# COG

https://nedbatchelder.com/code/cog/

Cog是一个文件生成工具。它允许您在源文件中使用Python代码片段作为生成器来生成所需的任何文本。

以下部分是：

* [它有什么作用？](https://nedbatchelder.com/code/cog/index.html#h_what_does_it_do)
* [设计](https://nedbatchelder.com/code/cog/index.html#h_design)
* [安装](https://nedbatchelder.com/code/cog/index.html#h_installation)
* [编写源文件](https://nedbatchelder.com/code/cog/index.html#h_writing_the_source_files)
* [cog模块](https://nedbatchelder.com/code/cog/index.html#h_the_cog_module)
* [跑cog](https://nedbatchelder.com/code/cog/index.html#h_running_cog)

**它有什么作用？**

Cog以一种非常简单的方式转换文件：它找到嵌入其中的Python代码块，执行Python代码，并将其输出插回到原始文件中。该文件可以包含Python代码周围的任何文本。它通常是源代码。

例如，如果您通过cog运行此文件：

// This is my C++ file.  
...  
/\*[[[cog  
import cog  
fnames = ['DoSomething', 'DoAnotherThing', 'DoLastThing']  
for fn in fnames:  
    cog.outl("void %s();" % fn)  
]]]\*/  
//[[[end]]]  
...

它会像这样出现：

// This is my C++ file.  
...  
/\*[[[cog  
import cog  
fnames = ['DoSomething', 'DoAnotherThing', 'DoLastThing']  
for fn in fnames:  
    cog.outl("void %s();" % fn)  
]]]\*/  
void DoSomething();  
void DoAnotherThing();  
void DoLastThing();  
//[[[end]]]  
...

带有三个方括号的行是标记行。**[[[cog**和**]]]**之间的行是生成器Python代码。**[ ]]**和**[[[end]]]**之间的线是发生器的输出。

当cog运行时，它会丢弃最后生成的Python输出，执行生成器Python代码，并将其生成的输出写入文件。特殊标记之外的所有文本行都保持不变。

除了三方括号标记之外，嵌齿标记行还可以包含任何文本。这使得可以从源文件中隐藏生成器Python代码。在上面的示例中，整个Python代码块都是C ++注释，因此Python代码可以保留在原位，而文件被视为C ++代码。

**设计**

Cog的设计易于操作。它将结果写回原始文件，同时保留它执行的代码。这意味着cog可以在同一个文件上运行任意次。而不是拥有源生成器文件和单独的输出文件，通常使用一个文件作为生成器和输出来运行cog。

由于标记行适用于任何语言语法，因此标记可以隐藏源文件中的cog Python代码。这意味着可以将cog文件检入源代码控制，而无需担心将源文件与输出文件分开，而无需修改构建过程，等等。

Cog允许您使用Python的全部功能来生成文本，而不需要模板系统为您的工具减少工具。

**安装**

Cog需要Python 2.6,2.7,3.3,3.4,3.5,3.6或Jython 2.5。

Cog安装了标准的Python distutils脚本：

1. 从[Python包索引](http://pypi.python.org/pypi/cogapp)下载Cog 。
2. 在某处解压缩分发存档。
3. 运行解压缩的setup.py脚本：

$ python setup.py install

您现在应该在Python脚本目录中有cog.py.

**执照**

Cog根据[MIT许可证](http://www.opensource.org/licenses/mit-license.php)分发 。用它来传播世界的美好。

**编写源文件**

要通过cog运行的源文件大多只是通过原始文本传递的纯文本。源文件中的Python代码是标准Python代码。你想用Python生成文本进入你的文件的任何方式都很好。每个Python代码块（在**[[[cog**和**]]]**行之间）称为*生成器*并按顺序执行。

删除每个生成器（在**]]**和**[[[end]]]**行之间的输出区域，并在其位置插入运行Python代码的输出。为了适应所有源文件类型，标记行的格式无关紧要。如果该行包含特殊字符序列，则将整行作为标记。这些行中的任何一行都标志着可执行Python代码的开头：

//[[[cog  
/\* cog starts now: [[[cog \*/  
-- [[[cog (this is cog Python code)  
#if 0 // [[[cog

Cog也可以在没有多行注释的语言中使用。如果标记行在三重括号之前都具有相同的文本，并且生成器代码中的所有行也将此文本作为前缀，则在执行之前从所有生成器行中删除前缀。例如，在SQL文件中，这个：

--[[[cog  
--   import cog  
--   for table in ['customers', 'orders', 'suppliers']:  
--      cog.outl("drop table %s;" % table)  
--]]]  
--[[[end]]]

会产生这个：

--[[[cog  
--   import cog  
--   for table in ['customers', 'orders', 'suppliers']:  
--      cog.outl("drop table %s;" % table)  
--]]]  
drop table customers;  
drop table orders;  
drop table suppliers;  
--[[[end]]]

最后，紧凑的形式可用于单线发电机。开始代码标记和结束代码标记可以出现在同一行上，它们之间的所有文本都将作为单个Python行：

// blah blah  
//[[[cog import MyModule as m; m.generateCode() ]]]  
//[[[end]]]

您也可以使用此表单来导入模块。模块中的顶级语句可以生成代码。

如果您对文件的语法有特殊要求，可以使用--markers选项定义新标记。

如果同一个文件中有多个生成器，则它们使用相同的全局字典执行，因此就好像它们都是一个Python模块一样。

Cog试图用白色空间做正确的事情。您的Python代码可以是块缩进的，以匹配源文件中的周围文本，而cog也会重新缩进输出以使其适合。生成器的所有输出都被收集为一个文本块，删除了一个常见的空白前缀，然后缩进该块以匹配嵌入式生成器的缩进。这意味着输出中最左边的非空白字符将与开始代码标记行具有相同的缩进。输出中的其他行保持相对缩进。

**cog模块**

名为cog的模块提供您调用的函数，以便将输出生成到您的文件中。功能是：

**cog.out***（sOut =''[，dedent = False] [，trimblanklines = False]）*

将文本写入输出。

*sOut*是要写入输出的字符串。

如果*dedent*为True，则在将它们添加到输出之前，从*sOut中*的行中删除公共初始空白区域。如果*trimblanklines*为True，则在将*sOut*添加到输出之前，将从*sOut*中删除初始和尾随空白行。这些选项参数一起使得更容易使用多行字符串，它们仅对多行字符串有用：

cog.out("""  
    These are lines I  
    want to write into my source file.  
""", dedent=True, trimblanklines=True)

**cog.outl**

与**cog.out**相同，但添加了一个尾随换行符。

**cog.msg***（msg）*

使用“Message：”前缀将*msg*打印到stdout。

**cog.error***（msg）*

使用*msg*作为文本引发异常。不包含回溯，因此使用代码生成器的非Python程序员不会感到害怕。

**cog.inFile**

属性，输入文件的路径。

**cog.outFile**

属性，输出文件的路径。

**cog.firstLineNum**

一个属性，即生成器中第一行Python代码的行号。如果需要，这可用于区分同一输入文件中的两个生成器。

**cog.previous**

属性，此生成器的上一次运行的文本输出。这可以用于您喜欢的任何目的，包括使用cog.out（）再次输出。

**跑cog**

Cog是一个命令行实用程序，它以标准形式接受参数。

cog - generate code with inlined Python code.  
  
cog [OPTIONS] [INFILE | @FILELIST] ...  
  
INFILE is the name of an input file, '-' will read from stdin.  
FILELIST is the name of a text file containing file names or  
    other @FILELISTs.  
  
OPTIONS:  
    -c          Checksum the output to protect it against accidental change.  
    -d          Delete the generator code from the output file.   
 标志告诉cog从输出文件中删除生成器。这使您可以在公共文件中生成代码，但不  
 必向客户显示生成器。  
    -D name=val Define a global string available to your generator code.  
    -e          Warn if a file has no cog code in it.  
    -I PATH     Add PATH to the list of directories for data files   
 and modules.   
 将目录添加到用于查找Python模块的路径。  
    -n ENCODING Use ENCODING when reading and writing files.   
 告诉cog读取和写入文件时要使用的编码。  
    -o OUTNAME  Write the output to OUTNAME.  
    -r          Replace the input file with the output.  
    -s STRING   Suffix all generated output lines with STRING.  
    -U          Write the output with Unix newlines (only LF line-endings)   
 使输出文件使用纯Unix新行而不是平台的本机行结尾。您可以在Windows上使用  
 它来生成Unix风格的输出文件。  
    -w CMD      Use CMD if the output file needs to be made writable.  
                A %s in the CMD will be filled with the filename.  
    -x      Excise all the generated output without running the generators.   
 告诉cog删除旧生成的输出而不运行生成器,可以从源文件中删除所有生成的输出。  
    -z      The end-output marker can be omitted, and is assumed at eof.   
 允许您省略[[[end]]]标记行，它将在文件末尾处假设。

    -v      Print the version of cog and exit.  
    --verbosity=VERBOSITY  
                Control the amount of output. 2 (the default) lists all   
 files,1 lists only changed files, 0 lists no files.   
 控制cog应该对它所处理的文件进行讨论的程度。--verbose = 2是默认值：  
 cog将为它考虑的每个文件命名，以及它是否已更改。--verbose = 1将仅命名  
 已更改的文件。--verbose = 0根本不会提及任何文件。  
    --markers='START END END-OUTPUT'  
                The patterns surrounding cog inline instructions. Should  
                include three values separated by spaces, the start, end,  
                and end-output markers. Defaults to '[[[cog ]]] [[[end]]]'.   
 控制标记线的语法。该值必须是包含两个空格的字符串。三个标记是由空格分隔的  
 三个标记。标记的默认值是“[[cog]]] [[[end]]]”  
    -h          Print this help.

除了在命令行上运行cog作为命令：

$ cog [options] [arguments]

您也可以使用Python解释器将其作为模块调用：

$ python -m cogapp [options] [arguments]

请注意，Python模块称为“cogapp”。

**输入文件**

命令行上的文件作为输入文件处理。假设所有输入文件都是UTF-8编码的。使用减号作为文件名（ - ）将读取标准输入。

文件也可以在命令行上用@命名的文本文件中列出：

$ cog @files\_to\_cog.txt

这些@ -files可以嵌套，每行可以包含开关以及要处理的文件。例如，您可以创建文件cogfiles.txt：

*cogfiles.txt*

# These are the files I run through cog  
mycode.cpp  
myothercode.cpp  
myschema.sql -s " --\*\*cogged\*\*"  
readme.txt -s ""

然后像这样调用cog：

cog -s " //\*\*cogged\*\*" @cogfiles.txt

现在cog将处理四个文件，在所有C ++文件上使用C ++语法，在.sql文件中使用SQL语法，在readme.txt文件中根本不使用任何标记。

另一个例子，cogfiles2.txt可以是：

*cogfiles2.txt*

template.h -D thefile=data1.xml -o data1.h  
template.h -D thefile=data2.xml -o data2.h

用cog这样调用：

cog -D version=3.4.1 @cogfiles2.txt

Cog将处理template.h两次，同时创建data1.h和data2.h。两个执行都将变量版本定义为“3.4.1”，但第一次运行将使文件等于“data1.xml”，第二次运行将使文件等于“data2.xml”。

**覆盖文件**

-r标志告诉cog将输出写回输入文件。如果输入文件不可写（例如，因为它尚未从源控制系统中检出），则可以使用-w提供使文件可写的命令：

$ cog -r -w "p4 edit %s" @files\_to\_cog.txt

**设置全局变量**

可以使用-D标志从命令行设置全局值。例如，像这样调用Cog：

cog -D thefile=fooey.xml mycode.txt

将在mycode.txt上运行Cog，但首先定义一个名为file的全局变量，其值为“fooey.xml”。然后可以在生成器代码中引用此变量。您可以在命令行上提供多个-D参数，并且所有参数都将被定义和可用。

该值始终被解释为Python字符串，以简化引用问题。这意味着：

cog -D NUM\_TO\_DO=12

将NUM\_TO\_DO定义为不是整数12，而是定义为字符串“12”，它们在Python中是不同的且不相等的值。使用int（NUM\_TO\_DO）来获取数值。

**校验输出**

如果使用-c标志运行cog，则生成的输出将伴随校验和：

--[[[cog  
--   import cog  
--   for i in range(10):  
--      cog.out("%d " % i)  
--]]]  
0 1 2 3 4 5 6 7 8 9  
--[[[end]]] (checksum: bd7715304529f66c4d3493e786bb0f1f)

如果生成的代码由误导的开发人员编辑，则下次运行cog时，校验和将不匹配，并且cog将停止以避免覆盖已编辑的代码。

**输出行后缀**

为了在grepping源文件时更容易识别生成的行，-s开关提供了一个后缀，该后缀附加到Cog生成的每个非空白文本行。例如，使用此输入文件（mycode.txt）：

*mycode.txt*

[[[cog  
cog.outl('Three times:\n')  
for i in range(3):  
    cog.outl('This is line %d' % i)  
]]]  
[[[end]]]

像这样调用cog：

cog -s " //(generated)" mycode.txt

将产生此输出：

[[[cog  
cog.outl('Three times:\n')  
for i in range(3):  
    cog.outl('This is line %d' % i)  
]]]  
Three times: //(generated)  
  
This is line 0 //(generated)  
This is line 1 //(generated)  
This is line 2 //(generated)  
[[[end]]]

**杂**

使用-n选项可以

使用--verbose选项可以使用--markers选项可以。

-x标志

-d

-U标志

-I标志

-z标志