链滴

vscode linux 远程 go 调试环境

作者：bingoct
原文链接：https://ld246.com/article/1661330811618
来源网站：链滴

不建议用vscode 开发go的大项目。

远程linux环境是wsl2-ubuntu20，原理上基本是一致。

习惯用vscdoe工作区统一配置设置和插件，再写一个同一个通用的task、launch模板，但是每个项（文件夹）的task和lauch单独配置（因为不同项目args、参数、环境要单独设置）。

1. 远程机器　安装go。配置 go env，国内的网络建议配置 GOPROXY="https://goproxy.cn/,direc"，不然后续下载go tools和包很慢。

2. vscode远程连接机器，安装　go 官方插件。安装go 工具（vscode中 ctrl + shift + p 搜索 go install）

3. 工作区设置。

```
"settings": {
        "go.inferGopath": false,    // 用的是go mod（go>1.11），所以没有配置
        "editor.tabSize": 4,
        "editor.insertSpaces": false,
        "go.docsTool": "godoc", // gopls下无影响
        // "go.buildOnSave": "workspace",
        "editor.detectIndentation": false,
        "go.formatTool": "default", // gopls下不影响
        "[go]": {
            "editor.insertSpaces": false,
            "editor.tabSize": 4,
            "editor.formatOnSave": true,
            "editor.codeActionsOnSave": {
                "source.organizeImports": true    // 保存时自动import
            },
            "editor.suggest.snippetsPreventQuickSuggestions": false
        },
        "go.toolsManagement.checkForUpdates": "proxy", // 提示go或go tools的版本更新，我配了goproxy
        "go.useLanguageServer": true,
        "go.languageServerFlags": [
            "-rpc.trace", // for more detailed debug logging
        ],
        "gopls": {
            "ui.completion.usePlaceholders": false,
            // "usePlaceholders": true, // add parameter placeholders when completing a function
            "completeUnimported": true, // autocomplete unimported packages
            "deepCompletion": true, // enable deep completion
        },
        "terminal.explorerKind": "integrated",
    },
```

目前gopls已经稳定了，内存占用差不多300M以内。

```
 procs gopls
 PID:▲ PPID │ User  │ CPU   VmRSS │ TCP │ Command
          │      │ [%]  [bytes] │     │
 4439  4228 │ bingo │ 0.0 210.832M │  [] │ ${GOPATH}/bin/gopls -mode=stdio -rpc.trace
```

4. 准备task和launch模板，参考了　官方文档

- task.json

```
{
    "version": "2.0.0",
    "cwd": "${workspaceFolder}",
    "tasks": [
        {
            // 最终的编译
            "label": "linux : go build",
            "command": "go",
            "args": [
                "build",
                "-o",
                "bin/${workspaceFolderBasename}", // 编译对应的目录
            ],
            "group": {
                "kind": "build",
            }
        },
        {
            // 调试的编译
            "label": "linux : go build(debug)",
            "type": "shell",
            "command": "go",
            "args": [
                "build",
                "-gcflags=all=-N -l", // -N 禁用优化 -l 禁用内联
                "-o",
                "bin/__debug_${workspaceFolderBasename}",
            ],
            "group": "build",
        },
        {
            "label": "linux : go test(debug)",
            "type": "shell",
            "command": "go",
            "args": [
                "test",
                "-c",
                "-o",
                "bin/__test_${workspaceFolderBasename}",
            ],
            "group": "test",
        },
        {
            // 远程终端内调试和相关的后面launch.json中要求一致
            // 如果是root用户可以不用sudo
            "label": "dlv start",
            "type": "shell",
            "command": "sudo",
            "args": [
                "dlv",
                "dap",
                "--listen=127.0.0.1:12345",
```

```
            "--log-dest=3",
            "--only-same-user=false"
        ],
        "group": "build"
    }
  ],
}
```

- launch.json

Launch: fastDebug，使用用默认的调试控制台，缺点是不能输入，不过F5运行是非常的方便。剩下个是当程序要模拟终端输入时，先 ctrl + shift +b运行task中dlv dap服务器。

对于已运行的go程序，可以用attach模式。

```
{
  // 使用 IntelliSense 了解相关属性。
  // 悬停以查看现有属性的描述。
  // 欲了解更多信息，请访问: https://go.microsoft.com/fwlink/?linkid=830387
  // 远程调试前必须先启动dlv
  // sudo dlv dap   --listen=127.0.0.1:12345 --log-dest=3 --only-same-user=false
  "version": "0.2.0",
  "configurations": [
    {
      "name": "goDebug",
      "type": "go",
      "debugAdapter": "dlv-dap",
      "preLaunchTask": "linux : go build(debug)", // 预先调用编译
      "request": "launch",
      "mode": "exec",
      "asRoot": true, // 以root用户执行
      "port": 12345,
      "host": "127.0.0.1",
      "program": "bin/__debug_${workspaceFolderBasename}",
      "cwd": "${workspaceFolder}",
      "env": {
        "GO111MODULE": "auto" // 使用go mod
      },
      "args": [],
    },
    {
      "name": "goTest",
      "type": "go",
      "debugAdapter": "dlv-dap",
      "request": "launch",
      "preLaunchTask": "linux : go test(debug)", // 语调调用编译
      "mode": "exec",
      "asRoot": true,
      "port": 12345,
      "host": "127.0.0.1",
      "program": "bin/__test_${workspaceFolderBasename}",
      "cwd": "${workspaceFolder}",
      "env": {
        "GO111MODULE": "auto"
      },
      "args": [],
```

```
        },
        {
            "name": "fastDebug",
            "type": "go",
            "debugAdapter": "dlv-dap",
            "request": "launch",
            "mode": "debug",
            "asRoot": true, // 以root用户执行
            "program": "${workspaceFolder}",
            "cwd": "${workspaceFolder}",
            "env": {

                "GO111MODULE": "auto"
            },
            "args": [],
        }
    ]
}
```

5. 在go工作区中添加项目（文件夹），将task和launch模板拷贝到项目下的 .vscode目录下，然就是用go mod init初始化项目了。（go mod 相关的操作其实也可以考虑配置成task）