2023-11-7

DingVideo

——Web 端短视频应用

设计文档

翁祯敏 丁子晨 崔里青
华东师范大学

目录

1. 项目介绍

1.1 项目简介

1.1.1 背景

依托第二届七牛云 1024 创作节的赛题，项目团队基于七牛云相关产品开发了一款 Web 端短视频应用：DingVideo。

1.1.2 名称

项目名称"DingVideo"的来源不仅是一位团队成员的姓氏（丁），而且音同微波炉加热完食品发出的"ding——"，选用此名的用意是：这款 Web 端短视频应用像微波炉一样高效，为用户带去轻松愉悦的用户体验。正如微波炉能在短时间内将食物加热到理想温度，DingVideo 也会在当下快节奏的现代生活中以高效便捷的方式为用户呈现丰富多样的短视频内容，成为一个集创意、娱乐、互动为一体的优质短视频平台。

1.1.3 logo

logo 的中心为大写的字母 D，即 DingVideo 的首字母，同时该字母延伸出一条线，让该字母呈现出摇曳着翅膀的蝴蝶模样，蝴蝶的四周是多彩的洋流，寓意着使用本平台的用户将像一只自由飞翔的蝴蝶一样，在平台丰富多彩的视频洋流中享受随时随地的乐趣和轻松。

1.1.4 意义

在全民自媒体的时代，DingVideo 为用户提供了一个创作和表达自我的平台，用户通过拍摄、编辑、上传短视频来展现自我，突出个性，传播创意和趣味，促进人与人之间的沟通交流。DingVideo 为所有用户提供了一个便利的社交互动场所，用户可以通过点赞、收藏、评论等方式与其他用户互动，形成以 DingVideo 为中心的社群。此外，DingVideo 的视频内容涉猎广泛，包括体育、游戏、美食、动漫等类别，让人们可以在碎片化的时间里快速获得短视频带来的轻松娱乐体验。

总之，DingVideo 是一个兼具娱乐、创作和社交的多功能应用，旨在为用户提供便捷多样的数字化体验，满足用户的日常娱乐需求。同时也推动了数字化内容产业的发展，带来了新的商机和影响力。

1.2 项目地址

项目地址：https://github.com/Tamiflu233/DingVideo

本项目主要由 main 分支、dev 分支和各自开发人员 fork 的项目组成，其中：

- main 分支是项目的最终合并分支，开发人员不能随机使用 main 分支提交修改内容，最后项目阶段性功能完成后，队长将 dev 分支的内容 merge 到 main 分支。
- dev 分支用于 merge 各个开发人员各自提交到 feature 分支的内容，并检查是否有冲突，加以处理。
- 其他开发人员在自己 fork 的项目上进行开发，并及时拉取 main 分支的更新，每做完一项功能后通过 pr 提交到 dev 分支。

1.3 使用说明

🚩请务必先运行项目后端再运行前端

首先将项目从 github 的 main 分支上拉取下来：

```Bash
git clone git@github.com:Tamiflu233/DingVideo.git
```

接着通过编译器打开项目，本项目前端采用 VSCode 编译器进行开发，后端基于 IntelliJ IDEA 环境进行开发。

用 VSCode 打开 `dingvideofrontend` 文件，这里默认 nodejs、vue3 和 pnpm 已经安装完成。在 TERMINAL 黑框中输入以下代码：

```Bash
# 安装项目前端所需要的工具包
```

```
pnpm install
# 运行前端代码
pnpm run dev
```

如果终端没有报错，在浏览器中输入 http://localhost:9527/ 地址即可访问项目的前端。

打开 IntelliJ IDEA，导入项目后端文件 `dingvideobackend`，这里默认 IDEA 中 java1.8 和 maven 已经配置完毕。如果 `pom.xml` 文件中有报错，说明后端环境所需要的部分依赖未导入，等待 maven 自动把依赖导入即可。点击进入 `src/main/java/com/dataispower/dingvideobackend/DingvideobackendApplication.java` 文件，选中如下运行按钮：

控制台没有报错或停止，则项目后端程序成功运行。

2. 项目分工

团队成员	主要贡献
翁祯敏	前端基础架构构建、视频播放器、视频分类页面编写，文档编写
崔里青	视频详情卡片和个人中心的前后端，搭建视频内容库，文档编写
丁子晨	后端基础架构搭建，登录、注册前后端构建，服务接口开发，文档编写

3. 项目实现

3.1 技术选型

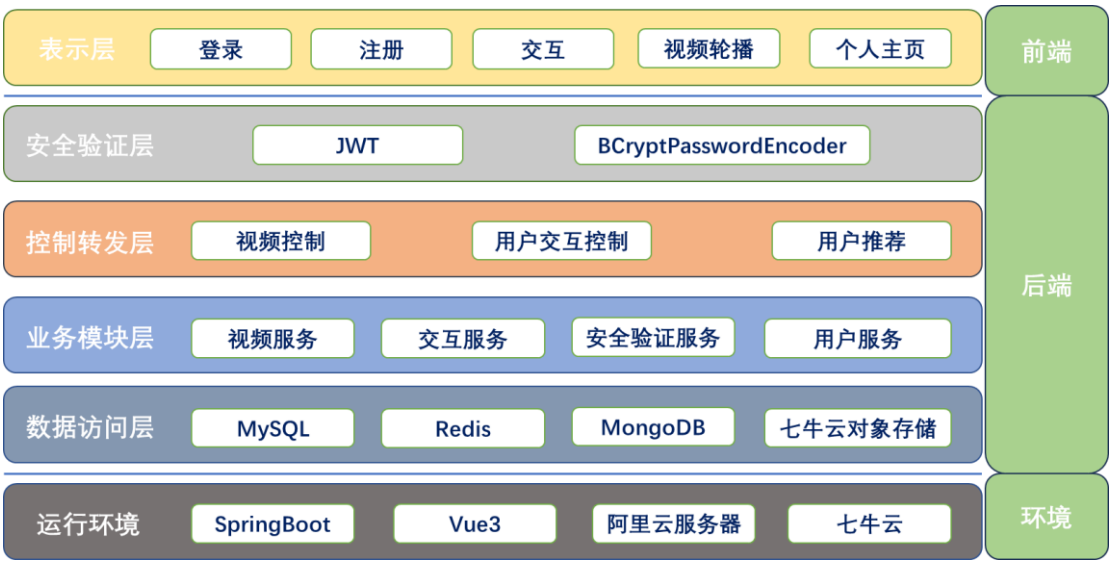
- Vue3：渐进式前端 JavaScript 框架。
- Springboot：基于 Java 的开源后端框架。
- JWT：用户 token 的生成和权限校验。

- BCryptPasswordEncoder：基于 hash 的对称加密算法，用于加密用户密码。
- MySQL：用户信息和视频信息的持久化存储。
- 七牛云：使用七牛云做对象存储，用来存储视频，图片等静态资源。
- Redis：缓存点赞、收藏、关注等用户操作，设置定时任务，并且使数据同步到 MySQL 数据库中。
- MongoDB：持久化用户 JSON 结构评论信息。

其中数据持久化部分，存储用户交互信息、评论信息和持久化信息的 Redis、MongoDB 和 MySQL 数据库部署在阿里云服务器上，视频和图片等静态资源文件存储在七牛云对象存储服务器中。

3.2 架构设计

项目基于三层架构的思想进行设计，可以分为表示层、安全验证层、控制转发层、业务模块层、数据访问层，其中安全验证层、控制转发层、业务模块层即三层架构中的业务逻辑层。



3.3 详细设计

项目采用前后端分离技术栈，前端采用 Vue3 框架进行开发，后端基于 Springboot 框架进行构建。前端和后端独立运行的开发模式将项目解耦，降低了维护上的复杂性，有利于团队之间的开发协作，前端可以专注于用户界面和用户体验，而后端可以专注于业务逻辑和数据处理。这可以加速开发速度，从而使团队能够更好地满足项目要求。

3.3.1 项目目录

3.3.1.1 Vue3 前端

```Markdown
dingvideofrontend
  ├─.env.development
  ├─.env.production
  ├─package.json
  ├─vite.config.ts
  └─src
      ├─App.vue
      ├─main.ts
      ├─api
      │   ├─login.ts
      │   └─video.ts
      ├─assets
      │   └─img
      │       ├─login.png
      │       └─register.png
      ├─components
      │   ├─common
      │       ├─IconButton.vue
      │       ├─LikeButton.vue
      │       ├─LoginAvatar.vue
      │       ├─NavHeader.vue
      │       ├─SideMenu.vue
      │       ├─ValidCode.vue
      │       ├─VideoCardRight.vue
      │       ├─VideoDescription.vue
      │       ├─VideoInfoCard.vue
      │       ├─VideoSwipe.vue
      │       └─VideoToolbar.vue
      ├─router
      │   └─index.ts
      ├─store
      │   ├─index.ts
      │   ├─global
      │   │   └─global.ts
      │   ├─login
      │   │   └─login.ts
      │   └─videos
      │       └─videos.ts
      ├─types
      │   ├─pinia-plugin-persist.d.ts
      │   ├─login
```

```
│   │   └─index.ts
│   ├─service
│   │   └─index.ts
│   ├─sidemenu
│   │   └─index.ts
│   ├─user
│   │   └─index.ts
│   └─videoInfo
│       └─index.ts
├─utils
│   ├─auth.ts
│   ├─date.ts
│   └─request.ts
└─views
    ├─detail
    │   └─VideoDetail.vue
    ├─home
    │   ├─index.vue
    │   ├─Category
    │   │   └─CategoryVideos.vue
    │   └─RecommendVideos
    │       └─RecommendVideos.vue
    ├─login
    │   ├─login.vue
    │   └─register.vue
    └─user
        └─index.vue
```

- .env.development：前端项目 development 模式环境文件，存放了一些环境变量，例如后端 api 的 base url、前端的端口号等
- .env.production：前端项目 production 模式环境文件，存放了一些环境变量，例如后端 api 的 base url、前端的端口号等
- package.json：前端依赖清单，在项目根目录下运行 pnpm install 或 npm install 即可安装
- vite.config.ts：vite 的配置文件
- src：存放了前端的核心代码：
  - App.vue：项目根组件文件
  - main.ts：项目入口文件
  - api：存放网络请求相关的代码

        - login.ts：与用户登录、注册和用户信息查找相关的网络请求代码
        - video.ts：与视频数据查询相关的网络请求代码
    - assets：存放一些静态资源，如图片
        - img：存放页面所用到的图片
    - components：存放项目封装组件的 vue 文件
        - common：存放可在多个页面复用的 vue 组件
            - IconButton.vue：视频工具栏的图标按钮组件
            - LikeButton.vue：点赞按钮组件
            - LoginAvatar.vue：登录头像组件
            - NavHeader.vue：顶部导航栏组件
            - SideMenu.vue：左侧菜单栏组件
            - ValidCode.vue：验证码组件
            - VideoCardRight.vue：视频右侧评论卡片组件
            - VideoDescription.vue：视频描述信息组件
            - VideoInfoCard.vue：视频展示卡片组件
            - VideoSwipe.vue：视频轮播器组件
            - VideoToolbar.vue：视频工具栏组件
    - router：存放页面路由配置文件
        - index.ts：页面路由配置文件
    - store：存放 pinia 的状态管理文件
        - index.ts：pinia 的基础配置文件
        - global：存放全局状态(如左侧菜单栏的激活 index)管理文件
        - login：存放登录状态（用户信息、token）管理文件
        - videos：存放视频状态（视频缓存池）管理文件
    - types：存放一些 interface 定义
        - login：存放登录、注册表单相关的 interface 定义
        - service：存放网络请求返回数据结构的 interface 定义
        - sidemenu：存放菜单项数据结构的 interface 定义
        - user：存放用户信息 interface 定义

        - videoInfo：存放视频信息 interface 定义
    - utils：存放工具函数
        - auth.ts：获取、删除 token 的工具函数
        - date.ts：将时间戳转换为日期、事件的工具函数
        - request.ts：对 axios 进行封装的网路请求工具函数
    - views：存放页面代码
        - detail
            - VideoDetail.vue：视频详情页面代码
        - home：存放主页的相关代码
            - index.vue：短视频主页的页面基础架构代码
            - Category
                - CategoryVideos.vue：视频分类页代码
            - RecommendVideos
                - RecommendVideos.vue：视频推荐页代码
        - login
            - login.vue：登录页代码
            - register.vue：注册页代码
        - user：
            - index.vue：个人信息中心页代码

3.3.1.2 Springboot 后端

```
Markdown
dingvideobackend
  ├─dingvideobackend.iml
  ├─pom.xml
  ├─src
  │   ├─main
  │   │   ├─java
  │   │   │   └─com
  │   │   │       └─dataispower
  │   │   │           └─dingvideobackend
  │   │   │               ├─DingvideobackendApplication.java
  │   │   │               ├─common
```

```
    |   |   |                   |   ├─Constants.java
    |   |   |                   |   └─TableColumns.java
    |   |   |                   ├─config
    |   |   |                   |   ├─AuthenticationConfigConstants.java
    |   |   |                   |   ├─QuartzConfig.java
    |   |   |                   |   ├─RedisConfig.java
    |   |   |                   |   └─SecurityConfiguration.java
    |   |   |                   ├─controller
    |   |   |                   |   ├─CollectController.java
    |   |   |                   |   ├─FollowController.java
    |   |   |                   |   ├─LikeController.java
    |   |   |                   |   ├─UserController.java
    |   |   |                   |   └─VideoController.java
    |   |   |                   ├─dto
    |   |   |                   |   ├─ResponseResult.java
    |   |   |                   |   ├─UserIndexResponse.java
    |   |   |                   |   ├─UserInfo.java
    |   |   |                   |   ├─UserLogin.java
    |   |   |                   |   ├─UserRegister.java
    |   |   |                   |   ├─UserResponse.java
    |   |   |                   |   └─UserUpdate.java
    |   |   |                   ├─entity
    |   |   |                   |   ├─Comment.java
    |   |   |                   |   ├─CommonEntity.java
    |   |   |                   |   ├─User.java
    |   |   |                   |   ├─UserCollect.java
    |   |   |                   |   ├─UserFollow.java
    |   |   |                   |   ├─UserLike.java
    |   |   |                   |   └─Video.java
    |   |   |                   ├─enums
    |   |   |                   |   └─Gender.java
    |   |   |                   ├─exception
    |   |   |                   |   ├─BusinessException.java
    |   |   |                   |   └─ErrorType.java
    |   |   |                   ├─filter
    |   |   |                   |   └─JWTAuthenticationFilter.java
    |   |   |                   ├─job
    |   |   |                   |   ├─CollectTask.java
    |   |   |                   |   ├─FollowTask.java
    |   |   |                   |   └─LikeTask.java
    |   |   |                   ├─mapper
    |   |   |                   |   └─UserMapper.java
    |   |   |                   ├─repository
    |   |   |                   |   ├─CollectRepository.java
```

```
    |   |   |               |   ├─CommentRepository.java
    |   |   |               |   ├─FollowRepository.java
    |   |   |               |   ├─LikeRepository.java
    |   |   |               |   ├─UserRepository.java
    |   |   |               |   └─VideoRepository.java
    |   |   |               ├─service
    |   |   |               |   ├─CollectServiceImpl.java
    |   |   |               |   ├─CommentServiceImpl.java
    |   |   |               |   ├─FollowServiceImpl.java
    |   |   |               |   ├─LikeServiceImpl.java
    |   |   |               |   ├─RedisCollectServiceImpl.java
    |   |   |               |   ├─RedisFollowServiceImpl.java
    |   |   |               |   ├─RedisLikeServiceImpl.java
    |   |   |               |   ├─UserServiceImpl.java
    |   |   |               |   ├─VideoServiceImpl.java
    |   |   |               |   └─interfaces
    |   |   |               |       ├─CollectService.java
    |   |   |               |       ├─CommentService.java
    |   |   |               |       ├─FollowService.java
    |   |   |               |       ├─LikeService.java
    |   |   |               |       ├─RedisCollectService.java
    |   |   |               |       ├─RedisFollowService.java
    |   |   |               |       ├─RedisLikeService.java
    |   |   |               |       ├─UserService.java
    |   |   |               |       └─VideoService.java
    |   |   |               └─util
    |   |   |                   ├─RedisCollectKeyUtil.java
    |   |   |                   ├─RedisFollowKeyUtil.java
    |   |   |                   ├─RedisLikeKeyUtil.java
    |   |   |                   └─UUIDUtil.java
    |   |   └─resources
    |   |       └─application.yml
    |   └─test
    |       └─java
    |           └─com
    |               └─dataispower
    |                   └─dingvideobackend
    |                       └─DingvideobackendApplicationTests.java
    └─target
```

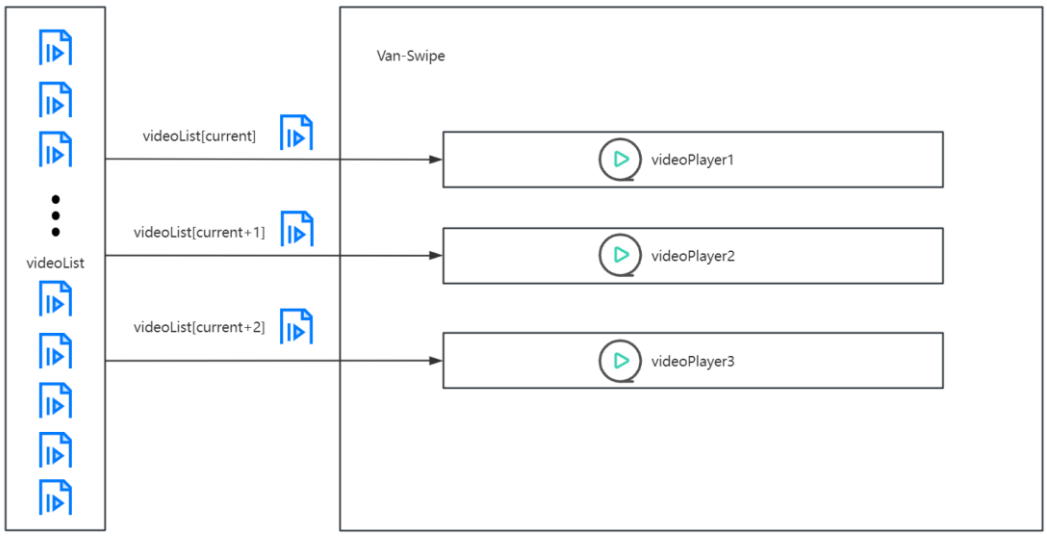
- pom.xml：项目所需依赖，maven 会根据 pom.xml 自动导入相关依赖。
- dingvideobackend.iml：maven 与 IDEA 集成的配置文件。

- src：存放项目后端相关文件。
  - main：存放项目后端相关的资源和 java 文件。
    - java：存放项目后端相关的 java 文件。
      - DingvideobackendApplication.java：springboot 项目启动文件。
      - common：存放后端通用的组件。
      - config：存放后端配置相关组件。
      - controller：存放后端表示层代码。
      - dto：存放用于前后端交互的实体类。
      - entity：存放与数据库交互的实体类。
      - enums：存放枚举类。
      - exception：存放项目定制的异常处理代码。
      - filter：存放过滤器代码文件。
      - job：存放定时任务的代码。
      - mapper：存放类之间相互映射的代码。
      - repository：存放 Java 持久层 API 代码。
      - service：存放服务实现代码。
        - interfaces：存放服务抽象类代码。
      - util：存放工具封装代码。
    - resources：存放项目后端相关的资源或配置文件。
  - test：存放项目模块与组件测试 java 文件。

3.3.2 核心模块

3.3.2.1 视频轮播

本项目的视频播放即翻页切换功能使用 Video.js 和 Vant4 组件库的 van-swipe 轮播器组件实现。考虑到短视频场景下往往视频数量很多，如果对每一个视频都渲染一个视频播放器可能会造成很大的开销，导致页面渲染缓慢、页面卡顿等现象。为此本项目的视频轮播功能使用了类似缓冲池的思想，在轮播器里只渲染三个视频播放器，页面初始化时先从视频列表中取出当前视频及其后面两个视频放入缓冲播放器内，如下图所示：



为了能实现无缝切换视频，需要考虑从 videoPlayer3 向下翻页和从 videoPlayer1 向上翻页的情况，可以利用 van-swipe 的 onDragStart 和 onDragEnd 钩子函数来实时监听翻页前后的页号来判断是哪一种情况，分别计算需要预加载到视频缓冲区的新视频的索引，在翻页结束的那一刻将新视频载入，代码如下：

```
/* 开始拖拽轮播item */
function onDragStart(args:{index: number}) {

  startIndex = args.index;

}
/* 结束拖拽轮播item */
function onDragEnd(args:{index: number}) {


  /*
    视频翻页修改

  */

  if (args.index === startIndex + 1 || (startIndex === 2 && args.index === 0)) {
    // 向下翻页
    if (videoQueue.current <= videoQueue.queue.length - 1) {
      videoQueue.current = (videoQueue.current + 1 + videoQueue.queue.length) % videoQueue.queue.length;

      if(args.index === 0) {
        myPlayer1.value.poster(videoQueue.queue[videoQueue.current % videoQueue.queue.length].coverUrl)
        myPlayer2.value.poster(videoQueue.queue[(videoQueue.current + 1) % videoQueue.queue.length].coverUrl)
        myPlayer3.value.poster(videoQueue.queue[(videoQueue.current + 2) % videoQueue.queue.length].coverUrl)

        myPlayer1.value.src(videoQueue.queue[videoQueue.current % videoQueue.queue.length].videoUrl)
        myPlayer2.value.src(videoQueue.queue[(videoQueue.current + 1) % videoQueue.queue.length].videoUrl)
        myPlayer3.value.src(videoQueue.queue[(videoQueue.current + 2) % videoQueue.queue.length].videoUrl)
        setPlayerVideoInfo(1,videoQueue.current % videoQueue.queue.length)
        setPlayerVideoInfo(2,(videoQueue.current + 1) % videoQueue.queue.length)
        setPlayerVideoInfo(3,(videoQueue.current + 2) % videoQueue.queue.length)
      }
    }
  } else if (args.index === startIndex - 1 || (startIndex === 0 && args.index === 2)) {

    // 向上翻页
    if (videoQueue.current >= 0) {
      videoQueue.current = (videoQueue.current - 1 + videoQueue.queue.length) % videoQueue.queue.length;

      if(args.index === 2) {
        myPlayer1.value.poster(videoQueue.queue[(videoQueue.current -2) % videoQueue.queue.length].coverUrl)
        myPlayer2.value.poster(videoQueue.queue[(videoQueue.current-1) % videoQueue.queue.length].coverUrl)
        myPlayer3.value.poster(videoQueue.queue[(videoQueue.current) % videoQueue.queue.length].coverUrl)
        myPlayer1.value.src(videoQueue.queue[(videoQueue.current - 2) % videoQueue.queue.length].videoUrl)
        myPlayer2.value.src(videoQueue.queue[(videoQueue.current - 1) % videoQueue.queue.length].videoUrl)
        myPlayer3.value.src(videoQueue.queue[(videoQueue.current) % videoQueue.queue.length].videoUrl)
        setPlayerVideoInfo(1,(videoQueue.current - 2) % videoQueue.queue.length)
        setPlayerVideoInfo(2,(videoQueue.current - 1) % videoQueue.queue.length)
        setPlayerVideoInfo(3,(videoQueue.current) % videoQueue.queue.length)
      }
    }
  }

}
```

同时，为了方便用户切换视频，除了用鼠标拖拽的方式，还实现了键盘上下键监听、

鼠标滚轮监听、上下按钮点击监听来实现相同的翻页逻辑:

```
/*
    监听滚轮翻页

 */
handleWheel = (event:any) => {
  if (event.deltaY < 0) {
    // 向上滚动
    prevPage();

  } else {
    // 向下滚动
    nextPage();

  }
};
/*
  监听按键翻页

 */
handleKeydown = (event:any) => {
  if (event.key === 'ArrowUp') {
    // 按上键
    prevPage();
  } else if (event.key === 'ArrowDown') {
    // 按下键
    nextPage();

  }
};

window.addEventListener('wheel', handleWheel);
window.addEventListener('keydown', handleKeydown);
```

3.3.2.2 安全验证

安全验证主要分为两个部分，一个是用户信息的安全验证，一个是用户密码的安全加密。用户信息的安全认证采用 JWT，它具有安全、稳定、易用的特点，是目前最流行的跨域身份验证解决方案。当服务器验证用户身份后，会生生成一个 JSON 对象返回给客户端，客户端接收服务器返回的 JWT，将其存储在本地，存储过程如下所示:

```
TypeScript
if(res.code == 200) {
    ElMessage({ type: 'success', message: '登录成功! ' })
    localStorage.setItem('token', res.data.token)
```

```
    $router.push('/home')
    store.setToken(res.data.token)
    store.setUserInfo(res.data.userInfo)
}
```

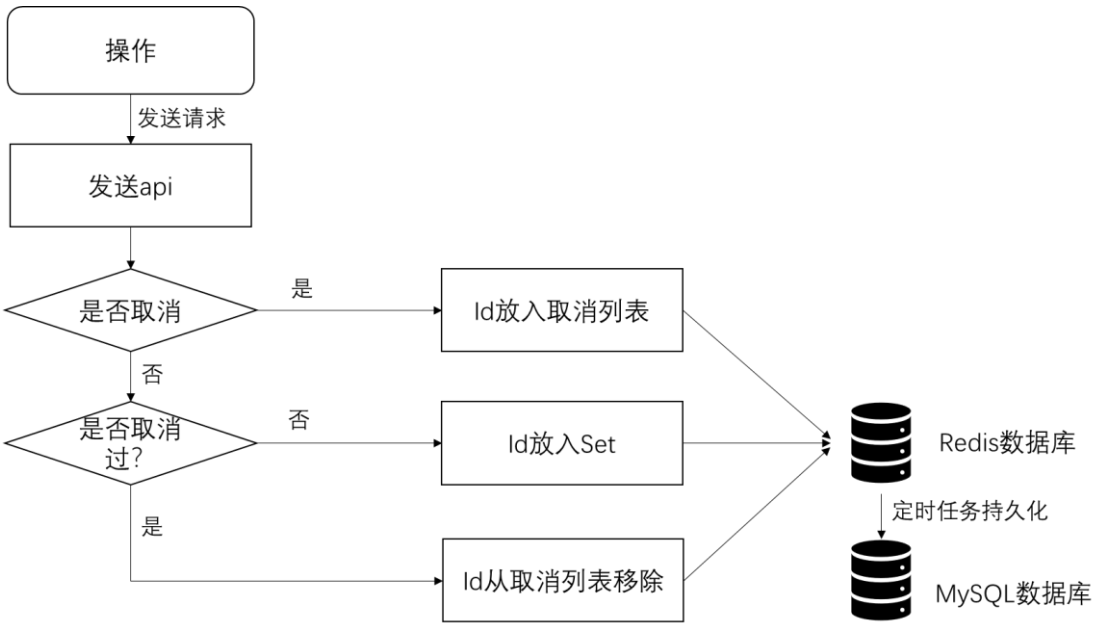
存储的位置以及之后调用的函数如下所示：

```TypeScript
export const loginStore = defineStore("login", {
  state: () => ({
    userInfo: null,
    token: null,
  }),
  actions: {
    setUserInfo(userInfo:any) {
      this.userInfo = userInfo;
    },
    setToken(token:any) {
      this.token = token;
    },
    clearUserInfo() {
      this.userInfo = null;
    },
    clearToken() {
      this.token = null;
    },
    existToken() {
      return this.token !== null;
    }
  },
  persist: {
    enabled: true,
    strategies: [
      {
        key: 'login',
        storage: localStorage,
        paths: ['userInfo', 'token']
      }
    ]
  },
},);
```

这样用户登录后，客户端就可以根据用户的 token 信息来确定登录状态，从而响应用

户相关的操作。

3.3.2.3 Redis 缓存



由于用户与视频的交互信息具有实时性，因此用户的点赞、收藏、关注行为首先缓存到 Redis 数据库中，再通过定时任务持久化回 MySQL 数据库。在 springboot 中，Redis 相关服务通过 `redisTemplate` 组件与 Redis 数据库进行交互，对于操作的用户，其操作信息将以 Set 数据结构进行存储，key 为包含用户 Id 的特殊 key，value 维护一个 Set 数据结构，Set 中的每一项为被操作视频的 Id。但是仅仅这样的设计这会出现一些问题，以用户点赞为例：

1. 用户点击爱心的操作都调用后端点赞接口，因此后端无法判断当前是点赞还是取消点赞操作。

2. `redisTemplate` 组件只有通过 `add` 和 `remove` 操作增删数据。

因此在维护用户点赞信息 Set 的同时，我们在 Redis 中还同时维护了一个 cancel 列表，用于存放用户取消点赞的视频 Id。这样一个点赞操作的逻辑就可以设计如下：

1. 用户点赞或者取消点赞。

2. 判断用户是否取消点赞，即 cancel 列表为空且 Set 中有点赞视频 Id，重复点赞则取消点赞，将当前取消点赞视频的 Id 放入 cancel 列表中。如果 Set 中没有点赞视频 Id，或者有点赞视频 Id 但同时 cancel 列表中也有该视频 Id，说明用户执行点赞操作，前者向 Set 中加入该视频 Id，后者直接将视频 Id 从 cancel 列表中移出。

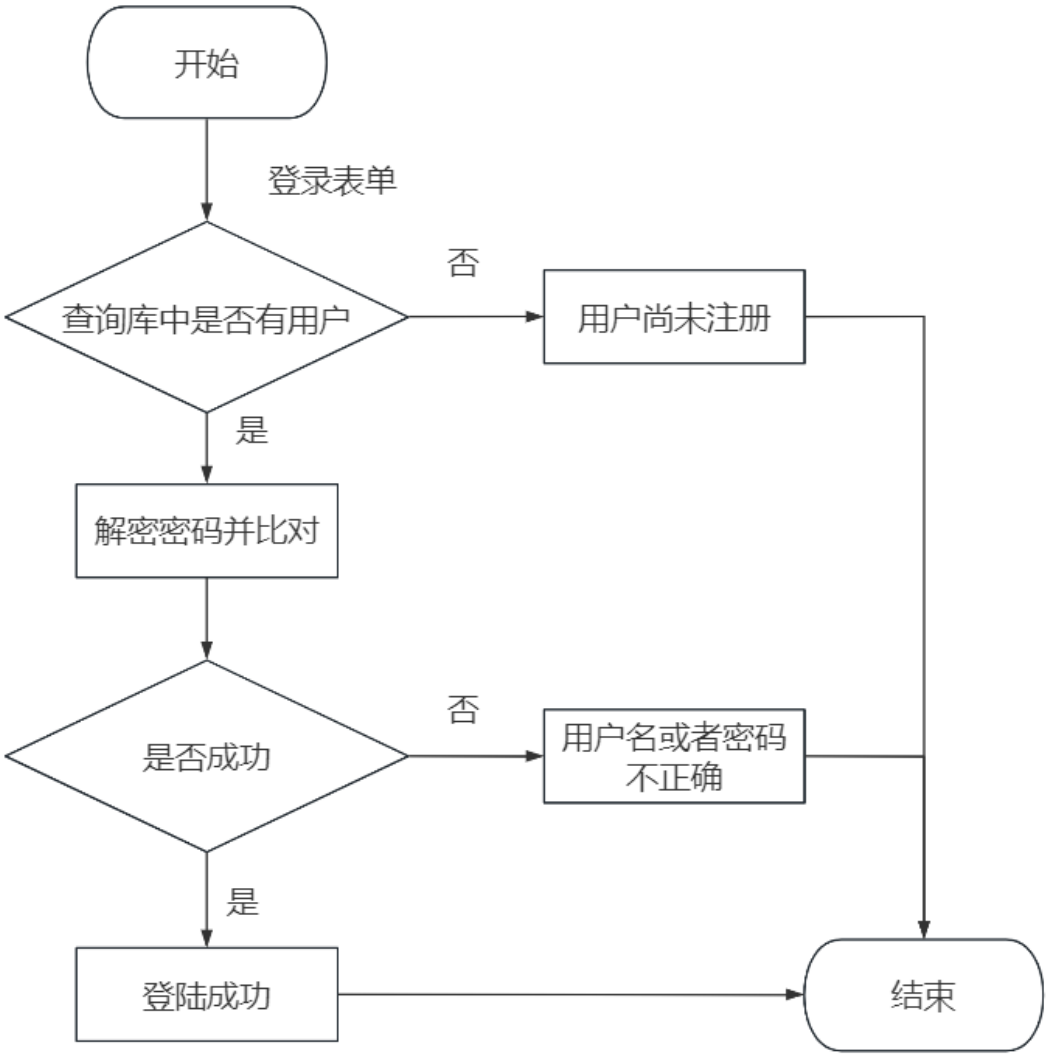
3. 到达定时任务执行时间，将 Redis 中的 Set 数据持久化回 MySQL 中，如果 cancel 列表中有相关视频 Id，则该视频 Id 不持久化回 MySQL 中，持久化结束清空 redis。

整体流程如上图所示，下面以点赞操作为例，展示点赞或者取消点赞的部分后端代码：

```
Java
@Override
public void saveLike(String userId, String postId, String type) {
    String result = "";
    String key, cancel_key;
    key = RedisLikeKeyUtil.USER_LIKE_KEY + type + "::" + userId;
    cancel_key = RedisLikeKeyUtil.CANCEL_KEY + type + "::" +
userId;
    long object = redisTemplate.opsForSet().add(key, postId);
    boolean flag = redisTemplate.opsForSet().isMember(cancel_key,
postId);
    if(object != 1) {
        if(!flag) {
            result = "取消点赞" + type;
            // 视频 id 加入取消点赞列表
            cancelLike(userId, postId, type);
            decreaseLikeCount(postId, type);
        } else {
            result = "点赞成功！" + type+ postId;
            redisTemplate.opsForSet().remove(cancel_key, postId);
            increaseLikeCount(postId, type);
        }
    } else {
        result = "点赞成功！" + type+ postId;
        increaseLikeCount(postId, type);
    }
    System.out.println(result);
}
```

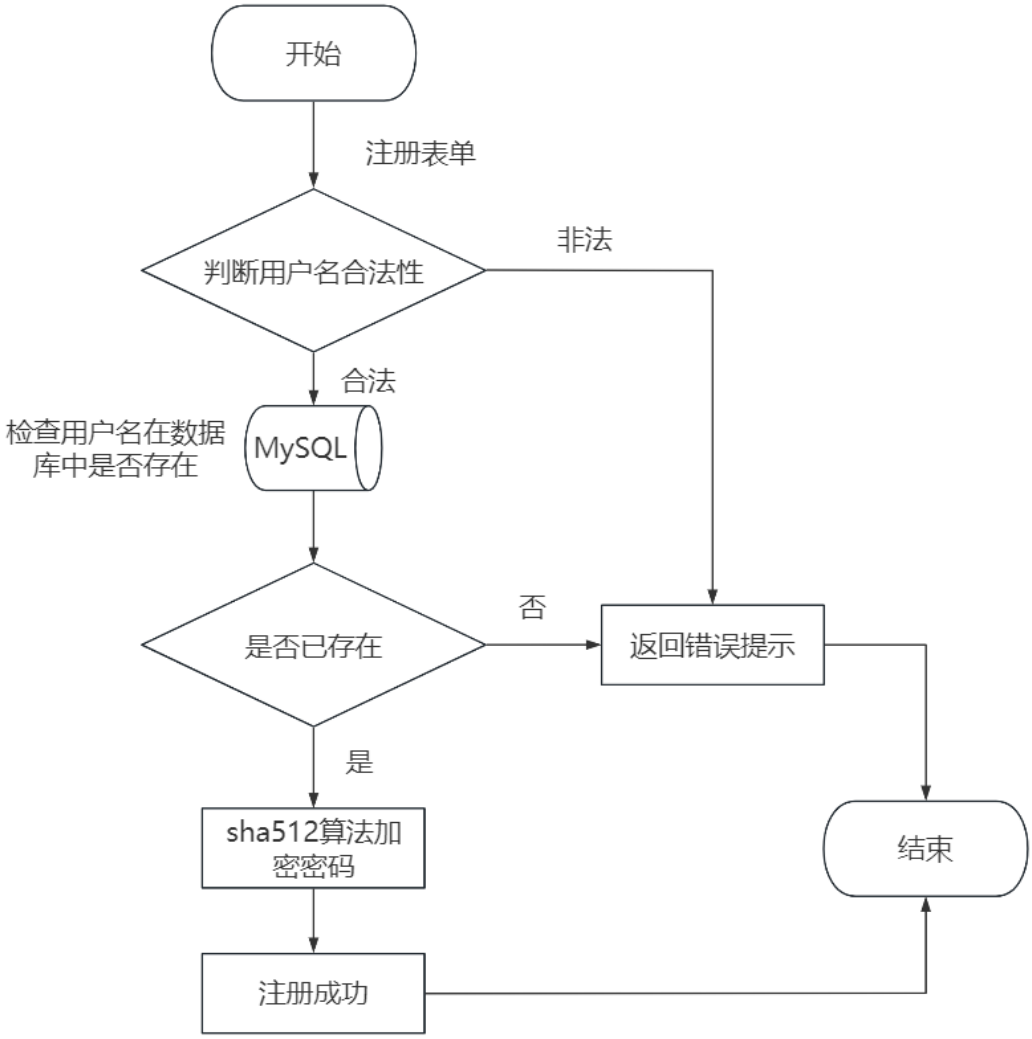
3.3.3 业务流程

3.3.3.1 登录



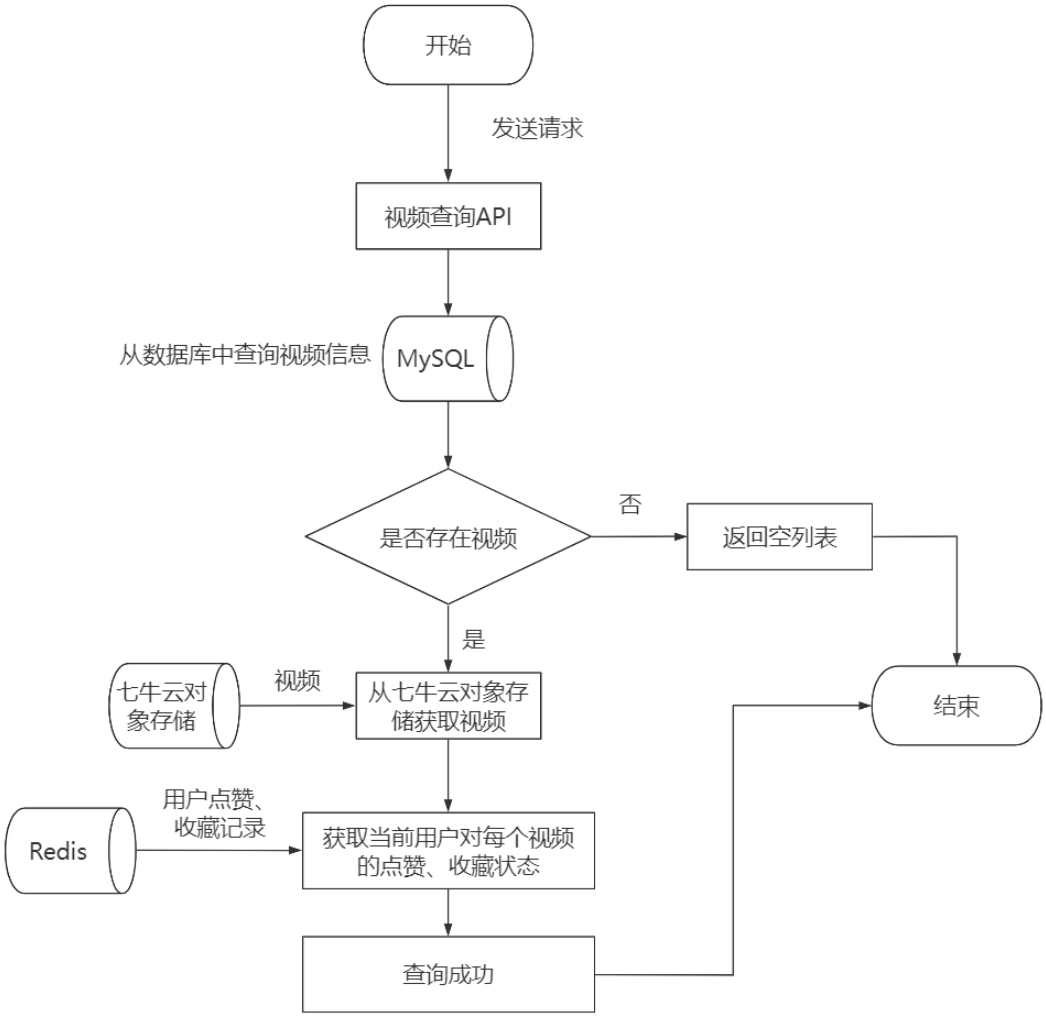
- 通过查询数据库中是否存在用户没有用户就返回。
- 有用户信息就比对密码。
- 登录成功向返回 token 和用户信息。

3.3.3.2 注册



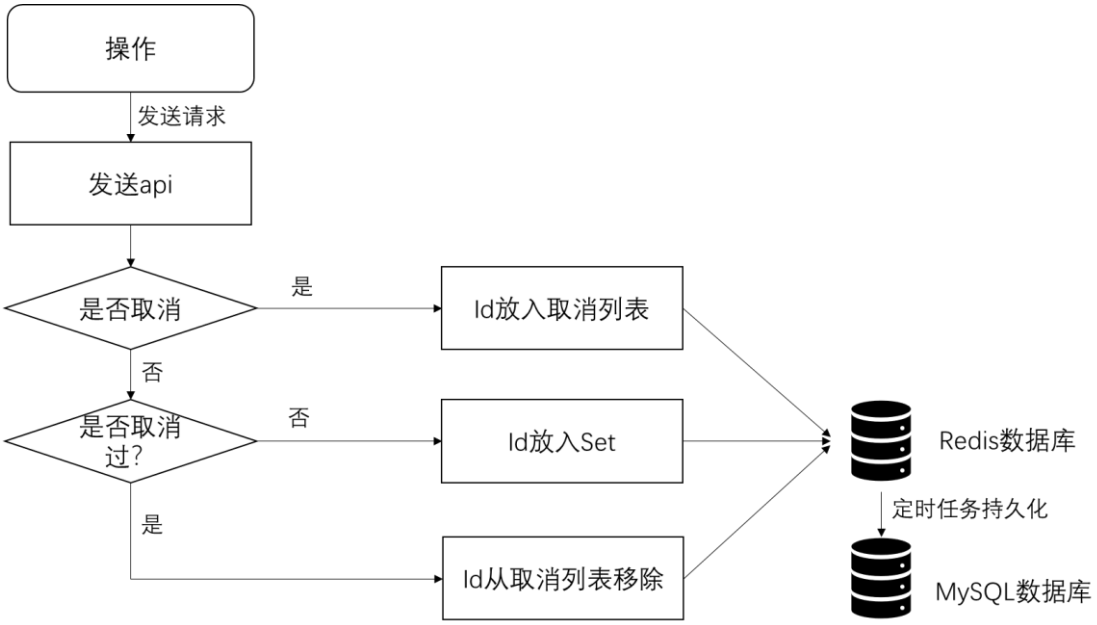
- 用户注册需要用正则判断是否合法。
- 查询数据库是否存在相同的用户名。
- 密码需要加密。

3.3.3.3 视频流



- 从七牛云获取视频 url。
- 分别查询 mysql 和 redis 后，进行封装，需要及时获取用户对每个视频的点赞、收藏等状态。

3.3.3.4 用户操作



- Redis 缓存用户操作。
- 定时任务持久化操作。

3.3.4 存储设计

在本项目中，用户的个人信息存储在 MySQL 数据库中，用户与视频的交互信息先缓存在 Redis 数据库上，通过定时任务持久化回 MySQL 数据库。由于用户的评论数据以 JSON 数据格式存储更能满足父评论-子评论的设计要求，因此用户的评论数据存储在 MongoDB 数据库中。视频和图片等静态资源文件存储在七牛云对象存储服务器中。

3.3.4.1 Redis 设计

当用户登录时，服务器会自动将用户的交互数据从 MySQL 数据库中导入到 Redis 数据库，以便及时更新用户的操作。当用户执行交互操作，包括点赞、收藏、关注，都会首先缓存到 Redis 中，此外，被交互的视频点赞、收藏信息也会缓存到 Redis 中。最后，Redis 中的数据通过定时任务持久化回 MySQL 数据库。下面是 Redis 中存在的部分数据结构。

用户点赞缓存数据

Key	Value 类型	说明
USER_LIKE_KEY	Set	用户点赞列表
CANCEL_KEY	Set	用户取消点赞列表

POST_LIKE_NUM_KEY	Long	视频点赞个数

用户收藏缓存数据

Key	Value 类型	说明
USER_COLLECT_KEY	Set	用户收藏列表
CANCEL_KEY	Set	用户取消收藏列表
POST_COLLECT_NUM_KEY	Long	视频收藏个数

用户关注缓存数据

Key	Value 类型	说明
USER_FOLLOW_KEY	Set	用户关注列表
CANCEL_KEY	Set	用户取消关注列表
POST_FOLLOW_NUM_KEY	Long	用户被关注数

3.3.4.2 MySQL 设计

MySQL 数据库中表项的设计基于 Java Persistence API 规范，在 springboot 的 entity 中，通过注解来设计表项，包括主键设计、数据类型、外键的连接等。`common` 目录下的 `TableColumns.java` 对每个表及其表项的命名进行了规范化设计。MySQL 中主要包括 User、UserCollect、UserFollow、UserLike、Video 等数据表。下面将通过表格对各个数据表进行说明。

表名	**功能说明**
User	用户信息
UserCollect	用户收藏信息
UserFollow	用户关注信息

UserLike	用户点赞信息
Video	视频信息

用户信息表

编号	名称	描述	数据类型	大小	备注
1	user_id	用户 Id	bigint	20	主键，自增
2	username	用户名	varchar	255	
3	nickname	昵称	varchar	255	
4	phone	电话	varchar	255	
5	password	密码	varchar	255	
6	email	邮箱	varchar	255	
7	age	年龄	varchar	255	
8	follows	关注数	bigint	20	
9	gender	性别	enum		枚举 female、male、unknow
10	avatar	头像	longtext		用户头像的 url
11	createTime	创建时间	Date		
12	updateTime	更新时间	Date		

视频信息表

编号	名称	描述	数据类型	大小	备注

1	video_id	视频 Id	varchar	255	主键
2	video_path	视频路径	longtext		视频地址的 url
3	cover_url	封面路径	longtext		视频封面的 url
4	title	标题	longtext		
5	description	视频描述	longtext		
6	category	类别	bigint	20	
7	likes	点赞数	bigint	20	
8	collections	收藏数	bigint	20	
9	comments	评论数	enum		
10	createTime	创建时间	Date		
11	updateTime	更新时间	Date		
12	useId	用户 Id	bigint	20	外键，连接用户 Id

用户点赞表

编号	名称	描述	数据类型	大小	备注
1	id	id	bigint	20	主键，自增，表项 id
2	user_id	用户名	varchar	255	
3	post_id	点赞对象 Id	varchar	255	点赞对象包括视频、评论

4	status	状态	varchar	255	是否点赞
5	type	类别	varchar	255	用于判断对象是视频还是评论
6	video_id	视频 Id	varchar	255	当前点赞视频的 Id
7	createTime	创建时间	Date		
8	updateTime	更新时间	Date		

用户收藏表

编号	名称	描述	数据类型	大小	备注
1	id	id	bigint	20	主键，自增，表项 id
2	user_id	用户名	varchar	255	
3	post_id	收藏对象 Id	varchar	255	收藏对象只有视频
4	status	状态	varchar	255	是否收藏
5	createTime	创建时间	Date		
6	updateTime	更新时间	Date		

用户关注表

编号	名称	描述	数据类型	大小	备注
1	id	id	bigint	20	主键，自增，表项 id
2	user_id	用户名	varchar	255	

3	post_id	关注对象 Id	varchar	255	
4	status	状态	varchar	255	是否关注
5	createTime	创建时间	Date		
6	updateTime	更新时间	Date		

3.3.4.3 MongoDB 设计

MongoDB 数据库中表项的设计基于 Java Persistence API 规范，在 springboot 的 entity 中，通过注解来设计表项，包括主键设计、数据类型等。`common` 目录下的 `TableColumns.java` 对每个表及其表项的命名进行了规范化设计。

对于评论功能，存在着复杂的嵌套结构（评论嵌套回复，回复又拥有子回复），并且每个评论除了要保存内容等信息，还要相应的保存作者信息、评论的点赞、回复数据等，故不适合用 MySQL 这类关系型数据库进行存储，而是使用半结构化数据库 MongoDB，利用 JSON 数据格式对评论数据进行存储。用户评论表结构如下：

用户评论表

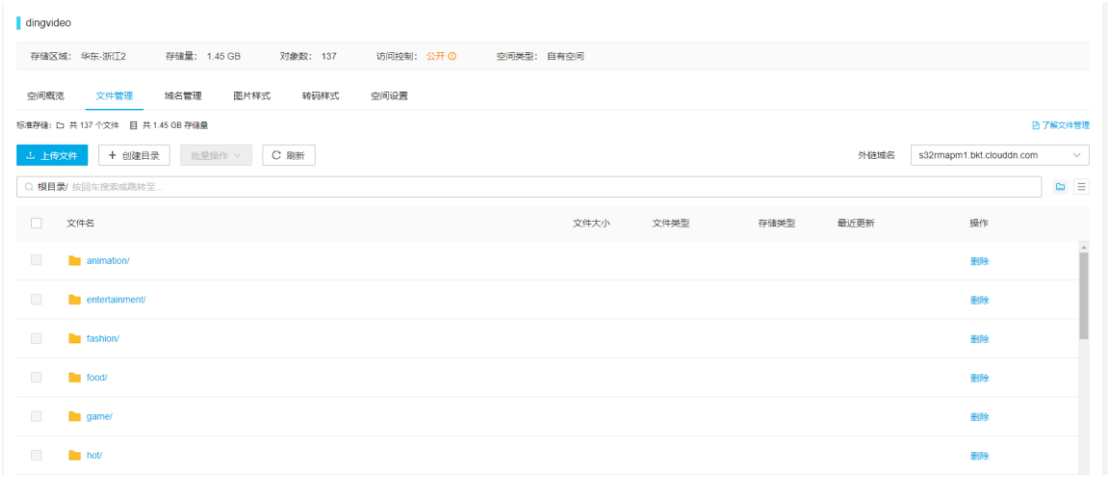
编号	名称	描述	数据类型	大小	备注
1	comment_id	评论 id	ObjectId		主键
2	parent_id	父回复 id	varchar	255	
3	type	回复类型	varchar	255	视频或者图文
4	resource_id	回复资源 Id	varchar	255	对应回复视频或图文的 Id
5	content	评论内容	varchar	255	
6	author	作者信息	varchar	255	
7	from	评论作者	varchar	255	

		昵称			
8	to	被评论用户昵称	varchar	255	
9	createTime	创建时间	Date		
10	updateTime	更新时间	Date		
11	like_cnt	点赞数	bigint	20	
12	reply_cnt	回复数	bigint	20	
13	status	状态	varchar	255	是否二级回复（0：否，1：是）
14	replies	回复	List<Comment>		当前评论下的回复列表

3.3.4.4 七牛云对象存储

七牛云对象存储具有高可靠、易扩展、低成本、数据智能化、存储加速和边缘计算等特点，拥有业界领先的纠删码存储方案，能够提供高达 11 个 9 的数据可靠性。跨数据中心的副本冗余，能够保障服务的超高可用性。此外，七牛云对象存储无需担心扩容问题，可以实现存储弹性伸缩，按需使用、按需付费。

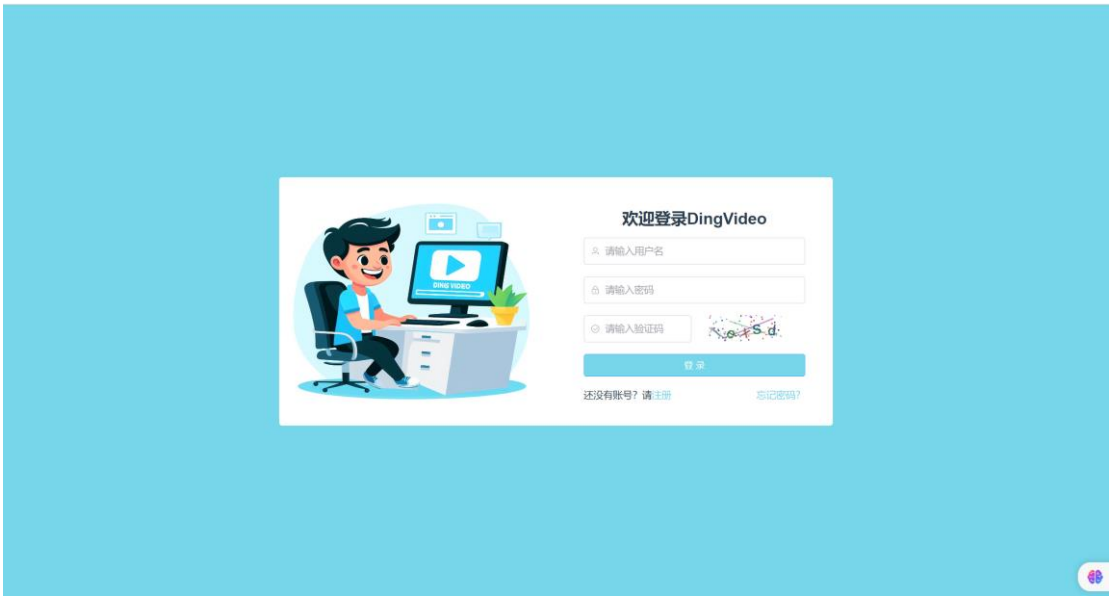
使用上，只需要新建空间，并根据需求建立转码样式，在文件管理模块上传文件即可，也可以通过 API 接口进行文件上传。此外还可以进行域名管理，设计图片样式和转码样式等，七牛云官网的管理界面也让操作更为便捷。



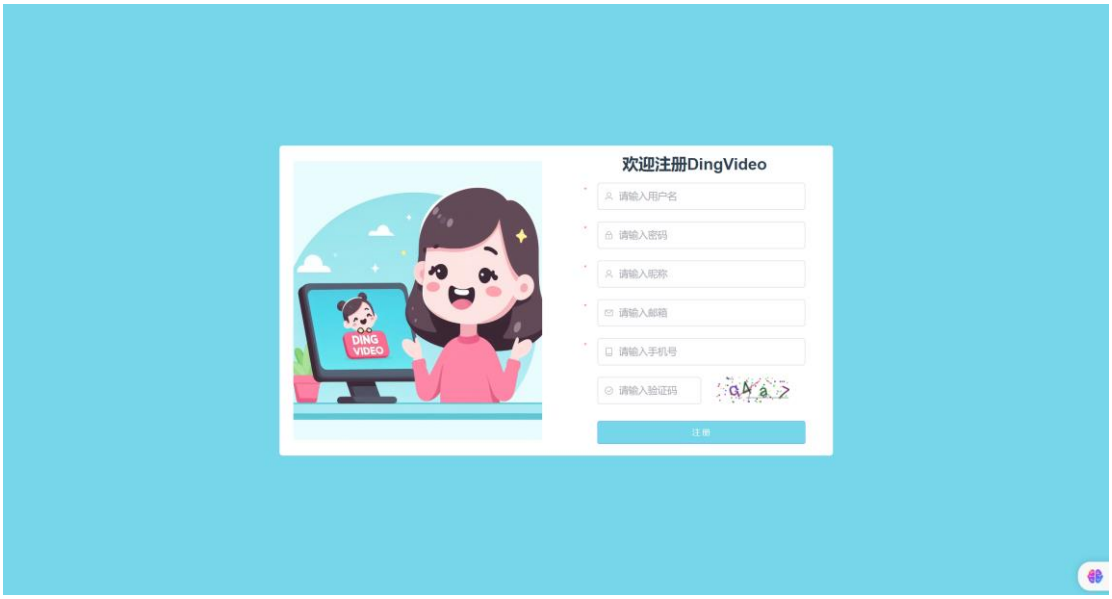
4. 功能测试

4.1 页面展示

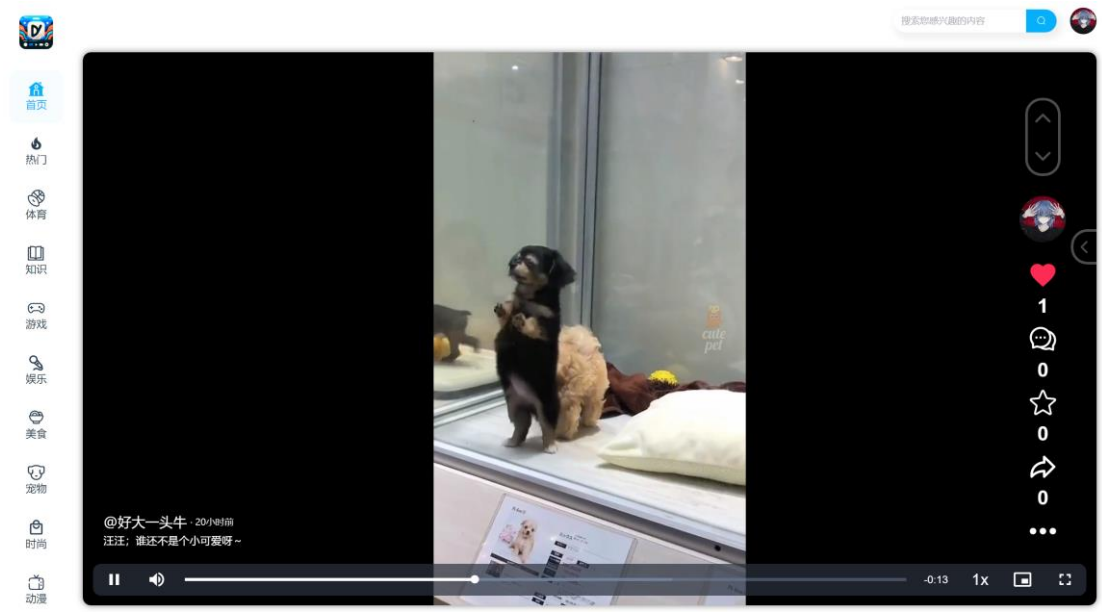
4.1.1 登录页面



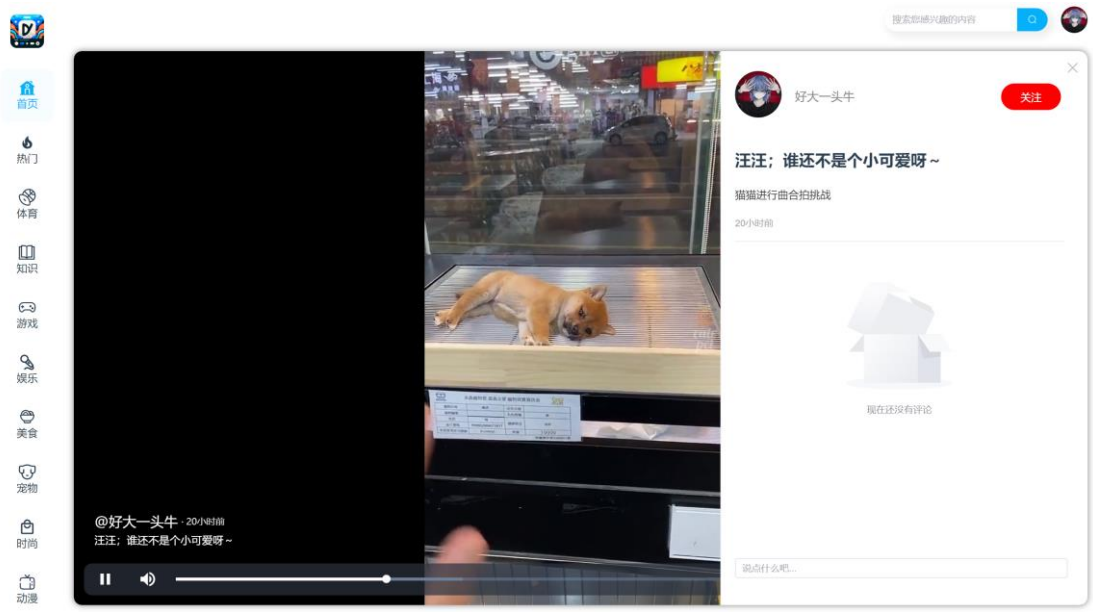
4.1.2 注册页面



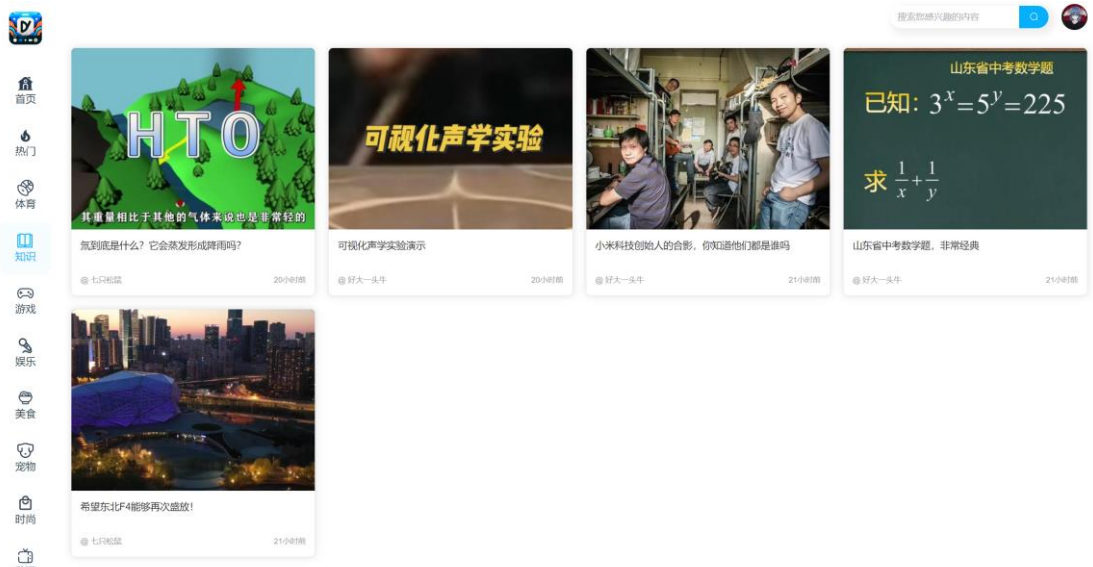
4.1.3 首页推荐



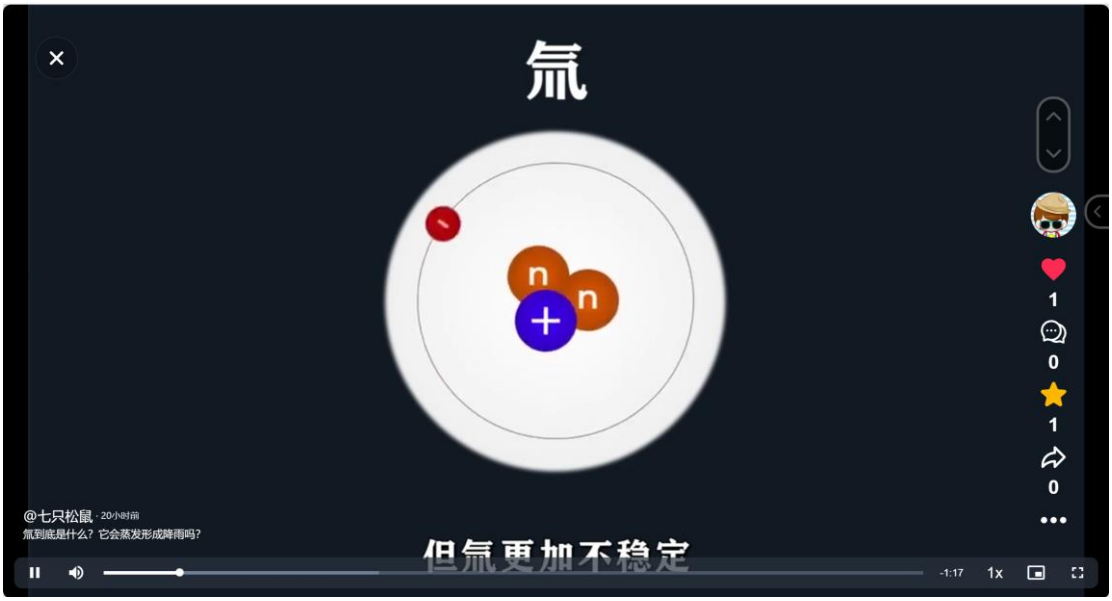
4.1.4 视频描述与评论



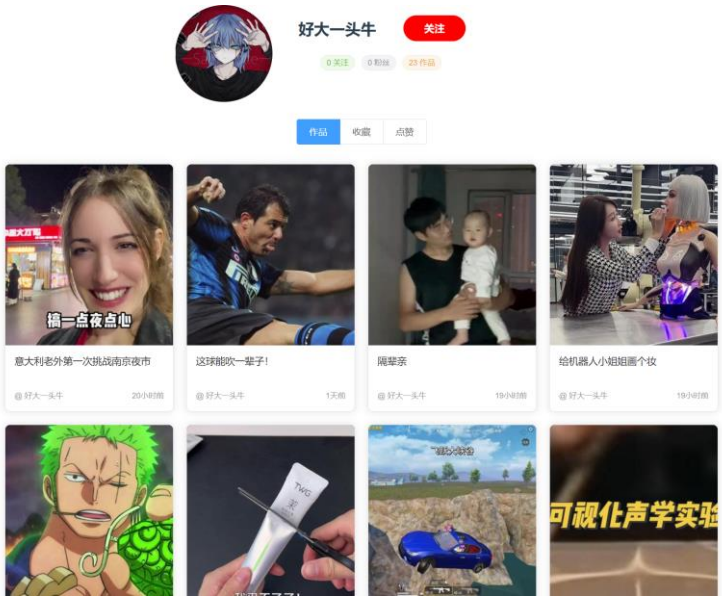
4.1.5 视频分类页面



4.1.6 视频播放

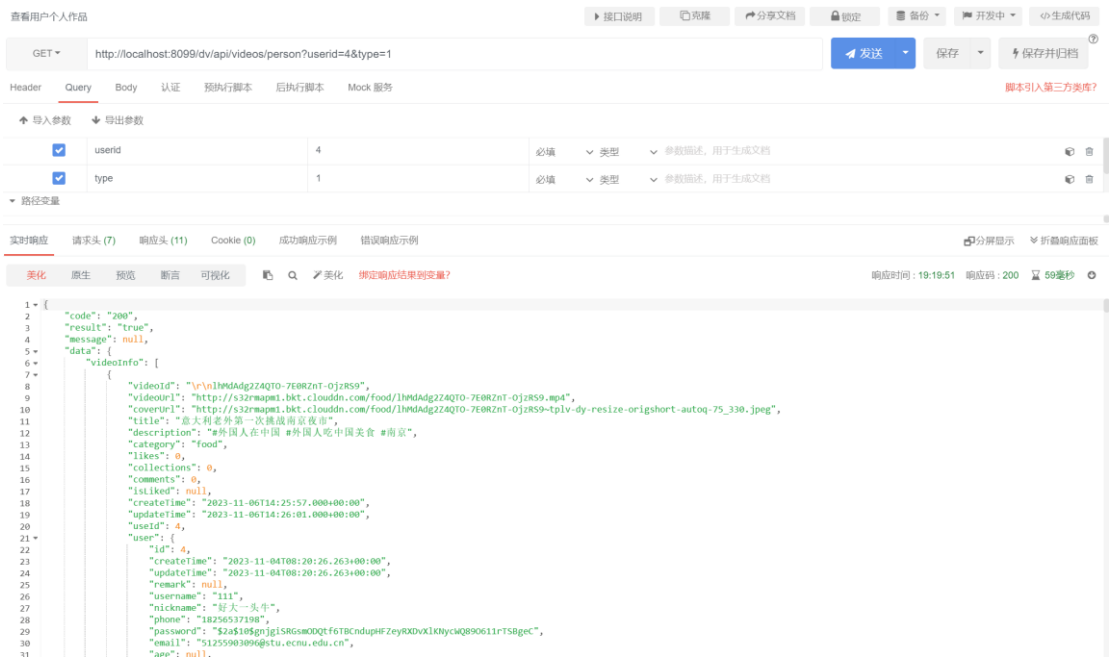


4.1.7 个人中心

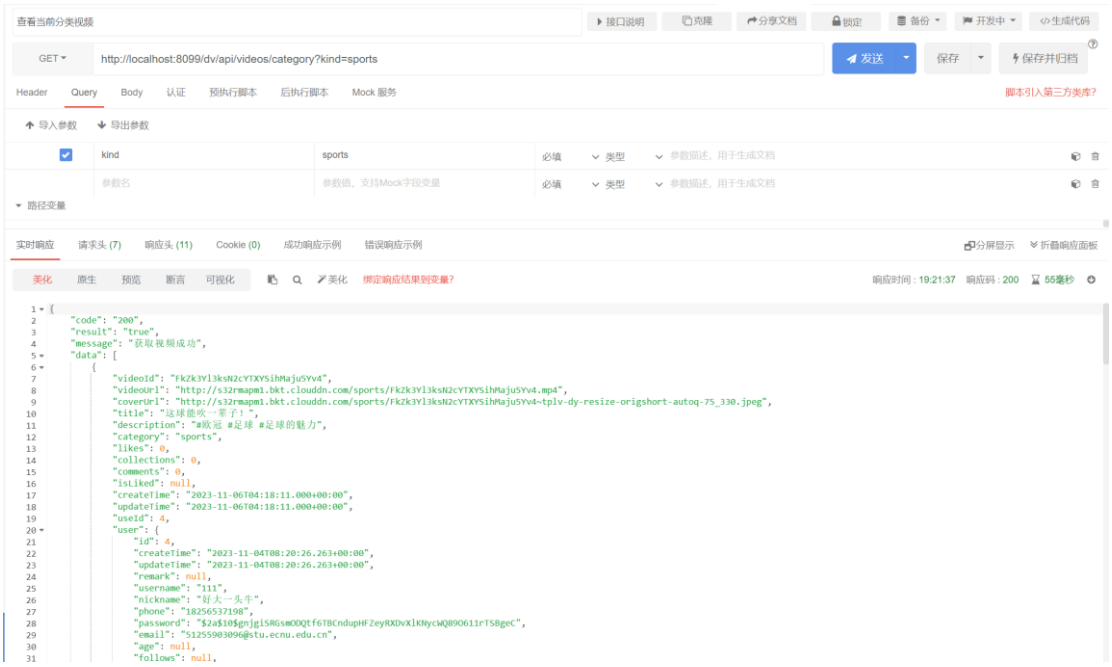


4.2 接口测试

4.2.1 查看用户个人作品接口测试

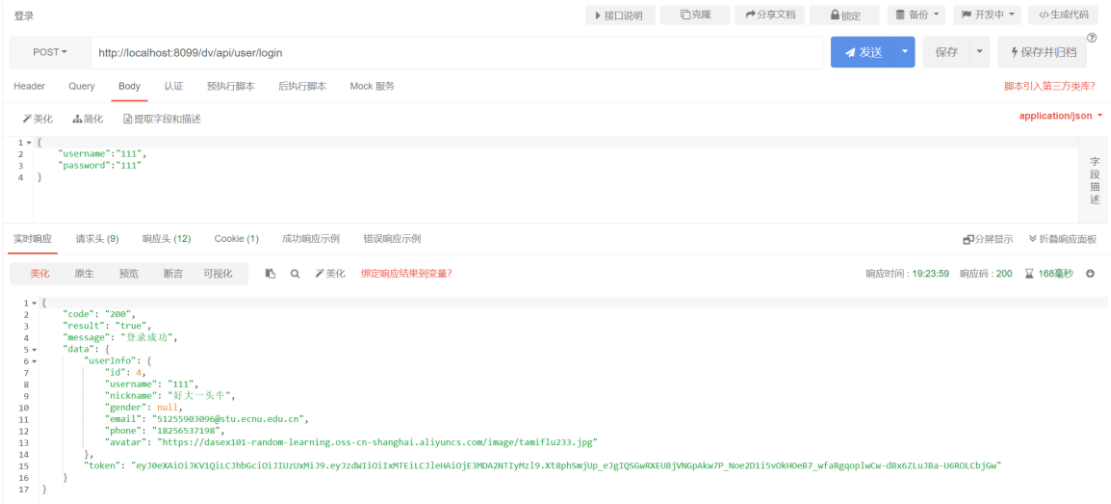


4.2.2 查看当前分类视频接口测试

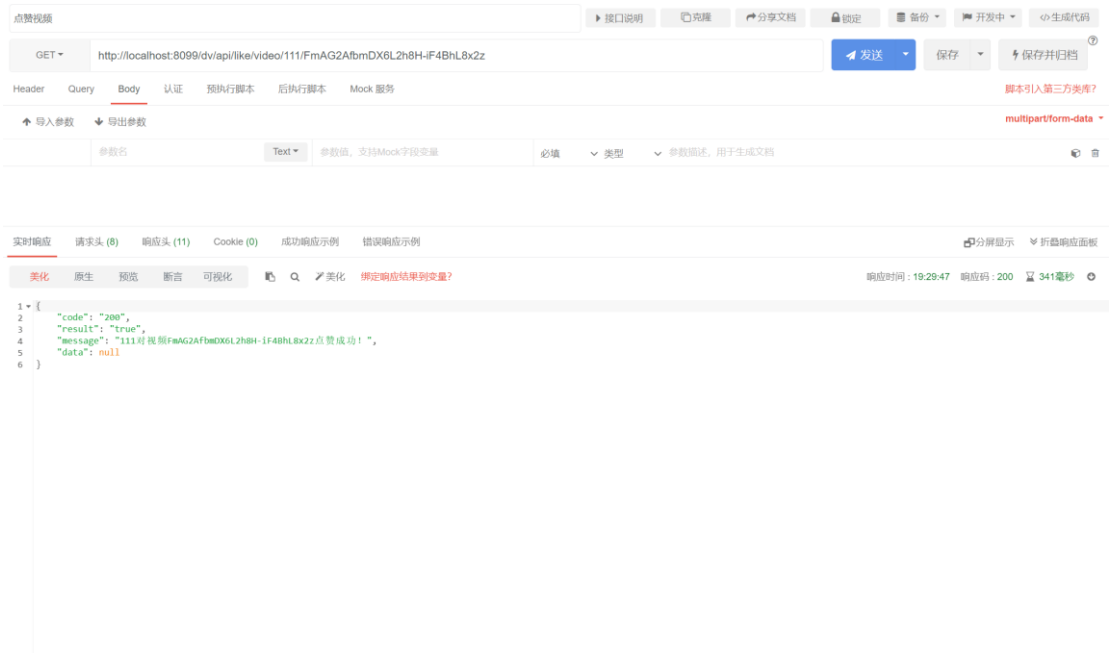


4.2.3 查看当前用户信息接口测试

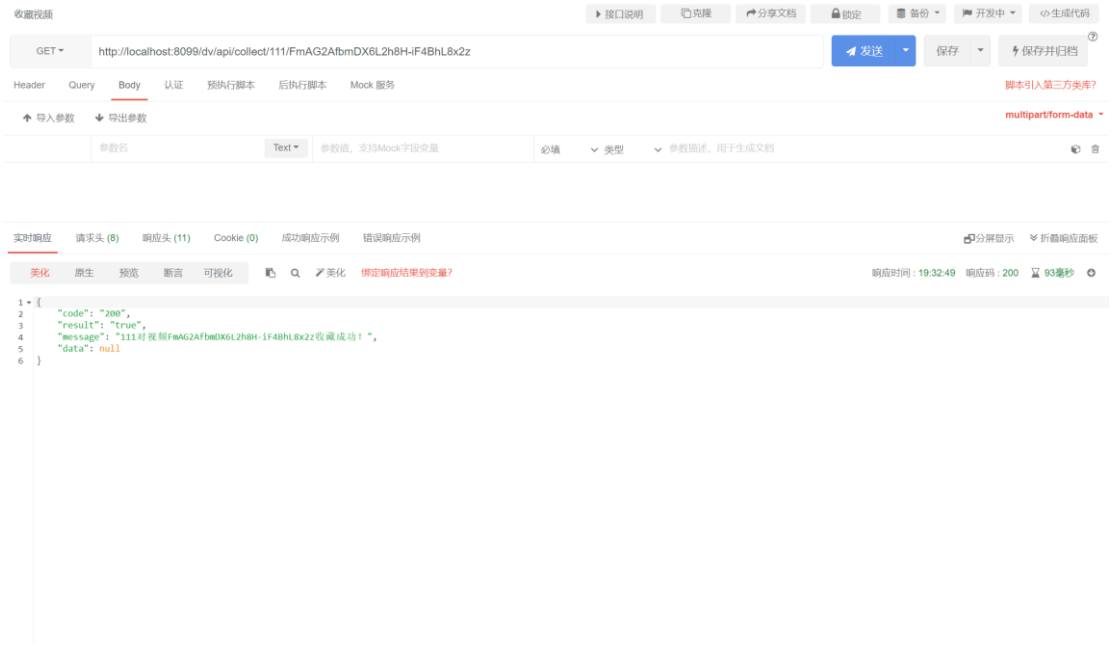
4.2.4 登录接口测试



4.2.5 点赞接口测试



4.2.6 收藏接口测试



5. 演示视频

视频地址

http://s32rmapm1.bkt.clouddn.com/DingVide%E6%BC%94%E7%A4%BA%E8%A7%86%E9%A2%91.mp4