**本 科 实 验 报 告**

实验名称：vscode插件开发

学 员： 马铁诚 刘思睿 韩启华 高雪原

学 号： 201902001031 201902001038 201902001045 201902001048

培养类型： 无军籍本科 年 级： 2019级

专 业： 软件工程 所属学院： 计算机学院

指导教员： 陈振邦 职 称： 教授

实 验 室： 306-705 实验日期： 2022.7.12

国防科学技术大学训练部制

一、实验要求：

1. 四人一组，自行挑选针对C/C++或Java程序的静态或动态测试工具，将其作为服务端完成扩展到vscode插件中。静态测试工具参考：CppCheck、KLEE、Clang-tidy等；动态测试工具参考：AFL、LibFuzzer等
2. 代码架构

采用客户端服务器架构，推荐使用gRPC进行客户端与服务端之间的通信

1. 客户端展示内容要求

针对不同类型的测试工具，应该能够对测试结果有直观的展示

例如，对于静态代码分析工具插件，如C/C++的静态代码分析工具CppCheck的vscode插件Cppcheck plug-in，在分析结束后应该能够在控制台Output中输出较为详细的分析结果；对于符号执行工具插件，如KLEE的vscode插件，应该能够在分析结束后生成并直观的展示测试用例；对于动态测试工具插件，如AFL的vscode插件，应该能够在测试结束后展示出特殊的测试用例（让程序崩溃的测试用例等）以及覆盖率信息。

1. 实验环境：

1.Ubuntu系统（18.04或20.04）。

2.vscode

3.nodejs（JavaScript的运行环境）

4.npm（nodejs的包管理工具）

1. 测试工具选择：
2. 选择CppCheck作为服务端工具
3. 插件前端选择使用JavaScript编写，后端使用python编写，使用gRPC进行通讯
4. 实验内容：

1.配置插件开发项目所需要的环境，包括下载并更新vscode，nodejs，npm等，然后在命令行输入Npm install @grpc/grpc-js

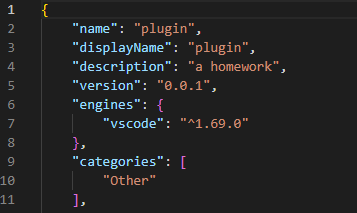
Npm install @grpc/proto-loader

安装所需要的包扩展。

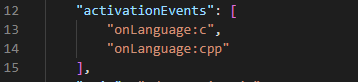
然后执行命令npm install -g yo generator-code，安装脚手架，接着进入工作目录，使用脚手架创建工程，其中language选择JavaScript，并设施好插件名称。

2. package.json内容详解

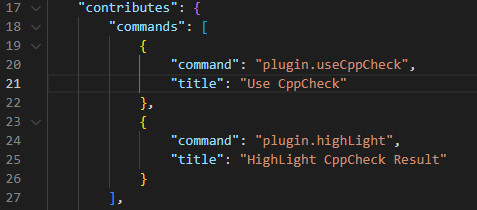
本次插件实验的插件名设置为“plugin”，其在显示在插件市场的名称也是“plugin”，插件版本号为“0.0.1”，如图所示：



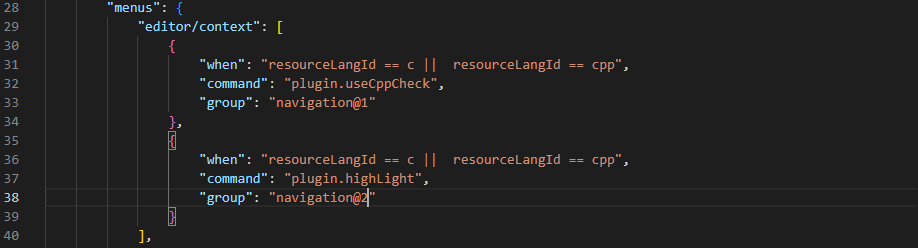
插件的激活数组设置为识别到使用语言为C语言或者cpp时激活，对应于集成在后端的cppcheck工具。如图所示：



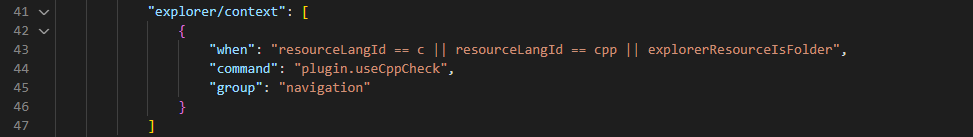
在contributes配置项中声明所开发插件使用的命令，包括两个，分别是plugin.useCppCheck使用cppcheck工具，和plugin.highlight将运行cppcheck工具产生的结果使用高亮在源代码中展示出来。如图所示：



然后定义菜单，首先将上述两个命令加入编辑器右键菜单中从而实现在代码中单机右键激活菜单后能显示并选择配置项中的两个命令，并设置为当源文件为C语言或者cpp编写时才在菜单中出现，如图所示：



同时，将plugin.useCppCheck命令添加到资源管理器右键菜单，如图所示:



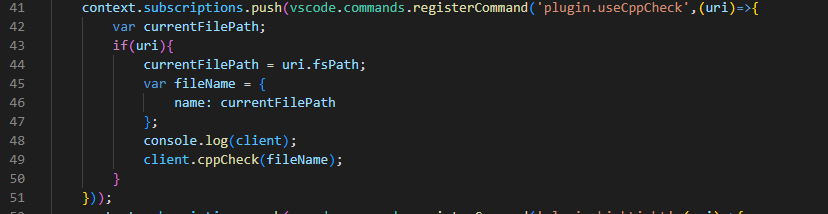
1. extension.js内容详解

设置activate()激活函数，分别注册并实现package.json中的两个命令。

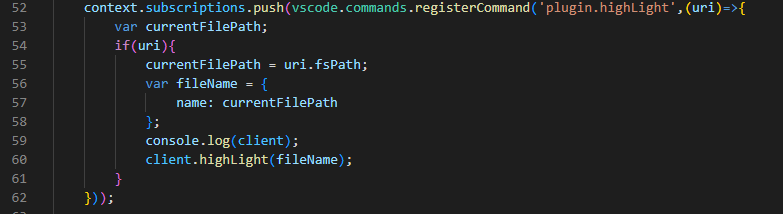
首先引入写好的client文件，其内部实现了传输数据的具体函数，将其命名为client。



对于plugin.useCppCheck命令，设置可选参数uri，当从编辑器右键菜单执行时将当前打开文件路径uri传进去。因此在函数内部判断是否得到当前文件的路径uri，如果得到，则将其作为参数传入client文件中实现的cppcheck函数。如图所示：



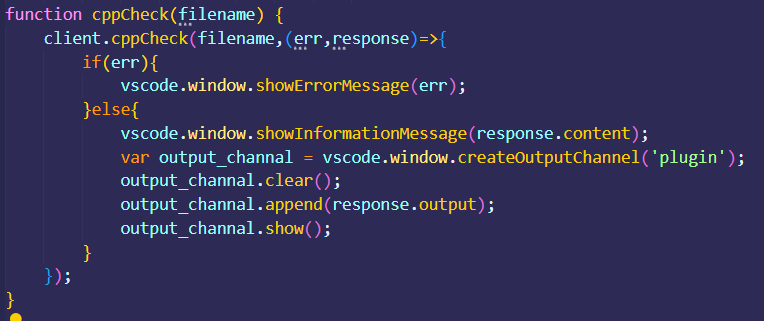
对于plugin.highLight命令，与上述命令类似，不同点为将得到的当前路径作为参数传入client文件的highLight函数。如图所示：

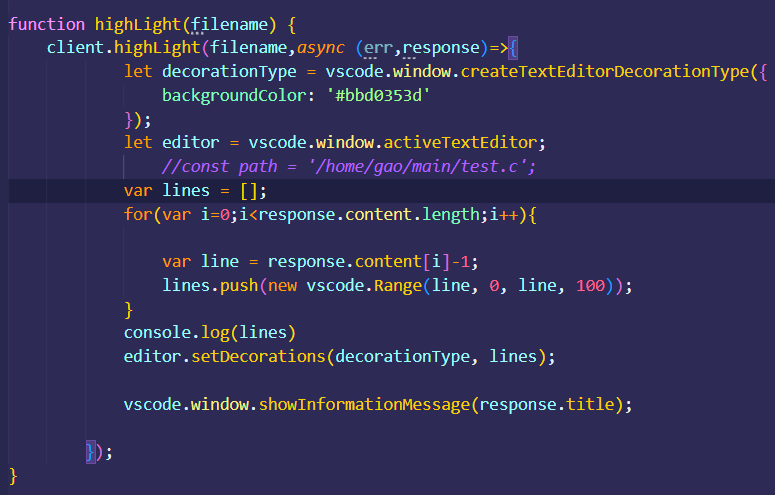


1. 安装插件： pip3 install grpcio grpcio-tools，sudo apt install cppcheck
2. 修改test.proto，指定包、函数、返回值、值的特性。（1,2代表次序）



1. Client.js：实现extension.js中调用的相关函数的具体情况，



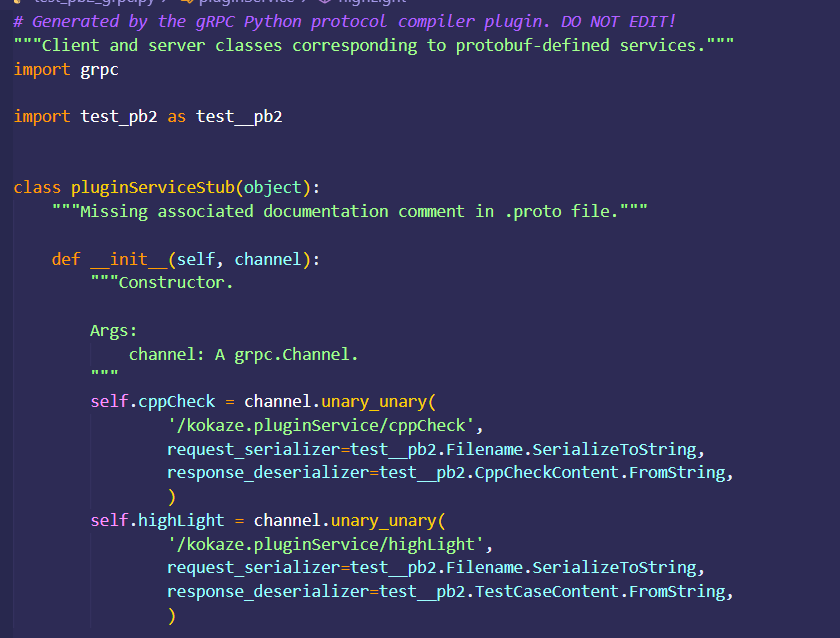


上述的cppcheck的函数，具体情况是：先监听后端client的情况，若有返回查看是报错还是信息，如果是报错就展示错误信息，否则将其返回值（对用户的发出提示框，对具体情况输出至output）

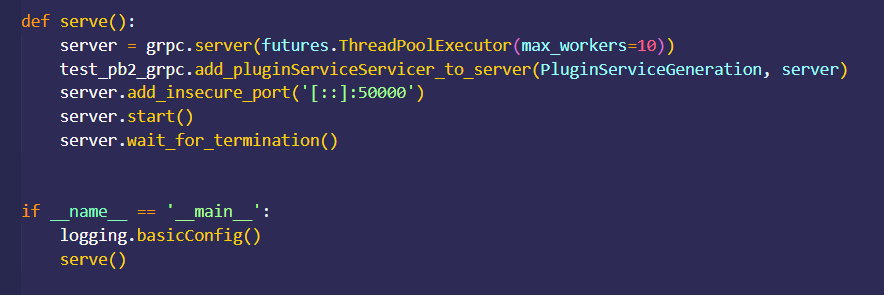
上述的highlight的函数使得在编辑器内对后端返回的报错的相应的行数标黄。

1. 生成辅助py文件：

python -m grpc\_tools.protoc -I . --python\_out=. --grpc\_python\_out=. ./test.proto

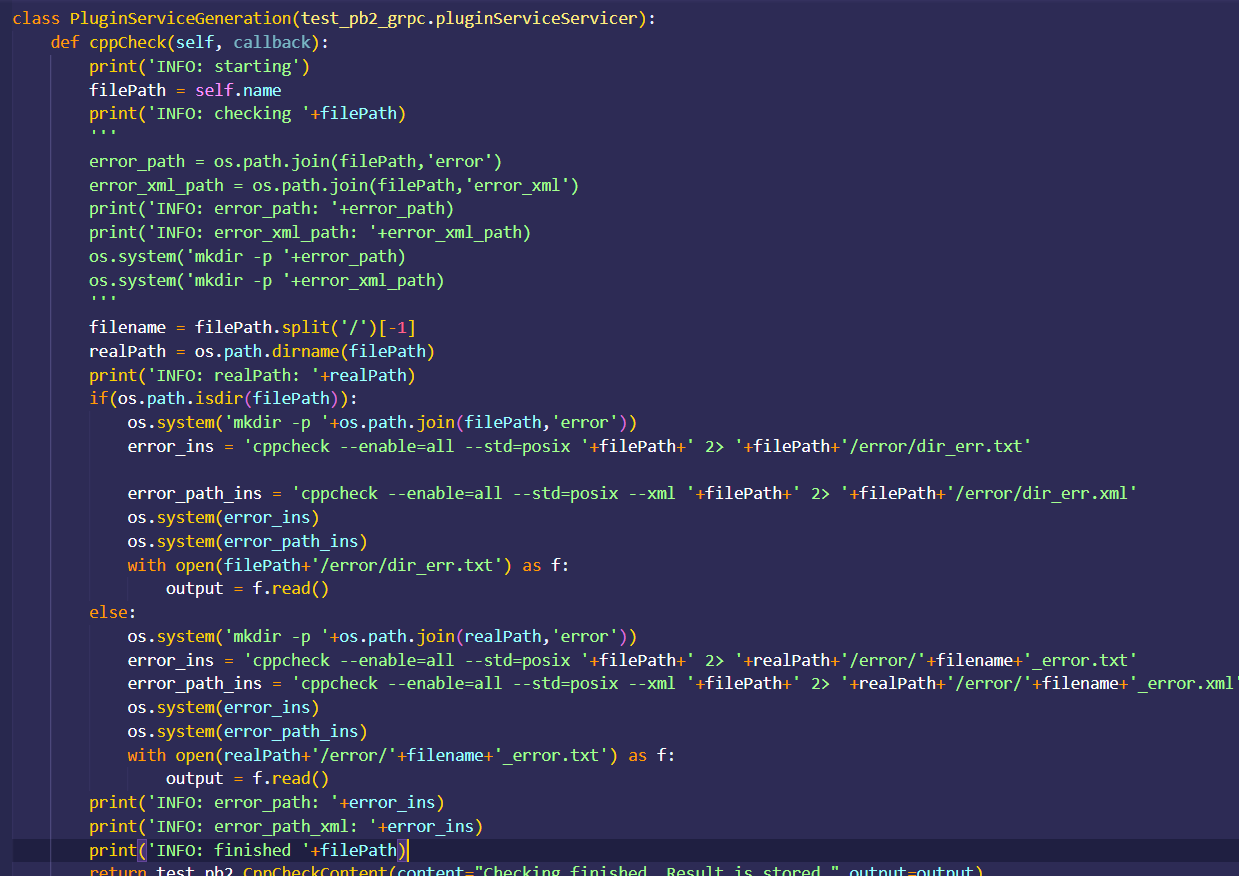


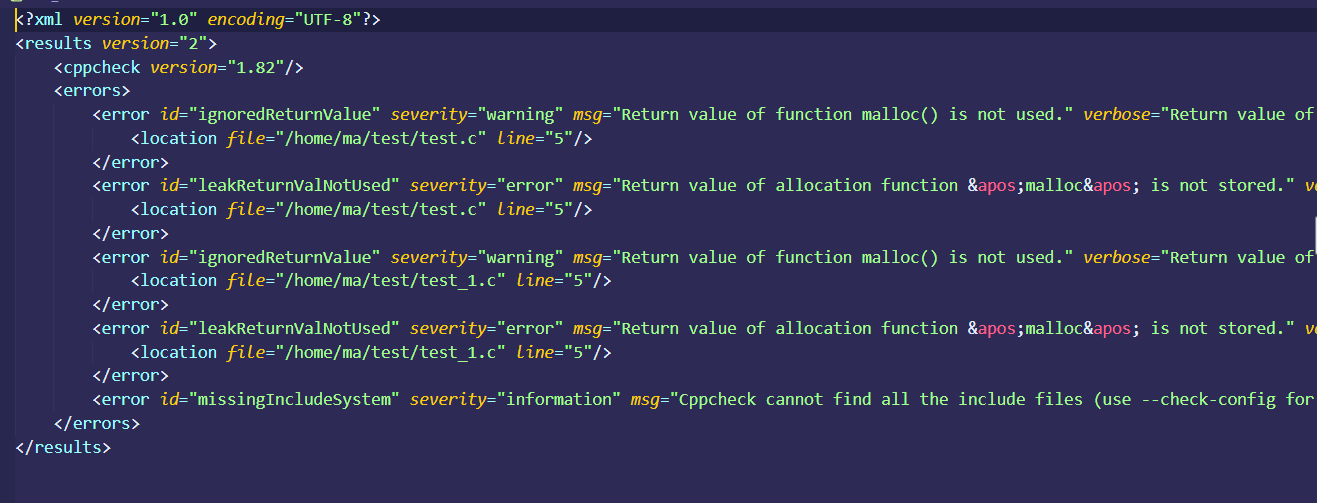
1. 编写后端代码cppcheck.py：这里要做的是首先连接服务器：（与client.js握手）



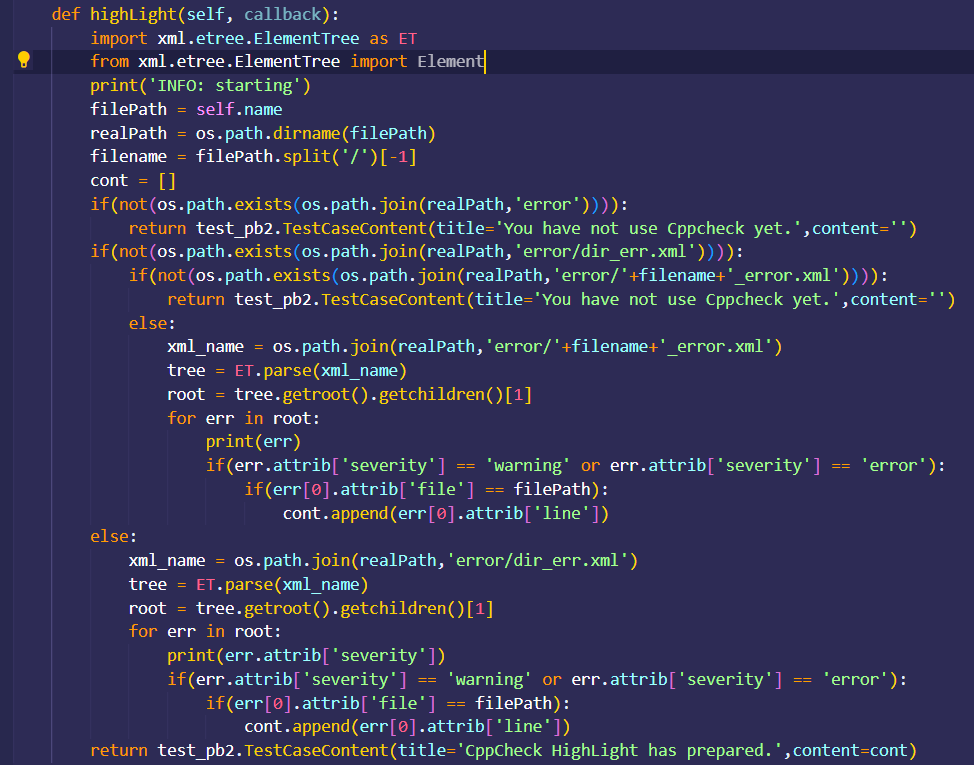
然后等待前端做出相应动作。等前端做出动作后，后端监听到动作的发生，就进行处理。

后端cppcheck函数原理是：将传入的路径按照文件和目录进行区分，分别调用cppcheck的相应命令生成.txt和.xml的报错文件，放入集合error文件夹。然后将txt文件的内容和给用户显示的信息打包传回前端。

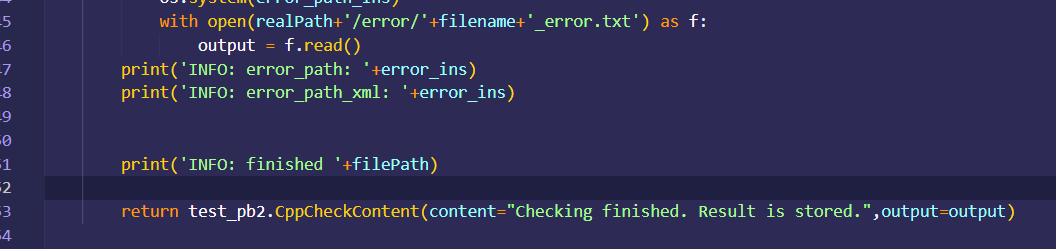


后端highlight函数原理是：找到集合error文件夹，然后分析xml文件。

找到其中的报错信息行数，把它和给用户显示的信息打包传回前端。

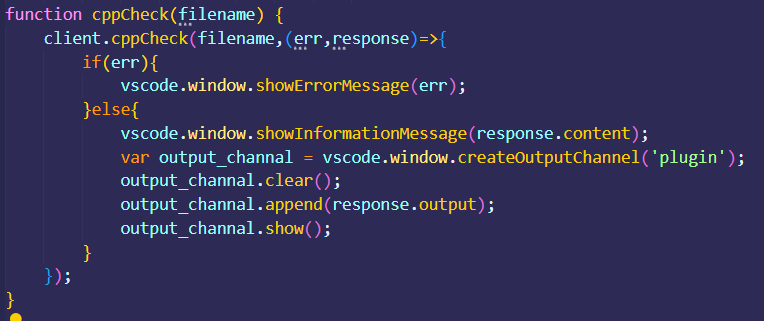


后端处理后将按照test.proto规定的输出格式输出至前端接收。



最后前端作最后的处理。

前端cppcheck函数原理是：先将给用户显示的信息显示出来，然后将返回的.txt报错文件打印到output里面。

  
 前端highlight函数原理是：highlight函数对报错的代码函数实现高亮，方便用户找到报错的代码行数。先创建decorationType,设定背景色等。然后获取报错的行数，将需要高亮的代码范围加入列表。最后通过setDecorations对所选范围进行高亮。



1. xml转换为html进行展示：

9.1 引入相关依赖

查阅网络资料[1]得知，将XML文件转为HTML文件需要引入sys和lxml包中的etree模块，因此，分别引入该两部分即可：



9.2. 确定XML文件路径

首先，获取开发目录的绝对路径，使用proto传到后端的数据的name就是开发目录的绝对路径：

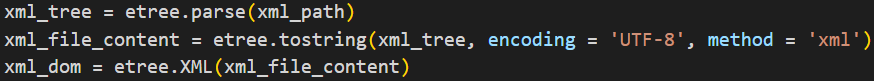


然后，获取生成的XML文件的路径，XML文件存放在开发目录下的/error目录下，所以直接使用字符串拼接即可：



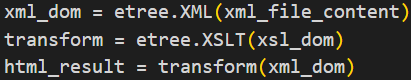
9.3. 生成XML DOM

使用etree的parse()方法，解析该XML文件，生成XML树，再使用etree的tostring()方法将XML树转换为字符串，最后，使用etree的XML()方法将该字符串转换为XML DOM：



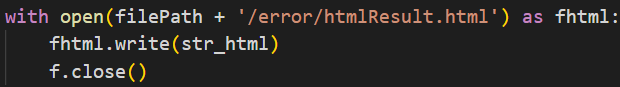
9.4. 生成HTML字符串

首先，使用etree的XSLT()方法，将之前生成的XSL DOM转换为对象transform，使用transform将XML DOM转换为HTML，最后将HTML强制转换为字符串类型：

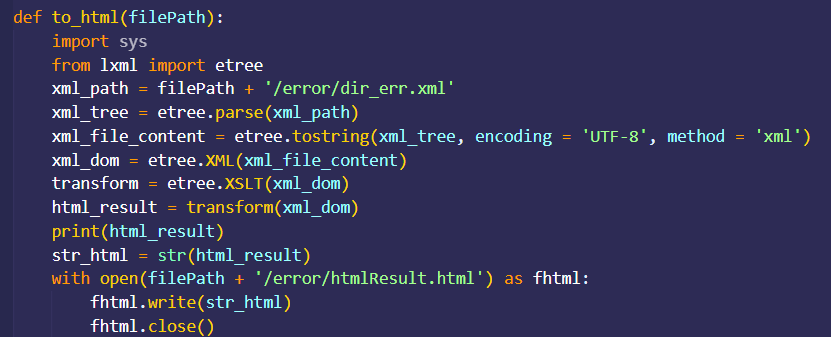


9.5. 生成HTML文件

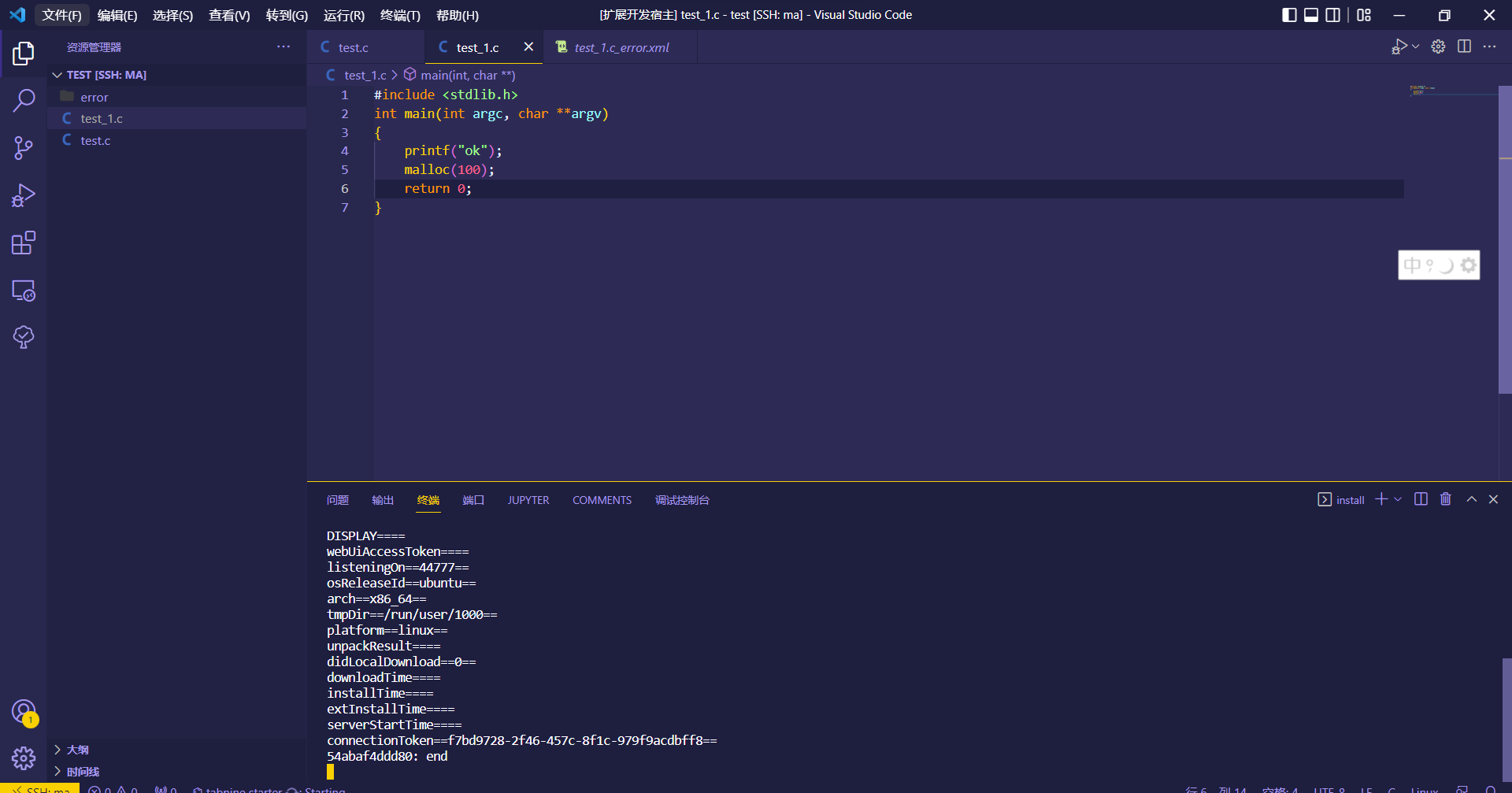
以写的方式打开目标HTML文件，由于不存在该文件，会自动创建该文件，然后将HTML字符串写入HTML文件，即生成了符合要求的HTML文件：



9.6 结果



1. 无误后运行。运行步骤：
   1. 在client文件夹内输入命令python3 cppcheck.py
   2. F5调试



* 1. 执行动作，观察结果。

1. 实验结果：

实验结果在视频内展示。