

## 摘要

本项目基于 HADOOP 分布式文件系统（HDFS），设计并实现了一个简易的 Web 文件管理系统。系统采用前后端分离架构：后端基于 Spring Boot 框架实现文件上传、下载、删除、元数据管理与 HDFS 交互功能，前端通过 Vue.js 构建用户友好的可视化界面。项目整合 Hadoop HDFS 的高可靠性存储能力与 Spring Boot 的高效 RESTful API 服务，同时利用 Vue.js 的动态组件实现实时文件操作反馈。通过模块化设计与分层解耦，系统兼具扩展性与维护性。

**关键词：**Hadoop HDFS, 分布式文件系统, Spring Boot, Vue.js, 文件管理, 模块化设计, 高可靠性存储



## ABSTRACT

This project designs and implements a lightweight web-based file management system based on the Hadoop Distributed File System (HDFS). Adopting a frontend-backend decoupled architecture, the backend utilizes the Spring Boot framework to handle file upload, download, deletion, metadata management, and HDFS interactions, while the Vue.js frontend delivers a user-friendly visual interface. The system integrates Hadoop HDFS's high-reliability storage capabilities with Spring Boot's efficient RESTful API services, employing Vue.js dynamic components to achieve real-time feedback for file operations. Through modular design and layered decoupling, the system ensures both extensibility and maintainability.

**Keywords:** Hadoop HDFS, Distributed File System, Spring Boot, Vue.js, File Management, Metadata Management, Modular Design, High-Reliability Storage



## 目 录

<b>第一章 绪 论 .....</b>	<b>1</b>
1.1 Hadoop 的背景与意义 .....	1
1.2 个人工作简述 .....	1
1.2.1 Hadoop 集群的搭建 .....	1
1.2.2 Springboot 后端的实现 .....	1
1.2.3 Vue 前端的实现 .....	1
1.3 开发系统版本与工具 .....	1
<b>第二章 系统分析 .....</b>	<b>2</b>
2.1 功能分析 .....	2
2.1.1 基本功能 .....	2
2.1.2 拓展功能 .....	2
2.2 各模块关系分析 .....	2
2.2.1 hadoop 集群原理及关系 .....	2
2.2.2 SpringBoot 关系 .....	7
2.2.3 Vue 关系 .....	7
<b>第三章 详细设计及实现 .....</b>	<b>9</b>
3.1 hadoop 集群搭建 .....	9
3.1.1 配置虚拟机 .....	9
3.1.2 设备配置 .....	9
3.1.3 Hadoop 集群启动与使用 .....	18
3.2 后端服务器搭建 .....	20
3.2.1 环境搭建 .....	20
3.2.2 用户登录注册 .....	23
3.2.3 文章功能 .....	26
3.2.4 文件后端 .....	29
3.2.5 WordCount 作业 .....	37
3.3 前端搭建 .....	37
3.3.1 前端框架 .....	37
3.3.2 拦截器与响应器 .....	38
3.3.3 登录注册页面 .....	40

3.3.4 文章管理页面 .....	43
3.3.5 文件页面 .....	45
<b>第四章 测试.....</b>	<b>49</b>
4.1 注册登录测试 .....	49
4.1.1 拦截器测试 .....	49
4.1.2 注册测试.....	49
4.1.3 登录测试.....	50
4.1.4 用户信息更改 .....	50
4.2 文章与文件区测试 .....	51
4.3 小结.....	55
<b>参考文献 .....</b>	<b>56</b>

# 第一章 绪论

## 1.1 Hadoop 的背景与意义

Hadoop 是一个开源的大数据处理框架，其核心包括分布式存储系统 HDFS 和资源调度系统 YARN。HDFS 实现了海量数据的高可靠性分布式存储，支持数据冗余和容错；YARN 负责集群资源的统一管理和作业调度，提升了系统的扩展性和资源利用率。Hadoop 为大数据存储与分析提供了高效、可扩展的基础平台。

## 1.2 个人工作简述

### 1.2.1 Hadoop 集群的搭建

实现了一个基于 Hadoop 的多机集群，配置了 NameNode 和 DataNode 节点。搭建 HDFS 文件系统，完成了数据的分布式存储和管理。

### 1.2.2 Springboot 后端的实现

实现了一个基于 Spring Boot 的 RESTful API，提供文件上传、下载、删除、预览等功能以及文章上传、编辑、删除等功能，并利用 Postman 进行接口测试。

### 1.2.3 Vue 前端的实现

实现了一个基于 Vue 的前端界面，并利用 element 等组件为用户提供友好的交互界面。

## 1.3 开发系统版本与工具

- VMware 版本：VMware Workstation 17 Pro
- 虚拟机操作系统：CentOS Linux release 7.7.1908 (Core)\*3
- hadoop 版本: 3.3.0
- IDE: IntelliJ IDEA 2024.3.5
- 服务器 Java 版本: Oracle JDK 21.0.2
- 服务器操作系统: windows 11
- 数据库版本: mysql Ver 8.0.36
- Spring Boot 版本: 3.3.4
- Vue 版本: 3.3.4
- Xshell: 8.0.0

## 第二章 系统分析

### 2.1 功能分析

#### 2.1.1 基本功能

- 文件上传：支持文件上传，支持多种文件格式。
- 文件下载：支持文件下载。
- 文件删除：支持文件删除，提供确认提示。
- hadoop 集群搭建，利用 HDFS 实现文件的分布式存储。

#### 2.1.2 拓展功能

- 文件预览：支持图片、word、pdf 等文件的在线预览。
- 文章管理：支持文章的上传、编辑、删除等功能。
- 用户管理：支持用户注册、登录、权限管理等功能。
- 文章分类：支持文章的分类管理，方便用户查找。
- 搜索功能：支持文件和文章的搜索功能，方便用户快速找到所需内容。
- hadoop 多节点集群：提升系统的性能和可靠性。

### 2.2 各模块关系分析

#### 2.2.1 hadoop 集群原理及关系

##### 2.2.1.1 Hadoop 的组成：[1]

- HDFS：高可靠，高吞吐的分布式文件系统
- YARN：作业调度与集群资源管理框架
- MapReduce：分布式离线并行计算框架
- Common：支持其他模块的工具模块 [2]
  - 安全与权限管理
    - \* Apache Ranger
  - 元数据管理
    - \* Apache Atlas
  - 协调与管理
    - \* Zookeeper
    - \* Ambari

如下图2-1为 Hadoop 生态圈。

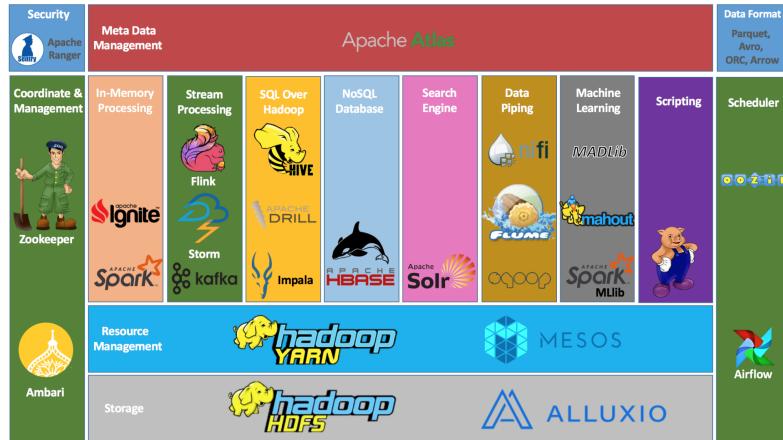


图 2-1 hadoop 生态圈 [3]

HDFS、YARN 和 MapReduce 之间的关系非常密切。在 Hadoop 生态系统中，它们协同工作以提供大数据处理能力。首先，用户或应用程序将数据存储在 HDFS 中，然后使用 MapReduce 来处理这些数据。YARN 负责管理和调度在集群中运行的 MapReduce 作业。

### 2.2.1.2 HDFS 简述

HDFS 主要是解决大数据如何存储问题的。分布式意味着是 HDFS 是横跨在多台计算机上的存储系统。HDFS 是一种能够在普通硬件上运行的分布式文件系统，它是高度容错的，适应于具有大数据集的应用程序，它非常适合存储大型数据（比如 TB 和 PB）。HDFS 使用多台计算机存储文件，并且提供统一的访问接口，使用户可以像访问一个普通文件系统一样使用分布式文件系统。

- BLOCK

物理磁盘中有块 (Block) 的概念，磁盘的物理 Block 是磁盘操作最小的单元。文件系统在物理 Block 之上抽象了另一层概念，文件系统 Block 物理磁盘 Block 的整数倍。HDFS 的文件被拆分成 block-sized 的 chunk，chunk 作为独立单元存储。

- Namenode 与 Datanode

图2-2为 HDFS 的工作简图。

NameNode 主要用于维护文件系统命名空间（文件/目录树），管理元数据（Block 位置、副本数等），不存储实际数据。而 DataNode 主要用于实际存储数据 Block，处理读写请求（来自客户端或 NameNode），定期向 NameNode

汇报 Block 列表。还有 Secondary NameNode 用来监控 HDFS 状态的辅助后台程序，每隔一段时间获取 HDFS 元数据的快照。最主要作用是辅助 NameNode 管理元数据信息。

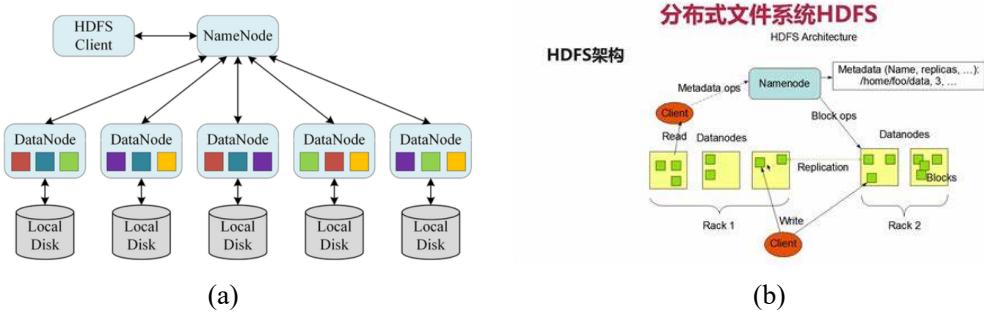


图 2-2 HDFS 工作简图

- 副本机制

为了容错，文件的所有 block 都会有副本。每个文件的 block 大小和副本系数都是可配置的。应用程序可以指定某个文件的副本数目。副本系数可以在文件创建的时候指定，也可以在之后改变。

### 2.2.1.3 Yarn 原理简述

是一个通用的资源管理系统和调度平台。包括批处理、交互式查询、流处理以及其他类型的工作负载。

- 核心组件在 Hadoop 集群中，YARN 主要有以下几个核心组件，如图2-3：[4]
  - ResourceManager（资源管理器）：ResourceManager 是 YARN 集群中的主节点，负责管理整个集群的资源分配和作业调度。它跟踪可用资源，并为提交到集群的应用程序分配资源。
  - NodeManager（节点管理器）：NodeManager 运行在每个集群节点上，负责管理该节点上的资源，并与 ResourceManager 通信以报告节点的资源使用情况和可用性。
  - ApplicationMaster（应用程序管理器）：每个提交到 YARN 集群的应用程序都有一个对应的 ApplicationMaster。ApplicationMaster 负责与 ResourceManager 协商资源、监控作业进度，并向 ResourceManager 请求更多资源或报告作业完成情况。

- 工作流程如图2-4

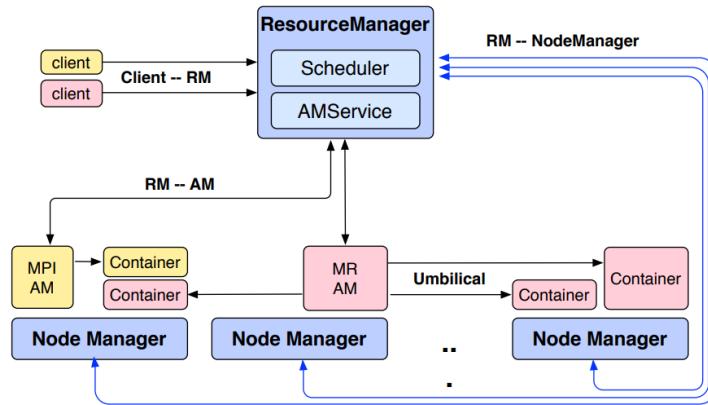


图 2-3 Yarn 架构 [5]

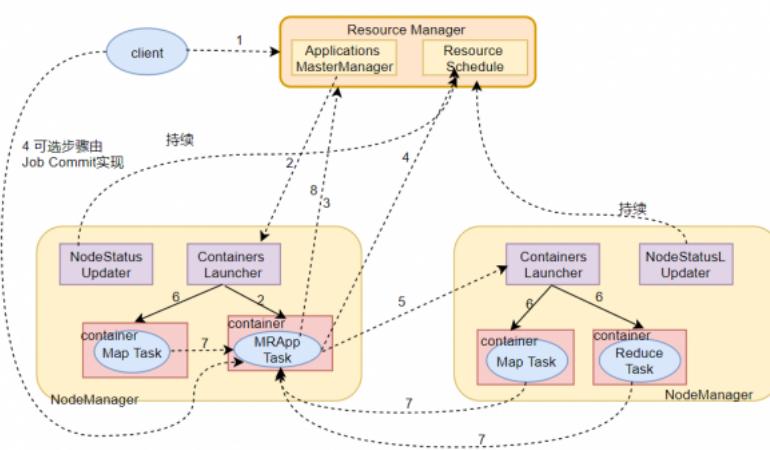


图 2-4 Yarn 运行流程 [6]

### 2.2.1.4 MapReduce 简述

核心思想：分而治之。将大数据集拆分为小块，分配到集群的多个节点并行处理。如图2-5。

- (1) Map 阶段：将输入的大数据集切分为若干数据块，由多个 Map 任务并行处理。每个 Map 任务对数据块进行分析、过滤和转换，输出键值对作为中间结果。
- (2) Shuffle 与 Sort 阶段：对 Map 阶段输出的所有键值对按照 key 进行分组和排序。该阶段会将相同 key 的数据聚集到一起，并将数据从 Map 节点传输到对应的 Reduce 节点，为后续归约操作做准备。
- (3) Reduce 阶段：对 Shuffle 与 Sort 阶段分组后的数据进行归约处理。每个 Reduce 任务对同一 key 的所有 value 进行合并、统计或聚合，最终输出结果数据集，完成整个分布式计算过程。

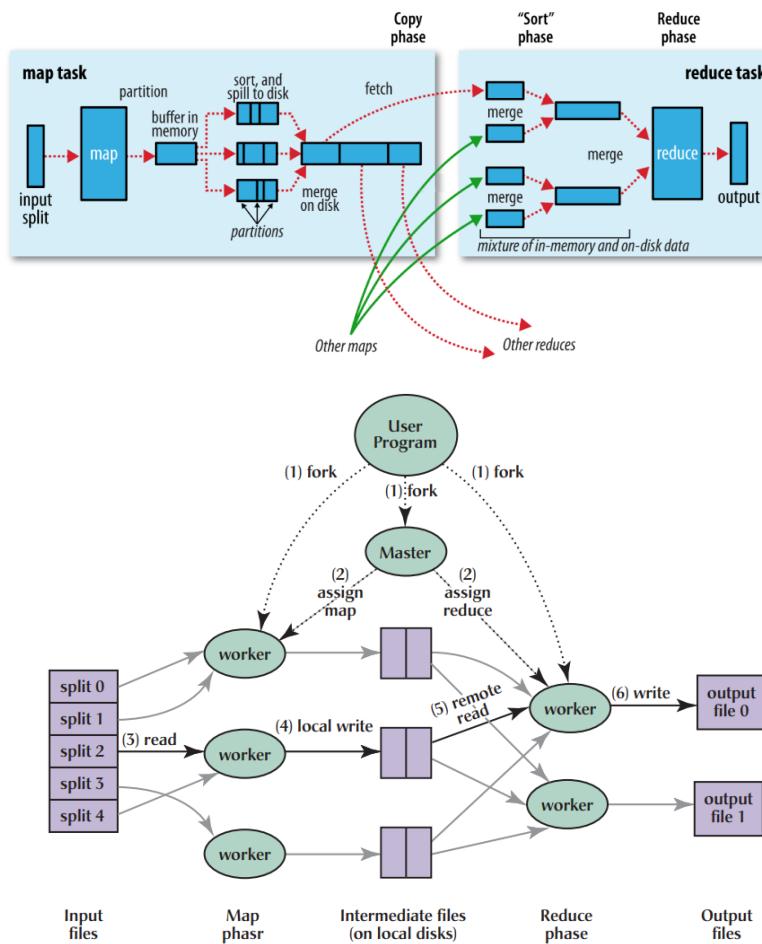


图 2-5 MapReduce 简易流程 [7] [8]

## 2.2.2 SpringBoot 关系

**项目结构:** [9] 前端向系统发起请求，首先由 controller 层进行接收和响应处理，该层负责接口路由和基本参数校验 [10]。随后请求被传递到 service 层，该层负责具体的业务逻辑处理。如果涉及数据库操作，service 层会调用 mapper 层，通过 MyBatis 框架访问底层的 MySQL 数据库，完成数据的增删改查操作。同时，service 层还可以通过 client 与 Hadoop 集群进行交互，实现分布式文件的上传、下载与删除等功能。如图2-6所示：

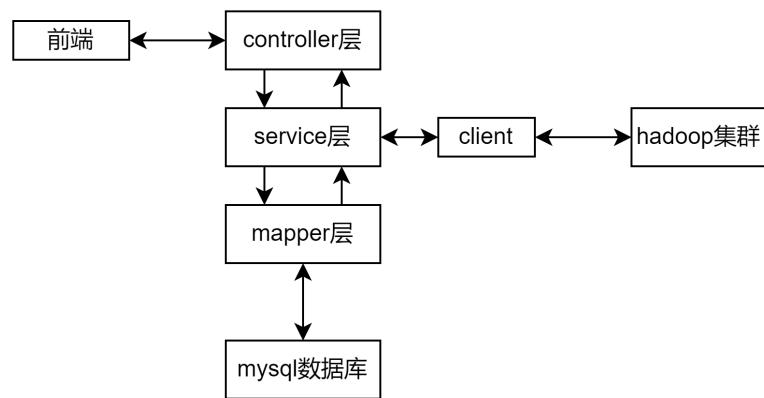


图 2-6 项目结构

## 2.2.3 Vue 关系

### 项目结构:

利用 vue [11] 及 element [12] 等组件构建用户友好的可视化界面。

结构如图2-7所示：

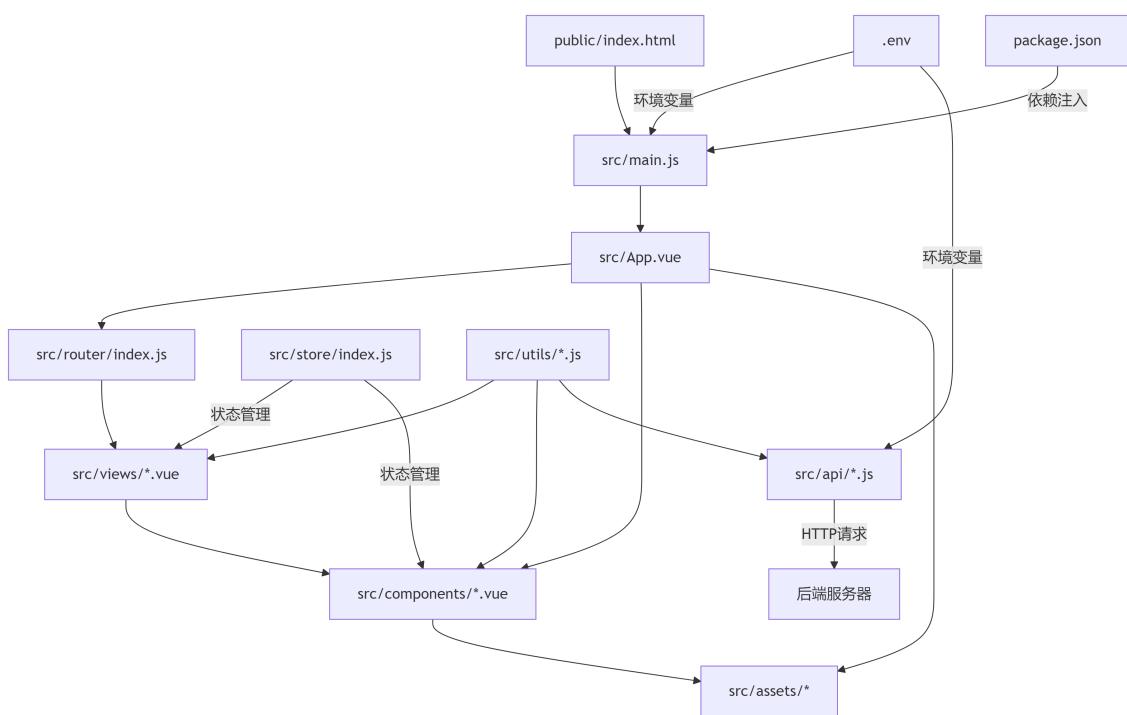


图 2-7 vue 结构

## 第三章 详细设计及实现

### 3.1 hadoop 集群搭建

#### 3.1.1 配置虚拟机

使用VmwareWorkstation Pro, 将准备好的CentOS7操作系统导入虚拟机中, 配置好网络, 选择NAT模式。配置是为了确保虚拟机能够通过宿主机访问外部网络, 同时宿主机也能与虚拟机通信。详细原理见图3-1。

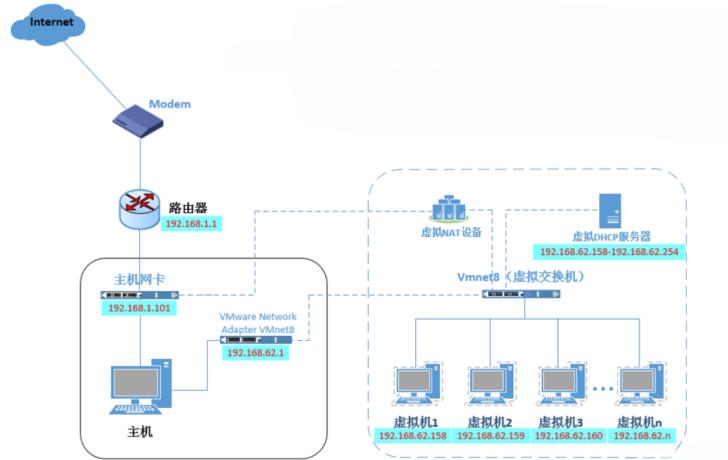


图 3-1 虚拟机网络配置

#### 3.1.2 设备配置

##### 3.1.2.1 角色分配

这里我们使用三个节点搭建hadoop集群, 角色分配如表3-1:

服务器	角色	Namenode	Datanode	Resourcemanager	Nodemanager	SecondaryNameNode
node1	Master	✓	✓	✓	✓	
node2	Slaver1		✓		✓	✓
node3	Slaver2		✓		✓	

表 3-1 Hadoop 集群节点角色分配

##### 3.1.2.2 Hadoop 编译配置

- 安装编译依赖

```
# 1. 编译工具链
yum install -y gcc gcc-c++ make autoconf automake libtool
# 2. 压缩库支持
yum install -y lzo-devel zlib-devel snappy-devel bzip2-devel lzo
lzo-devel lzop zlib
# 3. 其他依赖
yum install -y curl openssl openssl-devel ncurses-devel libXtst
# 4. SASL和文档
yum install -y doxygen cyrus-sasl* saslwrapper-devel*
```

第一个命令：安装编译和运行 Hadoop 的必需依赖，包括：编译工具链、压缩库、加密和网络、其他依赖

第二个命令：安装安全认证和工具。Hadoop 集群需要 Kerberos 安全认证，必须安装 SASL 相关库。

- **安装 CMake**

```
#yum卸载已安装cmake 版本低
yum erase cmake
#解压
tar zxvf CMake-3.19.4.tar.gz
#编译安装
cd /export/server/CMake-3.19.4
./configure
make && make install
#验证
# cmake -version
#如果没有正确显示版本 请断开SSH连接 重写登录
```

- **安装 snappy**

```
#卸载已经安装的
rm -rf /usr/local/lib/libsnappy*
rm -rf /lib64/libsnappy*
#上传解压
tar zxvf snappy-1.1.3.tar.gz
#编译安装
```

```
cd /export/server/snappy-1.1.3  
.configure  
make && make install  
#验证是否安装  
# ls -lh /usr/local/lib |grep snappy
```

- **配置 JDK**

```
#解压安装包  
tar zxvf jdk-8u65-linux-x64.tar.gz  
#配置环境变量  
vim /etc/profile  
export JAVA_HOME=/export/server/jdk1.8.0_241  
export PATH=$PATH:$JAVA_HOME/bin  
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar  
source /etc/profile  
#验证是否安装成功  
java -version
```

- **安装配置 maven**

```
#解压安装包  
tar zxvf apache-maven-3.5.4-bin.tar.gz  
#配置环境变量  
vim /etc/profile  
export MAVEN_HOME=/export/server/apache-maven-3.5.4  
export MAVEN_OPTS="-Xms4096m -Xmx4096m"  
export PATH=$MAVEN_HOME/bin:$PATH  
source /etc/profile  
#验证是否安装成功  
# mvn -v  
#添加maven 阿里云仓库地址 加快编译速度  
vim /export/server/apache-maven-3.5.4/conf/settings.xml  
<mirrors>  
<mirror>  
<id>alimaven</id>
```

```
<name>aliyun maven</name>
<url>http://maven.aliyun.com/nexus/content/groups/public/</url>
<mirrorOf>central</mirrorOf>
</mirror>
</mirrors>
```

- 安装 ProtocolBuffer

```
#解压
tar zxvf protobuf-3.7.1.tar.gz
#编译安装
cd /export/server/protobuf-3.7.1
./autogen.sh
./configure
make && make install
#验证是否安装成功
# protoc --version
```

- 编译 hadoop

```
#上传解压源码包
tar zxvf hadoop-3.3.0-src.tar.gz
#编译
cd /root/hadoop-3.3.0-src
mvn clean package -Pdist,native -DskipTests -Dtar -Dbundle.snappy
-Dsnappy.lib=/usr/local/lib
#参数说明：
Pdist,native : 把重新编译生成的hadoop动态库;
DskipTests : 跳过测试
Dtar : 最后把文件以tar打包
Dbundle.snappy : 添加snappy压缩支持【默认官网下载的是不支持的】
Dsnappy.lib=/usr/local/lib : 指snappy在编译机器上安装后的库路径
```

- 编译之后的安装包路径

```
/root/hadoop-3.3.0-src/hadoop-dist/target
```

### 3.1.2.3 Hadoop 集群分布式安装

将集群的环境全部编译完成之后，对一台机器进行配置，然后利用 scp 命令可以分发到其他机器上。

- 配置主机映射

```
vi /etc/hosts 命令修改主机映射
```

- JDK 安装与环境变量

```
#配置环境变量
vim /etc/profile
export JAVA_HOME=/export/server/jdk1.8.0_241
export PATH=$PATH:$JAVA_HOME/bin
export CLASSPATH=.:${JAVA_HOME}/lib/dt.jar:${JAVA_HOME}/lib/tools.jar
#重新加载环境变量文件
source /etc/profile
```

- 系统配置

```
# 集群时间同步
ntpdate ntp5.aliyun.com
# 防火墙关闭
firewall-cmd --state #查看防火墙状态
systemctl stop firewalld.service #停止firewalld服务
systemctl disable firewalld.service #开机禁用firewalld服务
# systemctl status firewalld.service
#创建统一目录
mkdir -p /export/server #软件安装路径
mkdir -p /export/data #数据存储路径
mkdir -p /export/software/ #安装包存储路径
```

- SSH 免密登录

```
# ssh免密登录（只需要配置node1至node1、 node2、 node3即可）
#node1生成公钥私钥
ssh-keygen
#node1配置免密登录到node1 node2 node3
```

```
ssh-copy-id node1  
ssh-copy-id node2  
ssh-copy-id node3
```

- **上传 Hadoop 安装包**

```
hadoop-3.3.0-Centos7-64-with-snappy.tar.gz  
tar zxvf hadoop-3.3.0-Centos7-64-with-snappy.tar.gz
```

- **修改配置文件**

hadoop-env.sh:

```
#文件最后添加  
export JAVA_HOME=/export/server/jdk1.8.0_241  
export HDFS_NAMENODE_USER=root  
export HDFS_DATANODE_USER=root  
export HDFS_SECONDARYNAMENODE_USER=root  
export YARN_RESOURCEMANAGER_USER=root  
export YARN_NODEMANAGER_USER=root
```

core-site.xml:

```
<property>  
    <name>fs.defaultFS</name>  
    <value>hdfs://node1:8020</value>  
</property>  
<!-- 设置Hadoop本地保存数据路径 -->  
<property>  
    <name>hadoop.tmp.dir</name>  
    <value>/export/data/hadoop-3.3.0</value>  
</property>  
<!-- 设置HDFS web UI用户身份 -->  
<property>  
    <name>hadoop.http.staticuser.user</name>  
    <value>root</value>  
</property>  
<!-- 整合hive 用户代理设置 -->
```

```
<property>
    <name>hadoop.proxyuser.root.hosts</name>
    <value>*</value>
</property>
<property>
    <name>hadoop.proxyuser.root.groups</name>
    <value>*</value>
</property>
<!-- 文件系统垃圾桶保存时间 -->
<property>
    <name>fs.trash.interval</name>
    <value>1440</value>
</property>
```

hdfs-site.xml:

```
<!-- 设置SNN进程运行机器位置信息 -->
<property>
    <name>dfs.namenode.secondary.http-address</name>
    <value>node2:9868</value>
</property>
```

mapred-site.xml:

```
<!-- 设置MR程序默认运行模式: yarn集群模式 local本地模式 -->
<property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
</property>
<!-- MR程序历史服务地址 -->
<property>
    <name>mapreduce.jobhistory.address</name>
    <value>node1:10020</value>
</property>
<!-- MR程序历史服务器web端地址 -->
<property>
```

```
<name>mapreduce.jobhistory.webapp.address</name>
<value>node1:19888</value>
</property>
<property>
    <name>yarn.app.mapreduce.am.env</name>
    <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
</property>
<property>
    <name>mapreduce.map.env</name>
    <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
</property>
<property>
    <name>mapreduce.reduce.env</name>
    <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
</property>
```

yarn-site.xml:

```
<!-- 设置YARN集群主角色运行机器位置 -->
<property>
    <name>yarn.resourcemanager.hostname</name>
    <value>node1</value>
</property>
<property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
</property>
<!-- 是否将对容器实施物理内存限制 -->
<property>
    <name>yarn.nodemanager.pmem-check-enabled</name>
    <value>false</value>
</property>
<!-- 是否将对容器实施虚拟内存限制。 -->
<property>
    <name>yarn.nodemanager.vmem-check-enabled</name>
```

```
<value>false</value>
</property>
<!-- 开启日志聚集 -->
<property>
    <name>yarn.log-aggregation-enable</name>
    <value>true</value>
</property>
<!-- 设置yarn历史服务器地址 -->
<property>
    <name>yarn.log.server.url</name>
    <value>http://node1:19888/jobhistory/logs</value>
</property>
<!-- 历史日志保存的时间 7天 -->
<property>
    <name>yarn.log-aggregation.retain-seconds</name>
    <value>604800</value>
</property>
```

编辑 hadoop 配置文件 workers:

```
#配置从节点
node1
node2
node3
```

- 分发同步安装包

```
cd /export/server
scp -r hadoop-3.3.0 root@node2:$PWD
scp -r hadoop-3.3.0 root@node3:$PWD
```

- hadoop 添加到环境变量（所有机器）

```
vim /etc/profile

export HADOOP_HOME=/export/server/hadoop-3.3.0
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
```

```
source /etc/profile
#scp给其他机器!
scp /etc/profile root@node2:/etc/
scp /etc/profile root@node3:/etc/
```

```
#重新加载验证
source /etc/profile
hadoop #验证环境变量是否生效
```

图3-2为三台机器的 hadoop 环境的验证

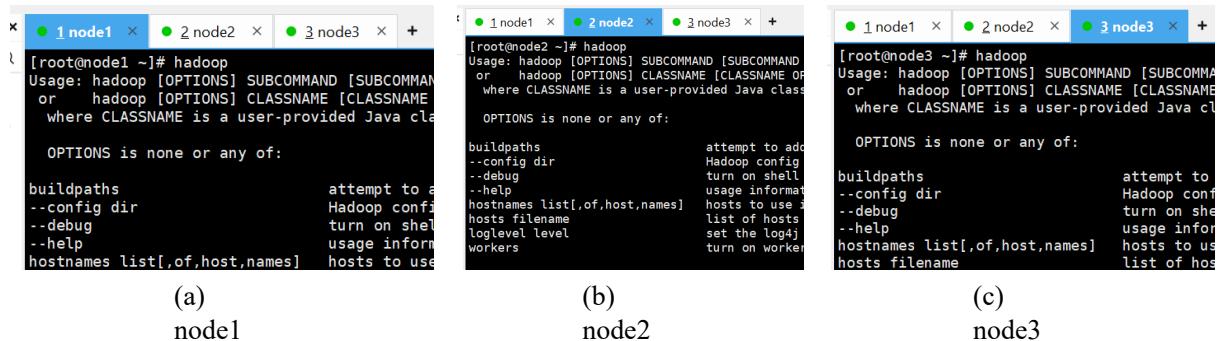


图 3-2 Hadoop 集群各节点示意图

- **hadoop 初始化（注意只执行一次!）** 首次启动 HDFS 时，必须对其进行格式化操作。format 本质上是初始化工作，进行 HDFS 清理和准备工作

```
hdfs namenode -format
```

### 3.1.3 Hadoop 集群启动与使用

#### 3.1.3.1 启动 Hadoop 集群

```
#HDFS集群
start-dfs.sh
stop-dfs.sh
#YARN集群
start-yarn.sh
stop-yarn.sh
#Hadoop集群
start-all.sh
```

stop-all.sh

一般利用：start-all.sh 可以直接全部启动

可以看到 yarn 如图3-3(a)和 hdfs 如图3-3(b)都已经启动成功了。

Name	Type	Size	Last Modified
0101.jpg	file	601.47 KB	Mar 16 22:31
2646230493-1-192.mp4	file	3.4 MB	Mar 16 22:34
article	file	0 B	May 18 22:27
files	file	0 B	May 18 22:29
input	file	0 B	May 05 22:20
output	file	0 B	May 05 22:20
result	file	0 B	May 18 22:26
test	file	0 B	Mar 17 17:53
tmp	file	0 B	May 05 22:13

图 3-3 可视化界面

### 3.1.3.2 Hadoop 常用指令

- 创建文件夹

```
hadoop fs -mkdir [-p] <path>
# path 为待创建的目录
# -p 选项的行为与 Unix mkdir -p 非常相似，它会沿着路径创建父目录。
```

- 查看指定目录下内容

```
hadoop fs -ls [-h] [-R] [<path> ...]
# path 指定目录路径
# -h 人性化显示文件 size
# -R 递归查看指定目录及其子目录
```

- 上传文件

```
hadoop fs -put [-f] [-p] <localsrc> ... <dst>
# -f 覆盖目标文件（已存在下）
# -p 保留访问和修改时间，所有权和权限。
# localsrc 本地文件系统（客户端所在机器）
# dst 目标文件系统（HDFS）
```

- 查看 HDFS 文件内容

```
hadoop fs -cat <src> ...
# 读取指定文件全部内容，显示在标准输出控制台。
# 注意：对于大文件内容读取，慎重。
```

- 下载 HDFS 文件

```
hadoop fs -get [-f] [-p] <src> ... <localdst>
# 下载文件到本地文件系统指定目录， localdst 必须是目录
# -f 覆盖目标文件（已存在下）
# -p 保留访问和修改时间，所有权和权限。
```

- HDFS 数据移动

```
hadoop fs -mv <src> ... <dst>
# 移动文件到指定文件夹下
# 可以使用该命令移动数据，重命名文件的名称
```

## 3.2 后端服务器搭建

基于 Spring Boot 框架的后端系统架构，采用了标准的分层设计模式，包括 Controller 层、Service 层、Service 实现层（Impl）、Mapper 层，并集成了本地缓存和 Hadoop 分布式存储系统，形成了清晰、高内聚低耦合的系统结构。后端结构如图3-4所示，文件结构如图3-5所示。项目文件结构如图3-5所示，主要涉及登录拦截器、HTTP 请求处理、数据库交互、业务逻辑、工具类、WordCount 作业（可能是数据处理任务）、启动类、配置文件以及环境依赖项。

### 3.2.1 环境搭建

- 配置 application.yaml 如图3-6所示
- pom.xml 引入依赖项：
  - spring-boot-starter-web： Spring Boot Web 开发的基础依赖，包含 Spring MVC 和嵌入式 Tomcat。
  - mybatis-plus-spring-boot3-starter： MyBatis-Plus 的 Spring Boot 集成，简化 MyBatis 开发。
  - mysql-connector-j： MySQL 数据库驱动。
  - lombok： 简化 Java 代码，自动生成 getter/setter 等。
  - spring-boot-starter-validation： 参数校验支持。
  - java-jwt： JWT 生成与解析。

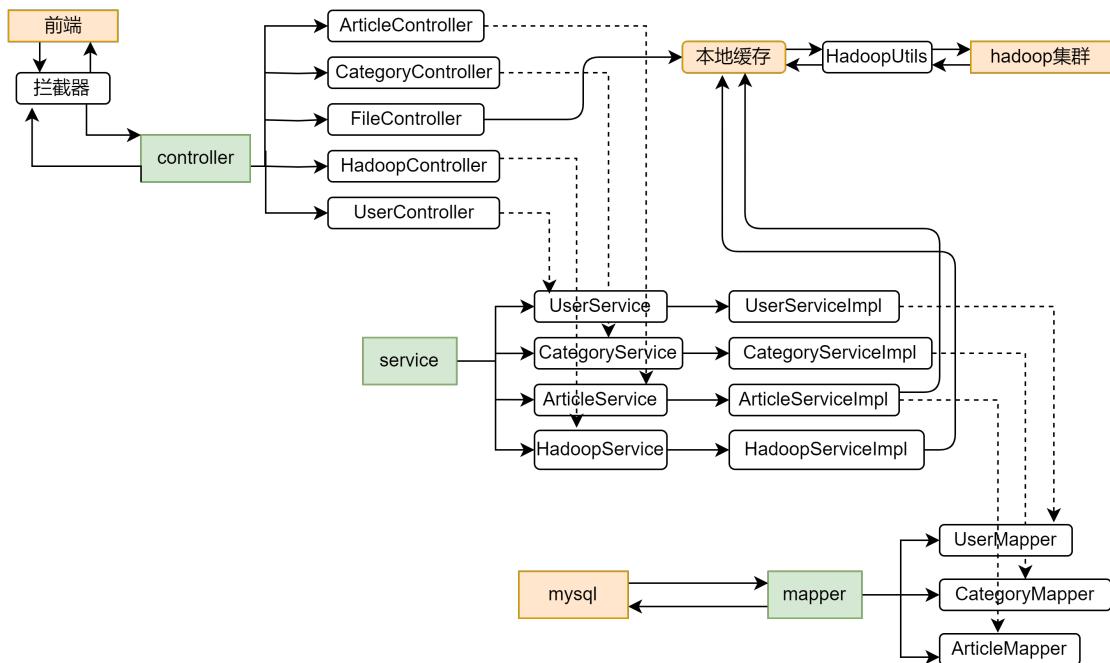


图 3-4 后端结构细节

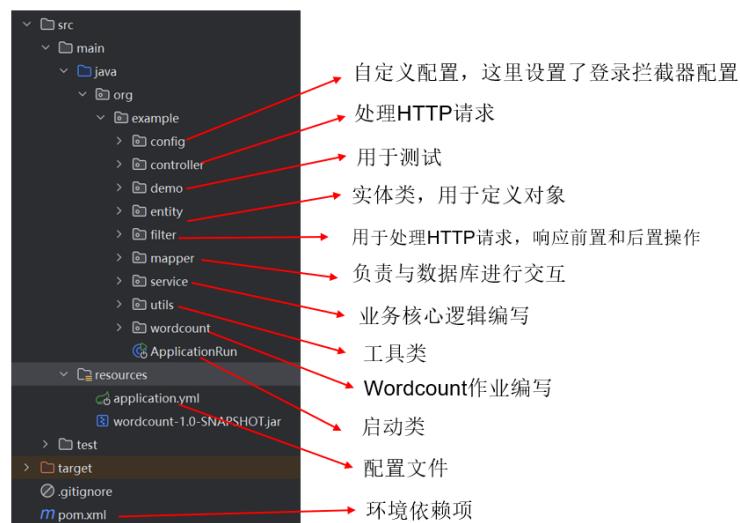


图 3-5 后端文件结构



图 3-6 yml 配置

- spring-boot-starter-test: Spring Boot 测试依赖。
- pagehelper-spring-boot-starter: MyBatis 分页插件。
- jaxb-api、activation、jaxb-runtime: Java XML 绑定相关依赖。
- spring-boot-starter-data-redis: Spring Boot 集成 Redis。
- jackson-databind: JSON 序列化与反序列化。
- jsoup: HTML 解析工具。
- javax.annotation-api: Java 注解支持。
- commons-lang3: Apache 常用工具类库。
- jsch: SSH 协议支持。
- hadoop-common、hadoop-hdfs: Hadoop 核心依赖和 HDFS 支持。
- 构建数据库表  
如图3-7(a)所示, 构建三个数据库表, 分别为用户表、文章表和文件表。例如, 图3-7(b)为用户表的具体内容。
- 与 hadoop 建立 client 连接在 HadoopUtils 里面编写函数

```

1 // 获取 HDFS 客户端
2 public FileSystem getFileSystem() throws Exception {
3     System.setProperty("HADOOP_USER_NAME", "root");
4     Configuration conf = new Configuration();
5     conf.set("fs.defaultFS", "hdfs://mycluster/");
6     conf.set("dfs.nameservices", "mycluster");
7     conf.set("dfs.ha.namenodes.mycluster", "nn1,nn2");

```

The figure consists of two parts. Part (a) shows the MySQL Workbench interface with a tree view of databases and tables. The 'user' table under the 'userdb' database is selected. Part (b) shows a table result set titled 'WHERE ORDER BY time DESC'. The table has columns: id, username, password, email, and time. It contains 9 rows of user data.

	id	username	password	email	time
1	34	westcollb	f379eaf3c831b04de153469d1bec345e	2876587146@qq.com	2025-05-18 22:22:28
2	33	westc001	e10adc3949ba59abbe56e057f20f883e		2025-05-18 22:18:46
3	32	22222	fcea920f7412b5da70e0cf42b8dc93759		2025-05-18 20:28:34
4	29	nihao	e10adc3949ba59abbe56e057f20f883e	lvlebin367@gmail.com	2025-05-18 20:22:43
5	31	海宝宝宝宝宝	b8fc75a10dd8b7b755c10808fce2be3		2025-05-18 20:16:17
6	30	95173	e0e1804a07ae56307dd2e78399f9c1ac		2025-05-18 20:16:15
7	28	skifisy	e10adc3949ba59abbe56e057f20f883e		2025-05-18 19:52:34
8	27	11111	b0baee9d279d34f1df71aa0b908c5f		2025-05-18 19:51:47
9	26	hello	827ccb0eea8a706c4c34a16891f84e7b		2025-05-18 19:49:26

(a) 数据库表

(b) 用户表

图 3-7 数据库表

```

8     conf.set("dfs.namenode.rpc-address.mycluster.nn1",
9         "192.168.88.151:8020");
10    conf.set("dfs.namenode.rpc-address.mycluster.nn2",
11        "192.168.88.152:8020");
12    conf.set("dfs.client.failover.proxy.provider.mycluster",
13        "org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider");
14    return FileSystem.get(conf);
15 }

```

### 3.2.2 用户登录注册

#### 3.2.2.1 注册

我们想要实现注册逻辑，首先在 controller 层，写一下主要的控制单元（register），里面是主体逻辑：判断用户是否存在，存在返回 result.error，不存在那么添加数据。添加数据的逻辑就会调用 service 层，service 层主要是用来实现服务，有两个服务，查询是否名字重复和将数据添加到数据库，那么我们再服务层就需要 mapper 层的对于数据库的操作。如图3-8所示。

#### 3.2.2.2 登录

1. 查询判断用户是否存在。
2. 判断密码是否正确。
3. 得到 jwt 令牌。

如图3-9所示。

#### 3.2.2.3 拦截器

其他页面在无令牌的情况下并且没有被排除在拦截器以外的请求，不让访问，因为其没有登录。

1. 通过实现 WebMvcConfigurer 接口并重写 addInterceptors 方法，

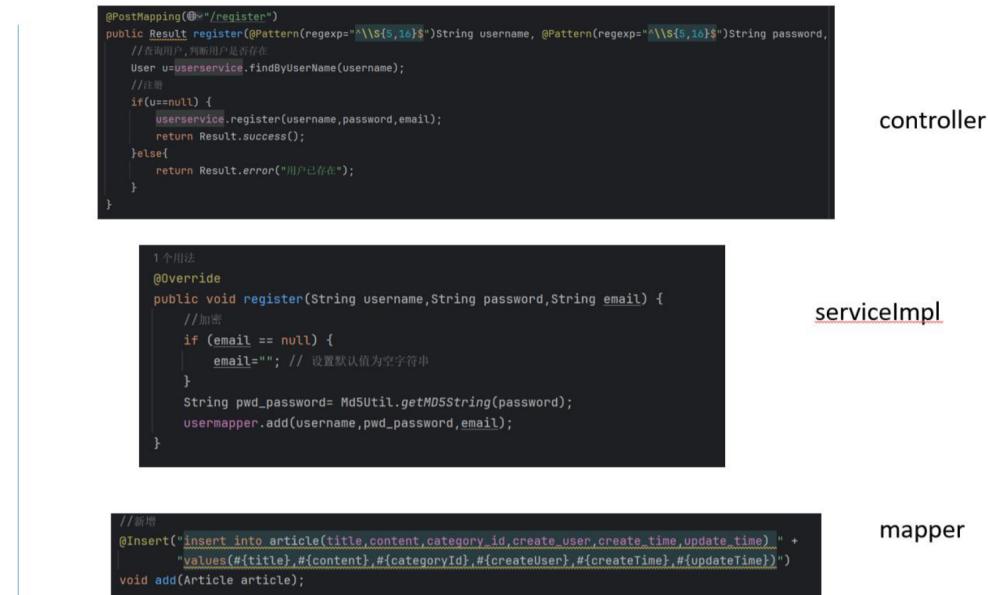


图 3-8 注册

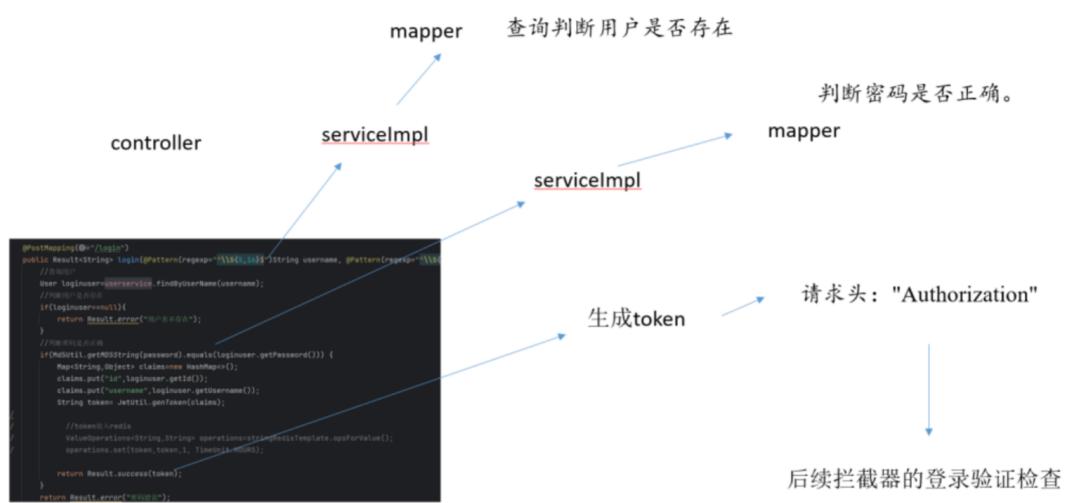


图 3-9 登录

Spring MVC 能够使用自定义的拦截器来对请求进行处理。2. LoginInterceptor 是 Spring MVC 中 HandlerInterceptor 接口的实现类。HandlerInterceptor 是 Spring 提供的一种拦截器机制，允许你在请求的处理链中，在控制器方法执行前、后或者完成时进行处理。具体来说，LoginInterceptor 作为自定义的拦截器，用于处理登录验证的逻辑。3. 自定义的 LoginInterceptor，用于在请求到达控制器之前检查用户的登录状态。具体来说，它通过解析请求头中的 Authorization 字段中的 JWT（JSON Web Token）来验证用户是否登录。如果验证成功，会将用户信息存储在当前线程的 ThreadLocal 中，供后续使用。如果验证失败，则拦截请求并返回 401 错误。如图3-10所示。

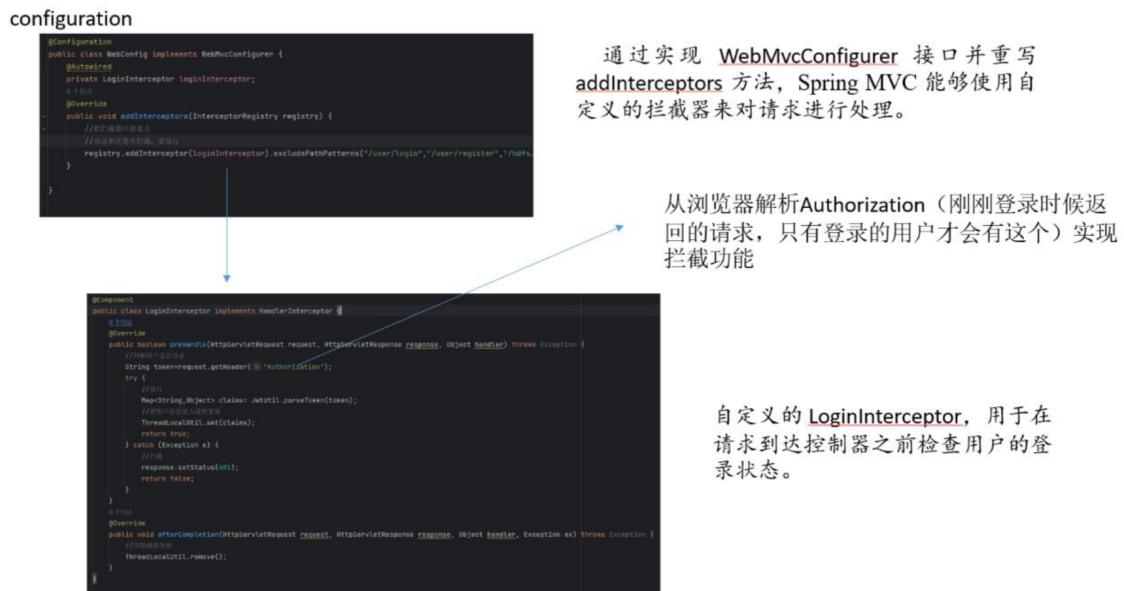


图 3-10 拦截器

### 3.2.2.4 更新用户信息

与之前的逻辑基本一致，不再赘述。在这里补充我们在 controller 层用到的请求方法：

- 1. GET 请求用途：用于从服务器获取资源。GET 请求是无副作用的，意味着它不会修改任何数据，只是请求数据。
- 2. POST 请求用途：用于向服务器提交数据，通常用于创建新的资源。POST 请求会在服务器上产生副作用，通常会创建、修改或触发某些操作。
- 3. PUT 请求用途：用于更新资源或创建资源。PUT 请求会覆盖指定的资源。如果资源不存在，则可以创建该资源。
- 4. DELETE 请求用途：用于删除资源。DELETE 请求会删除服务器上的资源。

### 3.2.3 文章功能

#### 3.2.3.1 文章分类

文章分类实现如图3-11所示，基本原理与用户注册登录类似，都是在通过分层设计实现。



图 3-11 文章分类增删改查

#### 3.2.3.2 文章增删改查

文章增删改查通过分层设计实现，主要代码如下：

- 1. 增加文章

```
1 # controller
2 articleService .add( article );
3 # serviceImpl
4 if( articleMapper . fileExist ( article . getTitle ())){
5     articleMapper . deleteById( article . getId ());
6 }
7 articleMapper .add( article );
8 # 上传到hadoop
9 String [] paths=path( article );
10 String hadooppath=paths[0];
```

```

11 String rootpath=paths[1];
12 // 将 HTML 内容转为纯文本再上传
13 String plainText = Jsoup.parse(article.getContent()).text();
14 hadoopUtils.uploadArticleToHadoop(plainText, rootpath, hadooppath);
15 # mapper
16
17 void add(Article article);
18 @Select("SELECT COUNT(*)>0 FROM article WHERE title=#{filename}")
19 Boolean fileExist(String filename);
20 # hadoopUtils
21 // 先保存内容到本地文件，再上传到 HDFS（去重）
22 public void uploadArticleToHadoop(String content, String localPath,
23                                     String hadoopPath) throws Exception {
24     // 1. 保存内容到本地
25     try (FileWriter writer = new FileWriter(localPath)) {
26         writer.write(content);
27     }
28     // 2. 上传到 HDFS（去重）
29     FileSystem fs = getFileSystem();
30     Path hdfsPath = new Path(hadoopPath);
31     try {
32         if (fs.exists(hdfsPath)) {
33             fs.delete(hdfsPath, true);
34         }
35         try (InputStream in = new FileInputStream(localPath);
36              OutputStream out = fs.create(hdfsPath, true)) {
37             IOUtils.copyBytes(in, out, 1024);
38         }
39     } finally {
40         fs.close();
41     }
42 }

```

- 2. 分页展示文章

```

1 # controller
2 PageBean<Article> pb =
    articleService . list (pageNum,pageSize,categoryId, state );
3 # serviceImpl
4 // 1. 创建PageBean对象
5 PageBean<Article> pb = new PageBean<>();
6 // 2. 开启分页查询 PageHelper
7 PageHelper. startPage (pageNum, pageSize);
8 // 3. 调用mapper
9 Map<String, Object> map = ThreadLocalUtil. get () ;
10 Integer userId = ( Integer ) map.get( "id" );
11 List<Article> as = articleMapper . list ();
12 //Page中提供了方法,可以获取PageHelper分页查询后
    得到的总记录条数和当前页数据
13 Page<Article> p = (Page<Article>) as;
14 //把数据填充到PageBean对象中
15 pb. setTotal (p. getTotal ());
16 pb. setItems (p. getResult ());
17 return pb;
18 # mapper
19 @Select( "SELECT * FROM article" )
20 List<Article> list ();

```

- 3. 更新文章

```

1 # controller
2 article a = articleService . update( article );
3 # serviceImpl
4 return articleMapper . update( article );
5 # mapper
6 Article update( Article article );

```

- 4. 删除文章

```

1 # controller

```

```

2 articleService.deleteById(id);
3 # serviceImpl
4 Article article =articleMapper.findById(id);
5 String [] paths=path( article );
6 String hadooppath=paths[0];
7 String rootpath=paths[1];
8 hadoopUtils.deleteFromHadoop(rootpath,hadooppath);
9 articleMapper.deleteById(id);
10 # mapper
11 @Delete("delete from article where id=#{id}")
12 void deleteById(Integer id);
13 @Select("select * FROM article where id=#{id}")
14 Article findById(Integer id);
15 # hadoopUtils
16 // 删除 HDFS 文件和本地文件
17 public void deleteFromHadoop(String localPath , String hadoopPath) throws
Exception {
18     FileSystem fs = getFileSystem();
19     try {
20         fs . delete (new Path(hadoopPath), true );
21     } finally {
22         fs . close ();
23     }
24     java . io . File file = new java . io . File (localPath);
25     if ( file . exists ()) {
26         file . delete ();
27     }
28 }

```

### 3.2.4 文件后端

文件增加、删除、更新、查询、下载、预览等功能。(主要在 controller 层实现)

- 1. 文件上传

```
1 @PostMapping(consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
```

```

2  public ResponseEntity<FileInfo> uploadFile(
3      @RequestParam("file") MultipartFile file) { // 接收前端上传的Multipart文件
4      try {
5          // 1. 校验文件（例如大小、类型等，具体逻辑在validateFile方法中）
6          validateFile(file);
7
8          // 2. 确保上传目录存在（不存在则创建）
9          Path uploadPath =
10             Paths.get(uploadDir).toAbsolutePath().normalize();
11            Files.createDirectories(uploadPath);
12
13          // 3. 获取原始文件名并生成目标路径
14          String originalFilename =
15             Objects.requireNonNull(file.getOriginalFilename());
16          Path targetPath = uploadPath.resolve(originalFilename);
17
18          // 4. 保存文件到本地（覆盖同名文件）
19          Files.copy(
20              file.getInputStream(),
21              targetPath,
22              StandardCopyOption.REPLACE_EXISTING
23          );
24          System.out.println("文件上传成功，文件地址：" + targetPath);
25
26          // 5. 上传文件到Hadoop（需替换为实际Hadoop工具类逻辑）
27          String hadoopPath = hadoop + "/" + originalFilename;
28          hadooputils.uploadFileToHadoop(targetPath.toString(),
29              hadoopPath);
30
31          // 6. 返回成功响应（包含文件名信息）
32          return ResponseEntity.ok(new FileInfo(originalFilename));
33      }
34  }

```

```

30
31     } catch (IllegalArgumentException e) {
32         // 校验失败: 返回400 Bad Request
33         return ResponseEntity.badRequest().build();
34     } catch (Exception e) {
35         // 其他错误: 返回500 Internal Server Error
36         e.printStackTrace(); // 实际项目中建议用日志记录
37         return ResponseEntity.internalServerError().build();
38     }
39 }
```

- 2. 文件展示与查询

```

1  @GetMapping
2  public ResponseEntity<Map<String, Object>> listFiles (
3      @RequestParam(defaultValue = "1") int pageNum,      //
4          // 当前页码, 默认为1
5      @RequestParam(defaultValue = "5") int pageSize,      //
6          // 每页条数, 默认为5
7      @RequestParam(required = false) String filename)      //
8          // 可选文件名过滤条件
9      throws IOException {
10
11
12     // 1. 获取上传目录的绝对路径
13     Path uploadPath = Paths.get(uploadDir).toAbsolutePath().normalize();
14     Map<String, Object> result = new HashMap<>();
15
16     // 2. 检查目录是否存在, 若不存在返回空结果
17     if (!Files.exists(uploadPath)) {
18         result.put("items", Collections.emptyList());
19         result.put("total", 0);
20         return ResponseEntity.ok(result);
21     }
22
23     // 3. 遍历目录下的文件, 过滤出符合条件的文件列表
24 }
```

```

20     List<FileInfo> allFiles = Files . list (uploadPath)
21         . filter ( Files :: isRegularFile ) // 只保留普通文件（排除目录）
22         . map(path -> new FileInfo (path.getFileName().toString ()) ) // 转换为FileInfo对象
23         . filter ( info -> filename == null || filename.isEmpty() ||
24             info.getFilename () . contains (filename)) // 按文件名模糊过滤
25         . collect ( Collectors . toList ());
26
27 // 4. 计算分页参数
28 int total = allFiles . size (); // 文件总数
29 int fromIndex = Math.max(0, (pageNum - 1) * pageSize); // 当前页起始索引
30 int toIndex = Math.min(fromIndex + pageSize, total ); // 当前页结束索引
31
32 // 5. 截取分页数据（防止越界）
33 List<FileInfo> pageList = fromIndex > total ?
34     Collections . emptyList () :
35     allFiles . subList (fromIndex, toIndex );
36
37 // 6. 返回分页结果
38 result . put( "items" , pageList); // 当前页数据
39 result . put( "total" , total ); // 总文件数
40 return ResponseEntity . ok( result );
41 }

```

- 3. 文件下载

```

1 @GetMapping( "/{filename}" )
2 public void downloadFile(
3     @PathVariable String filename, // 从URL路径中获取文件名
4     HttpServletResponse response) // 直接操作HTTP响应流
5     throws IOException {

```

```

6
7 // 1. 构建文件绝对路径（安全处理）
8 Path uploadPath = Paths.get(uploadDir).toAbsolutePath().normalize();
9 Path filePath = uploadPath.resolve(filename);
10
11 // 2. 检查文件是否存在
12 if (!Files.exists(filePath)) {
13     response.setStatus(HttpServletResponse.SC_NOT_FOUND); // 404
14     return;
15 }
16
17 // 3. 处理文件名编码（解决中文/特殊字符问题）
18 String encodedName = URLEncoder.encode(filename,
19                                         StandardCharsets.UTF_8)
20                                         .replaceAll("\\+", "%20"); // 替换空格编码
21
22 // 4. 自动探测MIME类型（失败时默认二进制流）
23 String contentType = Files.probeContentType(filePath);
24 if (contentType == null) {
25     contentType = "application/octet-stream"; // 通用二进制类型
26 }
27
28 // 5. 设置响应头
29 response.setContentType(contentType);
30 response.setHeader("Content-Disposition",
31                     "attachment; filename*=UTF-8''"+encodedName); // RFC
32                     5987编码
33 response.setHeader("Cache-Control", "no-cache, no-store,
34 must-revalidate"); // 禁用缓存
35
36 // 6. 流式传输文件内容
37 try (InputStream is = Files.newInputStream(filePath);
38      OutputStream os = response.getOutputStream()) {

```

```

35
36     byte[] buffer = new byte[8192]; // 8KB缓冲区（平衡内存和IO效率）
37     int len;
38     while ((len = is.read(buffer)) != -1) {
39         os.write(buffer, 0, len); // 分块写入响应流
40     }
41     os.flush();
42
43 } catch (Exception e) {
44     // 7. 处理传输异常
45
46     response.setStatus (HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
47     // 500
48     e.printStackTrace(); // 生产环境应替换为日志记录
49 }
50 }
```

- 4. 文件删除

```

1 @DeleteMapping("/{filename}")
2 public ResponseEntity<Void> deleteFile (
3     @PathVariable String filename) { // 从URL路径获取待删除文件名
4
5     // 1. 构建文件绝对路径（安全处理）
6     Path uploadPath = Paths.get(uploadDir).toAbsolutePath().normalize();
7     Path filePath = uploadPath.resolve(filename);
8
9     try {
10         // 2. 检查文件是否存在
11         if (!Files.exists(filePath)) {
12             return ResponseEntity.notFound().build(); // 404
13         }
14     }
```

```

15     // 3. 删除本地文件
16     Files . delete ( filePath ) ; // 可能抛出 IOException
17
18     // 4. 同步删除 Hadoop 中的文件 (假设 hadooputils 已实现)
19     String hadoopPath = hadoop + "/" + filename ;
20     hadooputils . deleteFromHadoop ( filePath . toString () , hadoopPath );
21
22     // 5. 返回 204 No Content (删除成功无返回值)
23     return ResponseEntity . noContent () . build ();
24
25 } catch ( IOException e ) {
26     // 6. 文件操作异常 (如权限不足、磁盘错误)
27     return ResponseEntity . internalServerError () . build (); // 500
28 } catch ( Exception e ) {
29     // 7. 其他未知异常 (如 Hadoop 连接失败)
30     throw new RuntimeException ( "文件删除失败" , e ); //
31         转换为非受检异常
32 }

```

- 5. 文件预览

```

1 @GetMapping ( "/preview/{filename}" )
2 public void previewFile (
3     @PathVariable String filename , // 从 URL 路径获取文件名
4     HttpServletResponse response ) // 直接操作 HTTP 响应流
5     throws IOException {
6
7     // 1. 构建文件绝对路径 (安全处理)
8     Path uploadPath = Paths . get ( uploadDir ) . toAbsolutePath () . normalize ();
9     Path filePath = uploadPath . resolve ( filename );
10
11    // 2. 检查文件是否存在
12    if ( ! Files . exists ( filePath ) ) {
13        response . setStatus ( HttpServletResponse . SC_NOT_FOUND ); // 404

```

```

14         return ;
15     }
16
17     // 3. 自动探测MIME类型（失败时默认二进制流）
18     String contentType = Files.probeContentType(filePath);
19     if (contentType == null) {
20         contentType = "application/octet-stream"; // 通用二进制类型
21     }
22
23     // 4. 设置响应头（与下载接口的区别：无Content-Disposition头）
24     response.setContentType(contentType);
25     response.setHeader("Cache-Control", "no-cache, no-store,
26                         must-revalidate"); // 禁用缓存
27
28     // 5. 流式传输文件内容
29     try (InputStream is = Files.newInputStream(filePath);
30          OutputStream os = response.getOutputStream()) {
31
32         byte[] buffer = new byte[8192]; // 8KB缓冲区
33         int len;
34         while ((len = is.read(buffer)) != -1) {
35             os.write(buffer, 0, len); // 分块写入响应流
36         }
37         os.flush();
38     } catch (IOException e) {
39         // 6. 处理传输异常（生产环境应记录日志）
40
41         response.setStatus(HttpStatus.SC_INTERNAL_SERVER_ERROR);
42         // 500
43         e.printStackTrace(); //
44         替换为日志记录，如log.error("文件传输中断", e);
45     }

```

43 }

### 3.2.5 WordCount 作业

作业整体架构如图3-12所示，主要分为 Mapper、Reducer、Driver 三个部分。代码结构如图3-13所示，分为 dowork 与 show。



图 3-12 WordCount 架构

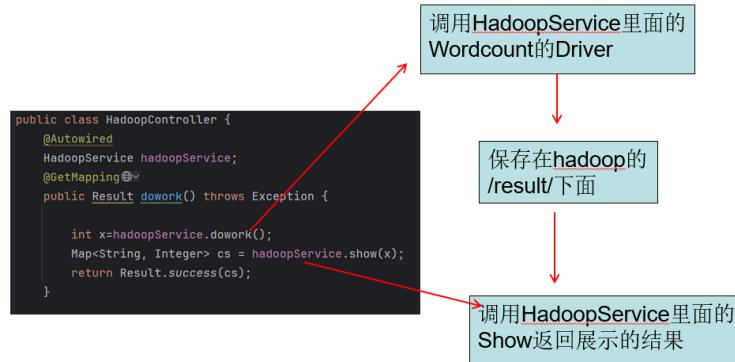


图 3-13 WordCount 作业

## 3.3 前端搭建

### 3.3.1 前端框架

如图3-14所示。`api/`: 封装所有后端 HTTP 请求; `assets/`: 静态资源(如图片、样式); `router`: 定义应用的路由映射关系，路径到组件的映射(如`/login`→`LoginView.vue`)；`stores`: 作用：集中管理跨组件共享的状态，例如记录用户登录状态(`token`)；`utils`: 工具函数；`view`: 视图组件；`App.vue`: 应用的根组件，用于全局布局(如导航栏、页

脚), <RouterView> 用于显示当前路由匹配的页面组件; main.js 作用: Vue 应用的启动入口, 关键操作: 初始化 Vue 实例并挂载到 index.html 的 #app 节点, 注册全局依赖 (路由 Router、状态管理 Pinia) ; index.html: 应用的 HTML 骨架, Vue 组件最终会渲染到 #app 容器中。



图 3-14 vue 框架

### 3.3.2 拦截器与响应器

请求拦截器:

动态从 Pinia (tokenStore) 获取 Token, 并自动附加到请求头。避免手动在每个请求中重复设置 Token, 提高代码复用性。

响应拦截器:

业务数据提取: 普通 API 请求直接返回 response.data, 减少冗余代码。文件下载兼容: 对 responseType: 'blob' 的请求保留完整响应, 确保能读取文件流和响应头 (如文件名)。统一错误处理: 拦截 401 状态码, 自动跳转登录页并提示用户, 增强安全性。

```

1 // 请求拦截器
2 instance . interceptors . request . use(
3   config => {
4     const tokenStore = useTokenStore()
5     if (tokenStore . token) {
6       config . headers . Authorization = tokenStore . token
  
```

```

7      }
8      return config
9    },
10   err => Promise.reject(err)
11 )
12 // 响应拦截器
13 instance.interceptors.response.use(
14   response => {
15     // 关键：下载接口返回完整 response
16     if (response.config.responseType === 'blob') {
17       return response
18     }
19     // 只处理 HTTP 层面，业务错误不 reject
20     return response.data
21   },
22   err => {
23     if (err.response && err.response.status === 401) {
24       ElMessage.error('请先登录')
25       router.push('/login')
26     }
27     return Promise.reject(err)
28   }
29 )
30 \end{lstlisting}
31
32 \subsection{路由配置}
33
34 \begin{lstlisting}[language=javascript]
35   // 定义路由关系
36 const routes = [
37   { path: '/login', component: LoginVue },
38   {
39     path: '/',
40     component: LayoutVue,
41     redirect: '/article/manage',
42     children: [
43       { path: 'category', component: CategoryList },
44       { path: 'article', component: ArticleList },
45       { path: 'comment', component: CommentList },
46       { path: 'user', component: UserList }
47     ]
48   }
49 ]
50 
```

```

40         { path: '/ article / category ', component: ArticleCategoryVue },
41         { path: '/ article / manage', component: ArticleManageVue },
42         {path :'/ article /hadoop',component: ArticleHadoop},
43         { path: '/ user / info ', component: UserInfoVue },
44         { path: '/ user / resetPassword ', component: UserResetPasswordVue },
45         { path: '/ file ',name: '/ file ',component: FileViewVue},
46     ],
47
48 },
49
50
51 ]
52
53 // 创建路由器
54 const router = createRouter({
55     history: createWebHistory(),
56     routes: routes
57 })
58
59 // 导出路由
60 export default router

```

### 3.3.3 登录注册页面

- 脚本部分：引入 Element Plus 图标、消息提示、表单等组件。使用 ref 定义响应式变量，如 isRegister 控制注册/登录表单切换，registerData 存储表单数据。定义表单校验规则和自定义校验函数（如确认密码一致性）。引入并调用后端接口（userRegisterService、userLoginService）实现注册和登录功能，登录成功后将 token 存储到 pinia，并跳转首页

关键代码：

```

1 // 控制注册与登录表单显示
2 const isRegister = ref( false )
3 // 数据模型
4 const registerData = ref({

```

```

5     username: '',
6     password: '',
7     rePassword: ''
8 }
9 // 注册
10 const register = async () => {
11     let result = await userRegisterService( registerData . value )
12     ElMessage.success( result . msg ? result . msg : '注册成功' )
13 }
14 // 登录
15 const login = async () => {
16     let result = await userLoginService( registerData . value )
17     ElMessage.success( result . msg ? result . msg : '登录成功' )
18     tokenStore . setToken( result . data )
19     router . push ( '/')
20 }
21 // 切换表单并清空数据
22 const clearRegisterData = () => {
23     registerData . value = { username: '', password: '', rePassword: '' }
24 }

```

- 模板部分：根据 `isRegister` 显示注册或登录表单，表单内包含用户名、密码、确认密码输入框及相关按钮。注册/登录按钮绑定对应事件，底部有切换表单的链接。定义 CSS 样式，设置表单布局、按钮样式等。

关键代码：

```

1 <!-- 表单切换与按钮绑定 -->
2 <el-card v-if="isRegister"><!-- 注册表单 -->
3     <el-button @click="register">注册</el-button>
4     <el-link @click="isRegister = false ; clearRegisterData () ">←
5         返回</el-link>
6 </el-card>
7 <el-card v-else> <!-- 登录表单 -->
8     <el-button @click="login">登录</el-button>
9     <el-link @click="isRegister = true ; clearRegisterData () ">注册

```

```

→</el-link>
9  </el-card>
```

- api 部分:

```

1 // 导入request.js请求工具
2 import request from '@/utils/request.js'
3 // 提供调用注册接口的函数
4 export const userRegisterService = (registerData) => {
5     // 借助于UrlSearchParams完成传递
6     const params = new URLSearchParams()
7     for (let key in registerData) {
8         params.append(key, registerData[key]);
9     }
10    return request.post('/user/register', params);
11 }
12 // 提供调用登录接口的函数
13 export const userLoginService = (loginData) => {
14     const params = new URLSearchParams();
15     for (let key in loginData) {
16         params.append(key, loginData[key])
17     }
18     return request.post('/user/login', params)
19 }
20 // 获取用户详细信息
21 export const userInfoService = () => {
22     return request.get('/user/userInfo')
23 }
24 // 修改个人信息
25 export const userInfoUpdateService = (userInfoData) => {
26     return request.put('/user/update', userInfoData)
27 }
28 export const updatePasswordService = (passwordData, token) => {
29     return request.patch('/user/updatePwd', passwordData, {
30         headers: {
```

```

31         'Authorization': 'Bearer ${token}'
32     }
33   });
34 };

```

### 3.3.4 文章管理页面

- 脚本部分：引入表格、分页、弹窗等 Element Plus 组件。定义文章列表数据、分页参数、查询条件（如分类、状态）。提供获取文章列表、删除文章、跳转编辑等方法。

关键代码：

```

1 const addArticle = async ()=>{
2   // 调用接口
3   let result = await articleAddService( articleModel .value);
4   ElMessage.success( result .msg? result .msg:'添加成功');
5
6   // 让抽屉消失
7   visibleDrawer .value = false ;
8
9   // 刷新当前列表
10  articleList ()
11 }
12 const showArticle = (row)=>{
13   // 显示抽屉
14   visibleDrawer .value = true ;
15   // 数据拷贝
16   articleModel .value . title  = row. title ;
17   articleModel .value . categoryId = row.categoryId;
18   articleModel .value . content = row.content;
19   articleModel .value . id = row.id;
20 }

```

- 模板部分：顶部为筛选表单（分类、状态、搜索）。中间为文章数据表格，展示标题、分类、作者、状态、操作（编辑/删除）。底部为分页组件。
- api 部分：

```
1 import request from '@/utils/request.js'
2 import { useTokenStore } from '@/stores/token.js'
3 // 文章分类列表查询
4 export const articleCategoryListService = ()=>{
5     // const tokenStore = useTokenStore();
6     // 在pinia中定义的响应式数据,都不需要.value
7     // return
8     request.get('/category',{ headers:{'Authorization':tokenStore.token}}) 
9     return request.get('/category')
10 }
11 // 文章分类添加
12 export const articleCategoryAddService = (categoryData)=>{
13     return request.post('/category',categoryData)
14 }
15
16 // 文章分类修改
17 export const articleCategoryUpdateService = (categoryData)=>{
18     return request.put('/category',categoryData)
19 }
20
21 // 文章分类删除
22 export const articleCategoryDeleteService = (id)=>{
23     return request.delete('/category?id='+id)
24 }
25
26 // 文章列表查询
27 export const articleListService = (params)=>{
28     return request.get('/article',{ params:params})
29 }
30
31 // 文章添加
32 export const articleAddService = (articleData )=>{
```

```

33     return request . post ('/ article ', articleData );
34
35 }
36 export const articleUpdateService = ( articleData )=>{
37     return request . put ('/ article ', articleData )
38 }
39 export const articleDeleteService = ( id )=>{
40     return request . delete ('/ article ?id=' + id )
41 }
42 export const articleupdataService = ( articleData )=>{
43     return request . put ('/ article ', articleData )
44 }
45 export const hadoopList=()=>{
46     return request . get ('/ hadoop ');
47 }

```

### 3.3.5 文件页面

- 脚本部分：主要用于文件的上传、下载、删除、预览和分页管理

```

1 // 文件列表获取
2 const fetchFiles = async () => {
3     const res = await fileListService ({
4         pageNum: pageNum.value,
5         pageSize: pageSize.value ,
6         filename: searchName.value
7     })
8     files . value = res ?. items || []
9     total . value = res ?. total || 0
10 }
11
12 // 文件上传
13 const uploadFile = async ( option ) => {
14     const formData = new FormData()
15     formData.append(' file ', option . file )

```

```
16 await fileUploadService(formData)
17 ElMessage.success('上传成功')
18 fetchFiles()
19 }
20
21 // 文件下载
22 const downloadFile = async (filename) => {
23   const response = await fileDownloadService(filename)
24   const blob = response.data || response
25   const url = window.URL.createObjectURL(blob)
26   const link = document.createElement('a')
27   link.href = url
28   link.setAttribute('download', filename)
29   document.body.appendChild(link)
30   link.click()
31   link.remove()
32   window.URL.revokeObjectURL(url)
33 }
34
35 // 文件删除
36 const deleteFile = (row) => {
37   ElMessageBox.confirm('你确认要删除该文件吗?', '温馨提示')
38     .then(async () => {
39       await fileDeleteService(row.filename)
40       ElMessage.success('删除成功')
41       fetchFiles()
42     })
43 }
44
45 // 文件预览
46 const previewFile = async (filename) => {
47   const response = await filePreviewService(filename)
48   const blob = response.data || response
```

```
49 previewUrl.value = window.URL.createObjectURL(blob)
50 // 根据文件类型设置预览方式
51 // ...
52 previewVisible.value = true
53 }
```

- 模板部分：基本和上述一致。
- api 部分：

```
1 import request from '@/utils/request.js'
2
3 export const fileListService = (params) => {
4     return request.get('/api/files', { params })
5 }
6
7 export const fileUploadService = (formData) => {
8     return request.post('/api/files', formData, {
9         headers: { 'Content-Type': 'multipart/form-data' }
10    })
11 }
12
13 export const fileDeleteService = (filename) => {
14     return request.delete('/api/files/${filename}')
15 }
16
17 // 正确导出下载服务
18 export const fileDownloadService = (filename) => {
19     return request({
20         url: '/api/files/${encodeURIComponent(filename)}',
21         method: 'GET',
22         responseType: 'blob'
23    })
24 }
25
26 // src/api/file.js
```

```
27 export const filePreviewService = (filename) => {
28   return request({
29     url: '/api/files/preview/${encodeURIComponent(filename)}',
30     method: 'GET',
31     responseType: 'blob'
32   })
33 }
```

## 第四章 测试

### 4.1 注册登录测试

#### 4.1.1 拦截器测试

访问: <http://localhost:5173/article/manage> 时, 由于没有登录, 拦截器会拦截请求并返回 401 错误, 如图4-1所示, 并跳转到界面: <http://localhost:5173/login>



图 4-1 拦截器测试

#### 4.1.2 注册测试

如图4-2, 4-3所示。

图 4-2 注册界面

ID	用户名	密码	邮箱	时间
1	uestctest001	e10adc3949ba59abb5e5e057f720ff883e		2025-05-20 18:03:36
2	uestctclb	f379eef3c831b84de15346fd1bec345e	287659714@qq.com	2025-05-18 22:22:28
3	uestc001	e10adc3949ba59abb5e5e057f720ff883e		2025-05-18 22:18:46

图 4-3 后端数据库数据

### 4.1.3 登录测试

登录界面：如图4-4，4-5所示。登录后的主界面：

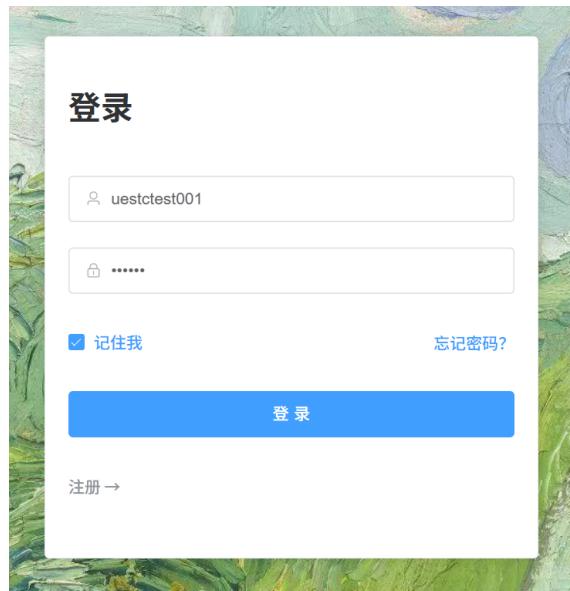


图 4-4 登录

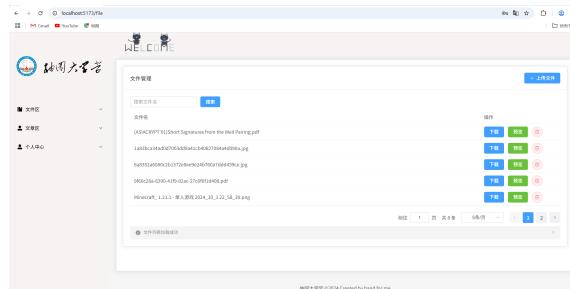


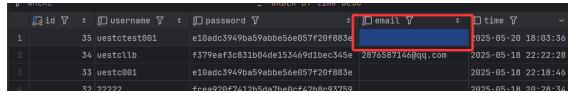
图 4-5 主界面

### 4.1.4 用户信息更改

用户信息更改，如图4-6所示。

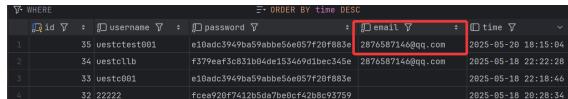
图 4-6 用户信息更改

用户信息更改前后对比，如图4-6,4-8所示。



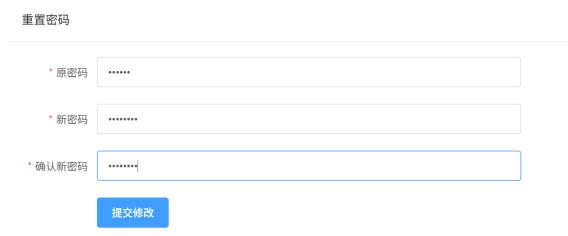
	<input type="checkbox"/> id	<input type="checkbox"/> username	<input type="checkbox"/> password	<input type="checkbox"/> email	<input type="checkbox"/> time
1	35	westctest001	e10adc3949b859abbbe56e057f20ff883	287658714@qq.com	2025-05-18 18:03:36
2	34	westctclb	f379eaef3c831bd4de153469d1bec345e	287658714@qq.com	2025-05-18 22:22:28
3	33	westct001	e10adc3949b859abbbe56e057f20ff883e		2025-05-18 22:18:46
4	32	22222	fcea920ff7412b5da7be0cf42b8c93759		2025-05-18 20:28:34

图 4-7 修改前



	<input type="checkbox"/> id	<input type="checkbox"/> username	<input type="checkbox"/> password	<input type="checkbox"/> email	<input type="checkbox"/> time
1	35	westctest001	e10adc3949b859abbbe56e057f20ff883	287658714@qq.com	2025-05-18 18:15:04
2	34	westctclb	f379eaef3c831bd4de153469d1bec345e	287658714@qq.com	2025-05-18 22:22:28
3	33	westct001	e10adc3949b859abbbe56e057f20ff883e		2025-05-18 22:18:46
4	32	22222	fcea920ff7412b5da7be0cf42b8c93759		2025-05-18 20:28:34

修改后



重置密码

\* 原密码

\* 新密码

\* 确认新密码

图 4-8 更改密码界面

用户密码更改：(原来为：123456，修改为：12345678) 可以看到密文已经发生了变化，说明密码已经被修改。如图4-9所示。



	<input type="checkbox"/> id	<input type="checkbox"/> username	<input type="checkbox"/> password	<input type="checkbox"/> email	<input type="checkbox"/> time
1	35	westctest001	25d55ad283aa400af44ac7ed713cd97ad	287658714@qq.com	2025-05-18 18:23:55
2	34	westctclb	f379eaef3c831bd4de153469d1bec345e	287658714@qq.com	2025-05-18 22:22:28
3	33	westct001	e10adc3949b859abbbe56e057f20ff883e		2025-05-18 22:18:46
4	32	22222	fcea920ff7412b5da7be0cf42b8c93759		2025-05-18 20:28:34

图 4-9 数据库修改

## 4.2 文章与文件区测试

文章分类测试如图4-10所示。文章上传测试如图4-11所示。文章删除测试如图4-12所示。文件上传测试如图4-13所示。文件多种格式测试如图4-14所示。文件下载测试如图4-15所示。文件预览测试如图4-16所示。文件删除测试如图4-17所示。

The figure consists of three parts:

- Top Panel:** A modal window titled "添加分类" (Add Category) with a text input field containing "test". Below the input are two buttons: "取消" (Cancel) and "确认" (Confirm). A red arrow points from the "确认" button to the "test" entry in the main list.
- Middle Panel:** A table titled "文章分类" (Article Categories) with columns "序号" (Index) and "分类名称" (Category Name). The entries are: 1 生活, 2 码农, 3 文学, 4 河畔, 5 学习, and 6 test. To the right of the table is a "操作" (Operation) column with edit and delete icons. A red arrow points from the "test" entry in the list to the corresponding row in the table.
- Bottom Panel:** A database query results table titled "WHERE" with columns "id" and "category\_name". The rows are numbered 1 to 6. The data is as follows:

	WHERE	ORDER BY
	id	category_name
1	13	生活
2	14	码农
3	15	文学
4	16	河畔
5	17	学习
6	18	test

图 4-10 文章分类

The figure consists of three parts:

- Top Panel:** A modal window titled "添加文章" (Add Article) with fields for "文章标题" (Title) set to "test01" and "文章分类" (Category) set to "test". Below the title is a rich text editor with the placeholder "Hello World, This is a test.". A red arrow points from the "增加" (Increase) button in the modal to the "test01" entry in the list below.
- Middle Panel:** A table titled "文章管理" (Article Management) with columns "文章标题" (Title), "分类" (Category), "发表时间" (Publish Time), and "操作" (Operation). The entries are: test01 (生活, 2025-05-18T22:26:18), 111 (生活, 2025-05-18T22:26:35), 222 (文学, 2025-05-18T22:27:18), and 你好 (生活, 2025-05-18T22:27:30). A red arrow points from the "test01" entry in the list to the corresponding row in the table.
- Bottom Panel:** A database query results table titled "WHERE" with columns "title" and "content". The rows are numbered 1 to 4. The data is as follows:

	WHERE	ORDER BY
	title	content
1	106 111	<p><span>/p><p>mountain</p><p>coffee</p>..
2	107 222	<p><span>/p><p>mountain</p><p>coffee</p>..
3	108 你好	<p>111 222</p>
4	109 test01	<p>Hello World, This is a test.</p>

图 4-11 文章上传

## 第四章 测试

The screenshot shows a '温馨提示' (Warm Reminder) dialog box with the message '你确认要删除吗?' (Do you confirm to delete?). Below it is a 'WELCOME' interface titled '文章管理' (Article Management). The table lists three articles:

文章标题	分类	发表时间	操作
111	生活	2025-05-18T22:26:18	
222	文字	2025-05-18T22:26:35	
你好	生活	2025-05-18T22:27:18	

Below the table is a detailed view of the first article '你好.txt':

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	root	supergroup	7 B	May 18 22:27	3	128 MB	你好.txt

At the bottom, there is a message 'Showing 1 to 3 of 3 entries' and navigation buttons 'Previous' and 'Next'.

图 4-12 文章删除

The screenshot shows a '文件上传' (File Upload) progress bar for 'test.jpg'. The progress is at 71%. Below it is a '显示' (Display) interface showing a list of files:

文件名	修改日期	类型	大小
9f66c28a-6390-41f9-82ac-27c0ff1d408.pdf	2025/5/17 22:30	Font PDF Reader D...	2,381 KB
rabbit.jpg	2025/5/17 22:41	JPG 文件	143 KB
test.jpg	2024/4/21 15:2	JPG 文件	71 KB
吕乐彬-50m游泳报名.zip	2025/5/18 11:05	36压缩 ZIP 文件	779 KB
人工智能 - train.py	2025-05-15 21:55:05.m...	脚本文件(mp4)	13,733 KB

Below the table is a search bar '搜索文件名' and a message '找到 1 个结果' (Found 1 result). At the bottom, there is a message 'Showing 1 to 9 of 9 entries' and navigation buttons 'Previous' and 'Next'.

图 4-13 文件上传

The screenshot shows a '文件管理' (File Management) interface with a search bar '搜索文件名' and a '搜索' (Search) button. The table lists several files:

文件名	操作
(ASIACRYPT'01)Short Signatures from the Weil Pairing.pdf	
9f66c28a-6390-41f9-82ac-27c0ff1d408.pdf	
Minecraft_1.21.1 - 单人游戏 2024_10_1_22_58_39.png	
test.jpg	
吕乐彬-50m游泳报名.zip	

At the bottom, there is a message '文件列表加载成功' (File list loading successful) and a page navigation bar.

图 4-14 多种文件格式

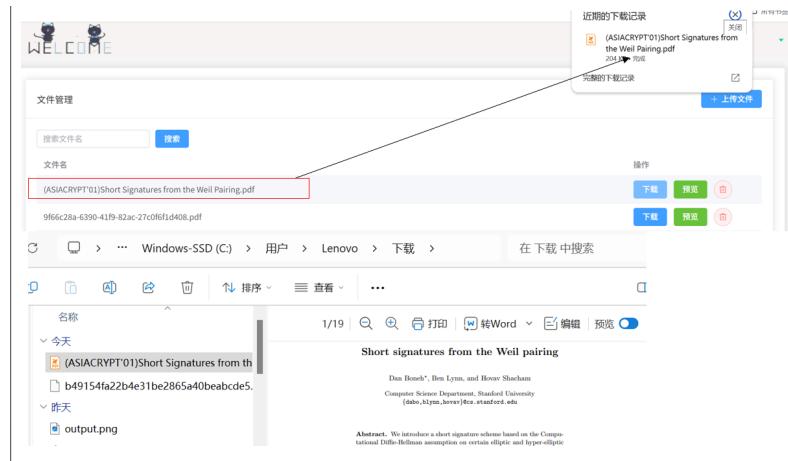


图 4-15 文件下载

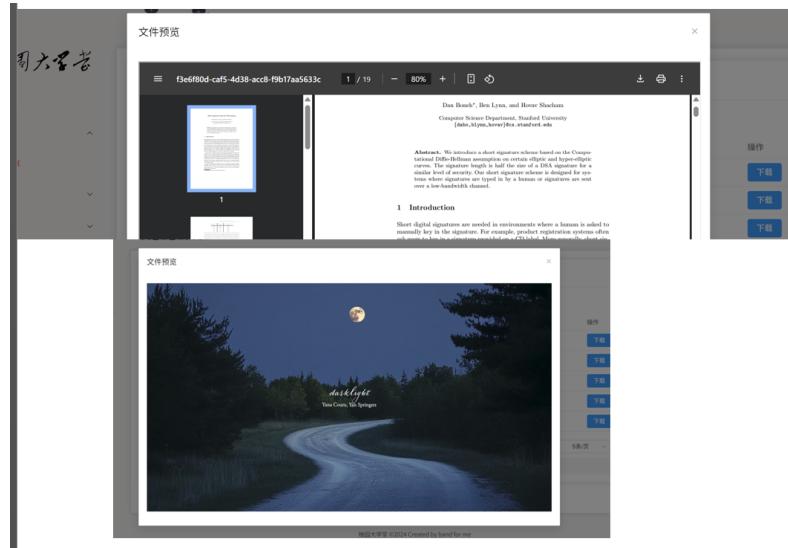


图 4-16 文件预览



图 4-17 文件删除

### 4.3 小结

本文围绕基于 Hadoop HDFS 的 Web 文件管理系统的整体架构与实现进行了详细阐述。首先，介绍了系统的整体架构和各模块的功能分工，包括 Hadoop 集群的搭建、Spring Boot 后端开发、Vue 前端实现等关键环节。随后，详细说明了系统的核功能实现过程，如文件与文章的增删改查、用户管理、权限控制以及与 Hadoop 的集成方式。最后，通过功能测试验证了系统的可用性和稳定性。

## 参考文献

- [1] Tom White. *HADOOP* 权威指南. 清华大学出版社, 北京, 2017.
- [2] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.
- [3] 知乎. 图解 hadoop 生态系统及其组件, 2025. Accessed: 2025-05-21.
- [4] CSDN. Hadoop 入门教程, 2020. Accessed: 2020-04-18.
- [5] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O’Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache hadoop yarn: yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC ’13, New York, NY, USA, 2013. Association for Computing Machinery.
- [6] Jason.Jia. Yarn 的架构及原理, 2024. Accessed: 2020-02-15.
- [7] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.
- [8] itcast. Hadoop, 2021. Accessed: 2021-12-09.
- [9] Spring Team. Spring boot 官方文档, 2025.
- [10] Postman. Postman 官方文档, 2025.
- [11] vue. Vue.js 官方文档, 2025.
- [12] element. element 官方文档, 2025.