

Vim-Sztools 使用手册

左晃右过 <shrek.wang 挨特 gmail.com>

May 6, 2012

第1章 简介

工欲善其事，必先利其器

对于开发人员来说，选好一个工具并用到纯熟，无疑对工作效率有很好的提升。文本编辑器，编译器，调试器就是开发必不可少的几个工具。Vim 是功能最强大的文本编辑器之一 (另一个是 Emacs)。但是 Vim 中可选的 java 开发类插件并不多，可能是用 Vim 做 java 开发的程序员太少了。许多 Java 程序员现在都用的是 eclipse, jbuilder 之类的 IDE，不过 eclipse 除了其臭名昭著的体积庞大，运行慢等缺点之外，其编辑功能跟 vim 比起来，实在连鸡肋都算不上。

vim 的 java 开发插件虽不多，但是在 vim 官网已经有一个比较著名的 java 开发插件，VJDE。另外 eclim 直接将 eclipse 做为后端，将很多功能直接引入了 vim。而 eclipse 本身作为一个开发工具平台，上面也有形形色色的仿 vim 插件 (参见附录)。因此无论是在 vim 中用 java 开发类插件或是在 eclipse 中安装 vim 插件启用 vim 模式，似乎都有不错的选择。为什么还要再做一个插件 Vim-Sztool 呢

1. VJDE 每次补全都需调用外部 java 命令用反射来读取类信息，速度比较慢。eclim 因为用 eclipse 作为后端，速度也是偏慢。
2. 无法在 vim 中调试 java 程序。
3. 此处省略 n 条理由...

相比于现有的 java 开发类插件，Vim-Sztool 有自己的一些特色

1. 快速的补全 (类相关信息缓存在 Agent 中)
2. java 程序调试功能，可能是仅有的支持 java 调试的 vim 插件
3. 定制的 ProjectTree，项目中引用的源码包 (source jar) 也按树结点显示。
4. 快速文件定位，类似于 Eclipse 中的 open resource 功能。
5. 源码跳转，支持跳转到 source jar 包中的文件。
6. 支持 java 程序调试并支持远程调试

Vim-Sztool 采用了类似 eclim 的方法，将 java 类的编译，调试等大部份的功能放在服务端执行，然后通过 gvim 支持的 remote-expr(或 remote-send) 功能和 netbeans 协议在服务端和 vim 之间进行通讯。在 eclim 里，这个服务端是 eclipse 的插件，所以用 eclim 的时候，用 vim 编辑个 java 文件还得开个 eclipse 作为后端，这个是比较影响速度的。Vim-Sztool 自己实现了 Agent，功能少，速度还行。这个 Agent 程序在 vim 启动时自动运行 (如果安装没问题的话)。

Vim-Sztool 除了 java 的编译补全之外，还另外加了些功能。比如 Shext，可以执行一些简单的 shell 命令，比如 Dbext，用来做一些简单的 SQL 查询。具体的可以看后面的”使用”一章。

由于 Vim-Sztool 大量采取了 split 窗口的方式，对于习惯单窗口操作的 vimer 可能会有点不适应。这个目前也没什么好的办法，有的功能不用 split 窗口的方式比较难以实现。如果要使用此插件的，定义快速切换窗口和 tab 页的 map 是有必要的。我自己在 vimrc 中做了如下设置：

```
map <C-j> <C-W>j
map <C-k> <C-W>k
map <C-h> <C-W>h
map <C-l> <C-W>l

map <M-h> gT
map <M-l> gt
```

这样切换窗口和 tab 就可以比较快捷一点。推荐大家也使用合适自己的 map。

第2章 安装和配置

2.1 需求

Vim-Sztool 是一个用 python, java, vim 脚本混合编写的插件, 要使此插件能运行, 需求如下:

软件:

- vim7.3, 只支持 gvim 版本
- Jdk1.6 以上
- Python2.7 (2.6 的也许可以运行), 需安装 pyparsing, BeautifulSoup, chardet 模块

使用人群:

- 熟悉 vim, 懂一点 vim script
- 了解 python, 会自己安装一些 python 模块, 比如上面提到的 pyparsing
- 有用 vim 写 java 程序的需求

2.2 安装

1. 确保已经安装了 JDK,python 和 vim73, 和 python 的依赖模块
2. 设置环境变量: JDK_HOME, path(需要把 gvim, python,java 的可执行文件目录加到 path 中)
3. 解压安装包, plugin 目录覆盖到 vim 插件路径上, 或者设置 runtimepath, 比如

```
set runtimepath+=D:\\soft\\vim-sztool
```

4. 在 vimrc 中添加全局变量 g:sztool_home, 注意是插件目录下的 sztools 目录。

```
let g:sztool_home="D:\\soft\\vim-sztool\\sztools"
```

5. 源码安装。源码位于 <https://github.com/shrekwang/vim-sztool>, 可以 git clone 此项目, 代替下载的安装包 (不覆盖到 vim 默认 plugin 目录, 只设置 runtimepath), 这样易于更新和管理。clone 完后, 需用 ant 运行 sztools/ant/build.xml, 以编译和生成 Agent 程序的 jar 包。

如果安装成功, 此时启动 gvim, 在系统托盘区会有个军刀的图标, 这个是 Agent 程序, 说明已经安装成功了。



Figure 2.1: 托盘区 Agent 程序截图

2.3 配置

Vim-Sztool 的默认配置文件位于安装目录的 sztools/share/conf 目录下, 一般不要改动此文件中的内容, 而是把文件复制到 ~/.sztools/ 目录下进行改动。Vim-Sztool 会读取此两个目录下的内容, 优先级以 home 目录下的为高。

sztools.cfg 主要配置文件, 用于配置 jde 的一些基本设置

vars.txt 配置.classpath 文件中引用的变量

stepfilters.txt 配置 jdb 调试时, 略过的一些 package

db.conf 配置 Dbext 用到的数据库连接

shext-bm.txt 配置书签, 参见 Shext 中的 bm 相关命令

第3章 使用

本章讲 Sztools 的使用。后面的文本可能会有大量的”窗口”, ”buffer”, ”tab 页” 这些术语, 如无特殊说明, 这些都是指 vim 中的概念。如果不熟悉这些术语, 请查看 vim 的帮助文档。可以使用 :help window。

3.1 Shext

Shext 是一个仿 shell 的东西, 启动后, vim 会被 split 成上下两个 buffer。如图

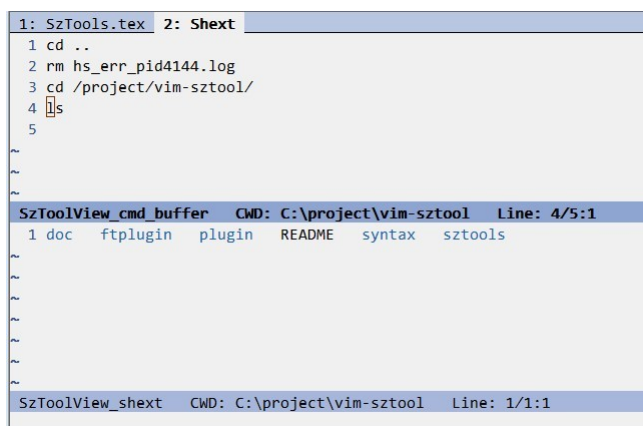


Figure 3.1: Shext 截图

用 `:Shext` 启动, 启动后, 会把当前 buffer 作为命令编辑区, 用于编辑和运行命令。split 出来的下侧 buffer 是命令输出区, 用于显示命令的输出。在启动前, 确保当前的 tab 页是空的。否则会把当前编辑的文件作为命令编辑区, 一般这不是想要的情况。Shext 支持内置实现的命令和系统的命令, 内置命令优先。不支持交互式和需要用到终端库的命令, 如 ftp, mysql, vim, mc 等。

在命令编辑 buffer, 如果输入 `"/`, 如果对应的参数是目录路径, 则在命令

输出 buffer 会显示对应路径的内容作为提示。比如

```
cd c:/
```

在”/”按下后，会列出 c: 根目录的内容。(Shext 默认在 linux 和 win 下都使用”/”作为目录分隔符)

在命令编辑区，在 insert 模式下输入”\$n;;”，则会把输出 buffer 的第 n 行补全到当前命令行中。在某些条件下会比较有用。

比如你可以先用 `find -name *.java -text Apple` 查找包含 Apple 的 java 文件，在找到后，如果想编辑其中的第 5 个文件，则可以用”`edit $5;;<cr>`”来实现。注意这个补全是带状态的，在 `$5;;` 后，如果你还要编辑上一个文件，可以用 `<C-p>`，如果想编辑下一个文件，可以用 `<C-n>`。

3.1.1 内置命令

内置命令是插件自己实现的命令，大多数是 linux 下命令的简化版。本来这些命令直接调用系统命令会更强大，但由于我的工作环境主要是 windows，另外像 `ls` 的带色彩化输出也难以做到，所以才用 python 和 java 简化实现了一点。`ls` 的输出做了些定制，支持对目录和压缩文件，以及可执行文件用不同的颜色显示。具体的颜色可以在 `~/.sztools/sztools.cfg` 文件中配置。

改变和显示目录

`pwd` : 显示当前目录
`cd args` : 更改当前目录，可使用bookmark和通配符。
`cdlist` : 列出cd过的目录历史。
`lsd` : 列出当前目录下的子目录。

举例:

```
cd ~/.vim/plugin
```

到 home 目录的.vim/plugin 目录

```
cd work/project
```

其中 work 是定义的一个书签，并非子目录

```
cd *.doc
```

如果当前目录下只有一个 doc 结尾的子目录，则进入该子目录

```
cd ../sr*/net/s*/../icons/
```

通配符和.. 可以结合着用

```
cd -
```

回退到 cd 前的目录

```
ls [ -l | -L ] [ -t | -s | -n ] [ --help ] [args]
```

列当前目录内容。

- l 表示长格式， 每一行列一个文件
- t 表示按时间排序
- s 表示按大小排序
- n 表示按名称排序
- help 打印ls命令的帮助信息

因为实现上的原因， 以"."开头的文件默认是不显示的， 你可以用 "ls ." 来显示所有文件

书签

bmadd : 将当前目录加到bookmarks中， 能在"cd"命令中引用
bmedit : 编辑"bookmarks"文件
bmlist : 列出所有bookmark。

举例:

```
bmadd
```

如果当前的目录为/usr/local/tomcat， 调用 bmadd 后， 会增加一个名为 tomcat 的书签。这时， 随便你在哪一级目录， 只要 cd tomcat 就能进到/usr/local/tomcat。书签以文本格式存放在 ~/.sztools/shext-bm.txt 文件中。可以用 bmedit 命令编辑

文件管理

touch [args] : 新建文件或者更新文件时间戳
rm [args] [-r] : 删除文件或目录
mkdir [args] : 新建目录
rmdir [args] : 删除空目录
cp [src...][dst] : 复制文件或目录
mv [src...][dst] : 移动文件或目录


```

echo [args]          : 打印文本消息
merge [src...][dst] : 合并多个文件
cat [arg]            : 显示文件内容
head [arg]           : 显示文件前10行
edit [args]          : 在打开新tab, 并编辑文件

```

```
find [-n name][-t text] [-s size] [-p path] [--help] [args]
```

查找文件。

`-n name` 按文件名查找

`-t text` 查找包含文件的文件(非正则)

`-s size` 按文件大小查找

命令默认在当前目录查找, 可以在args中指定要查找的目录。

Locate 命令

此命令在Locate一节描述

杂项

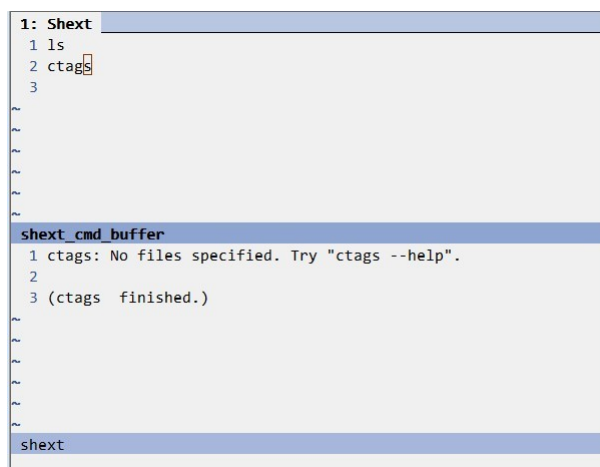
`help` : 打印Shext帮助信息

`exit` : 退出Shext。

`jde` : jde相关命令, 目前实现了 `jde start`(参见后面的启动Jde一节)和`jde help`。

3.1.2 系统命令

Shext 也支持直接执行系统命令。



```

1: Shext
1 ls
2 ctags
3
~
~
~
~
shext_cmd_buffer
1 ctags: No files specified. Try "ctags --help".
2
3 (ctags finished.)
~
~
~
~
shext

```

Figure 3.2: Shext 中执行系统命令截图

对于 windows 系统，可以执行”explorer.” 来用资源管理器打开当前目录。

不支持需要交互的命令，比如 `svn commit -m "asd"` 是可以的，直接 `svn commit` 却不行，因为后者需要从标准输入读入信息。

3.2 Jdext

Jdext 包含了 java 相关的功能，只有在 Jdext 启动后，本节介绍的功能才可以使用。另外在 Jdext 启动时，会定义很多的 map，比如 `<leader>go`¹等，如无特别说明，这些都是指 normal 状态下的 map。

3.2.1 项目结构

Jdext 读取的是 eclipse 的项目管理文件，在项目根目中，有一个 `.classpath` 和 `.project` 文件。但是即使没有在 eclipse 建项目，只要项目目录中有这两个文件，而且能正确解析的话，也是可以的。一般建议用 eclipse 建立项目，包括设置类路径等。在项目建立完成后，再在 Vim 中进行开发。

对于使用 maven 的项目来说，还可以直接用 maven 命令行建 maven 项目，然后用

```
mvn eclipse:eclipse
```

来生成 eclipse 项目文件。此种方式生成的 `.classpath` 文件，引用的 jar 包都是以 `M2_REPO` 来开头的，Vim-SzTool 在默认情况下找不到这个变量相对应的路径。需要在 `~/.sztools/vars.txt` 配置变量对应的路径。格式很简单

```
M2_REPO = /usr/maven/repo
```

老的 maven 项目，在生成 `.classpath` 文件后，需要再运行 `mvn compile` 编译项目代码，Jdext 启动时需读取类信息，以支持代码补全功能。

对于简单的类，如果只引用 JDK 自带类库的话，也可以不用项目结构。在 Jdext 启动后，直接在任意目录新建 Java 文件，然后就可以保存编译和调试（对于 jdk 自带类库的引用，也支持代码补全）。

对于需要几个 jar 包的小工程，也可以新建空目录后，然后 `cd` 到空目录，运行 `:InitProject` 命令来初始化项目结构。这个命令会生成 `.classpath` 和 `src`, `lib` 目录。然后你可以拷自己的 jar 包到 `lib` 目录，并自己编辑 `.classpath` 来更新类路径。

¹对于 `<leader>` 的含意，请查看 vim 中的帮助：`help <leader>`，在默认情况下这个是指按键 `\`，但是我一般设置 `let g:mapleader = ","`。

3.2.2 启动 Jdext

有几种方式可以启动 Jdext:

1. 调用:Jdext 启动, 此命令会设置很多的 map, 比如 `<leader>go`。一般需要在项目文件夹下运行此命令 (能在当前目录中查找到.classpath 文件), 则 Agent 程序会在后台缓存类信息, 以加快补全速度。
2. 在 Shext 中先 cd 到项目目录, 然后运行 jde start 命令, 功能同上。

3.2.3 补全

Vim-Sztool 自定义了 java 类型的 omni²的补全, 编辑 java 时, 用 `<ctrl-x><ctrl-o>` 来进行。

最佳实践: 安装 SuperTab 插件, 可以用 tab 键来做补全。Vim-Sztool 会做侦测, 如果已经安装 SuperTab 插件, 则对于 java 类型的文件, 默认的 tab 补全即是 omni 补全。

1. 可以补全包名, 类名, 名量名和对象的成员方法和字段
2. 方法和字段补全是 ignore case 的, 且可以引用通配符, 比如 aa.to*er, 可以匹配成 aa.toLower
3. 如果是大写开头的名字, 则自动在类路径中查找类名。比如即使没有 import FileReader 类, 打 FileRe 可以补全成 FileReader 类
4. 补全类名时, 不像 eclipse 那样能自动补上 import 语名。类补全完成需再 `<leader>ai` 一下

3.2.4 编译

保存 java 时自动编译, 如果有错误, 则会生成 quickfix 列表, 可以用:cn, :cp 命令转到前一个或后一个错误。对于在类路径下的资源文件, 如 xml 或 properties 文件, 在保存时也会自动复制到项目的编译输出目录。Jdext 默认的编译级别, 编译 encoding, 这些在 sztools.cfg 中设置。

想要针对对于当前的项目单独设置, 可以在项目根目录中增加.jde 文件, 用于配置编译级别和 encoding。文件内容样例参见 share/examples/jde.xml 文件。

²omni 补全为感知上下文的补全, 相当于 IDE 里的智能补全。vim 里有多种补全方式, 比如关键字补全。vim 自带了不少 omni 的补全功能, 比如 HTML,CSS 等, 具体请查看 vim 帮助。
:help new-omni-completion

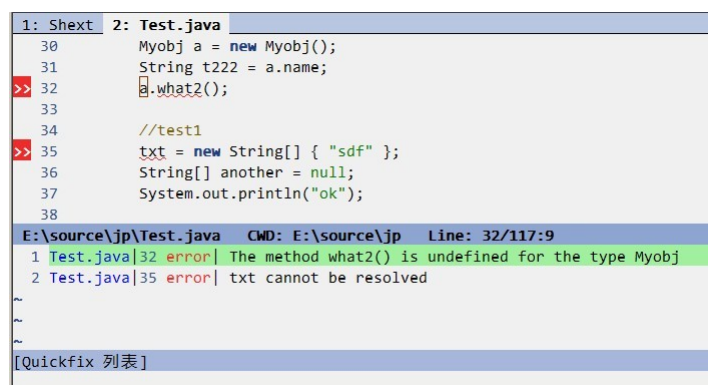


Figure 3.3: Jde 保存 java 文件时截图

最佳实践: 定义如下 map, 快速在 quickfix 中移动

map <C-n> :cn<cr>

map <C-p> :cp<cr>

3.2.5 运行

使用 `:Run` 或 `<leader>`, 来运行当前的 class。注意运行的程序中不能有从终端中读取的功能, 否则 java 程序将不能正常结束。此功能暂不支持加 arguments。如果需要测试程序参数, 可以在 Jdb 模式启动后用 `run` 命令进行。

3.2.6 其他功能

Import

对于代码中引用到的类, 没在 import 语句中声明的, 可以用 `:AutoImport` 一次性生成全部 import 语句。这个命令默认的 map 为 `<leader>ai`。如果 import 中引用的包在源码中无引用, 则这个命令也会自动清除该 import。由于此命令基于正则, 有时候会出不对的情况。

生成 getter,setter

用 visual 模式选中类中的属性 (可以多行), `<leader>gg` 来生成 getter, setter。在内部类中此命令生成的方法位置会不对。可以自己拷到正确的位置。

查看类信息

在编辑 Java 文件时，如果想快速查看某个类的信息，可以用 `<leader>dc` 功能。这个命令会打出类的所有公共方法，以及是否抽象方法等。(在继承实现某个类时可以用到)。

跳转到类定义

浏览 Java 源码时，当光标在某个方法或变量时，可以用 `<leader>gd` 来跳转到定义的位置。如果光标位置的调用是一个接口的方法，而这个接口又仅有一个实现时，可以用 `<leader>gi` 可以跳转到接口方法实现的位置。

定位本类成员

如果想要快速的跳转到本类的某一个方法，可以用 `<leader>go`。此功能会弹出一个类似 Locate 功能的界面，随着按键自动匹配成员名称。匹配到后按回车就可以跳到相应的成员所在的行上。如果想中止查找，可以用 `Esc` 键。

最佳实践: 推荐安装 Tagbar 插件。此插件功能类似著名的 TagList，但是对面向对象语言有更好的支持。此插件需要 ctags 程序支持，程序下载后也要放在 path 搜索目录中。在附录中有此插件的地址。插件安装后最好也自定义下 map 如: `nnoremap <silent> <F10> :TagbarToggle<cr>`

初始化项目结构

使用 `:InitProject` 在当前目录初始化一个 jdext 项目，如果编辑的文件是 eclipse 项目下的文件，则不需要再建项目。

3.3 Jdb

Jdb 是插件实现的 java 程序调试功能 (跟 jdk 里自带的 jdb 程序没关系)，需要在 Jdext 启动后再调用。

以 `:Jdb` 启动，启动后，默认会 split 两块 buffer，一块是 jdb 命令 buffer，一块是 jdb 命令的输出 buffer。jdb 命令的 buffer 也是普通 buffer，除了在回车时自动执行命令 (类似于 shext)，在 jdb 命令 buffer 运行 `help` 命令可以得到 jdb 命令的简要帮助。

jdb 的大部份调试功能是在 jdb 的 buffer 中用命令实现的，有个例外是加断点。只要启动了 jde 模式，在编辑 java 文件的时候，就可以用 `<leader>tb` 来

为当前行切换断点。

3.3.1 运行和附加

通过 `run` 命令运行 Class，后面为 `classname` 和参数，如 `classname` 省略，则运行当前编辑的 Class。

```
>run org.fake.test.Main -options args
```

`attach` 命令运行远程连接到 java 程序进行调试，要求已经运行的程序必须要 `jpda` 调试模式启动用此命令可以远程连接到比如 `tomcat` 之类的应用。此命令需要一个 `port` 参数为远程调试端口号，目前只能连接到本机启动的应用，其他 `host` 暂不支持。

```
>attach 8080
```

`disconnect` 命令用于断开 `attach` 上去的连接，`shutdown` 用于中止正在调试的程序，`hide` 用于隐藏 `jdb` 的窗口，`exit` 退出调试

```
>disconnect  
>shutdown  
>hide  
>exit
```

`runtomcat` 命令用于从 `jdb` 从运行 `tomcat` 应用。`tomcat` 的配置在 `sz-tools.cfg` 中进行，需要配置的三个属性是 `tomcat_version`，`tomcat_home`，`tomcat_jvmopts`，最后一个参数用来配置 `tomcat` 的 `jvm` 参数。配置中的目录，在 `windows` 下的“\”需要用“\\”来进行。比如

```
D:\\soft\\tomcat6
```

```
>runtomcat
```

3.3.2 检查变量

`print`(或 `eval`) 命令用于打印表达式的值。`reftype` 命令用于打印表达式的类型，`inspect` 命令打印表达式的各个成员的值。`locals` 打印函数内所有本地量的值，`fields` 用于打印当前对象的各个字段的值

```
>print expression  
>eval expression
```

```
>reftype expression  
>inspect expression  
>locals  
>fields
```

3.3.3 跟踪

单步进入，单步跳过，单步跳出，恢复。这个跟 eclipse 的功能差不多。快捷键也是一样的，分别是 <F5><F6><F7><F8>。注意这些快捷键也只能在 jdb 命令 buffer 使用，在其他 buffer 中是无效的。

```
>step_into  
>step_over  
>step_return  
>resume
```

3.3.4 线程和堆栈

threads 列出当前 jvm 的所有线程，thread 命令用于切换当前线程 (如果有线程是 suspend 状态的话)，frames 列出当前线程的所有调用栈。frame 命令用于切换当前的栈帧。(print, eval 等命令只能打印当前栈帧的变量值) breakpoints 列出当前 jvmf 所有的断点，bpa 添加条件断点，setvalue 改变 var 值

```
>threads  
>thread threadId  
>frames  
>frame n  
>breakpoints  
>bpa classname lineNum condition-expression  
>setvalue varname value-expression
```

3.3.5 调试实例

新建测试用空目录，启动 Jdext。然后新建个两个测试类 Test 和 Business，在 Test 类第 9 行用 <leader>tb 切换断点，接着启动 Jdb 时行调试。如下图

```

1: Shext 2: Business.java 3: SzToolView_Jdb
2 public class Test {
3     public static void main(String[] args) {
4         if (args.length < 1) {
5             System.out.println("arguments can't be empty");
6             return;
7         }
8         Business b = new Business();
9         String msg = b.isWolf(args[0]);
10        System.out.println(msg);
11    }
12 }

C:\project\test2\Test.java  CWD: C:\project\test2  Line: 9/12:32
1
~
~
JdeConsole
1 >run Test aa
2 >
~
~
~
Jdb  JdbStdOut
1 process terminated.

```

Figure 3.4: Jdb 运行 java 类

run 命令开始后，会又新增一个”JdeConsole”的 buffer 以打印 java 程序的输出。此时按 <F5> 或在 jdb 命令 buffer 中输入 step_info, 则跳转到 Business 类中。

```

1: Shext 2: Business.java 3: SzToolView_Jdb
R9 1 public class Business {
2
3     public String isWolf(String name) {
4         System.out.println("start rolling...");
5         if (name == null ) return "null string";
6         if (name.toUpperCase().equals("WOLF")) {
7             return "correct";
8         }
9         return "not a wolf";
10    }
11 }

C:\project\test2\Business.java  CWD: C:\project\test2  Line: 4/12:9
1 start rolling...
~
~
JdeConsole
1 >run Test aa
2 >print b
3 >print args[0]
4 >print name.toUpperCase().equals("WOLF")
~
~
~
Jdb  JdbStdOut
1 false

```

Figure 3.5: Jdb 运行 java 类

执行 print 语句，jdb 输出在”JdbStdout” buffer 中显示。此时可以继续调试或用 frame 切换到上一个栈帧去查看 Test 类中的变量的值。被调试的 java

类运行结束后，会在“JdbStdout”中显示“process terminated.”。调试完成后用 `exit` 命令退出。

3.4 ProjectTree

ProjectTree 从 NerdTree 借鉴了相当多的代码，基本上我写这个东西就是因为 NerdTree 有些地方还难以适合做 java 的项目树（比如源包 jar 包的显示），所以我用 python 又写了一套。

ProjectTree 启动时，会从当前目录向上查找 eclipse 项目文件，如果找不到，就以当前目录为树的根节点目录。如果找到项目配置 `.classpath` 文件，则会以此目录做为树的根节点目录，同时会在根节点上增加一个叫“Referenced Libraries”的虚拟目录节点，下面对应项目中引用的 source jar 包。对于没有关联源码的 jar 包，不会在节点中显示。一般至少会有一个节点“src.zip”，对应 jdk 自带的源码。

最佳实践：在用 `mvn eclipse:eclipse` 生成项目文件之前，可以配置 pom 文件下载关联的源码

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-eclipse-plugin</artifactId>
  <configuration>
    <downloadSources>true</downloadSources>
    <downloadJavadocs>true</downloadJavadocs>
  </configuration>
</plugin>
```

ProjectTree 的一些特点：

1. 按树结构显示 source jar
2. 打开的节点显示不同的颜色，一目了然看到哪些文件在编辑
3. 更方便的文件复制，移动，重命名功能
4. 标记多个节点，以进行多个节点的复制，删除等操作
5. 按目录递归关闭正在编辑的文件

3.4.1 打开和关闭节点

在 ProjectTree 中打开和关闭节点是最常用的功能。对于目录节点来说，打开和关闭指的是 `expand` 和 `collapse` 操作，而对于文件节点，打开指编辑，



Figure 3.6: ProjectTree 截图

关闭指关闭正在编辑的文件 buffer。(在 Nerdtree 中对于文件节点是没有关闭操作的)

正常情况下，打开文件节点后，节点会以不同的颜色显示。

按键	功能
?	打印 help 信息
<cr>	打开选中节点
o	打开选中节点
O	递归打开选中节点
t	在新 tab 页中打开选中节点
i	在新 window 中打开选中节点
go	打开选中节点，光标停留在 ProjectTree
r	刷新选中节点
x	关闭父节点
s	prompt to filter display node
z	关闭正在编辑的文件 buffer，如果当前是目录节点，则关闭目录下所有在编辑中的文件
Z	同上，但是强行关闭文件，即使文件已改动

s 命令用过滤节点，执行此命令后，会提示输入要显示的节点名称，以“,”分隔，不在此列表中的节点会在显示时被过滤掉。

3.4.2 移动和标记

由于 vim 本身有非常多的移动功能，在 ProjectTree 中扩展的移动不是很多。对于已经打开的文件，可以用”<”，”>” 在这些节点之前快速移动。

按键	功能
u	光标移动到父节点
m	标记当前节点
f	在当前节点中查找字符串
>	光标移动到下一个编辑中的文件节点
<	光标移动到上一个编辑中的文件节点

执行 f 命令时，会提示输入要在节点中查找的字符串。默认情况下，是进行简单匹配的。如果输入的字符串以”/” 开始和结束，则除去开头和结尾的”/” 的子串，会作为正则表达式，并以此在节点中查找。正则表达式的格式是 python 格式的。参见 python 的 re 模块的文档。

m 用来标记当前节点，标记出来的节点按不同颜色显示。在已经标记的节点上再次执行 m 可以取消标记。对于已经标记的多个节点，可以用 Dm 或 ym 来删除或复制。

另外在编辑文件的 buffer，可以用 ProjectTreeFind 命令来定位左侧的树结点。(此命令适用于左侧已经有 ProjectTree 打开的情况，如果还没有 ProjectTree，则可以用 :ProjectTree 打开，打开时也会自动定位节点)。在我自己的 vimrc 中，我做了如下映射

```
nnoremap <silent> <F9> :ProjectTree<cr>
nnoremap <silent> <F12> :ProjectTreeFind<cr>
```

3.4.3 文件操作

以下是一些删除剪切复制的功能，注意，这些操作是同步的，即在这些粘贴删除实际完成前，页面将无响应。所以不要用使用这里的功能来操作大的文件和目录，以避免可能的 vim 崩溃导致数据丢失。

按键	功能
DD	删除选中目录或文件
Dm	删除标记的目录或文件
A	新增目录或文件 (如文件名以/结尾, 则为目录)
ya	复制节点路径
cc	重命名当前节点
yy	复制当前文件或目录
ym	复制标记的文件或目录
dd	剪切当前文件或目录
p	粘贴复制或剪切的文件
C	更改树的显示根目录
B	返回老的根节点
U	更改树的根节点为父目录
QQ	关闭 ProjectTree

Table 3.1: 删除剪切复制等

ProjectTree 不同于 NerdTree 树, 它是单实例的。关闭 ProjectTree 树, 不会删除它的节点信息, 当重新打开时, 原来打开的节点或关闭的节点等都保持原来的状态。有时候会想要完全关闭 ProjectTree(比如当前目录转到其他项目文件夹之后), 这时可以先用 QQ 命令关闭 ProjectTree, 再重新打开时, 就会以当前目录以树结点的根目录了。

3.5 Locate

Locate 功能有点像 command-T 插件, 用来快速定位文件的。但是不同的是, 这个功能需要先对文件夹进行索引, 索引后的文件名信息存在 sqlite 的数据库中。这样无论你的当前目录是在哪里, 都可以快速按文件名定位到已索引的目录中的文件。相当于是 eclipse 中的 Open Resource 功能。

3.5.1 索引建立删除

索引管理索引需要在 Shext 中用 locatedb 命令管理 (建索引时需要先 cd 到需要索引的目录)

- locatedb add name : 建立索引, 名称为 name, 索引当前文件夹的内容
- locatedb remove name : 删除索引
- locatedb refresh name : 刷新索引
- locatedb list : 列出已建立的索引

建立索引时，并不是当前文件夹下的所有文件都会索引的。一些文件是编译输出的，或者是临时文件的话，则默认不索引。比如.pyc 文件，.class 文件等默认不索引。具体哪些文件不索引，可以通过 ~/.sztools/sztools.cfg 文件中的 default_exclude_pattern 属性值来设置。

3.5.2 索引更新

当 gvim 启动后，会有一个独立的 Agent 进程启动，此进程会自动监视被索引目录的文件的新建和删除，并自动更新索引。如果文件在 Agent 进程未启动条件下新建和删除，可以手动执行 locatedb refresh 命令更新

目录监视功能是系统级的，即使在 vim 外新增删除文件和目录也会更新索引。想知道相关实现的可以看 <http://jnotify.sourceforge.net/>

3.5.3 调用

以 <leader>lw 来启动 locate 模式，启动后，底下会有有一个小的 buffer 用来显示匹配的文件名。所有输入的字符都显示在 command line 上面，并显示相应的匹配内容。匹配不区分大小写，并支持通配符“*”。在匹配列表出来后，可用的功能如下

- <Esc>：退出 locate 模式
- <CR>：打开 (编辑) 当前光标所在文件
- <BS>：光标回退
- <C-j>：光标下移
- <C-k>：光标上移
- <C-v>：复制剪贴板内容
- <C-b>：在新 buffer 中打开文件
- <C-t>：在新 tab 中打开文件

3.6 Dbext

这个是 vim 官方站点上的 dbext 的模仿，因为之前我用原来的 dbext 的时候，感觉输出不太友好 (对不齐，没有表格线等，不知道新的版本有没有弄好)。然后我用 python 自己实现了一个简单的。就是用 pyodbc，或其他数据库驱动来执行 sql，把输出结果集用表格线排一下就得了。因为没有考虑太多，所以如果你查询大数据量的结果集，比如

```
select * from user
```

在 user 表中有几十万几上的数据的话，很可能把 vim 搞死（取决于机器性能等原因）。

由于实际执行 sql 的是 python 的数据库驱动模块，所以如果想执行各种库上的 SQL 就要装相应的 python 数据库驱动模块。一般连 Ms Sql server 就用 pyodbc，连 oracle 的库就用 cx_oracle。另外在程序中还写了 mysql 的驱动模块 MySQLdb 的支持，不过我自己用的不多，估计会有问题。

3.6.1 配置

在使用前先确保在 ~/.sztools 目录下的 db.conf 中配置了想要连接的数据库。配置文件格式为

```
servertype="mssql", host="127.0.0.1", user="sa", password="test"
servertype="mssql", host="127.0.0.1", user="sa", password="test"
```

数据库名称不需要配置，在启动 Dbext 后用 use databasename 来切换。servertype 值目前支持 oracle,mssql,mysql,sqlite。对于 sqlite，除了 servertype 外，另外只要一个 file 属性指出 sqlite 库的文件位置就行了。

请在空白的 tab 页上执行 :Dbext，执行完后，像 Shext 一样会 split 成两个 buffer，上侧 buffer 编辑 sql 和执行，下侧 buffer 输出结果。在 sql 编辑 buffer 可以用 <c-x><c-o> 来补全，可以补全表名，字段名...（可以用通配符）

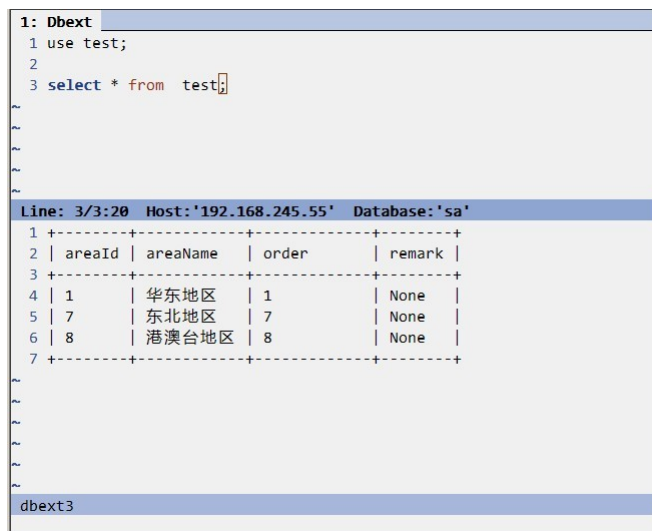


Figure 3.7: Dbext 截图

3.6.2 使用

执行查询

- visual 模式，选中 SQL 文本用 `”` 来执行，如果选中语句中有”；”，默认会隔分多条来执行
- normal 模式和 insert 模式，`”` 执行当前行所在的 SQL 语句
- visual 模式，`,gs` 把选中文本作为单条 SQL 执行，即使包含”；”也不分隔

查询元数据

- `,ld` 用来列出所有的数据库
- `,la` 用来列出当前数据库的所有表
- `,dt` describe table, 列出表的字段。执行前先要 visual 选中表的名称
- `,lt` 显示名称中包含当前选中文本的所有表。执行前先要 visual 选中文本

3.7 杂项功能

- `<leader>te` : 将选中文档用表格框起来
- `<leader>rc` : 删光 java 文件的注释
- `:Example name :split` 出一个 buffer，显示示例文件。示例参见安装目录下 `share/examples`

第 4 章 附录

以下是一些你可能会想要看一下的东西:

http://www.vim.org/scripts/script.php?script_id=1213 Vjde, Vim 插件

<http://eclim.org/> Eclim, vim 插件, 使用 eclipse 作为服务端

<http://vrapper.sourceforge.net/home/> vrapper, clipse 插件, 免费

<http://www.viplugin.com/> viplugin, eclipse 插件, 收费

推荐的 vim 插件

supertab http://vim.sourceforge.net/scripts/script.php?script_id=1643

tagbar http://vim.sourceforge.net/scripts/script.php?script_id=3465

pathogen http://vim.sourceforge.net/scripts/script.php?script_id=2332

easymotion http://vim.sourceforge.net/scripts/script.php?script_id=3526

vimwiki http://vim.sourceforge.net/scripts/script.php?script_id=2226

bufexplorer http://vim.sourceforge.net/scripts/script.php?script_id=42

第 5 章 已知问题

在 windows 下, 默认安装的 gvim73+python2.7.2 import pyodbc 时会出现 import error

默认不支持 64 位系统, 需要替换 64 位的 swt.jar

python 的默认编码要和 vim 的默认编码保持一致, 否则会出现乱码可以分别通过 python 安装目录下的 site-packages/sitecustomize.py 和 vimrc 中的 encoding 选项来设置

第 6 章 后记

本文档用 vim+latex 写成。感谢这两个伟大的工具

此插件虽然可以预见的不会有很多使用者，不过如果能收到大家的反馈，不论是 bug 提交，需求建议，或是仅仅告诉我此插件对你有帮助 (或插件写得太烂)，我都会很感谢大家。我的联系方式已写在此文档的第一页:)。