四川大學

本科生毕业论文（设计）

题　　目 基于 Node.js 的视频平台的设计和实现

学　　院 计算机学院

专　　业 计算机科学与技术

学生姓名 白　　琦

学　　号 2013141231103 年级 2014

指导教师 陈　　虎

二〇一八年五月十五日

基于 Node.js 的视频共享平台的设计和实现

计算机科学与技术

学生 白 琦 指导老师 陈 虎

[摘要]互联网的世界里，信息共享已经是网上交流的主要形式。而共享方式在互联网上有很多形态，如文字，图片，视频等，信息共享平台的技术也随着流行技术在不断的变化，从刚开始的 jsp，到后来 asp 的走红，然后 PHP 风靡一时并产出了 Discuz 这个优秀的框架，近些年来又有 Python，Ruby-on-Rails，Go 等都分别有自己众多的平台引擎实现。近些年来，Node.js 作为一个开源、跨平台、的语言，其可用在服务器端和 web 应用开发的性质使其变得越来越受欢迎，它使我们通过 JavaScript 和社区提供的一些模块来制作网络相关的应用和服务器端后台接口，目前国际上，官方社区的开源模块已经拥有了二十多万个。Node.js 的性质是事件驱动的特性和非阻塞 I/O 输入输出，适合高并发的 web 应用使用，所以论文的信息交流和视频共享平台使用 Node.js 这门新兴的技术来实现。

论文主要分析了用户对于视频信息交流平台的需求,描述了基于 Node.js 的视频分享网站的系统架构和网站数据库设计，并详细描述了登录、视频上传、评论、个人信息等模块的详细设计和实现算法。

系统采用 B/S 的开发模式，使用 Node.js 作为后端的脚本语言，采用 Express 作为 web 框架，数据库方面采用了关系型数据库 MySQL 和非关系型数据库 Redis。系统共包含有 2 个部分，后端 API 中心、前端网页页面。系统的开发采用 WebStorm 编辑器，客户端使用 Chrome 浏览器作为运行环境。通过该系统的实现，培养系统分析、设计和实现的能力。

[主题词] MySQL; Redis; Express; 事件驱动; 非阻塞 I/O

Information exchange and video sharing platform of Sichuan University Based on Node.js

Computer Science and Technology

Student： Bai Qi Adviser: Chen Hu

[Abstract]In the internet era, the information sharing is the main form of online communication. There are so many shapes in the internet for information sharing, such as text, picture, video and so on. The technology of information sharing platform is also changing with the popular technology. From the beginning of JSP to a trendy ASP, and then from PHP to Discuz gradually. Recent Ruby on Rails, Python. Go all have their own platform implementation. Node.Js become very popular as an open-source, cross-platform runtime environment for developing server-side Web applications. It enables us to make network - related applications and server - side backstage interfaces through JavaScript and some of the modules provided by the community. At present, the open source module of the official community has about two hundred thousand.. Node.js has an event driven architecture capable of asynchronous asynchronous I/O. It‘s very suitable for high concurrency web application. So my project use Node.Js to implement.

The paper analyzes that the needs of video information exchange platform from the people, describes the system architecture of community site based Node.js and database design, and a detailed description of the login, video upload, comments, profile, social platform detailed design and algorithm sharing module.

The system uses B/S pattern of development, uses Node.js as the script language of the back end, uses express as web frameworks, and uses the relational database MySQL and the non relational database Redis.. The system contains two parts, the backend API Center and front-end web pages. The development of the system adopts WebStorm editor, and the client uses Chrome browser as the running environment. Through the realization of the system, the ability of system analysis, design and implementation can be cultivated.

[Key Words] MySQL; Redis; Express; Event-driven; Asynchronous I/O

目录

1 绪论

1.1 概述

基于 Node.js 的视频共享平台，即提供一个为所有发布个人视频和讨论的平台，目的是让用户能够分享身边的人和事，真正表达出自己想表达的想法和观点。视频发布的内容可以是多种多样的，比如自己发现的新事物，亦或是身边人的行为，也可以发布一些独有的信息。而采用 Node.js 来开发这个视频共享平台，主要是为了填补用 Node.js 实现视频共享平台的空白，之前的大多数视频共享平台都是采用 PHP 或者 python 实现的。

系统采用 Node.js 开发，使用经典的 Node.js 的 MVC 模式 express 框架，数据库采用关系型数据库 MySQL 和非关系型数据库 Redis，通过 Node.js 原生接口连接数据库，前端使用 ajax 向服务器发异步请求，无刷新地操作 DOM 来更新页面。

1.2 选题意义

近年来我国互联网迅速发展，大量增长的网络用户规模、互联网普及率及带宽催生出新的视听路径与需求。视频分享在 Web2.0 的各种新型应用中，成为了热点。网络视频已经成为人们之间传递重要信息的一种方式。视频共享平台可以为用户提供视频上传、播放和分享服务，传播快捷，操作简单，让互联网用户在线发布、浏览和分享视频。

随着时间的推移，JavaScript 的社区力量正在变得强大，据 2015 年 GitHub（国外著名 git 源码托管网站）公布的十大流行编程语言[1]，JavaScript 已经高居榜首，势必会看到将会有更多的应用采用 Node.js 来开发。另一方面 Node.js 适合处理实时的、大规模和高并发的应用，非常适合信息共享平台使用，所以系统意图探索信息共享应用在 Node.js 中的实现。

1.3 JavaScript 在国内外的研发现状

JavaScript 语言是由 Netscape 公司的布兰登·艾奇（Brendan Eich）在 1995 年随着 Netscape Navigator 2 的发布而发布的，它最初是为 Web 交互而设计的脚本语言，距今已有 21 年的历史了，但是随着时间的发展，JavaScript 语言的语法变得越来越复杂，而且从前端领域走向了服务端领域，2015 年 6 月，ECMAScript 6 正式被采纳为国际标准。新增了多项规范，使 JavaScript 成为更易用和更加现代化的语言。

目前流行的几个 JavaScript 库有：jQuery，React，Argular，Ext 等，每个库都有自己非常活跃的社区和一大批在其框架下进行二次开发的开发者[2]。

当浏览器正在渲染网页时，它被分为渲染引擎和 JavaScript 引擎，所以 JavaScript

引擎的性能对于浏览器渲染网页非常重要。在过去的几年中，浏览器厂商提供的 JavaScript 引擎的性能已经超过了几十倍，这样开发者就可以在前端实现更多的功能。目前主流的 JavaScript 引擎有以下几种：

1. Rhino 引擎，是由 Mozilla 基金会维护，开源的引擎，用 Java 编写
2. SpiderMonkey 引擎，第一个 JavaScript 引擎，Mozilla Firefox 浏览器使用
3. V8 引擎，开源的引擎，由 Google 开发维护，Google Chrome 浏览器使用
4. JavaScriptCore 引擎，开源的引擎，Safari 浏览器使用
5. Chakra 引擎，Microsoft Edge 浏览器使用

而在服务器编程领域，对于大规模、实时性、高并发的需求越来越大，JavaScript 原生的异步机制刚好十分适合 Node.js 的事件驱动型和异步 I/O 操作的特性，所以 2009 年 Node.js 在 Ryan Dahl 的创建下诞生了,Node.js 的包管理工具 npm 目前已拥有超过 250000 的模块。

1.4 系统的开发方法

系统的服务端主要采用 Node.js 开发，使用到的相关工具也都是 Node.js 周边的工具。

在 Node.js 的基础上，还使用了 Express 作为框架来构建应用。在 Express 中，设置路由，访问 MySQL 数据库，然后输出 JSON 格式的响应发送给客户端。

系统的前端页面均使用 HTML5 编写，在 CSS 方面，系统也采用最新的 CSS3 的标准构建，以及包含大量的 JavaScript 来操作 Dom 文档[3]。

编辑器方面采用了 Jetbrains 出品的 WebStorm 编辑器，开发的时候使用了大量方便的插件。

开发过程中，使用 Chrome 浏览器作为测试浏览器。Chrome 中内嵌的开发者工具对系统的调试、测试起了非常大的帮助。

1.5 论文研究的主要内容

网站主要实现用户视频作品的上传和在线播放，及评论、回复功能。网站主要功能包括用户注册、登陆，视频上传、发布、分享，视频的播放，视频下的评论、回复，及用户组管理等。

搭建 Web 服务器是系统的第一个难点，选择一个合适的环境和后端语言对于一个良好的网站来说必不可少。其次，随着使用人数的增多，网站和数据库也将面临阻塞的问题，所以要选择一个能很好处理高并发的解决方案和在网络压力过大时候进行代码层面的改善。

且视频分享是随时随地的；所以，移动端的适配必不可少，网站的前端需要很好地适

配移动端，尤其是手机端的分辨率，才能给用户带来最好的体验。

在以前的传统网站开发中，前端简单地将原型拼接成 HTML 静态页面，而其中的页面交互逻辑大多是后端的，如动态数据渲染，或许前后端紧紧耦合，如混编形式。这些网站是由架构决定的，前端只能依靠后端。这样的开发模式，不仅仅大量降低了开发效率，而且代码维护起来也非常吃力。所以，我决定在项目中采用前后端分离的架构设计和开发模式，更多的交互逻辑传递到前端，而后端的精力则集中在接口上，权限控制，和逻辑分析和实现，做高效的代码。前后端分离的模式也是项目中的最大难点和突破。

系统旨在采用 B/S 模式，系统中，包含有四个子模块：

- 登录模块：主要包括用户注册登录和登录状态维护
- 个人信息模块：主要包括个人信息的修改和头像的上传
- 视频列表模块：主要包括各标签下的视频信息传输
- 视频上传模块：主要包括视频信息的上传

1.6 论文组织与结构

第一部分：绪论。主要介绍了视频信息平台的国内外发展现状，JavaScript 的国内外发展现状，以及系统的意义及主要研究工作；

第二部分：相关概念及技术简介。对与论文相关的一些核心概念作一个大概的介绍，便于论文后面的描述；

第三部分：系统分析。包括可行性分析、需求分析；

第四部分：系统概要设计。包括系统功能结构划分和数据库架构设计；

第五部分：详细设计及实现。包括登录模块、网站首页、个人信息模块、视频模块、上传模块的实现与展示；

第六部分：系统测试。包括设计测试用例及测试结果；

第七部分：总结与展望。

2 相关概念及技术简介

2.1 网络体系架构

Node.js 视频信息共享平台使用三层 B/S 架构模式， 在 B/S 架构下，用户界面由浏览器实现。主要的逻辑是在服务器端实现的。通过服务器端直接访问和操作数据库。这种体系结构模式是随着 Web 应用的普及而改进 C/S 产生的体系结构。这将大大减轻客户端的负担，降低系统维护和升级的难度，并降低用户的使用门槛[4]。

B/S 模式的主要优势和劣势如下：

（1）维护、管理、更新方便。

目前，应用的更新越来越频繁，B/S 架构的程序在这方面很明显有非常大的优点。对一个具体的应用来说，修改任何地方都只需要在 Server 端进行更改、部署，所有的客户端即可同步看到更新后的应用，客户端只需要是一个浏览器，所以任何设备上的浏览器皆可成为客户端，服务提供者无需在客户端方面做任何的维护，所有的操作都只需在服务器上进行即可。

（2）降低成本、用户使用门槛低

如果是采用 C/S 架构的话，为了服务各个平台的用户，我们需要针对各平台单独开发应用，这样不仅极大的提高了开发的成本，而且使得项目变得十分复杂。而选择 B/S 架构的话，我们只需要考虑服务器的平台选择而不用担心客户端的平台选择，任何设备只要有浏览器就可以成为一个客户端。对于服务端而言，我们一般会选择开源的 linux 系统，这是一个流行而且免费的系统，非常适合作为服务端的操作系统。

比如说很多人经常要上淘宝购买东西，但是作为用户他根本不需要知道淘宝所使用的服务器用了什么平台，他只需要有一个浏览器就可以了。

（3）劣势：服务端运行压力较大。

由于 B/S 架构的应用只运行在服务器端（Server）上，所有的数据请求响应均需要经过服务器，把负担集中在了服务器这一端，这使得应用服务器运行更加紧张。一旦服务器宕机，后果将是不可想象的。因此，许多公司都使用了主从服务器，异地双活等技术，防止出现问题。

2.2 Node.js

Node.js 的底层核心模块是 C++语言编写的。它也是一个 JavaScript 的运行环境，就像浏览器一样。JavaScript 领域一直是 Web 前端工程师的世界，直到 Node.js 出现后才改变了这种情况。Node.js 是一个后端 JavaScript 的运行平台，开发者可以通过编写 JavaScript 代码就可以让程序在系统里跑起来。Node.JS 使用的是 Chrome 浏览器的 V8 引擎，这是当前

JavaScript 引擎中最好的性能之一，NoDE.js 提供了许多系统级接口，例如文件操作。

Node.js 中绝大多数操作都是异步调用，比如文件读取、网络请求等，这样以来我们就可以很方便的进行各种并行 I/O 操作，每个调用之间无需等待之前 I/O 操作结束，可以极大的提高效率。

到底 Node.js 是事件驱动的，它给出的大多数接口都是异步和事务的风格，这样我们可以充分的利用系统，非阻塞的完成操作，这样的设计非常适合于高并发、I/O 操作密集型的后端应用。我们都知道，同步请求一直是服务器开发中的一个大问题，被阻塞的 API（如 PHP）会导致系统资源的极大浪费和时间延迟[5]。而通过事件驱动，我们可以极大的改善这个问题。

Node.js 提供了一个叫 npm 的包管理器，在这个包管理器里面，你可以重用超过 250000 个全世界各地的开发者所发布的模块，你可以使用包括 express、koa 等优秀的 web 应用框架快速的生成一个网站[6]。

2.3 Express

Express 是由 TJ Holowaychuk 开发的一个简单而高效的 Web 应用框架，提供了一整套 Web 相关的模块帮助你搭建一个 Web 应用。它提供了非常丰富的 HTTP 工具和继承自 Connect 框架的中间件，使得我们可以很容易的就可以创建一个强大、友好的网站。

Express 框架的重要特点是：

- 可以设置中间件来轮流处理和响应 HTTP 请求
- 方便的自定义路由来分流处理 HTTP 请求
- 可以通过使用各种模板引擎来动态渲染 HTML 页面

2.4 数据库技术

数据库技术主要研究如何存储、使用和管理数据。数据库技术的对象是数据，因此数据库技术主要包括：在对数据进行抽象和管理之后，根据一定的结构建立相应的数据库，并使用数据库管理系统来设计各种功能，如对数据库中的数据进行删除、修改、检查、处理和分析的数据管理系统[7]。

数据库技术是信息系统中最重要的技术。它是一种基于计算机的数据处理技术。它主要包括如何组织、存储和共享数据，以及如何高效地读取和处理数据。数据库的存储库是数据库存储库，它是存储在计算机中的大量数据的集合。

2.5 MySQL

MySQL 以其良好的性能、低成本和高可用性而成为最流行的开源数据库。它可以适应高要求，例如 Web 应用程序。随着 MySQL 的不断升级，其在谷歌、新浪微博、脸谱网等大型 Web 应用中得到了越来越广泛的应用。特性主要有如下：

- 源代码经过了各种编译器的编译，提高了源码的可移植性
- 跨平台，支持多种操作系统
- 绝大多数编程语言都有 MySQL 的 API
- 支持多线程，支持多用户，多核 CPU
- 极大的优化了 SQL 语句查询算法，提升了数据库的性能。
- 可单独使用，也可嵌入程序使用。
- 多语言支持，可使用各种编码的语言
- 提供多种数据库连接方法。
- 有大量的周边工具可供使用。
- 可以处理大量的数据。

2.6 Redis

ReDIS 是一个关键的价值存储系统。类似于 MeMcChash，它支持更多的存储值类型，包括字符串（String）、列表（链表）、SET（SET）、Zset（排序集-有序集）和哈希（哈希类型）。这些数据类型支持 PUP/POP、添加/移除和交叉、联合和差异集，以及更丰富的操作。所有这些操作都是原子性的。在此基础上，ReIDIS 以多种方式支持排序。与 MMECALH 一样，数据被缓存在内存中以确保效率。不同之处在于 ReDIS 周期性地将更新后的数据写入磁盘，或者将修改后的操作写入到添加的记录文件中，并在此基础上实现主从（主从）同步[8]。

ReDIS 是一个高性能的关键价值数据库。ReIIS 的出现极大地弥补了 MIMCHACK/Valy 存储的不足。在某些情况下，它可以补充关系数据库。它提供了 java，C/C++，C，PHP，JavaScript，Perl，Python，Ruby，Object-C，二郎神和其他客户。使用起来非常方便。

ReIDIS 支持主从同步。数据可以从主服务器同步到任意数量的从服务器，从服务器到主服务器和其他服务器。这允许 ReDIS 执行单树复制。保存可以有意或无意地写入数据。由于发布/订阅机制的全面实现，当数据库从数据库中随时随地同步时，可以订阅数据库，并且可以接收主服务器的完整的消息发布记录。同步有助于读取操作的可伸缩性和冗余性。

3 系统分析

3.1 系统功能分析

视频共享平台是一个可以在线上传和分享视频的网站，系统分为 3 个模块，分别为：登录模块、用户模块、视频模块和评论模块。系统提供用户的注册、登录、注销功能；用户信息的查看和修改功能；视频的上传、删除功能；和评论的发布、删除功能；下面进行细化分析。

3.2 可行性分析

No.js 使用事件驱动的异步编程，设计用于网络服务，其 IO 处理的非阻塞模式为系统资源消耗相对较低带来了 No.js 的高性能和出色的负载能力[9]。

Node.js 可以很方便地实现如搭建 Sever，配置、设置路由、中间件，参数与处理，错误处理，渲染，Session 等功能。且 Node.js 有强大的 node package manager(NPM)，能解决 Node.JS 代码部署上的很多问题，也能快速便捷地使用第三方包，如连接 MySQL 的第三方包：sequelize 等等[10]。

因此，网站的开发是完全可以实现的。

3.3 外部角色

网站有游客、普通用户、视频发布者、评论发布者四个角色，他们的权限分别如下：

- 游客：游客可以在首页浏览所有的视频。
- 普通用户：普通用户拥有游客的所有权限，以及可以修改自己个人信息的权限。
- 评论作者：评论作者拥有普通用户的所有权限，以及可以删除自己发布过的评论权限。
- 视频发布者：视频发布者拥有普通用户所有权限，以及可删除自己发布过的视频权限。

3.4 需求分析

在可行性分析之后，此节将进一步明确设计所需要实现的各个子系统的具体功能。

3.4.1 系统的顶层用例

系统的顶层用例图如图 3.1 所示：

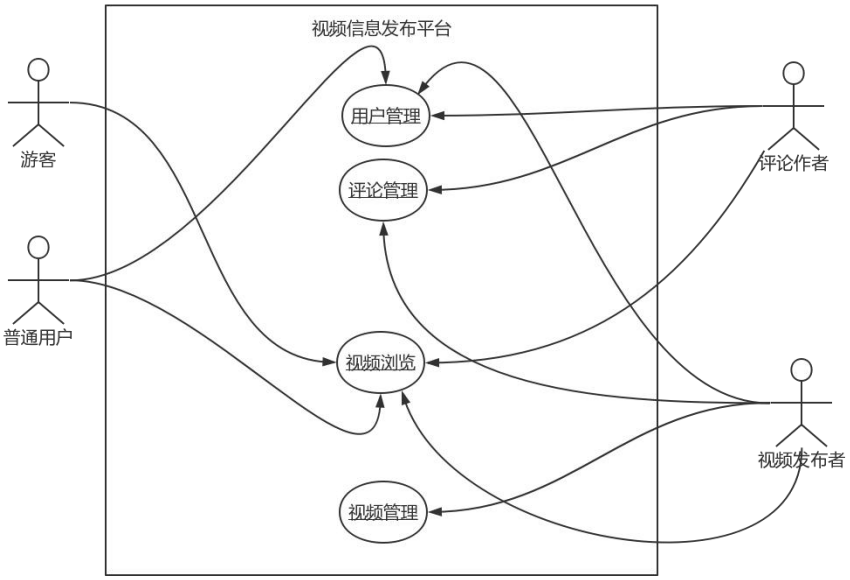

图 3.1 系统的顶层用例图

3.4.2 用户管理用例

系统提供用户登录，用户注销以及用户个人信息查询功能。

用户管理的用例图如图 3.2 所示：

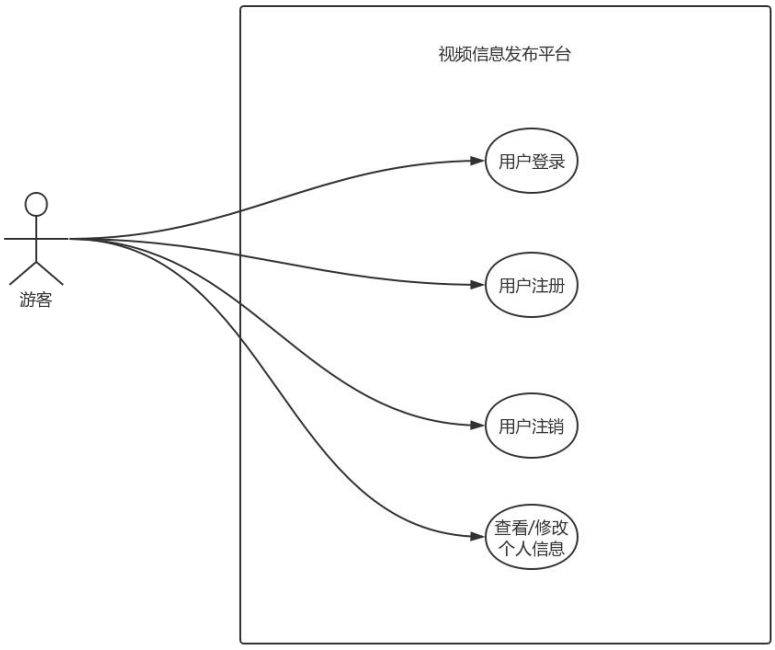

图 3.2 用户管理用例图

用户登录：系统提供帐号注册登录的形式来登录信息发布平台，进而进行视频发布、评论等操作。

- 前置条件：用户进行登录操作

- 输入：用户已注册的账号信息
- 输出：用户的相关信息以及用户的登录状态

用户注销：系统允许已经登录的用户退出登录状态，退出登录状态后将不能进行视频发布、评论等行为。

- 前置条件：用户已处于登录状态
- 输入：无
- 输出：退出成功的提示

事件流如下：

1) 用户点击注销按钮
2) 注销成功

查看个人信息：系统允许已登录用户查看自己的个人相关信息，如昵称、头像、个人信息等。

- 前置条件：用户已处于登录状态
- 输入：无
- 输出：个人的相关信息

事件流如下：

1) 用户进入个人主页
2) 用户可看到个人相关信息，包括昵称、头像、电话、电子邮件等

3.4.3 视频管理用例

系统提供查看视频列表，查看视频详情，发布视频，删除视频功能。

用例图如图 3.4 所示：

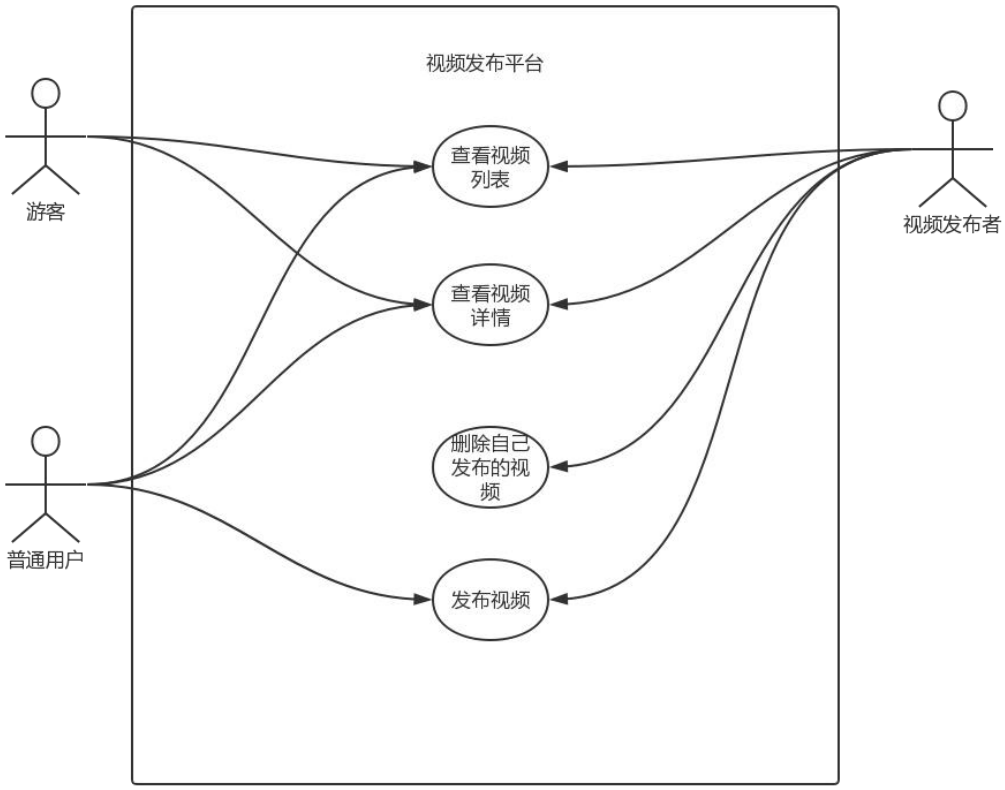

图 3.4 视频信息管理用例图

查看帖子列表：系统允许用户浏览网站的全部视频。

- 前置条件：无
- 输入：无
- 输出：帖子列表

事件流如下：

1) 用户进入首页
2) 系统按发帖时间倒序显示帖子列表

查看帖子详情：系统允许用户浏览网站的某个视频的详情。

- 前置条件：无
- 输入：无
- 输出：帖子详情

事件流如下：

1) 用户进入首页
2) 用户点击某个帖子
3) 系统显示帖子详情
4) 系统按评论时间正序显示帖子评论列表

发布帖子：系统允许已登录的用户在视频平台发布视频。

- 前置条件：用户已登录
- 输入：视频信息，视频文件
- 输出：发布成功的提示

事件流如下：

1) 用户进入发布页面
2) 用户输入视频信息
3) 用户上传视频
4) 用户提交内容
5) 系统提示发布成功后转入首页

发布帖子的活动图如图 3.5 所示：

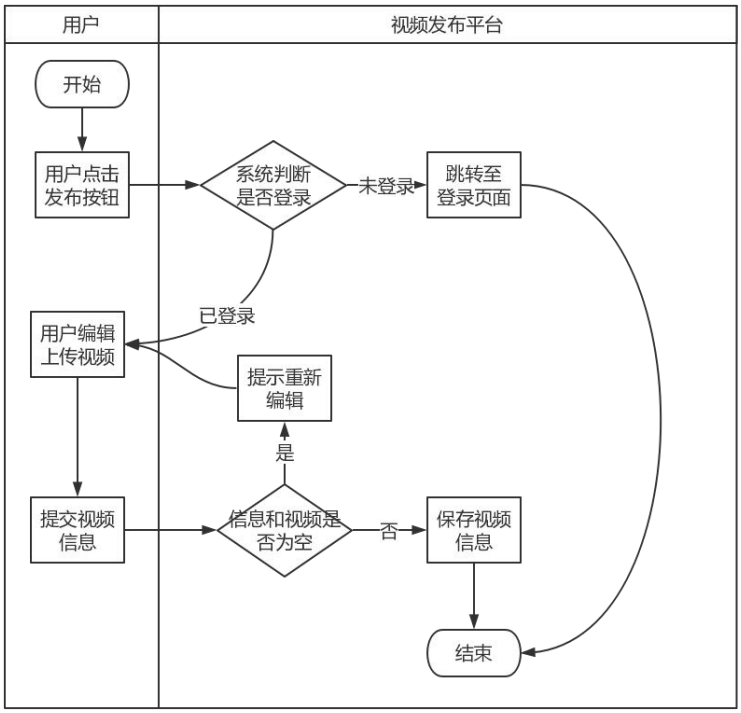

图 3.5 发布视频流程图

删除自己的帖子：系统允许已登录用户删除自己所发布的某个帖子

- 前置条件：用户已登录
- 输入：无
- 输出：删除成功的提示

事件流图如图 3.6 所示：

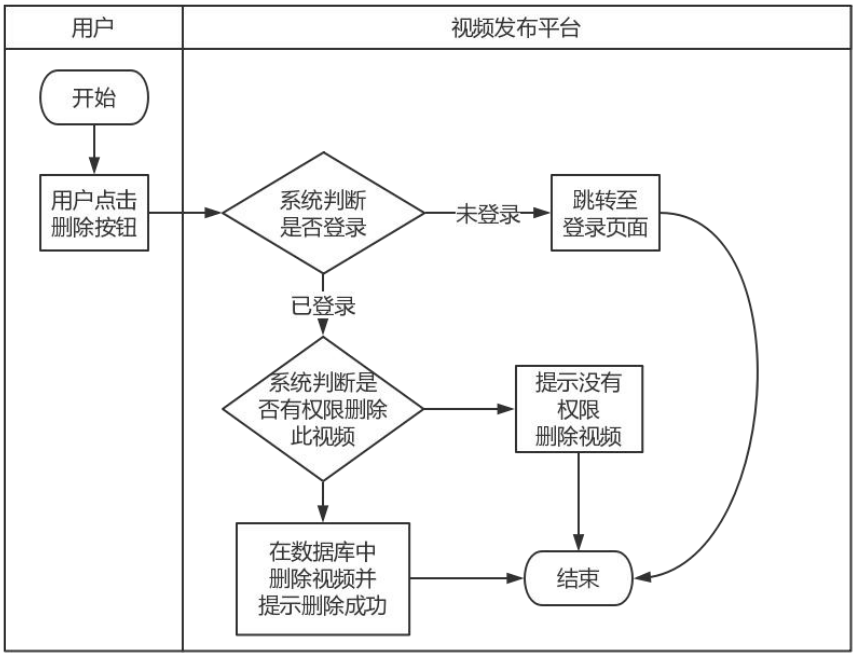

图 3.6 删除自己的视频活动图

3.4.4 评论管理用例

系统提供查看评论列表、发布评论、功能。

用例图如图 3.8 所示：

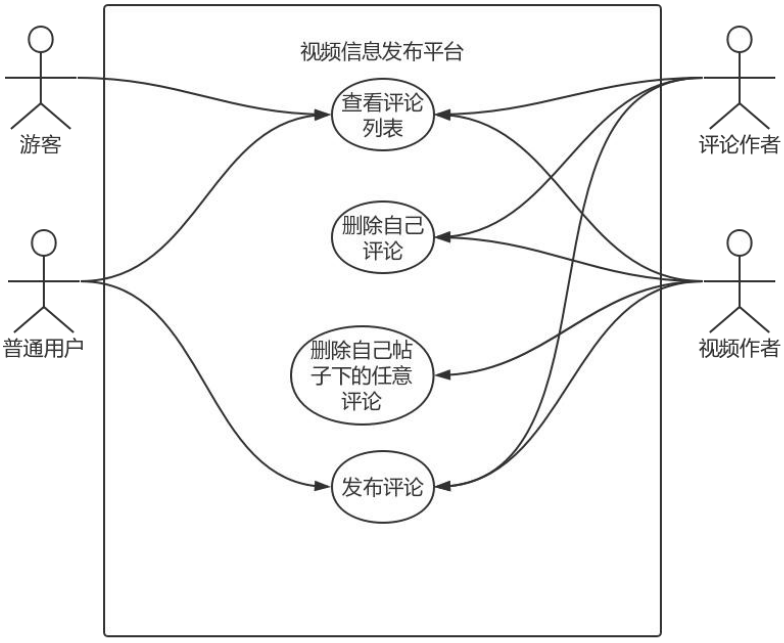

图 3.8 评论用例图

查看评论列表：系统允许用户浏览某个帖子下全部评论内容。

- 前置条件：无
- 输入：无
- 输出：评论列表

事件流如下：

1) 用户进入某个视频的详情页面
2) 系统显示该帖子的所有评论

发布评论：系统允许已登录的用户在帖子下发布评论。

- 前置条件：用户已登录
- 输入：评论内容
- 输出：评论成功的提示

事件流如图 3.9 所示：

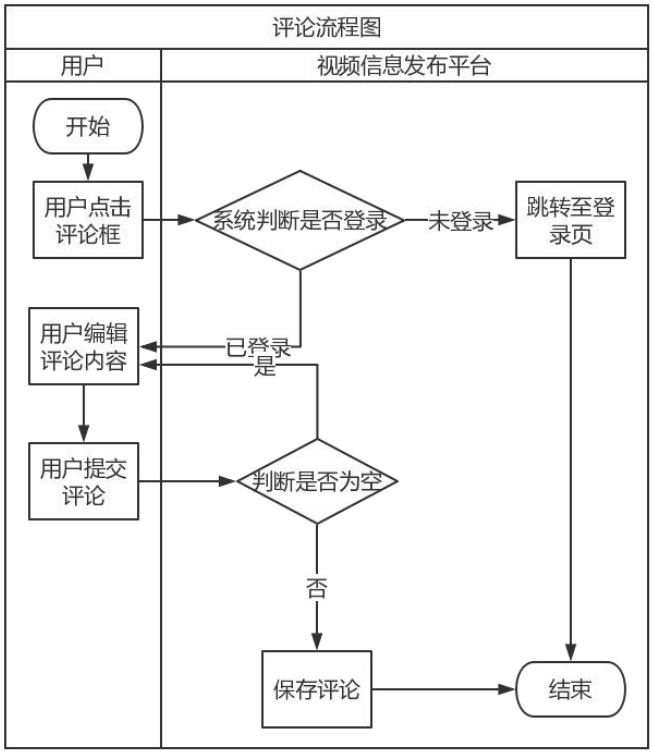

图 3.9 评论流程活动图

删除自己的评论：系统允许评论作者删除自己发布的评论。

- 前置条件：用户已登录
- 输入：无
- 输出：删除成功的提示

事件流图如图 3.10 所示：

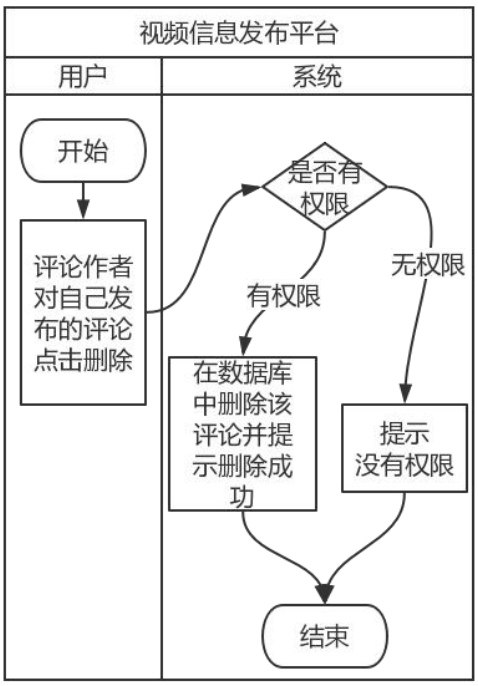

图 3.10 删除自己的评论活动图

删除自己帖子下的任意评论：系统允许贴主删除自己发布的帖子下的任意评论。

- 前置条件：用户已登录
- 输入：无
- 输出：删除成功的提示

事件流图如图 3.11 所示：

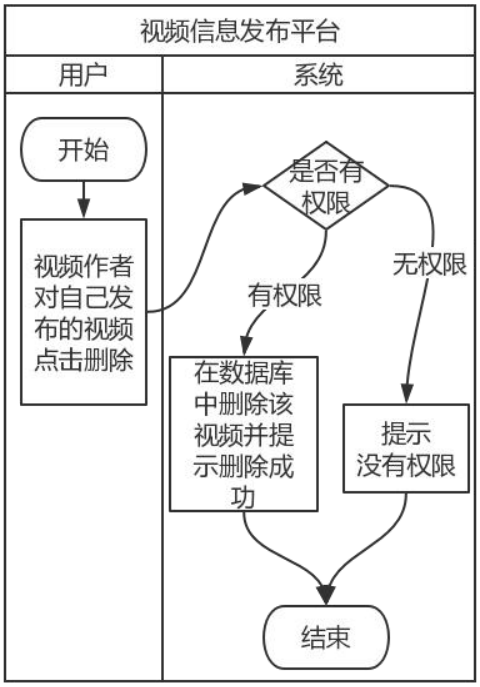

图 3.11 删除自己帖子下的任意评论

3.5 数据字典

数据字典就是对整个系统所需要用到的数据进行描述的一个列表。系统的数据字典如下：

数据元素定义如下表 3-1 所示：

表 3-1 数据元素定义表

名称	数据元素名	数据类型	取值范围	使用说明
用户编号	user_id	int	数字	用户的唯一标识
用户性别	gender	int	数字	用户性别
用户名	username	varchar	字符、数字	用户名
用户头像地址	avatar	varchar	字符、数字	用户头像地址
用户密码	password	varchar	字符、数字	用户登录密码
用户昵称	real_name	varchar	字符、数字	用户昵称
用户邮箱	email	varchar	字符、数字	用户邮箱
用户电话	phone	varchar	数字	用户电话
视频编号	video_id	int	数字	视频的唯一标识
视频标题	title	varchar	字符、数字	帖子的标题
视频地址	video	varchar	字符、数字	视频地址
评论编号	comment_id	int	数字	评论的唯一标识
评论内容	comment	varchar	字符、数字	评论的内容

数据结构定义如下表 3-2 所示：

表 3-2 数据结构定义表

数据结构	别名	数据组成	使用说明
用户信息	user	user_id+username+password	记录用户的登录信息
用户扩展信息	user_extend	user_id+gender+avatar+real_name+email+phone	记录用户的个人信息
视频信息	video	video_id+video+user_id+title+real_name	记录视频的详细信息
评论信息	comment	comment_id+comment+user_id+avatar+video_id	记录评论的详细信息

4 系统概要设计

4.1 系统功能结构

系统功能结构如下图 4.1 所示：

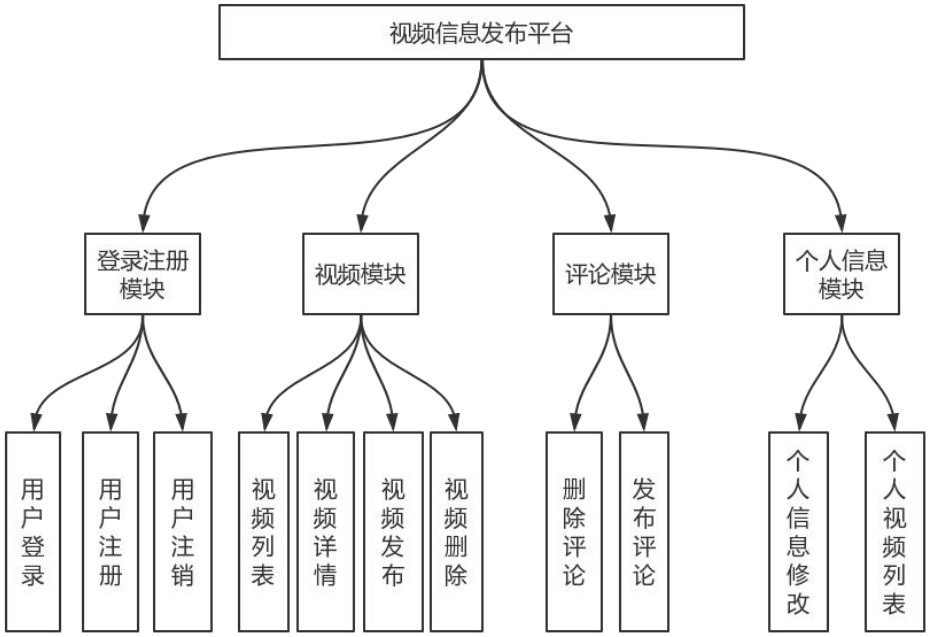

图 4.1 系统功能结构图

其中系统各个模块的主要功能如下：

登录注册模块：主要包括用户注册、登录，该模块使用了 MySQL+Redis 模式，保存用户登录状态，这样会大大的方便用户，用户不需要每次登录都输入密码，也大大减少了每次需要验证权限时的时间[11]。

视频模块：主要包括视频列表、视频详情、发布视频、删除视频功能，这是视频分享平台的主要功能之一。

评论模块：主要包括评论列表、发布评论、删除评论功能。

个人信息模块：主要包括个人视频列表、个人信息修改功能。

4.2 数据库选择

系统采用的数据库是MySQL，使用Node.Js原生MySQL模块来与MySQL沟通。MySQL是关系型数据库，非常适合做信息发布平台的基础数据库。

系统还采用了 Redis 数据库，使用 Node.Js 原生 Redis 模块来与 Redis 沟通。Redis 是非关系型数据库，非常适合做用户 token 的保存。

4.3 E-R 图

数据库是整个系统的基石，数据库设计的好坏直接影响到整个系统的设计，此节将专门进行 E-R 图的设计。

整个系统的数据主要包括帖子相关数据、评论相关数据、点赞相关数据、账户相关数据。可将这些数据抽象为以下 E-R 图，如图 4.2 所示：

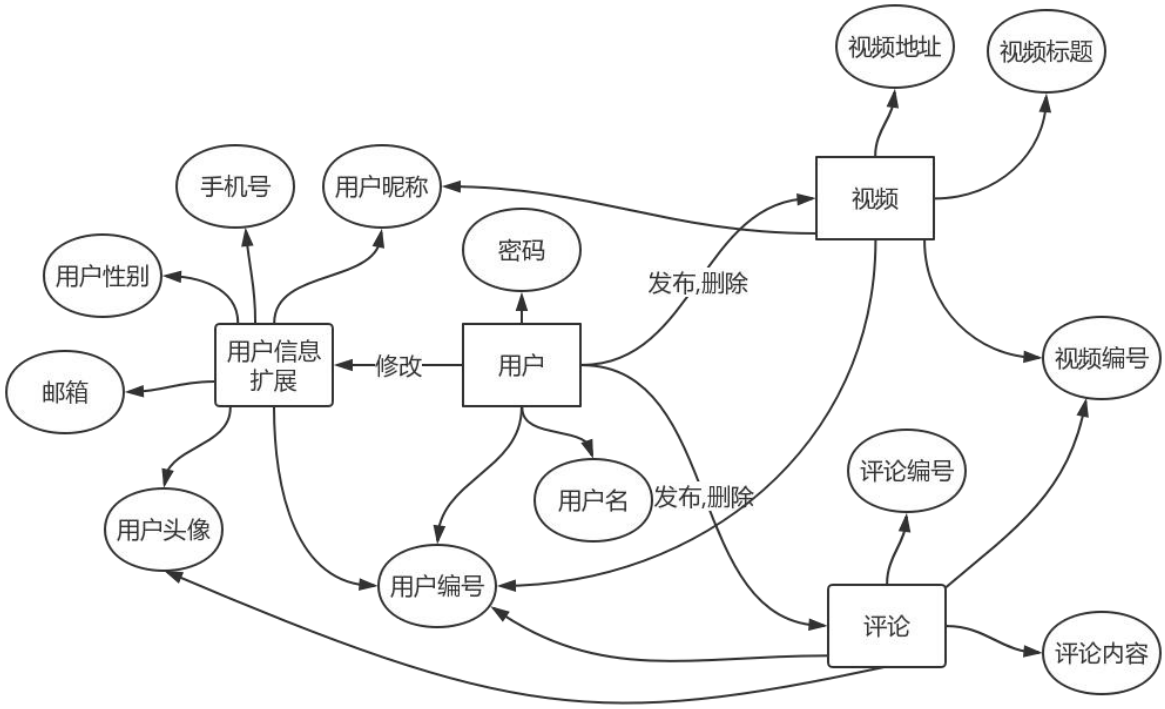

图 4. 2 数据库 E-R 图

4.4 数据逻辑结构设计

4.4.1 用户表

用户表主要用于存储用户登录信息，如表 4-1，共有 3 个字段，分别是：

用户 ID：系统中用户的唯一身份，主键，自增；

用户名：用户登录所需用户名，与用户 ID 一一对应；

密码：用户登录所需密码；

表 4-1 用户表

中文字段名	英文字段名	类型	是否索引	备注
用户 ID	user_id	int(11)	是	主键，自增
用户名	username	char(64)	是	
密码	password	char(16)	否	

4.4.2 用户信息扩展表

用户信息扩展表主要用于存储用户个人信息，共有 7 个字段，分别是：

用户信息扩展表 ID：无意义，主键，自增；

用户 ID：系统中用户的唯一身份；

用户昵称：用户在系统中所用昵称；

头像地址：用户的头像地址；

手机号：用户的手机号；

用户邮箱：用户的邮箱。

性别：用户的性别，为节省空间，用 int 类型保存，其中 0 是未知、1 是男性、2 是女性；

表的具体内容如下表 4-2 所示：

表 4-2 用户表

中文字段名	英文字段名	类型	是否索引	备注
用户信息扩展表 ID	id	int(11)	是	主键，自增
用户 ID	user_id	int(11)	是	
用户昵称	real_name	char(64)	是	
手机号	phone	char(16)	否	
性别	gender	int(1)	否	
头像地址	avatar	char(255)	否	
邮箱	email	char(255)	否	

4.4.3 视频表

该表主要用于存储用户发布的视频信息，另外为了让查询加快，特地设定了用户昵称、这个冗余字段，以空间换时间。

共有 5 个字段，分别如下：

1. ID：视频的唯一 ID，主键，自增；
2. 用户 ID：视频发布者 ID；
3. 昵称：视频发布者的昵称；
4. 标题：视频的标题；
5. 视频地址：视频的地址；

表的具体内容如下表 4-3 所示：

表 4-3 帖子表

中文字段名	英文字段名	类型	是否索引	备注
ID	id	int(11)	是	主键，自增
用户 ID	use_id	int(11)	是	
昵称	real_name	varchar(32)	否	
视频地址	video	varchar(255)	否	
标题	title	varchar(255)	否	

4.4.4 评论表

该表主要用于存储用户所有的评论信息。

共有 5 个字段，分别如下：

1. ID：评论的唯一 ID，主键，自增；
2. 用户 ID：评论所属的作者 ID；
3. 视频 ID：评论所属的视频 ID；
4. 头像地址：评论作者的头像地址；
5. 内容：评论的内容；

表的具体内容如下表 4-4 所示：

表 4-4 评论表

中文字段名	英文字段名	类型	是否索引	备注
ID	id	int(11)	是	主键，自增
视频 ID	video_id	ini(11)	是	
用户 ID	user_id	int(11)	是	
头像地址	avatar	char(255)	否	
内容	content	char(255)	否	

5 详细设计和实现

5.1 登录

登录、注册界面如下图 5.1.1，图 5.1.2 所示：

图 5.1.1 登录页

图 5.1.2 注册页

登录主要有以下几个步骤：

1) 引导需要授权的用户到登录/注册标签下。

2) 如果用户输入用户名和密码，则通过 ajax 请求后端数据，验证正确后，向 LocalStorage 写入保存用户登录状态的 token，页面会跳转至首页。

关键流程如下：

前端：

1. 前端产生点击事件
2. 异步 ajax 请求
3. 请求参数，请求地址，
4. 改变 button 样式，使用户了解正在登录的状态
5. 如果 ajax 请求成功返回数据
6. 向 localstorage 写入用户名；
7. 向 localstorage 写入 token；
8. 向 localstorage 写入昵称；
9. 执行个人信息刷新函数；
10. 标题栏用户昵称的显示
11. 延迟 0.5 秒以后
12. 改变按钮样式使用户知道登录成功了
13. 延迟 0.5 秒以后跳到首页
14. 输出错误信息
15. 页面刷新

后端：

1. 路由分配请求到相应模块
2. 判断应有参数
3. 进行 mysql 数据库操作查询用户名是否存在
4. 如果用户名存在
5. 判断密码是否一致
6. 生成随机 token
7. 将 token 写入 redis
8. 如果写入成功
9. 给前端返回成功
10. 如果失败给前端返回，写入 redis 失败
11. 给前端返回密码输入错误

12. 给前端返回用户名不存在
13. 给前端返回参数错误

注册主要有以下几个步骤：

1) 引导需要授权的用户到登录/注册标签下。

2) 如果用户输入用户名和密码，则通过 ajax 请求后端数据，验证格式正确后，后端写入数据库，向 LocalStorage 写入保存用户登录状态的 token，页面会跳转至首页。

关键流程如下：

前端：

1. 前端产生点击事件
2. 进行 ajax 请求
3. 请求地址，请求参数，
4. 请求发送的同时，改变按钮样式使用户知道当前状态
5. 按钮变为不可点击
6. 请求成功，返回数据
7. 判断 http 状态码为成功
8. 自动点击登录按钮
9. 返回错误信息
10. 5 秒以后，自动刷新网页

后端：

1. 注册接口
2. 判断参数是否存在
3. 如果参数存在
4. 进行 mysql 数据库操作，查询所传输的用户名是否为已存在
5. 如果已存在，给前端返回已存在错误码
6. 如果用户名不存在，进行 mysql 数据库操作，插入注册用户的数据
7. 回调函数:给前端返回注册成功

5.2 网站首页

网站首页界面如图 5.2 所示：

图 5. 2 网站首页图

网站首页主要是按照时间倒序展示所有视频，列表里的内容包括：视频题目，作者和视频缩略图，剩余的评论需要打开展开评论页签。首页还运用了 ajax 技术做了无刷新底部自动翻页，极大的优化了用户体验。此外，主页还使用 HTML5 的本地存储技术来提高用户体验和减轻服务器负担[12]。

首页的前端 JavaScript 通过 Ajax 技术加载 html 的关键流程如下：

1. 进行 ajax 请求方式配置
2. 进行 ajax url 配置
3. 如果成功的回调函数
4. 循环渲染新建网页节点元素
5. 向元素中插入题目，作者，视频链接，评论链接
6. 如果数据量小于 10
7. 在尾部加入”没有更多了文字节点”

8. 滚动监听

5.3 视频评论展开

视频评论展开界面如下图 5.3 所示：

图 5. 3 视频评论展开图

视频评论展开主要是展示视频的评论和评论作者列表， 其中评论列表的伪代码后端实现如下：

1. 评论列表接口
2. 判断参数是否正确；
3. 如果不正确，给前端返回相应的错误信息；
4. 如果正确，则进行账号密码验证操作；
5. 如果账号密码验证未通过；
6. 给前端返回账号或密码错误信息；
7. 如果账号密码验证通过；
8. 进行 mysql 数据库操作读取评论列表；
9. 给前端返回使用 mysql 返回的数据编码成的 json 数据格式

5.4 发布视频

发布视频的界面如下图 5.5 所示：

图 5. 4 发布视频页面

发布视频主要用于登录用户发布视频，用户可以在这个界面选择文件，编辑标题，其中关于帖子内容的合法性检查在前端和后端都有设置[13]，这样既保证了快速反馈又保证了安全性[14]。

上传视频的后端关键伪代码如下：

1. 上传视频接口
2. 进行用户登录状态校验，如果未通过，给前端返回相应错误信息；
3. 如果通过，则生成 multiparty 对象，并配置上传目标路径
4. 上传完成后进行处理
5. 给处理函数传入视频文件和视频文件一起传入的表单信息；
6. 进行参数判断和帖子内容的合法性检查
7. 如果通过，则进行视频文件名处理，生成随机文件名
8. 将上传的缓存文件移动至相应文件夹并重命名为随机文件名
9. 进行 mysql 数据库操作，插入视频信息，并和作者建立对应关系
10. 如果失败，则给用户返回相应错误信息
11. 如果成功，则返回前端，上传成功

6 系统测试

系统测试是以整个系统为对象，检查各项设计是否符合需求分析的要求[15]。如果存在不合需求规格的情况，则需要找出问题的根源并及时进行修改。

6.1 测试目的

系统测试的目的在于根据需求分析的功能设计定义对整个系统各项功能点进行详细核对，查找出与需求分析不符或存在功能遗漏的地方，对这些不完善的功能点进行调整修改[16]。系统测试是为了确保系统功能的一致性和完整性，系统在投入实际使用以后能否正常运行正是通过全面而详细的系统测试来保证的[17]。

6.2 测试范围

此次系统测试的范围包括界面测试，功能测试。功能测试主要是检测软件的功能是否如预期，主要的根据是需求分析里的内容，功能测试是必不可少的，界面测试是确保各个窗口的图标、提示信息、风格等符合功能设计要求，保证用户界面的易操作性、无歧义。

6.3 测试方法

此次系统测试采用手工进行黑盒测试。

黑盒子测试只关注应用暴露给外界的功能，而不关注软件的内部逻辑的结构是什么[19]。黑盒子测试是一种在应用的暴露给外界的接口上进行测试的方法，即看最终测试结果是否满足系统需求的功能要求，正确输入时能否保证被系统正确接收并正常输出，以及是否可以保持外部信息的完整性等20。

6.4 测试环境

测试环境的具体参数如下表 6-4 所示：

表 6-4 系统测试环境表

硬件环境	用户端	应用服务器	数据库服务器
硬件配置	处理器：Intel Core i5-3470 主频 3.20GHz 内存 8.00GB	CPU 1 核 内存 1GB	CPU 1 核 内存 1GB
软件配置	Windows8.1 专业版 Chrome 浏览器	Ubuntu Server 14.04.1 LTS 64 位 Node.js 5.0.0	Ubuntu Server 14.04.1 LTS 64 位 MySQL 5.6
网络环境	200M WLAN	公网带宽 1Mbps	公网带宽 1Mbps

6.5 测试用例

测试用例如下表 6-5 所示：

表 6-5 系统测试用例表

编号	用例	操作	期望结果	实际结果	通过
1	浏览网页	点击页面上的链接或按钮按钮，随意查看各个页面	测试页面结构中框架、颜色、字体等符合基准，导航、链接、按钮、提示信息的正确一致性。	同期望	是
2	登录	登录按钮，输入正确用户名和密码	提示登录成功，并跳转到首页	同期望	是
3	登录失败	点击登录按钮，输入错误密码	提示用户密码输入错误，登录失败	同期望	是
4	未登录发布视频	在未登录状态下，点击发布视频的链接	提示需要登录，并跳转至登录页	同期望	是
5	发布视频	登录状态下，上传视频，并点击发布按钮	视频出现在首页帖子列表里	同期望	是
6	发布一个标题为空的视频	登录状态下，在标题框内不输入任何内容点击提交	提示标题不能为空	同期望	是
7	发布视频时未上传文件	登录状态下，在发布视频时未上传文件	提示未上传文件	同期望	是

表 6-5 系统测试用例表（续表）

8	未登录发布评论	未登录状态下，点击某个视频下的评论框	提示需要登录，并跳转至登录页	同期望	是
9	发布一条评论	登录状态下，在某个视频下的评论框内输入一些内容	内容出现在评论列表里	同期望	是
10	发布一条空评论	登录状态下，在评论框内不输入任何内容点击提交	提示内容不能为空	同期望	是
11	视频发布者删除本人发布的视频	登录状态下，点击自己发布过的某视频的删除按钮	提示删除成功	同期望	是
12	普通用户删除非本人的视频	登录状态下，调用删除非本人发布的视频的接口	提示只有本人才能删贴	同期望	是
13	未登录删除视频	未登录状态下，调用删贴的接口	提示需要登录	同期望	是

6.6 测试结论

此次系统测试执行了系统用例 13 个，通过 13 个，系统已经完成了需求分析中的所有功能需求。除此之外系统还具有很好的易用性，操作简明，界面清新，易于使用[20]。

7 总结与展望

论文分析了一个异步视频分享平台的原理和实现思想，设计了一个具有简单功能的视频上传分析和评论系统，实现了主要功能模块。探索了二进制数据的传送和前后端分离模式下模块的实现，研究了数据在前后端的交互和系统模块的连接。

设计从后端接口开始,设计整个系统的架构,在前后端分离的模式下,对后端数据库到前端渲染页面的各个阶段进行了简单介绍，包括 ajax 异步请求，还有二进制文件的传输。接下来就是系统模块设计和实现。使用模块图，用例图对模块功能进行了简单描述和划分，使用流程图对每个模块的具体实现进行了讲述。然后是系统的应用，也就是测试，分别阐述了登录模块、注册模块、评论模块、上传模块等 13 个测试用例的具体操作。

总的来说，论文不单单是一个文件上传平台的设计与实现，还包括对异步传输、文件编码解码，无刷新页面加载数据这些方面的研究。

设计实现了需求分析中的需求,但还存在以下问题：不良视频和标题信息过滤，目前只能通过管理员发现后再手动删除，以后的优化中会加入举报机制来避免这个问题；在视频的上传中，采用了异步上传的方式，导致可能在上传后，不能及时地加载出最新的视频，在以后的优化中，可以通过上传进度条的方式来改善用户体验；在开发过程中太过于注重接口和逻辑等后端功能的实现，以至于前端界面的样式和用户体验侧重得不多，所以页面看起来比较简陋，在以后的优化中，将会改进这方面问题。

致 谢

在这里我首先要向培养和关心我的导师陈虎老师表示深深的感谢，从大四的毕业设计开始以来，在选题、开题报告、提纲、系统的设计和编写论文开始，他都给了我关心和指导，不仅让我在大学将要毕业的时候系统的体会了一个计算机作品从最初的设计到开发出来所需要的严谨求实的态度，导师渊博的知识也让我受益匪浅。

感谢这四年以来学校教过我的和没教过我的各位老师，尽管很多老师相处仅有一个学期，也总是那么认真负责，尽心尽力的上完每一堂课，教会我各种知识。课下也不顾劳累的解答我们的问题。

感谢这四年以来的同窗，我们曾踏着铃声进入教室，曾在长桥上追逐打闹，也曾排着队挤进在图书馆奋战到傍晚，是你们丰富了我的大学生活，无论伤心快乐都陪在我身边，一起欢笑，一起成长。

感谢开创计算机科学的先驱们，是你们锐意进取和严谨的工作态度让我能够生活在这个互联网的世界。

最后，感谢我的父母把我送进了四川大学，也感谢我的母校给我带来的视野和学习环境，在今后的工作与学习中，我将不断的努力，发扬川大人的风采，为学校争光。

参考文献

[1]Nicholas C Zakas 《JavaScript 高级程序设计》 修订 3 版 北京：人民邮电出版社，2012：55。

[2]Paco Hope 《Web Security Testing Cookbook》 修订 1 版 北京：清华大学出版社，2010：38。

[3]邱永华 《XSS 跨站脚本攻击剖析与防御》 修订 2 版 北京：人民邮电出版社，2013：112。

[4]上野 宣 《图解 HTTP》 修订 2 版 北京：人民邮电出版社，2014：101。

[5]Jeremy Keith 《JavaScript DOM 编程艺术》 修订 2 版 北京：人民邮电出版社，2013：55。

[6]曾探 《学习 Javascript 数据结构与算法》 修订 4 版 北京：人民邮电出版社，2006：233。

[7]Nicbolas C. Zakas 《高性能 JavaScript》 修订 2 版 北京： 电子工业出版社，2008：43。

[8]Steve Souders 《高性能网站建设指南》 修订 2 版 电子工业出版社，2007：119。

[9]KYLE SIMPSON 《你不知道的 JavaScript》 修订 2 版 北京：人民邮电出版社，2010：228。

[10]单东林 张晓菲 魏然 《锋利的 jQuery》 修订 2 版 北京：人民邮电出版社，2005：201。

[11]Matthew MacDonald 《HTML5 秘籍》 修订 4 版 北京：人民邮电出版社，2007：303。

[12]Eric A.Meyer 《CSS 权威指南》 修订 3 版 北京：中国电力出版社，2009：330。

[13]David Sawyer McFarland 《CSS3 秘笈》 修订 2 版 北京：电子工业出版社，2013：177。

[14]徐涛 《深入理解 Bootstrap》 修订 2 版 北京：机械工业出版社，2015：163。

[15]W · Richard Stevens 《TCP/IP 详解 卷 1：协议》 修订 1 版 北京：机械工业出版社，2014：211。

[16]Randal E.Bryant / David O'Hallaron 《深入理解计算机系统》 修订 2 版 北京：机械工业出版社，2012：57。

[17]Thomas H.Cormen / Charles E.Leiserson / Ronald L.Rivest / Clifford Stein 《算法导论》 修订 2 版 北京：机械工业出版社，2011：90。

[18] lfred V. Aho / Monica S.Lam / Ravi Sethi / Jeffrey D. Ullman 《编译原理》 修订 1 版 北京：机械工业出版社，2010：11。

[20] Erich Gamma / Richard Helm / Ralph Johnson / John Vlissides 《设计模式》 修订 1 版 北京：机械工业出版社，2016：220。

[20]朴灵 《深入浅出 Node.js》 修订 2 版 北京：人民邮电出版社，2013：35

[21]阮一峰 《ES6 标准入门》 修订 1 版 北京：电子工业出版社，2015：6。

附录 1 论文译文与原文

Source：http://www.computingreviews.com/hottopic/hottopic_essay_09.cfm

译文：

开源：软件的黑马？

1983 年，一个叫 RichardStallman 的人创建了一个叫做 GNU（GNU 不是 Unix 的首字母缩写）的类 Unix 操作系统，他是使用的一个开源许可条款发布的该系统，这样就可以让其他开发者在该许可条款下来使用这个开源软件或重新开发它。八年后，在赫尔辛基大学毕业的 Linus Torvalds，也开发了一个叫做 Linux 的类 Unix 操作系统，和 GNU 一样，它也是免费提供给公众使用的。一直到现在 Linux 操作系统和 GNU 操作系统都还在被全世界各地的人们广泛的使用着，它们的演化过程在事实上促进了许多开源软件（OSS）计划的实施。而在 1999 年，一个叫做 Eric Raymond 的伟大的开源软件开发者，发表了他的著名论文《开源软件与商业软件开发之间的比较》，里面详细的讲述了商业软件开发的秘诀，几乎就是宗教的经验。由 Stallman, Torvalds, Raymond 和其他人所创建的文化和进程奠定了开源软件运动的基础。

OSS(开源软件)指的就是任何人都可以免费使用、修改和再发行任何软件，只要其遵守该软件许可条款即可。开源软件被广泛地部署在全世界各地的学术结构，商业公司，政府和非营利企业中，它在一个机构没有大型软件预算的情况下发挥了非常重要的作用，尤其是在欠发达的国家地区和非营利组织中。总的来说，成千上万的开发者们作为志愿者，在他们间接获益的老板的支持下，花费了大量的时间在开源项目上。

还有一些软件是除了开发它的个人、团队或组织以外，任何人都不可以修改它们的源代码，这种软件通常称为“私有软件”或“闭源”软件，只有原作者团队才可以合法复制或修改它的源代码。 微软的 Microsoft Word 软件和 Adobe 的 Adobe Photoshop 软件的是非常典型的闭源软件。为了能够使用闭源软件，计算机用户必须同意他们的协议（通常是在首次运行它的时候出现其用户协议条款），如果不同意的话，那么你就无法使用该该软件做任何事情。

但是开源软件就不一样了，开源软件的作者把他们的源代码提供给那些愿意阅读他们的代码、愿意复制他们的代码、愿意学习他们的代码、愿意修改他们的代码或愿意分享他们的代码的人。 LibreOffice 的和 GNU 图像处理程序就是非常典型的开源软件的例子。和闭源软件一样，当你在使用开源软件的时候，也必须接受它们的许可条款，但是，开源许可证的条款与闭源软件的许可证的条款有着显著的差异。因为开源软件的许可条款是允许其他人来修改它的源代码并且把它变成自己的项目，所以开源软件的许可条款其实是促进协作和共享的。有一些开源许可条款规定：修改过其源代码的人，必须也共享其开发的程序的源代码，而且绝对不允许向用户收取任何费用。换句话说，只要程序员喜欢，那么他就可以阅读、查看和修改其源代码，但是他们在分享自己的成果时，也必须采用开源并且得采用同样的许可条款。事实上，如果他们不这么做的话，他们就可能会违反一些开源许可证的条款。

因此，作为开放源码促进会的负责人解释说，“开源并不仅仅意味着能够阅读源代码”。它还意味

着，任何人都能够为了满足其他人的需要去修改其源代码，而且也不应该阻止他人做同样的操作。

开源软件不管对程序员还是非程序员来说其实都是非常有利的。事实上，很多互联网服务本身都是建立在开源技术之上的，如使用 Linux 操作系统和 Apache Web 服务器作为运行环境，而每次计算机用户在浏览网页，收取和发送电子邮件，与朋友们在网络上在线聊天，播放流音乐，或者玩多人的视频游戏，他们的电脑、手机或游戏机也会通过路由连接到全球的网络中。

而做这项重要工作的计算机通常会位于很远的地方，用户一般来讲是看不到他们的，这也就是为什么有些人会把这些电脑称为“远程计算机”的原因。越来越多的人开始依赖远程计算机来完成自己的事情。例如，他们使用在线文字处理软件，使用在线电子邮件管理系统，他们不需要在个人电脑上安装和运行图像编辑软件、电子邮件处理软件，相反，他们只需通过使用 Web 浏览器或手机应用程序访问远程计算机上的这些程序即可。

大部分关于 OSS(开源软件)的研究并不直接涉及到 OSS(开源软件)本身的流程或源码，相反的是，大量的研究使用可以进行分析和测试的方便的工业级别的软件模型 OSS(开源软件)。开源软件也被广泛的应用在从幼儿园到高中再到本科研究生的课堂上，也有很多篇论文中提到了在教育中使用开源软件的重要性。但是，具体在涉及到 OSS(开源软件)的研究中的几个重要的点其实已经出现了很多年了，下面将分别来叙述：

一、采用开源软件的决策和其商业价值

开源软件目前已经成为 IT 基础设施的大多数非营利性和许多营利性企业的重要组成部分。大多数公司可能都会有一些 OSS(开源软件)的部署，也许有些是在不知不觉中使用它的，并且有可能这些开源软件是禁止他使用的。例如，有很多“不开源”的企业却在使用 Apache 的 Tomcat Web 服务器或者在使用用一些开源的 Java 库。再比如说这些被广泛部署的开源软件：比如说 Linux、 JBoss 的 Java 应用服务器，再比如说编程语言里的 PHP 语言，Python 语言和 Ruby 语言，又或者是开源的 MySQL 数据库。

使用开源软件的商业价值集中体现在其获得了大量免费的工业级别的代码。然而，由于在特定开源许可协议中所规定的义务与开源社区对待特定应用程序支持的活跃程度的不确定性，以及对于开源软件的误解，这些通常会导致商业组织不选择使用开源软件。其他的一些针对 OSS（开源软件）的反对是由于他们对 OSS(开源软件)质量固有的不信任造成的，然而其实一些研究已经表明，开源软件的质量实际上比闭源软件的质量还要高，不过也有一部分研究反对这个说法。

在企业中使用开源软件有很多种方法，从组件到单个的应用程序，再到相关的应用程序套件，甚至是完整的企业解决方案都可以使用开源软件来构建。我们可以在开源软件代码仓库比如说 SourceForge 中找到各种各样的 OSS(开源软件)项目。因此，对于大部分决定是否使用 OSS(开源软件)的决策者来说，主要需要评估 OSS（开源软件）在企业中使用的兼容性，以及使用开源软件所带来的整体风险和整体收益。这个研究项目已经催生了许多为开源软件准备使用模式的公司，比如说 Capgemini，Navica，Spikesource，Intel，(SEI)-West 这些公司。

该研究对此提供了以下结论：

- 没有任何一个适用于所有 OSS(开源软件)的解决方案
- 企业需要谨慎的获取开源项目和适应开源项目的环境（而不是相反）

- 开源模式对解决环境一致性有非常大的帮助

二、法律问题（许可条款和知识产权）

有超过 100 的授权模式，他们自称是开放源代码的变种，每一个许可模式都具有不同的权利和责任。其中有 4 个起源于 20 世纪 90 年代的许可模式，这 4 个模式是目前最为普遍使用的模式，如下：

- 通用公共许可证（GNU GPL）
- GNU 通用公共许可证（LGPL），
- BSD 许可证
- MIT 许可证

这些许可证和其他的许可证之间的区别有时是重要的，有时也是微小的，而对这些差异的研究却是很成熟的。总体来说，开源软件许可条款允许任何人不受限制地使用开源软件，只需他们由此产出的任何衍生作品与未来可能的衍生作品都遵守相同的许可条款即可。软件专利往往和 OSS(开源软件)是冲突的，当一个开源项目的许可证写的不清楚时，后者以某种方式引入他人的知识产权就是很可能发生的情况。因此，某些对 OSS(开源软件)的研究集中在知识产权、许可、义务、责任、犯罪、法律策略和案例研究上。

关于这一领域的研究我们可以参考以下指导意见：

- 当你决定选择使用一个 OSS（开源软件）时，很重要的一个前提就是阅读它的许可条款
- 使用开源软件是一个技术的决策，项目管理的决策和法律的决策，所有企业在做这个决策前请务必咨询有关这 3 个方面的事情。

三、开源软件的质量

开源软件经常说其相比于闭源软件具有更大的优势，比如说它出现的问题机率会更少。Eric Raymond 有句名言，“只要开源软件有很多人关注，那么所有的问题都将被发现”。但是有一个在比较了各种特性尤其是安全性，可靠性，可维护性和可测试性的研究却表明：流行的开源软件和闭源软件所暴露的问题有时是不确定的甚至是相互矛盾的。

许多对开源软件和闭源软件比较的研究都是从两者截然不同的发展过程和可用的文档开始的。开源软件是一个高度进化发展（有些人会说“敏捷”）的过程。开源社区里的软件往往不使用测试计划，测试工具，代码覆盖率的概念和工具（对于 OSS 项目的随机查看证明了这一事实）。但是，有证据表明，开源软件中出现的问题被发现并被修复的速度要比闭源软件更快，也许这是因为开源问题跟踪工具的存在，比如说 Bugzilla（www.bugzilla.org）。

研究表明，开源软件的质量参差不齐，一些 OSS(开源软件)的展示可以导致某些软件品质的改善。而另外一些研究却得出了相反的结论，还有一些则没有定论。所以评估 OSS(开源软件)的项目，还是要根据具体情况分析，不能泛泛而谈。

四、开源社区的特征

Raymond 比较了 OSS(开源软件)的一个社区，在那里，很多开发者在贡献自己的代码（代码贡献者被称为提交者）或对社区里的软件进行同行评审（在这种情况下，测试和对软件的新功能建议都会保存在其社区中）。很多的研究一直专注于如何去激励个人加入这样的社区，它们是如何组织的，以及社区

群体的动力和管理策略。例如，Scacchi 声称开源社区拥有高度的适应性，他们是比较松散的组织，最佳团队人数为 5 到 15 人。开源社区的文化甚至被比喻为黄蜂和其他昆虫聚集在社交网络进行互动。

对开源社区这个集体的研究暗示了它的性质是权衡问题出现的比例，新版本上线的速度以及有一个响应用户的社区是否重要。开源社区在做出采用企业开源代码解决方案的决定时，一定要对以上问题进行评估。

五、源代码的结构和演化

由于源代码的可用性，往往会有大量多年前发行的版本存在，OSS(开源软件)存储库为其提供了一种方便实验室检查的相对大型和成熟的系统。例如,代码结构进化的纵向研究可以使用数十个指标进行任意组合。最近的研究特别关注 Linux 和 Apache 等重要项目,但也关注一些在游戏和其他公用事业发展的工作。

为了辨别一些有趣的软件或社区，其他构件如开发人员日志、错误报告、用户手册、和其他文档都被用来研究一种称为“软件考古学”的项目。而数据在公共存储库中使得一些有趣的研究变成了可能,比如一个流行的游戏 NetHack,是一个第一人称的地牢类游戏，其 Dragons-type 版本的源代码已经过去了 20 年。

对开放源代码结构和演化的研究是鱼龙混杂的。有些研究显示，软件结构随着时间的推移变得复杂，而另外的研究表明，软件结构实际上是提升了效率的。再说一次，不要对于 OSS(开源软件）的代码结构的质量太过依赖。

六、启用 OSS(开源软件)和应用的工具

在 OSS(开源软件)领域，包括脚本语言 Perl，Python 和 PHP 和 Ruby 等重要的软件开发工具都在蓬勃的发展中。开源软件发行中其他有用的开发工具还包括 Ant 和 Maven 构建应用程序; Hibernate，作为一种面向对象的持久层数据库; xUnit，用来测试; CVS 和 Subversion 用来做源代码的版本控制; Eclipse 和 NetBeans 是集成的开发环境。软件工程师也在使用开源的 Struts，它提供了一个框架，使应用程序可以快速地使用 MVC 体系架构来构建。Swing，提供了一个管理经典的业务对象的分层结构。协同市场的本质，导致了对创造条件、开放的延伸和高度互操作性的工具和库的前所未有的运动。

大多数围绕开源工具的研究大多包括案例研究和新的工具项目申请报告。一些研究涉及代码结构，演化，或对工具本身的社区进行分析。

七、哲学和道德问题

软件应该是免费的是开源软件的基本主张。所有提出的有关哲学和伦理的问题都围绕软件的所有权,个人动机、社会权利和义务等等。有几个小组的存在在客观上促进了开源软件的开发和共享，特别是开源组和自由软件基金会这两个组织。虽然一些以营利为目的的公司已经找到了一些方法来使得开源软件对他们有利，一些其他企业，主要软件供应商们，要么故意视而不见 OSS(开源软件)的重要性，要么直接攻击它，有些人呢，会比较注重 OSS(开源软件)对软件产业本身的影响。这些问题对于以营利为目的的软件厂商和开源社区的发展的研究又贡献了一个有趣的领域。

正如任何哲学的研究,在这一领域的研究也是多样化的,包括促进开源模式及其在相关领域中的应用,如开放获取期刊或反对某些应用领域使用 OSS(开源软件)的概念。

结论

如今已经有超过 25 万个开源项目，范围从简单的学术研究到游戏，编程语言，工具和企业级应用程序都有其的身影。众多知名的桌面和企业级应用都是在克隆开源的软件之上进行开发的，开源软件对于小企业，非营利性机构，甚至贫穷国家的政府都是非常重要的。显然，OSS(开源软件)具有重要的全球性的经济价值，而且还 OSS(开源软件)还提出了一些具有挑战性的哲学问题，与时俱进的软件工程师所思考软件的方式应该包括它。

未来对于开源软件的研究势必需要解决更多的问题和挑战，这些问题包括：需要用经济激励以促进开源来吸引以赢利为目的的企业以推动创新项目吗？需要将 OSS(开源软件)主要停留在学者和自由软件支持者的范围吗？参与开源社区的动机是什么，如何将这些动机和企业、国家协调起来呢？OSS(开源软件)强大的安全性和可靠性够不够商业化，甚至可提供给以安全为第一要素的政府软件来部署？OSS(开源软件）的质量和同等条件的闭源软件相比，哪个更好？如雨后春笋般涌现的开源运动以及各地以营利为目的的公司只是一时的现象还是可以长久的发展呢？

开源软件总是会存在，但其技术作用仍然没有得到有效解决。OSS(开源软件)即使是在商业软件世界里也非常重要，不容忽视，敬请期待未来更多的变化吧。

表 1： OSS 术语

术语	定义
提交者	开源社区中按照一定的规则修改源代码的开发者
GPL	GNU 公共许可，是最常见的开源许可协议之一，本质上它也是使用 GPL 许可的，在 GPL 许可证下的任何代码都必须提供源代码。GPL 因为它的自我传播性经常被戏称为“病毒”许可协议
开源软件 OSS	软件是免费使用的，但是再次开发的软件必须遵守其的许可条款，大量的许可模式拥有各种各样不同的权利和责任
开源软件存储仓库	可以在这里找到各种各样的开源软件，常用的有 SourceForge、RubyForge、Freshmeat 等
软件考古学	在软件程序的基础上，对它的开源存储仓库中的东西如开发日志、错误报告信息和文档的研究

原文：

Open Source: The Dark Horse of Software?

In 1983, Richard Stallman created a Unix-like operating system called GNU (a recursive acronym for “GNU is Not Unix”) and released it under a license that provided certain rights for use and redistribution—an open-source license. Eight years later, a graduate student at the University of Helsinki, Linus Torvalds, created another Unix-like operating system, Linux, which he also made available for free. Both Linux and GNU are still widely available, and their evolution spurred the creation of many other open-source software (OSS) programs. By 1999, a prodigious open-source software developer, Eric Raymond, published his famous treatise, comparing the development of open-source software to the market conditions found in a bazaar, and describing the development of commercial software as a secret, almost religious experience . The process and culture created by Stallman, Torvalds, Raymond, and others formed the basis for the open-source software movement.

OSS refers to any software that is available for free use, modification, and redistribution, provided the terms of one of many license models are followed. Open-source software is deployed throughout the world in academic, business, and government enterprises, and plays an important role in those settings where large software budgets are not possible, particularly in underdeveloped countries and in nonprofit organizations. Collectively, thousands of individuals contribute to open-source projects, as volunteers on their own time, and frequently with the support of their employers, who may indirectly benefit.

Some software has source code that cannot be modified by anyone but the person, team, or organization who created it and maintains exclusive control over it. This kind of software is frequently called "proprietary software" or "closed source" software, because its source code is the property of its original authors, who are the only ones legally allowed to copy or modify it. Microsoft Word and Adobe Photoshop are examples of proprietary software. In order to use proprietary software, computer users must agree (usually by signing a license displayed the first time they run this software) that they will not do anything with the software that the software's authors have not expressly permitted.

Open source software is different. Its authors make its source code available to others who would like to view that code, copy it, learn from it, alter it, or share it. LibreOffice and the GNU Image Manipulation Program are examples of open source software. As they do with proprietary software, users must accept the terms of a license when they use open source software—but the legal terms of open source licenses differ dramatically from those of proprietary licenses. Open source software licenses promote collaboration and sharing because they allow other people to make modifications to source code and incorporate those changes into their own projects. Some open source licenses ensure that anyone who alters and then shares a program

with others must also share that program's source code without charging a licensing fee for it. In other words, computer programmers can access, view, and modify open source software whenever they like—as long as they let others do the same when they share their work. In fact, they could be violating the terms of some open source licenses if they don't do this.

So as the Open Source Initiative explains, "open source doesn't just mean access to the source code." It means that anyone should be able to modify the source code to suit his or her needs, and that no one should prevent others from doing the same. The Initiative's definition of "open source" contains several other important provisions.

Open source software benefits programmers and non-programmers alike. In fact, because much of the Internet itself is built on many open source technologies—like the Linux operating system and the Apache Web server application—anyone using the Internet benefits from open source software. Every time computer users view webpages, check email, chat with friends, stream music online, or play multiplayer video games, their computers, mobile phones, or gaming consoles connect to a global network of computers that routes and transmits their data to the "local" devices they have in front of them.

The computers that do all this important work are typically located in faraway places that users don't see or can't physically access—which is why some people call these computers "remote computers." More and more, people rely on remote computers when doing things they might otherwise do on their local devices. For example, they use online word processing, email management, and image editing software that they don't install and run on their personal computers. Instead, they simply access these programs on remote computers by using a Web browser or mobile phone application.

Much of the research involving OSS does not pertain directly to the processes or artifacts of OSS itself. Instead, a great deal of research uses OSS as a convenient model of “industrial-strength” software that can be easily accessed for analysis and testing. Open-source software is also widely used in the classroom, from grades K-12 through the undergraduate and graduate levels, and many papers have been written about the use of OSS in education. However, several important lines of research pertaining specifically to OSS have emerged over the years.

Research Line 1: Open-Source Adoption Decision-Making and Business Value Proposition

Open-source software has become a critical component of the IT infrastructure for most nonprofit and many for-profit enterprises. Most companies likely have some OSS deployed, perhaps unknowingly, even if its use is forbidden. For example, many “no open-source” enterprises use the Apache Tomcat Web server or employ open-source Java libraries. Linux; the JBoss Java application server; programming languages such as PHP, Python, and Ruby; and the MySQL database are some of the most widely deployed open-source applications.

The business value proposition for using OSS focuses on access to a large amount of "free" industrial-strength code. However, uncertainty in the obligations under a particular open-source license, concerns about the vitality of the open-source community supporting a particular application, and simple misunderstanding of the model often lead organizations to choose not to use OSS. Other objections stem from an inherent distrust of OSS quality. Some research, however, has shown that OSS can achieve quality levels beyond that possible with closed-source software, although there is other research to contradict that assertion.

There are many ways to use OSS in the enterprise, ranging from a few components to single applications, suites of related applications, and complete enterprise solutions [2]. OSS projects can be found in open-source repositories, most notably SourceForge. Much of the research, therefore, with respect to the OSS adoption decision, centers around assessing the compatibility of OSS in the enterprise, and on estimating the overall risk and return on investment of using OSS. Some of this research has led to the creation of open-source readiness models, in particular, those of Capgemini, Navica, and Spikesource/Intel/Software Engineering Institute (SEI)-West.

The research offers the following conclusions in this regard:

- There is no "one-size-fits-all" version for any OSS solution.
- Organizations need to carefully access open-source projects and fit the solution to the environment (not vice versa).
- Open-source readiness models can be helpful in achieving proper alignment.

Research Line 2: Legal Issues (Licensing and Intellectual Property)

There are more than 100 licensing models, claiming to be open-source variants, with widely varying rights and responsibilities. However, four licensing models, which originated in the 1990s, are by far the most prevalent:

- general public license (GNU GPL),
- GNU lesser general public license (LGPL),
- BSD license, and
- MIT license.

The differences between these and other licenses are important and sometimes nuanced; the study of these differences provides a ripe line of research. But generally, an OSS license allows unrestricted use of the software, requiring only that any derivative works be redistributed along with the source code, and that future users of the derivative work comply with the same license. Software patents can often clash with OSS if the latter somehow includes intellectual property that belongs to another party, a situation that can occur when the technological genealogy of an open-source project is murky. Therefore, certain research in OSS focuses on the issues of intellectual property rights, licensing implications, obligations, liability, criminality, legal strategy, and case studies.

Studies in this area provide the following guidance:

- The licensing model attached to an open-source project is an important consideration when electing to use OSS.
- The decision to adopt OSS is a technical decision, a project management decision, and a legal decision. All three entities in an enterprise need to be consulted.

Research Line 3: Qualities of Open-Source Software

Open-source software is often claimed to be superior to closed-source code in many ways, including having less defects. Eric Raymond famously asserted that, “given enough eyeballs, all bugs are shallow” . But the research comparing various qualities—especially security, reliability, maintainability, and testability—of open-source to closed-source code, while vigorous, is sometimes inconclusive or contradictory.

Many of the difficulties comparing open- and closed-source code arise from the very different development process models and available documentation. Open-source software is developed in a highly evolutionary (some would say “agile”) process. Open-source development communities tend to not use test plans, testing tools, or code coverage concepts and tools (a random perusal of OSS projects will confirm this fact). However, there is evidence that defects are found and repaired faster than in closed-source software , perhaps because of the existence of open-source defect tracking tools like Bugzilla (www.bugzilla.org).

Research into open-source software quality is mixed. Some shows that OSS models can lead to improvement in certain software qualities. Some studies reach the opposite conclusion, and some are inconclusive. Evaluate OSS projects for quality on a case-by-case basis, and do not rely on generalities.

Research Line 4: Open-Source Community Characteristics

Raymond has compared the development of OSS to a bazaar, where large numbers of interested individuals congregate to contribute (code contributors are called committers) or to provide peer review of the wares in the market (in this case, testing and new feature recommendations for the software in the repositories). A great deal of research, then, has focused on what motivates individuals to join such communities, how they are organized, the group dynamics, and governance structures. For example, Scacchi asserts that open-source communities form a highly adaptive but loosely coupled virtual enterprise, organized in a “layered meritocracy” with a usual team critical mass of five to 15 people . OSS community cultures have even been likened to those of wasps and other swarming insects, in terms of the social network interactions .

The collective research in open-source community dynamics implies that the nature of the open-source community is important with respect to defect resolution rate, rate of new releases, and responsiveness to the user community. Open-source communities need to be evaluated when making the decision to adopt an open-source solution in the enterprise.

Research Line 5: Source Code Structure and Evolution

Because of the availability of source code, often over many years of releases, OSS repositories provide a convenient laboratory for examining relatively large and mature systems over time. For example, longitudinal studies of code structure evolution using any combination of dozens of metrics can be conducted . Recent research has focused particularly on important projects such as Linux and Apache, but there is interesting work available on the evolution of games and other utilities.

Other artifacts, such as developer logs, bug reports, users manuals, and other documentation, can be studied (an approach termed “software archeology”) in order to discern interesting properties about the software or the community. Data in public repositories make interesting research possible—for some projects, such at NetHack, a popular first-person Dungeons and Dragons-type of game, versions of the source code go back more than 20 years.

The research in open-source code structure and evolution is a mixed bag. Some shows that software structure becomes entangled over time, while other research shows that code structure actually improves over time. Again, do not rely on generalities with respect to the code structure qualities of OSS.

Research Line 6: Tools for Enabling OSS and Applications

Important software development tools have thrived in the OSS arena, including the scripting languages Perl, Python, PHP, and Ruby. Other useful developer tools released as open source include Ant and Maven for building applications; Hibernate, which acts as an object-oriented persistence layer for databases; xUnit for testing; CVS and Subversion for source code control; and Eclipse and NetBeans as integrated development environments. Software engineers also use the open-source Struts, which provides a framework in which an application can be built quickly using the model-view-controller architecture, and Swing, which provides a layered structure for managing (typically) business objects. The very nature of the collaborative “bazaar” has led to an unprecedented movement toward the creation of tools and libraries that are open to extension and highly interoperable .

Most of the research surrounding open-source tools mostly comprises case studies and project application reports of new tools. Some studies involve an analysis of the code structure, evolution, or community of the tool itself.

Research Line 7: Philosophical and Ethical Issues

The underlying proposition that software should be free to all raises philosophical and ethical issues surrounding software ownership, personal motivations, societal rights and obligations, and more. Several groups exist today that promote the development and sharing of OSS, notably the Open Group and the Free Software Foundation. While some for-profit companies have found ways to leverage free software to their advantage, other companies, primarily software vendors, either turn a willful blind eye to the importance of

OSS, or outright attack it. Some have looked at the effects of OSS on the software industry itself . The intersection of those issues for for-profit software vendors and the open-source community provides an interesting field for informed discussion and research.

As in any philosophical endeavor, the research in this area is diverse, including arguments promoting the open-source model (and its application in related areas, such as open access journals) or against the notion of using OSS in certain application domains.

Conclusion

There are more than 250,000 open-source projects, ranging from simple academic pursuits to games, programming languages, tools, and enterprise-level applications. Clones of many well-known desktop and enterprise applications are available in open source, and these have become important to small businesses, nonprofit entities, and even governments of small and poor nations. Clearly, OSS has important worldwide economic value, but OSS also presents a number of challenging and philosophical issues. It is definitely changing the way that software engineers think about software.

Future research in open-source systems is going to be needed to resolve many of these issues and challenges. Questions that need to be answered by research include: Is the economic incentive to contribute to open-source projects sufficient to drive innovation at a level attractive to for-profit enterprises? Will OSS remain largely in the purview of academicians and free software proponents? What are the motives for participating in open-source communities and can these be reconciled with loyalty to company and country? Is the security and reliability proposition for OSS strong enough for commercial applications and even secure government deployment? Is the quality of OSS better, worse, or the same as closed-source software? Will the for-profit companies that have sprung up around the open-source movement be able to survive and grow, or are they simply a momentary phenomenon?

Open-source software is always going to exist, but its technological role remains unresolved. OSS is too important to ignore, even in the commercial software world. Stay tuned for more changes to come.

Table 1: OSS terminology

TERM	DEFINITION
Committer	A member of an open-source community who makes changes to the source code based on the rules of the community.
GPL	GNU public license, the most common open-source licensing model, which essentially says that any code that uses GPL licensed code must also be made available and freely distributed under the GPL. GPL is often derisively called a “viral” licensing model because of its self-propagation.
Open-source software (OSS)	Software that is free for use or redistribution provided that the terms of a licensing agreement are followed. There are many different licensing models with varying degrees of rights and responsibilities.
Repository	A place where OSS can be found and where the associated communities are hosted. Typical repositories include SourceForge, RubyForge, and Freshmeat.
Software archeology	The study of a software program based on artifacts found in its open-source repository, such as developer logs, bug reporting information, and documentation.