高效的编辑器vim

vim初级篇

一、初识vim

vim是目前最为高效的编辑器。其高效的原因有：

1. 键盘上的每个按键都是一个快捷指令
2. 有良好的扩展性，有丰富的扩展插件

> **接下来我主要以mstar fae工作场景下介绍如何快速高效实用vim，并不涵盖所有vim的功能**
> 希望通过这篇文档能够给你的工作带来帮助，或者让你喜欢上vim。

vim的四种模式

1. 普通模式
   [Esc] or [C-[]：在任何模式下切换到普通模式。
2. 编辑模式
   在普通模式下按下以下按键即可进入编辑模式
   i(I)：字符前插入；行头插入
   a(A)：字符后插入；行尾插入
   o(O)：行后插入；行前插入
   s(S)：删除字符后插入；删除行后插入
   c(C)：配合其他命令后插入；删除当前位置到行尾并插入
3. 命令模式
   ：：用于输入vim命令
   / or ？：用于向下搜索；向上搜索
4. 可视化模式
   v:进入可视化选择
   V:进入可视化行选择
   c-v:进入可视化块选择

二、认识vim的配置(.vimrc)

.vimrc文件是vim的配置文件，放在用户的更目录下。如果没有该文件可以自行新建。
vim每次开启时就会去解析.vimrc的配置。

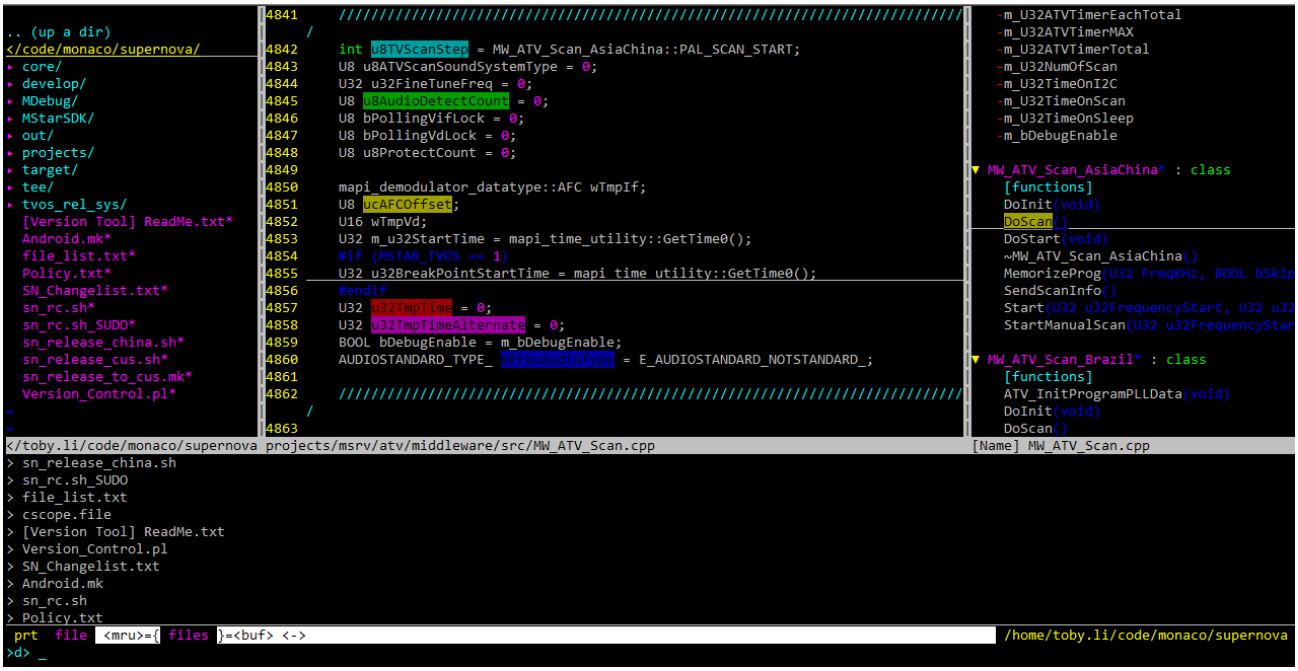
1. 未配置的vim

```
void MW_ATV_Scan_AsiaChina::DoScan() // including NTSC check
{
    ////////////////////////////////////////////////////////////////////////////////
    int u8TVScanStep = MW_ATV_Scan_AsiaChina::PAL_SCAN_START;
    U8 u8ATVScanSoundSystemType = 0;
    U32 u32FineTuneFreq = 0;
    U8 u8AudioDetectCount = 0;
    U8 bPollingVifLock = 0;
    U8 bPollingVdLock = 0;
    U8 u8ProtectCount = 0;

    mapi_demodulator_datatype::AFC wTmpIf;
    U8 ucAFCOffset;
    U16 wTmpVd;
    U32 m_u32StartTime = mapi_time_utility::GetTime0();
    #if (MSTAR_TVOS == 1)
    U32 u32BreakPointStartTime = mapi_time_utility::GetTime0();
    #endif
    U32 u32TmpTime = 0;
    U32 u32TmpTimeAlternate = 0;
    BOOL bDebugEnable = m_bDebugEnable;
    AUDIOSTANDARD_TYPE_ ePrevAudioType = E_AUDIOSTANDARD_NOTSTANDARD_;

    ////////////////////////////////////////////////////////////////////////////////

    //U8 waitTime;
    //char buf[10];
    ATV_Scan_LOG_FILE(FILE * pFile;)
    ATV_Scan_LOG_FILE(pFile = fopen("/Customer/scan_history.txt", "w");)
    ATV_Scan_LOG_FILE(fclose(pFile);)
    ATV_Scan_LOG_FILE(pFile = fopen("/Customer/vd_lock_history.txt", "w");)
    ATV_Scan_LOG_FILE(fclose(pFile);)
    ATV_Scan_DBG("=====================================================\nEnter Thre
```

2. 已配置的vim（未使用插件）

```
4839 void MW_ATV_Scan_AsiaChina::DoScan() // including NTSC check
4840 {
4841     ////////////////////////////////////////////////////////////////////////////////
4842     int u8TVScanStep = MW_ATV_Scan_AsiaChina::PAL_SCAN_START;
4843     U8 u8ATVScanSoundSystemType = 0;
4844     U32 u32FineTuneFreq = 0;
4845     U8 u8AudioDetectCount = 0;
4846     U8 bPollingVifLock = 0;
4847     U8 bPollingVdLock = 0;
4848     U8 u8ProtectCount = 0;
4849
4850     mapi_demodulator_datatype::AFC wTmpIf;
4851     U8 ucAFCOffset;
4852     U16 wTmpVd;
4853     U32 m_u32StartTime = mapi_time_utility::GetTime0();
4854     #if (MSTAR_TVOS == 1)
4855     U32 u32BreakPointStartTime = mapi_time_utility::GetTime0();
4856     #endif
4857     U32 u32TmpTime = 0;
4858     U32 u32TmpTimeAlternate = 0;
4859     BOOL bDebugEnable = m_bDebugEnable;
4860     AUDIOSTANDARD_TYPE_ ePrevAudioType = E_AUDIOSTANDARD_NOTSTANDARD_;
4861
4862     ////////////////////////////////////////////////////////////////////////////////
4863
4864     //U8 waitTime;
4865     //char buf[10];
4866     ATV_Scan_LOG_FILE(FILE * pFile;)
4867     ATV_Scan_LOG_FILE(pFile = fopen("/Customer/scan_history.txt", "w");)
4868     ATV_Scan_LOG_FILE(fclose(pFile);)
4869     ATV_Scan_LOG_FILE(pFile = fopen("/Customer/vd_lock_history.txt", "w");)
4870     ATV_Scan_LOG_FILE(fclose(pFile);)
4871     ATV_Scan_DBG("============================================================\nEnter Thr
```

3. 已配置的vim（已使用插件）



4. 我的.vimrc

由于.vimrc配置比较复杂，这里解释一些比较常用配置的意思。
没兴趣的可以直接忽略以下的解释

```
set number          显示行号
set incsearch       边输入边进行搜索
set hlsearch        搜索高亮
set ignorecase      忽略大小写
set smartcase       智能搜索（输入为小写时忽略大小写，为大写时则大小写敏感）
set cindent         使用c语法进行缩进
set shiftwidth=4    设置缩进为4个字符
set tabstop=4       设置缩进为4个字符
set softtabstop=4   设置缩进为4个字符
set showcmd         状态栏显示命令
set ruler           状态栏显示当前位置

let mapleader = ','                 设置[leader]键为", "
nnoremap <leader>" ciw""<esc>P      nnoremap:映射普通模式的快捷键
inoremap <c-b> <left>               inoremap:映射编辑模式的快键键
cnoremap <c-d> <del>                cnoremap:映射命令模式的快键键
vnoremap <leader>" di""<esc>P       vnoremap:映射可视化模式的快捷键
...
```

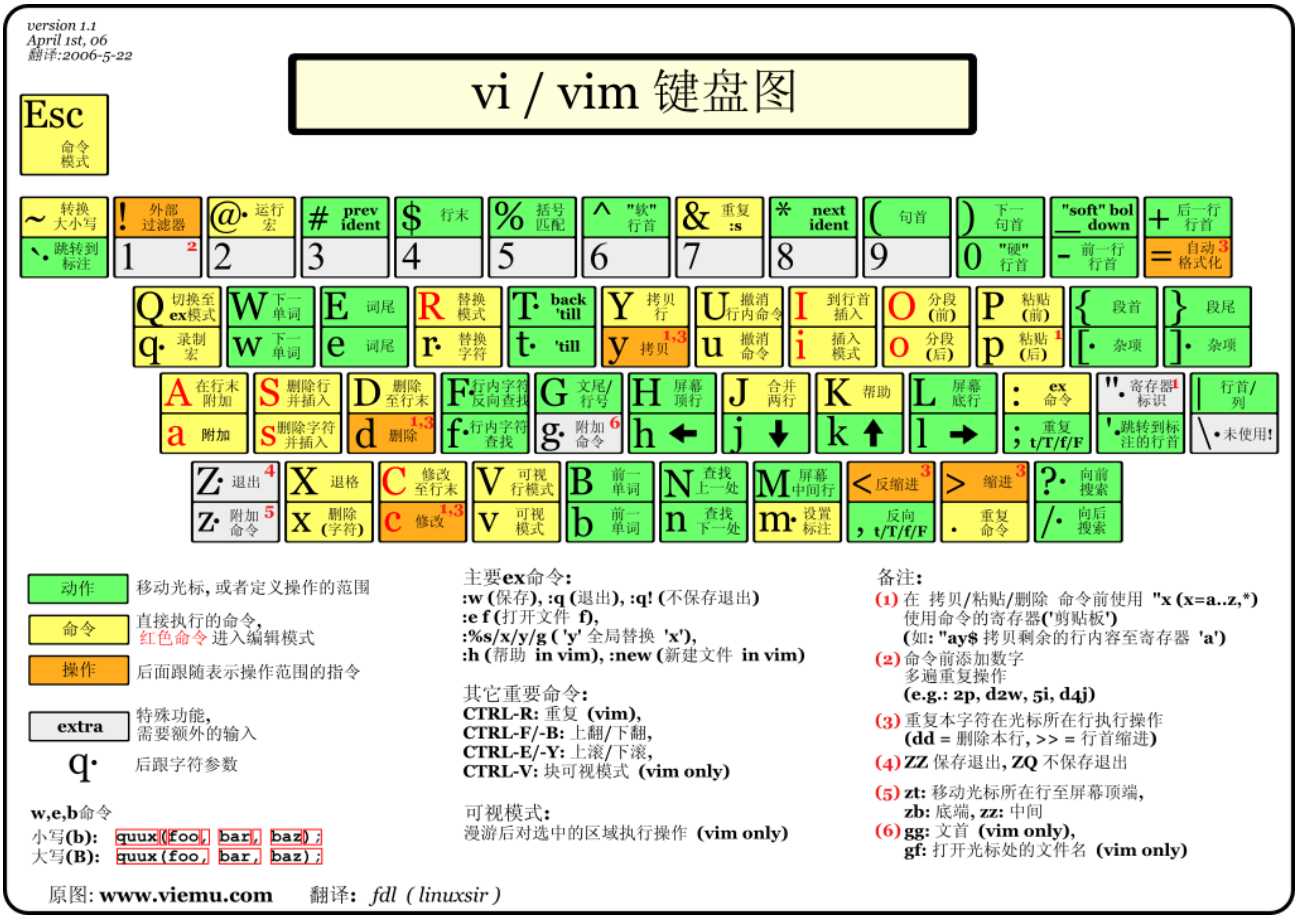
可以在vim通过执行（:help 关键词）去查看关键词的意思，或者百度一下。

> 附件上的.vimrc是我目前正在使用的配置文件，有兴趣的同事可以拷贝到自己的用户更目录下

三、vim高效使用技巧

vim键盘图

在普通模式下，每个按键都是一个快捷指令



使用技巧

1. vim快速跳转

文件内移动

[C-d] or [C-u]:翻半页
[C-f] or [C-b]:翻一页
gg:跳到文件头
G:跳到文件尾
[line]G:跳到某一行（可用：[line]代替）

> 定位的第一步：定位到对应行

行内移动

^ $：移动到行头；行尾
w: 移动到下一个单词头部
b: 移动到上一个单词头部
e: 移动到下一个单词尾部
ge:移动到上一个单词尾部

> 定位的第二步：定位到对应函数和变量

标签跳转

m{a-zA-Z}: m后面加字母，在当前行设置标签
‘{a-zA-Z}: ‘后面加字符，跳转到对应标签
:marks : 查看所有标签

> 小写只支持文件内跳转，大写支持文件外跳转

其他跳转

%：切换到对应的符号（包括各种括号，还有#if #endif）

> 使用场景：浏览代码时遇到长代码if/while/for以及#if #endif时可用

[[:跳到函数头
]]:跳到函数尾

[c-o]：向前跳转
[c-i]：向后跳转

> 配合ctags工具使用，类似sourceinsight的向前向后跳转功能

[c-w][c-w]:窗口之间跳转
[c-w]{hjkl}:跳转到对应方向的窗口

> 使用插件时会出现多窗口，利用这个快捷键可以在窗口之间来回切换

2. vim快速编辑

普通模式下

yy dd p:复制删除粘贴
<< >>：向左缩进，向右缩进
==：自动缩进

> 输入该指令前可先敲入数字[num]，表示往下[num]行都执行该命令

x:删除键
daw/caw/yaw:删除/修改/拷贝当前光标下的单词
. :重复上一个命令

编辑模式下

[C-p] or [C-n]: 自动补全

3. vim快速搜索

*：向下搜索当前光标下的单词
#：向上搜索当前光标下的单词

/ ?：进入搜索模式
[c-p],[c-n]：可以查看上一个搜索历史和下一个搜索历史

4. vim其他技巧

1. 批量注释

```
^           移动到行首
c-v         进入可视化块选择
jjj...      输入多个j,选择你要注释的行
I           进入编辑模式
//          插入注释符号//
[esc]       退出编辑模式
```

1. 批量去注释

```
^           移动到行首
c-v         进入可视化块选择
hjkl        通过hjkl选择要删除的字符块
x           删除
```

5. 《vim实用技巧》

> 这里极力推荐一本书，这本书能够使你的vim能力得到进一步提升

vim高级篇

vim的初级篇介绍的是不带任何插件下的vim支持的功能。而接下来是介绍如何通过vim插件来武装自己的生产工具vim。
用户根目录下.vim是放置vim插件的目录

> **附件上的bundle.tar.bz2已经包含了工作需要的vim插件，只需解压到用户根目录下的.vim即可**
>
> vim插件主要下载来源：
> https://github.com
> vim官网

> **由于mstar fae的工作性质，我仅介绍以下能够提高工作效率的高效插件，而且也仅介绍在工作使用到的功能。**
> 如果想了解该插件更深入的功能，可以点击以下超链接进入该工具的相关网页，或者在vim敲入:help [插件名]进行查看。
>
> 1. Vundle.vim
> 2. tagbar
> 3. ctrlp.vim
> 4. cscope.vim
> 5. ctags
> 6. vim-snippets
> 7. ultisnips
> 8. mark.vim
> 9. nerdtree

1. Vundle.vim

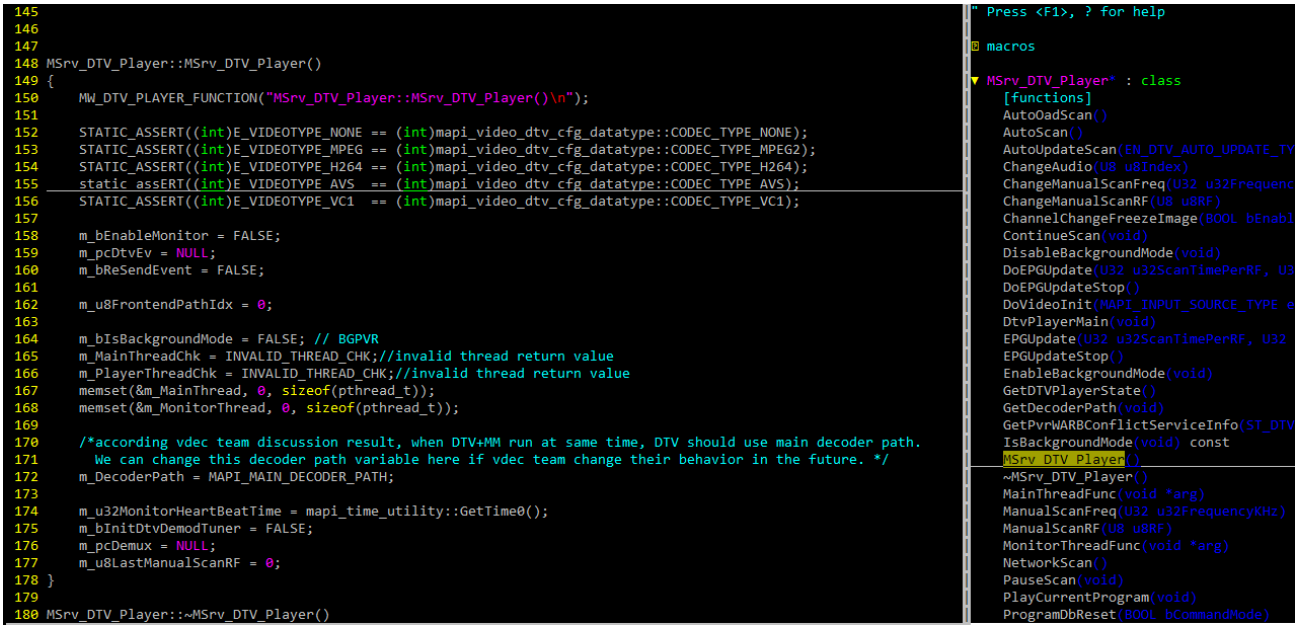
vundle.vim是vim插件管理工具，得力于该插件，使得vim能够快速安装和管理插件。
由于附件上已经有了工作所需插件，所以这个工具的使用方法可以跳过。

> 举例（如要安装ctrlp.vim插件）：
>
> 1. 下载插件ctrlp.vim
> 2. 放在~/.vim/bundle目录里
> 3. 并在.vimrc以下添加对应语句：

```
set nocompatible
syntax on
filetype off
if isdirectory(expand("~/.vim/bundle/Vundle.vim"))
    set rtp+=~/.vim/bundle/Vundle.vim/
    call vundle#begin()
    Plugin 'VundleVim/Vundle.vim'
    Plugin 'majutsushi/tagbar'
    Plugin 'scrooloose/nerdtree'
    Plugin 'kien/ctrlp.vim'                    在这添加对应语句
    Plugin 'brookhong/cscope.vim'
    Plugin 'honza/vim-snippets'
    Plugin 'SirVer/ultisnips'
    Plugin 'OmniCppComplete'
    Plugin 'mark.vim'
    call vundle#end()
endif
filetype plugin indent on
```

2. tagbar

用于查看函数列表



快捷键

,t : 打开或关闭tagbar

（快捷键记忆：，为前缀，t取tagbar的开头字母）

这里解释一下为何,t是打开tagbar的快捷键。

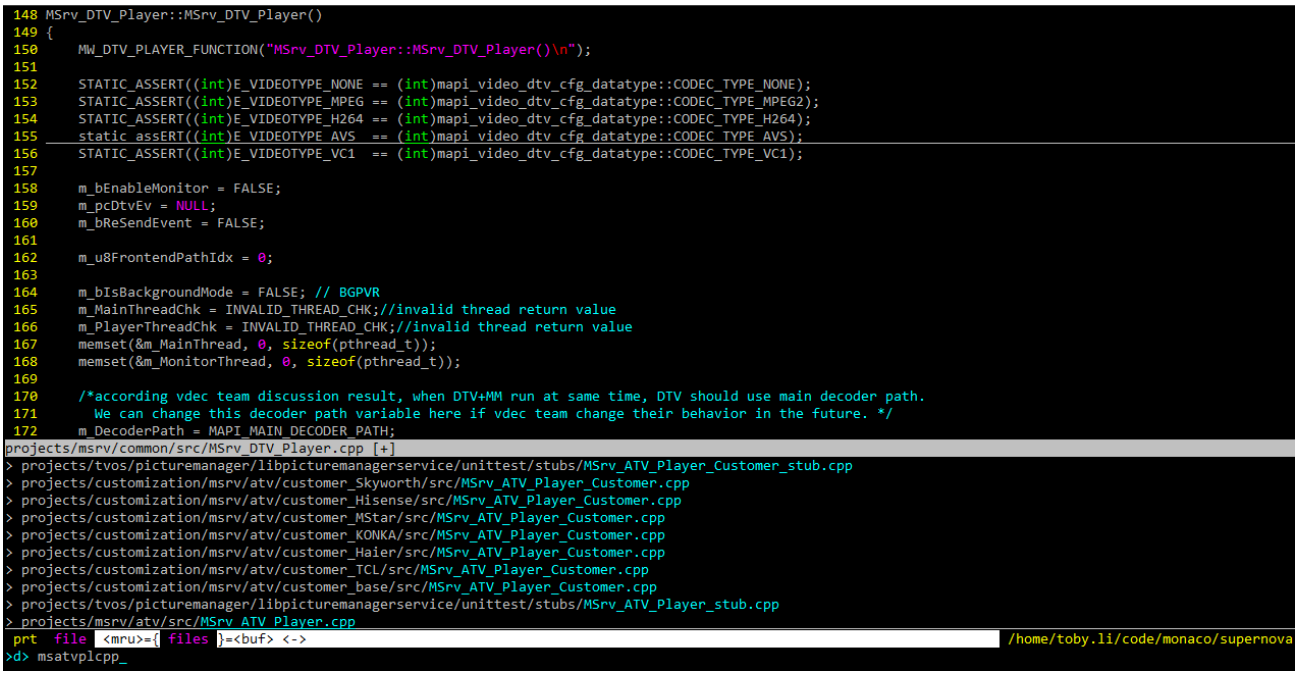
基本上所有插件的操作都为命令，也可以说是函数。如tagbar的打开方式其实为:TagbarToggle,其实就是调用了TagbarToggle的函数。而,t只不过是我将其映射而已，所以方便了操作。以下的插件基本都映射了快捷键，所以在之后的插件介绍就不做解释了。

```
.vimrc文件里已经将打开tagbar映射为,t
" tagbar
nnoremap <silent> <leader>t :TagbarToggle<CR>
```

> **在tagbar窗口上**
> [enter] : 跳转到该函数
> s : 切换排序（按字母排序，按文件顺序排序）
> ? : 查看帮助

3. ctrlp.vim

快速文件模糊搜索，这个插件可以说是提高工作效率的必备武器。



> **快捷键**
> ,p : 打开ctrlp，默认搜索包含.git,.svn,ctags目录下的所有文件或者当前工作目录下的所有文件
> （快捷键记忆：，为前缀，p取ctrlp的最后一个字母）
> ,b : 打开ctrlp，默认搜索当前被使用的所有文件
> （快捷键记忆：vim将当前被使用的文件为buffer，所以快捷键取buffer的第一个字母）
> ,u : 打开ctrlp，默认搜索最近使用的文件
> （快捷键记忆：快捷键取use的第一个字母）
> :help ctrlp : 更高阶的使用技巧请查看ctrlp的帮助

> **在ctrlp窗口上**
> [enter] : 打开选择的文件
> [c-j][c-k] : 选择文件
> [c-p][c-n] : 搜索上一个或下一个搜索历史

4. cscope.vim

该插件用于函数查找，类似SI的功能。不过cscope能够显示函数的调用者以及行数。

```
1067 }
1068
1069 mapi_storage * mapi_interface_common::Get_mapi_storage_factory_backup()
1070 {
1071     mapi_storage *pRet;
1072     ASSERT(m_pMapiObj[E_OBJ_STORAGE_FACTORY_BACKUP] != NULL);
1073     pRet = dynamic_cast<mapi_storage *>(m_pMapiObj[E_OBJ_STORAGE_FACTORY_BACKUP]);
1074     ASSERT(pRet);
1075     return pRet;
1076 }
1077
1078 #endif
1079 mapi_video_out *mapi_interface_common::Get_mapi_video_out(MAPI_VIDEO_OUT_DEST_TYPE enDest)
1080 {
1081     mapi_video_out *pRet = NULL;
1082 #if (VE_ENABLE == 1 || CVBSOUT_ENABLE==1)
1083     if(enDest == MAPI_VIDEO_OUT_MONITOR_MODE)
1084     {
1085         ASSERT(m_pMapiObj[E_OBJ_VIDEO_OUT_CVBSOUT1] != NULL);
1086         pRet = dynamic_cast<mapi_video_out *>(m_pMapiObj[E_OBJ_VIDEO_OUT_CVBSOUT1]);
1087     }
1088 #if (STB_ENABLE == 0)
1089     else
1090     {
1091         ASSERT(m_pMapiObj[E_OBJ_VIDEO_OUT_CVBSOUT2] != NULL);
~/code/monaco/supernova/MStarSDK/src/interface/mapi_interface_common.cpp
  1 /home/toby.li/code/monaco/supernova/MStarSDK/src/interface/mapi_interface_common.cpp|1079| <<Get_mapi_video_out>> mapi_video_out *mapi_interface_common::Ge
    t_mapi_video_out(MAPI_VIDEO_OUT_DEST_TYPE enDest)
```

使用该插件前，请确认当前使用的服务器已安装cscope。
如果没有安装，可通过以下途径解决：

1. apt-get install cscope 安装cscope
2. 拷贝附件的bin/cscope到~/bin/

> 快捷键
> ,fg : 查找到函数定义
> ,fc : 查找到函数调用
> ,fs : 查找到c符号（一般用于变量）
> ,ft : 查找到该字符串
> ,fa : 自定义查找(大小写敏感)
> 另外，上文介绍的[ctrl-o][ctrl-i]在这里就可以排上用场了

> .vimrc配置：
> let g:cscope_interested_files = '.c$|.cpp$|.h$|.java$'
> g:cscope_interested_files是设置感兴趣的文件。
> 后面的字符串为正则表达式，表示只添加.c,.cpp,.h,.java文件

> 第一次使用时会提示创建数据库

```
770     eCountry = eCountry;
771 #if (ATSC_SYSTEM_ENABLE == 1) // NTSC
772     m_bIsMTSMonitorEnabled = FALSE;
773     mScan = (MW_ATV_Scan *) new(std::nothrow) MW_ATV_Scan_NTSC;
projects/msrv/atv/src/MSrv_ATV_Player.cpp
Can not find proper cscope db, please input a path to generate cscope db for.
/home/toby.li/code/monaco/supernova/
```

> 在任意单词下敲,fg,则会弹出如上图的提示框。
> 只要输入要创建的目录即可，如/home/toby.li/code/monaco/supernova

由于cscope本身的缺陷，当code修改得比较多时，会出现定位不准确的情况，建议重新创建数据库。

1. :CscopeClear 删除数据库
2. 如上面步骤创建数据库

5. ctags

该工具其实不是vim插件，但也算是vim的工具之一。该工具与cscope类似，用于函数跳转，可是ctags只支持跳转到定义。当时建议安装，弥补cscope自定义搜索的缺陷。
使用该工具前，请确认当前使用的服务器已安装ctags。
如果没有安装，可通过以下途径解决：

1. apt-get install exuberant-ctags 安装ctags
2. 拷贝附件的bin/ctags到~/bin/

快捷键
[c-]] : 查找函数定义
:tag [函数名]：查找函数定义（大小写不敏感）
:ts : 显示当前的所有查找

第一次使用时需创建的数据库

1. 可以将附件上bin/ctags_generate.sh拷贝到~/bin
2. 在要生成数据库的目录执行ctags_generate.sh，则会在当前目录下生成ctags文件。(如在supernova下执行ctags_generate.sh)

6. vim-snippets

该工具用于辅助ultisnips插件，包含了片段补全规则。语法非常简单，以至于可以自定义片段补全规则。
可惜我们的工作性质并不是开发，导致这个插件的功能打了折扣，不过我们还是可以利用它来方便快捷地为code添加log。

小tip：我们在添加log时尽量使用高亮和特殊颜色的log，方便我们快速分析log。

如何添加片段规则

1. 进入.vim/bundle/vim-snippets/UltiSnips/
2. 该目录下有很多以snippet为后缀的文件，如c.snippet文件为c/c++文件使用的，java.snippet为java文件使用的，cpp.snippet为c++文件使用的。
3. 由于公司的code为c/c++文件，所以可在c.snippet或者cpp.snippet进行添加自定义的s片段补全规则。
4. 以下是我在c.snippet添加的三个片段补全规则，分别是pp,pbl,pgr.

```
# mstar
snippet pp "print position"
printf("\033[1;44m[toby.li][%s:%s:%d]\033[m\n", __FILE__, __FUNCTION__, __LINE_
endsnippet
snippet pbl "print blue"
printf("\033[1;44m[toby.li]${1:string}\033[m\n", ${2:args});
endsnippet
snippet pgr "print green"
printf("\033[1;42m[toby.li]${1:string}\033[m\n", ${2:args});
endsnippet
```

1. snippet的入门语法(以第一个补全规则为例)
   a. 其中snippet…endsnippet为片段补全的基本语法，编写的规则必须在这里面。
   b. pp为片段关键词，在vim编辑时敲入pp[tab]，就会将第三行的printf…输入到vim上。
   c. “print position”, 该片段规则的注释。
   d. printf…,pp片段的补全。
2. snippet的进阶语法（以第二个补全规则为例）
   a. 其中可以看到${1:string}和${2:args}，分别表示片段的第一个位置和第二个位置，编辑时通过[c-j][c-k]进行选择，string和args表示默认时显示的字符串。
   试一下你就知道这个怎么玩了。
3. snippet的高级语法
   可以通过自定义函数来达到更高级的补全。目前来看，工作根本用不上，可以忽略。
   喜欢的可以在这个网址学习其用法：https://github.com/SirVer/ultisnips

7. ultisnips

该工具为片段补全工具，不过该工具需要运行在vim7.4以上且支持python的版本。
目前公司的大部分服务器使用的还是vim7.3版本，而且服务器上apt-get源支持的vim最新也是7.3版本。所以建议手动将vim更新为8.0版本。

> 如何手动编译安装带有python的vim8.0
>
> 1. 解压附件的vim-8.0.tar.bz2
> 2. cd vim80/src
> 3. ./configure –enable-pythoninterp（如果要自定义安装目录，使用./configure –enable-pythoninterp –prefix=[安装目录]，安装完则需在PATH添加安装路径）
> 4. make
> 5. sudo make install
> 6. 如果需要用vi直接调用vim，可以如下操作

```
which vi    查看vi的path,一般为/usr/bin/vi
whick vim   查看vim的path,一般为/usr/local/bin/vim
mv /usr/bin/vi /usr/bin/vi.bak            备份vi
ln -s /usr/local/bin/vim /usr/bin/vi      用vi链接到vim
```

> 快捷键
> [tab]：输入片段关键词，后按[tab]进行补全
> [c-j][c-k]:向前一个位置/后一个位置移动

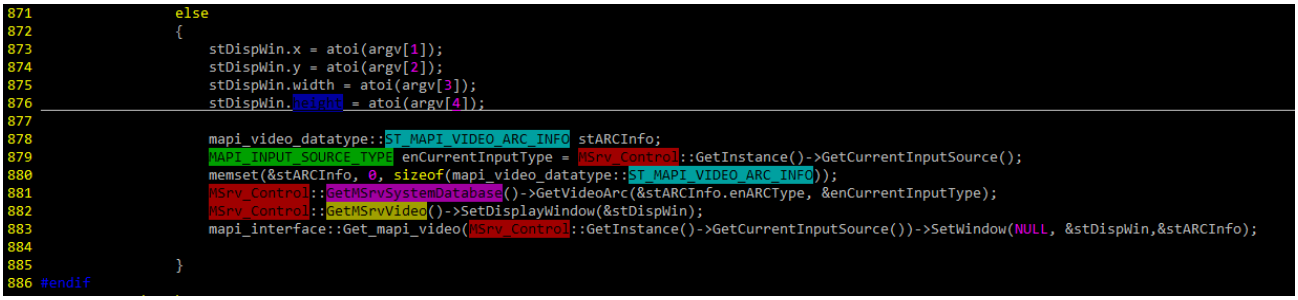
> 个人觉得在工作上快速加log很有必要。但如果有觉得为了这点效率而去配置这么复杂的事情相当的没必要。那么下面推荐另外一个技巧：

```
1. 在.vimrc中映射快捷键，如
2. nnoremap <leader><leader>p iprintf("\033[1;44m[toby.li][%s:%s:%d]\033[m\n", __FILE_
```

则在vim普通模式，敲入,,p,就会自动输入printf(“\033[1;44m[toby.li][%s:%s:%d]\033[m\n”, **FILE**, **FUNCTION**, **LINE**);

8. mark.vim

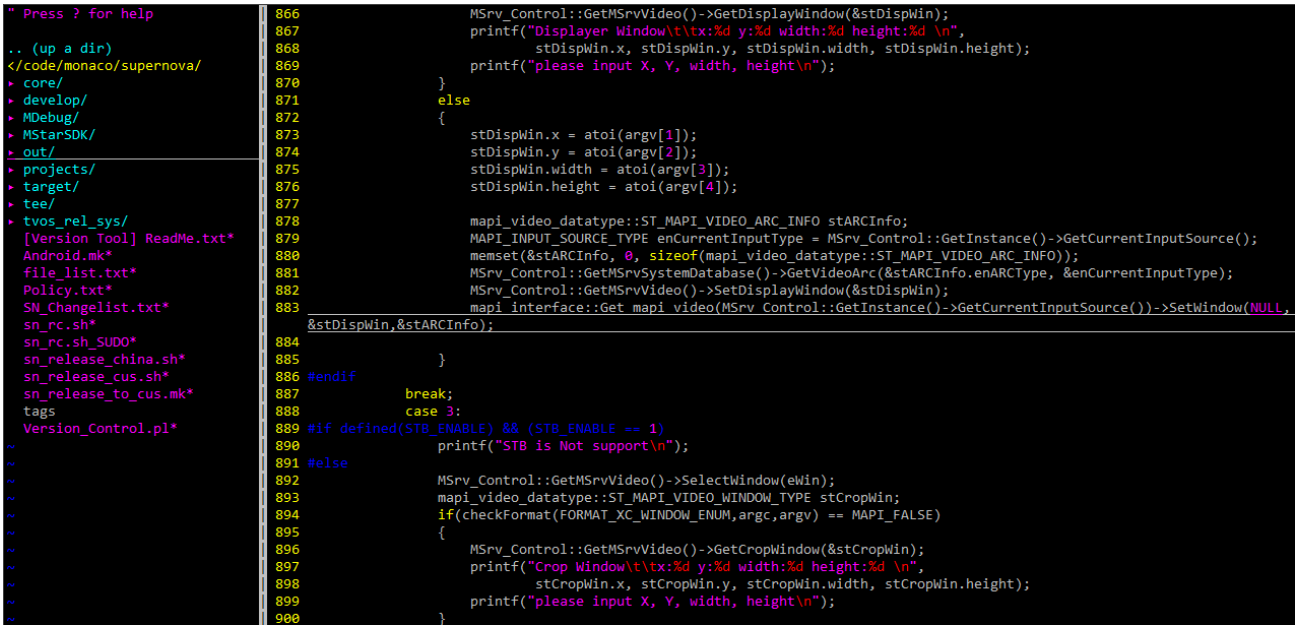
高亮光标下的单词，最多支持6种颜色。该工具主要用于查看code flow和log文件分析。



> 快捷键
> ,m : 高亮或取消高亮当前光标下的单词

9. nerdtree

一个文件管理和查看工具，不过工作上使用很少。



> 快捷键
> ,nt：打开nerdtree

> ,nf：打开nerdtree,并定位到当前文件