内存越界错误修复

vscode怎么配置AddressSanitizer？

在 VSCode 中配置 **AddressSanitizer (ASan)**，你需要修改 **C/C++ 任务配置 (tasks.json) 或 CMake 配置**。以下是详细的配置步骤：

方法 1：使用 tasks.json（适用于 `tasks.json` 配置项目）

如果你使用 `tasks.json` 作为编译方式（使用 `g++` 直接编译），请按照以下步骤操作：

1. 打开 VSCode 任务配置

在 VSCode 中按下 `Ctrl + Shift + P`，输入 **Tasks: Configure Tasks**，然后选择 **Create tasks.json file from template**，选择 **Others**。

2. 编辑 `tasks.json`

在 `.vscode/tasks.json` 中，添加如下配置：

```
json

{
    "version": "2.0.0",
    "tasks": [
        {
            "label": "Build with AddressSanitizer",
            "type": "shell",
            "command": "g++",
            "args": [
                "-g",
                "-fsanitize=address",
                "-fno-omit-frame-pointer",
                "-Wall",
                "-Wextra",
                "-o", "main",
                "main.cpp"
            ],
            "group": {
                "kind": "build",
                "isDefault": true
            },
            "problemMatcher": ["$gcc"]
        }
    ]
}
```

解释：

- `-fsanitize=address` → 启用 AddressSanitizer
- `-fno-omit-frame-pointer` → 提供更好的错误堆栈追踪
- `-Wall -Wextra` → 启用额外的警告信息

- `-g` → 启用调试符号，方便调试
- `"main.cpp"` → 你的 C/C++ 源代码文件，请修改成你的文件名

3. 运行任务

按下 `Ctrl + Shift + B` 运行任务，它会编译 `main.cpp`，并生成 `main` 可执行文件。

方法 2：使用 CMake（适用于 CMake 项目）

如果你的项目使用 **CMake**，你可以在 `CMakeLists.txt` 中添加 AddressSanitizer 选项。

1. 修改 `CMakeLists.txt`

在 `CMakeLists.txt` 文件中添加：

```
cmake

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fsanitize=address -fno-omit-frame-pointer -g -Wall -Wextra"
```

然后编译：

```
sh

cmake -B build
cmake --build build
```

这样编译出来的可执行文件就包含 AddressSanitizer 了。

方法 3：使用 launch.json 配置运行

如果你希望在 **VSCode 里直接调试 ASan**，你需要配置 `launch.json`。

1. 打开 VSCode 调试配置

按下 `Ctrl + Shift + D` 进入调试窗口，点击 `create a launch.json file`（创建调试配置）。

2. 编辑 `launch.json`

在 `.vscode/launch.json` 中添加：

```
json

{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Run with AddressSanitizer",
            "type": "cppdbg",
            "request": "launch",
            "program": "${workspaceFolder}/main",
            "args": [],
            "stopAtEntry": false,
            "cwd": "${workspaceFolder}",
```

```
            "environment": [
                { "name": "ASAN_OPTIONS", "value": "detect_leaks=1" }
            ],
            "externalConsole": true,
            "MIMode": "gdb"
        }
    ]
}
```

解释：

- `"ASAN_OPTIONS": "detect_leaks=1"` → 让 AddressSanitizer 检测内存泄漏
- `"program": "${workspaceFolder}/main"` → 你的可执行文件路径
- `"MIMode": "gdb"` → 使用 `gdb` 进行调试

3. 运行程序

1. 在 VSCode 里点击 **运行和调试 (Run and Debug)**。
2. 选择 **Run with AddressSanitizer**。
3. 运行后，如果有内存错误，VSCode 会高亮错误代码位置。

方法 4：终端手动运行

如果你已经编译好了一个带 ASan 的程序，你可以直接运行：

```
sh

ASAN_OPTIONS=detect_leaks=1 ./main
```

这样可以检测 **内存泄漏**，如果有错误，会显示详细的错误报告。

总结

配置方式	适用场景
`tasks.json`	适用于手动编译的 C/C++ 项目
`CMakeLists.txt`	适用于 CMake 项目
`launch.json`	适用于在 VSCode 里 **调试** ASan
终端运行 ASan	适用于已经编译好程序后手动运行

建议：

1. **如果你是手写 `g++` 编译命令，使用 `tasks.json` 方法。**
2. **如果你是用 CMake，修改 `CMakeLists.txt`。**
3. **如果你想在 VSCode 里调试 ASan 错误，配置 `launch.json`。**

这样，你就可以在 VSCode 里高效使用 AddressSanitizer 了！🚀