链滴

vscode 插件开发经历

作者：someone27889
原文链接：https://ld246.com/article/1564566088575
来源网站：链滴

对vscode一无所知的前提下开发了聊天室插件
https://github.com/ferried/hacpai-cr

1.明确需求

首先，明确了自己的需求和开发步骤

1.必须登录得到token设置到cookie里
2.通过查看浏览器得到了wss的链接，并通过D大获取了必要的ws头信息
3.得出了需要让用户输入用户名密码，基于nodejs发送请求到hacpai换取token
4.换取token存储到vscode中等待wss连接时使用
5.wss设置头User-Agent等等等
6.wss返回信息类型划分为online为获取当前在线用户msg为用户输入的信息

所以我需要

1.一个输入框(用户名密码发消息等)
2.一个消息列表(用来显示消息)
3.一个在线用户列表

通过官方的GettingStart一步步走下去了解到
1.输入框为vscode.window.InputBox
2.vscode原生列表(文件列表)为树格式需要自己写provider提供data

最后找到一个官方项目集合示例
https://github.com/Microsoft/vscode-extension-samples/tree/master/tree-view-sample
并仿照尝试,可以渲染数据才开始进行开发

2.分阶段开发

将需求拆分成几个步骤
1.用户认证
2.长链接
3.渲染数据

3.进行开发

1.首先安装脚手架生成项目

```
npm install -g yo generator-code
yo projectName
```

2.command和package.json

Ctrl+Shift+P呼出命令行界面输入插件命令何以出发函数主要是通过package.json定义好的comman

名称作为入口

commands就是配置该插件当前所有的命令比如下面配置出了如下几个命令

```
//extension.hacpaicr.signin-命令id,在函数中使用
//Hacpaicr: Signin-命令值,在  Ctrl+Shift+P 命令输入框中使用
extension.hacpaicr.signin:Hacpaicr: Signin
extension.hacpaicr.signout:Hacpaicr: Signout
```

最后把配置的命令加入到activationEvents数组中表示启用此命令

```
 "activationEvents": [
     "onCommand:extension.hacpaicr.signin",
    "onCommand:extension.hacpaicr.signout",
    "onCommand:extension.hacpaicr.connect",
     "onCommand:extension.hacpaicr.send",
    "onView:view.hacpai.cr",
    "onView:view.hacpai.cru"
 ],
"contributes": {
    "viewsContainers": {
        "activitybar": [{
            "id": "hacpaicr",
            "title": "hacpaicr",
            "icon": "resources/hacpai.svg"
        }]
    },
    "views": {
        "hacpaicr": [{
                "id": "view.hacpai.cr",
                "name": "ChatRoom"
            },
            {
                "id": "view.hacpai.cru",
                "name": "Users"
            }
        ]
    },
    "commands": [{
            "command": "extension.hacpaicr.signin",
            "title": "Hacpaicr: Signin"
        },
        {
            "command": "extension.hacpaicr.signout",
            "title": "Hacpaicr: Signout"
        },
        {
            "command": "extension.hacpaicr.connect",
            "title": "Hacpaicr: Connect"
        },
        {
            "command": "extension.hacpaicr.send",
            "title": "Hacpaicr: Send"
        }
```

```
    ]
}
```

用户认证

项目入口为src/xtension.ts

```
// context为当前插件的域对象
export function activate(context: vscode.ExtensionContext) {
   // 创建一个User对象并将当前域传入其中
    const user = new User(context);

// vscode注册命令,传入的值为`extension.hacpaicr.signin`也就是package.json中声明的命令id,当户输入此命令的Title敲击回车,将出发此函数
    vscode.commands.registerCommand(COMMADN_SIGN_IN, async() = >{
        await user.sign();
    });

  ......
```

user.sign()

```
async sign() {
    // 获取用户输入的用户名
    await this.inputUserName();
   // 获取用户输入的密码
    await this.inputPassword();
  // 如果少了一个
    if (!this.userName || !this.userPassword) {
        // 呼出右下角的错误提示框提示信息并中断函数执行
        vscode.window.showErrorMessage("you must input the username or password");
        return;
    }
   // 进行用户认证
    await this.auth();
   // 获取wsToken
    await this.wsAuth();
}

async inputUserName() {
     // 这是vscode输入框调用方式，直接获取了输入框的值
    await vscode.window.showInputBox({
        password: false,
        ignoreFocusOut: true,
        placeHolder: "input your username",
        prompt: "use enter to the next steup"
    }).then(async username = >await(this.userName = username as string));
}

// 注意此时password为true,用户无法看到自己输入了什么,密码进行`encryption`函数加密，我没有储用户密码和用户名
async inputPassword() {
    await vscode.window.showInputBox({
```

```
        password: true,
        ignoreFocusOut: true,
        placeHolder: "input your password",
        prompt: "use enter to signin"
    }).then(async passoword = >await(this.userPassword = encryption(passoword as string)));
}

// 用户认证
async auth() {
  // axios发送了请求
    await axios.post($SIGN_URL, {
        userName: this.userName,
        userPassword: this.userPassword
    },
    {
      // header中携带`User-Agent`传入后台进入统计信息
        headers: {
            "User-Agent": this.headerUtil.agent
        }
    }).
    // 如果抛异常了,那么直接在右下角提示出来比如后台会返回`用户名密码错误`
    catch(async error = >await vscode.window.showErrorMessage(error.message)).then(async(r
sponse: any) = >{
      // 如果有token,通过tokenUtil存入vscode全局的State管理器中
     // 此token为cookie携带token还需要获取wsToken
        if (response.status === 200) {
            if (response.data.sc === 0) {
                await this.tokenUtil.saveSignToken(response.data.token);
            } else {
                await vscode.window.showErrorMessage(response.data.msg);
            }
        } else {
            await vscode.window.showErrorMessage("Server Status 500");
        }
    });
}

// 获取ws的token
// 从D大处了解了发送请求到主页获取Document页面可以过滤出wsToken
async wsAuth() {
    await axios.get($INDEX_URL, {
        headers: this.headerUtil.headers
    }).
    catch(async error = >{
        await vscode.window.showErrorMessage(error);
    }).then(async(response: any) = >{
      // 这里感谢 @jinjianh 提供正则表达式！谢谢你鸭~
        let regex: RegExp = /(?<=wsToken=)[0-9a-zA-Z]+/;
        let wsToken = (regex.exec(response.data) as RegExpExecArray)[0];
      // 一样存入vscode全局的state管理器中
        await this.tokenUtil.saveWsToken(wsToken);
     // 最后提示登陆成功
        await vscode.window.showInformationMessage("sign in successed!");
    });
```

```
}
```

至此第一步骤用户认证完成

Ws连接

```
// 当用户输入 connect命令时触发该函数
vscode.commands.registerCommand(COMMAND_CONNECT, async() = >{
    await chatRoomClient.connect();
    vscode.window.registerTreeDataProvider(VIEW_CHAT_ROOM, chatRoomMessageProvider);
    vscode.window.registerTreeDataProvider(VIEW_CHAT_ROOM_USERS, chatRoomUserProvid
r);
});

// 连接
connect() {
   // ws头必须设置好了
    let headers = {
        Host: $HACPAI_HOST,
        Cookie: this.headerUtil.cookie,
        "User-Agent": this.headerUtil.cookie,
        Upgrade: "websocket"
    };
  // 这里用的webscoket包的client连接服务器
    this.client.on("connect", (connection: WebSocket.connection) = >{
        // 异常处理直接右下角提示
        connection.on("error", error = >{
            vscode.window.showErrorMessage(error.message);
        });
        connection.on("close", (code, desc) = >{
            vscode.window.showErrorMessage(`$ {
                code
            }: $ {
                desc
            }`);
        });
     // 这里获取服务器推送的消息
        connection.on("message", data = >{
            if (data.type === "utf8") {
                let msg = JSON.parse(data.utf8Data as string);
               // 我丢？这个console没删干净
                console.log(msg);
                switch (msg.type) {
               // online类型标识当前在线用户
                case "online":
                    let users:
                    Array < ChatRoomUser > =[];
                    msg.users.forEach((user: any) = >{
                        users.push(new ChatRoomUser(user.userName));
                    });
                    // 这个一会儿再说，就是给视图服务推了一个数据去刷新vscode视图了
                    this.userDataProvider.setUsers(users);
                    break;
                case "msg":
```

```
                // 接收到的消息推送给Message视图服务刷新视图
                this.messageDataProvider.add(new ChatRoomMessage(msg.content));
                break;
            }
        }
    });
});
this.client.on("connectFailed", error => {
    vscode.window.showErrorMessage(error.message);
});
 // LINKSTART 连接开始
this.client.connect(this.headerUtil.wsuri, [], $HACPAI_ORIGIN, headers, {
    headers: headers
});
}
```

视图

视图容器由package.json配置,id为hacpaicr一个容器中可以有多个view在views中配置

```
"viewsContainers": {
    "activitybar": [{
        "id": "hacpaicr",
        "title": "hacpaicr",
        "icon": "resources/hacpai.svg"
    }]
},
"views": {
    "hacpaicr": [{
            "id": "view.hacpai.cr",
            "name": "ChatRoom"
        },
        {
            "id": "view.hacpai.cru",
            "name": "Users"
        }
    ]
},
```

在入口中创建视图数据提供器并注册给视图通过视图ID就可以将数据和视图绑定了

```
const chatRoomMessageProvider = new ChatRoomMessageProvider();
const chatRoomUserProvider = new ChatRoomUserProvider();

// 第一参为视图ID:如view.hacpai.cr
 vscode.window.registerTreeDataProvider(
    VIEW_CHAT_ROOM,
    chatRoomMessageProvider
  );
  vscode.window.registerTreeDataProvider(
    VIEW_CHAT_ROOM_USERS,
    chatRoomUserProvider
```

```
    );
  });
```

视图数据提供器

```
// 这是一个消息数据提供器,需要实现`vscode.TreeDataProvider`以标识这是TreeDataView数据提
器

export class ChatRoomMessageProvider implements vscode.TreeDataProvider < ChatRoomM
ssage > ,
vscode.Disposable {
   // 这个数组是用来显示消息列表的所以是一个数组
    private history: Array < ChatRoomMessage > =[];

    // tree数据改变事件
    private _onDidChangeTreeData: vscode.EventEmitter < ChatRoomMessage | undefined > =
ew vscode.EventEmitter < ChatRoomMessage | undefined > ();
    readonly onDidChangeTreeData: vscode.Event < ChatRoomMessage | undefined > =this._
nDidChangeTreeData.event;

    // 一个TreeView有多个TreeItem,这个TreeItem类似拦截器可以拦截每一个TreeItem进行修改
    getTreeItem(element: ChatRoomMessage) : vscode.TreeItem | Thenable < vscode.TreeItem
> {
        return element;
    }
 // 这里主要是返回该TreeView数据用的,直接返回声明的消息列表
    getChildren(element ? :ChatRoomMessage | undefined) : vscode.ProviderResult < ChatRo
mMessage[] > {
        return Promise.resolve(this.history);
    }

// 刷新当前View
    refresh() {
        this._onDidChangeTreeData.fire();
    }
// 这里主要是给WebSocketClient用的,当收到信息后,插入到队列最前方,然后刷新视图
    add(chatRoomMessage: ChatRoomMessage) {
        this.history.unshift(chatRoomMessage);
        this.refresh();
    }
// 忘了...
    dispose() {
        this.history = [];
        this._onDidChangeTreeData.dispose();
    }
// 清楚消息列表
    clean() {
        this.history = [];
        this.refresh();
    }

}
// TreeView要显示的TreeItem类型
export class ChatRoomMessage extends vscode.TreeItem {}
```

再来看如何在WebSocket中使用定义的数据提供器

```
// 主要通过构造函数把引用传递过来
export class ChatRoomClient {
  // 存储域
   private context: vscode.ExtensionContext;
  // 消息数据提供器
   private messageDataProvider: ChatRoomMessageProvider;
 // 用户数据提供器
   private userDataProvider: ChatRoomUserProvider;
   private client: WebSocket.client;
   private headerUtil: HeaderUtil;
   constructor(context: vscode.ExtensionContext, messageDataProvider: ChatRoomMessageP
ovider, userDataProvider: ChatRoomUserProvider) {
       this.context = context;
  // 这里传递引用
       this.messageDataProvider = messageDataProvider;
       this.userDataProvider = userDataProvider;
       this.headerUtil = new HeaderUtil(context);
       this.client = new WebSocket.client();
   }

...
//接收到消息直接调用提供器的函数添加数据数据类型为TreeItem类型,这里只用了一个content参数,
有toolip?desc?没有用到
case "msg":
   this.messageDataProvider.add(new ChatRoomMessage(msg.content));
   break;
```

TODOLIST

希望社区朋友们可以推PR,写了三天了,今天老大问我你禅道东西怎么那么多没做,差点被锤.....我去研究avinci了

1.注销功能
2.注销后清除数据列表和数据提供器
3.现在消息列表全为html,所以需要过滤信息和圈人的信息(@88250)
4.消息太长时不太友好
5.消息清除功能没有写
6.没有显示出是谁发的几点发的消息
7.没有设置应用市场图标和应用市场详细信息

为何写这个

1.没写过插件,想试试

2.聊天室挺好玩

3.希望大佬们可以抽时间完善这个或者重写一个更好用的,推荐vscodewebView这样代码鸭图片呀啥就可以解析出来了类似Slack

鸣谢

推pr帮改下README.md写个鸣谢列表呗,嘿嘿嘿

@88250 大哥联调
@Vanessa svg图标
@jinjianh 正则表达式
@csfwff 老哥提供建议