代码审计有很多方法，我比较喜欢先整体，后细节的审计方法。

整体，指整个代码的架构，把自己想象成开发，了解代码每个模块大概是干什么的，可以不用了解的太细，知道各个文件夹的大概是干啥的就行，建立全局观。

不同的系统，有时候用的设计模式不一样，这里建议多储备一点 java 的开发知识。

搞安全不要把自己局限化，审计多看点东西也没事的，按照基本逻辑来推

开发写代码-->安全人员挖洞

如果把开发那部分也吃透了，其实挖洞就是顺其自然的事情，因为开发的知识是基础，类似于 spring，springboot，ioc，aop 这些基础知识，其实都是需要掌握的。

单单局限于平面挖洞搞久了也没意思，学习一门语言，关键还是学习思想，学习模式，学习方法论，这样才能深入的去思考问题，才能看到别人看不到的东西。

细节，即面向漏洞来去看代码，然后通读对应代码的上下文，找出漏洞。

有了整体观的时候，各个细节才能对应上，才能联系起来。

一方面是因为代码会跳转，跳到了陌生的地方，大概率一脸懵逼，但如果先看了整体架构，就会好很多。

另一方面，漏洞有时候是需要组合利用的，挖到了一个，还需要和另一个串联起来才能 rce，因此这里也需要利用全局观来帮忙组合漏洞。

下面开始介绍泛微 ec9 这套东西的架构。

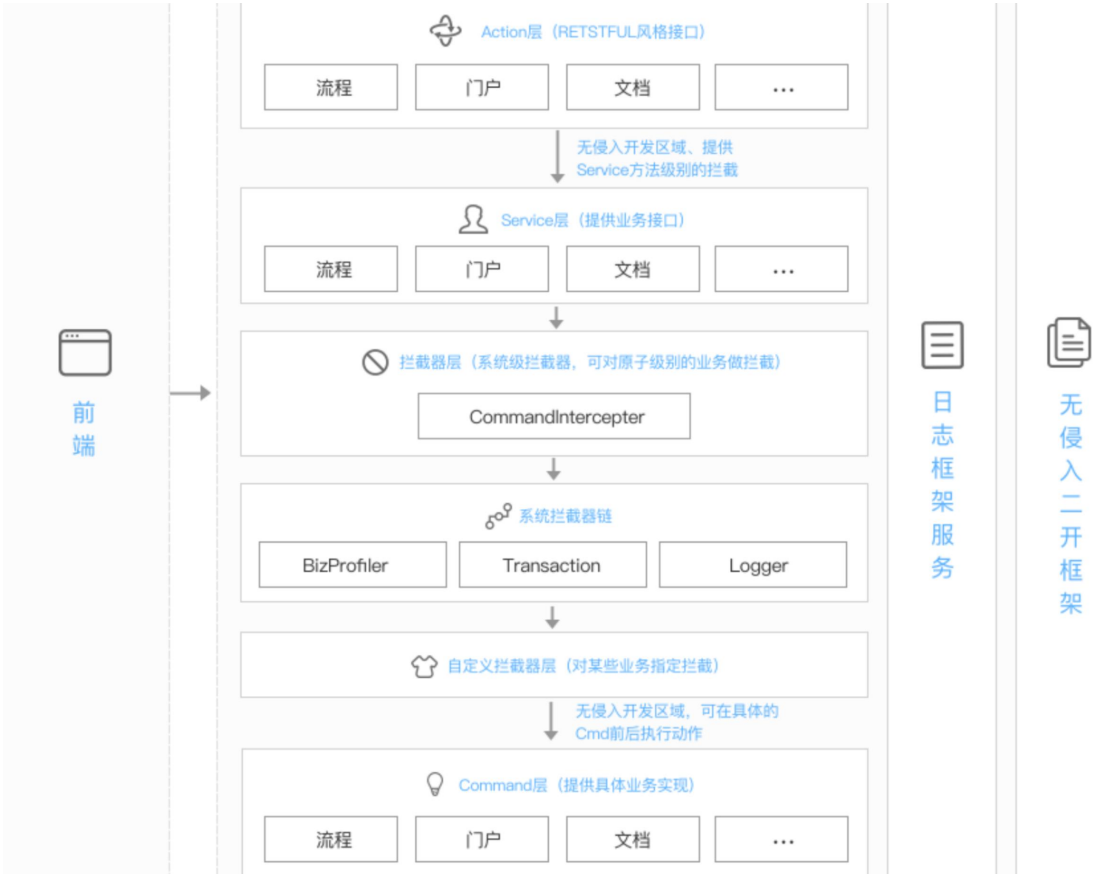
步骤一，先看官方文档：

//https://e-cloudstore.com/doc.html?appId=84e77d7890a14c439590b37707251859

这个好像没什么东西，我是想先看看代码架构，那就换一份文档

这一份文档中看到了 service 和 command 这两个关键字

再看看架构图



行，用的 ssh 框架，一套很早的框架

Java 框架进化是这样

ssh(structs+spring+hibernate)-->ssm(springmvc+spring+mybatis)-->springboot-->springcloud

框架的相关资料网上很多，可以自行百度

Weaver 用的这套东西，简单来说，就是以下几个模块

分别是

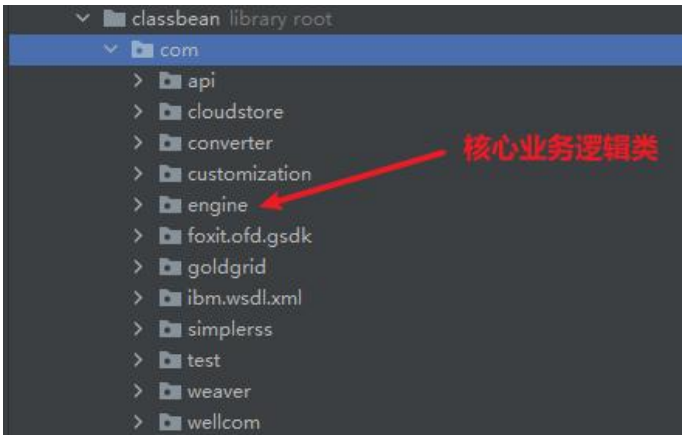
Action:对外的接口

Service：逻辑实现

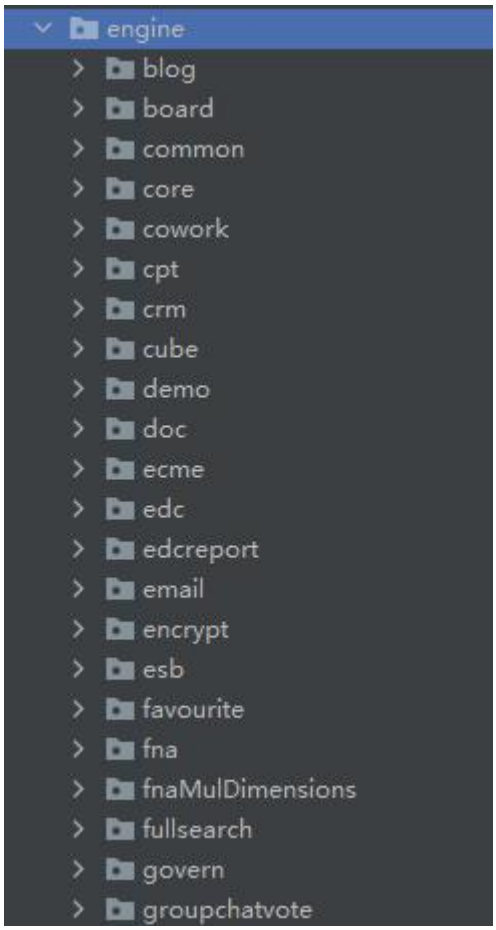
Command：功能实现

大体流程就是 action 接口被调用-->进入 service 逻辑-->service 调用 command-->return

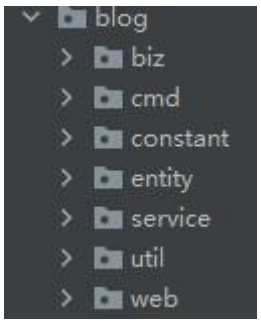
Eg：



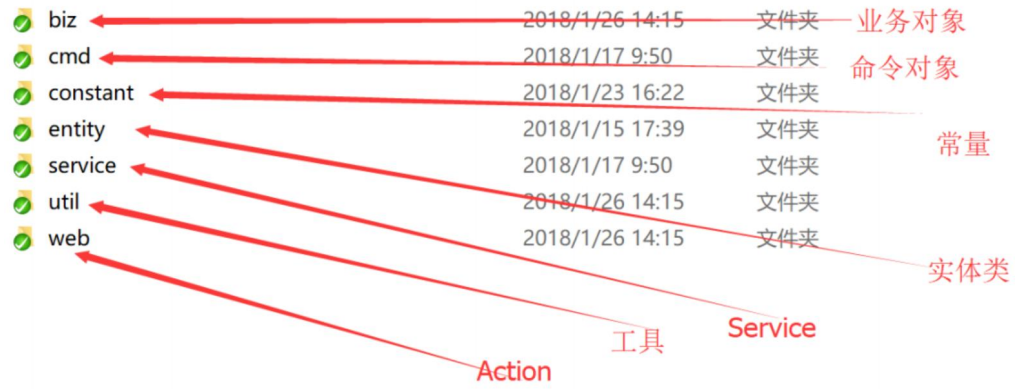
进入该类，可以看到一堆功能



点开第一个 blog

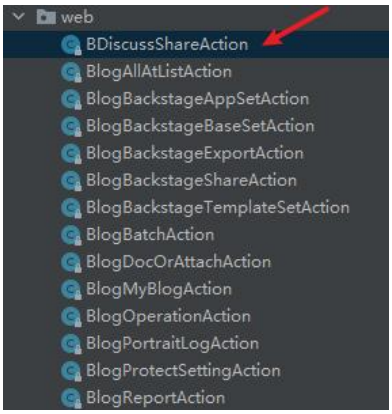


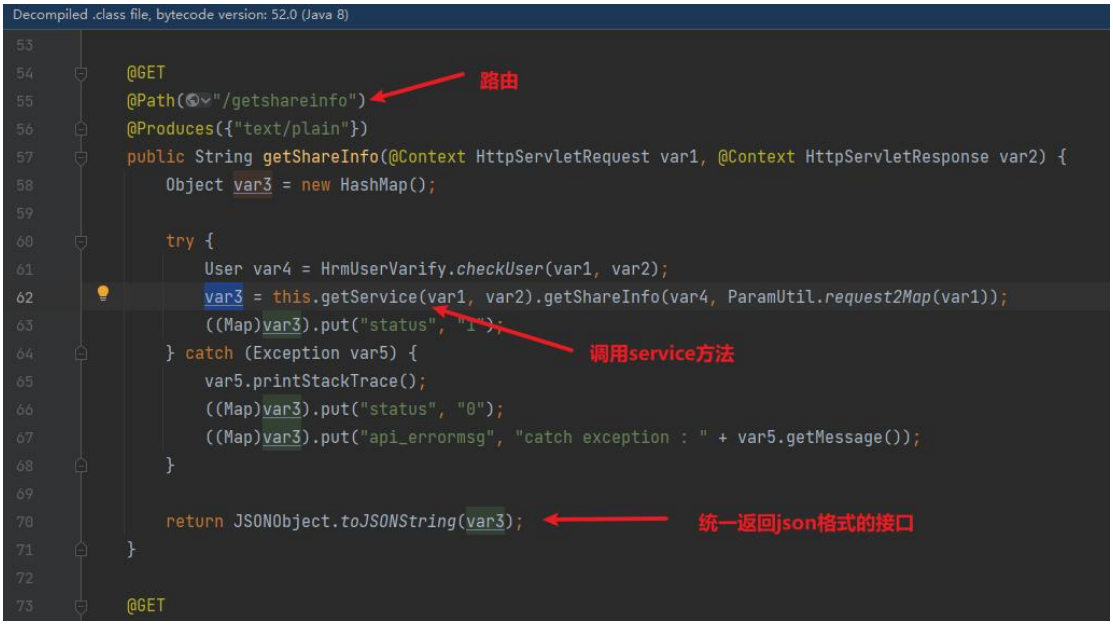
参考官方文档



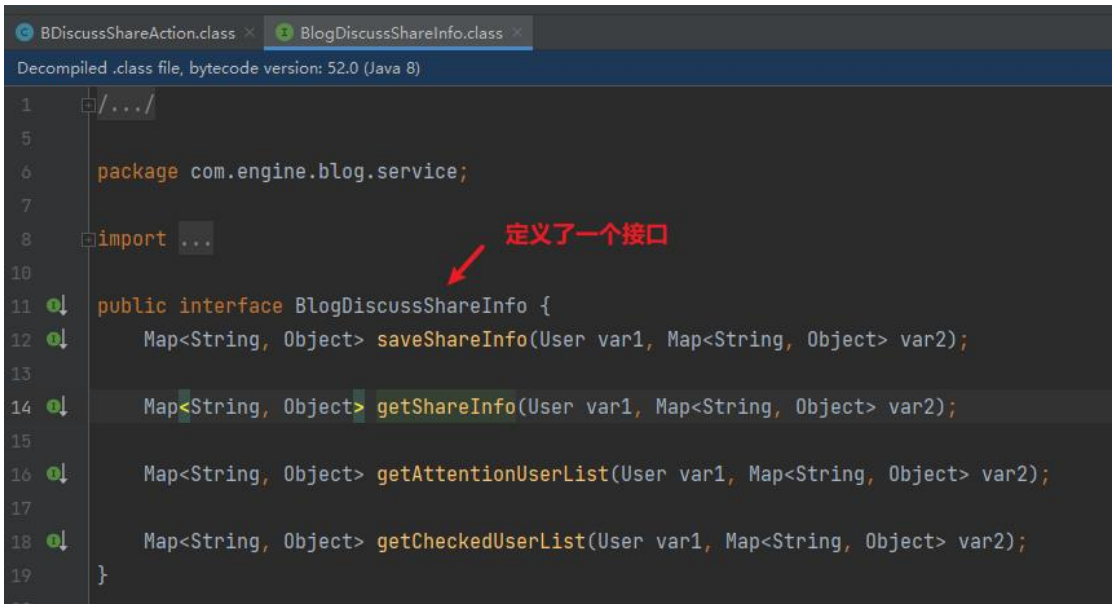
action --> 点开 web 目录

随意查看一个

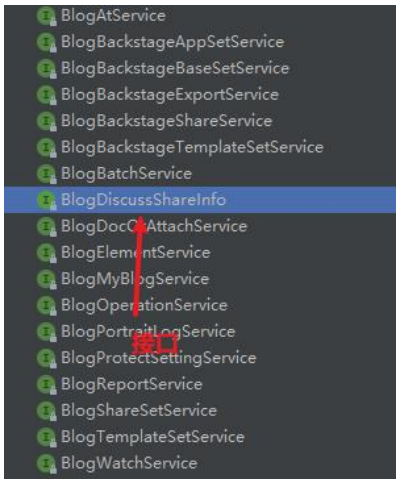




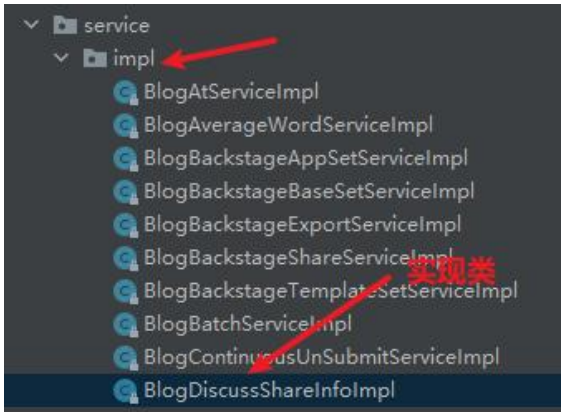
跟进这个方法



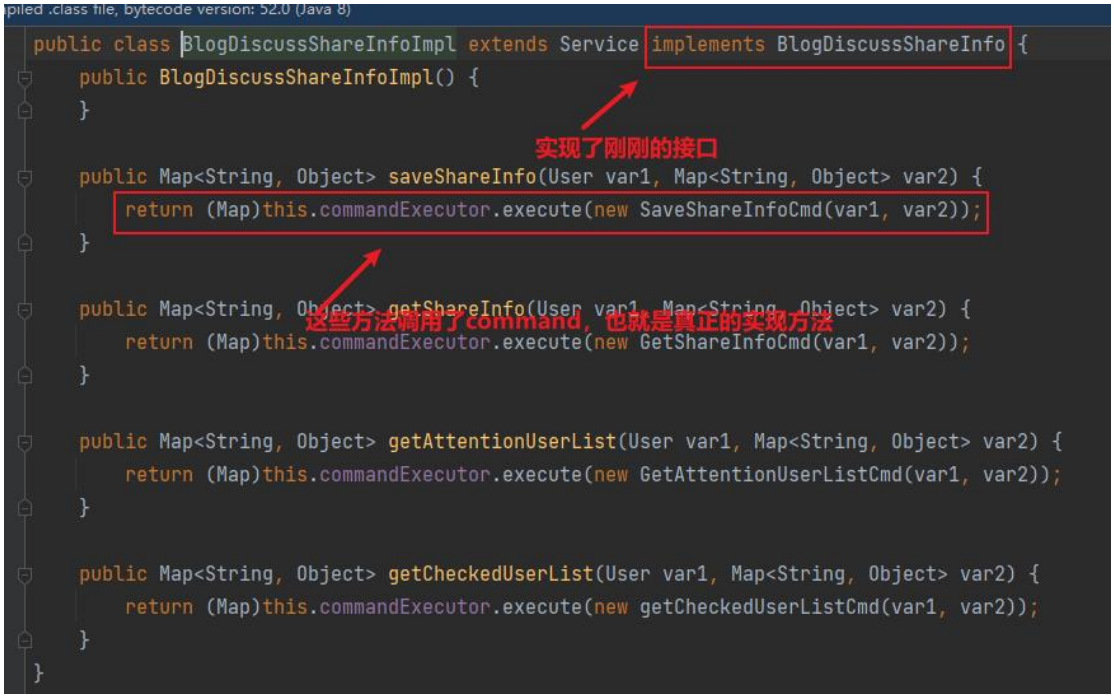
接口需要实现



在旁边找到 impl 实现类

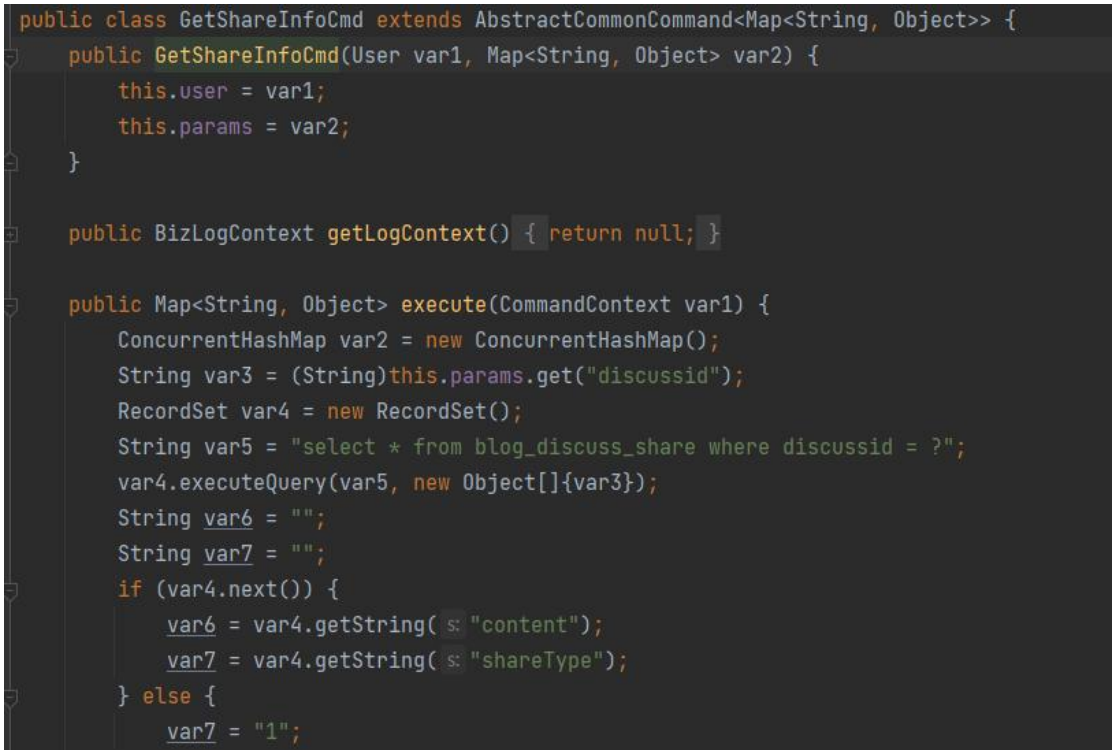


然后跟进



跟进这个方法

成功找到对应功能的实现类



方法执行完毕，return 回来参数

```
        var2.put("shareContent", var6);
        var2.put("replaceDatas", var8);
        var2.put("shareType", var7);
        return var2;
    }
```

再进行 return

```
public Map<String, Object> getShareInfo(User var1, Map<String, Object> var2) {
    return (Map)this.commandExecutor.execute(new GetShareInfoCmd(var1, var2));
}
```

最后再回到 action 层

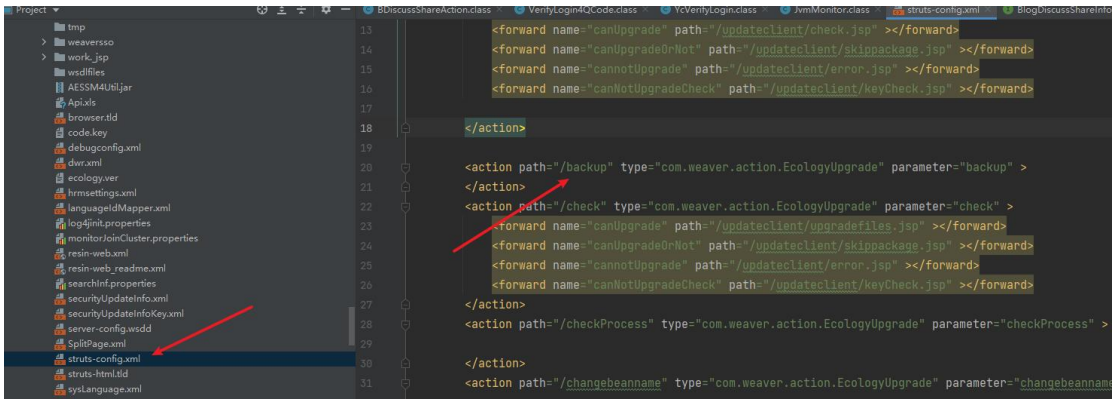
```
public String AttentionUserList(@Context HttpServletRequest var1, @Context HttpServletResponse var2) {
    Object var3 = new LinkedHashMap();

    try {                    值给到action层的var3
        User var4 = HrmUserVarify.checkUser(var1, var2);
        var3 = this.getService(var1, var2).getAttentionUserList(var4, ParamUtil.request2Map(var1));
        ((Map)var3).put("status", "1");
    } catch (Exception var5) {
        var5.printStackTrace();
        ((Map)var3).put("status", "0");
        ((Map)var3).put("api_errormsg", "catch exception : " + var5.getMessage());
    }

    return JSONObject.toJSONString(var3);   返回json格式的var3
}
```

然后更多功能的实现，无非就是更多个这样的闭环模块增加，像累积木一样。
以上就是架构部分的简单描述。

这里再补充几个识别 ssh 架构的方法
首先 structs 直接看配置文件就好了
Struts-config.xml
这个文件名是 structs 独有的



定义了路由以及对应的加载类
Done