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：Wed Dec 07 09:48:35 CST 2022

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

- 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，我们想要查看该个测试环境的日志，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

在我我司的内网测试环境，查看日志

那么，有没有办法把这一过程自动化呢？答案是肯定的。这便是本项目的设计初衷与目的！

功能特性

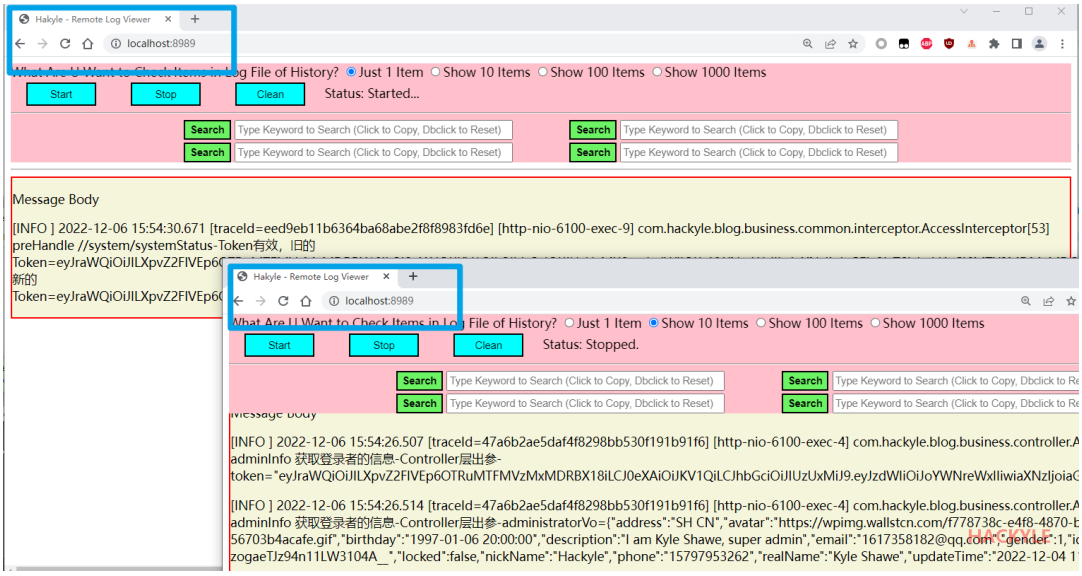
功能特性列表

- 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

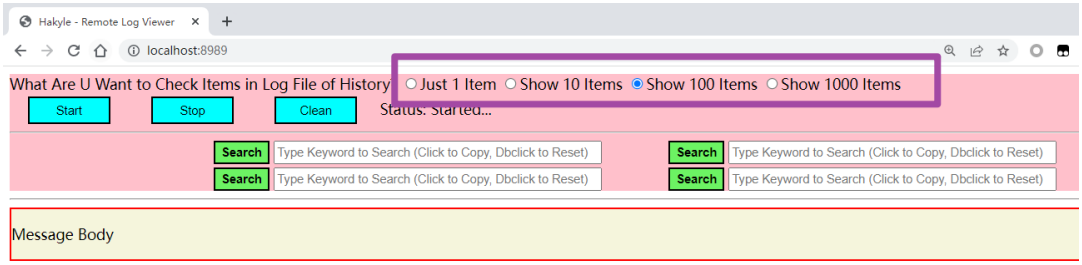
后续将会支持的功能

- 用户可以在前端页面上自定义SSH服务地址和日志文件的位置，并支持保存，一次配置，以后可以多次使用
- 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
- ……

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

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

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此个搜索框内
- 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：jsch开头的一系列配置参数
- 远程服务器上的日志所在位置：log-path

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

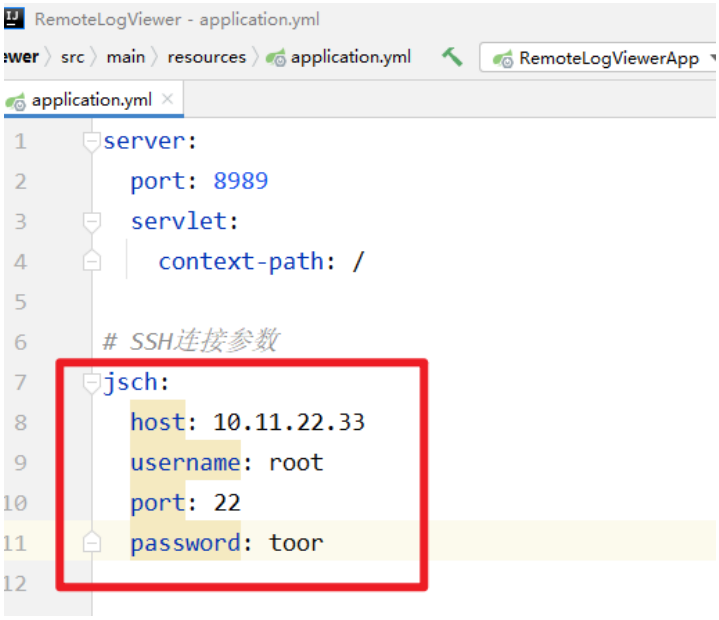
后端

整合SSH

主要步骤

1. 导入jsch的POM依赖
2. 在配置文件（yml）中定义SSH的连接参数
3. 写一个业务类，定义创建SSH会话、关闭会话的方法
    1. 使用@Value(“${jsch.host}”)注解从配置文件中载入参数
    2. 创建会话方法：Session buildConnect()
    3. 关闭会话方法：void destroyConnect(Session sshSession)

application.yml



JschService

```java
@Override
public Session buildConnect() {
    Session sshSession = null;
    try {
        JSch jSch = new JSch(); //创建一个ssh通讯核心类
        sshSession = jSch.getSession(username, host, port); //传主机、端口、用户名获

        Properties config = new Properties();
        config.put("StrictHostKeyChecking","no"); //不进行严格模式检查
        sshSession.setPassword(password); //设置密码
        sshSession.setConfig(config);

        sshSession.connect(); //连接会话

        if(sshSession.isConnected()) {
            System.out.println("SSH连接成功: " + sshSession.getHost() + ":" + sshSe
        } else {
            throw new RuntimeException("SSH连接失败");
        }
    } catch (Exception e) {
        System.out.println("SSH连接出现异常: " + e);
    }

    return sshSession;
}

@Override
public void destroyConnect(Session sshSession) {
    if(sshSession != null) {
        sshSession.disconnect();
        if(!sshSession.isConnected()) {
            System.out.println("SSH已断开连接: " + sshSession.getHost()+":"+ sshSes
        }
    }
}
```

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

LogServiceImpl.java

```
39 @    public void sendLog2BrowserClient(WsSessionBean wsSessionBean) throws Exception {
40          WebSocketSession wsSession = wsSessionBean.getWebSocketSession();
41          Session sshSession = wsSessionBean.getSshSession();
42
43          //从域对象中获取调用WebSocketClient传递过来的参数
44          Object countObj = wsSession.getAttributes().get("count");
45          int count = 1;
46          if(countObj != null) {
47              try {
48                  count = Integer.parseInt(String.valueOf(countObj).trim());
49              } catch (Exception e) {
50                  System.out.println("read2Integer转换出现异常：" + e);
51              }
52          }
53
54          //String command = "ssh tpbbsc01 \"tail -" +count+ "f " +logPath+ "\""; //二级SSH跳板机在这里修改
55          String command = "tail -" +count+ "f " +logPath;
56          System.out.println("command: " + command);
57
58          //创建一个执行Shell命令的Channel
59          ChannelExec channelExec = (ChannelExec) sshSession.openChannel( type: "exec");
60          channelExec.setCommand(command);
61          channelExec.connect();
62          InputStream inputStream = channelExec.getInputStream();
63
64          //包装为字符流，方便每次读取一行
65          BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream, StandardCharsets.UTF_8));
66          String buf = "";
67          while ((buf = reader.readLine()) != null) {
68              if(wsSession.isOpen()) {
69                  //往WebSocket中推送数据
70                  wsSession.sendMessage(new TextMessage(buf));
71              }
72
```


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

- 缓存当前已经创建WebSocket的连接会话
- 创建一个SSH会话，也放入缓存
- 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
- 执行日志查看命令，向前端推送日志数据

afterConnectionClosed方法：关闭连接后调用

- 从缓存中移除该个已经创建了的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查看多少条历史日志的参数

对外暴露ws接口

com/hackyle/log/viewer/config/WebSocketConfig.java

- 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

前端

整合WebSocket Client

WebSocket客户端

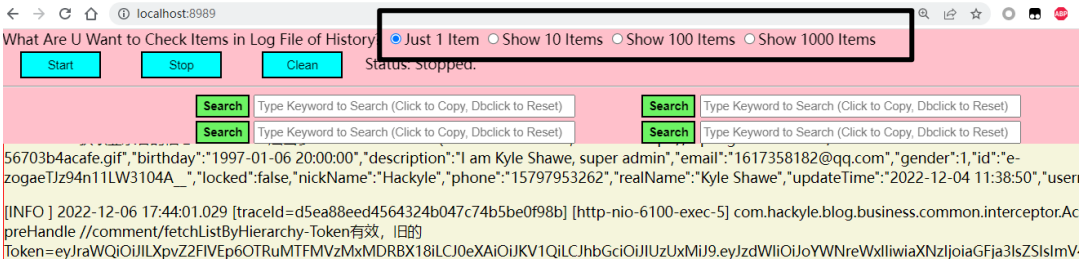
- 初始化实例对象，打开**WebSocket**：var ws = new WebSocket('ws://localhost:8989/ws/hello');
- **readyState**中枚举了不同的**状态**，可根据状态指定状态（ws的建立连接、发送消息、接收消息、关闭连接）的处理逻辑
- **关闭WebSocket**：close();

index.js

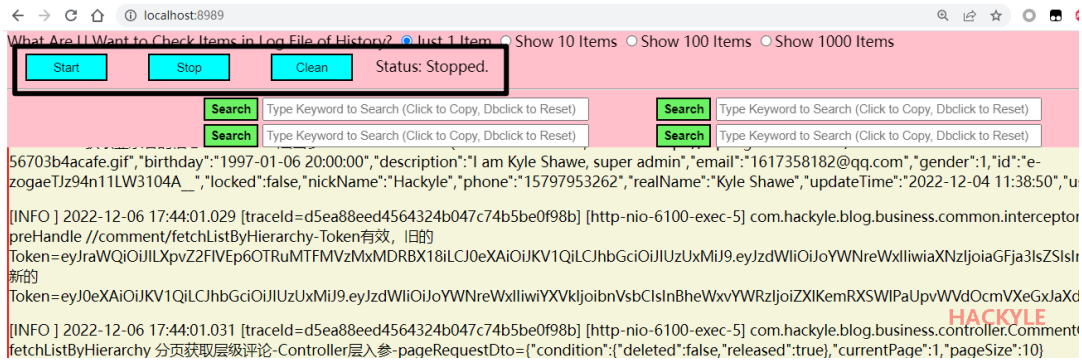
```
53  /**
54   * 发起WebSocket请求，获取数据
55   */
56  function obtainLogBySocket(count) {
57      count = count < 1 ? 1 : count;
58      webSocket = new WebSocket( url: webSocketRootUrl + "?count=" + count)
59      webSocket.onopen = function(evt : Event ) { //连接成功后的回调函数
60          console.log("WebSocketClient 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
75          } else {
76              writeLog2DOM(evt.data);
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  }
84
```


src/main/resources/static/js/index.js

显示历史日志的条数



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

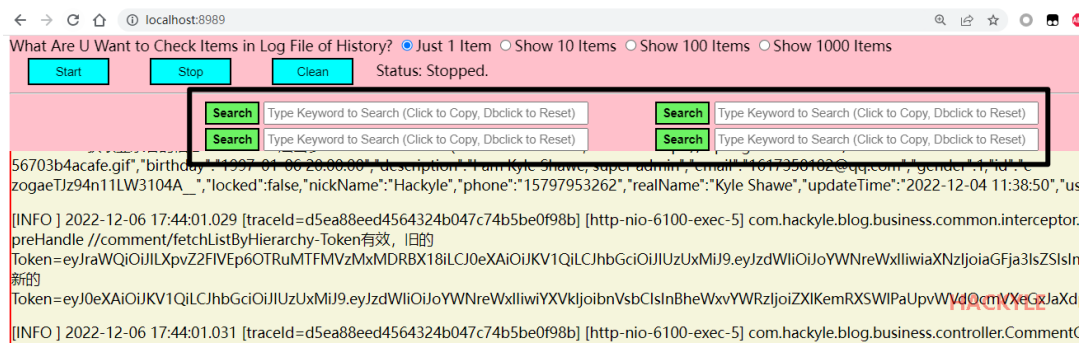
index.js

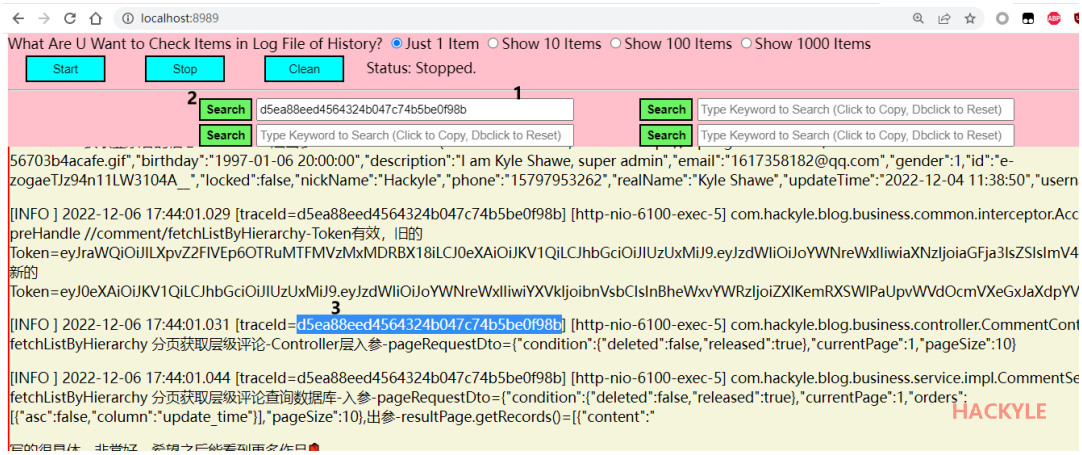
```
let webSocketRootUrl = "ws://localhost:8989/log/ws"
let webSocket = null;

$(function () {
    console.log("Remote Log Viewer Designed and Implementer By Hackyle Shawe.
});

//点击了页面上的开始按钮
$("#start").click(function () {...});

//点击了页面上的停止按钮
$("#stop").click(function () {...});

//点击了页面上的清除按钮
$("#clean").click(function () {...});
```

src/main/resources/static/js/index.js

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

```
/**
 * 复制内容到剪贴板
 * Notice：需要导入clipboard.min.js
 * @param content 要复制的内容
 */
function copyHandle(content){
    let copy = (e)=>{
        e.preventDefault()
        e.clipboardData.setData('text/plain',content)
        // alert('复制成功')
        document.removeEventListener('copy',copy)
    }
    document.addEventListener('copy',copy)
    document.execCommand("Copy");
}
```

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

1. 在前后端建立连接时，后端就把sessionId放入缓存，并响应给前端
2. 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

3. 前端在请求关闭接口时，携带上该个id
4. 后端移除该个id的缓存，并关闭所有会话信息

后端

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

LogController.java

```java
package com.hackyle.log.viewer.controller;

import com.hackyle.log.viewer.service.LogService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class LogController {
    @Autowired
    private LogService logService;

    /**
     * 提供一个普通接口，强制关闭WebSocketServer端
     */
    @RequestMapping("/log/stopWebSocket")
    public boolean stopWebSocket(@RequestParam("sid") String sid) {
        if(null == sid || "".equals(sid.trim())) {
            return false;
        }

        return logService.closeWebSocketServer(sid);
    }
}

```

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

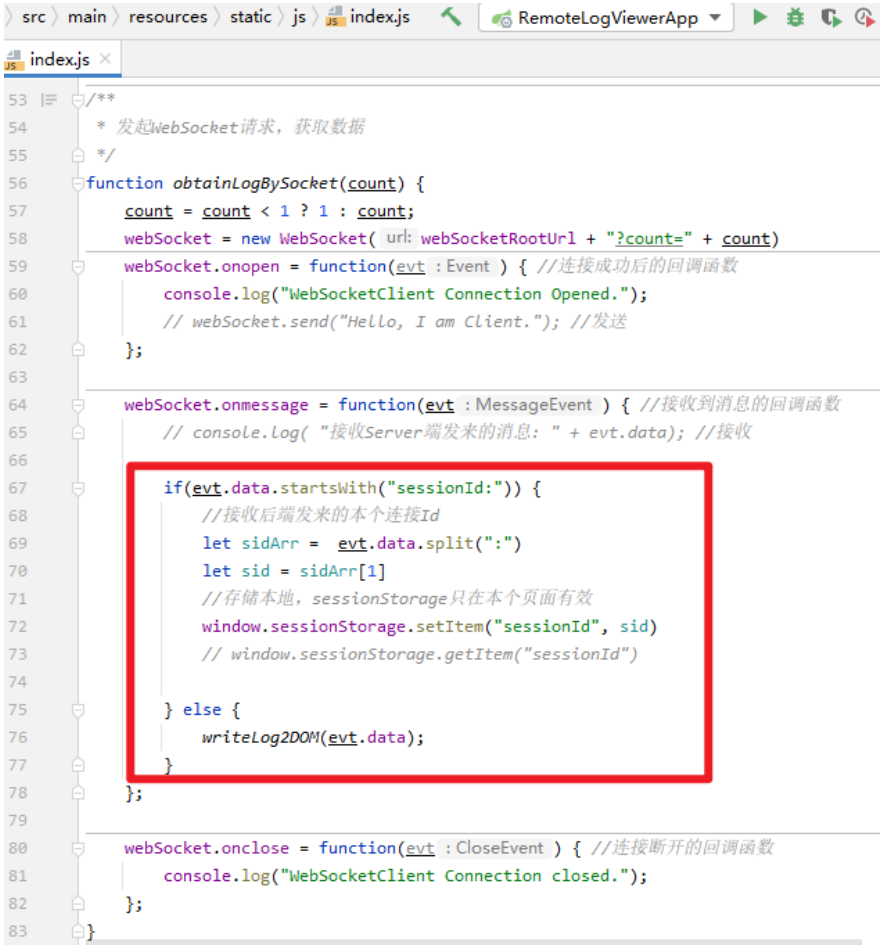
实现：com/hackyle/log/viewer/handler/LogWebSocketHandler.java#closeWebSocketServer

LogWebSocketHandler.java

```java
    /**
     * 关闭连接后调用
     * @param session 连接
     * @param status 状态
     */
    @Override
    public void afterConnectionClosed(WebSocketSession session, CloseStatus statu
        livingSessionMap.remove(session.getId());
        System.out.println("WebSocketServer已关闭: " + session.getId());
    }

    public boolean closeWebSocketServer(String sid) {
        WsSessionBean wsSessionBean = livingSessionMap.get(sid);
        if(null != wsSessionBean) {
            try {
                //关闭WebSocket、SSH的连接会话
                wsSessionBean.getWebSocketSession().close();
                jschService.destroyConnect(wsSessionBean.getSshSession());

                return true;
            } catch (IOException e) {
                System.out.println("closeWebSocketServer出现异常: "+e);
            }
        }

        return false;
    }
}
```

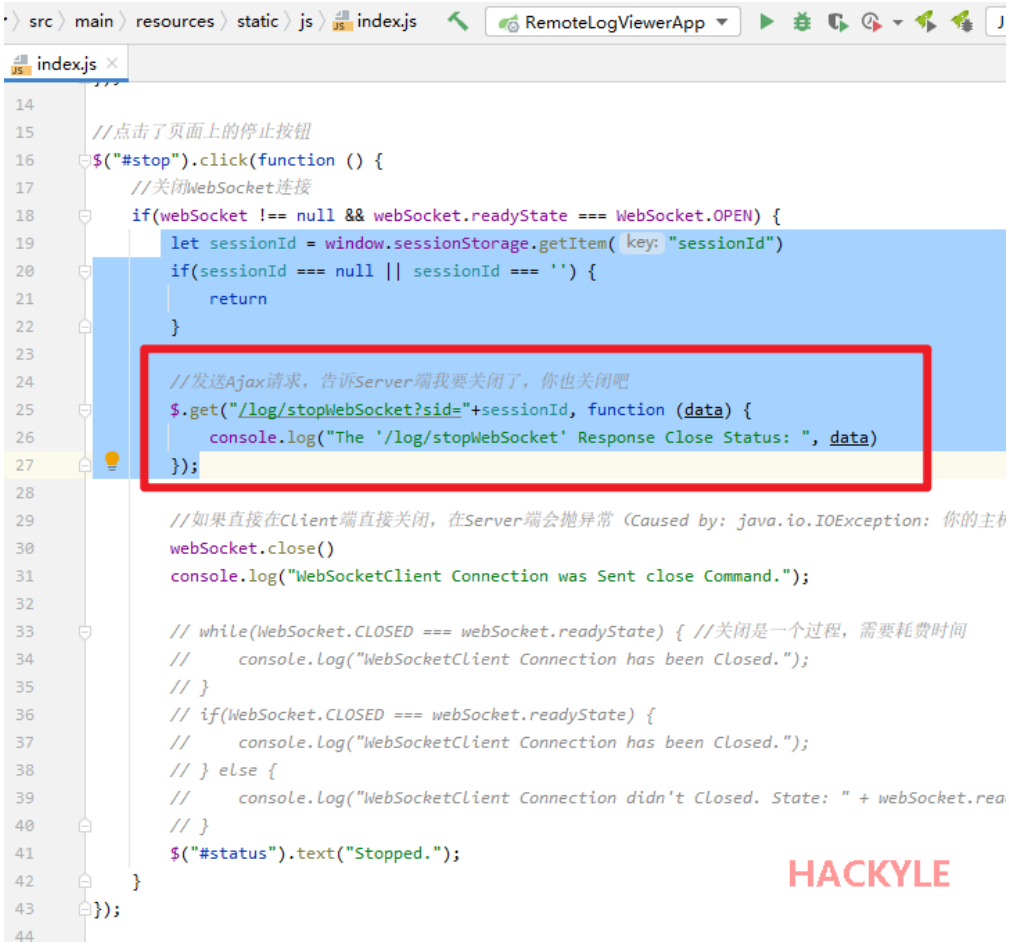
前端

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



```
/**
 * 发起WebSocket请求，获取数据
 */
function obtainLogBySocket(count) {
    count = count < 1 ? 1 : count;
    webSocket = new WebSocket( url: webSocketRootUrl + "?count=" + count)
    webSocket.onopen = function(evt : Event ) { //连接成功后的回调函数
        console.log("WebSocketClient Connection Opened.");
        // webSocket.send("Hello, I am Client."); //发送
    };

    webSocket.onmessage = function(evt : MessageEvent ) { //接收到消息的回调函数
        // console.log( "接收Server端发来的消息: " + evt.data); //接收

        if(evt.data.startsWith("sessionId:")) {
            //接收后端发来的本个连接Id
            let sidArr =  evt.data.split(":")
            let sid = sidArr[1]
            //存储本地，sessionStorage只在本个页面有效
            window.sessionStorage.setItem("sessionId", sid)
            // window.sessionStorage.getItem("sessionId")

        } else {
            writeLog2DOM(evt.data);
        }
    };

    webSocket.onclose = function(evt : CloseEvent ) { //连接断开的回调函数
        console.log("WebSocketClient Connection closed.");
    };
}
```

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



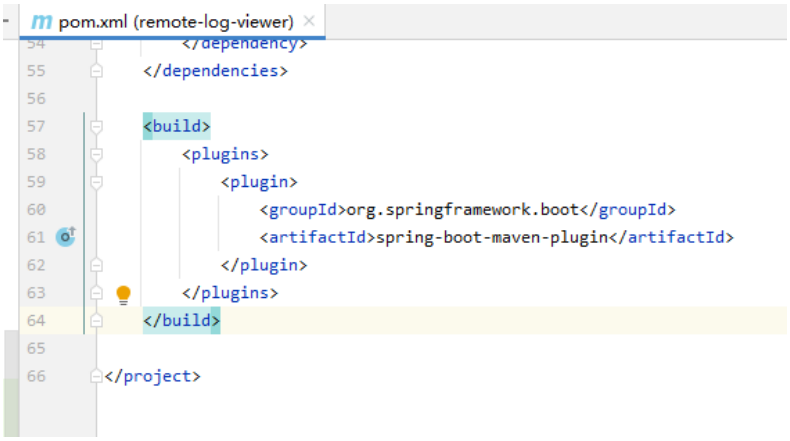
```
//点击了页面上的停止按钮
$("#stop").click(function () {
    //关闭WebSocket连接
    if(webSocket !== null && webSocket.readyState === WebSocket.OPEN) {
        let sessionId = window.sessionStorage.getItem( key: "sessionId")
        if(sessionId === null || sessionId === '') {
            return
        }

        //发送Ajax请求，告诉Server端我要关闭了，你也关闭吧
        $.get("/log/stopWebSocket?sid="+sessionId, function (data) {
            console.log("The '/log/stopWebSocket' Response Close Status: ", data)
        });

        //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
        webSocket.close()
        console.log("WebSocketClient Connection was Sent close Command.");

        // while(WebSocket.CLOSED === webSocket.readyState) { //关闭是一个过程，需要耗费时间
        //     console.log("WebSocketClient Connection has been Closed.");
        // }
        // if(WebSocket.CLOSED === webSocket.readyState) {
        //     console.log("WebSocketClient Connection has been Closed.");
        // } else {
        //     console.log("WebSocketClient Connection didn't Closed. State: " + webSocket.rea
        // }
        $("#status").text("Stopped.");
    }
});
```


打成Jar运行

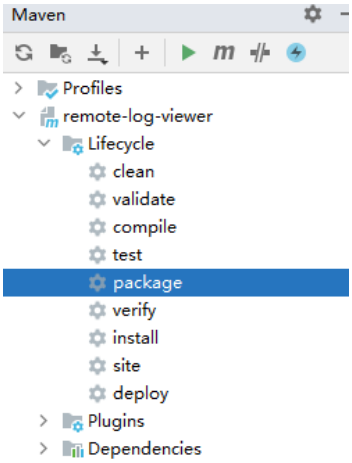
背景

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

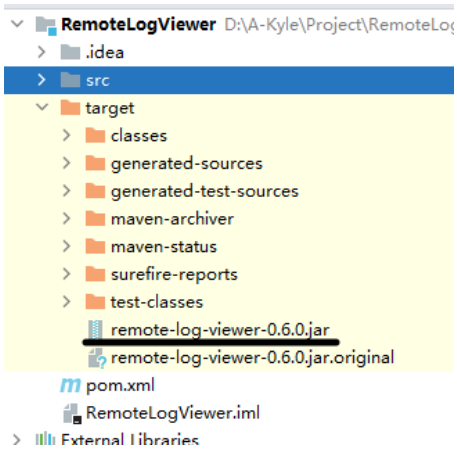
Step1：在POM.xml中添加打包插件



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

```
# Windows操作系统批处理脚本，文件拓展名为：.cmd
set JAVA_HOME=D:\ProgramFilesKS\Java\JDK11
set path=%JAVA_HOME%\bin;%path%

java -jar D:\D-Project\DevelopTools\remote-log-viewer-0.6.0.jar

pause
```

留下你的评论

Name: Input your name, please

Email: Input your email, please

Link: Input your phone or website, please

File Edit View Format Tools Table Help

Input comment, please

p 0 words tiny

SUBMIT RESET