26
Декабрь 2019 г.

Написание плагина Vim

Лукаш *Ян Нимьер*

Домашняя страница
GitHub Твиттер
Электронная почта
hauleth на Freenode

Хотя существует множество «руководств» по написанию плагинов в Vim, я надеюсь, что это будет немного отличаться от того, что есть, потому что оно не будет посвящено написанию плагина *как такового* . Если вы хотите найти информацию об этом, то вам следует заглянуть в `:h write-plugin` . Я хочу, чтобы эта статья была о том, как плагины оживают, используя мой собственный опыт написания `vim-backscratch` в качестве примера.

Проблема

Все плагины должны начинаться с проблемы. Если проблемы нет, то не должно быть и кода, поскольку нет лучшего кода, чем его отсутствие . В этом случае моя проблема была довольно тривиальной: мне нужен был временный буфер, который позволял бы мне выполнять быстрые правки и просматривать SQL-запросы, оптимизируя их (и запускать их оттуда с помощью dadbod Тима Поупа).

«Простое очевидное решение»

Теперь, когда мы определили проблему, нам нужно попробовать первое возможное решение. В нашем случае это открытие

нового буфера в новом окне, его редактирование и закрытие, когда он больше не нужен. В Vim это просто:

```
:new
" Edits
:bd!
```

К сожалению, здесь есть ряд проблем:

- если мы забыли закрыть этот буфер, то он будет висеть там бесконечно,
- запуск `:bd!` в неправильном буфере может иметь неприятные последствия,
- этот буфер все еще указан в `:ls`, что не нужно (так как он временный).

Систематическое решение в Vim

К счастью, в Vim есть решения всех наших проблем:

- раздел «scratch» в `:h special-buffers`, который решает первые две проблемы,
- `:h unlisted-buffer`, что решает третью проблему.

Итак, теперь наше решение выглядит так:

```
:new
:setlocal nobuflisted buftype=nofile bufhidden=
" Edits
:bd
```

Однако это длинная цепочка команд для написания. Конечно, мы могли бы объединить первые две в одну:

```
:new ++nobuflisted ++buftype=nofile ++bufhidden
```

Но на самом деле это ничего не сокращает.

Создать команду

К счастью, мы можем создавать собственные команды в Vim, поэтому мы можем сократить это до одной, легко запоминающейся команды:

```
command! Scratch new ++nobuflisted ++buftype=r
```

Для большей гибкости я предпочитаю, чтобы это было так:

```
command! Scratchify setlocal nobuflisted bufty
command! Scratch new +Scratchify
```

Мы также можем добавить ряд новых команд, которые позволят нам лучше контролировать местоположение нашего нового окна:

```
command! VScratch vnew +Scratchify
command! TScratch tabnew +Scratchify
```

Эти команды откроют новый буфер ввода в новом вертикальном окне и новый буфер

ввода на новой вкладке соответственно.

Сделайте его более «жизнерадостным» гражданином

Хотя наши команды `:Scratch`, `:VScratch`, и `:TScratch` хороши, они все еще недостаточно гибки. В Vim мы можем использовать модификаторы, например, `:h :aboveleft` чтобы точно определить, где мы хотим, чтобы появлялись новые окна, а наши текущие команды не учитывают этого. Чтобы исправить эту проблему, мы можем просто объединить все команды в одну:

```
command! Scratch <mods>new +Scratchify
```

И мы можем удалить `:VScratch` и `:TScratch` так как теперь это можно сделать с помощью `:vert Scratch` и `:tab Scratch` (конечно, вы можете оставить их, если хотите, я просто хотел, чтобы UX был минимальным).

Сделай это мощным

Это было у меня `$MYVIMRC` некоторое время в описанной выше форме, пока я не обнаружил фрагмент Ромена Лафуркада, который предоставлял одну дополнительную функцию: он позволял открывать буфер с выводом Vim или команды оболочки. Моей

первой мыслью было - эй, я знаю это, но я знаю, что могу сделать это лучше! Итак, мы можем написать простую функцию VimL (которая в основном скопирована из фрагмента Ромена Лафуркада с несколькими улучшениями):

```
function! s:scratch(mods, cmd) abort
    if a:cmd is# ''
        let l:output = []
    elseif a:cmd[0] is# '!'
        let l:cmd = a:cmd =~' %' ? substitute(
        let l:output = systemlist(matchstr(l:c
    else
        let l:output = split(execute(a:cmd), '
    endif

    execute a:mods . ' new'
    Scratchify
    call setline(1, l:output)
endfunction

command! Scratchify setlocal nobuflisted nosw
command! -nargs=1 -complete=command Scratch (
```

Основные отличия:

- особый случай для пустой команды, она просто откроет пустой буфер,
- использование `is#` вместо `==`,
- использование `:h execute()` вместо `:redir`.

Так как он достаточно самостоятелен и (будем честны) слишком специфичен для нас, `$MYVIMRC` мы можем извлечь его в его собственное местоположение в `plugin/scratch.vim`, но чтобы сделать

это правильно, нам понадобится еще одна дополнительная вещь — команда, предотвращающая двойную загрузку скрипта:

```
if exists('g:loaded_scratch')
    finish
endif
let g:loaded_scratch = 1

function! s:scratch(mods, cmd) abort
    if a:cmd is# ''
        let l:output = []
    elseif a:cmd[0] is# '!'
        let l:cmd = a:cmd =~' %' ? substitute(
        let l:output = systemlist(matchstr(l:c
    else
        let l:output = split(execute(a:cmd), '
    endif

    execute a:mods . ' new'
    Scratchify
    call setline(1, l:output)
endfunction

command! Scratchify setlocal nobuflisted noswa
command! -nargs=1 -complete=command Scratch (
```

Смело идти…

Теперь моя идея была, эй, я использую макросы Vim время от времени, и это просто простые списки нажатий клавиш, сохраненные в регистрах Vim. Может быть, было бы неплохо иметь доступ к этому также в нашей команде. Поэтому мы просто добавим новое условие, которое проверяет, `a:cmd` начинается ли со

`@` знака и имеет ли длину два. Если да, то устанавливаем `l:output` сплайсированное содержимое регистра:

```
function! s:scratch(mods, cmd) abort
    if a:cmd is# ''
        let l:output = ''
    elseif a:cmd[0] is# '@'
        if strlen(a:cmd) is# 2
            let l:output = getreg(a:cmd[1], 1
        else
            throw 'Invalid register'
        endif
    elseif a:cmd[0] is# '!'
        let l:cmd = a:cmd =~' %' ? substitute(
        let l:output = systemlist(matchstr(l:c
    else
        let l:output = split(execute(a:cmd), '
    endif

    execute a:mods . ' new'
    Scratchify
    call setline(1, l:output)
endfunction
```

Это дает нам довольно мощное решение, с помощью которого мы можем `:Scratch @a` открыть новый буфер с содержимым register `A`, отредактировать его и восстановить с помощью `"ayy`.

Плагинизировать

Теперь было бы стыдно оставить такой полезный инструмент себе, так что

давайте поделимся им с большим миром. В этом случае нам понадобится:

- правильная структура проекта,
- документация,
- хорошее броское название.

Помощь по первым двум темам можно найти в `:h write-plugin` и `:h write-local-help` или в любом из миллиона руководств в Интернете.

Найти хорошее название — это то, с чем я не могу вам помочь. Я выбрал `vim-backscratch`, потому что мне нравятся почесывания спины (всем они нравятся) и, как приятное совпадение, потому что оно содержит слово «scratch».

Краткое содержание

Создавать плагины для Vim легко, но не каждая функциональность должна быть плагином с первого дня. Начните с простого и малого. Если что-то можно сделать с помощью простой команды/отображения, то это следует сделать с помощью простой команды/отображения в первую очередь. Если вы найдете свое решение действительно полезным, тогда и только тогда вам следует подумать о том, чтобы превратить его в плагин. Весь процесс, описанный в этой статье, не произошел за неделю или две. Мне потребовалось около года, чтобы достичь

шага *Make it a more “vimmy” citizen* , когда я услышал о скрипте romainl в IRC. Мне больше ничего не нужно было, так что не торопитесь.

Дополнительные советы:

- сделайте его маленьким, большие плагины потребуют много обслуживания, маленькие плагины гораздо проще в обслуживании,
- если что-то можно сделать с помощью команды, то это следует сделать командой, не навязывайте свои сопоставления пользователям.

 GitHub RSS Твиттер

0 Выберите издание