



本科毕业设计（论文）

**WISLab管理系统-在线网盘系统**

学 院 计算机学院

专 业 计算机科学与技术

年级班别 2013级（8）班

学 号 3113006129

学生姓名 梁振东

指导教师 刘文印

2017年 5 月



**摘 要**

网络硬盘（简称网盘）是一种基于用户空间的在线存储服务，用户可以通过网络进行数据上传、下载、共享等操作。用户对面向企业网或校园网移动办公与存储的网盘系统有着广阔的市场需求。

本毕业设计是为WisLab实验室开发一个网盘的系统，可供实验室的老师与学生使用，协助WisLab实验室解决一些日常存储文件或共享文件的问题。可以进行文件的上传和下载操作，独立的管理各自的网盘中的文件和目录。该网盘系统可作为管理系统的一个子系统进行整合，更加丰富了实验室管理系统的功能与实用性。

在项目的开发实现上，系统采取Flask作为主要的Web开发框架，开发效率高效且各个模块的工作量与难度也不大；针对网盘数据与用户的对应关系，使用了关系数据库Mysql进行相关模块的数据库开发；在文件分享的开发上，个人的思路采用了key-value的形式来作为文件与路径之间映射，所以采用了缓存Redis进行后端的支持；在文件存储上，考虑到数据的独立性与安全性，为了让数据与程序分离，采用的是FastDFS分布式文件系统，部署在另一的云端主机上，配置的环境为ubuntu，通过Nginx提供http的服务进行通讯；在前端的Web界面开发上，为了开发效率与质量，采用的是Bootstrap和Jinja2模板引擎。

**关键词：**网盘，WisLab，Flask，FastDFS，Web

**Abstract**

Network hard disk(referred to netdisk) is a kind of online storage service base on user space, users can upload、download、share data and others operations through network. Users of the enterprise network or campus network mobile office and storage network disk system has a broad market demand.

This graduation design is a netdisk system for WisLab, which is the MIS system for a lab. The the netdisk system provides the lab members with file storage and sharing functions in WisLab. Each user can manage personal files and folders. It can be integrated as a subsystem of the WisLab system.

In the project development and implementation, I adopt Flask as the main Web development framework, because it is efficiency and small workload and not big difficulty of each module development. For the corresponding relationship between the netdisk and the user, the Mysql is used to solve the database development. For the file sharing, I use a key-value form for the mapping between file and the path, so I adopt the Redis cache system for back-end support. In the file storage, considering the data independency and security, in order to separate the data and the procedures, using the FastDFS distributed file system, deployed in another cloud host, the configuration environment for ubuntu, through the Nginx to provide http services. Finally, using the Bootstrap and Jinja2 template engine for the Web UI design.

**Key words：**netdisk，WisLab，Flask，FastDFS，Web

目 录

[1 绪论 1](#_Toc483336455)

[1.1 课题背景 1](#_Toc483336456)

[1.2 发展现状 1](#_Toc483336457)

[1.3 设计思路 2](#_Toc483336458)

[2 相关知识与技术 3](#_Toc483336459)

[2.1 Flask简介 3](#_Toc483336460)

[2.2 WSGI接口规范 5](#_Toc483336461)

[2.3 Bootstrap、Jinja2 6](#_Toc483336462)

[2.3.1 BootStrap 6](#_Toc483336463)

[2.3.1 Jinja2 7](#_Toc483336464)

[2.4 FastDFS 7](#_Toc483336465)

[2.5 Nginx 10](#_Toc483336466)

[2.6 Redis 11](#_Toc483336467)

[2.7 Mysql 11](#_Toc483336468)

[2.8 HTTP协议 11](#_Toc483336469)

[2.9 Git与GitHub 13](#_Toc483336470)

[3 需求分析 14](#_Toc483336471)

[3.1 功能需求 14](#_Toc483336472)

[3.1.1 注册、登录需求 14](#_Toc483336473)

[3.1.1 查看网盘中文件情况需求 14](#_Toc483336474)

[3.1.1 文件上传需求 14](#_Toc483336475)

[3.1.1 文件下载需求 15](#_Toc483336476)

[3.1.1 网盘文件夹管理需求 15](#_Toc483336477)

[3.1.1 文件分享需求 15](#_Toc483336478)

[3.2 可行性分析 15](#_Toc483336479)

[3.2.1 经济性 15](#_Toc483336480)

[3.2.1 技术性 16](#_Toc483336481)

[3.2.1 操作性 16](#_Toc483336482)

[3.2.1 法律性 16](#_Toc483336483)

[4 概要设计 17](#_Toc483336484)

[4.1 数据实体 17](#_Toc483336485)

[4.2 整体E-R图 17](#_Toc483336486)

[4.3 数据库逻辑结构 17](#_Toc483336487)

[4.4 Redis中数据逻辑结构 19](#_Toc483336488)

[4.5 系统总体框架 19](#_Toc483336489)

[4.5.1 系统结构 19](#_Toc483336490)

[4.5.2 系统架构 20](#_Toc483336491)

[5 具体设计与实现过程 22](#_Toc483336492)

[5.1 界面设计与实现 22](#_Toc483336493)

[5.1.1 基础界面 22](#_Toc483336494)

[5.1.2 登录界面 24](#_Toc483336495)

[5.1.3 注册界面 25](#_Toc483336496)

[5.1.4 个人信息界面 26](#_Toc483336497)

[5.1.5 网盘文件界面 27](#_Toc483336498)

[5.2 后端具体功能设计与实现 30](#_Toc483336499)

[5.2.1 用户模型 30](#_Toc483336500)

[5.2.2 用户的根目录模型 31](#_Toc483336501)

[5.2.3 FastDFS与Nginx配置 31](#_Toc483336502)

[5.2.4 文件上传 32](#_Toc483336503)

[5.2.5 文件下载 33](#_Toc483336504)

[5.2.6 公开分享 33](#_Toc483336505)

[5.2.7 私密分享 34](#_Toc483336506)

[6 系统测试 36](#_Toc483336507)

[6.1 注册与登录测试 36](#_Toc483336508)

[6.1.1 注册 36](#_Toc483336509)

[6.1.2 登录 37](#_Toc483336510)

[6.2 文件上传测试 38](#_Toc483336511)

[6.3 文件下载测试 40](#_Toc483336512)

[6.4 文件分享测试 42](#_Toc483336513)

[6.4.1 公开分享 42](#_Toc483336514)

[6.4.2 私密分享 42](#_Toc483336515)

[参 考 文 献 44](#_Toc483336516)

[致 谢 45](#_Toc483336517)

# 绪论

## 课题背景

随着互联网的飞速发展，数据也急剧膨胀。为了给用户提供更好的服务，互联网企业保存这些数据并挖掘它，这产生了云计算的概念。云计算是当今研究的热门课题，它很好的解决了存储和计算大数据的问题，而云存储作为云计算的衍生物也已经成为一个热门的研究。

计算机通过文件系统管理、存储数据，而信息爆炸时代中人们可以获取的数据成指数倍的增长，单纯通过增加硬盘个数来扩展计算机文件系统的存储容量的方式，在容量大小、容量增长速度、数据备份、数据安全等方面的表现都差强人意。分布式文件系统可以有效解决数据的存储和管理难题：将固定于某个地点的某个文件系统，扩展到任意多个地点/多个文件系统，众多的节点组成一个文件系统网络。每个节点可以分布在不同的地点，通过网络进行节点间的通信和数据传输。人们在使用分布式文件系统时，无需关心数据是存储在哪个节点上、或者是从哪个节点从获取的，只需要像使用本地文件系统一样管理和存储文件系统中的数据。FastDFS是由国人余庆所开发，其项目地址：https://github.com/happyfish100 。FastDFS是一个轻量级的开源分布式文件系统，主要解决了大容量的文件存储和高并发访问的问题，文件存取时实现了负载均衡。

本课题在WISLab管理系统上开发在线网盘系统，可供用户上传、下载、删除和共享文件等操作，采用B/S的架构进行设计该网盘系统，Web开发使用Python Flask框架进行开发，网盘的数据存储在FastDFS中，通过Web进行交互的设计。

## 发展现状

网络硬盘（简称网盘）是一种基于用户空间的在线存储服务，用户可以通过网络进行数据上传、下载、共享等操作。用户对面向企业网或校园网移动办公与存储的网盘系统有着广阔的市场需求。 随着存储技术的不断发展，传统的网盘技术已经显得力不从心，存在传输速度慢、容灾备份及恢复能力低、安全性差、营运成本高等诸多瓶颈。最新应用的云存储技术，为网盘行业带来了新的革命。 云存储是构建在高速分布式存储网络上的数据中心，它将网络中大量不同类型的存储设备通过应用软件集合起来协同工作，形成一个安全的数据存储和访问系统，适用于各大中小型企业与个人用户的数据资料存储、备份、归档等一系列需求。近年来， 国外典型的云存储服务产品有 iCloud、Cloud Drive、Dropbox、Sky Drive 和 Amazon S3 等 。 自 2009年以来，云存储在我国得到了大力发展和推广，联想、新浪、华为、金山、奇虎 360、腾讯等著名企业分别推出了相应的云存储产品或服务，清华大学也为其内部用户推出了云存储服务（Corsair 和 Meepo）。

## 设计思路

系统采取B/S的架构，使用Python Flask进行Web的开发，前端页面使用Bootstrap和Jinja2模板引擎进行开发，客户端与服务器之间使用WSGI接口的规范进行相互之间的通信，主要通过的是http协议进行请求和响应。网盘文件存储方式采用FastDFS分布式文件系统进行网盘数据的存储与管理，该系统在独立的服务器上运行，网盘的文件通过网络连接上传到FastDFS中进行存储，保证了数据的独立和管理上的方便，而Web服务器上仅保存基本的文件信息和实际文件所在的地址和在FastDFS中的文件标识。对于分享的功能使用Redis进行实现，生成一随机分享链接，并与分享的文件的基本信息对应，以Key-Value的形式存放在Redis中，当需要解析分享链接时可通过分享链接在Redis中查询到文件的基本信息。

# 相关知识与技术

## Flask简介

Flask是一个使用Python编写的轻量级Web应用框架。基于Werkzeug WSGI工具箱和Jinja2 模板引擎。Flask使用BSD授权。Flask也被称为“microframework”，因为它使用简单的核心，用extension增加其他功能。Flask没有默认使用的数据库、窗体验证工具。然而，Flask保留了扩增的弹性，可以用Flask-extension加入这些功能：ORM、窗体验证工具、文件上传、各种开放式身份验证技术。

Flask 的作者是 Armin Ronacher（他也是 Werkzeug 及 Jinja2 的作者。）。本来只是作者的一个愚人节玩笑，不过后来大受欢迎，进而成为一个正式的项目。

在Web应用的开发过程中，不同的请求可能会调用相同的处理逻辑。这里有着相同业务处理逻辑的 HTTP 请求可以用一类 URL 来标识。比如论坛站点中，对于所有的获取Topic内容的请求而言，可以用 /topic/<int:topic\_id>/ 这类URL来表示，这里的 topic\_id 用以区分不同的topic。接着在后台定义一个 get\_topic(topic\_id) 的函数，用来获取topic相应的数据，此外还需要建立URL和函数之间的一一对应关系。这就是Web开发中所谓的路由分发，如图2.1

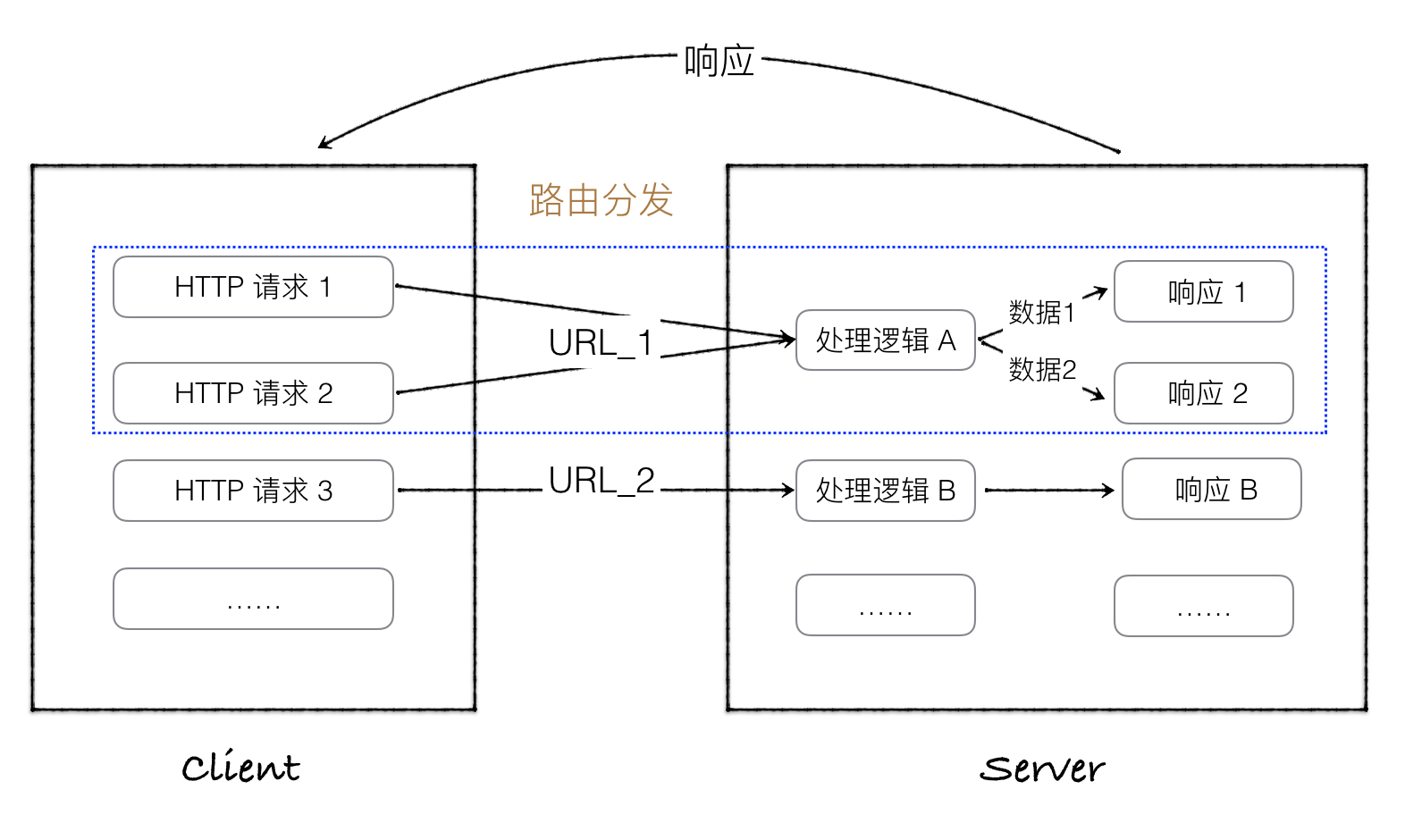


图2.1 Flask路由分发

Flask底层使用werkzeug来做路由分发，代码写起来十分简单，如下：

@app.route('/topic/<int:topic\_id>/')

def get\_topic(topic\_id):

# Do some cal or read from database

# Get the data we need.

通过业务逻辑函数拿到数据后，接下来需要根据这些数据生成HTTP响应（对于Web应用来说，HTTP响应一般是一个HTML文件）。Web开发中的一般做法是提供一个HTML模板文件，然后将数据传入模板，经过渲染后得到最终需要的HTML响应文件。

一种比较常见的场景是，请求虽然不同，但响应中数据的展示方式是相同的。仍以论坛为例，对不同topic而言，其具体topic content虽然不同，但页面展示的方式是一样的，都有标题拦，内容栏等。也就是说，对于 topic 来说，我们只需提供一个HTML模板，然后传入不同topic数据，即得到不同的HTTP响应。这就是所谓的模板渲染，如图2.2所示：

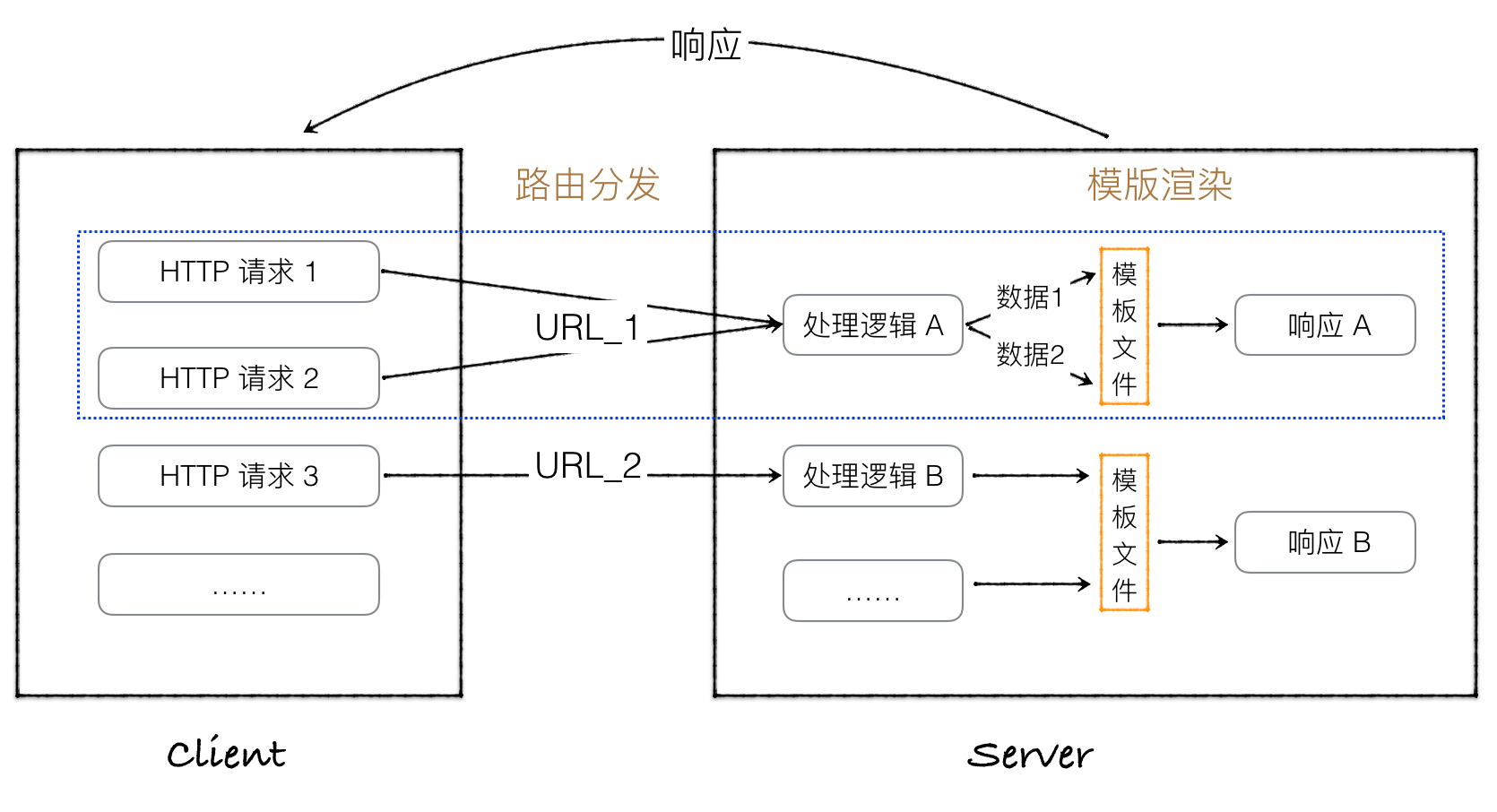


图2.2 Flask中请求-响应过程

Flask 使用 Jinja2 模板渲染引擎来做模板渲染，代码如下：

@app.route('/topic/<int:topic\_id>/')

def get\_topic(topic\_id):

# Do some cal or read from database

# Get the data we need.

return render\_template('path/to/template.html', data\_needed)

总结一下，Flask处理一个请求的流程就是，首先根据 URL 决定由那个函数来处理，然后在函数中进行操作，取得所需的数据。再将数据传给相应的模板文件中，由Jinja2 负责渲染得到 HTTP 响应内容，然后由Flask返回响应内容。

## WSGI接口规范（Web Server Gateway Interface）

WSGI是一套接口规范。一个WSGI程序用以接受客户端请求，传递给应用服务器，再返回服务器的响应给客户端。WSGI程序通常被定义成一个函数，当然你也可以使用类实例来实现。

WSGI简化了编写Web app的复杂度，使程序员不必关注底层的数据传输而专注于Web本身。框架则基于WSGI进一步抽象，用一个函数处理一个URL。而URL与函数的绑定，称为路由(route)，而这些就交给Web框架来做了。Python Flask的路由，是由装饰器实现的。例如：

from flask import Flask

from flask import request

app = Flask(\_\_name\_\_)

@app.route('/', methods=['GET', 'POST'])

def index():

return '<h1>Home</h1>'

@app.route('/user/<name>')

def user(name):

return '<h1>Hello, %s!</h1>' % name

像index()这样的函数称为视图函数(view function)。

<name>属于动态部分，这是一个动态路由。

## Bootstrap、Jinja2

### BootStrap

BootStrap是基于HTML、CSS和JavaScript的框架，使你只需要写简单的代码就可以很快的搭建一个还不错的前端框架，他是后端程序员的福音，使他们只需要专注业务逻辑，而无须浪费太多的精力在界面设计上。

它可以开发全响应式网页——不论你使用手机、平板电脑、普通个人电脑浏览网站内容，所有的元素都可以很优雅的呈现。所以，可以用他来开发适合各种设备浏览的页面，避免了大量的因为兼容性而导致的重复劳动，是目前最受欢迎的前端框架之一。

### Jinja2

Jinja2 是一个现代的，设计者友好的，仿照 Django 模板的 Python 模板语言。 它速度快，被广泛使用，并且提供了可选的沙箱模板执行环境保证安全。

示例：

<title>{% block title %}{% endblock %}</title>

<ul>

{% for user in users %}

<li><a href="{{ user.url }}">{{ user.username }}</a></li>

{% endfor %}

</ul>

类似于常规的标记语言与其他模板语言，{% ... %}为专门的语法语句，{{ ... }}读取服务端传送过来的数据中的变量，例如上面的{{ user. username }}就是读取后端推送过来的user信息中的username。

Jinja2具有以下的特性：

沙箱中执行；

强大的 HTML 自动转义系统保护系统免受 XSS；

模板继承；

及时编译最优的 python 代码；

可选提前编译模板的时间；

易于调试。异常的行数直接指向模板中的对应行；

可配置的语法

## FastDFS

FastDFS的作者是国人余庆，现在淘宝网Java中间件团队从事Java基础平

台研发工作，有10年互联网开发和架构经历，曾担任新浪网开发工程师、雅虎中国架构师。开源分布式文件系统FastDFS和分布式哈希系统FastDHT的作者，对分布式数据存储架构有比较深入的研究。

FastDFS是一款类Google FS的开源分布式文件系统，它用纯C语言实现，支持Linux、FreeBSD、AIX等UNIX系统。

FastDFS主要解决了大容量的文件存储和高并发访问的问题，文件存取时实现了负载均衡；实现了软件方式的RAID，可以使用廉价的IDE硬盘进行存储；支持存储服务器在线扩容；支持相同内容的文件只保存一份，节约磁盘空间；它只能通过专有API对文件进行存取访问，不支持POSIX接口方式，不能mount使用；特别适合大中型网站使用，用来存储资源文件（如：图片、文档、音频、视频等等）。准确地讲，Google FS以及FastDFS、mogileFS、HDFS、TFS等类Google FS都不是系统级的分布式文件系统，而是应用级的分布式文件存储服务。

系统架构如图2.3：

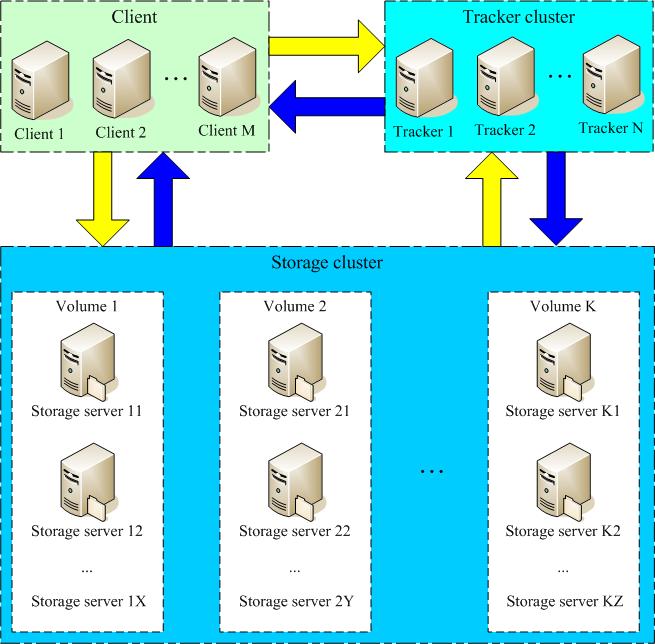


图2.3 FastDFS系统架构

客户端和Storage server主动连接Tracker server。Storage server主动向Tracker server报告其状态信息，包括磁盘剩余空间、文件同步状况、文件上传下载次数等统计信息。Storage server会连接集群中所有的Tracker server，向他们报告自己的状态。Storage server启动一个单独的线程来完成对一台Tracker server的连接和定时报告。需要说明的是，一个组包含的Storage server不是通过配置文件设定的，而是通过Tracker server获取到的。

不同组的Storage server之间不会相互通信，同组内的Storage server之间会相互连接进行文件同步。

上传文件流程如图2.4所示：

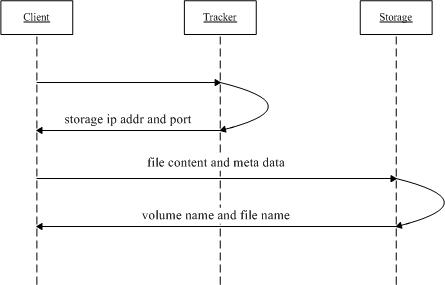


图2.4 FastDFS文件上传

1. client询问tracker上传到的storage，不需要附加参数；

2. tracker返回一台可用的storage；

3. client直接和storage通讯完成文件上传

下载文件流程如图2.4所示：

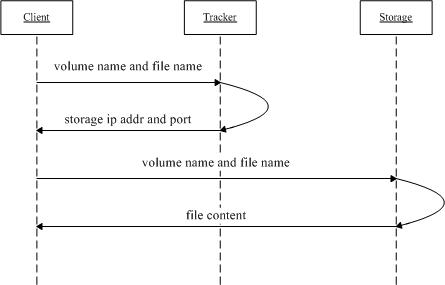


图2.5 FastDFS下载文件流程

1. client询问tracker下载文件的storage，参数为文件标识（组名和文件名）；

2. tracker返回一台可用的storage；

3. client直接和storage通讯完成文件下载。

## Nginx

Nginx是一个高性能的HTTP和反向代理服务器，也是一个IMAP/POP3/SMTP服务器。Nginx是由Igor Sysoev为俄罗斯访问量第二的Rambler.ru站点开发的，第一个公开版本0.1.0发布于2004年10月4日。其将源代码以类BSD许可证的形式发布，因它的稳定性、丰富的功能集、示例配置文件和低系统资源的消耗而闻名。2011年6月1日，nginx 1.0.4发布。

Nginx是一款轻量级的Web 服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器，并在一个BSD-like 协议下发行。由俄罗斯的程序设计师Igor Sysoev所开发，供俄国大型的入口网站及搜索引擎Rambler（俄文：Рамблер）使用。其特点是占有内存少，并发能力强，事实上nginx的并发能力确实在同类型的网页服务器中表现较好，中国大陆使用nginx网站用户有：百度、京东、新浪、网易、腾讯、淘宝等。

## Redis

Redis 是完全开源免费的，遵守BSD协议，是一个高性能的key-value数据库。

Redis 与其他 key - value 缓存产品有以下三个特点：

Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用；

Redis不仅仅支持简单的key-value类型的数据，同时还提供list，set，zset，hash等数据结构的存储；

Redis支持数据的备份，即master-slave模式的数据备份。

## Mysql

MySQL是一个关系型数据库管理系统，由瑞典MySQL AB 公司开发，目前属于 Oracle 旗下产品。MySQL 是最流行的关系型数据库管理系统之一，在 WEB 应用方面，MySQL是最好的 RDBMS (Relational Database Management System，关系数据库管理系统) 应用软件。

MySQL是一种关系数据库管理系统，关系数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。

MySQL所使用的 SQL 语言是用于访问数据库的最常用标准化语言。MySQL 软件采用了双授权政策，分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发都选择 MySQL 作为网站数据库。

## HTTP协议

HTTP协议定义Web客户端如何从Web服务器请求Web页面，以及服务器如何把Web页面传送给客户端。HTTP协议采用了请求/响应模型。客户端向服务器发送一个请求报文，请求报文包含请求的方法、URL、协议版本、请求头部和请求数据。服务器以一个状态行作为响应，响应的内容包括协议的版本、成功或者错误代码、服务器信息、响应头部和响应数据。

以下是 HTTP 请求/响应的步骤：

1、客户端连接到Web服务器

一个HTTP客户端，通常是浏览器，与Web服务器的HTTP端口（默认为80）建立一个TCP套接字连接。例如，http://www.oakcms.cn。

2、发送HTTP请求

通过TCP套接字，客户端向Web服务器发送一个文本的请求报文，一个请求报文由请求行、请求头部、空行和请求数据4部分组成。

3、服务器接受请求并返回HTTP响应

Web服务器解析请求，定位请求资源。服务器将资源复本写到TCP套接字，由客户端读取。一个响应由状态行、响应头部、空行和响应数据4部分组成。

4、释放连接TCP连接

若connection 模式为close，则服务器主动关闭TCP连接，客户端被动关闭连接，释放TCP连接;若connection 模式为keepalive，则该连接会保持一段时间，在该时间内可以继续接收请求;

5、客户端浏览器解析HTML内容

客户端浏览器首先解析状态行，查看表明请求是否成功的状态代码。然后解析每一个响应头，响应头告知以下为若干字节的HTML文档和文档的字符集。客户端浏览器读取响应数据HTML，根据HTML的语法对其进行格式化，并在浏览器窗口中显示。

例如：在浏览器地址栏键入URL，按下回车之后会经历以下流程：

1、浏览器向 DNS 服务器请求解析该 URL 中的域名所对应的 IP 地址;

2、解析出 IP 地址后，根据该 IP 地址和默认端口 80，和服务器建立TCP连接;

3、浏览器发出读取文件(URL 中域名后面部分对应的文件)的HTTP 请求，该请求报文作为 TCP 三次握手的第三个报文的数据发送给服务器;

4、服务器对浏览器请求作出响应，并把对应的 html 文本发送给浏览器;

5、释放 TCP连接;

6、浏览器将该 html 文本并显示内容;

## Git与GitHub

git是一个开源的分布式版本控制系统，用以有效、高速的处理从很小到非常大的项目版本管理。git是个工具，在linux里面也就类似gcc这样的工具一样，是一个shell命令。git是Linus Torvalds为了帮助管理Linux内核开发而开发的一个开放源码的版本控制软件。Torvalds开始着手开发git是为了作为一种过渡方案来替代BitKeeper，后者之前一直是Linux内核开发人员在全球使用的主要源代码工具。开放源码社区中的有些人觉得BitKeeper的许可证并不适合开放源码社区的工作，因此 Torvalds决定着手研究许可证更为灵活的版本控制系统。尽管最初git的开发是为了辅助Linux内核开发的过程，但是我们已经发现在很多其他自由软件项目中也使用了git。

GitHub是一个基于git的代码托管平台，付费用户可以建私人仓库，我们一般的免费用户只能使用公共仓库，也就是代码要公开。Github由Chris Wanstrath, PJ Hyett 与TomPreston-Werner三位开发者在2008年4月创办。迄今拥有59名全职员工，主要提供基于git的版本托管服务。

# 需求分析

## 功能需求

需求分析是指理解用户需求，就软件功能与客户达成一致，估计软件风险和评估项目代价，最终形成开发计划的一个复杂过程。需求分析之所以重要，就因为他具有决策性、方向性、策略性的作用，需求分析是软件定义的最后一个阶段，它的基本任务是回答“系统必须做什么” 这个问题。需求分析的任务不是确定系统怎样完成他的工作，而仅仅是确定系统完成哪些工作，也就是对目标系统提出完整、准确、清晰、具体的要求，它在软件开发的过程中具有举足轻重的地位。

### 注册、登录需求

提供用户注册账号与登录的功能，给用户提供相应的用户空间，建立相应

的用户模型、用户资源，为系统提供服务作准备。

### 查看网盘中文件情况需求

常规的需求，客户端需要向用户展示个人网盘中数据情况，目录结构，文

件的一些信息，文件名，文件大小，修改日期。

### 文件上传需求

提供用户上传文件的功能，从浏览器端上传文件，通过网络连接传输到统

一保存网盘文件的FastDFS服务器上，Web程序服务器则保存与文件相关的基本信息，文件名，存储的ip与文件标识，用于下载时使用，不保存具体的文件数据。

### 文件下载需求

提供用户下载文件的功能，能从浏览器端下载相应的文件。

### 网盘文件夹管理需求（新建文件夹，删除文件夹，返回上一级目录）

类似一般的文件系统，如日常所使用的操作系统中的资源管理器中，能够

新建文件夹，删除文件夹和返回上一级目录。

### 文件分享需求（公开分享，私密分享）

公开分享：

对某个文件进行分享，生成分享链接，任何人获取到链接后，访问链接后都能获取到相应的下载链接。

私密分享：

对某个文件进行私密分享，生成分享链接与提取密码，当获取到链接后，访问链接后，要提示输入提取密码，密码正确方可获取相应的下载链接。

## 可行性分析

可行性分析也称为可行性研究，针对新系统的开发是否具备必要性和可能性，对新系统的开发从技术，经济，社会的方面进行分析和研究，以免投资失误，保证新系统的开发成功。可行性分析是要决定做还是不做。可行性研究的目的就是用最小的代价在尽可能短的时间没确定问题是否能够解决。

### 经济性

系统开发采用轻量高效的Web框架，开发方便且高效；文件存储方面使用

可靠的分布式文件系统；客户端为浏览器，无须安装其他软件，直接通过浏览器访问即可。系统整体经济上可行。

### 技术性

使用Python和Flask进行Web开发，高效快捷；数据库使用Mysql与

Redis，简单易用且难度不大；文件存储使用分布式系统FastDFS，安装方便，API完整可用，通过专有的API进行文件的上传下载操作。整体上技术上可行。

### 操作性

系统操作简单，类似于常规的操作系统文件资源管理器或常用网盘的操

作，对用户要求不高，操作性好。

### 法律性

本系统是自行开发的网盘管理系统，是很有实际意义的系统，开发环境软件和使用的技术都是开源项目，开发这个系统不同于开发普通的系统软件，不存在侵权等问题，即法律上是可行的。

# 概要设计

## 数据实体

用户：网盘系统的使用者。

根文件夹信息：用户所拥有的根文件夹信息（注：保存在Web程序服务器中，每个用户拥有一个根文件夹，里面保存各自的目录结构与文件基本信息，以json文件形式保存，占用存储量极小）。

存储信息：文件存储在分布式系统的信息。

分享信息：用户分享出来的文件信息（分享时随机生成的字符串作为唯一id与文件相关联起来）。

## 整体E-R图

1

1

拥有

根文件夹

用户

1

拥有

n

关联

1

1

存储信息

分享信息

图4.1 整体E-R图

## 数据库逻辑结构

数据库中的用户信息表设计（users）**：**

**表4.1 用户信息表**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 列名 | 数据类型 | | 长度 | 允许空值 | 自动递增 | 主键 | 外键 | 备注 |
| id | | int | 11 | 否 | 是 | 是 | 否 | 用户id |
| username | | varchar | 64 | 否 | 否 | 否 | 否 | 用户名 |
| password\_hash | | varchar | 128 | 否 | 否 | 否 | 否 | 密码 |
| role\_id | | int | 11 | 否 | 否 | 否 | 否 | 角色 |
| location | | varchar | 64 | 是 | 否 | 否 | 否 | 所在地 |
| about\_me | | text |  | 是 | 否 | 否 | 否 | 关于我 |
| member\_since | | datatime |  | 否 | 否 | 否 | 否 | 注册时间 |
| last\_seen | | datatime |  | 否 | 否 | 否 | 否 | 最后访问时间 |

数据库中的用户根目录表设计（UserFolderPath）**：**

**表4.2 用户根目录表**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 列名 | 数据类型 | 长度 | 允许空值 | 自动递增 | 主键 | 外键 | 备注 |
| id | int | 11 | 否 | 是 | 是 | 否 | 用户id |
| username | varchar | 64 | 否 | 否 | 否 | 否 | 用户名 |
| folder\_path | varchar | 1024 | 否 | 否 | 否 | 否 | 根目录 |

## Redis中数据逻辑结构

主要用于分享功能的实现。

公开分享的链接：

{ranstr, filepath}

ranstr -- 随机生成的分享链接，在缓存中唯一的存在，默认为8位字符串，有26个大小写英文字母与阿拉伯数字随机组成。

filepath -- 所分享文件的相对路径，用于解析分享链接后返回文件信息。

缓存过期时间 – 默认配置值为 3 \* 24 \* 60 \* 60，即三天。

私密分享的链接：

{ranstr, rancode, filepath, username}

ranstr -- 随机生成的分享链接，在缓存中唯一的存在，默认为8位字符串，由26个大小写英文字母与阿拉伯数字随机组成。

rancode -- 随机生成提取密码，默认为4位字符串，由26个大小写英文字母与阿拉伯数字随机组成。

Username – 分享人。

filepath -- 所分享文件的相对路径，用于解析分享链接后返回文件信息。

缓存过期时间 – 默认配置值为 3 \* 24 \* 60 \* 60，即三天。

## 系统总体框架

### 系统结构

主要分成用户信息，文件夹管理，文件上传，文件下载，文件分享这几个模块。

WisLab Netdisk System

文件分享

文件下载

文件上传

文件夹管理

用户信息

访问分享

私密分享

公开分享

删除文件夹

新建文件夹

图4.2 系统结构图

### 系统架构

基础环境层是系统最基础的部分。系统中的存储设备通过网络连接在一起，存储设备之上通过操作系统实现了对硬件资源的统一管理。

数据资源层是系统最核心的部分。数据资源层通过服务器集群和分布式文件系统等技术，实现云存储中多个存储设备之间的协同工作。

应用接口层是系统最灵活多变的部分。开发者根据不同的用户需求，开发不同的应用服务接口，提供不同的应用服务。应用接口层融合了数据库操作、FastDFS 操作、Web Service，提供用户认证、文件管理、资源的共享与输出等。

访问层是指用户访问系统的客户端。任何一个授权用户都可以登录到系统享受独立的网盘服务。系统为用户提供Web 模式访问。用户可通过浏览器对文件进行上传下载等，还可以直接对文件进行分享，实现与其他用户之间的文件共享。

访问层

服务层

Firefox

…

Chrome

应用接口层

用户认证、文件管理、共享、上传、下载

数据库

数据资源层

FastDFS

存储层

基础环境层

服务器、存储、网络、操作系统等

图4.3 系统架构图

# 具体设计与实现过程

## 界面设计与实现

前端界面使用Flask-Bootstrap和Jinja2模板引擎进行快速代码编写与开发。

### 基础界面

其他的页面都依赖于这个界面，都继承于这个。

主要代码（base.html）：

{% extends "bootstrap/base.html" %}  
  
{% block title %}WisLab NetDisk{% endblock %}  
  
{% block head %}  
{{ super() }}  
<link rel="shortcut icon" href="{{ url\_for('static', filename='favicon.ico') }}" type="image/x-icon">  
<link rel="icon" href="{{ url\_for('static', filename='favicon.ico') }}" type="image/x-icon">  
{% endblock %}  
  
{% block navbar %}  
<div class="navbar navbar-inverse" role="navigation">  
 <div class="container">  
 <div class="navbar-header">  
 <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">  
 <span class="sr-only">Toggle navigation</span>  
 <span class="icon-bar"></span>  
 <span class="icon-bar"></span>  
 <span class="icon-bar"></span>  
 </button>  
 <a class="navbar-brand" href="{{ url\_for('main.index') }}">WisLab NetDisk</a>  
 </div>  
 <div class="navbar-collapse collapse">  
 <ul class="nav navbar-nav">  
 <li><a href="{{ url\_for('main.index') }}">Home</a></li>  
 </ul>  
 <ul class="nav navbar-nav">  
 {% if current\_user.is\_authenticated %}  
 <li>  
 <a href="{{ url\_for('main.user', username=current\_user.username) }}">  
 Profile  
 </a>  
 </li>  
 {% endif %}  
 </ul>  
 <ul class="nav navbar-nav">  
 {% if current\_user.is\_authenticated %}  
 <li>  
 <a href="{{ url\_for('main.user\_files', username = current\_user.username, cur\_dir=current\_user.username) }}">  
 Files  
 </a>  
 </li>  
 {% endif %}  
 </ul>  
 <ul class="nav navbar-nav navbar-right">  
 {% if current\_user.is\_authenticated %}  
 <li><a href="{{ url\_for('auth.logout') }}">Log Out</a></li>  
 {% else %}  
 <li><a href="{{ url\_for('auth.login') }}">Log In</a></li>  
 {% endif %}  
 </ul>  
 </div>  
 </div>  
</div>  
{% endblock %}  
  
{% block content %}  
<div class="container">  
 {% for message in get\_flashed\_messages() %}  
 <div class="alert alert-warning">  
 <button type="button" class="close" data-dismiss="alert">&times;</button>  
 {{ message }}  
 </div>  
 {% endfor %}  
  
 {% block page\_content %}{% endblock %}  
</div>  
{% endblock %}  
  
{% block scripts %}  
{{ super() }}  
{{ moment.include\_moment() }}  
{% endblock %}

界面效果如图5.1：

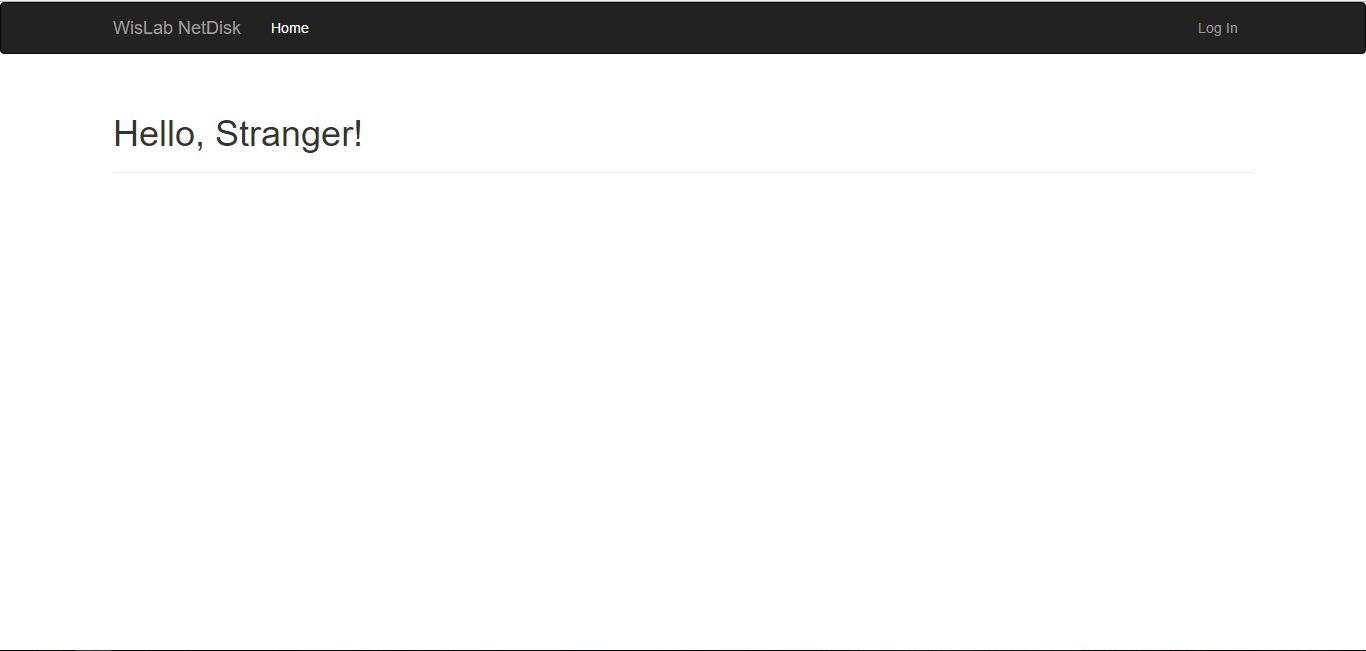


图5.1 基础界面

### 登录界面

呈现给用户的登录表单中包含一个用于用户名的文本字段、一个密码字段、一个“记住我”复选框和提交按钮。表单使用Flask-WTF类。

登陆界面主要代码(login.html)：

{% extends "base.html" %}  
{% import "bootstrap/wtf.html" **as** wtf %}  
{% block title %}WisLab NetDisk - Login{% endblock %}  
{% block page\_content %}  
<div class="page-header">  
 <h1>Login</h1>  
</div>  
<div class="col-md-4">  
 {{ wtf.quick\_form(form) }}  
 <br>  
 <p>New user? <a href="{{ url\_for('auth.register') }}">Click here to register</a>.</p>  
</div>  
{% endblock %}

上述代码中，核心为{{ wtf.quick\_form(form) }}，通过后端向前端推送的form变量，解析成不同的表单，登录的表单定义如下：

class LoginForm(FlaskForm):   
 username = StringField('Username', validators=[DataRequired()])  
 password = PasswordField('Password', validators=[DataRequired()])  
 remember\_me = BooleanField('Keep me logged in')  
 submit = SubmitField('Log In')

表单的定义在后端的forms.py中，不同的表单类代表不同的表单，上述的为登录表单类。

界面效果如图5.2：

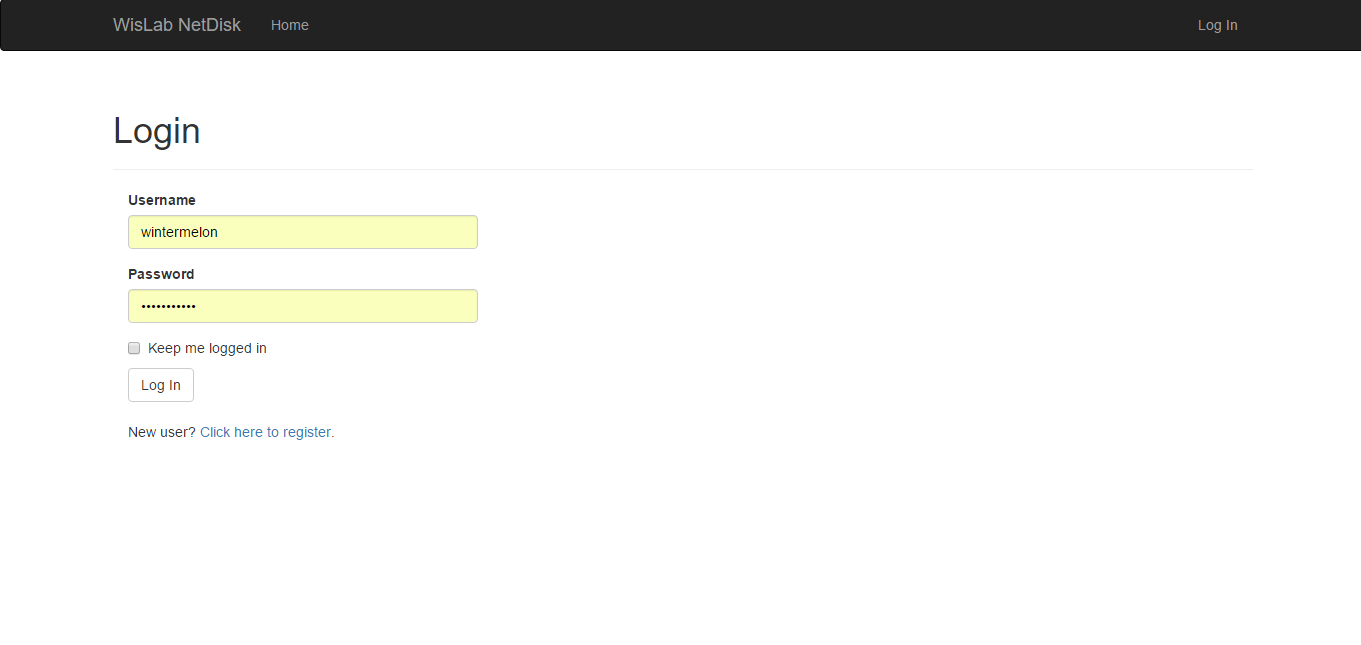


图5.2 登录界面

### 注册界面

注册页面使用的表单要求用户输入用户名和二次确认密码进行注册。

注册界面主要代码(register.html)：

{% extends "base.html" %}  
{% import "bootstrap/wtf.html" **as** wtf %}  
  
{% block title %}WisLab - NetDisk - Register{% endblock %}  
  
{% block page\_content %}  
<div class="page-header">  
 <h1>Register</h1>  
</div>  
<div class="col-md-4">  
 {{ wtf.quick\_form(form) }}  
</div>  
{% endblock %}

同上述的登录表单类，上述代码的form即为forms.py中定义的注册信息类，相应的表单定义为：

class RegistrationForm(FlaskForm):  
 username = StringField('Username', validators=[  
 DataRequired(), Length(1, 64), Regexp('^[A-Za-z][A-Za-z0-9\_.]\*$', 0,  
 'Usernames must have only letters, '  
 'numbers, dots or underscores')])  
 password = PasswordField('Password', validators=[  
 DataRequired(), EqualTo('password2', message='Passwords must match.')])  
 password2 = PasswordField('Confirm password', validators=[DataRequired()])  
 submit = SubmitField('Register')

界面效果如图5.3：

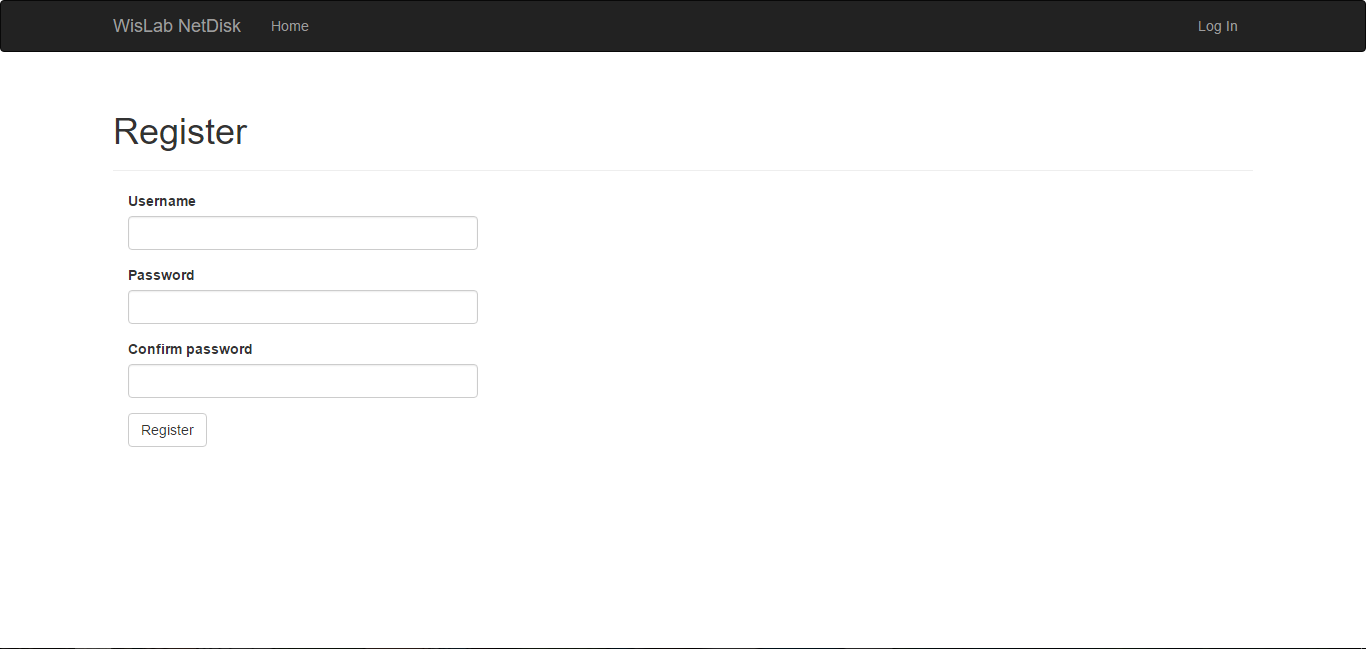


图5.3 注册界面

### 个人信息界面

展示该用户的基本信息，用户名，位置，注册时间，最后一次登录系统操作的时间等。

界面代码（user.html）：

{% extends "base.html" %}  
{% block page\_content %}  
<div>  
 <h1>{{ user.username }}</h1>  
 {% if user.name or user.location %}  
 <p>  
 {% if user.name %}{{ user.name }}{% endif %}  
 位置:  
 {% if user.location %}  
 From <a href="http://maps.google.com/?q={{ user.location }}">  
 {{ user.location }}  
 </a>  
 {% endif %}  
 </p>  
 {% endif %}  
 {% if user.about\_me %}<p>自我介绍:{{ user.about\_me }}</p>{% endif %}  
 <p>  
 注册时间： {{ moment(user.member\_since).format('L') }}.  
 </p>  
 <p>  
 上一次访问时间： {{ moment(user.last\_seen).fromNow() }}.  
 </p>  
</div>  
{% endblock %}

界面效果如图5.4：

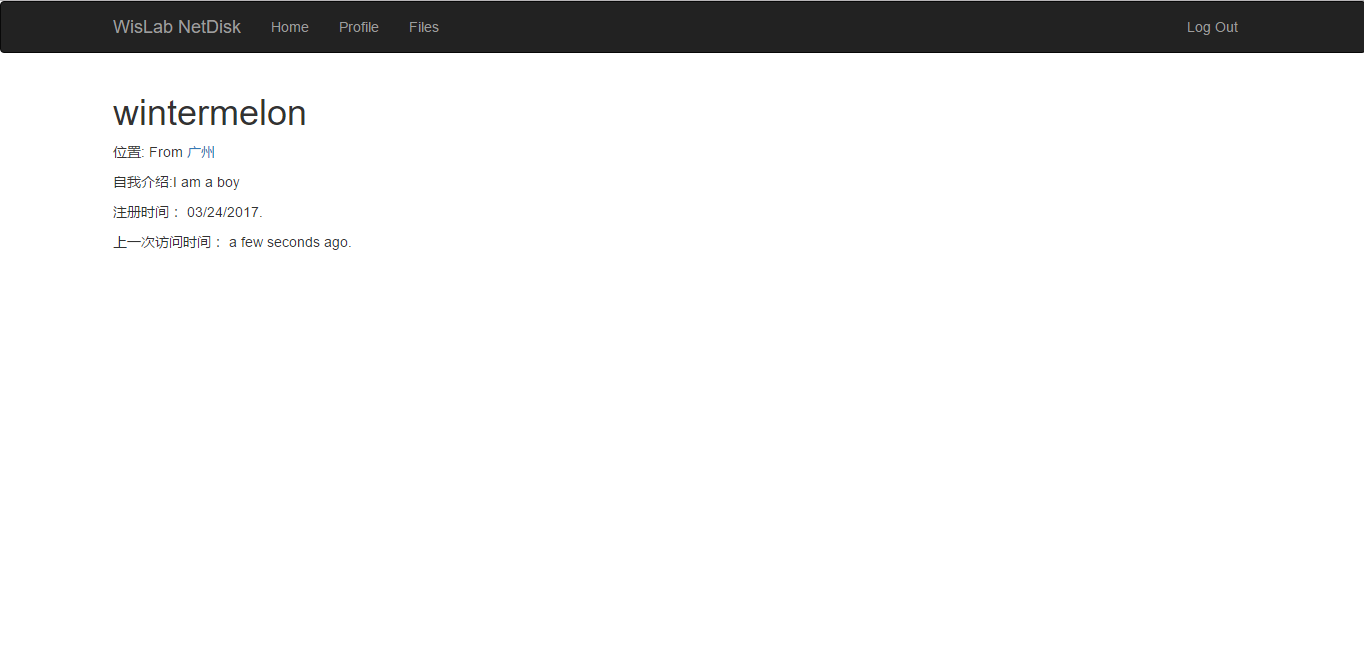


图5.4 个人信息界面

### 网盘文件界面

展示个人网盘的文件情况，具有选择文件上传的控件、返回上一级目录按钮、

新建文件夹对话框、下载按钮、删除操作对话框、分享与私密分享按钮和展示该级目录下的文件与文件夹。

主要代码（user\_file.html）：

上传：

{{ wtf.quick\_form(uploadform) }}

返回上一级目录：

<div class="btn-group" role="group" aria-label="...">  
 <button type="button" class="btn btn-default">  
 <a href="{{ url\_for('main.user\_files', username = current\_user.username, cur\_dir = up\_dir) }}">  
 返回上一级目录  
 </a>  
 </button>  
</div>

新建文件夹模态对话框：

<button type="button" class="btn btn-default" data-toggle="modal" data-target="#myModal">  
新建文件夹  
</button>  
<div class="modal fade" id="myModal" tabindex="-1" role="dialog" aria-labelledby="myModalLabel">  
 <div class="modal-dialog" role="document">  
 <div class="modal-content">  
 <div class="modal-header">  
 <button type="button" class="close" data-dismiss="modal" aria-label="Close">  
 <span aria-hidden="true">×</span>  
 </button>  
 <h4 class="modal-title" id="myModalLabel">新建文件夹</h4>  
 </div>  
 <div class="modal-body">  
 {{ wtf.quick\_form(newfform) }}  
 </div>  
 </div>  
 </div>  
</div>

从后端获取文件数据然后进行解析渲染：

{% for file in files %}

...

{% endfor %}

删除：

<button type="button" class="btn btn-default" data-toggle="modal" data-target="#myDeleteModal">  
删除  
</button>  
<div class="modal fade" id="myDeleteModal" tabindex="-1" role="dialog" aria-labelledby="myModalDelete">  
 <div class="modal-dialog" role="document">  
 <div class="modal-content">  
 <div class="modal-header">  
 <button type="button" class="close" data-dismiss="modal" aria-label="Close">  
 <span aria-hidden="true">×</span>  
 </button>  
 <h4 class="modal-title" id="myModalLabel">删除</h4>  
 </div>  
 <div class="modal-body">  
 确认是否删除？该操作不可逆  
 </div>  
 <div class="modal-footer">  
 <a href="{{ url\_for('main.delete\_folder', delfolderpath=cur\_dir+'/'+dir.dirname, folderpath=cur\_dir, username=current\_user.username) }}">  
 <button type="button" class="btn btn-primary">确认删除</button>  
 </a>  
 <button type="button" class="btn btn-default" data-dismiss="modal">关闭</button>  
 </div>  
 </div>  
 </div>  
</div>

界面效果如图5.5：

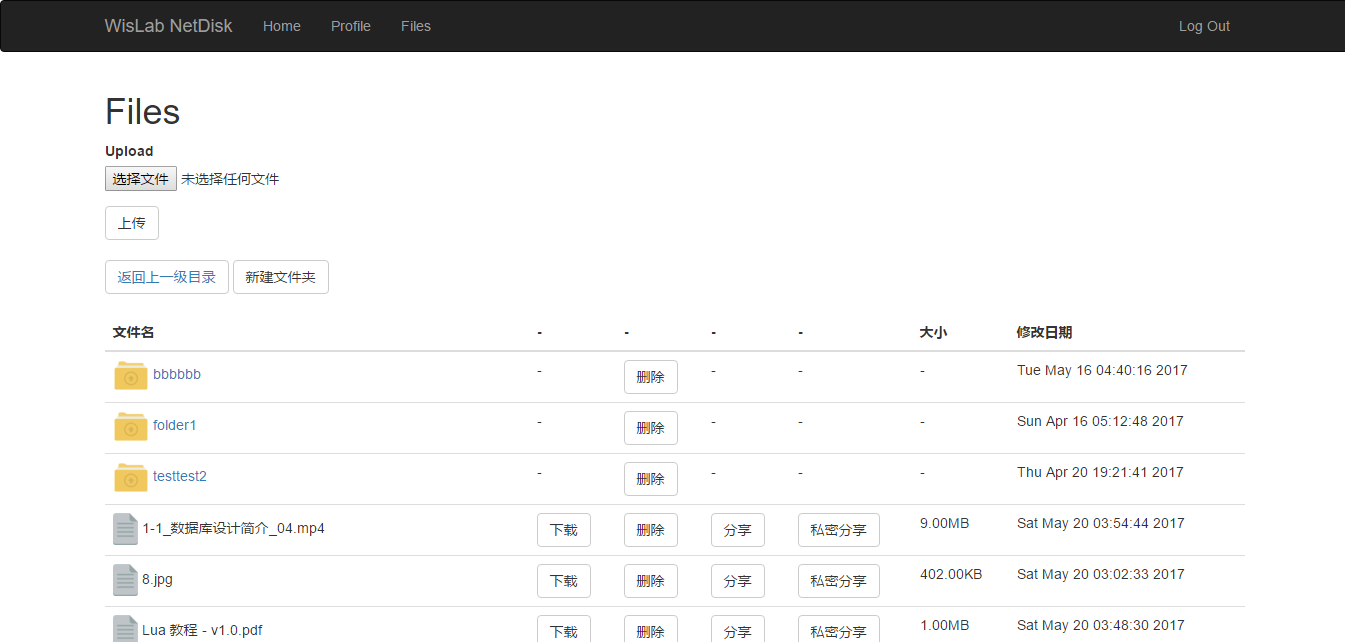


图5.5 网盘文件界面

## 后端具体功能设计与实现

### 用户模型

Flask中对应的用户模型：

class User(UserMixin, db.Model):  
 \_\_tablename\_\_ = 'users'  
 id = db.Column(db.Integer, primary\_key=True)   
 username = db.Column(db.String(64), unique=True, index=True)  
 password\_hash = db.Column(db.String(128))  
 role\_id = db.Column(db.Integer, db.ForeignKey('roles.id'))  
 location = db.Column(db.String(64)) # 所在地  
 about\_me = db.Column(db.Text()) # 关于我  
 member\_since = db.Column(db.DateTime(), default=datetime.utcnow) # 注册时间  
 last\_seen = db.Column(db.DateTime(), default=datetime.utcnow) # 最后访问时间  
  
 def \_\_repr\_\_(self):  
 return '<User %r>' % self.username  
  
 role\_id = db.Column(db.Integer, db.ForeignKey('roles.id'))  
  
 # users = db.relationship('User', backref='role')  
  
 password\_hash = db.Column(db.String(128))  
  
 @property  
 def password(self):  
 raise AttributeError('password is not a readable attribute')  
  
 @password.setter  
 def password(self, password):  
 self.password\_hash = generate\_password\_hash(password)  
  
 def verify\_password(self, password):  
 return check\_password\_hash(self.password\_hash, password)  
  
 def ping(self): # 刷新最后访问的时间  
 self.last\_seen = datetime.utcnow()  
 db.session.add(self)

### 用户的根目录模型

Flask中对应的模型：

class UserFolderPath(db.Model):  
 \_\_tablename\_\_ = 'UserFolderPath'  
 id = db.Column(db.Integer, primary\_key=True)  
 username = db.Column(db.String(64), unique=True, index=True)  
 folder\_path = db.Column(db.String(1024), unique=True, index=True)

### FastDFS与Nginx配置

使用独立的云服务器作为存储设备，安装FastDFS作为管理文件的系统，安装与配置Nginx使其为FastDFS提供Http的服务，通过专有的API与WisLab NetDisk System进行交互和通信。

FastDFS配置过程：（配置环境：Linux操作系统，这里采用的是Ubuntu 14.04）

1、下载安装libfastcommon

2、下载安装fastdfs

3、修改服务器配置文件（在tracker和storage的服务器配置文件中配置好相应的主机ip与端口号）

4、启动tracker服务器

5、安装nginx

6、安装FastDFS的nginx插件，nginx将为FastDFS提供http服务

7、在nginx配置文件中增加对FastDFS的server配置，如下：

server {

　　 listen 80；

　　 location / {

root /home/wintermelon/fastdfs/data;

index index.html index.htm;

}

location /group1/M00 {

root /home/wintermelon/fastdfs/data;

ngx\_fastdfs\_module;

}

}

8、启动storage服务器

9、启动nginx

### 文件上传

前端获取待上传的文件，后端获取文件后，若待上传的文件符合上传的文件

类型与允许的大小要求，则通过FastDFS的Python API连接FastDFS的服务器，通过Http把文件上传到FastDFS中进行保存，上传成功后FastDFS返回文件上传的结果，类似的结果如下：

{'Status': 'Upload successed.', 'Storage IP': '123.207.114.115', 'filename': u'8.jpg', 'Remote file\_id': 'group1/M00/00/00/CmjzU1kfQVCAVwm9AAZKZ2Q7xtQ467.jpg', 'Group name': 'group1', 'Local file name': u'H:\\XJB\_Projects\\WisLabNetDisk\\files\\wintermelon\\8.jpg', 'Uploaded size': '402.00KB'}

返回上传的结果、文件在FastDFS的文件id、文件大小等信息，下载时可根据这些信息进行下载相应的文件。

上传成功后，Flask后端服务器也相应保存一份关于该文件的基本信息，以json文件的形式存放在后端，形式与上述FastDFS返回的上传结果信息一样。

相关代码：

try:  
 my\_fdfs\_client\_file = current\_app.config.get('MY\_FDFS\_CLIENT\_FILE')  
 f\_client = Fdfs\_client(my\_fdfs\_client\_file) # 连接远程FastDfs数据库  
 ret\_upload = f\_client.upload\_by\_filename(os.path.join(upload\_file\_dir, filename))  
 file\_id = ret\_upload['Remote file\_id'].replace('\\', '/') # 新版本文件存放Remote file\_id格式变化  
 ret\_upload['Remote file\_id'] = file\_id  
 ret\_upload['filename'] = filename  
 print ret\_upload  
 print type(ret\_upload)  
 dict\_write\_to\_file(ret\_upload, filename, upload\_file\_dir)  
 os.remove(os.path.join(upload\_file\_dir, filename))  
 flash(u'上传成功！')  
except Exception, ex:  
 print ex

### 文件下载

前端发起下载请求后，Flask后端通过保存的json文件，查找出文件在

FastDFS中的存储位置，从而拼接出相应的下载链接。

相关代码：

@main.route('/files/download/<path:filepath>', methods=['GET'])  
def download\_file(filepath):  
 print("filepath: " + filepath)  
 filepath2 = filepath.split('/')  
 file\_real\_path = current\_app.config.get('UPLOAD\_FOLDER')  
 for i in range(0, len(filepath2) - 1):  
 file\_real\_path += "\\" + filepath2[i]  
 print("file\_read\_path: ", file\_real\_path)  
 filename = filepath2[len(filepath2) - 1] + '.json'  
 print("fllename: ", filename)  
 json\_file = open(os.path.join(file\_real\_path, filename), 'r')  
 json\_load = json.load(json\_file)  
 Storage\_IP = json\_load['Storage IP']  
 Remote\_file\_id = json\_load['Remote file\_id']  
 response = make\_response()  
 print 'http://' + Storage\_IP + '/' + Remote\_file\_id  
 r = requests.get('http://' + Storage\_IP + '/' + Remote\_file\_id)  
 response.data = r.text  
 return response

### 公开分享

Flask后端配置并开启Redis缓存服务，对待分享的文件，生成一个随机的

字符串作为一个key，该key也是作为拼接分享链接的一部分，也是关键的一部分，该key在Redis中唯一存在，文件的路径作为value，把这一{key，value}键值对插入到Redis中，默认设置的缓存失效时间为3 \* 24 \* 60 \* 60，即3天，可根据需要修改配置文件。当解析分享链接时，通过链接中的key，即所生成的字符串，在Redis中查找出相对应的value，即文件的路径，找到该路径下的json文件，拼接出相应的下载链接，即可下载文件。

相关代码：

@main.route('/files/share/<username>/<path:filepath>')  
def share\_file(username, filepath):  
 r = redis.Redis(host="localhost", port=6379, db=0) # 连接redis服务  
 ranstr = random\_str()  
 while r.exists(ranstr):  
 ranstr = random\_str()  
 r.set(ranstr, filepath, ex=current\_app.config.get('REDIS\_EX\_TIME'))  
 print(filepath)  
 print(ranstr)  
 filepath2 = filepath.split('/')  
 sharefilename = filepath2[len(filepath2) - 1]  
 return render\_template('myfiles/share.html', id=ranstr, sharefilename=sharefilename)

### 私密分享

与公开分享实现过程类似，在生成分享链接过程中多一步生成一提取密码，

并在解析分享链接时需要提供提取密码方可获取分享的文件。

相关代码：

@main.route('/files/share\_private/<username>/<path:filepath>')  
def share\_private\_file(username, filepath):  
 r = redis.Redis(host="localhost", port=6379, db=0) # 连接redis服务  
 ranstr = random\_str()  
 while r.exists(ranstr):  
 ranstr = random\_str()  
 rancode = random\_str(4) # 提取码  
 r.rpush(ranstr, rancode)  
 r.rpush(ranstr, filepath)  
 r.rpush(ranstr, username) # 分享人  
 r.expire(ranstr, current\_app.config.get('REDIS\_EX\_TIME')))

print(filepath)  
 print(ranstr)  
 filepath2 = filepath.split('/')  
 sharefilename = filepath2[len(filepath2) - 1]  
 return render\_template('myfiles/share\_private.html',id=ranstr,

extractcode=rancode, sharefilename=sharefilename)

# 系统测试

## 注册与登录测试

### 注册

填写注册的用户名与密码：

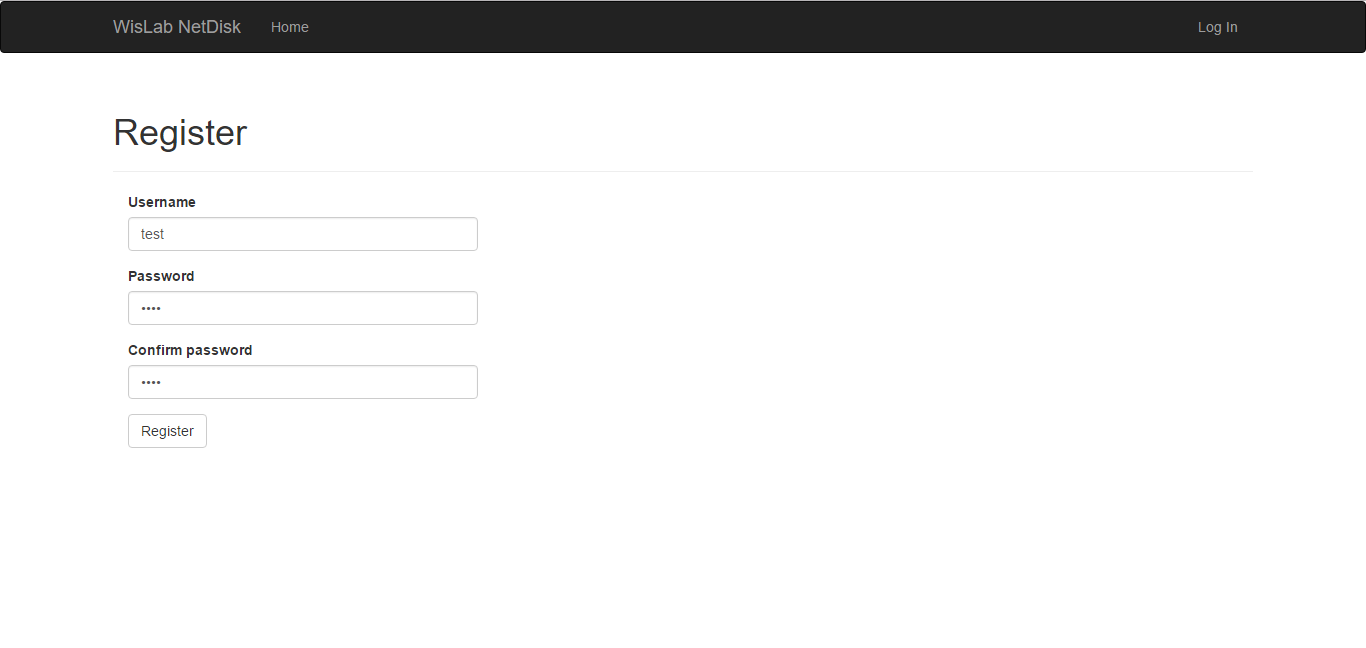


图6.1 注册测试

注册成功：

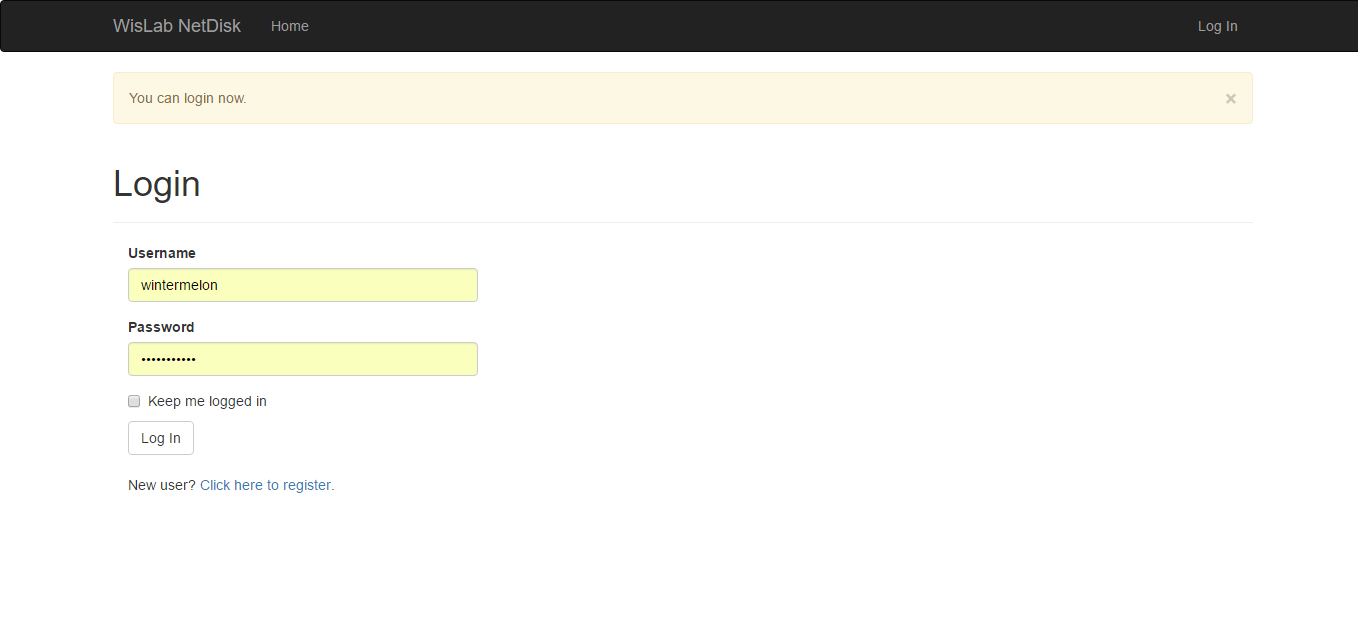


图6.2 注册测试

若注册信息有缺省：

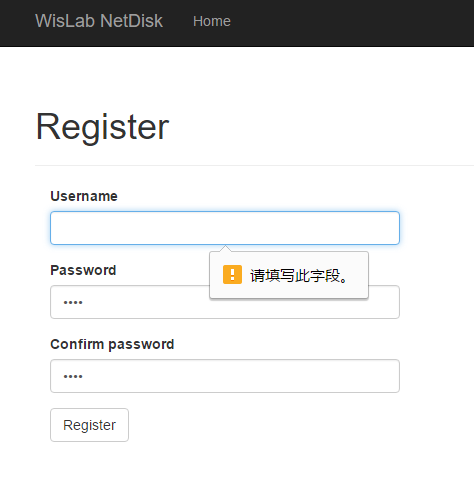
 

图6.3 注册测试

### 登录

输入用户名与密码：

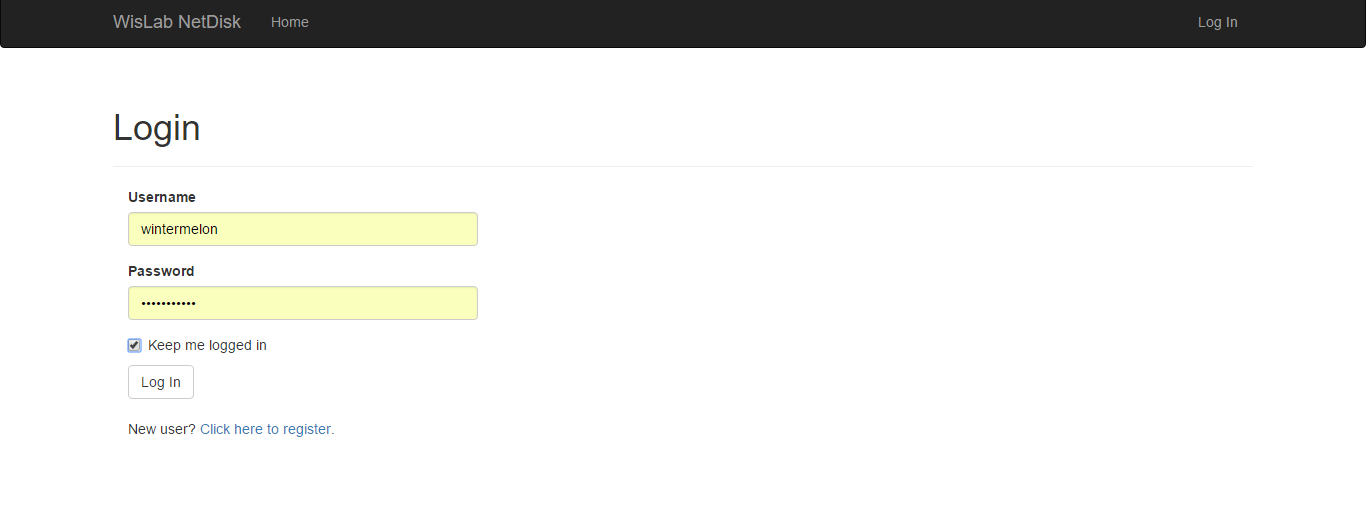


图6.4 登录测试

登录成功：

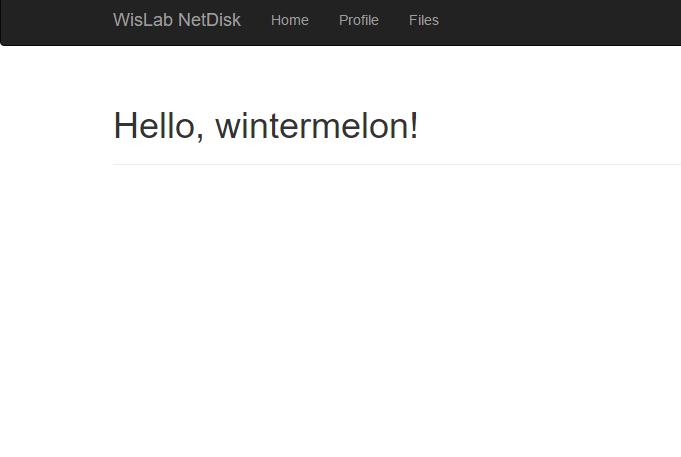


图6.5 登录测试

若输入错误的用户名或密码：

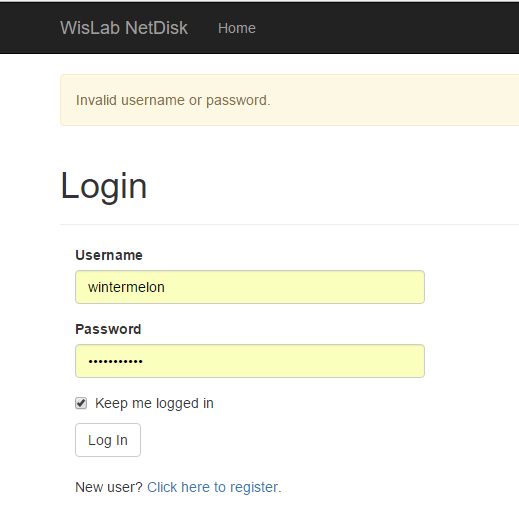


图6.6 登录测试

## 文件上传测试

选择待上传的文件：

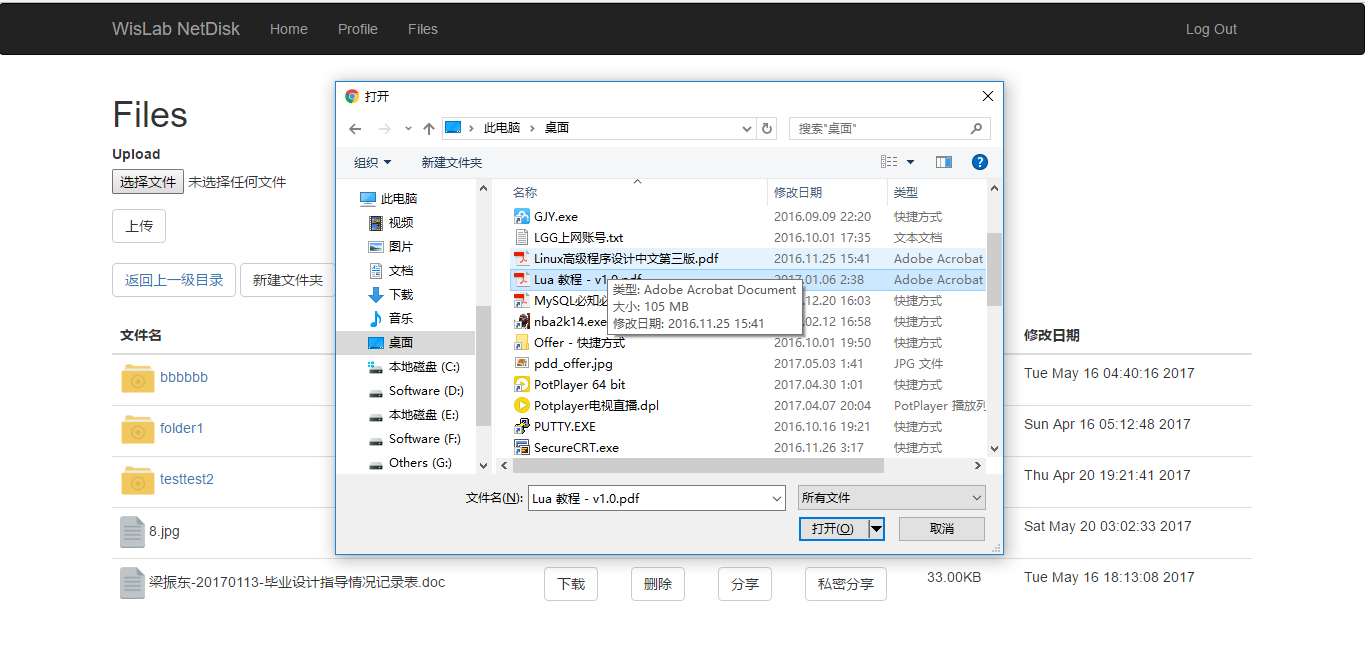


图6.7 选择文件



图6.8 选择文件

若上传成功：



图6.9 上传成功

若文件不符合允许上传的格式：

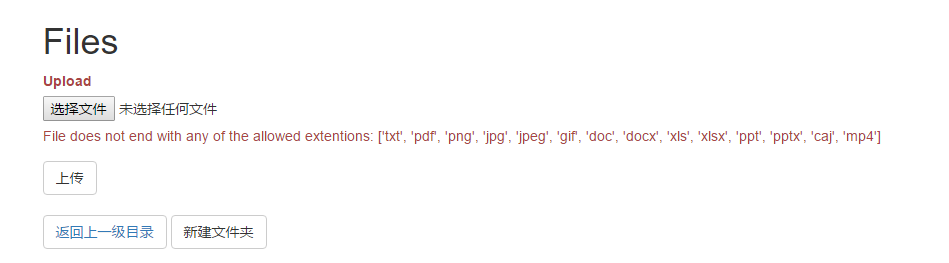


图6.10 上传不符合规则

## 文件下载测试

选择文件进行下载：



图6.11 选择下载文件

下载通过http连接到FastDFS，若文件可浏览,可在浏览器预览文件:



图6.12 文件预览

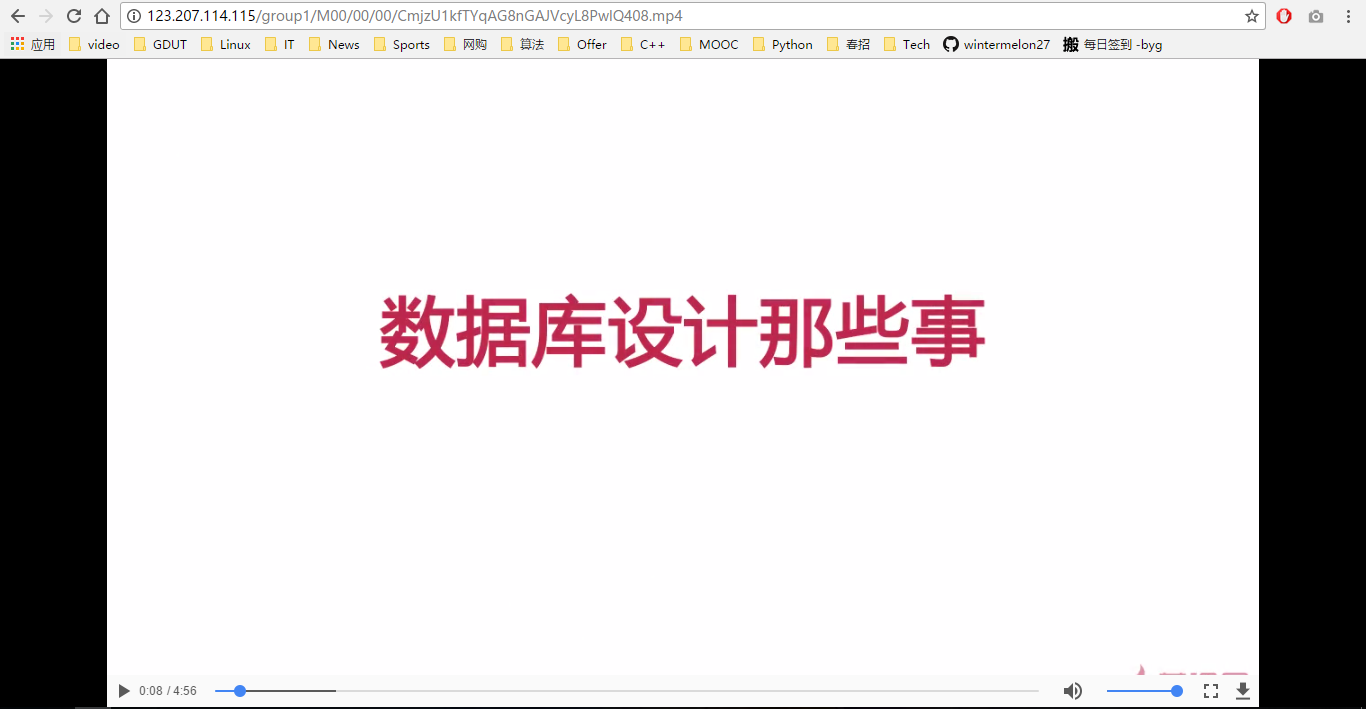


图6.13 文件预览

若浏览器不弹出下载对话框，可手动复制下载链接进行下载。

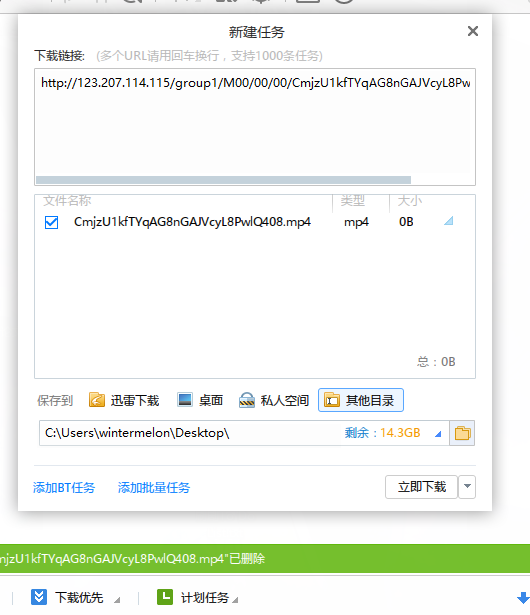


图6.14 下载链接

## 文件分享测试

### 公开分享



图6.15 公开分享

访问分享链接即可获取下载：

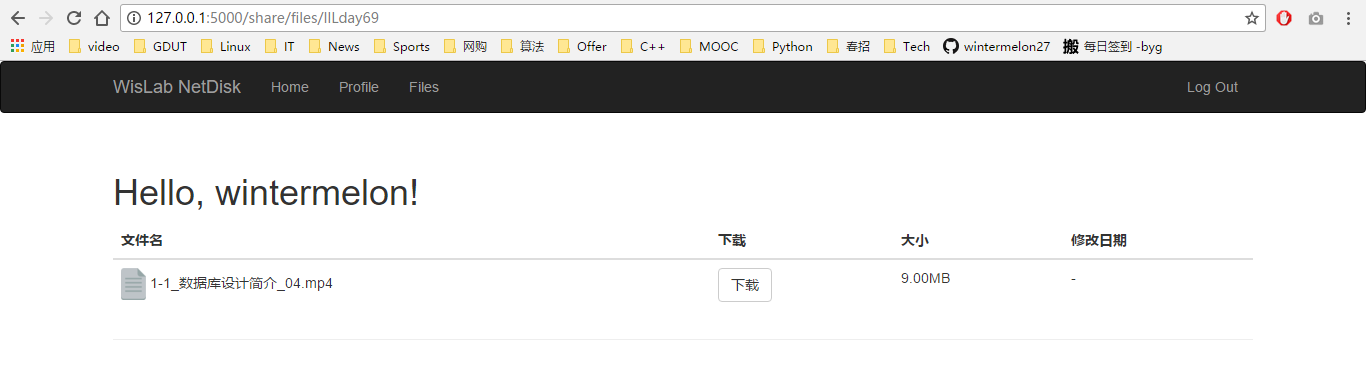


图6.16 访问分享链接

### 私密分享



图6.17 私密分享

访问分享链接时需提供提取码：

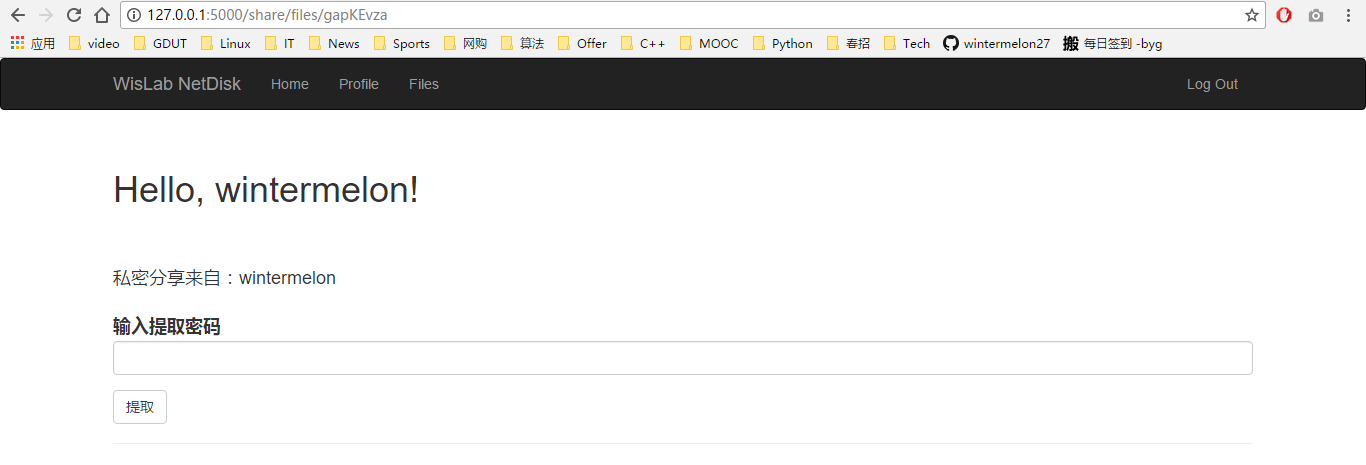


图6.18 访问分享链接

若提取码错误：

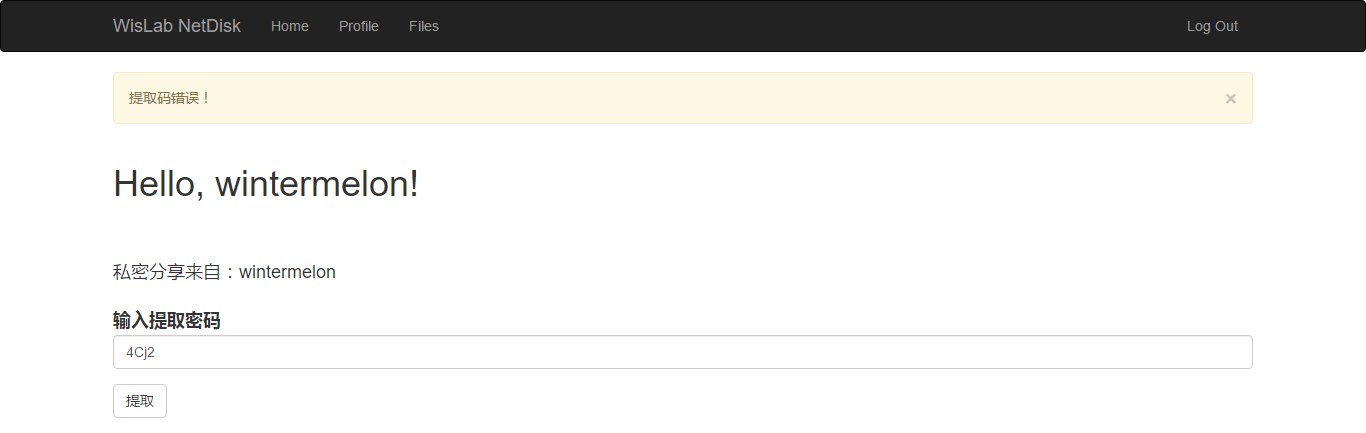


图6.19 提取码错误

提取码正确时即可获取下载：

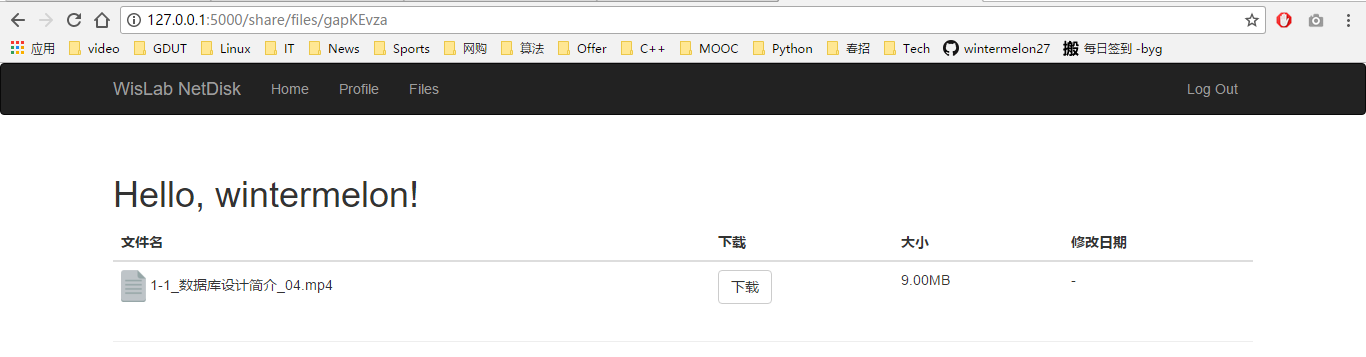


图6.20 提取码正确

若访问无效的分享链接时：



图6.21 分享链接无效

# 参 考 文 献

[1] 谢希仁．计算机网络[M]．北京：电子工业出版社，2013：235-290

[2] 王珊，萨师煊．数据库系统概论[M]．北京：高等教育出版社，2014：205-240

[3] Miguel Grinberg，安道．Flask Web开发[M]．北京：人民邮电出版社，2015：3-74

[4] Magnus Lie Hetland，司维，曾军崴，谭颖华．Python基础教程[M]．北京：人民邮电出版社，2014：26-164，255-273

[5] 李心语．在线网盘系统的设计与实现[D]．吉林：吉林大学，2010．

[6] 杨岳湘，邓文平，邓劲生，李阳．基于云存储的网盘系统架构及关键技术研究[D]．长沙：国防科技大学信息中心， 2012

[7] 沈根海．基于私有云的云存储平台的设计与实现[D]．嘉兴，嘉兴南洋职业技术学院，2015

[8] 何海东，张文秋．基于Web的网络硬盘的设计与实现[D]．四川：四川理工学院，2010

[9] 刘意先，耿鹏涛．基于Web的网盘系统的设计与实现[D]．西安：西安邮电大学信息，2014

[10] W·Richard Stevens，TCP/IP ILLustrated Volume 1: The Protocols[M]．America，2000．

[11] Mark Pilgrim，Dive Into Python[M]．America，2004

# 致 谢

大四的最后这一个多学期以来，毕业设计的学习与开发断断续续进行，历经几个月的学习和刘老师与实验室师兄们的辛勤指导，WisLab网盘系统的开发也完成的差不多。

在这几个月中，通过刘老师的辛勤指导和严格的要求，让我意识到做什么事情都要有责任感和使命感，每天不能松懈，每天都要有进展才能更好的实现预期的目标。刘老师这种严谨的作风不断地鞭策着我，让我在离开校园，投入工作之前，学多了一些对于工作的态度，对于做事情的态度，严谨要求自己总是好的，抓紧时间完成好每一件事总是没有坏处的。

实验室中的师兄也在我的毕业设计中给了很多的帮助，每周的进度汇报会上都会给予很多的帮助与建议，以便更好的继续毕业设计的开发工作。

感谢刘老师与实验室的师兄同学们，在毕业设计的进行过程中，有了一种归属感，至少让我感受到做的东西有意义，有实际意义，可以在实验室的日常生活中使用起来。

最后祝刘老师工作顺利，WisLab发展越来越好。