

## 2.6.2 实验 2：把 vim 打成一个比 source insight 强大好用的 IDE 编辑工具

### 2.6.2.1 实验目的

通过配置把 vim 打成一个和 source insight 媲美的 IDE 工具

### 2.6.2.2 实验步骤

vim 工具可以支持很多的个性化和使用的插件来完成代码浏览和编辑的功能。使用过 source insight 的同学一定对如下功能赞叹有加：

- 自动列出一个文件的函数和变量的列表
- 查找函数和变量的定义
- **Loopup Reference**：查找哪些函数调用了该函数和变量
- 高亮显示
- 自动补全

这些功能在 vim 里都可以实现，而且比 source insight 还高效好用，本实验就带领大家来 DIY 打造一个属于自己的 IDE 编辑工具。

在打造之前先安装 git 工具。

```
sudo apt-get install git
```

#### 1. 按照插件管理工具 Vundle<sup>1</sup>

vim 支持很多插件，在老早的时候都需要到每个插件网站上下载然后拷贝到 home 主目录的 .vim 目录中。现在 vim 社区有好几个插件管理工具，其中 vundle 就是很出色的一个，它可以在 .vimrc 中跟踪和管理插件，自动更新插件等。

安装 vundle 需要使用 git 工具。

```
git clone https://github.com/VundleVim/Vundle.vim.git
~/.vim/bundle/Vundle.vim
```

接下来需要在 home 主目录下面的 .vimrc 配置文件中配置 vundle 了。

```
" Vundle manage
set nocompatible          " be iMproved, required
filetype off              " required

" set the runtime path to include Vundle and initialize
set rtp+=~/.vim/bundle/Vundle.vim
call vundle#begin()

" let Vundle manage Vundle, required
Plugin 'VundleVim/Vundle.vim'

" All of your Plugins must be added before the following line
call vundle#end()          " required
filetype plugin indent on  " required
```

只需要在该配置文件中添加 “Plugin xxx” 即可。

<sup>1</sup> <https://github.com/VundleVim/Vundle.vim>

# 奔跑吧linux内核教学视频

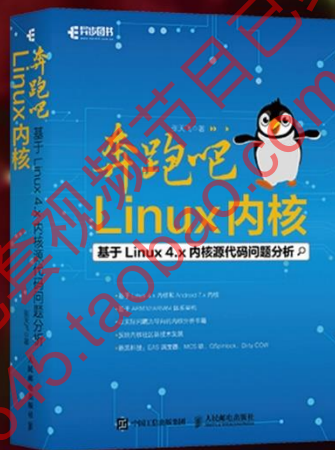
旗舰篇一次订阅，持续更新

配套视频 **旗舰篇**



官方淘宝店

第**1**季  
内存管理



微信公众号

## 课程优势

1. 最有深度和广度的 Linux 内核视频
2. 手把手解读 Linux 内核代码
3. 紧跟 Linux 内核社区技术热点
4. 一键订阅，持续更新
5. 图书 + 视频，全新学习模式
6. 笨叔叔的 VIP 私密微信群答疑

# 第一季内存管理旗舰篇课程目录

课程名称	时长	课程名称	时长
序言一：Linux内核学习方法论	0:09:13	奔跑2.4.2 Linux内核中的伙伴系统和碎片化	0:11:14
序言二：学习前准备		奔跑2.4.3 Linux的页面分配器	0:21:37
序言2.1 Linux发行版和开发板的选择	0:13:56	2.5 slab分配器	
序言2.2 搭建Qemu+gdb单步调试内核	0:13:51	奔跑2.5.1 slab原理和核心数据结构	0:18:36
序言2.3 搭建Eclipse图形化调试内核	0:10:59	奔跑2.5.2 Linux内核中slab机制的实现	0:16:56
实战运维1：查看系统内存信息的工具（一）	0:20:19	2.6 vmalloc分配	
实战运维2：查看系统内存信息的工具（二）	0:16:32	奔跑2.6 vmalloc分配	0:15:48
实战运维3：读懂内核log中的内存管理信息	0:25:35	2.7 VMA操作	
实战运维4：读懂 proc meminfo	0:27:59	奔跑2.7 VMA操作	0:16:42
实战运维5：Linux运维能力进阶路线图	0:09:40	2.8 malloc分配器	
实战运维6：Linux内存管理参数调优（一）	0:19:46	奔跑2.8.1 malloc的三个迷惑	0:17:41
实战运维7：Linux内存管理参数调优（二）	0:31:20	奔跑2.8.2 内存管理的三个重要的函数	0:17:38
实战运维8：Linux内存管理参数调优（三）	0:22:58	2.9 mmap分析	
运维高级如何单步调试RHEL—CENTOS7的内核一	0:15:45	奔跑2.9 mmap分析	0:23:14
运维高级如何单步调试RHEL—CENTOS7的内核二	0:41:28	2.10 缺页中断处理	
vim:打造比source insight更强更好用的IDE（一）	0:24:58	奔跑2.10.1 缺页中断一	0:31:07
vim:打造比source insight更强更好用的IDE（二）	0:20:28	奔跑2.10.2 缺页中断二	0:16:58
vim:打造比source insight更强更好用的IDE（三）	0:23:25	2.11 page数据结构	
实战git项目和社区patch管理		奔跑2.11 page数据结构	0:29:41
2.0 Linux内存管理背景知识介绍		2.12 反向映射机制	
奔跑2.0.0 内存管理硬件知识	0:15:25	奔跑2.12.1 反向映射机制的背景介绍	0:19:01
奔跑2.0.1 内存管理总览一	0:23:27	奔跑2.12.2 RMAP四部曲	0:07:31
奔跑2.0.2 内存管理总览二	0:07:35	奔跑2.12.3 手撕Linux2.6.11上的反向映射机制	0:07:35
奔跑2.0.3 内存管理常用术语	0:09:49	奔跑2.12.4 手撕Linux4.x上的反向映射机制	0:10:08
奔跑2.0.4 内存管理究竟管些什么东西	0:28:02	2.13 回收页面	
奔跑2.0.5 内存管理代码框架导读	0:38:09	奔跑2.13.1 页面回收一	0:16:07
2.1 Linux内存初始化		奔跑2.13.2 页面回收二	0:11:41
奔跑2.1.0 DDR简介	0:06:47	2.14 匿名页面的生命周期	制作中会更新
奔跑2.1.1 物理内存三大数据结构	0:19:39	2.15 页面迁移	制作中会更新
奔跑2.1.2 物理内存初始化	0:11:13	2.16 内存规整	制作中会更新
奔跑2.1 内存初始化之代码导读一	0:43:54	2.17 KSM	制作中会更新
奔跑2.1 内存初始化之代码导读二	0:23:31	2.18 Dirty COW内存漏洞	制作中会更新
奔跑2.1 代码导读C语言部分（一）	0:27:34	2.19 内存数据结构和API总结	制作中会更新
奔跑2.1 代码导读C语言部分（二）	0:21:28	2.20 Meltdown漏洞分析	
2.2 页表的映射过程		奔跑2.20.1 Meltdown背景知识	0:10:13
奔跑2.2.0 ARM32页表的映射	0:08:54	奔跑2.20.2 CPU体系结构之指令执行	0:11:25
奔跑2.2.1 ARM64页表的映射	0:10:58	奔跑2.20.3 CPU体系结构之乱序执行	0:11:03
奔跑2.2.2 页表映射例子分析	0:11:59	奔跑2.20.4 CPU体系结构之异常处理	0:03:48
奔跑2.2.3 ARM32页表映射那些奇怪的事	0:09:42	奔跑2.20.5 CPU体系结构之cache	0:10:56
2.3 内存布局图		奔跑2.20.6 进程地址空间和页表及TLB	0:17:39
奔跑2.3.1 内存布局一	0:10:35	奔跑2.20.7 Meltdown漏洞分析	0:06:04
奔跑2.3.2 内存布局二	0:13:30	奔跑2.20.8 Meltdown漏洞分析之x86篇	0:12:07
2.4 分配物理页面		奔跑2.20.9 ARM64上的KPTI解决方案	0:25:39
奔跑2.4.1 伙伴系统原理	0:10:10	2.21 spectre漏洞分析	制作中会更新
		2.22 异构内存管理	制作中会更新
		2.23 Hugepage巨页	制作中会更新
		2.24 运维人员必会的内存调优	制作中会更新
		2.25 实战内存泄漏	制作中会更新
		2.26 从内存管理代码中学会的优化技巧	制作中会更新

后期课程不定期更新中，等您来提交topic

截至 18 年 5 月已录制完成约 20 小时，后续精彩视频不断

规划中

第二季 虚拟化

第三季 Linux 内核和应用开发调试必杀技

第四季 进程管理和调度 / 中断 / 锁等

第五季 红帽系列



接下来就是在线安装插件了，启动 vim，然后运行命令 “:PluginInstall”，就会从网络上下载插件并安装。

## 2. 安装 ctags 工具

ctags 工具全称 Generate tag files for source code，它扫描指定的源文件，找出其中包含的语法元素，并把找到的相关内容记录下来，这样在代码浏览和查找的时候就可以利用这些记录来方便实现查找和跳转功能。ctags 工具已经集成到各大 Linux 发行版中，在 Ubuntu 使用如下命令可安装。

```
| sudo apt-get install ctags
```

在使用 ctags 之前需要手工生成索引文件。

```
| #ctags -R . //递归扫描源代码根目录和所有子目录的文件并生成索引文件
| 会在当前目录下面生成一个 tags 文件。启动 vim 之后需要加载这个 vim 文件。
```

```
| :set tags=tags
```

下面是在 vim 中使用 ctags 最常用的几个用法。

命令	用法
Ctrl + ]	跳转到光标处的函数或者变量的定义所在的地方。
Ctrl + T	返回到跳转之前的地方。

## 3. cscope 工具

刚才介绍的 ctags 工具可以跳转到标签定义的地方，但是如果想查找函数在哪里被调用过，或者标签在哪些地方出现过，那么 ctags 就无能为力了，这时候 cscope 可以找到这些功能，这也是 source insight 强大的功能之一。

cscope 最早由贝尔实验室开发，后来由 SCO 公司已 BSD License 开源发行。我们可以在 ubuntu 发行版中很容易安装它。

```
| sudo apt-get install cscope
```

在使用 cscope 之前需要对源代码生成索引库。

```
| #cscope -Rbq
```

上述命令会生成三个文件：cscope.cout、cscope.in.out 和 cscope.po.out。其中 cscope.out 是基本符合的索引，后面两个文件是使用 “-q” 选项生成的，用于加快 cscope 索引速度。

在 vim 中使用 cscope 非常简单，首先调用“cscope add”命令添加一个 cscope 数据库，然后就可以调用“cscope find”命令进行查找了。vim 支持 8 种 cscope 的查询功能，如下：

- ✓ s: 查找 C 语言符号，即查找函数名、宏、枚举值等出现的地方
- ✓ g: 查找函数、宏、枚举等定义的位置，类似 ctags 所提供的功能
- ✓ d: 查找本函数调用的函数
- ✓ c: 查找调用本函数的函数
- ✓ t: 查找指定的字符串
- ✓ e: 查找 egrep 模式，相当于 egrep 功能，但查找速度快多了
- ✓ f: 查找并打开文件，类似 vim 的 find 功能
- ✓ i: 查找包含本文件的文件

为了方便使用，我们可以在.vimrc 配置文件中添加如下快捷键。

```
"-----
" cscope:建立数据库 : cscope -Rbq;  F5 查找c符号; F6 查找字符串; F7 查找函数定
义; F8 查找函数谁调用了,
"-----
if has("cscope")
    set csprg=/usr/bin/cscope
    set cst=1
    set cst
    set nocsverb
    " add any database in current directory
    if filereadable("cscope.out")
        cs add cscope.out
    endif
    set csverb
endif

:set cscopequickfix=s-,c-,d-,i-,t-,e-

"nmap <C->s :cs find s <C-R>=expand("<cword>")<CR><CR>
"F5 查找c符号; F6 查找字符串; F7 查找函数谁调用了
nmap <silent> <F5> :cs find s <C-R>=expand("<cword>")<CR><CR>
nmap <silent> <F6> :cs find t <C-R>=expand("<cword>")<CR><CR>
nmap <silent> <F7> :cs find c <C-R>=expand("<cword>")<CR><CR>
```

快捷键如下：

F5: 查找 C 语言符号，即查找函数名、宏、枚举值等出现的地方

F6: 查找指定的字符串

F7: 查找调用本函数的函数

#### 4. Tagbar 插件

tagbar 插件可以把源代码文件生成一个大纲，包括类、方法、变量以及函数名等，可以选中快速跳转到目标位置。

安装 Tagbar 插件，在.vimrc 文件中添加：

```
Plugin 'majutsushi/tagbar' " Tag bar"
```

然后重启 vim，输入然后运行命令“:PluginInstall”完成安装。

配置 tagbar 插件。

```
" Tagbar
let g:tagbar_width=25
autocmd BufReadPost *.cpp,*.c,*.h,*.cc,*.cxx call tagbar#autoopen()
```

上述配置让打开常见的源代码文件的时候会自动打开 tagbar 插件。

#### 5. 文件浏览插件 NerdTree

nerdtree 插件可以显示树形目录。

安装 nerdtree 插件，在.vimrc 文件中添加：

```
Plugin 'scrooloose/nerdtree'
```

然后重启 vim，输入然后运行命令“:PluginInstall”完成安装。

配置 nerdtree 插件。

```
" NetRedTree
autocmd StdinReadPre * let s:std_in=1
autocmd VimEnter * if argc() == 0 && !exists("s:std_in") | NERDTree |
endif
let NERDTreeWinSize=15
let NERDTreeShowLineNumbers=1
let NERDTreeAutoCenter=1
let NERDTreeShowBookmarks=1
```

## 6. 自动补全插件 YouCompleteMe<sup>2</sup>

代码补全功能在 vim 历史中一直是一个鸡肋,因此一直被使用 source insight 的人诟病。比如之前出现的 AutoComplPop, omniscppcomplete, neocomplcache 等插件在效率上低的惊人,特别是把整个 Linux 内核代码添加到工程的时候,要使用代码补全功能每次都要等待 1~2 分钟的时间,简直让人抓狂。

YouCompleteMe 插件是最近几年才冒出的新插件,以及 clang 为 C/C++ 代码提供代码提示和补全功能。借助 clang 强大功能,补全效率和准确性极高,和 source insight 可以一比高下。因此 Linux 开发人员在 vim 上配备了 YouCompleteMe 插件之后完全可以抛弃 source insight 了。

在安装 YouCompleteMe 插件之前,需要保证 vim 的版本必须是大于 7.4.1578,并且支持 Python 2 或者 Python 3。Ubuntu 16.04 的版本中 vim 满足这个要求,使用其他发行版的同学可以通过如下命令来检查:

```
#vim -version
```

安装 YouCompleteMe 插件,在 .vimrc 文件中添加:

```
Plugin 'Valloric/YouCompleteMe'
```

然后重启 vim,输入然后运行命令“:PluginInstall”完成安装,这个过程需要从网络上下载代码,需要等待一段时间。

插件安装完成之后,需要编译它,所以在编译之前需要保证如下软件包已经安装了:

```
sudo apt-get install build-essential cmake python-dev python3-dev
```

接下来就要进入到 YouCompleteMe 插件代码中进行编译了。

```
cd ~/.vim/bundle/YouCompleteMe
./install.py --clang-completer
```

--clang-completer 表示对 C/C++ 的支持。

编译完成之后,还需要做一些配置工作,把 ~/.vim/bundle/YouCompleteMe/third\_party/ycmd/examples/.ycm\_extra\_conf.py 这个文件拷贝到 ~/.vim 目录下。

```
cp
~/.vim/bundle/YouCompleteMe/third_party/ycmd/examples/.ycm_extra_conf.py
~/.vim
```

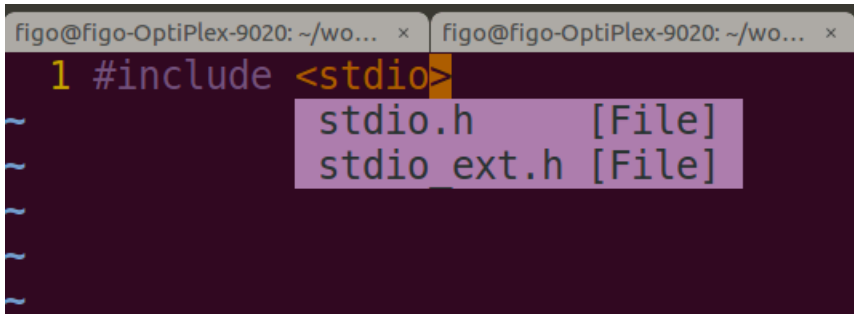
在 .vimrc 配置文件中还需要添加如下配置:

```
let g:ycm_server_python_interpreter='/usr/bin/python'
let g:ycm_global_ycm_extra_conf=~/.vim/.ycm_extra_conf.py'
```

这样 YouCompleteMe 插件。下面做一个简单测试,启动 vim,输入“#include <stdio>”

<sup>2</sup> <https://github.com/Valloric/YouCompleteMe>

这些字符的时候看看是否会出现提示补全。



## 7. vimrc 的其他一些配置

vimrc 还有一些其他常用的配置，比如显示行号等。

```
set nu!           " 显示行号

syntax enable
syntax on
colorscheme desert

:set autowrite    " 自动保存
```

## 8. 使用 vim 来阅读 Linux 内核源代码

我们已经把 vim 打造成一个媲美 source insight 的 IDE 工具了，下面介绍如何来阅读 Linux 内核源代码。

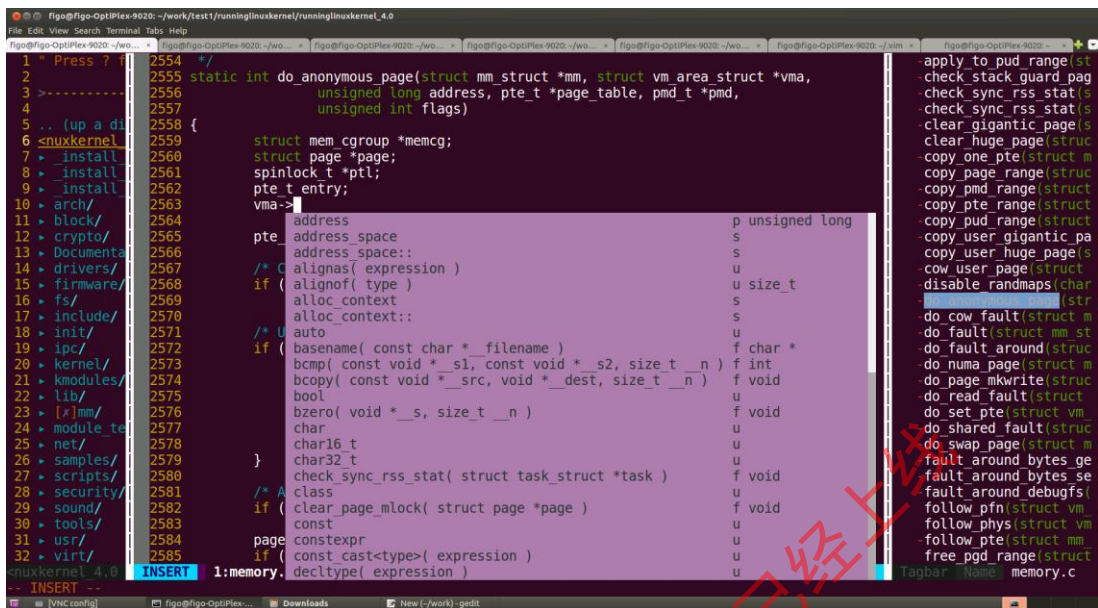
下载 Linux 内核官方源代码或者 runninglinuxkernel 的源代码。

```
git clone https://github.com/figozhang/runninglinuxkernel_4.0.git
```

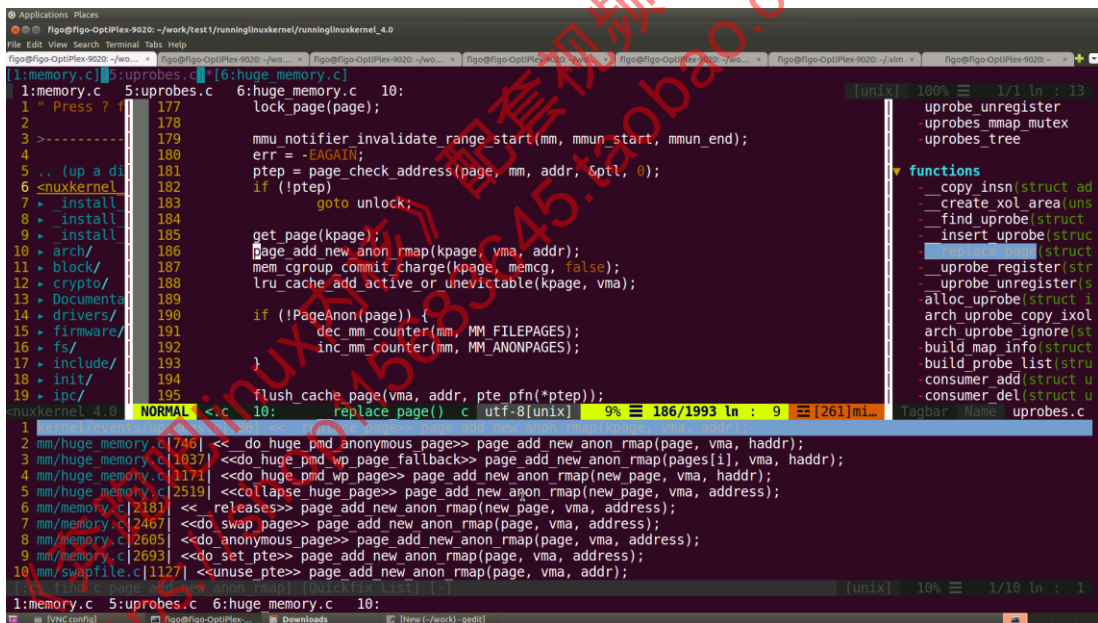
Linux 内核已经支持 ctags 和 cscope 来生成索引文件，而且会根据编译的 config 文件来选择需要扫描的文件。我们使用 make 命令来生成 ctags 和 cscope，下面以 ARM vexpress 平台为例：

```
#export ARCH=arm
#export SUBARCH=arm
#export CROSS_COMPILE=arm-linux-gnueabi-
#make vexpress_defconfig
#make tags cscope TAGS //生成tags,cscope, TAGS等索引文件
```

启动 vim，通过“:e mm/memory.c”命令来打开 memory.c 源文件，然后我们在 do\_anonymous\_page()函数即在第 2563 上输入“vma->”，你会发现 vim 自动出现了 struct vm\_area\_struct 数据结构的成员来让你选择，而且速度快的惊人。



另外我们同样在 do\_anonymous\_page() 函数的第 2605 行的 page\_add\_new\_anon\_rmap() 上按一下 F7 快捷键，我们发现很快查找了 Linux 内核中所有调用该函数的地方，如图所示：



更多精彩 Linux 内核视频节目请登录：

<https://shop115683645.taobao.com/>



更多精彩Linux内核视频节目请登录：  
<https://shop115683645.taobao.com/>