02-环境准备：如何安装和配置一个基本的Go开发环境？

你好，我是孔令飞。

上一讲我们讲了 IAM 系统的功能和架构，接下来的两讲，我们就将它部署到你的服务器上。不过，在正式部署之前，我们还需要准备一个Go 开发环境，这是因为我们是通过编译源码来获取部署需要的二进制文件的。

因此，今天这一讲，我先手把手带你配置好一个 Go 的开发环境，供你以后开发、编译用，下一讲再带你部署IAM系统。

想要配置一个 Go 开发环境，我们可以通过以下 4 步实现：

1. Linux 服务器申请和配置
2. 依赖安装和配置
3. Go 编译环境安装和配置
4. Go 开发 IDE 安装和配置

Linux 服务器申请和配置

毫无疑问，要安装一个 Go 开发环境，你首先需要有一个 Linux 服务器。Linux 服务器有很多操作系统可供选择，例如：CentOS、Ubuntu、RHEL、Debian 等，但目前生产环境用得最多的还是 CentOS 系统，为了跟生产环境保持一致，我们选择当前最新的 CentOS 版本：CentOS 8.2。

因为本专栏的所有操作都是在 CentOS 8.2 系统上进行的，为了避免环境不一致导致的操作失败，我建议你也使用 CentOS 8.2。安装一个 Linux 服务器需要两步：服务器申请和配置。

Linux 服务器申请

我们可以通过以下 3 种方式来安装一个 CentOS 8.2 系统。

1. 在物理机上安装一个 CentOS 8.2 系统。
2. 在 Windows/MacBook 上安装虚拟机管理软件，用虚拟机管理软件创建 CentOS 8.2 虚拟机。其中，Windows 建议用 VMWare Workstation 来创建虚拟机，MacBook 建议用 VirtualBox 来创建虚拟机。
3. 在诸如腾讯云、阿里云、华为云等平台上购买一个虚拟机，并预装 CentOS 8.2 系统。

Linux 服务器配置

申请完 Linux 服务之后，我们需要通过 SecureCRT 或 Xshell 等工具登录 Linux 服务器，并对服务器做一些简单必要的配置，包括创建普通用户、添加 sudoers、配置 `$HOME/.bashrc` 文件。接下来，我们一一来说。

第一步，用Root 用户登录Linux 系统，并创建普通用户。

一般来说，一个项目会由多个开发人员协作完成，为了节省企业成本，公司不会给每个开发人员都配备一台服务器，而是让所有开发人员共用一个开发机，通过普通用户登录开发机进行开发。因此，为了模拟真实的企业开发环境，我们也通过一个普通用户的身份来进行项目的开发，创建方法如下：

```
# useradd going # 创建 going 用户，通过 going 用户登录开发机进行开发
# passwd going # 设置密码
Changing password for user going.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

不仅如此，使用普通用户登录和操作开发机也可以保证系统的安全性，这是一个比较好的习惯，所以我们在日常开发中也要尽量避免使用 Root 用户。

第二步，添加 sudoers。

我们知道很多时候，普通用户也要用到 Root 的一些权限，但 Root 用户的密码一般是由系统管理员维护并定期更改的，每次都向管理员询问密码又很麻烦。因此，我建议你将普通用户加入到 sudoers 中，这样普通用户就可以通过 sudo 命令来暂时获取 Root 的权限。具体来说，你可以执行如下命令添加：

```
# sed -i '/^root.*ALL=(ALL).*ALL/a\going\tALL=(ALL) \tALL' /etc/sudoers
```

第三步，用新的用户名（going）和密码登录Linux 服务器。这一步也可以验证普通用户是否创建成功。

第四步，配置 $HOME/.bashrc 文件。

我们登录新服务器后的第一步就是配置 $HOME/.bashrc 文件，以使 Linux 登录 shell 更加易用，例如配置 LANG 解决中文乱码，配置 PS1 可以避免整行都是文件路径，并将 `$HOME/bin` 加入到 PATH 路径中。配置后的内容如下：

```
# .bashrc

# User specific aliases and functions

alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# User specific environment
# Basic envs
export LANG="en_US.UTF-8" # 设置系统语言为 en_US.UTF-8，避免终端出现中文乱码
export PS1='[\u@dev \W]\$ ' # 默认的 PS1 设置会展示全部的路径，为了防止过长，这里只展示："用户名@dev 最后的目录名"
export WORKSPACE="$HOME/workspace" # 设置工作目录
export PATH=$HOME/bin:$PATH # 将 $HOME/bin 目录加入到 PATH 变量中

# Default entry folder
cd $WORKSPACE # 登录系统，默认进入 workspace 目录
```

有一点需要我们注意，在 export PATH 时，最好把 $PATH 放到最后，因为我们添加到目录中的命令是期望被优先搜索并使用的。配置完 $HOME/.bashrc 后，我们还需要创建工作目录 workspace。将工作文件统一放在 $HOME/workspace 目录中，有几点好处。

- 可以使我们的$HOME目录保持整洁，便于以后的文件查找和分类。
- 如果哪一天 /分区空间不足，可以将整个 workspace 目录 mv 到另一个分区中，并在 /分区中保留软连接，例如：/home/going/workspace -> /data/workspace/。
- 如果哪天想备份所有的工作文件，可以直接备份 workspace。

具体的操作指令是$ mkdir -p $HOME/workspace。配置好 $HOME/.bashrc 文件后，我们就可以执行 bash 命令将配置加载到当前 shell 中了。

至此，我们就完成了 Linux 开发机环境的申请及初步配置。

依赖安装和配置

在 Linux 系统上安装 IAM 系统会依赖一些 RPM 包和工具，有些是直接依赖，有些是间接依赖。为了避免后续的操作出现依赖错误，例如，因为包不存在而导致的编译、命令执行错误等，我们先统一依赖安装和配置。安装和配置步骤如下。

第一步，安装依赖。

首先，我们在 CentOS 系统上通过 yum 命令来安装所需工具的依赖，安装命令如下：

```
$ sudo yum -y install make autoconf automake cmake perl-CPAN libcurl-devel libtool gcc gcc-c++ glibc-header
```

虽然有些 CentOS 8.2 系统已经默认安装这些依赖了，但是为了确保它们都能被安装，我仍然会尝试安装一遍。如果系统提示 Package xxx is already installed.，说明已经安装好了，你直接忽略即可。

第二步，安装 Git。

因为安装 IAM 系统、执行 go get 命令、安装 protobuf 工具等都是通过 Git 来操作的，所以接下来我们还需要安装 Git。由于低版本的 Git 不支持--unshallow 参数，而 go get 在安装 Go 包时会用到 git fetch --unshallow 命令，因此我们要确保安装一个高版本的 Git，具体的安装方法如下：

```
$ cd /tmp
$ wget https://mirrors.edge.kernel.org/pub/software/scm/git/git-2.30.2.tar.gz
$ tar -xvzf git-2.30.2.tar.gz
$ cd git-2.30.2/
$ ./configure
$ make
$ sudo make install
```

```
$ git --version          # 输出 git 版本号，说明安装成功
git version 2.30.2
```

注意啦，按照上面的步骤安装好之后，我们要把 Git 的二进制目录添加到 PATH 路径中，不然 Git 可能会因为找不到一些命令而报错。你可以通过执行以下命令添加目录：

```
tee -a $HOME/.bashrc <<'EOF'
# Configure for git
export PATH=/usr/local/libexec/git-core:$PATH
EOF
```

第三步，配置 Git。我们直接执行如下命令配置 Git：

```
$ git config --global user.name "Lingfei Kong"    # 用户名改成自己的
$ git config --global user.email "colin404@foxmail.com"    # 邮箱改成自己的
$ git config --global credential.helper store    # 设置 Git，保存用户名和密码
$ git config --global core.longpaths true # 解决 Git 中 'Filename too long' 的错误
```

除了按照上述步骤配置 Git 之外，我们还有几点需要注意。

首先，在 Git 中，我们会把非 ASCII 字符叫做 Unusual 字符。这类字符在 Git 输出到终端的时候默认是用 8 进制转义字符输出的（以防乱码），但现在的终端多数都支持直接显示非 ASCII 字符，所以我们可以关闭掉这个特性，具体的命令如下：

```
$ git config --global core.quotepath off
```

其次，如果你觉得访问 github.com 太慢，可以通过国内 GitHub 镜像网站来访问，配置方法如下：

```
$ git config --global url."https://github.com.cnpmjs.org/".insteadOf "https://github.com/"
```

这里你要注意，通过镜像网站访问仅对 HTTPS 协议生效，对 SSH 协议不生效，并且 github.com.cnpmjs.org 的同步时间间隔为 1 天。

最后，GitHub 限制最大只能克隆 100M 的仓库，为了能够克隆容量大于 100M 的仓库，我们还需要安装 Git Large File Storage，安装方式如下：

```
$ git lfs install --skip-repo
```

好啦，现在我们就完成了依赖的安装和配置。

Go 编译环境安装和配置

我们知道，Go 是一门编译型语言，所以在部署 IAM 系统之前，我们需要将代码编译成可执行的二进制文件。因此我们需要安装 Go 编译环境。

除了 Go，我们也会用 gRPC 框架展示 RPC 通信协议的用法，所以我们也需要将 ProtoBuf 的.proto 文件编译成 Go 语言的接口。因此，我们也需要安装 ProtoBuf 的编译环境。

Go 编译环境安装和配置

安装 Go 语言相对来说比较简单，我们只需要下载源码包、设置相应的环境变量即可。

首先，我们从 Go 语言官方网站下载对应的 Go 安装包以及源码包，这里我下载的是 go1.16.2 版本：

```
$ wget https://golang.org/dl/go1.16.2.linux-amd64.tar.gz -O /tmp/go1.16.2.linux-amd64.tar.gz
```

在下载的时候，我们要选择：linux-amd64 的格式。如果因为被墙的原因访问不了 golang.org，你也可以执行下面的命令下载：

```
$ wget https://marmotedu-1254073058.cos.ap-beijing.myqcloud.com/tools/go1.16.2.linux-amd64.tar.gz -O /tmp/g
```

接着，我们完成解压和安装，命令如下：

```
$ mkdir -p $HOME/go
$ tar -xvzf /tmp/go1.16.2.linux-amd64.tar.gz -C $HOME/go
$ mv $HOME/go/go $HOME/go/go1.16.2
```

最后，我们执行以下命令，将下列环境变量追加到`$HOME/.bashrc`文件中。

```
tee -a $HOME/.bashrc <<'EOF'
# Go envs
export GOVERSION=go1.16.2 # Go 版本设置
export GO_INSTALL_DIR=$HOME/go # Go 安装目录
```

```
export GOROOT=$GO_INSTALL_DIR/$GOVERSION # GOROOT 设置
export GOPATH=$WORKSPACE/golang # GOPATH 设置
export PATH=$GOROOT/bin:$GOPATH/bin:$PATH # 将 Go 语言自带的和通过 go install 安装的二进制文件加入到 PATH 路径中
export GO111MODULE="on" # 开启 Go moudles 特性
export GOPROXY=https://goproxy.cn,direct # 安装 Go 模块时，代理服务器设置
export GOPRIVATE=github.com # 指定不走代理的 Go 包域名
export GOSUMDB=off # 关闭校验 Go 依赖包的哈希值
EOF
```

为什么要增加这么多环境变量呢？这是因为，Go 语言就是通过一系列的环境变量来控制 Go 编译器行为的。因此，我们一定要理解每一个环境变量的含义。

环境变量	含义
GOROOT	Go 语言编译工具、标准库等的安装路径
GOPATH	Go 的工作目录，也就是编译后二进制文件的存放目录和 import 包时的搜索路径
GO111MODULE	通过设置 on、off、auto 来控制是否开启 Go Modules 特性。其中，**on** 代表开启 Go modules 特性，这会让 Go 编译器忽略$GOPATH 和 vendor 文件夹，只根据 go.mod 下载依赖。**off** 代表关闭 Go modules 特性，这会让 Go 编译器在$GOPATH 目录和 vendor 目录来查找依赖关系，也就是继续使用“GOPATH 模式”。而 **auto** 在 Go1.14 和之后的版本中是默认值。当设置为 auto 后，源码在$GOPATH/src 下，并且没有包含 go.mod 则关闭 Go modules，其它情况下都开启 Go modules
GOPROXY	Go 包下载代理服务器。众所周知的原因，在大陆的网络环境下是无法访问 golang.org 等 Google 网站的，但在日常开发中使用的很多依赖包都要从 Google 的服务器上下载。为了解决无法加载依赖的问题，需要设置一个代理服务器。以便我们能够使用 go get 下载 direct 作用：当 go 在抓取目标模块时，若遇见了 404 错误，那么就直接去目标模块的源头（比如 GitHub） 去抓取，而不再通过代理服务器
GOPRIVATE	指定不走代理的 Go 包域名。go get 通过代理服务拉取私有仓库（内部仓库或者托管站点的私有仓库），而代理服务不可能访问到私有仓库，会出现了 404 错误。go1.13 版本提供了一个方便的解决方案：GOPRIVATE 环境变量，通过该变量，可以使得指定的包不通过代理下载，而是直接下载
GOSUMDB	GOSUMDB 的值是一个 web 服务器，默认值是 sum.golang.org， 该服务可以用来查询依赖包指定版本的哈希值，保证拉取到的模块版本数据未经过篡改

因为 Go 以后会用 Go modules 来管理依赖，所以我建议你将 GO111MODULE 设置为 on。

在使用模块的时候，$GOPATH 是无意义的，不过它还是会把下载的依赖储存在 $GOPATH/pkg/mod 目录中，也会把 go install 的二进制文件存放在 $GOPATH/bin 目录中。

另外，我们还要将$GOPATH/bin、$GOROOT/bin 加入到 Linux 可执行文件搜索路径中。这样一来，我们就可以直接在 bash shell 中执行 go 自带的命令，以及通过 go install 安装的命令。

最后就是进行测试了，如果我们执行 go version 命令可以成功输出 Go 的版本，就说明 Go 编译环境安装成功。具体的命令如下：

```
$ bash
$ go version
go version go1.16.2 linu x/amd64
```

ProtoBuf 编译环境安装

接着，我们再来安装 protobuf 的编译器 protoc。protoc 需要 protoc-gen-go 来完成 Go 语言的代码转换，因此我们需要安装 protoc 和 protoc-gen-go 这 2 个工具。它们的安装方法比较简单，你直接看我下面给出的代码和操作注释就可以了。

```
# 第一步：安装 protobuf
$ cd /tmp/
$ git clone --depth=1 https://github.com/protocolbuffers/protobuf
$ cd protobuf
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
$ protoc --version # 查看 protoc 版本，成功输出版本号，说明安装成功
libprotoc 3.15.6

# 第二步：安装 protoc-gen-go
$ go get -u github.com/golang/protobuf/protoc-gen-go
```

当你第一次执行 go get 命令的时候，因为本地无缓存，所以需要下载所有的依赖模块。因此安装速度会比较慢，请你耐心等待。

Go 开发 IDE 安装和配置

编译环境准备完之后，我们还需要一个代码编辑器才能开始 Go 项目开发，并且为了提高开发效率，我们需要将这个编辑器配置成 Go IDE。

目前，GoLand、VSCode 这些 IDE 都很优秀，我们使用的也很多，但它们都是 Windows 系统下的 IDE。因此，在 Linux 环境下我们可以选择将 Vim 配置成 Go IDE，熟悉 Vim IDE 的操作之后，它的开发效率不输 GoLand 和 VSCode。

比如说，我们可以通过 SpaceVim 将 Vim 配置成一个 Go IDE。SpaceVim 是一个社区驱动的模块化的 Vim IDE，以模块的方式组织管理插件以及相关配置， 为不同的语言开发量身定制了相关的开发模块，该模块提供代码自动补全、 语法检查、格式化、调试、REPL 等特性。我们只需要载入相关语言的模块就能得到一个开箱即用的 Vim IDE 了。

Vim 可以选择 NeoVim，NeoVim 是基于 Vim 的一个 fork 分支，它主要解决了 Vim8 之前版本中的异步执行、开发模式等问题，对 Vim 的兼容性很好。同时对 vim 的代码进行了大量地清理和重构，去掉了对老旧系统的支持，添加了新的特性。

虽然 Vim8 后来也新增了异步执行等特性，在使用层面两者差异不大，但是 NeoVim 开发更激进，新特性更多，架构也相对更合理，所以我选择了 NeoVim，你也可以根据个人爱好来选择（都是很优秀的编辑器，这里不做优缺点比较）。Vim IDE 的安装和配置主要分五步。

第一步，安装 NeoVim。我们直接执行 pip3 和 yum 命令安装即可，安装方法如下：

```
$ sudo pip3 install pynvim
```

```
$ sudo yum -y install neovim
```

第二步，配置 `$HOME/.bashrc`。先配置 nvim 的别名为 vi，这样当我们执行 vi 时，Linux系统就会默认调用 nvim。同时，配置 EDITOR 环境变量可以使一些工具，例如 Git 默认使用 nvim。配置方法如下：

```
tee -a $HOME/.bashrc <<'EOF'
# Configure for nvim
export EDITOR=nvim # 默认的编辑器（git 会用到）
alias vi="nvim"
EOF
```

第三步，检查 nvim 是否安装成功。我们可以通过查看 NeoVim 版本来确认是否成功安装，如果成功输出版本号，说明 NeoVim 安装成功。

```
$ bash
$ vi --version # 输出 NVIM v0.3.8 说明安装成功
NVIM v0.3.8
Build type: RelWithDebInfo
...
```

第四步，离线安装 SpaceVim。安装 SpaceVim 步骤稍微有点复杂，为了简化你的安装，同时消除网络的影响，我将安装和配置 SpaceVim 的步骤做成了一个离线安装包 marmotVim 。marmotVim 可以进行 SpaceVim 的安装、卸载、打包等操作，安装步骤如下：

```
$ cd /tmp
$ wget https://marmotedu-1254073058.cos.ap-beijing.myqcloud.com/tools/marmotVim.tar.gz
$ tar -xvzf marmotVim.tar.gz
$ cd marmotVim
$ ./marmotVimCtl install
```

SpaceVim 配置文件为：`$HOME/.SpaceVim.d/init.toml` 和 `$HOME/.SpaceVim.d/autoload/custom_init.vim`，你可自行配置（配置文件中有配置说明）：

- init.toml：SpaceVim 的配置文件
- custom_init.vim：兼容 vimrc，用户自定义的配置文件

SpaceVim Go IDE 常用操作的按键映射如下表所示：

按键	功能描述
F2	显示函数、变量、结构体等列表
F3	显示当前目录下文件列表
gd/ctrl +]/<Enter>	:GoDef，跳转到光标所在标识符的声明或者定义的位置
Ctrl+I	:GoDefPop，跳转到跳转堆栈的上一个位置
Ctrl+O	回上一次位置
Shift+K	:GoDoc，在新 Vim 窗口中显示光标处 word 或者给定 word 的 Go 文档
Shift+L	:GolfErr，生成 if err != nil { return ... }示例代码
Shift+T	:GoDefType，跳转到光标所在标识符的类型定义的位置
Shift+M	:GoInfo，显示光标所在的标识符的信息， 比如显示函数的声明信息，变量的数据类型
Crl + N	自动补全时下一个补全项
Ctrl + P	自动补全时上一个补全项

第五步，Go 工具安装。

SpaceVim 会用到一些 Go 工具，比如在函数跳转时会用到 guru、godef 工具，在格式化时会用到 goimports，所以我们也需要安装这些工具。安装方法有 2 种：

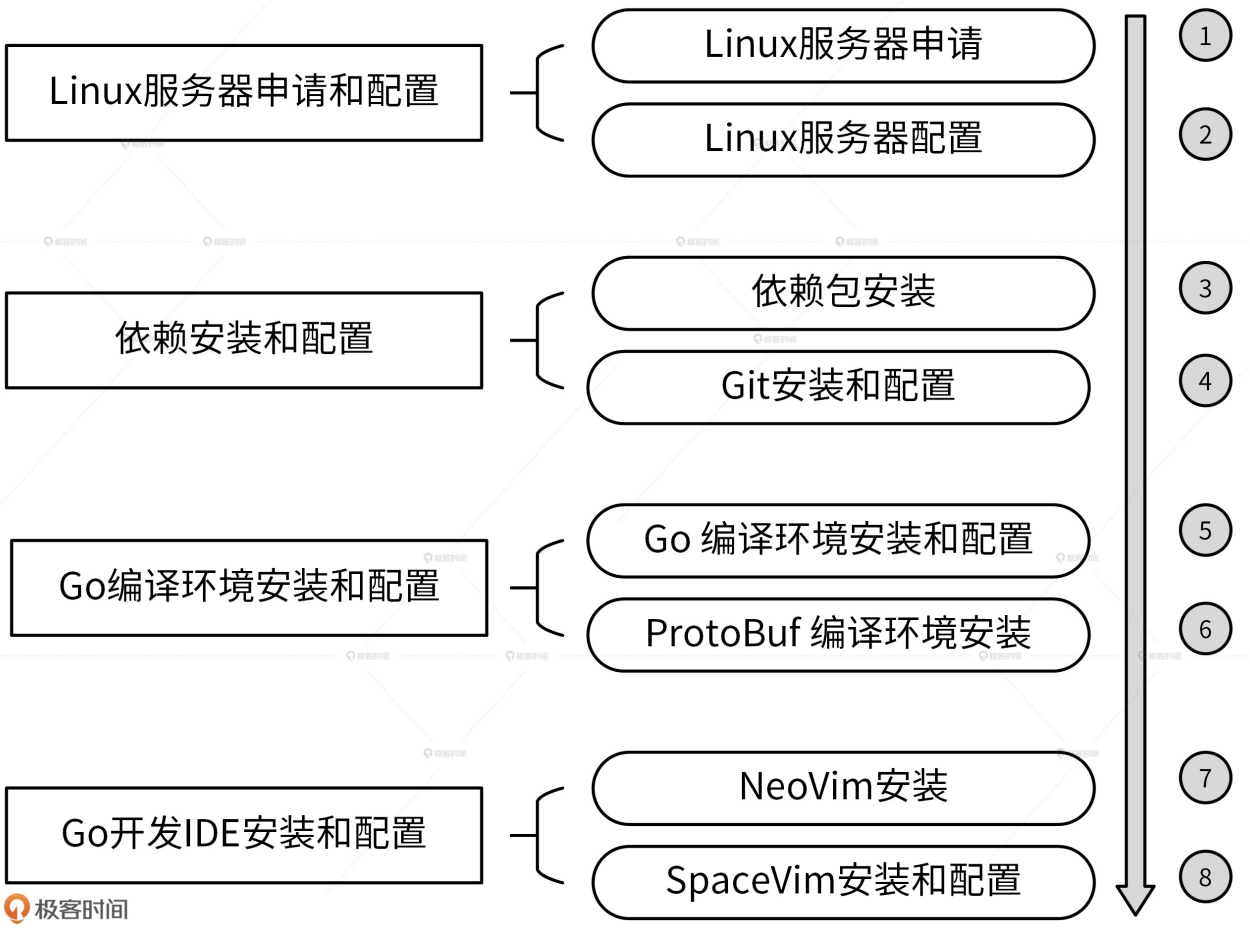
1. Vim 底线命令安装：vi test.go，然后执行：`:GoInstallBinaries` 安装。
2. 拷贝工具：直接将整理好的工具文件拷贝到`$GOPATH/bin` 目录下。

为了方便，你可以直接拷贝我已经打包好的 Go 工具到指定目录下：

```
$ cd /tmp
$ wget https://marmotedu-1254073058.cos.ap-beijing.myqcloud.com/tools/gotools-for-spacevim.tgz
$ mkdir -p $GOPATH/bin
$ tar -xvzf gotools-for-spacevim.tgz -C $GOPATH/bin
```

总结

这一讲，我们一起安装和配置了一个 Go 开发环境，为了方便你回顾，我将安装和配置过程绘制成了一个流程图，如下所示。



有了这个开发环境，接下来我们就可以在学习的过程中随时进行编码，来熟悉和验证知识点了，所以你一定要在学习后面的课程之前先完成这一讲的部署。

课后练习

1.你起试着编写一个 main.go，在 main 函数中打印 `Hello World`，并执行 `go run main.go` 运行代码，测试 Go 开发环境。
2. 你请试着编写一个 main.go，代码如下：

```
package main

import "fmt"

func main() {
    fmt.Println("Hello World")
}
```

将鼠标放在 **Println** 上，键入 **Enter** 键跳转到函数定义处，键入 **Ctrl + I** 返回到跳转点。

期待在留言区看到你的思考和答案，也欢迎和我一起探讨环境安装过程中的问题，我们下一讲见！

精选留言：

- pedro 2021-05-26 22:29:59
  已经在腾讯云学生服务器上全部搞定了，但是文中应该有两个问题。
  一、$HOME/.SpaceVim.d/init.toml 应该是 init.vim 文件。

二、wget https://marmotedu-1254073058.cos.ap-beijing.myqcloud .com/tools/gotools-for-spacevim.tgz$ 这个命令稍微有点问题，中间多了一个空格。

另外，由于我的主机版本是 centos7.5，所以默认没有 python3 环境，如果大家跟我一样，可先通过 yum 来安装 python3，然后在安装 neovim，不过由于 centos7 自带 python2，所以进入 nvim 的时候还是会报错，所以吧，还是建议将 python2 的 neovim 也装上，如下：
```sh
yum install -y python-pip
sudo pip install neovim
``` [3赞]

作者回复2021-05-27 19:52:13
看专栏是没有空格的哈

- Q 2021-05-27 11:03:51
#我是本地安装虚拟机部署CentOS8
#编译git过程中出错的话，需要先安装这两个依赖
sudo yum install expat-devel
sudo yum -y install openssl-devel [1赞]

作者回复2021-05-27 12:36:50
感谢Q哥补充

- Adrian 2021-05-28 10:26:54
Centos8都废弃了呀，为啥不选择centos7或者ubuntu系列呢

作者回复2021-05-29 10:00:51
用新不用久哈，大趋势是高版本

- 柒城 2021-05-28 08:39:48
这里没有说对虚拟机的资源要求？具体需要多少cpu。多少内存？

作者回复2021-05-29 10:01:25
cpu这里是不区分虚拟机和物理机的。前提是虚拟机的cpu不能是超卖

- Geek_sky 2021-05-28 02:00:41
老师，我在光标定位使用enter跳转之后，服务器的负载会达到15到20，这个是正常的吗？我选中的服务器配置是1H1G的

作者回复2021-05-29 10:01:49
刚开始，gopls会解析go的标签，计算量大，正常

- m404 2021-05-27 23:07:00
能否问下，现在centos8.2已经无法下载镜像了，是用8.3还是8.0才能更好的跟随课程呢？

作者回复2021-05-28 08:24:58
8.3吧，系统版本号都是向前兼容的。
另外8.x中不同x主要是用来做一些功能优化和bug修复，不会有大变动，所以只要是8.x就没啥问题

- pedro 2021-05-27 09:33:38
怕坑了大家，现在赶紧过来说明一下，对于服务器还是要选择和作者一样的 centos8，后面的 IAM 项目里

面安装的依赖全是 centos8 的依赖，大家切记，血的教训！！！

作者回复2021-05-27 12:36:14
是的，建议还是直接centos8操作。也可以在centos7上部署开发，前提是你理解专栏，知道如何去做适配。

- Daiver 2021-05-26 21:31:52
还可以用vagrant 快速创建centos8

作者回复2021-05-26 23:40:07
老哥，优秀!

- 梭罗的小屋 2021-05-26 21:18:22
安装NeoVim的第一步中使用pip3安装pynvim这个Python的nvim client，是后续课程会用Python替代vimL写nvim插件吗?
网上找到一篇介绍pynvmo用法的文章 https://blog.csdn.net/weixin_28853079/article/details/112666776

作者回复2021-05-26 23:39:49
不会哈，pynvim只是neovim的一个依赖。这里可以不用太多关注

- Leven 2021-05-26 21:14:58
老师，有没有mac环境搭建的教程推荐，远程Linux的效果不稳定

作者回复2021-05-26 23:38:05
Mac教程还没有，Go是跨平台的语言，你可以尝试用mac部署下，会有些地方需要做适配，不过应该不多。

后面还有不少课程，都是基于Linux，还是建议你用Linux，这样问题会少很多，学习效率会好很多。使用Linux有什么问题欢迎留言。