

## 前后端的完善

后续完善：

1、用户前端的注册用户页，要与管理前端的添加用户页面相区别；

- 管理前端添加用户，要选择普通用户 or 管理员；
- 用户前端只能注册普通用户；
- 普通用户的账户，不能登录管理前端；

- |   |   |
|---|---|
| 1 | 具体要求：   |
| 2 | 修改后端，当普通用户尝试登录管理前端时，不能登录，并在回传消息msg:invalid account. |

3、对后端java增加全局异常类处理代码，并且将信息log日志输出；

mapper->service->controller，如果mapper层出现异常，没有对应处理的话，异常会一层的上传，最终异常会传给用户。参考如下视频：

[Day12-16. 异常处理哔哩哔哩bilibili](#)

全局异常处理器：

创建exception目录，并创建GlobalExceptionHandler类：

```
1  @Slf4j
2  @RestControllerAdvice
3  public class GlobalExceptionHandler {
4
5      @ExceptionHandler(Exception.class)
6      public Result exceptionhandle(Exception ex){
7          //ex.printStackTrace();
8          log.error("发生异常:",ex);
9          return Result.error("操作失败，请联系管理员！");
10     }
11 }
```

2、当前的java后端直接对接来自前端的api访问，不利于后续配置https和负载均衡；建议采用Nginx作为java后端的反向代理；

如果Nginx作为java后端的反向代理，那么java后端可以不用进行跨域设置，即删除config包下的类：

```
1  @Configuration
2  public class CorsConfig implements WebMvcConfigurer {
3      @Override
4      public void addCorsMappings(CorsRegistry registry) {
5          registry.addMapping("/**")
6              .allowedOrigins("http://localhost:8081")
7              .allowedMethods("GET", "POST", "OPTIONS")
8              .allowedHeaders("*")
9              .allowCredentials(true);
10     }
11 }
```

原来前后端的访问路径:

```
1 | 前端(8081)->java后端(8080)
```

添加Nginx作为反向代理后的访问路径:

```
1 | 前端(8081)->Nginx(80)->java后端(8080)
```

Nginx对应的配置文件:

```
1 | server {
2 |     listen 80;
3 |     server_name localhost;
4 |
5 |     location / {
6 |         proxy_pass http://localhost:8080/;
7 |
8 |         add_header 'Access-Control-Allow-Origin' $http_origin;
9 |         add_header 'Access-Control-Allow-Credentials' 'true';
10 |        add_header Access-Control-Allow-Methods 'GET, POST, OPTIONS';
11 |        add_header Access-Control-Allow-Headers '*';
12 |        if ($request_method = 'OPTIONS') {
13 |            add_header 'Access-Control-Allow-Credentials' 'true';
14 |            add_header 'Access-Control-Allow-Origin' $http_origin;
15 |            add_header 'Access-Control-Allow-Methods' 'GET, POST,
16 |            OPTIONS';
17 |            add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-
18 |            Requested-With,If-Modified-Since,Cache-Control,Content-Type,Range';
19 |            add_header 'Access-Control-Max-Age' 1728000;
20 |            add_header 'Content-Type' 'text/plain; charset=utf-8';
21 |            add_header 'Content-Length' 0;
22 |            return 204;
23 |        }
24 |    }
25 | }
```

步骤1:

修改Nginx中的配置文件, 添加反向代理和跨域设置, 并且重启Nginx;

步骤2:

删除config类, 重新打包jar包, 通过共享文件夹, 传入到ubuntu;

并且启动新版本的没有设置跨域访问的jar包;

步骤3:

修改用户端前端的请求地址:

```
1 | http://localhost:8080/login->http://localhost/login
2 | http://localhost:8080/getQuestion->http://localhost/getQuestion
3 | http://localhost:8080/register->http://localhost/register
```

问题：

- 1 login,register这两个接口可以正常的访问，
- 2 但是getQuestion接口不能正常访问。

解决：

### 1. quiz.html 的请求触发了 OPTIONS 预检请求

- quiz.html 中的 getQuestion 请求可能携带了额外的请求头（如 Authorization、自定义头等），或者使用了 Content-Type: application/json。
- 这类请求会被浏览器识别为“非简单请求”，触发 OPTIONS 预检。

### 2. OPTIONS 响应头 Access-Control-Allow-Headers 不完整

- 您的配置中，OPTIONS 响应的 Access-Control-Allow-Headers 固定返回一组特定头：  
nginx

```
1 'DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Range'
```

- 如果 getQuestion 请求携带了超出此列表的请求头（如 Authorization），浏览器会因未在 Access-Control-Allow-Headers 中找到对应值而报错。

### 3. login.html/register.html 为何正常？

- 这些页面的请求未触发 OPTIONS 预检，因为它们是“简单请求”：
  - 使用 GET/POST 方法。
  - 仅携带简单头（如 Content-Type: application/x-www-form-urlencoded）。
- 这类请求跳过预检，直接受非 OPTIONS 请求的 CORS 配置控制（Access-Control-Allow-Headers: '\*' 生效）。

主要的修改：

```
1 add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-  
with,If-Modified-Since,Cache-Control,Content-Type,Range';  
2 修改为动态：  
3 add_header 'Access-Control-Allow-Headers'  
$http_access_control_request_headers; # ☒ 动态获取请求头
```

参考内容：

之前我设置的，可以正常访问的配置：

## 6 跨域问题

前端的域名是user-front，而后端的域名是user-backend，两者不同，所以会导致跨域问题。

要解决跨域问题，有很多方式：

- 最简单的我们可以将后端的域名修改为用户-front和前端一样；
- 另外，还可以通过nginx网关支持来实现；

此处前端访问后端的request中的域名已经写死，那我们就采用网关支持的方式，

主要的操作就是在nginx配置文件中增加如下：

```
1  server {
2      listen 80;
3      server_name user-backend.66bond.com;
4
5      location /api/ {
6          proxy_pass http://127.0.0.1:8080/api/;
7
8          add_header 'Access-Control-Allow-Origin' $http_origin;
9          add_header 'Access-Control-Allow-Credentials' 'true';
10         add_header Access-Control-Allow-Methods 'GET, POST, OPTIONS';
11         add_header Access-Control-Allow-Headers '*';
12         if ($request_method = 'OPTIONS') {
13             add_header 'Access-Control-Allow-Credentials' 'true';
14             add_header 'Access-Control-Allow-Origin' $http_origin;
15             add_header 'Access-Control-Allow-Methods' 'GET, POST,
16 OPTIONS';
17             add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-
18 Requested-With,If-Modified-Since,Cache-Control,Content-Type,Range';
19             add_header 'Access-Control-Max-Age' 1728000;
20             add_header 'Content-Type' 'text/plain; charset=utf-8';
21             add_header 'Content-Length' 0;
22             return 204;
23         }
24     }
25 }
```

完成上述设置后，前端就可以顺利地调用后端的接口，解决了跨域问题。

至此位置，前端和后端的部署上线即宣告完成。