如何让 Doom Emacs 高亮所有变量？

with or without tree-sitter in Doom Emacs

this-is-a-computer

日期：December 17, 2022

目录

1 太长不看

1.1 使用 lsp 进行高亮

如果你只是想让 C/C++/Rust 拥有不亚于 Vscode 的补全体验

首先要安装启用 lsp mode，然后安装对应语言的 lsp 服务器。支持 lsp 的语言都可以通过开启 `lsp-semantic-token-mode` 来使用 lsp 服务器进行语义级别的高亮，这就和 vscode 中使用 clangd 进行高亮一样，所以启用他就好了，这主要针对 C/C++/rust，会拥有比 `tree-sitter` 好许多的体验

如果想要自动开启，根据 lsp-mode 文档，在配置文件中设置这两个变量即可 activate=false

```
(setq lsp-semantic-tokens-enable t) ; 启用 lsp 语义高亮
(setq lsp-semantic-tokens-honor-refresh-requests t) ; 自动刷新
```

如果是别的语言，lsp 的高亮没那么强，可能就得考虑使用 tree-sitter 了。

1.2 使用 tree-sitter

如果你还是想用 `tree-sitter` 进行高亮,或者只能用 `tree-sitter`（比如 go 似乎就无法用 `lsp-semantic-token-mode` 达到更好的高亮，而且 `tree-sitter` 本身就是一个很棒的工具），可以这样开启，非常的容易

在 `~/.doom.d/init.el` 中，取消 tree-sitter 的注释。

然后在 `~/.doom.d/config.el` 中，添加

activate=false

```
(global-tree-sitter-mode)
(add-hook 'tree-sitter-after-on-hook #'tree-sitter-hl-mode)
```

第一行可以全局启用 tree-sitter，但是并不会真正进行高亮，第二行可以让他对任何 major mode 都生效。参考emacs tree-sitter 文档。

doom emacs 是默认使用 `tree-sitter-langs` 提供的“高亮策略”的，而这个高亮策略下，变量引用使用的是 buffer 的默认颜色，所以看起来就和没有高亮一样。想要解决这个问题，可以尝试下面两种方法（我个人觉得第二种效果比较好）

1.2.1 一个不怎么好的解决方法

在 `~/.doom.d/config.el` 中添加

activate=false

```
(tree-sitter-hl-add-patterns 'c ; change c to your language's name
  [(identifier) @variable])
(tree-sitter-hl-add-patterns 'cpp
  [(identifier) @variable])
(tree-sitter-hl-add-patterns 'rust
  [(identifier) @variable])
```

1.2.2 一个比较 Dirty 但是相对比较好的解决方法

在 `~/.doom.d/config.el` 中添加

activate=false

```
(add-hook 'prog-mode-hook (lambda ()
    ;; set the foreground color
    (setq buffer-face-mode-face '(:foreground "DeepSkyBlue"))
    (buffer-face-mode)))
```

这种方法是通过设置 buffer 的字体默认颜色，把 `DeepSkyBlue` 换成你想要的颜色就可以了。这样原来的，看起来没高亮的地方，看起来就都高亮了。

2 Doom Emacs 不是本来就有变量高亮吗？

doom emacs 有 tree-sitter 的模块，也支持了许多语言的高亮，可以起到不错的代码高亮效果，我的 doom emacs 效果如下

```
AstNodePtr Parser::RelExpL() {
    LexerIterator iter_back = token_iter_;
    // RelExpL -> ('<' | '>' | '<=' | '>=') AddExp RelExpL | e
    auto rel_exp_l =
        AstNodePool::get(SyEbnfType::END_OF_ENUM, (token_iter_)->line_);
    if ((token_iter_)->getAstType() != SyAstType::LNE &&
        (token_iter_)->getAstType() != SyAstType::GNE &&
        (token_iter_)->getAstType() != SyAstType::LE &&
        (token_iter_)->getAstType() != SyAstType::GE) {
        // no need to reset the token_iter_
        // this is not a failure
        // just return e
        return AstNodePool::get(SyEbnfType::E, 0);
    }
    rel_exp_l->a_ = *(token_iter_);  // link the '<' or '>' or '<=' or '>='
    ++token_iter_;
    auto add_exp = AddExp();
    if (add_exp == nullptr) {
```

但是有一个小问题，可以看到每个变量在引用处有时不会高亮

3 tree-sitter 这么弱？

tree-sitter 在 Doom Emacs 中，大抵是这样的工作逻辑：

1. tree-sitter 工作时，都是使用一个由 scheme 写成的描述文件进行 parse，而在进行高亮的时候，他会去搜索 `highlights.scm` 这个 *`.scm`*，*就好像 flex 的 `.l`*
2. Doom Emacs 为每个语言提供的 `highlights.scm` 是由 tree-sitter-langs 整理的
3. 我们可以发现这个仓库中，至少 c 语言的 `highlights.scm` 与 tree-sitter-c 仓库中的 highlights.scm 是不一样的
4. 如果使用 tree-sitter-c 仓库（也就是 `tree-sitter-langs` 的上游仓库之一）的 `highlights.scm`，可以起到（我认为）更不错的效果

也就是说，tree-sitter 是能分辨变量的（废话，一个基于 parse 的语法高亮器怎么可能不能分辨！）。然后看了一眼 `tree-sitter-langs` 的文档，才发现原来是这样的

> Highlighting query patterns for a language are in the file queries/<lang>/highlights.scm. Most of them are intentionally different from those from upstream repositories, which are more geared towards GitHub's use cases. We try to be more consistent with Emacs's existing conventions. (For some languages, this is WIP, so their patterns may look similar to upstream's.)

看来这是 `tree-sitter-langs` 故意搞的。让对变量的读和写拥有不一样的高亮

所以我们就知道解决方案了——直接改掉 tree-sitter 的高亮策略就行了

4 如何配置 emacs tree-sitter？

那么问题就变成了配置 `tree-sitter` 使用的 `highlight.scm`，让他对变量的引用和定义用同样方式高亮。尴尬的是我看不太懂他的文档上的配置方法，也不想为了配个语法高亮学一个 parser generator 怎么写

所以我选择 diff `tree-sitter-c` 和 `tree-sitter-langs` 中的 `highlight.scm` 。发现主要不同就是他多了一行 `(identifier) @variable]` ，所以我的尝试，是这样的：activate=false

```
(tree-sitter-hl-add-patterns 'c
  [(identifier) @variable])
(tree-sitter-hl-add-patterns 'cpp
  [(identifier) @variable])
(tree-sitter-hl-add-patterns 'rust
  [(identifier) @variable])
```

这样的的确可以让变量的所有引用都高亮，效果是这样的

但是还是不够好：在这种情况下非成员函数变成了和变量一样的颜色，只是加粗了。

但是我已经满意了，因为我实在不会了！

4.1 不够五光十色

实际用了几天后我发现，其实我并不满意，变量和函数颜色一样，辨识度很低，看起来满屏就是一个颜色，和没高亮一样，所以为了解决这个问题，我直接 patch 了 `tree-sitter-langs` 的 `highlights.scm` ，强制使用上游仓库的高亮策略。即修改 `~/.emacs.d/.local/straight/repos/tree-sitter-langs/queries/<language name>/highlight.scm` 。比如 C 语言，就直接改成 tree-sitter-c 的 highlights.scm。

5 太不优雅了

直接 patch 掉人家仓库的代码，为 Doom 的未来更新留下了隐患，而且这么做非常不优雅。另外，让变量的读和写有不一样的高亮，这也很酷啊，所以我的想法就转变了：与其修改 `tree-sitter` ，不如先看看如

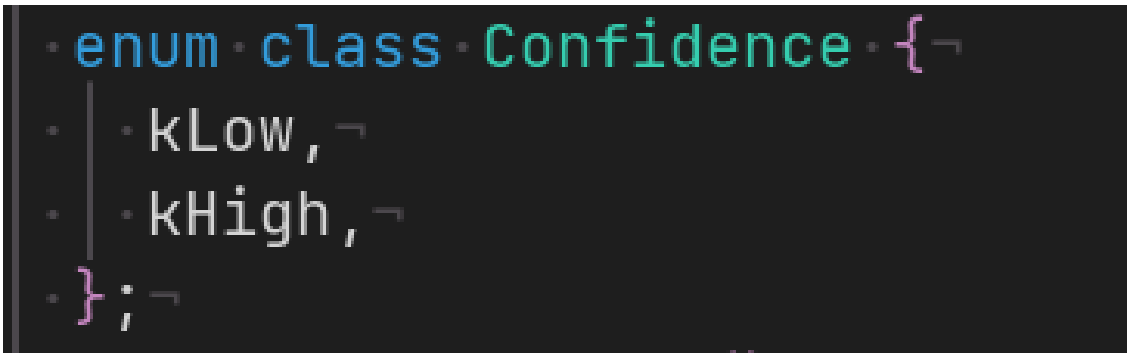
何配置 emacs 的 font-lock 这个 minor-mode，毕竟 tree-sitter 是继承他的嘛！看了一会，实话说没怎么看明白。。

不过我又想到了一个更 dirty 的解决方法，反正我主要不爽的是有白色，而没有被高亮的也只有变量引用了，所以我直接把 buffer 的字体默认颜色改掉不就好了。要修改 buffer 的字体颜色，可以启用 buffer-face-mode 然后设置 buffer-face-mode-face ，添加 '(:foreground "color name") ，所以我在 config.el 中添加了这样的配置，展示配置成深天蓝色（DeepSkyBlue）activate=false

```
(add-hook 'prog-mode-hook (lambda ()
    ;; set the foreground color
    (setq buffer-face-mode-face '(:foreground "DeepSkyBlue"))
    (buffer-face-mode)))
```

直接设置前景色，这样高亮的效果就是这样了

颜色就出来了。这样子剩余的问题就是由于我是 vscode 迁移过来的，看到这个颜色，很像 vscode 中对枚举的高亮，于是我看了一眼 tree-sitter 对枚举的高亮，关掉 buffer-face-mode



好家伙，根本就是没有高亮。。所以我准备就保留这个配置了，当然之后可以考虑换一个更合适的颜色（现在我看着就感觉满屏全是枚举），甚至直接换成和变量定义这样一样的颜色也没关系。

最后检查一下别的语言，rust 下

没什么问题，rust 的高亮规则本来就把枚举当成类型。也不再存在白色啦！

好了，这个解决方案更不优雅，但是我很满意

6 凭啥 lsp 不能帮我高亮？

没错，我过去在使用 vscode 的时候，C/C++ 的高亮都是 clangd 帮我做的，在 emacs 中当然也可以：`LSP Mode` 提供了 `lsp-semantic-token-mode`，这个模式对 C/C++/rust 的提升看起来非常巨大。

不过现在看起来，如果想要使用 lsp 提供高亮，首先性能会下降（个人感觉会卡一点），另外对于比较复杂的 C++ 项目，也需要等 clangd index 完了才能开始高亮，相比之下 `tree-sitter` 就会快很多。

但是用 lsp 的高亮，效果真的很好啊！一开始我还没搞明白怎么自动开启，直接在 `prog-mode-hook` 上加一行 `lsp-semantic-token-mode` 并不行，他似乎并不会自动刷新的样子。事实证明还是要多看文档，里面说到，添加这两行即可 activate=false

```
(setq lsp-semantic-tokens-enable t)
```

```
(setq lsp-semantic-tokens-honor-refresh-requests t)
```

另外,参考这个 issue,其实确实会碰到一点小坑——对定义变量时的高亮出错了,我在 `custom-set-faces` 中加入了这个

activate=false

```
'(lsp-face-semhl-interface
  ((t (:inherit font-lock-variable-name-face))))
```

（似乎 doom emacs 中并不应该使用 `custom-set-faces`，但是我先暂时用着了）

我最近折腾了很久工具，代码反而是没写几行，这样不是好。我决定就暂时使用 `tree-sitter` 加我的 dirty solution 了。