使用 https 加密传输协议时，需配置 SSL 证书，即放置证书私钥至项目根目录，然后将证书类型和证书密码写入配置文件，并将端口置为 443，具体配置如图 4-29 所示。

```

server:
  port: 443
  address:
  ssl:
    key-store: classpath:migotimes.com.jks
    key-store-password: *
    keyStoreType: JKS
  
```

图 4-29 在 Spring Boot 中配置 ssl 证书实现 https

4.6.2 前端部署

(1) 项目打包与发布。在 vue-cli 脚手架中，我们通过在终端中执行命令 "npm run build" 会将整个项目打包为一个由 js、css 等静态文件组成的，名称为 "dist" 的文件夹，接着将这个文件夹上传至 Nginx 容器中，再通过宝塔管理面板开放相应端口即可。

(2) 后端接口设置。此项配置在不同版本的脚手架中有不同的写法，而在我们的项目中，管理前端采用 vue-cli 2.0，而官网前端则是 vue-cli 4.0。但无论哪个版本都需要配置这两项接口：一是用户访问的网址和接口，二是前端访问后端的网址和接口。

4. 6. 3 小程序部署

- (1) 发布审核。在完成对代码的编辑后，直接点击微信开发工具面板上的“上传”选项，然后在微信公众平台推送审核，等待审核完毕再选择发布即可。
- (2) 后端接口设置。和 vue-cli 脚手架不同的是，它无需配置访问的网址和端口。需要注意的是，小程序仅支持 https 传输协议，且需提前在管理平台添加合法备案域名，注意对应的小程序 APPID 是否一致。

4. 6. 4 服务器部署

(1) Nginx 反向代理。由于我们需要在同一个服务器上架设多个网站，故选用宝塔网站管理助手，它实现了各项插件运行环境配置的可视化，免去了传统开发中的繁琐部署，也实现了单端口（80、443）的多网站部署，如图 4-30 所示。



图 4-30 使用宝塔面板在单个服务器中创建多个站点

(2) 分布式端口交互。纵观这整个系统，每个独立的框架或组件都要和其他的进行信息交互，具体交互配置如表 4-1 所示：

表 4-1 系统交互端口配置表

交互端口	Vue.js	Spring boot	Python	MySQL	RabbitMQ
Vue.js	/	9090	/	/	/
Spring boot	9090	/	/	3306	5672
Python	/	/	/	3306	5672
MySQL	/	3306	3306	/	/
RabbitMQ	/	5672	5672	/	/
对外管理	27056	22	22	888	15672
生产环境	80	443	/	3306	5672
开发环境	8099	8080	/	3306	5672

4.6.5 代码托管

(1) Git 简介。Git 是一个开源的分布式版本控制系统，用于开发者敏捷高效地处理任何或小或大的项目。其中代码托管的平台有很多，本项目使用 GitHub。

(2) 基本操作。由于是单人开发，故只需用到合并、推送、分支、拉取这几个功能。

(3) 版本管理。在项目开发中，个人对代码推送的版本管理规则如下：若仅修复模块中的 bug，完成小范围的版本更新，则项目版本编号加 0.1，若完成整个迭代周期，即当前周期中所有模块的代码编写，则版本编号加 1。Master 分支合并的代码为各项目的整数版本，若特殊时期需要开发特殊功能，则从当前版本开设新分支开发该特殊版本，特殊时期过后再合并必需代码至源分支，接着贮存该最新分支，从源分支再次进行开发。

5 系统测试

5.1 压力测试

5.1.1 模拟高并发

通过接口测试工具 PostMan 测试后端接口，得到的结果如图 5-1 所示，随着请求的持续涌入，腾讯云和宝塔面板都会智能地延缓请求的响应时间，将它们并入队列，给予 Spring Boot 后端充分的时间来逐一响应，故可见随着时间的流逝，用户的响应时间也会停留在一个合适的长度。

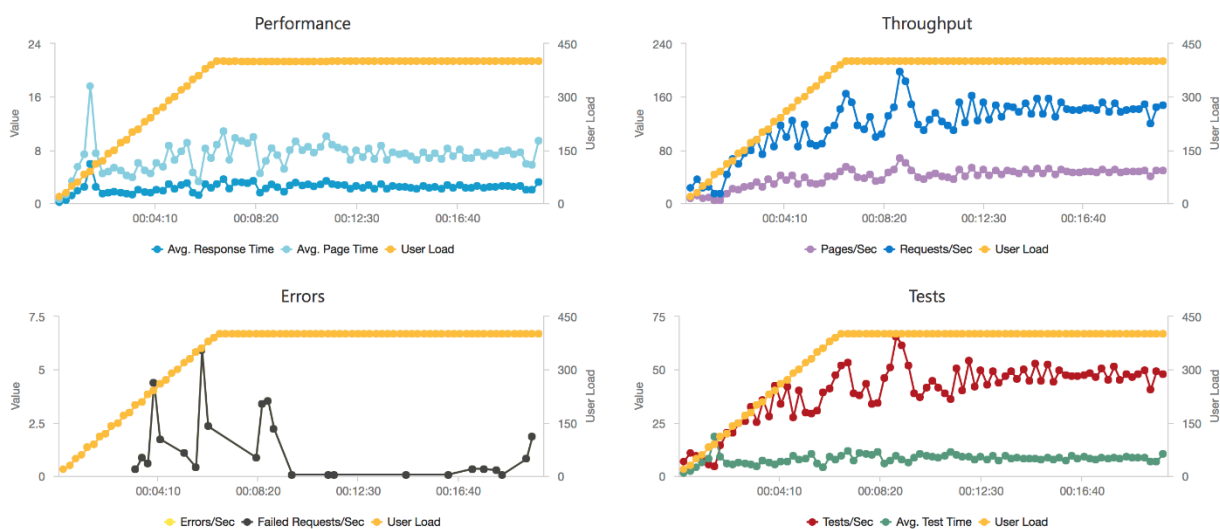


图 5-1 高并发请求测试中后端的响应状况

5.1.2 模拟高负载

使用 Python 脚本绕过腾讯云安全监测，同时在关闭宝塔流量防护的情况下对服务器进行 CC 攻击，模拟攻击次数为 50 万次/秒，测试服务器的抗压能力，得到的结果如下图所示，虽然网站暂时无法访问，但停止测试后一切立刻恢复正常。



图 5-2 裸机高负载测试后服务器运行状况

完成裸机测试后，开启各项防护措施，进行 CC 攻击和 DDOS 攻击，得到的结果如图所示，以上结果说明服务器初步具备防止恶意攻击的能力。



图 5-3 防护状态下高负载测试后服务器运行状况

5.2 逻辑测试

分模块测试完毕后，通过发布上线，进入 α 测试（内测），主要检测几项关键点的实现逻辑，具体测试结果如表 5-1 所示。

表 5-1 系统功能测试结果表（节选）

用例编号	测试功能	测试步骤	预期结果	测试结果
A01	用户登陆	输入账号和密码	提示信息错误或跳转至主页	正常
A02	用户刷新	按下 F5 或点击刷新按钮	重载当前页面	正常
B01	新增条目	点击新增按钮，填写完必要信息后点确认	列表中出现新增条目	正常
B02	查询条目	在搜索框填入关键字后点击查询	列表中出现对应条目	正常

5.3 漏洞检测

使用 Linux 平台最著名的漏洞检测系统 Kali 进行常规注入测试，通过 sqlmap 的多项批量扫描，我们发现最终也未能定向爆破，这说明该系统成功抵御了如今黑客最流行的端口扫描自动注入的入侵方案，测试截图如图 5-4 所示。

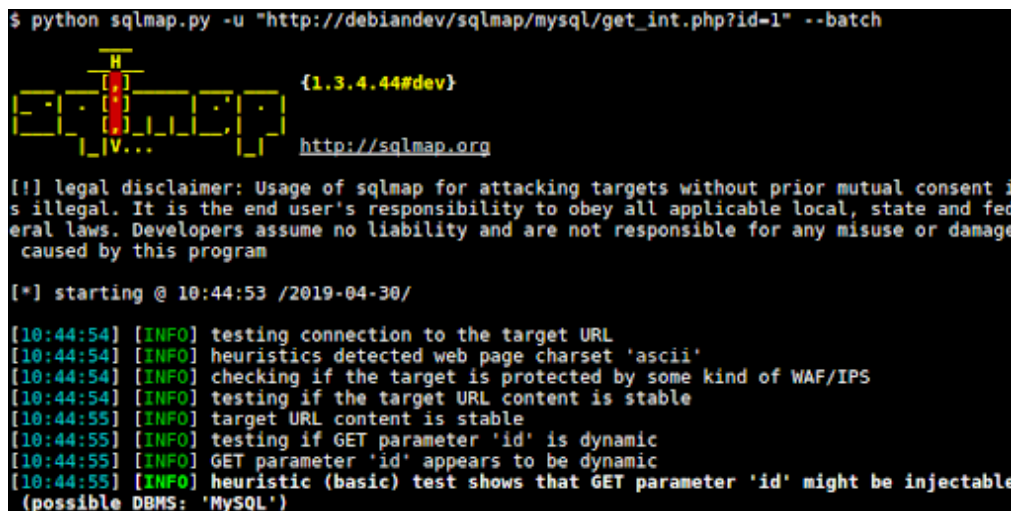


图 5-4 使用 Kali 系统下的 sqlmap 工具的漏洞测试

6 结论

IT 和媒体这两个行业实际上都并非实体产业，而它们具备一个相似的特点，那就是打通个体之间的信息壁垒，创造出更多的联系。与传统媒体不同，信息化时代的新媒体更讲求效率，如何病毒式地扩散新闻和有针对性的投放广告，这些都脱离不开互联网的基座。如何运用好这一工具，去扩宽其他的行业，实际上就是众多互联网公司运营的本质。仅拿内容创作平台举例，微信公众号、优土豆、今日头条、抖音、哔哩哔哩……如果没有互联网，没有一系列技术与框架的支持，它们就如同空中楼阁，而如今也只会留存传统的报纸、电台、电视传媒等形式的媒体类型。

青出于蓝而胜于蓝，长江后浪推前浪。这些依托于 IT 的平台迅速席卷并大有吞并传统媒体单位的形式（得益于政策的保护，官媒继续发光发热），而那些传统的媒体机构，也并没有善罢甘休，还是紧跟形式，尾随其后的央视频、南方 Plus、新华社等官方 APP 相继推出，与互联网私营平台相持不下，最终和平共处。相较于新平台，这些传统媒体单位的转型才是最复杂最艰难的——它们原本就拥有一套既定的采编工作流程，新的平台意味着加入新的工作节点，新的处理方案。若非那些背景雄厚的大型官媒，普通的地方性媒体公司完全没有竞争力，只能继续依托于原来的工作流，然后日渐式微……但这也并非不可逆的，如果有一套新的框架或系统，在原有工作流不变的情况下就能投入使用，且上手简单快捷，

仅需花费很少的时间精力就能跟上时代的步伐，借助第三方平台踏入互联网的新天地，那么或许可以摆脱这种被人们渐渐抛弃的宿命。所以我以此为契机，开发了这套基于 Spring Boot 的新媒体企业信息化系统，旨在帮助中小型媒体企业或事业单位完成面向互联网的转型，让“资深元老”们也可以无缝和年轻的媒体环境接轨，继续在新的时代创造出更多的价值。

在互联网产品的开发中，个人认为最富魅力的便是它将传统重复劳动化繁为简的实现，而传统产业的转型也恰好需要这一特点发挥作用。最初的调研过程中，我们发现有几项技术由于较为大众，使用面较广，已被相继开发，它们分别是自媒体运营工具、传统媒体单位的采编 workflow、普通公司的信息化管理系统（如图 6-1 所示）。而我们所处的行业则恰好需要三位一体，集它们之大成。而上述的技术大多都基于 JavaScript、Spring Boot、MySQL 等语言或框架，于是最终选用了相近的开发手法，并最终如愿以偿地实现了这些功能，取缔了传统的重复劳动，甚至减少了互联网平台本身的重复操作。

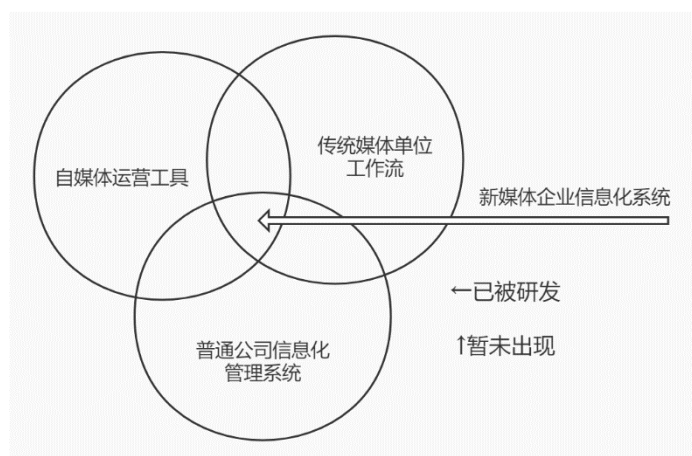


图 6-1 本项目最初的功能设计图

实际上，前后端的框架与开发方法已经是老生常谈了，不计其数的学生与开发者都在这些框架上实现了自己的业务。但能否做到真正跳脱出技术本身，和实际需求息息相关呢？这让我想起了刚开始实习时一位同事和我说过的话：“我们每天不辞辛劳地外出采集素材，回来夜以继日地编辑视频和文案，最终为的不仅仅是让自己满意，让自己感动，更要让客户满意和赞许。永远记得你做出来的东西是给谁看的，给谁用的。”经过实习后的我真真切切地感受到了这一点，绝对不要为了开发而开发，一定要考虑到使用者的感受。这也促使了内容自动分发业务的诞生，它们使得我开发出的系统更加合乎实际，且目前已交付给公司，投入了正式生产。

参考文献:

- [1] 建鼎, 博申. 多媒体内容管理系统[P]. 新加坡: Pccw Vuclip (singapore) Pte Ltd, 2017.
- [2] 汉克·亨德米尔. 媒体项目流程化管理的系统和方法[D]. 美国: Tribune Broadcasting Co LLC, 2016.
- [3] 刘博文. 深入浅出 Vue.js[M]. 北京: 人民邮电出版社, 2019: 34-41.
- [4] Mercyblitz. Spring Boot 编程思想核心[M]. 第二版. 北京: 电子工业出版社, 2019: 69-75.
- [5] 黄永祥. Python 自动化开发实战[M]. 北京: 清华大学出版社 2019: 114-120.
- [6] 肖顺, 严碧波. 一种基于 RabbitMQ 的消息推送系统的设计与实现[J]. 电子世界. 2019(07)
- [7] 潘佳辰. vue-element-admin 后台前端解决方案[R]. <https://panjiachen.gitee.io/vue-element-admin-site/zh/guide>, 2017.
- [8] 张帆. 微信小程序项目开发实战: 用 WePY、mpvue、Taro 打造高效的小程序[M], 2019.
- [9] 唐毅. H5 技术在移动客户端中的应用研究[D]. 湖南: 永州职业技术学院, 2017.
- [10] 王秋. H5 类融媒体产品的创新路径探析[D]. 河南: 洛阳理工学院, 2017.
- [11] 何腾蛟. 基于 CDN 的视频流媒体内容分发策略的研究[D]. 深圳: 深圳大学, 2017.
- [12] 李宇, 刘彬. 前后端分离框架在软件设计中的应用[J]. 无线互联科技. 2018(17)
- [13] 鱼朝伟, 詹舒波. 基于 RabbitMQ 的异步全双工消息总线的实现[J]. 软件, 2016(2): 139-146.


Design and implementation of new media enterprise information system based on Spring Boot

Abstract: This paper probes into the improvement of the pain spots and deficiencies faced by traditional media enterprises and discusses the significance of establishing a new media information engineering system that is three-dimensional, comprehensive and convenient. Meanwhile, it summarizes the key initiative of “Build omni-media and promote the fused and in-depth development of media” by President Xi, alongside a comparison of the differences between new and old media. By analyzing industry demands via practices, it offers a preliminary discussion of the proper way of serving local media enterprises with IT technologies in a general environment and how to complete fast transitions and transformations for new media enterprises that are privately owned. Moreover, it elaborates on the completed informatization system of new media enterprises and discusses possible industry reforms in the future.


Keywords: New media; Information engineering; Distributed development; Spring Boot; Vue.js; Python automation; Message queue; Media distribution; Applet

附录 A 系统数据库全表一览


media_video

 id: varchar(127)
 video_title: varchar(255)
 video_profile: text
 video_url: varchar(255)
 video_cid: varchar(255)
 video_pic: varchar(255)
 video_status: varchar(255)
 video_class: varchar(255)
 video_tag: varchar(255)
 video_reviewer: varchar(255)
 video_path: varchar(255)
 video_author: varchar(255)
 create_at: datetime(0)
 create_by: varchar(255)
 is_del: bit(1)
 update_at: datetime(0)
 update_by: varchar(255)


client_video

 id: varchar(127)
 video_title: varchar(255)
 video_profile: text
 video_url: varchar(255)
 video_cid: varchar(255)
 video_pic: varchar(255)
 video_date: datetime(6)
 video_status: varchar(255)
 video_author: varchar(255)
 video_reviewer: varchar(255)
 create_at: datetime(0)
 create_by: varchar(255)
 is_del: bit(1)
 update_at: datetime(0)
 update_by: varchar(255)


client_article

 id: varchar(127)
 article: longtext
 title: varchar(255)
 author: varchar(255)
 reviewer: varchar(255)
 readings: int(11)
 status: varchar(255)
 pic: varchar(255)
 create_at: datetime(0)
 create_by: varchar(255)
 is_del: bit(1)
 update_at: datetime(0)
 update_by: varchar(255)


client_swiper

 id: varchar(127)
 swiper_pic: varchar(255)
 swiper_name: varchar(255)
 swiper_status: varchar(255)
 swiper_reviewer: varchar(255)
 create_at: datetime(0)
 create_by: varchar(255)
 is_del: bit(1)
 update_at: datetime(0)
 update_by: varchar(255)


vote_groups

 id: varchar(127)
 activity_id: varchar(255)
 group_name: varchar(255)
 create_by: varchar(255)
 update_by: varchar(255)
 create_at: datetime(0)
 update_at: datetime(0)
 is_del: bit(1)
 group_profile: text
 group_votes: int(64)
 group_pic: varchar(255)
 group_pic_hd: varchar(255)


east_staff_list

 id: varchar(127)
 east_name: varchar(255)
 east_live: varchar(255)
 east_phone: varchar(255)
 east_card: varchar(255)
 east_vehicle: varchar(255)
 create_at: datetime(0)
 create_by: varchar(255)
 is_del: bit(1)
 update_at: datetime(0)
 update_by: varchar(255)


vote_records

 id: varchar(127)
 group_id: varchar(255)
 record_nickname: varchar(255)
 record_openid: varchar(255)
 record_date: date
 create_at: datetime(0)
 create_by: varchar(255)
 is_del: bit(1)
 update_at: datetime(0)
 update_by: varchar(255)
 activity_id: varchar(255)


com_staff

 id: varchar(127)
 name: varchar(255)
 nickname: varchar(255)
 password: varchar(255)
 level: varchar(255)
 create_by: varchar(255)
 update_by: varchar(255)
 create_at: datetime(6)
 update_at: datetime(6)
 is_del: bit(1)


vote_swipers

 id: varchar(127)
 activity_id: varchar(255)
 swiper_pic: varchar(255)
 swiper_name: varchar(255)
 create_at: datetime(0)
 create_by: varchar(255)
 is_del: bit(1)
 update_at: datetime(0)
 update_by: varchar(255)


east_staff_health

 id: varchar(127)
 east_id: varchar(255)
 staff_temp: varchar(255)
 staff_status: varchar(255)
 staff_place: varchar(255)
 create_at: datetime(0)
 create_by: varchar(255)
 is_del: bit(1)
 update_at: datetime(0)
 update_by: varchar(255)


vote_activities

 id: varchar(127)
 activity_name: varchar(255)
 activity_profile: text
 activity_active: bit(1)
 activity_live: text
 create_at: datetime(0)
 create_by: varchar(255)
 is_del: bit(1)
 update_at: datetime(0)
 update_by: varchar(255)

vote_merchants

 id: varchar(127)
 activity_id: varchar(255)
 merchant_name: varchar(255)
 merchant_info: varchar(255)
 merchant_logo: varchar(255)
 create_at: datetime(0)
 create_by: varchar(255)
 is_del: bit(1)
 update_at: datetime(0)
 update_by: varchar(255)

vote_notes

 id: varchar(127)
 activity_id: varchar(255)
 note_name: varchar(255)
 note_display_time: datetime(6)
 note_details: text
 create_at: datetime(0)
 create_by: varchar(255)
 is_del: bit(1)
 update_at: datetime(0)
 update_by: varchar(255)

致 谢

首先感谢肇庆学院四年来对我的辛勤培育！感谢教师们孜孜不倦的教导和同学们的友爱互助，同时也要为我实习单位里关照过我的同事们说一声谢谢。

本论文是在我的指导老师，和校外企业工程师的亲切关怀和悉心指导下完成的。从课题的选择到项目的最终完成，宋强老师和梁展鹏老师都始终给予了我细心的指导和不懈的支持，在此谨向宋强老师和梁展鹏老师致以诚挚的谢意和崇高的敬意！