# Netty+Spring Boot仿微信全栈开发高性能后台及客户端

# Netty

<https://blog.csdn.net/qq_26975307/article/details/85004424>

是一个提供了易于使用的api的客户端/服务器框架

并发高 – NIO 非阻塞IO

传输快 零拷贝

简单来讲，Netty是一个提供了易于使用的API的客户端/服务端框架。Netty并发非常高，一个非阻塞的IO，Netty传输速度也非常快，因为他是0拷贝，什么是零拷贝？NIO中的特性之一就是零拷贝，在Java中，内存分为堆和栈以及字符串常量值等等，如果有一些数据从IO中读取并且放到堆里面，中间会经过一些缓冲区。

具体来讲，如果要从IO中读取数据，分为两个步骤：

(1）从IO流中读取出来放到缓冲区，程序从缓冲区中读取，再放到堆中，此时数据就会被拷贝两次才能到达堆或者堆内存中。如果数据量很大，那么就会造成资源的浪费

(2）Netty其实就是利用NIO中的零拷贝特性，当Netty需要接收数据或者传输数据的时候，就会新开辟一块堆内存，然后数据就直接通过IO读取到了新开辟的堆内存中，这样也就加快了数据传输的速度。

<https://netty.io/wiki/user-guide-for-4.x.html>

## 阻塞和非阻塞

线程访问资源，该资源是否准备就绪的一种处理方式

线程在访问某一个资源的时候，该资源是否准备就绪的一种处理方式，如果说该资源当前没准备就绪，这个时候就会有两种处理方式：阻塞与非阻塞

（1）阻塞：这个线程会一直持续等待这个资源就绪并处理完毕，直到响应返回一个结果，这个时候线程是一直阻塞状态，不可以去做任何事情

（2）非阻塞：这个线程直接返回结果，不会持续等待这个资源处理完毕才响应，它会去请求别的资源。



## 同步和异步

是访问数据的一种机制

同步：主动请求，等待io操作完了会通知

异步：主动请求可以发起其他请求，异步通知

这里的 “同步与异步” 指的是访问数据的一种机制，类似于Ajax。  
    （1）同步：主动请求，并且会等待IO操作完成之后，IO会有一个通知  
    （2）异步：当一个线程主动请求数据之后，可以继续处理其他任务，发起其他请求，多个请求完成之后再逐一的通过异步形式通知



## BIO

（1）同步式阻塞IO：Block IO，IO在进行读取的时候，这个线程是会被阻塞的，无法去做其他操作，传统简单，通信方式也是如此。并发处理能力非常低，线程之间访问资源通信的时候，耗时也非常久，所以会比较依赖于网速与带宽，JDK1.4之前均是如此。

（2）服务器会有一个专门的线程称之为Acceptor，专门用于负责监听来自客户端之间的请求，只要客户端与服务端有建立请求，此时客户端与服务端之间都会创建一个新的线程进行处理，这是一种典型的 一应一答的模式。随着客户端逐渐增多，两者之间会频繁的创建和频繁销毁线程，此时服务器端会有很大的压力，甚至宕机。

（3）改进之后则通过线程池来处理，这种方式也可以称之为伪异步IO。

同步阻塞IO Block Io

Io在进行读写，这个线程是被阻塞，无法进行其他的操作 并发能力很低 资源间通信慢 依赖贷款网速



一应一答 客户端逐渐增多，频繁创建线程，销毁线程 很差劲 改良之后，就是线程池 伪异步io 这个bio就像是上厕所找不到位置一直等着

## NIO

（1）同步非阻塞IO：New IO（Non-Block IO）：JDK1.4之后。selector（其实就是一个线程，并且会主动轮询），也称之为多路复用器Buffer，它是一个缓冲区；Chanel，是一个双向通道。

（2）客户端在与服务端进行通信的时候就会使用到selector，两者要建立链接的时候，客户端到selector进行一个注册，注册完毕之后就会有一个chanel01，每一个客户端和selector建立连接之后都会有一个chanel，chanel是一个双向通道，可以进行一些相应数据的读写，这些数据的读写都会到Buffer缓冲区中

（3）通过selector注册并建立chanel通道就可以实现两者之间的通信，chanel的数据是一种非阻塞的读取，如果没有数据会直接跳过，不会同步等待数据。selector其实一个单线程，整体来讲，线程开销会非常小，光是一个selector就可以处理成千上万个客户端，客户端的增多不会影响它的性能，这也是与BIO的区别所在。

（4）chanel相当于是一个读取的工具，每一个客户端都可以理解为一个单独的chanel，每一个服务端会有一个selector，Buffer的数据会进行读取，数据被读完之后还是会存在Buffer中，不会因为数据被读取之后会被消息，String中的数据读完之后就没有了。

同步非阻塞io

去上厕所，发现没坑了。就开始玩手机，时不时看看有没有坑，不是光等着。

Selector 选择器，轮询



## AIO

异步非阻塞io 相比下面不等着，在做自己的事情

异步阻塞io 几乎用不到 比如上厕所，没位置了。就等着。就让哪个人好了告诉我我再去。这种方式非常傻

（1）异步非阻塞IO：AIO其实也是NIO2.0，它是一种异步非阻塞的通信方式，在NIO没有的基础上引入了一个异步概念：在读写的时候所有返回的类型其实就是一个feature对象，这个对象模型其实就是异步的在这个过程中会有一些事件监听  
（2）异步阻塞IO：几乎用不到





## Netty

Java 的开源框架 网络服务器客户端框架

使用netty的优点 java开源框架 原始nio有bug存在。

## Reactor线程模型

（1）Reactor线程模型：

1）单线程模型：所有的IO操作都由同一个NIO线程处理，仅限于一些小型应用场景。但在高负载、高并发等情况下使用单线程肯定就不太合理，主要是因为NIO的一个线程同时要去处理成千上万的请求 的时候，在性能上会支撑不了，即便CPU负载100%，对于海量消息的处理，编码解码以及读取、发送消息等情况，依然满足不了。

2）当NIO的线程负载过重之后，整体服务性能处理就会变慢，结果就是导致客户端在向服务端发起请求、链接就会超时，由于客户端一般都会有一种超时机制，反复地向服务端再次发起请求，此时就相当于陷入了死循环，更加加重了服务器负载。

（2）多线程模型：由一组NIO线程处理IO操作

（3）主从线程模型：一组线程池接受请求，一组线程池处理IO



独立处理所有io的操作 完成所有的操作

适合场景：高并发、海量数据复杂的场景

多线程：一组nio线程处理io操作



重点概念：线程池 等待队列

主从线程模型：一组线程池接受请求，一组线程池，处理io



<https://blog.csdn.net/quxing10086/article/details/80296245>

# java仿微信全栈开发高性能后台+移动客户端

## hello netty步骤：

这些东西多看官方文档

一个最简单的Netty服务端包含了五个步骤：

（1）构建一对主从线程组  
（2）定义服务器启动类  
（3）为服务器设置Channel  
（4）设置处理从线程池的助手类初始化器  
（5）监听启动和关闭的服务器





### 先写helloserver



### 编写初始化器



### 子处理器





## 聊天小练习

### 实时通信

实时通信的分类：

（1）Ajax轮训

通过JS以Ajax异步地让浏览器每隔一段时间（10S）发送请求到后端，去询问服务端是否有新消息、新状态等，如果有则取出并通过前端再渲染。但这很容易造成无限循环，也就是前端Ajax会不停地循环后端的数据 （使用场景：浏览器不需要一直刷新，简单的后台管理系统中的数据更新等）

2）Long Pull

与Ajax轮训类似，也是使用异步请求，只不过它的轮训方式不太友好，阻塞式轮训：当客户端发起请求之后，服务端如果未响应，则Long Pull就不会有响应，直到服务端返回response。过程中不停地建立Http请求，等待服务器端进行处理，被动响应，缺点也是非常明显，也很耗费资源，性能低。

3）webSokect - 推荐

Http本身就不支持长连接，Http1.1支持长连接，WebSokect就是使用了Http1.1协议来完成一小部分的握手，简单来讲就是，客户端发起请求到服务端，服务端会去找一个副助理，找到之后服务器端会和客户端一直保持连接，为客户端进行服务，并且可以主动推送一些消息给客户端。

### Websocket

WebSokect有哪些协议，又有什么优点？

1）首先WebSokect相对于Http这种非持久化来讲，是一种持久化的协议，Http的生命周期可以说是通过一个request来进行判定，有一个request请求到后端，后端也会相应的返回一个response给客户端，或者有多个request对应到多个response，两者之间都是一一对应的，有多少个request请求就会有多少个response相应，不会有偏差。此时response其实也是被动的，它不能由服务器端主动发起相应，必须先有request请求。

2）WebSokect由此诞生，它使得资源不会像以前一样浪费，并且它也是非常的主动，只要链接一旦被建立完毕之后，那么服务端就可以不停的主动推送消息给客户端，客户端不需要主动请求服务端也可以达到一样的效果。 也就是说，只要建立一次Http请求就能达到信息的源源不断的传输。类似于在线Online小游戏，一开始建立连接，就可以一直保持在线了。



前端采用hbuilder来开发

### 主程序



### 初始化器



### 子处理器





### 前端

采用hbuilder编写 感觉这个ide不错



补充一个聊天室应用

<https://blog.csdn.net/xiaoping0915/article/details/81202851>

# 使用mui与h5+构建移动端

### 工具

Mui:

<https://www.dcloud.io/mui.html>

h5+

<http://www.html5plus.org/doc/>

hbuilder开发工具

阿里的图标库：

<https://www.iconfont.cn/>

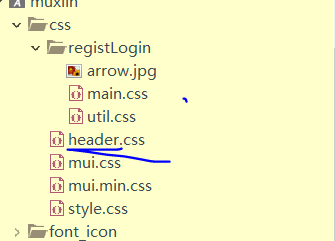
加入购物车，然后添加到项目， 下载到本地 复制文件夹到项目里。然后打开index.html照着做

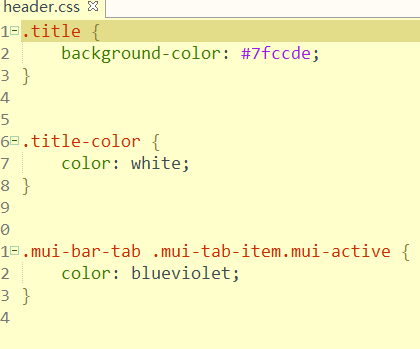
### 步骤

调试的时候需要打开手机的usb调试模式

新建工程的时候用mui项目 不要用默认的。

Js代码块一般放在body底部





主页面 index.html





顶部显示飞鸽传书，标题栏、状态栏颜色一样。

接下去用mt底部选项卡 做类似微信底下那一栏



开发过程要用mui代码块 很方便



这东西在浏览器上看不见效果。用到plus必须要用真机调试

页面被遮挡了 用一个浏览器调试器里的小箭头看高度就可以了

页面切换 事件管理 绑定事件

Tap手机触摸事件



注意tabindex 用来切换视图



# 使用springboot整合netty构建后台

### 数据库设计

数据库：可以下载phpstudy来替代mysql 也可以直接安装mysql

数据库密码：123456

新建数据库：feige-dev



User: 不采用自增长 分库分表



头像有大小之分 qrcode用来扫码 扫码用来添加好友 cid用于消息的推送

用户请求



朋友



聊天记录



这里时间根据当前时间戳来设置

### idea下springBoot+mybatis的逆向工程



MAVEN工程

Java <https://www.cnblogs.com/shirley-0021/p/8510051.html>

Maven <https://blog.csdn.net/qq_37904780/article/details/81216179>

Idea版

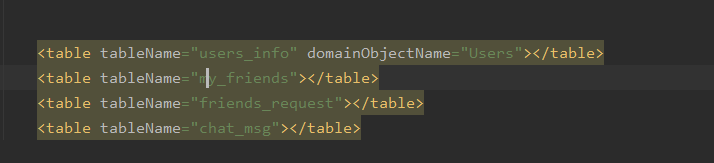
数据库生成实体类

工程目录



如果数据库发生了变化，那么需要把刚刚生成的文件都要删掉。因为逆向工程是追加的方式。

配置文件需要这样改



这样生成出来的没有错。

### 创建maven工程、添加依赖

注意检查 idea的配置 要用阿里的镜像 不然特别慢

先把工具类依赖全部注释掉。不然会报错



发现一个问题就是 之前逆向工程的时候就users生成有问题 但是改成users\_copy就好了

### Netty

这部分不知道对不对 后面需要修改

先复制之前的netty的wssserver 进行一定的修改

作为一个工具类，在项目加载之后再去启动 使用额外的线程去启动

应该是一个单例



新建nettybooter 调用上面这个单例



将以前另外两个文件复制过来



# 用户注册登陆个人信息

## 前端登陆页面



先将这些页面实现一遍 main.css util.css要看看学一下怎么用别人的设计

这里推荐一个css学习网站：<https://www.w3school.com.cn/cssref/pr_padding-top.asp>



这一部分其实不需要增加也可以。就是效果有点丑，他是重写了



## 后端准备工具类





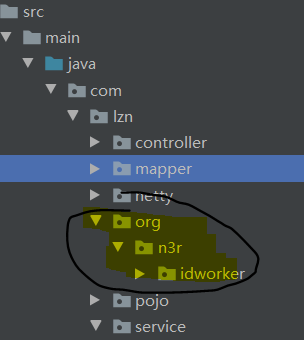
*/\*\*  
 \** ***@Description:*** *自定义响应数据结构  
 \* 这个类是提供给门户，ios，安卓，微信商城用的  
 \* 门户接受此类数据后需要使用本类的方法转换成对于的数据类型格式（类，或者list）  
 \* 其他自行处理  
 \* 200：表示成功  
 \* 500：表示错误，错误信息在msg字段中  
 \* 501：bean验证错误，不管多少个错误都以map形式返回  
 \* 502：拦截器拦截到用户token出错  
 \* 555：异常抛出信息  
 \*/*



*/\*\*  
 \* @Description: 自定义响应结构, 转换类  
 \*/*

MD5Utils

对字符串进行md5加密 密码加密 我们登陆的时候要对密码进行加密



Idworker是唯一id生成器

在sid文件上面加上@Component

Sid类里面有个nextshort我们需要用

修改启动类中的包扫描



## 数据库事物

Example的用法

<https://blog.csdn.net/qq_38719039/article/details/80488515>

查询需要事物注解    @Transactional(propagation = Propagation.SUPPORTS)

事物的资料

<https://blog.csdn.net/shi750989074/article/details/78855770>



## BO和VO的概念

Bo 从前端传到controller

vo 从controller返回前端

很多东西不能直接返回前端



## 后端登录注册控制器



## 后端服务接口



## 后端服务





## 修改前端入口页面



修改页面入口 改成login.html 原来是index.html

## 前端全局工具类

在js文件夹下新建app.js作为全局的工具类

判断空、谈出信息框



## 前端登录

Login.html



userform绑定一个监听事件 监听有submit事件发生

登录的大致流程是：

获取表单 获取用户名 获取密码

如果用户名没填 就获得焦点

如果密码没填 就获得焦点

都填了 就开始判断长度 (如果是注册的话其实应该加一个验证 验证可以在前端增加也可也在后端增加) 这里调用了弹出信息框的函数

## 登录注册页面前后端联调

修改 mui ajax 那个实例代码 进行修改

mui.ajax: <https://dev.dcloud.net.cn/mui/ajax/>



遇到问题的时候要用postman去测试 看看是不是后端的问题

cid是手机的唯一id 用于手机接受消息用的

var cid = plus.push.getClientInfo().clientid;

在app.js中定义工具类函数 保存、设置全局用户信息



get方法返回改成 return JSON.parse(userInfoStr);   
ajax请求这里请求的对象是User 不是UserBO

User包含所有的属性

ajax请求成功之后 需要取消用户名和密码input的焦点，然后调用openwindow来打开index.html index.html默认在第一页



但是在安卓设备页面跳转之后 滑动界面会回到登陆界面 这是一个bug



自动登陆

修改一下 get userInfo返回类型是json对象



增加几行代码 判断本地缓存有没有user 有的话直接跳转页面 免登录



用户名和密码都是1 即使数据库清除了。本地缓存依然在，所以这个后期也需要调整。

## 个人信息页面的开发

Mbody body主体

Mlist 图片再右 生成列表

<https://blog.csdn.net/a2013126370/article/details/82786681> line-hight+font-size

采用谷歌浏览器小箭头查看样式 调整css

这里总结一下调整左右两部分可以

有头像、昵称、账号、二维码、退出登录这几部分







## 个人信息页面js开发

监听 视图的show事件

图片服务器地址。。。后面需要完善



feige-me.html该页面要定义监听事件，一旦有show事件发生。就刷新用户信息

从本地缓存中读取user的faceimage 图片路径 nickname username 账号

其中账号是唯一的不能改

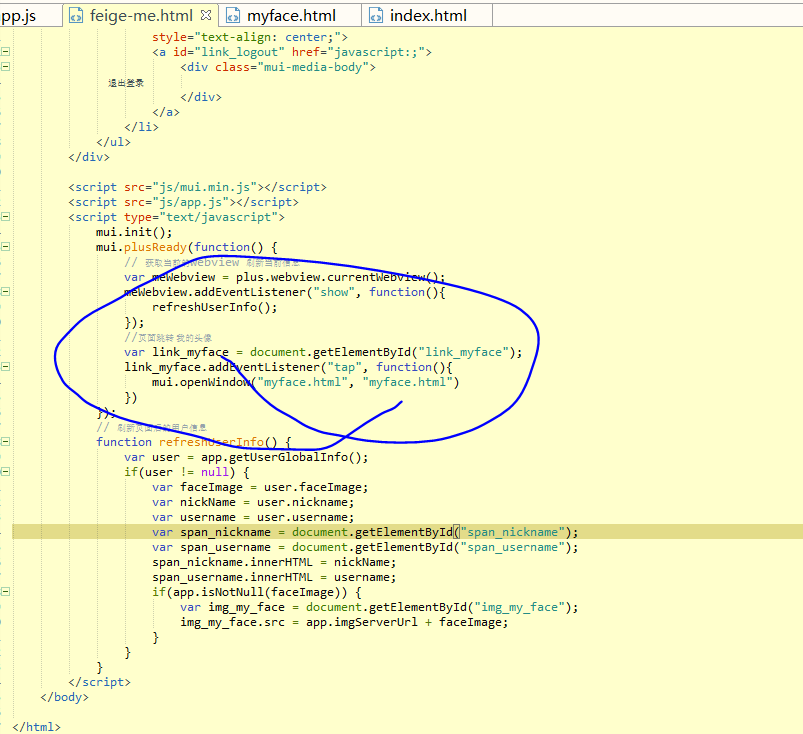
但需要做一个判断 有没有图片



img\_my\_face.src = app.imgServerUrl + faceImage;

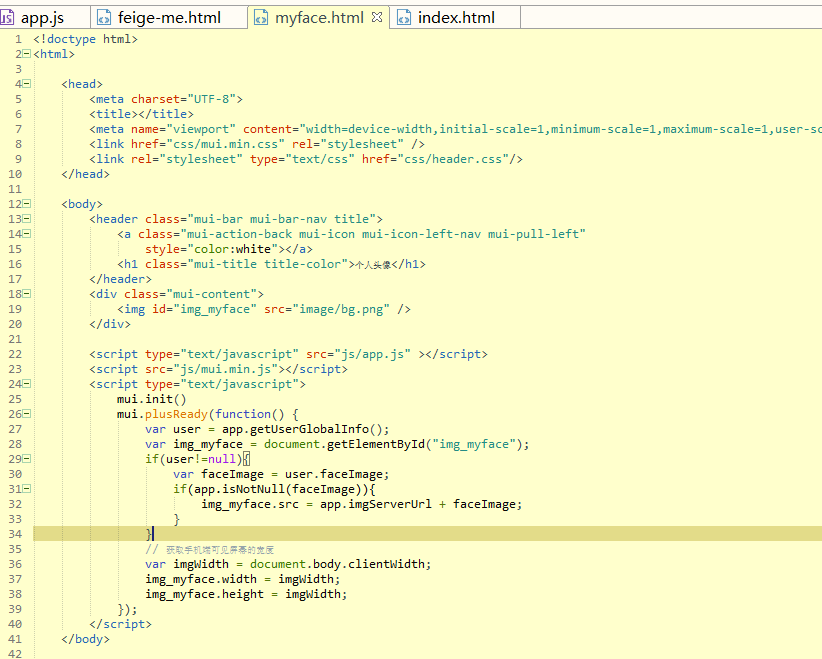
跳转到我的头像

给图片id 绑定事件 有tap触摸事件的时候 发生跳转 跳转到myface.html

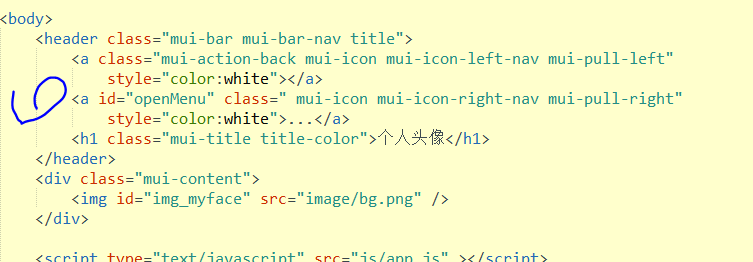


## 个人头像开发

读取本地用户缓存信息 读取用户的信息，如果用户图像信息不为空的话就加上app.js中图片服务器的地址 读取服务器里存储的图像，并且显示 这里还读取了手机屏幕器可见品目的宽度 并且把相关的数值赋值给图片。这个方式主要用来在不同的手机上显示图片的格式是相同的。尺寸是相对与不同手机的屏幕。



增加可选菜单 并且增加id 用id绑定事件

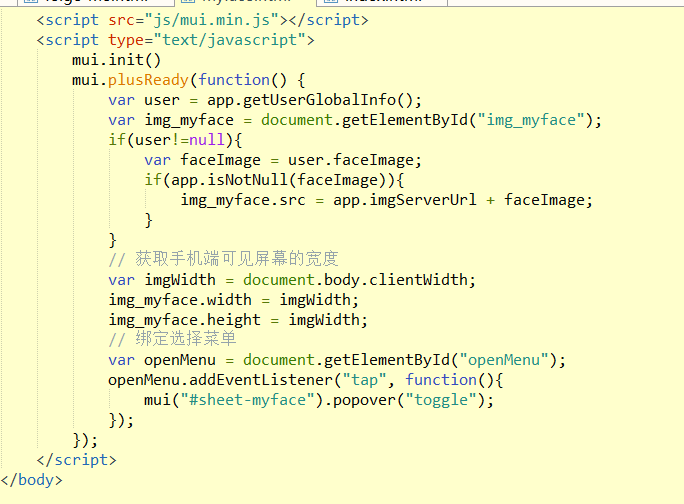


Mac 代码块h5 弹出式照片 显示在屏幕底部的弹出式菜单 选择照片 下载照片

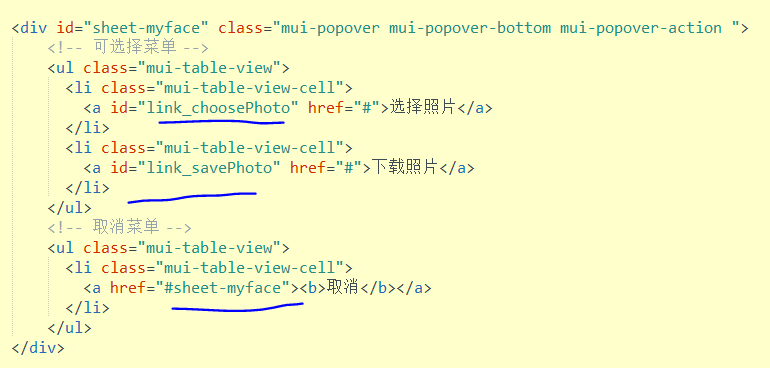


上面给三个小点定义了id 给openwindow增加一个监听事件，有触摸事件时

谈出选择菜单



给三个菜单选项增加id

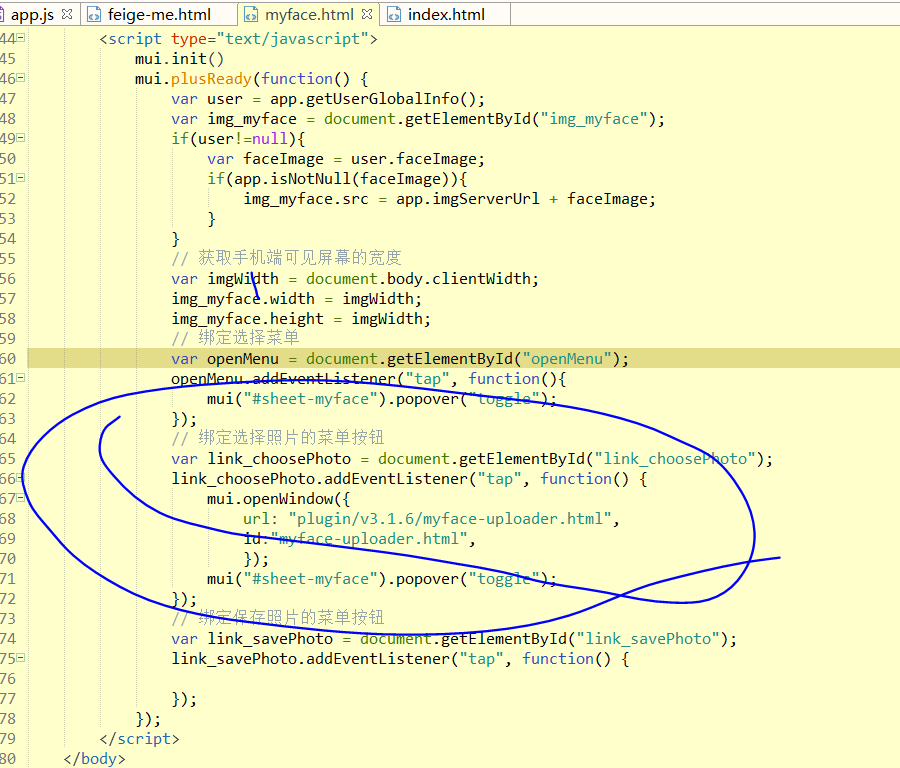


给弹出的菜单选项添加监听事件 触摸触发函数



选择照片

添加一个插件 plugin 这个插件后期还需要修改



## 文件上传



跨系统查看，是访问不到。文件太乱了。没有统一规划

推荐用下面这种方式来上传文件 文件服务器可以是单机也可以是集群



文件服务器 1 第三方 云 只要实现一些接口就好

2 让企业自己搭建服务器

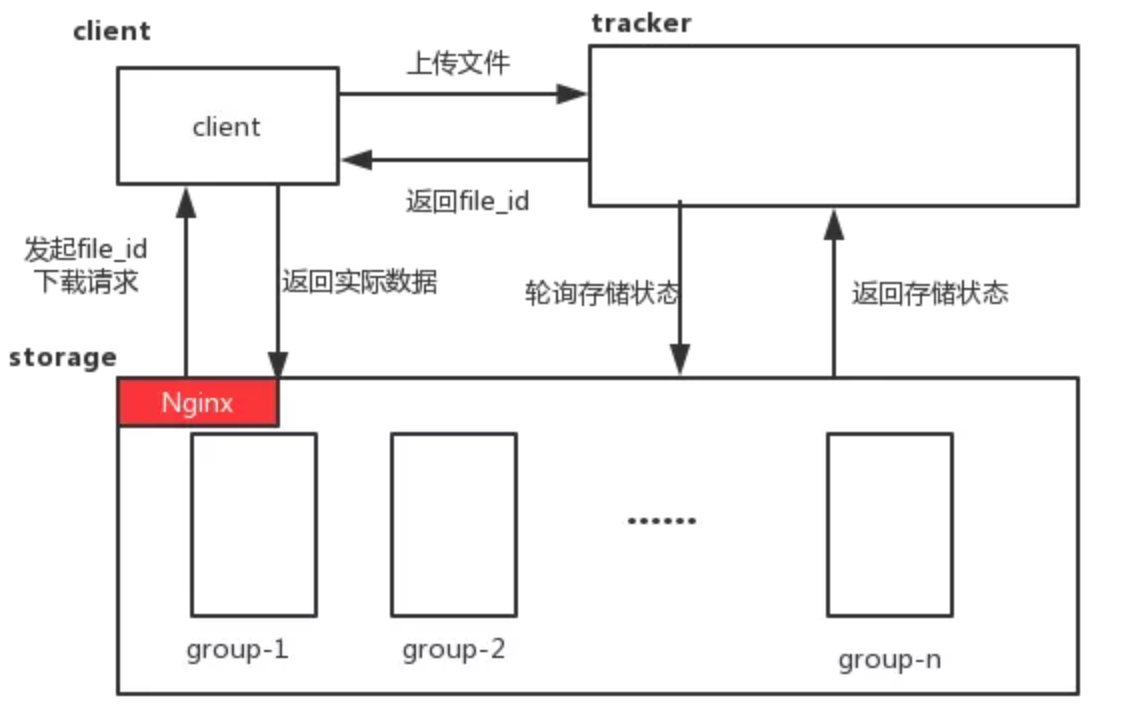
## FastDFS



## 文件上传的流程

Ts相当于一个项目经理 ss相当于程序员要去汇报 定时汇报状态





这一部分他是在linux系统下配置的 我们可以在docker里面配置 就没事了

## Docker配置无敌

Docker配置

四. 在脚本的registry-mirrors中设置为国内的镜像地址,我这里使用的是163的镜像地址,点击Apply等待docker重新启动

推荐几个常用的镜像地址.

杀死所有正在运行的容器

docker kill $(docker ps -a -q)

删除所有已经停止的容器

docker rm $(docker ps -a -q)

删除所有未打 dangling 标签的镜

docker rmi $(docker images -q -f dangling=true)

删除所有镜像

docker rmi $(docker images -q)

docker kill $(docker ps -a -q)

docker rm $(docker ps -a -q)

docker rmi $(docker images -q)

docker pull delron/fastdfs

docker run -d --network=host --name tracker -v /root:/var/root delron/fastdfs tracker

docker run -d --network=host --name storage -e TRACKER\_SERVER=47.98.209.48:22122 -v /root:/var/root -e GROUP\_NAME=group1 delron/fastdfs storage

阿里云要开放端口

docker run -d --network=host --name tracker -v D:\var\fastdfs:/var/root delron/fastdfs tracker

docker run -d --network=host --name storage -e TRACKER\_SERVER=192.168.1.102:22122 -v D:\var\fastdfs:/var/root -e GROUP\_NAME=group1 delron/fastdfs storage

wget https://v3u.cn/v3u/Public/images/logo.png

/usr/bin/fdfs\_upload\_file /etc/fdfs/client.conf logo.png

Win10一堆问题

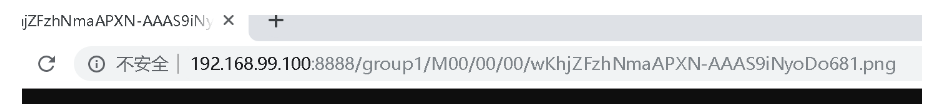
https://registry.docker-cn.com

http://hub-mirror.c.163.com

https://3laho3y3.mirror.aliyuncs.com

http://f1361db2.m.daocloud.io

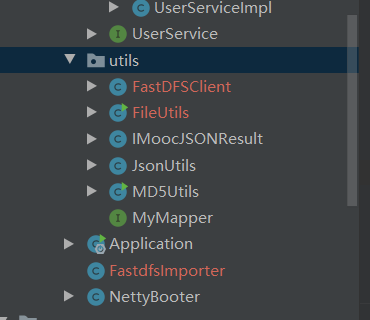
https://mirror.ccs.tencentyun.com



## 文件上传实现

<https://github.com/tobato/FastDFS_Client>

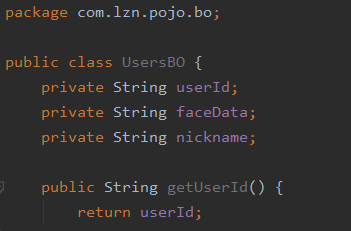
## 后端文件传输工具类



这几类要复制 下面的那个红的是官方的代码复制改一下名字

## UserBO

新建bo



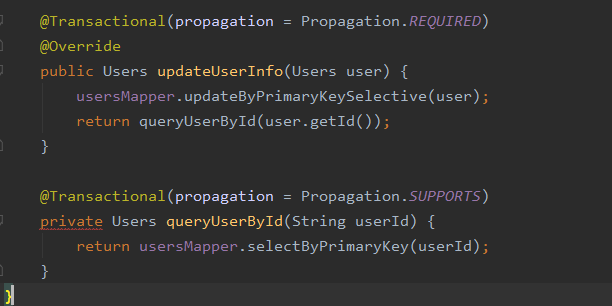
Bo是前端传过来的数据

## 更新用户信息服务接口和服务

updateByPrimaryKeySelective（user）

user必须包含User的主键

先实现服务



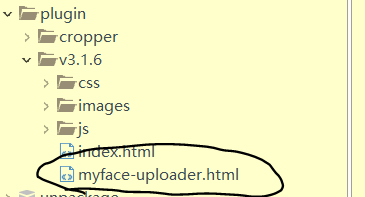
部分更新 传了什么值就更新哪些值

## 后端图片上传控制器



## 前端图片裁剪插件

前端直接复制plugin那个包就可以了



这里要注意：

后端的图片上传url 传输的data是UserBO 没有定义的比如nickname就是null

注意这里要有显示上传等待 和 关闭的函数

上传成功之后，页面跳转到我的个人信息页面



增加图片服务器地址



## 解决一些图像上传的bug

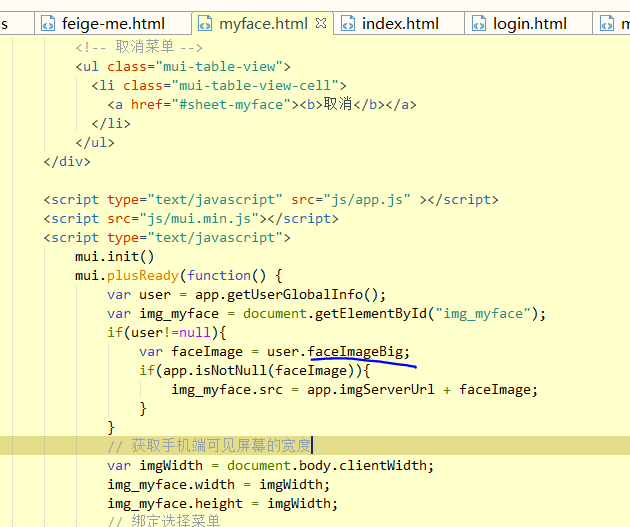
现在的重新点一下我 才发现自己的头像变了 这是一个bug 可以

feige-me.html 监听一个刷新事件 一旦当前页面被重新进入。就刷新用户信息

测试一下只要进入这个界面就会触发refresh事件



myface.html修改 我的头型显示的是大图像



myface..html

头像上传好之后 大头像不会马上就变

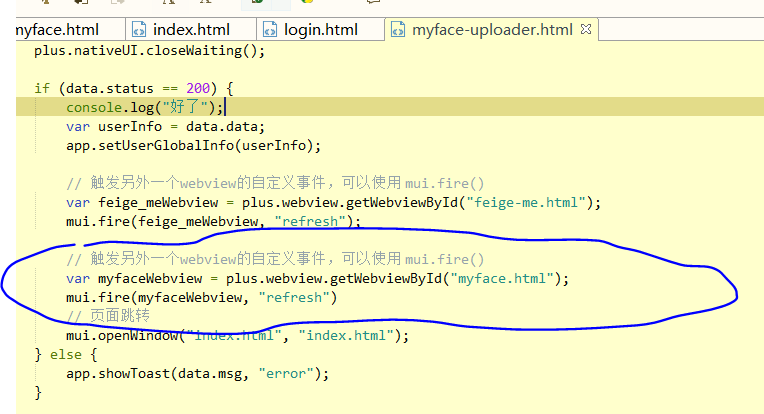
还有个进程的问题 多次上传之后 必须要结束进程

解决进程问题

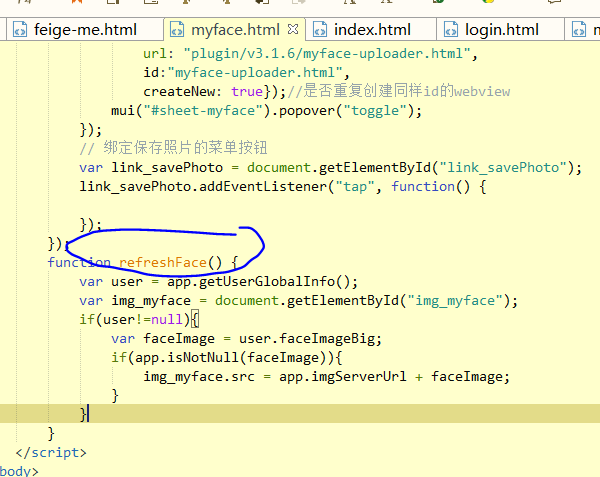
然后解决大图没法及时更新的问题

然后修改裁剪图片插件裁剪图片 之后自定义事件 触发自定义事件

feige-me和myface 这样的话feige-me myface就有refrash事件 触发刷新用户信息



myface.html 刷新用户信息function() 点进来查看我的头像的时候 会刷新用户信息 显示最新的头像



## 前端下载照片

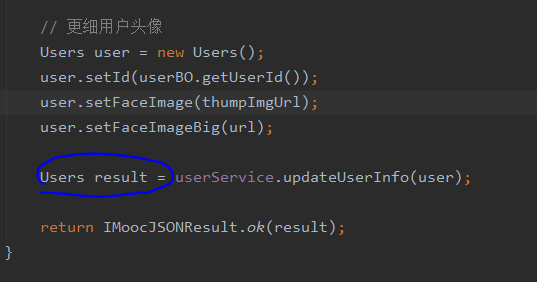
myface.html

学习h5+downloader的知识

完善下载照片



下载照片之后。返回有可能出现一些信息不全的bug 需要增加下面



## 前端修改昵称

复制myface.html 重命名mynickname.html

mynickname.html

清空body script



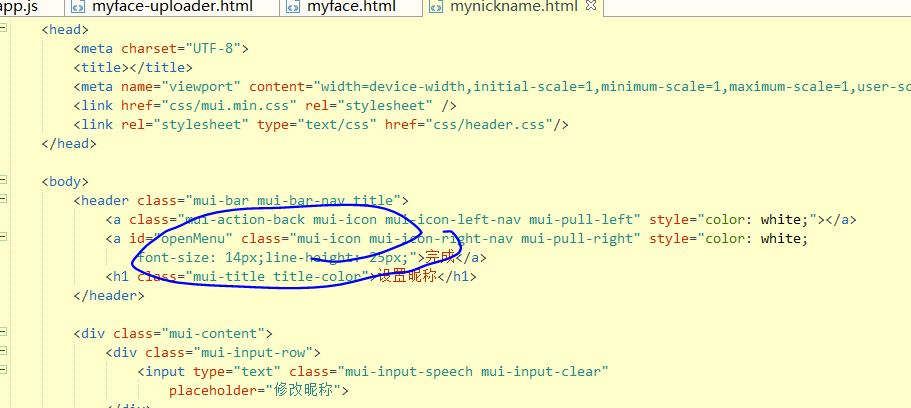
中间那个文本框输入快捷键Mni



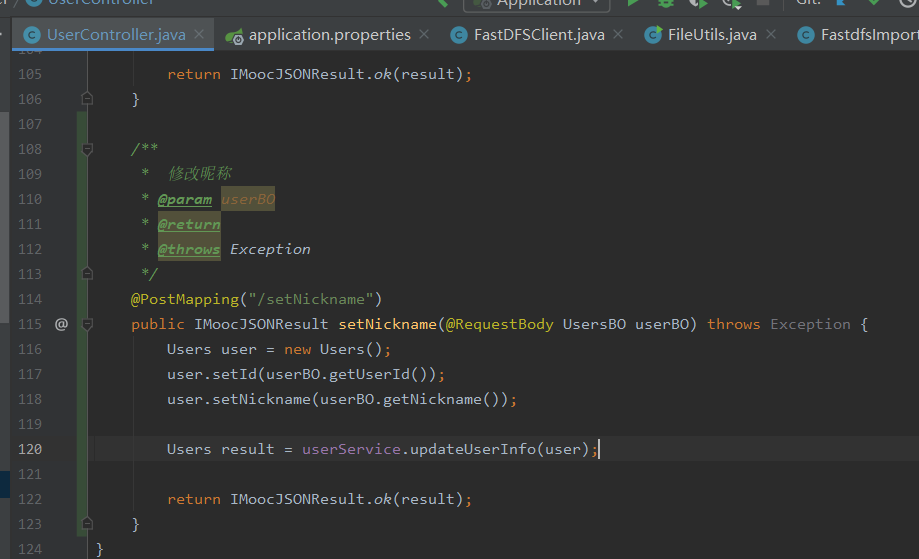
在feife-me.html中 增加跳转



修改样式 这个是为了让返回箭头和完成按钮在同一行里 大小也很舒服



## 后端修改昵称控制器

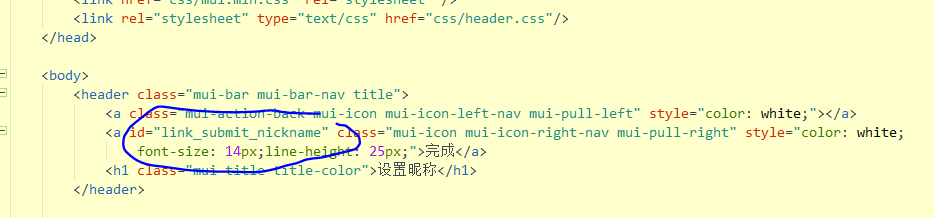


## 后端昵称界面显示昵称



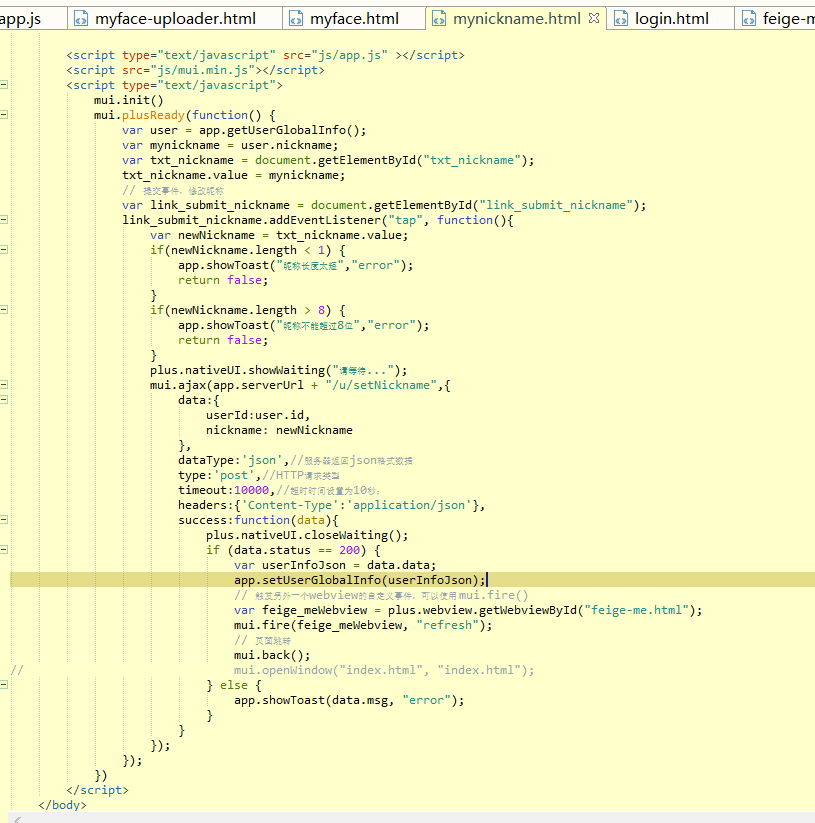
可以显示昵称

昵称修改的限制条件后面可以加上



修改id

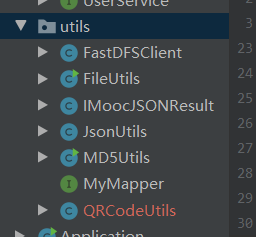
修改昵称 并且返回显示



## 后端二维码

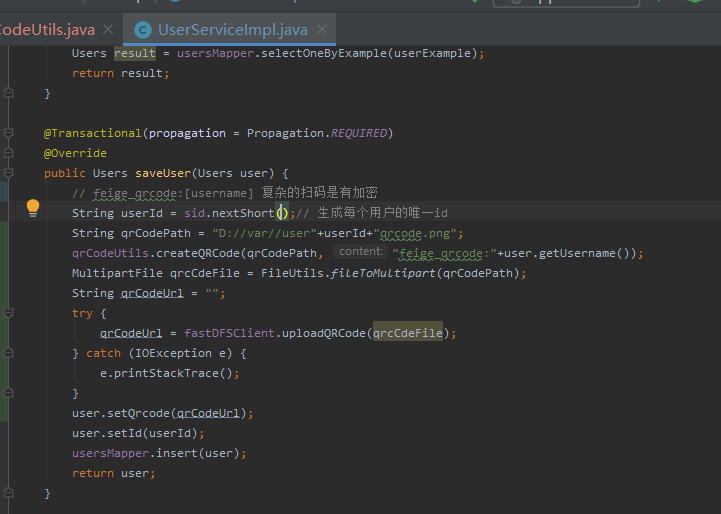
二维码用于扫描去添加好友

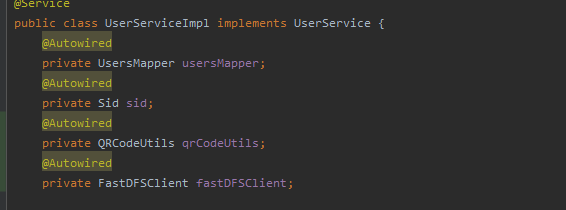
先增加工具类



## 修改后端保存用户服务 添加二维码

修改保存用户的方法 目的是注册用户的时候 保存二维码





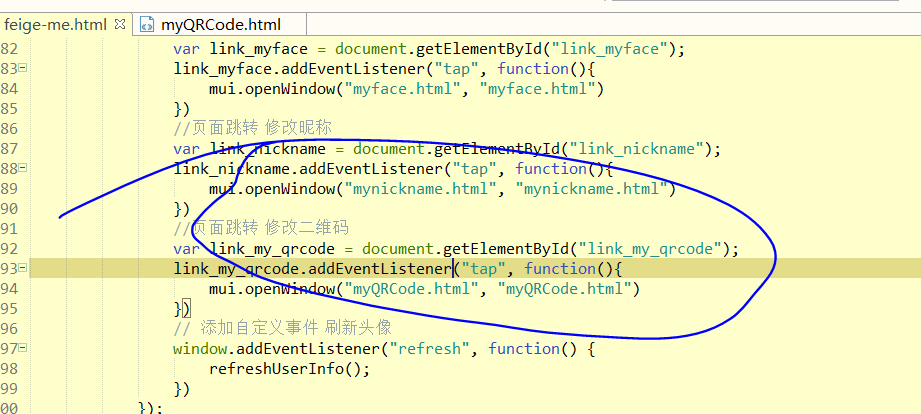
要想删掉缓存可以直接删掉应用 删除应用直接删除缓存

然后就可以看到电脑里有二维码的图片了。说明服务器也有二维码的图片

用微信扫一扫去扫这个二维码就是我们设计的字符串

完成二维码页面查看 前端设计

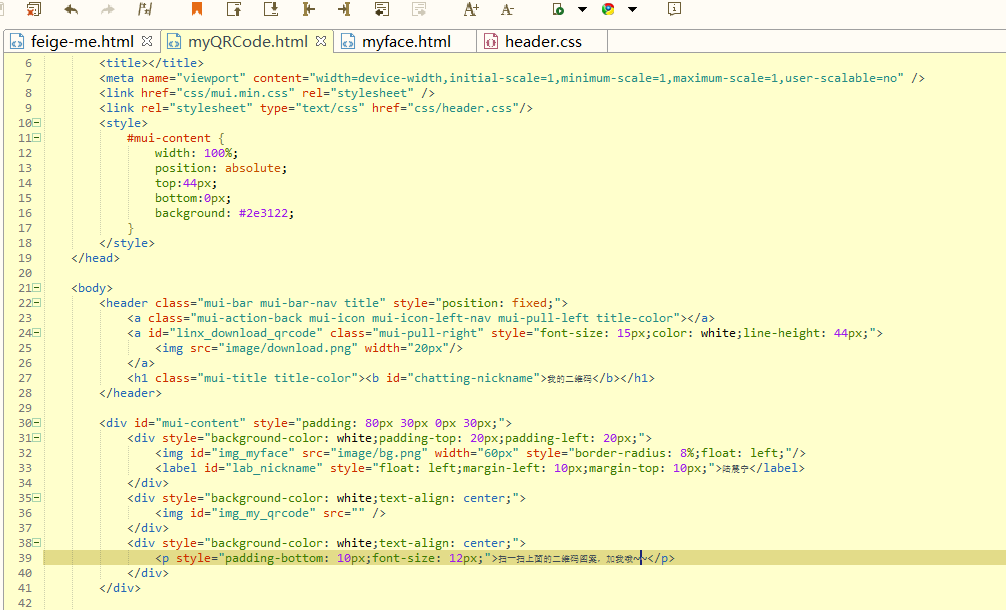
feige.html添加跳转到mynickname.html 页面 所以给响应的id增加监听事件



## 前端二维码显示

显示二维码 纯css

下载 下载和照片的保存是一样的





# 发现页面

朋友圈 添加好友 扫一扫

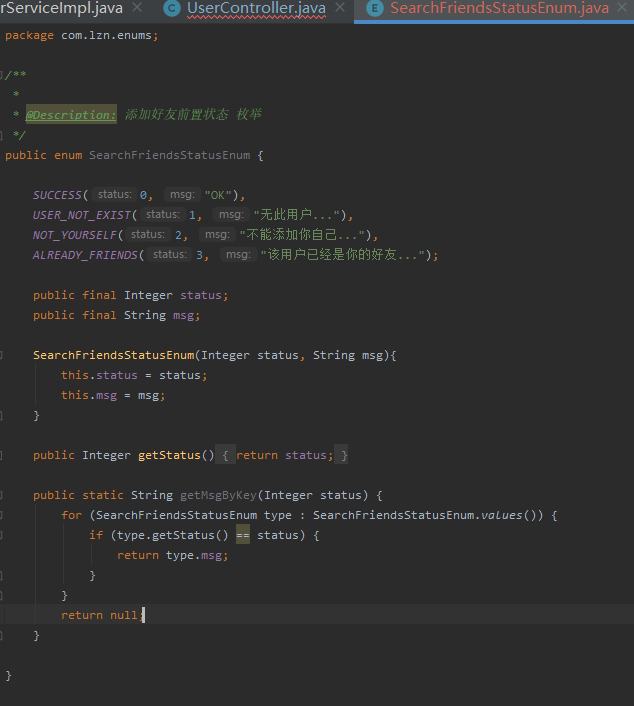
## 前端发现样式

mb 主题 ml列表

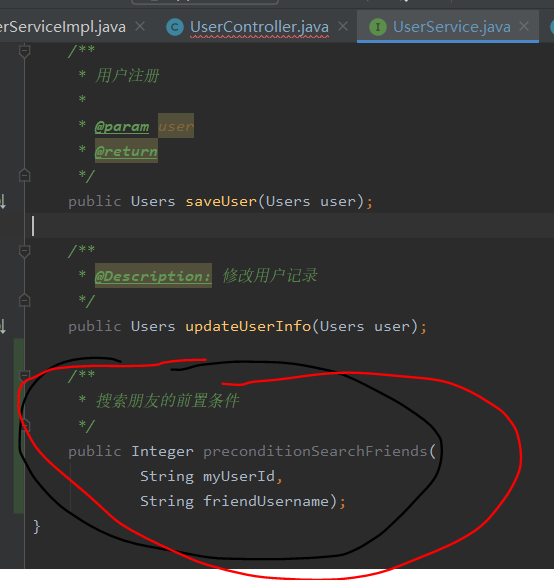


## 查找用户后台接口

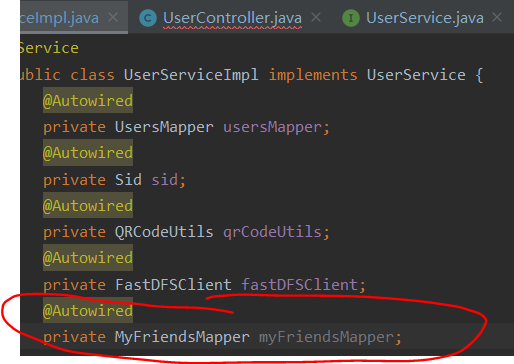
先准备一个枚举文件 表示添加好友的前置状态



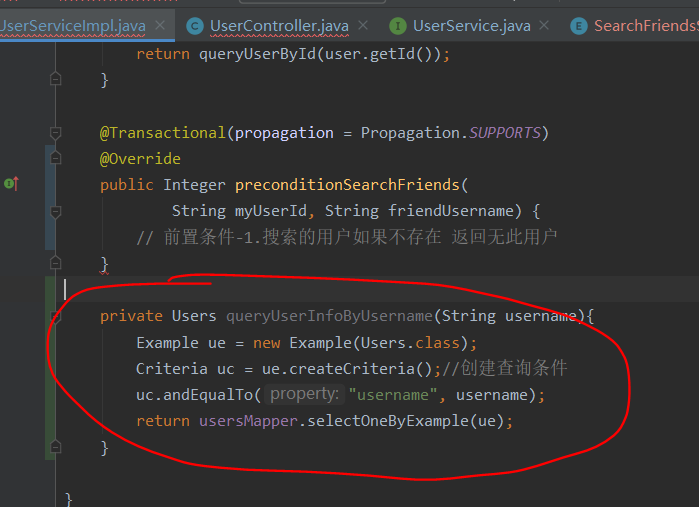
服务接口

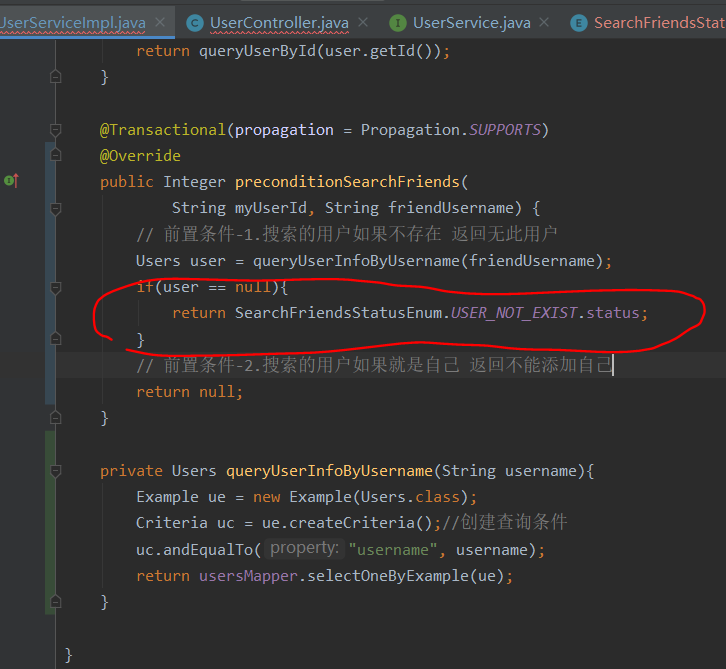


实现服务



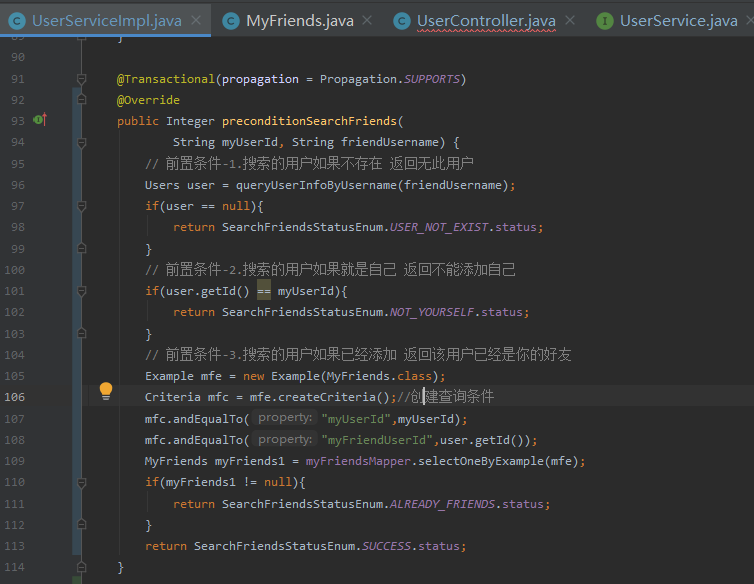
查询事物一般是support



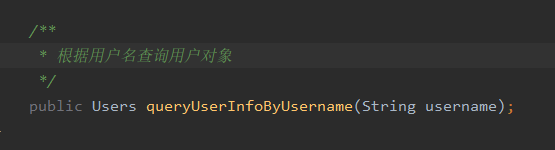




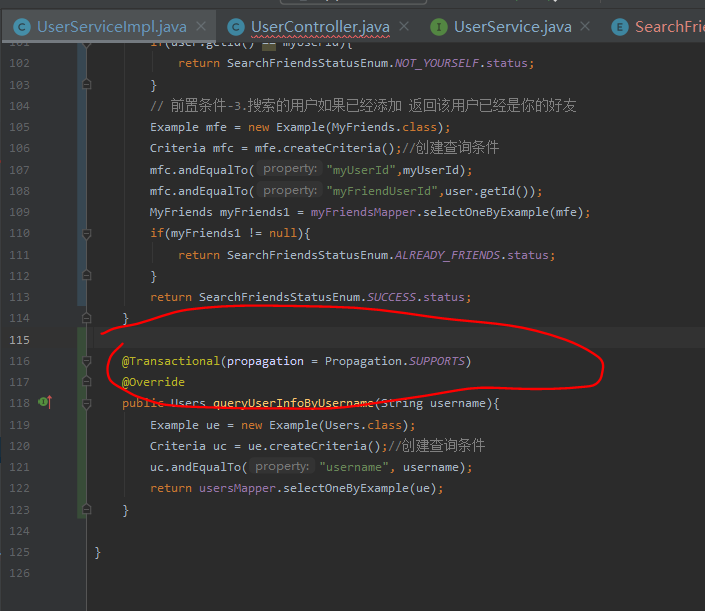
服务 完成前置条件

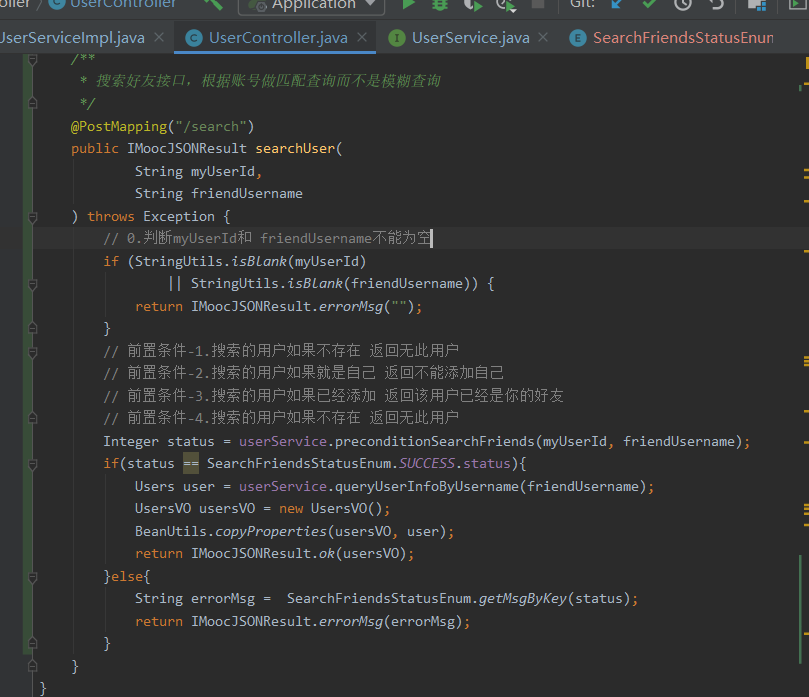


在服务接口中增加



private变成public 并且加上注释

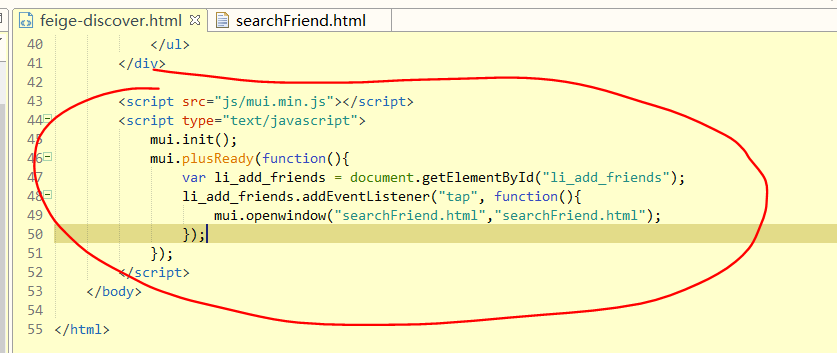




## 搜索好友前台



写js



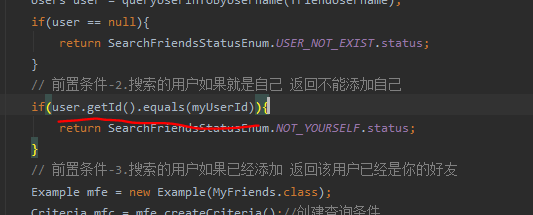
openWindow

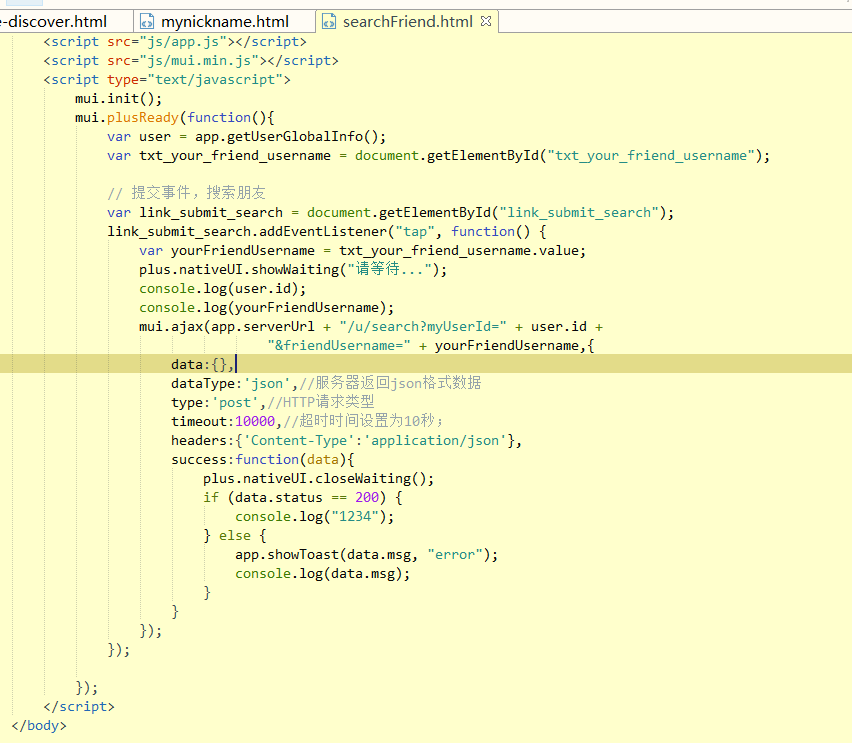
复制mynickname.html 重命名searchFriend.html



写js

修复一个bug 不然不对





## 搜索结果页面

# 通讯录页面

# 聊天页面开发

# 课程总结