

一款简单易用的远程日志查看器，可实时查看云服务器上的日志数据

文章分类: Project; 标签: WebSocket, SSH; 作者: Hackyle;

更新时间: Thu Aug 10 13:54:30 CST 2023

1. 项目背景

2. 功能特性

3. 技术栈

4. 本地运行

5. 设计说明

1. 后端

1. 从YML中注入日志目标的参数

2. SSH工具类

1.

3. 日志数据获取与推送逻辑

4. 整合WebSocket Server

1. 事件处理器

2. 握手拦截器

3. 对外暴露ws接口

2. 前端

1. 整合WebSocket

本文主要内容

- 介绍一款开发者工具（远程日志查看器）的使用说明和技术实现思路
- 源码地址: <https://github.com/HackyleShawe/RemoteLogViewer>

前置知识

- SpringBoot基础知识
- SSH: Secure Shell
- Web前端基础: HTML、CSS、JavaScript、jQuery
- WebSocket

如果你对以上基础技术很陌生，本篇文章内容可能不适合你！

内容导览

- [项目背景](#)
- [功能特性](#)
- [技术栈](#)
- [本地运行](#)
- [设计说明](#)
 - [后端](#)
 - [从YML中注入日志目标的参数](#)
 - [SSH工具类](#)
 - [日志数据获取与推送逻辑](#)
 - [整合WebSocket Server](#)
 - [事件处理器](#)
 - [握手拦截器](#)
 - [对外暴露ws接口](#)
 - [前端](#)
 - [整合WebSocket Client](#)
 - [显示历史日志的条数](#)
 - [抓取控制](#)
 - [页内关键字搜索](#)
 - [快速粘贴](#)
 - [手动关闭WS连接](#)
 - [后端](#)
 - [前端](#)
- [打成Jar运行](#)

项目背景

- 场景1: 在企业级开发中，公司的测试环境一般部署在某个远程的内网服务器上，我们想要查看该测试环境的日志，就需要手动建立SSH，再执行日志查看命令，在终端查看日志
- 场景2: 我们自己写的小项目部署到云服务器上后，想要查看日志，也需要通过SSH连接到云服务器，通过执行文件查看命令，来看到日志信息

在这个过程中:

1. 需要打开SSH客户端工具，例如MobaXterm、putty
2. 连接到远程服务器：输入密码、用户名
3. 手工键入日志文件查看命令：tail -f 日志文件路径
4. 在Shell Terminal查看日志
5. 存在问题：在Terminal上看得眼睛痛，不要根据关键字搜索日志，不好查看日志信息



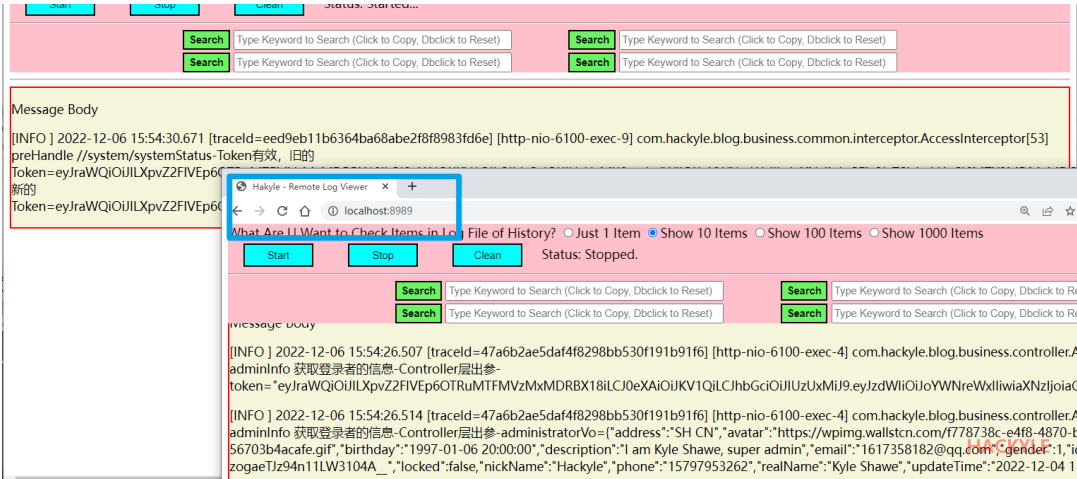
功能特性

- 支持打开**多个的前端页面**，**分别**抓取日志数据**渲染**到页面，但只能抓取一个日志文件的数据
- 可查看该日志文件的**历史数据**
- 可**实时**抓取日志文件中**新产生**的日志数据
- 对当前页面上的日志数据进行**关键字查询**
- 可在**YML配置文件中自定义SSH服务地址和日志文件的位置**

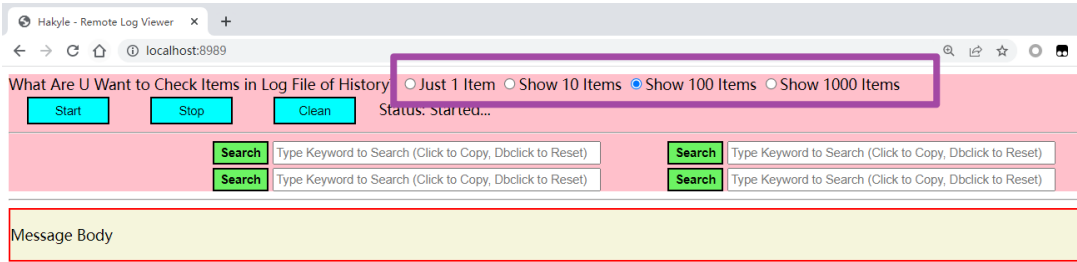
- 现阶段只支持获取文本文件中的日志数据，后续将可支持其他格式（例如**压缩文件**）的日志数据
-

Hackyle Remote Log Viewer

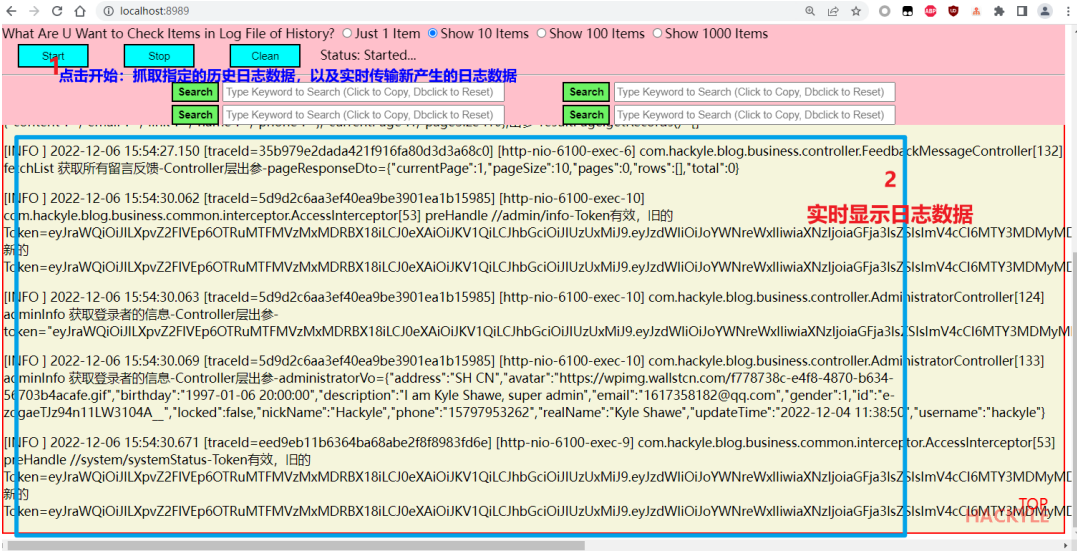
支持打开多个的前端页面，分别抓取日志数据渲染到页面，但只能抓取一个日志文件的数据



可查看该日志文件的历史数据



可实时抓取日志文件中新生成的日志数据

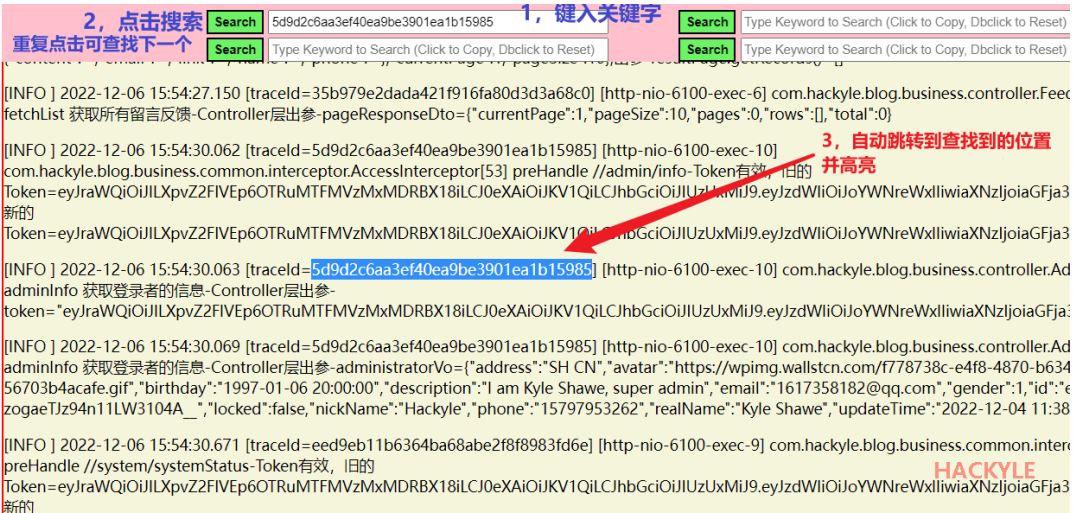


Start：开始抓取日志文件中的历史记录，然后实时获取新生成的日志

Stop：停止抓取

Clean：清除当前页面上的所有日志数据，但不会断开连接，还是会实时地呈现后端推送过来的日志信息

对当前页面上的日志数据进行关键字查询



- 单击搜索框，将粘贴板上的数据复制到这个搜索框内
- 双击搜索框，清除此个搜索框内的数据

技术栈

后端技术

- Spring Boot
- SSH客户端的Java实现工具: jsch
- Spring封装的WebSocket Server API: 将SSH中执行命令后返回的数据，推送给前端

前端技术

- jQuery
- JavaScript封装的WebSocket Client API: 接收后端发来的数据，将其渲染到HTML页面

本地运行

Step1: 环境准备或检查

- Java: 11
- SpringBoot: 2.3.12.RELEASE
- Apache Maven: 3.6.3
- Chrome Version: 108.0.5359.94, 在地址栏输入 (chrome://version/) 可获取

Step2: 克隆项目到本地，从IDEA中打开，等待Maven自动配置完毕

Step3: 填写项目的配置文件 (application.yml)

- 指定SSH的连接参数: host, port, username, password
- 远程服务器上的日志所在位置: logPath

Step4: 运行启动类: src/main/java/com/hackyle/log/viewer/RemoteLogViewerApp.java

Step5: 进入Chrome，在地址栏输入: <http://localhost:8989/>，进入日志查看首页

设计说明

主要流程

1. 前端发起一个WebSocket连接到后端
2. 连接建立成功后，后端通过SSH连接到远程服务器
3. 执行日志文件查看命令: tail -1f 日志文件的绝对路径，例如: tail -1f /data/blog.hackyle.com/blog-business-logs/blog-business.log
4. 获取到该个命令的执行结果，通过WebSocket推送到前端页面上

后端

从YML中注入日志目标的参数

```
1  log.  
2  targets:  
3      - code: A001 #需要唯一标识此条记录  
4        host: 47.97.178.120 #SSH连接参数  
5        port: 22  
6        username: root  
7        password: hackyle.1916  
8        # 远程服务器上的日志文件的绝对路径  
9        # 例: /data/logs/app.log #本质是执行命令"tail -10f /data/logs/app.log", 查看a  
10       logPath: /data/blog.hackyle.com/blog-business-logs/blog-business.log  
11     - code: A002  
12       host: 47.97.178.120  
13       port: 22  
14       username: root  
15       password: hackyle.1916  
16       logPath: /data/blog.hackyle.com/blog-consumer-logs/blog-consumer.log
```

定义实体类去映射接收: com/hackyle/log/viewer/pojo/LogTargetBean.java

注入到Spring容器: com/hackyle/log/viewer/config/LogTargetConfiguration.java

为什么不将日志目标的连接信息放置在MySQL数据库中?

- 适用于被查看的日志目标量不大、比较固定
- 这是一款面向开发人员的工具，而非面向普通用户。开发人员肯定懂得如何在YML配置文件中定义连接信息。
- 为了使得本工具更加的轻量化、便捷化，尽可能地减少依赖，因此不使用MySQL数据库。

SSH工具类

使用jsch工具模拟SSH客户端，与SSH服务端建立连接

- com/hackyle/log/viewer/util/JschUtils.java
- Session buildSshSession (String host, int port, String username, String password) 构建并返回SSH连接会话
- void releaseSshSession (Session sshSession) 释放一个SSH连接会话

日志数据获取与推送逻辑

com/hackyle/log/viewer/service/impl/LogServiceImpl.java

主要逻辑

1. 准备要执行的Shell命令: tail -1f 日志文件的绝对路径, 例如: tail -1f /data/blog.hackyle.com/blog-business-logs/blog-business.log
2. 获取sshSession, 创建一个执行Shell命令的Channel
3. 从Channel中读取流, 包装为字符流, 一次读取一行日志数据
4. 获取WebSocket Session, 只要它没有被关闭, 就将日志数据通过该Session推送出去


```
39 Session sshSession = wsSessionBean.getSshSession();
40
41 //String command = "ssh tpbbsc01 \"tail -n +count+ \"f\" +LogPath+ \"\""; //二级SSH跳板机在这里修改
42 String command = "tail -n +wsSessionBean.getHistoryItems()+ \"f\" + wsSessionBean.getLogTargetBean().getLogPath();
43 System.out.println("command: " + command);
44
45 //创建一个执行Shell命令的Channel
46 ChannelExec channelExec = (ChannelExec) sshSession.openChannel( type: "exec");
47 channelExec.setCommand(command);
48 channelExec.connect();
49 InputStream inputStream = channelExec.getInputStream();
50
51 //包装为字符流，方便每次读取一行
52 BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream, StandardCharsets.UTF_8));
53 String buf = "";
54 while ((buf = reader.readLine()) != null) {
55     if(wsSession.isOpen()) {
56         //在WebSocket中推送数据
57         wsSession.sendMessage(new TextMessage(buf));
58     }
59 }
60 }
```

HACKYLE

整合WebSocket Server

主要步骤

1. 导入WebSocket的starter依赖
2. 事件处理器：通过继承 TextWebSocketHandler 类并覆盖相应方法，可以对 websocket 的事件进行处理
3. WS握手（连接）拦截器
 - 通过实现 HandshakeInterceptor 接口来定义握手拦截器，完全等价于SpringMVC中的拦截器
 - 最佳应用场景是：通过拦截器可以对ws请求进行认证
4. 定义ws对前端暴露的API接口
 - 通过实现 WebSocketConfigurer 类并覆盖相应的方法进行 websocket 的配置。
 - 我们主要覆盖 registerWebSocketHandlers 这个方法。
 - 通过向 WebSocketHandlerRegistry 设置不同参数来进行配置。其中 addHandler方法添加我们上面的写的 ws 的 handler 处理类，第二个参数是你暴露出的 ws 路径。
 - addInterceptors 添加我们写的握手过滤器。
 - setAllowedOrigins("*") 这个是关闭跨域校验，方便本地调试，线上推荐打开。

事件处理器

com/hackyle/log/viewer/handler/LogWebSocketHandler.java

- 定义WebSocket的一系列回调函数
- 使用一个静态Map缓存当前所有已经建立了连接的会话

afterConnectionEstablished方法：连接建立成功时调用

- 创建WS会话
- 接收前端传递的参数
- 创建SSH连接会话
- 根据前端传递的targetCode获取LogTargetBean
- 缓存当前已经创建WebSocket的连接会话
- 把WebSocket会话ID先发给前端，便于前端通过该会话ID关闭WebSocket连接
- 调用日志获取服务，向前端推送日志数据

afterConnectionClosed方法：关闭连接后调用

- 从缓存中移除这个已经创建了的WebSocket连接会话

握手拦截器

com/hackyle/log/viewer/interceptor/WebSocketInterceptor.java

- beforeHandshake：在握手前触发；afterHandshake：在握手后触发。
- 功能与SpringMVC拦截器类似
- 这里获取前端传递来的一些参数：要查看的是那个目标的日志、这次查看多少条日志

- 定义ws对外的访问接口
- 将时间处理器、握手拦截器注入到WebSocketHandlerRegistry
- 设置跨域访问

前端

整合WebSocket Client

WebSocket客户端

- 初始化实例对象，打开WebSocket: var ws = new WebSocket('ws://localhost:8989/ws/hello');
- readyState中枚举了不同的状态，可根据状态指定状态（ws的建立连接、发送消息、接收消息、关闭连接）的处理逻辑
- 关闭WebSocket: close();

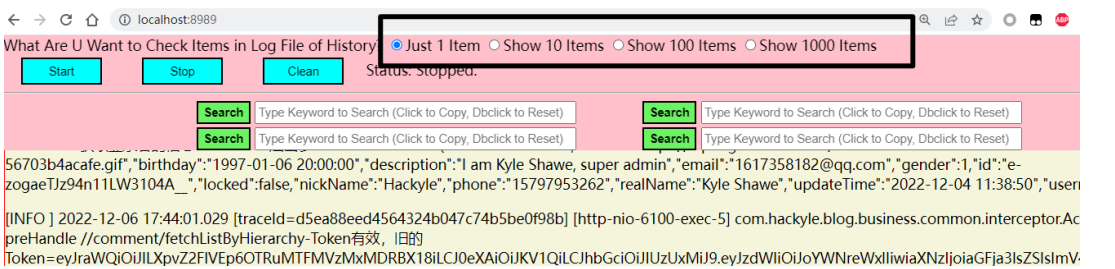


```
53  /**
54   * 发起WebSocket请求，获取数据
55   */
56   function obtainLogBySocket(count) {
57       count = count < 1 ? 1 : count;
58       websocket = new WebSocket( url: websocketRootUrl + "?count=" + count)
59       websocket.onopen = function(evt : Event ) { //连接成功后的回调函数
60           console.log("WebSocketClient Connection Opened.");
61           // websocket.send("Hello, I am Client."); //发送
62       };
63
64       websocket.onmessage = function(evt : MessageEvent ) { //接收到消息的回调函数
65           // console.log( "接收Server端发来的消息: " + evt.data); //接收
66
67           if(evt.data.startsWith("sessionId:")) {
68               //接收后端发来的本个连接Id
69               let sidArr = evt.data.split(":")
70               let sid = sidArr[1]
71               //存储本地，sessionStorage只在本个页面有效
72               window.sessionStorage.setItem("sessionId", sid)
73               // window.sessionStorage.getItem("sessionId")
74           } else {
75               writeLog2DOM(evt.data);
76           }
77       }
78   };
79
80   websocket.onclose = function(evt : CloseEvent ) { //连接断开的回调函数
81       console.log("WebSocketClient Connection closed.");
82   };
83   }
```

src/main/resources/static/js/log.js

HACKYLE

显示历史日志的条数



What Are U Want to Check Items in Log File of History: ☒ Just 1 Item ☐ Show 10 Items ☐ Show 100 Items ☐ Show 1000 Items

Start Stop Clean Status: stopped.

Search Type Keyword to Search (Click to Copy, Dbclick to Reset)

56703b4acafe.gif", "birthday": "1997-01-06 20:00:00", "description": "I am Kyle Shawe, super admin", "email": "1617358182@qq.com", "gender": 1, "id": "e-zogaeTJz94n11LW3104A_", "locked": false, "nickName": "Hackyle", "phone": "15797953262", "realName": "Kyle Shawe", "updateTime": "2022-12-04 11:38:50", "usern

[INFO] 2022-12-06 17:44:01.029 [traceld=d5ea88eed4564324b047c74b5be0f98b] [http-nio-6100-exec-5] com.hackyle.blog.business.common.interceptor.Ac

preHandle //comment/fetchListByHierarchy-Token有效，旧的

Token=eyJraWQwQ0UjLXpvZ2FIVEp6OTRuMTFhZjYxMjM0MDRBRX18iLCJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eYJzdWliOiJ0eYWNreWxliiWiaXNzIjoiaGFja3IsZSIsImV

抓取控制



Start: 开始抓取日志文件中的历史记录，然后实时获取新产生的日志

Stop: 停止抓取

Clean: 清除当前页面上的所有日志数据，但不会断开连接，还是会实时地呈现后端推送过来的日志信息

为三个按钮分别添加一个Click事件，定义动作函数

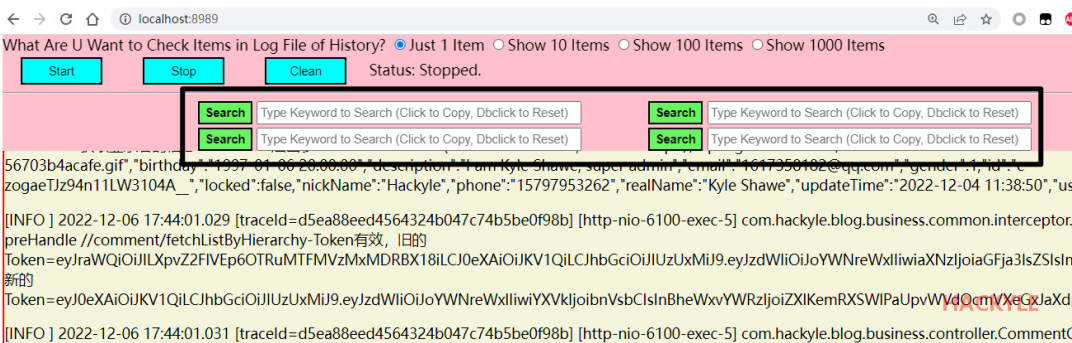
Start: 创建WebSocket实例，将后端发来的数据，不断追加到某个标签下

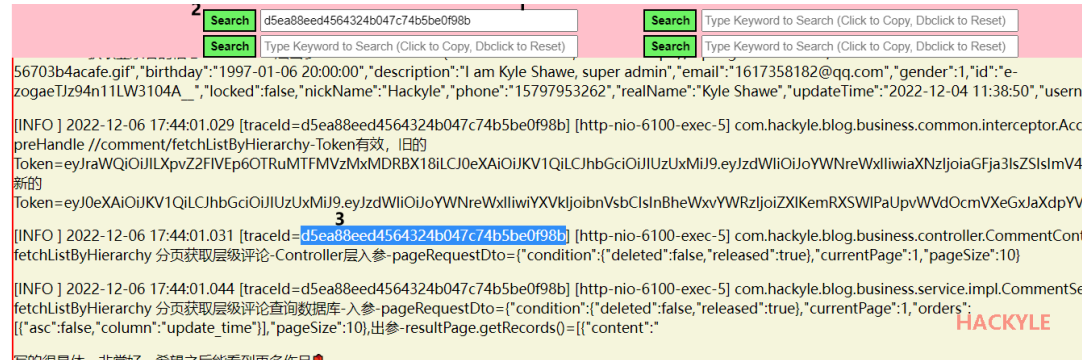
Stop: 前端手动关闭WebSocket，请求后端接口，关闭WebSocket Server

```
log.js
1 let websocketRootUrl = "ws://localhost:8989/ws/log"
2 let websocket = null;
3
4 $(function () {
5     console.log("Remote Log Viewer Designed and Implementer By Hackyle Shawe.");
6 });
7
8 // 点击了页面上的开始按钮
9 $("#start").click(function () {...});
10
11 // 点击了页面上的停止按钮
12 $("#stop").click(function () {...});
13
14 // 点击了页面上的清除按钮
15 $("#clean").click(function () {...});
```

页内关键字搜索

在本个页面内，进行关键字搜索。本质是模拟浏览器的Ctrl+F，进行HTML内容搜索





调用window.find()方法

- 官方文档：<https://developer.mozilla.org/zh-CN/docs/Web/API/Window/find>
- API：find(aString, aCaseSensitive, aBackwards, aWrapAround, aWholeWord, aSearchInFrames, aShowDialog);
- 参数释义
 - aString：将要搜索的字符串
 - aCaseSensitive：布尔值，如果为true,表示搜索是区分大小写的。
 - aBackwards：布尔值。如果为true, 表示搜索方向为向上搜索。
 - aWrapAround：布尔值。如果为true, 表示为循环搜索。

快速粘贴

单击搜索框，将粘贴板上的数据复制到此个搜索框内

- 获取到该个搜索框
- 调用execCommand(“copy”)，把粘贴板上的数据写入

使用第三方库clipboard接管粘贴板

```
1  /**
2   * 复制内容到剪贴板
3   * Notice: 需要导入clipboard.min.js
4   * @param content 要复制的内容
5   */
6  function copyHandle(content){
7      let copy = (e)=>{
8          e.preventDefault()
9          e.clipboardData.setData('text/plain',content)
10         // alert('复制成功')
11         document.removeEventListener('copy',copy)
12     }
13     document.addEventListener('copy',copy)
14     document.execCommand("Copy");
15 }
```

双击搜索框，清除此个搜索框内的数据

- 添加一个双击事件
- 清除元素内的值

手动关闭WS连接

背景

- 如果直接在Client端直接关闭，在Server端会抛异常（Caused by: java.io.IOException: 你的主机中的软件中止了一个已建立的连接。）
- 所以，后端设计一个接口，当要关闭某个WebSocket连接时，请求该个接口，并携带上WebSocket的SessionId

设计思想

- 在前后端建立连接时，后端就把sessionId放入缓存，并响应给前端
- 前端得到sessionId，将其放在sessionStorage中，目的是使得该个id仅在本页面内有效

后端

接收前端请求: com/hackyle/log/viewer/controller/LogController.java

```
LogController.java
48 /**
49  * 提供一个普通接口，强制关闭WebSocketServer端
50  */
51 @RequestMapping("/log/stop")
52 @ResponseBody
53 public String stopWebSocket(@RequestParam("sid") String sid) {
54     if(null == sid || "".equals(sid.trim())) {
55         return "SessionID缺失";
56     }
57
58     return logService.closeWebSocketServer(sid) ? "WebSocketServer关闭成功" : "WebSocketServer关闭失败";
59 }
```

业务: com/hackyle/log/viewer/service/impl/LogServiceImpl.java#closeWebSocketServer

实现: com/hackyle/log/viewer/handler/LogWebSocketHandler.java#closeWebSocketServer

前端

存入sessionStorage: src/main/resources/static/js/log.js

```
log.js
57 /**
58  * 发起WebSocket请求，获取数据
59  */
60 function obtainLogBySocket(targetCode, historyItems) {
61     historyItems = historyItems < 1 ? 1 : historyItems;
62     websocket = new WebSocket( url: websocketRootUrl + "?targetCode=" + targetCode + "&historyItems=" + historyItems );
63     websocket.onopen = function( evt : Event ) { //连接成功后的回调函数
64         console.log("WebSocketClient Connection Opened.");
65         // websocket.send("Hello, I am Client."); //发送
66     };
67
68     websocket.onmessage = function( evt : MessageEvent ) { //接收到消息的回调函数
69         // console.log( "接收Server端发来的消息: " + evt.data); //接收
70         if(evt.data.startsWith("sessionId:")) {
71             //接收后端发来的本个连接id
72             let sidArr = evt.data.split(":");
73             let sid = sidArr[1]
74             //存储本地，sessionStorage只在本个页面有效
75             window.sessionStorage.setItem("sessionId", sid)
76             // window.sessionStorage.getItem("sessionId")
77         } else {
78             writeLog2DOM(evt.data);
79         }
80     }
81 };
82
```

关闭WebSocket连接时，携带sessionId: src/main/resources/static/js/index.js

```
log.js
19 //点击了页面上的停止按钮
20 $("#stop").click(function () {
21     //关闭WebSocket连接
22     if(websocket !== null && websocket.readyState === WebSocket.OPEN) {
23         let sessionId = window.sessionStorage.getItem( key: "sessionId")
24         if(sessionId === null || sessionId === '') {
25             return
26         }
27
28         //发送Ajax请求，告诉Server端我要关闭了，你也关闭吧
29         $.get("/log/stop?sid="+sessionId, function (data) {
30             console.log("The '/log/stop' Response Close Status: ", data)
31         });
32     }
33 });
```

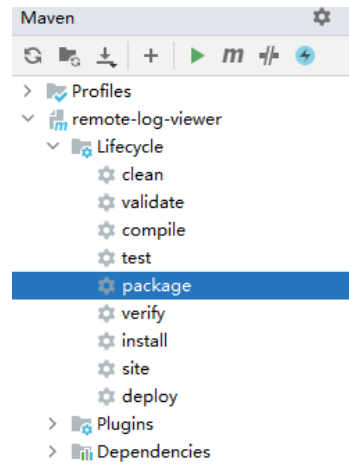
✂

1. 每次需要查看日志时，都需要打开IDE环境，也挺麻烦的
2. 解决办法是将本项目打成Jar，一键启动

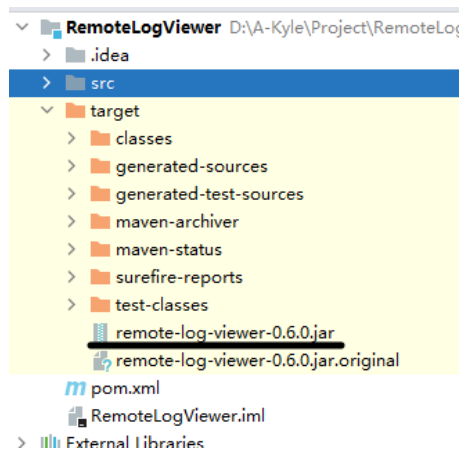
Step1: 在POM.xml中添加打包插件



Step2: 执行打包命令



Step3: 将Jar放在合适的位置



Step4: 写个启动脚本。本项目基于JDK11，建议手动设置临时的JDK环境变量，再启动Jar

```
1 # Windows操作系统批处理脚本，文件拓展名为: .cmd
2 set JAVA_HOME=D:\ProgramFilesKS\Java\JDK11
3 set path=%JAVA_HOME%\bin;%path%
4
5 java -jar D:\D-Project\DevelopTools\remote-log-viewer-0.6.0.jar
6
```

版权声明：非明确标注皆为原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上本文链接及此声明。
原文链接：<https://blog.hackyle.com/article/project/remote-log-viewer>

留下你的评论

Name:

Email:

Link:

[illegible]