

1 背景

本章ではシンタックスハイライトについて説明し、一般的なテキストエディタにおけるシンタックスハイライト機能について説明する。

1.1 シンタックスハイライト

シンタックスハイライトとは、テキストエディタで、編集するソースコードが見やすくなるよう、色をつける機能である。予約語や変数名が色で区別できると、見てプログラムの構造が分かりやすくなる。Eclipse, Emacs, Vim, Nano, TextMate, 秀丸, TextMate, Gedit といった一般的なプログラミングに用いられるテキストエディタにはシンタックスハイライトが実装されている。

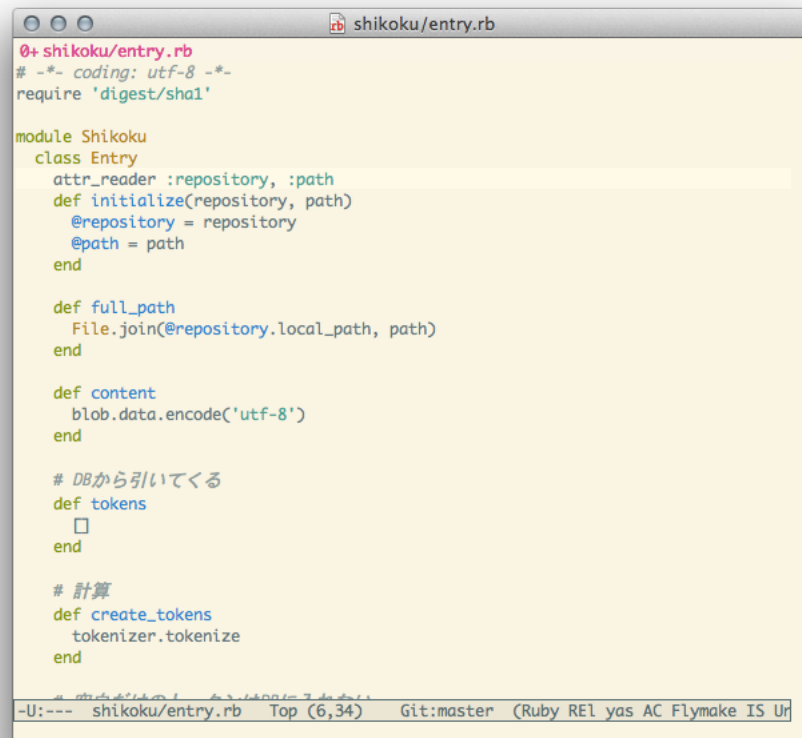
下にシンタックスハイライト前後でのソースコードの見た目の変化を示す。ハイライト前は 1 色でただのテキストだったのが、色々な色がついて、ソースコードが見やすくなっている。

シンタックスハイライトは、色だけを変更し、フォントの種類やフォントサイズを変えないのが一般的である。Emacs の org-mode や ylatex-mode などのように、自然言語を編集する場合では、見出しを大きくするなど、フォントサイズを変えるものもある。同じキーワードについては同じ色を割り当てるのが一般的である。例えば、if というキーワードが青い場合、他の場所の if も青いのが一般的である。ただし、if というキーワードであっても、青くない場合もある。文字列リテラルの中の if という字は文字列リテラルの一部であって if というキーワードではないため、文字列リテラルの色がつく。

シンタックスハイライトには見やすい色の組を選ぶ必要がある。背景が白で、薄い色をつかうと、コントラストが低くなり、ソースコードを読みにくい。また、意味の違う言葉に同じ色や似た色を割り当てると見間違いが生じる可能性があるので、なるべく似た色は使わず、色を分散させるのが一般的である。色は、予めシンタックスハイライトの設定で決められている場合や、あらかじめ用意された色の組み合わせから選べる場合や、個別に設定できる場合がある。

ハイライトのルールはハイライト対象の言語別に用意する必要がある。Lisp のシンタックスハイライトを C のプログラムに適用すると適切に色付けされない。Lisp の予約語は C では予約語ではないので、誤った色が付き、プログラムの構造が分かりにくくなる。

シンタックスハイライトはエディタ別に実装されていて、エディタ間のシンタックスハイライトは振舞いは似ているが互換性がない。新しいエディタが生まれた際には、世に存在するプログラミング言語の数だけシンタックスハイライトをサポートする必要がある。逆に、



```
0+ shikoku/entry.rb
# -*- coding: utf-8 -*-
require 'digest/sha1'

module Shikoku
  class Entry
    attr_reader :repository, :path
    def initialize(repository, path)
      @repository = repository
      @path = path
    end

    def full_path
      File.join(@repository.local_path, path)
    end

    def content
      blob.data.encode('utf-8')
    end

    # DBから引いてくる
    def tokens
      []
    end

    # 計算
    def create_tokens
      tokenizer.tokenize
    end
  end
end
```

図 1: シンタックスハイライト適用前のソースコード

新しいプログラミング言語が生まれた際には世に存在するエディタの数だけシンタックスハイライトを実装する必要がある。


1.2 nano におけるシンタックスハイライト

nano は、.nanorc という設定ファイルを編集することでエディタの振舞いを変更できる。nano 本体の機能として、シンタックスハイライト機能がある。

Python のシンタックスハイライトを有効にするには、nanorc から、以下のように、python.nanorc を読み込む。

```
## Python
include "/usr/share/nano/python.nanorc"
```

python.nanorc の内容を示す。

A screenshot of a code editor window titled 'shikoku/entry.rb'. The code is written in Ruby and features syntax highlighting. The code defines a module 'Shikoku' containing a class 'Entry'. The class has an 'attr_reader' for ':repository' and ':path', an 'initialize' method, a 'full_path' method, a 'content' method, a 'tokens' method (commented as '# DBから引いてくる'), and a 'create_tokens' method (commented as '# 計算'). The status bar at the bottom shows '-U:--- shikoku/entry.rb Top (6,34) Git:master (Ruby RE1 yas AC Flymake IS Ur)'.

```
0+ shikoku/entry.rb
# -*- coding: utf-8 -*-
require 'digest/sha1'

module Shikoku
  class Entry
    attr_reader :repository, :path
    def initialize(repository, path)
      @repository = repository
      @path = path
    end

    def full_path
      File.join(@repository.local_path, path)
    end

    def content
      blob.data.encode('utf-8')
    end

    # DBから引いてくる
    def tokens
      []
    end

    # 計算
    def create_tokens
      tokenizer.tokenize
    end
  end
end
```

図 2: シンタックスハilight適用後のソースコード

```
## Here is an example for Python.
##
syntax "python" "\.py$"
header "^#!.*python[-0-9._]*"
icolor brightblue "def [0-9A-Z_]+"
color brightcyan "\<(and|as|assert|break|class|continue|def|del|elif|else|except|exec|finally|
color brightgreen "['][^']*[^\\']" "[']{3}.*[^\\']" "[']{3}"
color brightgreen "["[^"]*[^\\"]" "["{3}.*[^\\"]" "["{3}"
color brightgreen start="\"\"\"[^"]" end="\"\"\"" start="'''[^']" end="'''"
color brightred "#.*$"
```

以下のような設定が書かれている .

- ファイル名が.py で終わるときに Python のシンタックスハイライトを有効にする
- Shebang に python が含まれるとき Python のシンタックスハイライトを有効にする
- def [0-9A-Z_]+ にマッチする部分は明い青色にする
- and または as または assert または break など、予約語にマッチするときは明い青緑にする

などである。

nano のシンタックスハイライトでは、ハイライトの設定で色まで決められていて、ユーザーは色を変更することができない。上の例では、Python の予約語は常に明い青緑で表示される。

1.3 Vim におけるシンタックスハイライト

Vim では syntax コマンドでシンタックスハイライトを有効にできる。

```
:syntax enable
```

nano では、シンタックスハイライトのルールに色まで決めていたが、vim では、シンタックスハイライトのルールでは、トークンのクラスまでを決めている。トークンのクラスだけを正規表現で切り出し、そのクラスに対応する色は別の設定で決める。これによって、正しい設定が用意されていれば、編集する言語によらず、statement は緑、といったように、一貫性がある。また、Vim スクリプトというスクリプト言語を実行することができ、プログラムによって色付けできる。Vim のシンタックスハイライトは Nano よりも柔軟である。

以下の例では、python_highlight_all という変数を true にしているときには、ハイライトするルールを増やす、ということが書かれている。

```
if exists("python_highlight_all")
    let python_highlight_numbers = 1
    let python_highlight_builtins = 1
    let python_highlight_exceptions = 1
    let python_highlight_space_errors = 1
endif

if exists("python_highlight_numbers")
    " numbers (including longs and complex)
    syn match pythonNumber "\<0x\x\+[Ll]\>"
```

```

syn match pythonNumber "\<\d\+[Lljj]\=\>"
syn match pythonNumber "\.\d\+\([eE][+-]\=\d\+\)\=[jJ]\=\>"
syn match pythonNumber "\<\d\+\.\([eE][+-]\=\d\+\)\=[jJ]\=\>"
syn match pythonNumber "\<\d\+\.\d\+\([eE][+-]\=\d\+\)\=[jJ]\=\>"
endif

```

どのクラスがどの色か，という対応は，カラースキームを選ぶことで決められる．下にカラースキームの定義を示す．Statement というクラスは，Vim が GUI で起動していて，フルカラー表示できるときは文字色は #dfdf6f，これは RGB で，R = 223，G = 223，B = 111，という意味である，それ以外で，ターミナル内に起動している場合は，yellow という色が割り当てられる．カラースキームには，文字のクラスごとに，文字色，背景色，文字の装飾 (太字かどうか)，が定義されている．

```

hi Identifier    guifg=#dfdf6f      guibg=NONE      gui=NONE
                  \  ctermfg=yellow   ctermbg=NONE    cterm=NONE

hi Statement     guifg=#6fef7f      guibg=NONE      gui=bold
                  \  ctermfg=green    ctermbg=NONE    cterm=bold

hi PreProc       guifg=#afafaf      guibg=NONE      gui=NONE
                  \  ctermfg=darkgreen ctermbg=NONE    cterm=NONE

hi Type          guifg=#9f9fef      guibg=NONE      gui=bold
                  \  ctermfg=lightblue ctermbg=NONE    cterm=bold

```

nu42dark.vim

1.4 Emacs のシンタックスハイライト

Emacs では，オーバーレイという仕組みがあり，テキストにオーバーレイを設定することで，色やフォントなどの見た目を変更できる．fontlock-mode というシンタックスハイライトをするためのメジャーモードを使うと，簡単にシンタックスハイライトを実装できる．各言語の fontlock-mode 用の設定を書くと色付けできる．また，Vim と同様に，font-lock-variable-name-face のように，変数名の色を決められる．

fontlock-mode は、他のシンタックスハイライト機能と同様に、キーワードや正規表現のリストをもとに色付けしているが、自力でオーバーレイを組み立てることで、独自にシンタックスハイライトを実装することができる。

js2-mode は、Emacs Lisp で書かれた JavaScript パーサーを持っており、Emacs 上で JavaScript のソースコードをコンパイルして色付けしている。しかし、Emacs Lisp で JavaScript をパーサーを書くのは、正規表現を用意するだけに比べると手間が大きい。Nano の Python の設定が 10 行程度であるのに対し、Emacs の js2-mode は 1 万 1000 行もある。

1.5 シンタックスハイライトの実現

プログラムを正しく理解して色をつけるには、構文解析が必要であるが、一般的なシンタックスハイライトは、正規表現などで色をつけている。そのため、誤認識されて、実行時とは異なる構造が示されることがある。

以下は、Emacs の Ruby モードにおける誤認識の例である。class.aaa という変数名の、class の部分に、クラスを宣言するときに利用する class キーワードの色が誤って割り当てられている。

```
class.aaa = Aaa
```

以下は Emacs の CoffeeScript モードの誤認識の例である。2 回の割り算が正規表現リテラルの色だと誤認識されて色付けされている。

```
a = (1 / 2) * (2 / 3)
```

色付けの正規表現のルールを間違えると正しく色付けされない。誤った色付けによってプログラムの構造を誤ってとらえると、ソフトウェアを誤って理解してしまうことも考えられる。正しく色付けすることはシンタックスハイライトの課題である。

クオートが閉じられないと、次の行までずっと文字列という扱いになってしまっていて、色が変わって見にくい。