

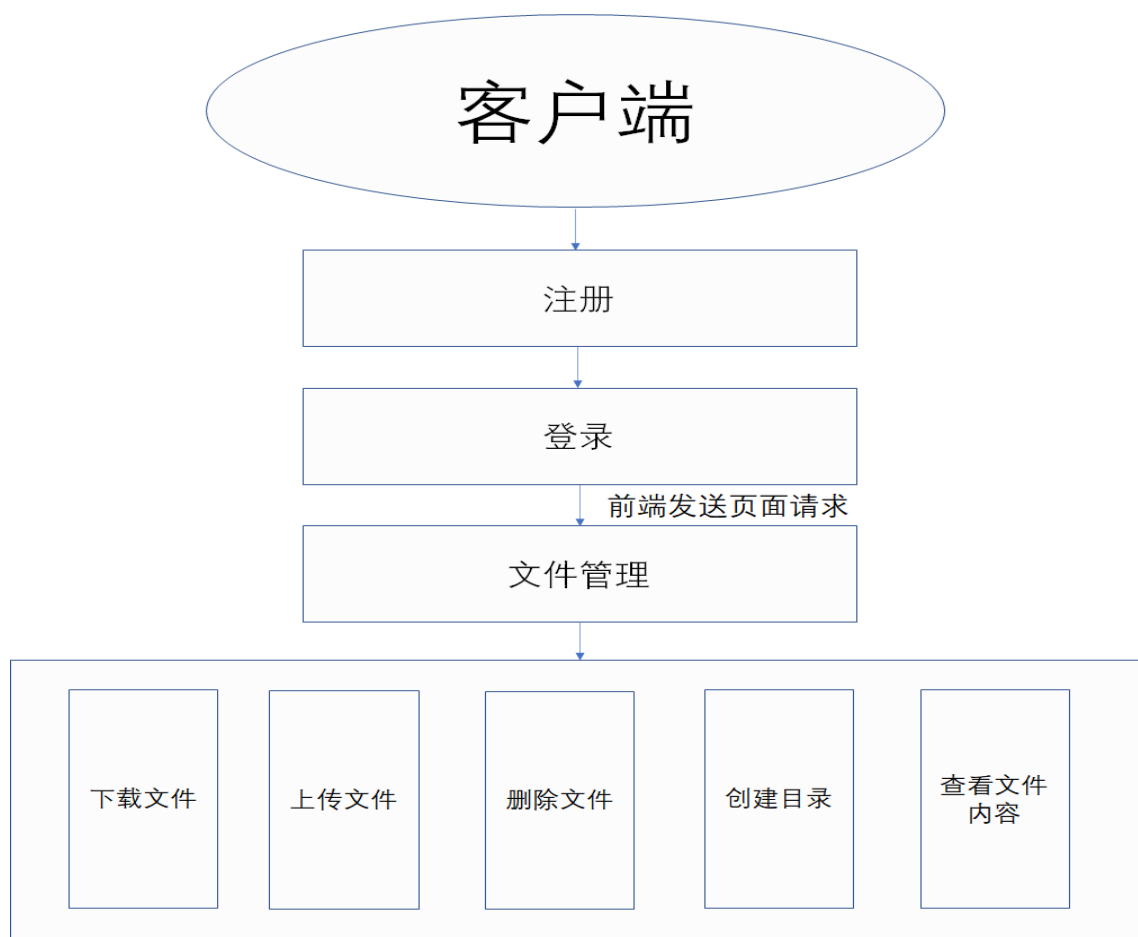
## 目录

一、功能需求概述 .....	1
二、总体设计 .....	2
1、客户端页面设计 .....	2
2、服务器后端控制器操作 .....	2
3、domain 层设计 .....	3
4、dao 层设计 .....	3
5、Server 层操作 .....	3
7、工程目录图 .....	4
8、钉钉内网穿透 .....	5
三、重点模块 .....	5
1、Maven 工程所用到的坐标依赖注入 .....	5
2、springboot 设置上传文件无限制: .....	7
3、读文件操作(浏览器客户端查看和下载): .....	7
4、URL 地址的编码与解码 .....	8
5、获取 hdfs 中存储文件块的长度,实现读的循环 .....	8
6、文件系统 fs 的获取操作 .....	9
7、断点续传 .....	9
8、控制器 .....	10
9、Configuration 配置 .....	15
10、ShiroConfig 类 .....	15
四、重要结果截图 .....	17
五、总结与个人收获体会 .....	20



## 一、功能需求概述

为支持 windows、MAC、Linux 和安卓平台使用云盘系统,所以本次实训使用了 B/S 模式实现了云盘的功能.该云盘基于网络方式,向用户提供了文件的上传、下载、删除、文件夹的创建以及文件内容的查看功能的实现.服务器主要使用了 Springboot 框架来实现,具体调用了 hadoop 自带的第三方 jar 包来实现文件操作的功能.客户端使用了 html、javascript、thymeleaf 来实现网页的动态显示效果,其中在视频播放页面使用了 video.js,完美实现了 mp4 和 flv 格式的视频播放,而且在播放页面使用了断点续传功能实现了视频任意位置的播放功能,倍数播放最高支持 9 倍速播放.在连接数据库方面使用了 mybatis 和 springboot 来进行整合连接实现了用户的注册和登录功能的实现,在云盘的安全方面使用了 Shiro 来实现了每个页面访问之间的页面权限的管理,每个用户之间不能相互看到对方的文件,使用了控制器的重定向来实现,在网络方面使用了钉钉内网穿透工具,实现了可以不在同一个局域网内就能使用本次实验所设计的云盘系统.



## 二、总体设计

### 1、客户端页面设计

借鉴了 `hadoop` 自身的 `html`, 在它的基础上删掉一些不必要的部分, 重新使用了 `javascript` 实现了按钮的功能和页面的跳转功能, `thymeleaf` 实现了重复代码的复用以及接收 `java` 后台返回的数据处理并在前台进行展示, 这次的页面设计使用了 `Bootstrap` 来实现了网页的自适应, 可以在不同平台, 不同屏幕上完美显示网站的内容, 在注册的页面中使用了 `jQuery` 来判断输入的两次密码是否相同来起到密码确认功能的实现。

### 2、服务器后端控制器操作

`@RequestMapping("/main")` 初始化 给前端设计一些数据进行展示  
`@GetMapping("/click")`

页面跳转操作, 根据前端传递过来的参数进行 `api` 操作并返回文件目录信息进行展示, 在这期间要判断点击的是目录还是文件, 如果是视频文件就跳转到

`@GetMapping("/setlook")` 进行视频的播放, 其他文件则跳转到

`@GetMapping("/look_some")`, 而且该控制器还起到每个用户对应于每个根目录的文件夹的控制作用, 自己所登录的用户不能查看别人用户文件的操作。

`@GetMapping("/setlook")` 设置视频播放信息并跳转到视频播放页面。

`@GetMapping("/look")`

根据文件名选择是普通文件查看还是视频文件查看. 视频文件内容的读取, 其中涉及到了断点续传的功能, 该功能可以在客户端视频播放页面视频跳转播放, 该控制器的本质是读文件操作。

`@GetMapping("/look_some")`

除视频文件外其他内容的读取, 发送的内容使用的是客户端自带的可支持格式打开文件, 如果浏览器不支持, 这自动下载文件, 该控制器的本质是读文件操作。

`@GetMapping("/delete")` 删除文件, 根据前端传递过来的参数路径进行 `api` 调用删除

`@GetMapping("/create")` 创造文件目录, 根据前端传递过来的参数路径进行 `api` 调用创造文件目录

`@GetMapping("/download")` 下载文件内容, 读文件操作

`@PostMapping("/upload")` 上传文件, 写文件操作

`@PostMapping("/login")` 登录控制器, 使用了 `Shiro` 来进行网页的认证于授权。

`@RequestMapping("/regist")` 注册控制器

`@RequestMapping("/logout")` 使用了 `subject.logout()`; 来进行了用户的安全退出操作。

具体代码参见重点模块中的控制器

### 3、domain 层设计

- Path\_info 查询 hdfs 文件相关信息,查询后就显示到后台上

private String files1; 除文件名外的路径

private String files2; 文件名

private boolean isfile; 是否是文件

private String BlockSize; 文件的大小

private String path; 文件的具体路径

- User 登录用户的相关信息,实体类对象

private int id

private String username;

private String password;

### 4、dao 层设计

- Path\_infoDao

private List<Pathinfo> pathinfos = null; 对 Path\_info 的一些操作

- UserMapper 接口 使用 Mybatis 进行数据库相关的操作

public User queryUserByName(String name);

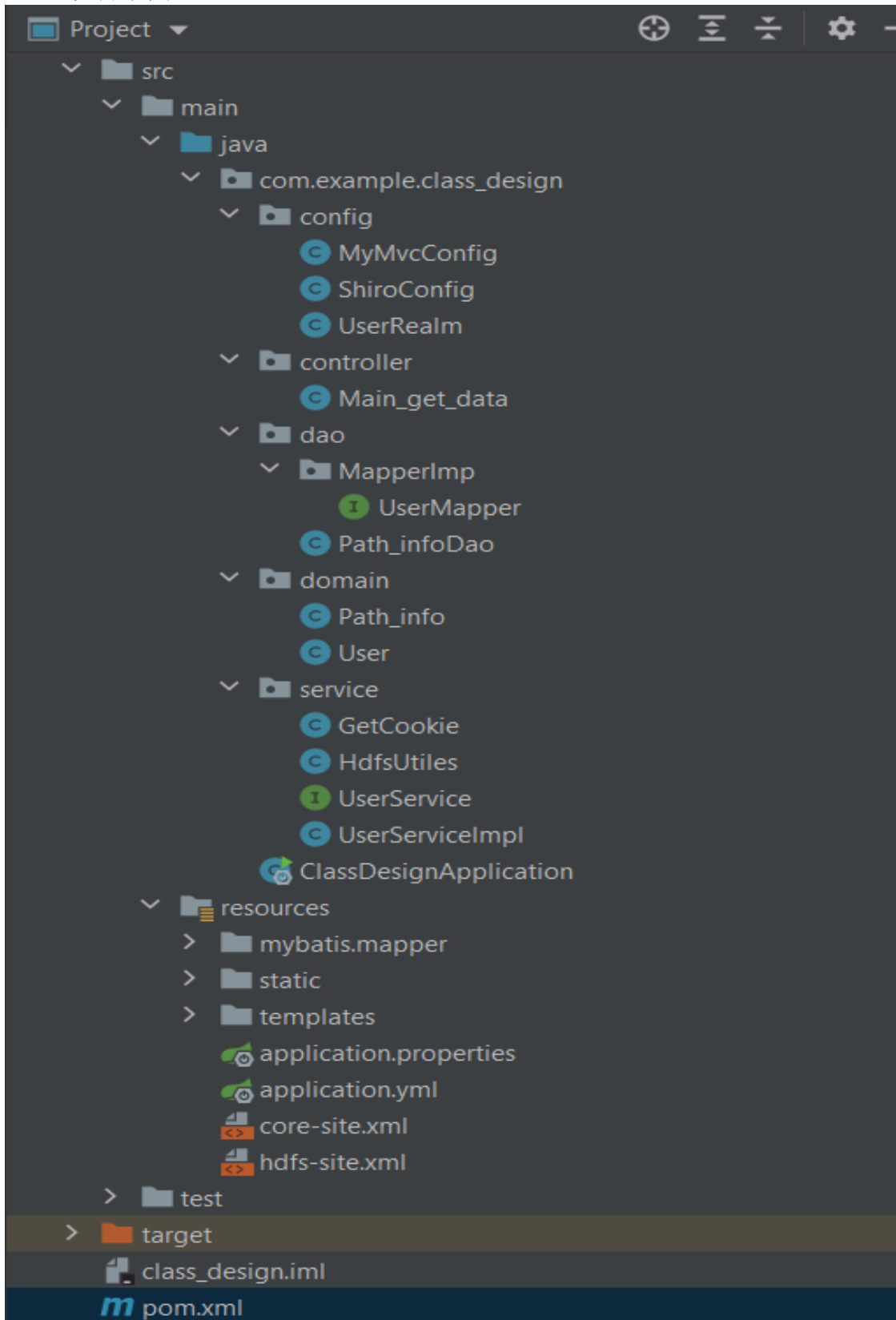
public boolean existUsername(String username);

public void insert\_user(User user);

### 5、Server 层操作

- GetCookie 获取客户端的 Cookie 信息
- HdfsUtils 操作 Hdfs 的工具类
- UserService 数据库操作,连接 dao 层
- UserServiceImpl UserService 的实现类

## 7、工程目录图



## 8、钉钉内网穿透

<https://github.com/open-dingtalk/pierced.git>

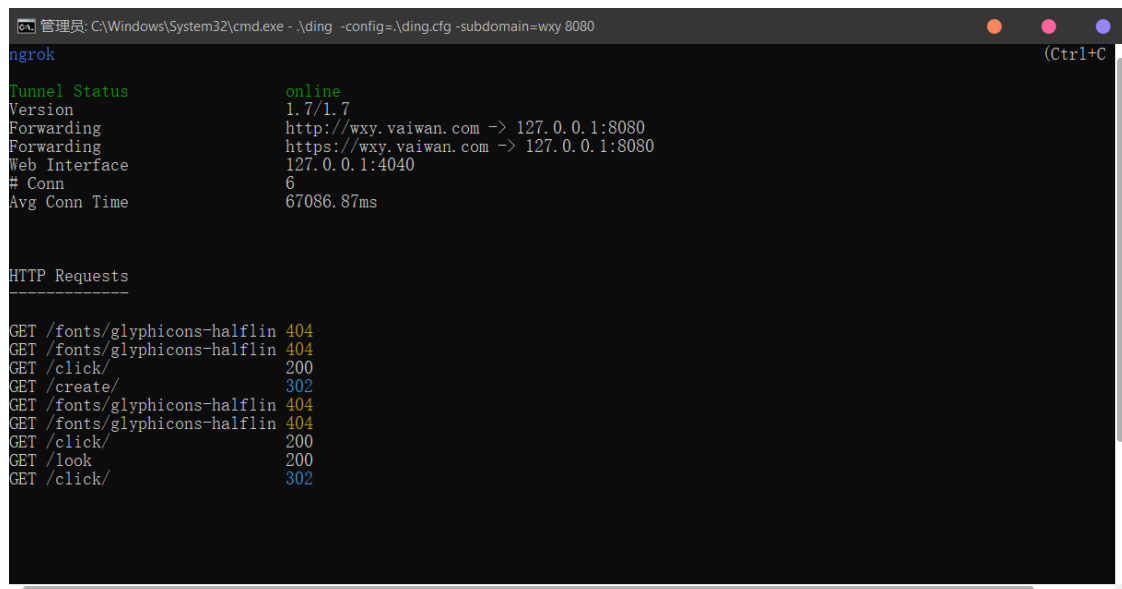
运行钉钉内网穿透工具命令:

Mac:

```
./ding -config=./ding.cfg -subdomain=wxy 8080
```

Windows:

```
.\ding -config=.\ding.cfg -subdomain=wxy 8080
```



```
管理员: C:\Windows\System32\cmd.exe - \ding -config=.\ding.cfg -subdomain=wxy 8080
ngrok
Tunnel Status      online
Version            1.7/1.7
Forwarding          http://wxy.vaiwan.com -> 127.0.0.1:8080
Forwarding          https://wxy.vaiwan.com -> 127.0.0.1:8080
Web Interface       127.0.0.1:4040
# Conn              6
Avg Conn Time       67086.87ms

HTTP Requests
-----
GET /fonts/glyphicons-halflin 404
GET /fonts/glyphicons-halflin 404
GET /click/                     200
GET /create/                    302
GET /fonts/glyphicons-halflin 404
GET /fonts/glyphicons-halflin 404
GET /click/                     200
GET /look                      200
GET /click/                    302
```

## 三、重点模块

### 1、Maven 工程所用到的坐标依赖注入

```
<dependencies>
    <dependency>
        <groupId>org.thymeleaf</groupId>
        <artifactId>thymeleaf-spring5</artifactId>
    </dependency>
    <dependency>
        <groupId>org.thymeleaf.extras</groupId>
        <artifactId>thymeleaf-extras-java8time</artifactId>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>apache-log4j-extras</artifactId>
        <version>1.2.17</version>
    </dependency>
</dependencies>
```

```

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-client</artifactId>
    <version>2.7.7</version>
</dependency>
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-common</artifactId>
    <version>2.7.4</version>
</dependency>
<dependency>
    <groupId>org.apache.velocity</groupId>
    <artifactId>velocity-tools</artifactId>
    <version>2.0</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>2.0.1</version>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
</dependency>
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-spring</artifactId>
    <version>1.5.3</version>
</dependency>

```



```

    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>druid</artifactId>
        <version>1.2.5</version>
    </dependency>
</dependencies>

```

## 2、springboot 设置上传文件无限制:

```

@Bean
public MultipartConfigElement multipartConfigElement() {
    MultipartConfigFactory factory = new MultipartConfigFactory();
    factory.setMaxRequestSize(DataSize.parse("-1"));
    factory.setMaxFileSize(DataSize.parse("-1"));
    return factory.createMultipartConfig();
}

```

## 3、读文件操作(浏览器客户端查看和下载):

```

public static void read(String path,String name, HttpServletResponse rs,
    HttpServletRequest re, boolean flag) throws Exception{
    Path File=new Path(path);
    FSDataInputStream in=HdfsUtiles.getfs().open(File);
    //flag 为true 是在浏览器查看文件内容, 否者为下载文件
    if(flag){
        if("mp4".equals(name)) {
            rs.setContentType("video/mp4");
            show_mp4_and_flv(path, rs, re, in);
        }else if("flv".equals(name)){
            show_mp4_and_flv(path, rs, re, in);
        }
        return;
    }else{
        rs.setContentType("application/octet-stream");
        rs.setHeader("Content-disposition", String.format("attachme
nt;filename=\"%s\"", path.substring(path.lastIndexOf('/')+1)));
    }
    List<Integer> size=HdfsUtiles.blks(HdfsUtiles.getfs(), path);
    OutputStream out=rs.getOutputStream();
    for (Integer i:size) {
        IOUtils.copyBytes(in, out, i);
    }
    out.close();
}

```

#### 4、URL 地址的编码与解码

```
public static String Url_decode(String path) {
    try {
        path = java.net.URLDecoder.decode(path, "UTF-8");
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
    return path;
}
public static String Url_encode(String path) {
    try {
        path = java.net.URLEncoder.encode(path, "UTF-8");
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
    return path;
}
```

前端 javascript 的实现:

```
window.onload = function () {
    var a = document.getElementsByTagName("a");
    for(var i = 0, len = a.length; i < len; i++){
        a[i].href = encodeURI(a[i].href);
    }
}
```

#### 5、获取 hdfs 中存储文件块的长度,实现读的循环

```
public static List<Integer> blks(FileSystem fs, String path) throws Exception{
    Path p = new Path(path);
    FileStatus ifile = fs.getFileStatus(p); // 获取文件信息
    BlockLocation[] blks = fs.getFileBlockLocations(ifile, 0, ifile.getLength()); // 获取块信息
    List<Integer> sum = new ArrayList<>();
    for (BlockLocation b:blks) {
        //b 偏移量, 存放字节数, 副本策略
        sum.add((int) b.getLength());
    }
    return sum;
}
```

## 6、文件系统 fs 的获取操作

```
private static FileSystem fs=null;
static {
    try {
        Configuration conf = new Configuration(); // 读取配置文件
        fs = FileSystem.get(conf); // 获取客户端，
文件系统
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public static FileSystem getfs() throws Exception {
    return fs;
}
```

## 7、断点续传

```
public static void show_mp4_and_flv(String path, HttpServletResponse rs,
HttpServletRequest re, FSDataInputStream in) throws Exception{
    rs.setHeader("Accept-Ranges", "bytes"); //表示支持断点续传
    long p = 0L;
    long contentLength = 0L;

    // 0, 从头开始的全文下载; 1, 从某字节开始的 (bytes=27000-)
    int rangeSwitch = 0;

    long fileLength=0;
    String rangBytes = "";
    List<Integer> size=HdfsUtiles.blks(HdfsUtiles.getfs(), path);
    for (Integer i:size) {
        fileLength += i;
    }
    String range = re.getHeader("Range");
    if (range != null && range.trim().length() > 0 && !"null".equal
s(range)) {
// 200 是 OK (一切正常), 206 是 Partial Content (服务器已经成功处理了部分内
容),
// 416 Requested Range Not Satisfiable (对方(客户端)发来的Range 请求头
不合理)。
        rs.setStatus(javax.servlet.http.HttpServletResponse.SC_PART
IAL_CONTENT);
        rangBytes = range.replaceAll("bytes=", "");
        if (rangBytes.endsWith("-")) { // bytes=270000-
            rangeSwitch = 1;
            p = Long.parseLong(rangBytes.substring(0, rangBytes.ind
exOf("-")));
            contentLength = fileLength - p; // 客户端请求的是 270000
        }
    }
}
```

之后的字节（包括bytes 下标索引为270000 的字节）

```
    }
    } else {
        contentLength = fileLength;
    }
    rs.setHeader("Content-Length", new Long(contentLength).toString
());
    OutputStream out=rs.getOutputStream();
    if (rangeSwitch == 1) { // 1,从某字节开始的 (bytes=27000-); 2,
从某字节开始到某字节结束的 (bytes=27000-39000)
        String contentRange = new StringBuffer("bytes ").append(new
Long(p).toString()).append("-")
            .append(new Long(fileLength - 1).toString()).append
("/")
            .append(new Long(fileLength).toString()).toString();
        rs.setHeader("Content-Range", contentRange);
        long start=p;
        // IOUtils.copyBytes(in, out, start, fileLength-p);
        IOUtils.skipFully(in, p);
        try {
            IOUtils.copyBytes(in, out, (int) (fileLength - p));
        } catch (Exception e){
            out.close();
            return;
        }
    } else {
        String contentRange = new StringBuffer("bytes ").append("0-
").append(fileLength - 1).append("/")
            .append(fileLength).toString();
        rs.setHeader("Content-Range", contentRange);
        for (Integer i:size) {
            IOUtils.copyBytes(in, out, i);
        }
    }
    out.close();
}
```

## 8、控制器

```
@Controller
@CrossOrigin
public class Main_get_data {
    @Autowired
    UserServiceImpl userServiceImpl;

    @RequestMapping("/main")
    public String sent_table_data(Model mode, HttpServletRequest re){
        try {
            Path_infoDao tables_data = HdfsUtiles.list_catalog("/");
```

```

        System.out.println(tables_data);
        mode.addAttribute("get_datas", tables_data.getPath_infos());
        mode.addAttribute("path", tables_data.getPath_infos().get
(0).getFiles1());
        mode.addAttribute("bring_path", "/");
    }catch (Exception e){
        e.printStackTrace();
    }finally {
        return "redirect:/click?path="+GetCookie.getCook(re);
    }
}

@GetMapping("/click")
public String click_show(@RequestParam("path")String path, Model mo
de, HttpServletRequest re){
    path = HdfsUtiles.Url_decode(path);
    String root_path = GetCookie.getCook(re);

    if(path.length()>=root_path.length()){
        if(! path.substring(0,root_path.length()).equals(root_path))
{
            path = root_path + path;
        }
    }else {
        path = root_path + path;
    }
    try {
        if(!HdfsUtiles.getfs().exists(new Path(path))){
            mode.addAttribute("info_exists", "路径不存在!!!");
            return "hadoop";
        }
        Path_infoDao tables_data = HdfsUtiles.list_catalog(path);
        mode.addAttribute("get_datas", tables_data.getPath_infos());
        for(Path_info a:tables_data.getPath_infos()){
            if(a.getPath().substring(9).equals(path) && a.isIsfile
()){
                String name = path.substring(path.lastIndexOf('.')+
1);

                path = HdfsUtiles.Url_encode(path);
                if("mp4".equals(name) || "flv".equals(name)) {
                    return "redirect:/setlook?path=" + path;
                }else{
                    return "redirect:/look_some?path=" + path;
                }
            }
        }
        mode.addAttribute("path", path);
        if (path.equals("/")){
            mode.addAttribute("bring_path", '/');
        }else{

```

```

        mode.addAttribute("bring_path", path+'/');
    }
} catch (Exception e){
    e.printStackTrace();
}
return "hadoop";
}

@GetMapping("/setlook")
public String setlook(@RequestParam("path")String path, HttpServletRequest
Request re, HttpServletResponse rs, Model model) throws Exception{
    model.addAttribute("path",path);
    model.addAttribute("type",path.substring(path.lastIndexOf('.')+
1));
    return "player";
}

@GetMapping("/look")
public String concon(@RequestParam("path")String path, HttpServletRequest
equest re, HttpServletResponse rs, Model model) throws Exception{
    path = HdfsUtiles.Url_decode(path);
    String name="";
    try{
        name = path.substring(path.lastIndexOf('.')+1);
    } catch (Exception e){
        path = HdfsUtiles.Url_encode(path);
        return "redirect:/look_some/?path="+path;
    }
    HdfsUtiles.read(path,name, rs, re, true);
    return "redirect:/player.html";
}

@GetMapping("/look_some")
@ResponseBody
public String look_some(@RequestParam("path")String path, HttpServl
etRequest re, HttpServletResponse rs, Model model) throws Exception{
    path = HdfsUtiles.Url_decode(path);
    String name = path.substring(path.lastIndexOf('.')+1);
    Path File=new Path(path);
    FSDataInputStream in=HdfsUtiles.getfs().open(File);
    if("mp3".equals(name)){
        rs.setContentType("video/mp4");
    }
    else if("pdf".equals(name)){
        rs.setContentType("application/pdf");
    }
    else if("bmp".equals(name)){
        rs.setContentType("image/bmp");
    }
    else if("gif".equals(name)){

```

```

        rs.setContentType("image/gif");
    }
    else if("jpeg".equals(name)){
        rs.setContentType("image/jpeg");
    }else if("png".equals(name)){
        rs.setContentType("image/png");
    }
    else{
        rs.setContentType("text/plain; charset=utf-8");
    }
    List<Integer> size=HdfsUtils.blks(HdfsUtils.getfs(), path);
    OutputStream out=rs.getOutputStream();
    for (Integer i:size) {
        IOUtils.copyBytes(in, out, i);
    }
    out.close();
    return "";
}

@GetMapping("/delete")
public String delete(@RequestParam("path")String path, HttpServletRequest re, HttpServletResponse rs) throws Exception{
    path = HdfsUtils.Url_decode(path);
    HdfsUtils.getfs().delete(new Path("hdfs://ns"+path));
    int count=0;
    for(char a:path.toCharArray()){
        if(a=='/'){
            count++;
            if (count > 1)
                break;
        }
    }
    if(count==1){
        return "redirect:/";
    }
    return "redirect:/click/?path="+path.substring(0, path.lastIndexOf('/'));
}

@GetMapping("/create")
public String create(@RequestParam("path")String path, HttpServletRequest re, HttpServletResponse rs) throws Exception{
    path = HdfsUtils.Url_decode(path);
    HdfsUtils.getfs().mkdirs(new Path(path));
    System.out.println(path);
    return "redirect:/click/?path="+path;
}

@GetMapping("/download")
public String download(@RequestParam("path")String path, HttpServlet

```

```

tRequest re, HttpServletResponse rs) throws Exception{
    path = HdfsUtils.Url_decode(path);
    HdfsUtils.read(path, "", rs, re, false);
    return "";
}

@PostMapping("/upload")
public String upload(@RequestParam("path")String path, @RequestParam("file") MultipartFile srcFile[], RedirectAttributes redirectAttributes) throws Exception{
    path = HdfsUtils.Url_decode(path);
    for (MultipartFile file : srcFile) {
        System.out.println(file.getOriginalFilename());
        Path File=new Path("hdfs://ns/"+path+'/'+file.getOriginalFilename());
        FSDataOutputStream out=HdfsUtils.getfs().create(File);
        FileCopyUtils.copy(file.getInputStream(),out);
    }
    return "redirect:/click/?path="+path;
}

@RequestMapping("/login")
public String login(String username, String password, Boolean rememberMe, Model model, HttpServletResponse rs, HttpServletRequest re){
    Subject subject = SecurityUtils.getSubject();
    UsernamePasswordToken token = new UsernamePasswordToken(username, password, rememberMe);
    System.out.println("rememberMe:"+rememberMe);
    try{
        subject.login(token);
        Cookie cookie=new Cookie("cookie", username);
        cookie.setMaxAge(365*24*60*60);
        rs.addCookie(cookie);
        System.out.println("remember:"+subject.isRemembered()+token.isRememberMe());
        return "redirect:/main";
    }catch (UnknownAccountException e){
        model.addAttribute("msg","用户名不存在!!!");
        return "index";
    }catch (IncorrectCredentialsException e){
        model.addAttribute("msg","密码错误!!!");
        return "index";
    }
}

@RequestMapping("/noauth")
@ResponseBody
public String unauthorized(){
    return "未经授权无法访问此页面";
}

```



```

    }

    @RequestMapping("/logout")
    public String logout() {
        Subject subject = SecurityUtils.getSubject();
        subject.logout();
        return "index";
    }

    @RequestMapping("/regist")
    public String regist(String username, String password, Model model)
    {
        if(userServiceImpl.exixtUsername(username)){
            model.addAttribute("exit", "用户名已经存在");
            return "reset";
        }
        User user = new User();
        user.setUsername(username);
        user.setPassword(password);
        userServiceImpl.insert_user(user);
        return "index";
    }
}

```

## 9、Configuration 配置

```

@Configuration
public class MyMvcConfig implements WebMvcConfigurer {
    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/index.html").setViewName("index");
        registry.addViewController("/reset.html").setViewName("reset");
        registry.addViewController("/player.html").setViewName("player
    );
    }
}

```

## 10、ShiroConfig 类

```

@Configuration
public class ShiroConfig {
    @Bean
    public ShiroFilterFactoryBean getShiroFilterFactoryBean(@Qualifier
("securityManager") DefaultWebSecurityManager defaultWebSecurityManager)
    {
        ShiroFilterFactoryBean bean = new ShiroFilterFactoryBean();
        bean.setSecurityManager(defaultWebSecurityManager);
        Map<String, String> filterMap = new LinkedHashMap<>();
    }
}

```

```

        bean.setFilterChainDefinitionMap(filterMap);
        filterMap.put("/login", "anon");
        filterMap.put("/noauth", "anon");
        filterMap.put("/regist", "anon");
        filterMap.put("/reset.html", "anon");
        filterMap.put("/", "anon");
        filterMap.put("/css/**", "anon");
        filterMap.put("/js/**", "anon");
        filterMap.put("/**", "user");
        bean.setUnauthorizedUrl("/noauth");
        bean.setLoginUrl("/");
        return bean;
    }

    @Bean(name="securityManager")
    public DefaultWebSecurityManager getDefaultWebSecurityManager(@Qualifier("userRealm") UserRealm userRealm, @Qualifier("rememberMeManager")
RememberMeManager rememberMeManager){
        DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();
        securityManager.setRealm(userRealm);
        securityManager.setRememberMeManager(rememberMeManager);
        return securityManager;
    }

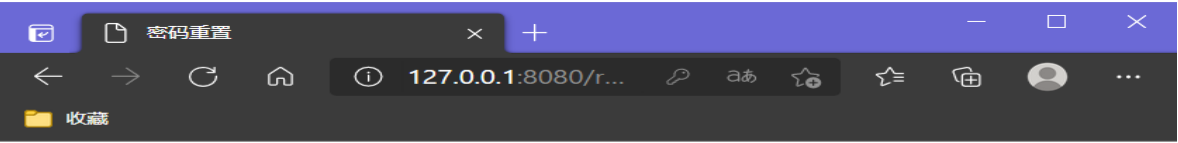
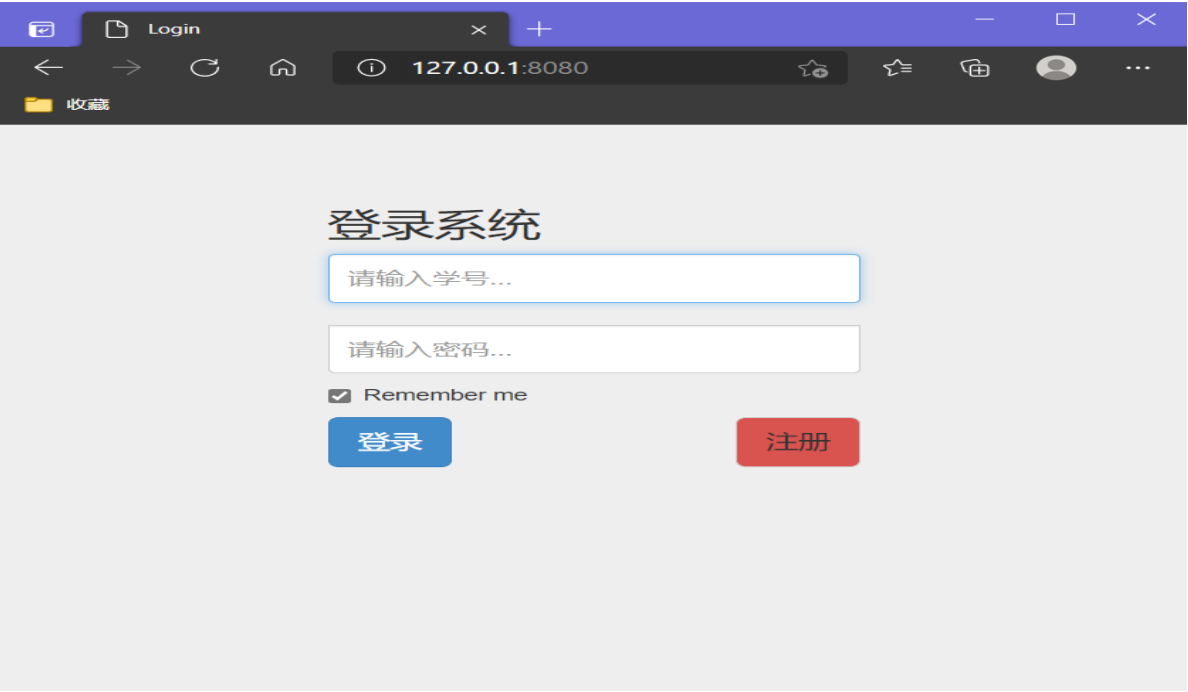
    @Bean(name="userRealm")
    public UserRealm userRealm(){
        return new UserRealm();
    }

    @Bean(name = "rememberMeManager")
    public CookieRememberMeManager rememberMeManager(){
        CookieRememberMeManager cookieRememberMeManager = new CookieRememberMeManager();
        cookieRememberMeManager.setCookie(rememberMeCookie());
        cookieRememberMeManager.setCipherKey(Base64.decode("4AvVhmFLUs0KTA3Kprsdaq=="));
        return cookieRememberMeManager;
    }

    @Bean
    public SimpleCookie rememberMeCookie() {
        SimpleCookie cookie = new SimpleCookie("rememberMe");
        cookie.setHttpOnly(true);
        cookie.setMaxAge(1 * 60 * 60);
        return cookie;
    }
}

```

四、重要结果截图



注册

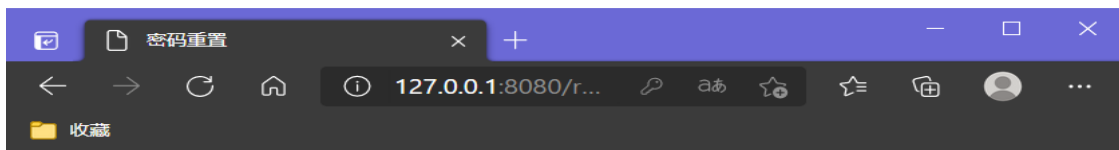
123

...

..

两次输入的密码不一样!!!

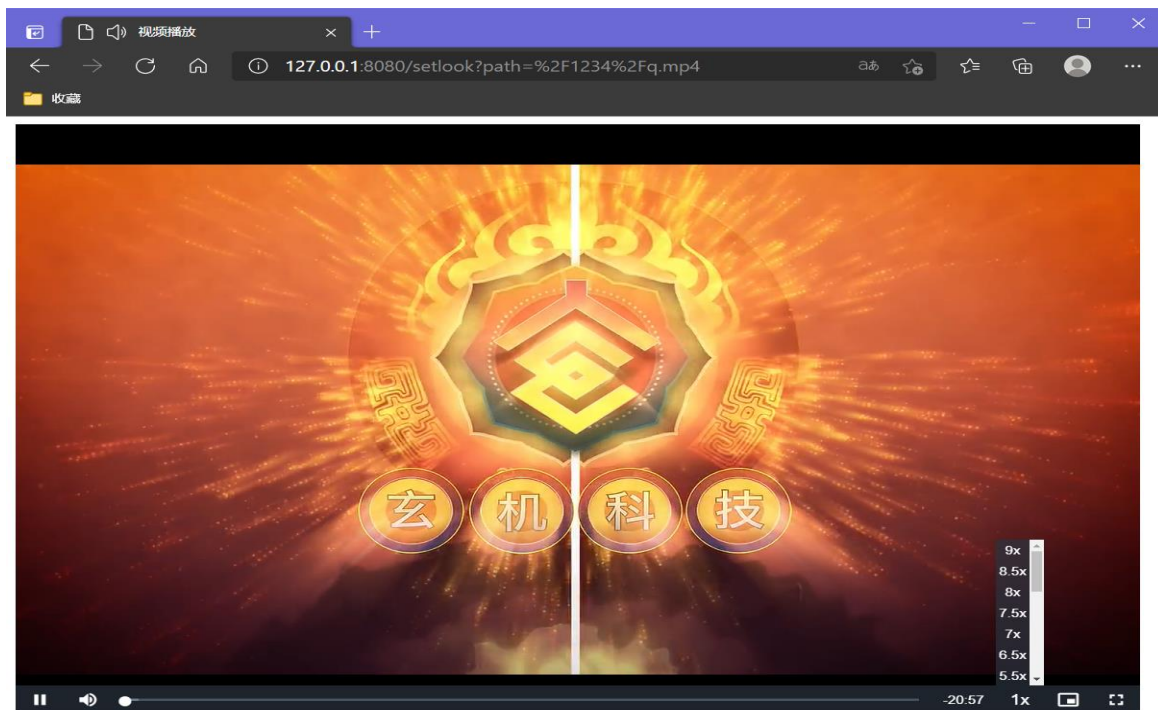
提交

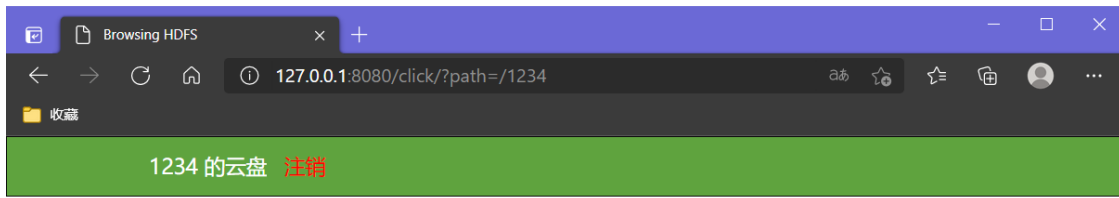


## 注册

用户名已经存在

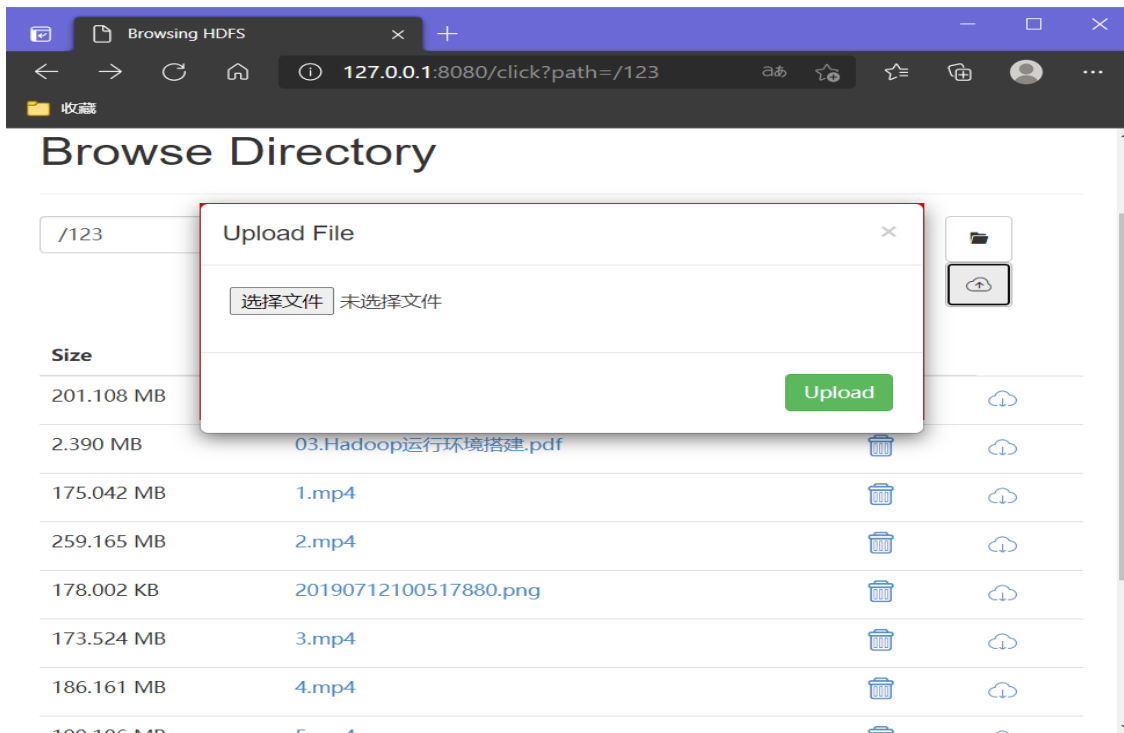
提交





## Browse Directory

<input type="text" value="/1234"/>		<input type="button" value="Go!"/>	<div><div></div><div></div></div>	
Size	Name			
594.629 MB	任务44: 本地提权.flv			
78.550 MB	利用Python进行数据分析.pdf			
178.002 KB	20190712100517880.png			
203.185 KB	heart.png			
201.108 MB	q.mp4			



## 五、总结与个人收获体会

这次的 **hadoop** 应用实训完成得很美满,以前的我很早就想搞一个云盘系统了,自己存资料放视频在上面,只要有网就能够很方便的进行访问,而且访问也不会受到网络的限制,这次的实验在视频播放方面花了我很长时间,我们知道网页自带的播放功能不支持 **flv** 格式的视频播放,但是我就是想要支持 **flv** 格式的播放,然后我就在网上找到了 **video.js** 的播放器,要在上面播放视频,控制器的类上要加上跨域(**CORS**)支持 **@CrossOrigin** 的注解, **video.js** 也能支持倍数播放,看视频的时候就能快进了,刚开始在服务器读取视频内容发给客户端的时候,只能从头开始播放,不能跳着播放,这是因为在服务器端每次的请求文件都是重头开始向客户端发送的内容,没有实现在客户端想在哪个位置要数据,服务器就返回指定位置内容的信息给客户端,所以我就在服务器端写了一个断点续传的功能,实现了在客户端每次都不需要都是从头开始看起了。

这次实验收获最大的就是能够使用 **springboot** 框架来开发项目,以前我用的是 **tomcat** 来开发的网页,后来听说 **springboot** 配置简单、很容易就能上手,然后就抱着试试看的心态,在网上找了一些视频资源就开始学了起来,刚看了前面的几集视频就把网页搭建起来了,而且现在使用的 **springboot** 感觉还是蛮顺手的,环境搭起来后最重要的就是能够使用 **hadoop** 的 **jar** 包来操作 **hadoop** 的文件系统了,看了网上的资料,导入 **hadoop-client** 和 **hadoop-common** 的 **maven** 坐标,把虚拟机中的 **core-site.xml** 和 **hdfs-site.xml** 导入 **resources** 中,就可以操作 **hadoop** 中的 **api** 了。

在实验的过程中遇到的困难就是 **url** 中出现中文的情况了,对于这种情况,要在前端使用 **javascript** 对 **url** 进行编码,点击连接进入后台的控制器,再进行解码,就不会出现乱码了,如果在后端进行地址的重定位,带有中文的参数也要进行编码。

还有就是上传文件和下载文件的操作,其实就是文件的读和写,要考虑到能完整的把文件上传和下载,刚开始的时候没有考虑到这方面的内容,上传文件使用了 **MultipartFile** 的 **getBytes()** 方法一次性读取文件的内容,这就会导致在客户端上传很大文件的时候,超过了 **byte** 数组的长度,导致程序报错,解决方法就是使用了 **FileCopyUtils.copy(file.getInputStream(),out)**,来把客户端传递过来的文件拷贝到 **out** 中实现文件的上传,而且读的时候也要考虑这方面的情况,要循环着读来防止数组的越界。

其次就是要熟悉 **Shiro** 的使用,**Shiro** 的功能是非常的强大的,刚开始使用的时候还不是很娴熟,然后慢慢的琢磨,发现也没那么的困难了,主要是认证、授权、安全退出、记住我等的一些安全方面的功能。

还有就是 **thmleaf** 方面的一些知识:

`mode.addAttribute("getdatas", tablesdata.getPathinfos());`实现后端向前端发送数据,前端可以以循环的方式来进行接收 `th:each="data:${getdatas}"`,接收后显示数据 `th:text="${data.getFiles2()}"`,还有就是前端的重复代码块复用操作使用了 `th:fragment=""`和

th:insert=~{common::main}">来进行操作.后端重定向 return "redirect:路径" 转发  
return "forward:路径"

最后感谢老师的细心指导.