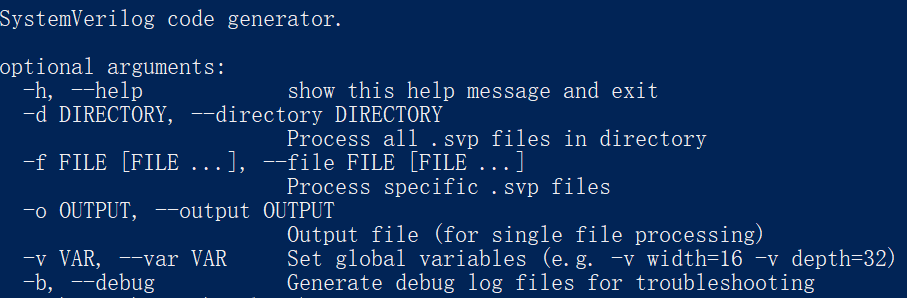
expand\_gen.py脚本说明文档

# 脚本简介

Expand\_gen.py脚本是一个SystemVerilog的轻量级代码生成器。通过处理带有特殊注释命令的源文件(svp)，解析对应语法，实现包括变量的声明替换，执行循环(for)语句以及执行条件判断（if/elsif/else）语句，最终生成具有较强重复性或规律的标准SystemVerilog代码（sv）。

# 使用说明

2.1 脚本参数说明



该脚本支持如下参数：

-h: 输出帮助以及各个参数的使用方法。

-f：指定要处理的源文件，在输入文件语法正确的前提下，支持任意后缀的文本文件。

-o: 指定对应输入文件的输出文件，如未指定，则在输入源文件的相同目录下自动生成名字与输入文件相同，以sv为后缀的输出文件。与-f可以配合使用。

-d: 指定要批量处理的源文件目录，指定输入目录后，脚本会自动获取目录下的所有的svp后缀文件，并进行批量处理，并在相同目录下生成相同文件名的sv后缀文件。经过批处理的文件名与目录固定，不会被-o指定。

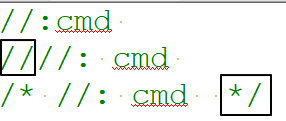
-v：进行全局参数变量传递，所有被处理的源文件，均可以直接使用通过-v传递的变量。

-b：对源文件生成进行debug，生成对应的debug文件

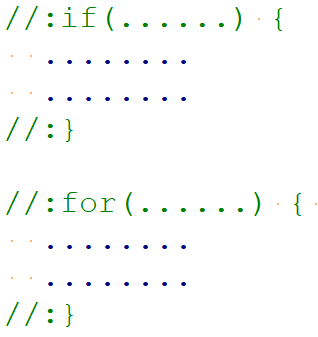
2.2 源文件特殊命令说明

**2.2.1 基本规则**

1.源文件中以”//:”开头的行为生成命令，且不在意命令中的空格等无效字符，对于生成命令，不会输出到目标文件中。另外，对于命令行也支持”//” “/\*\*/”等对该命令进行注释



2.对于循环以及分支等命令语句，需要通过”{}”明确显示出其作用域，其中花括号需要配对使用，括号内的文本为待生成的内容，另外，支持循环与分支等命令语句之间进行嵌套。

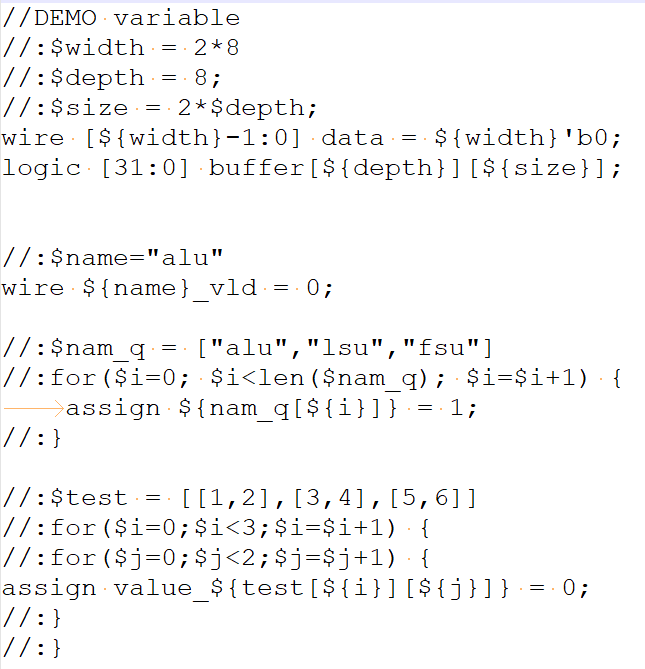
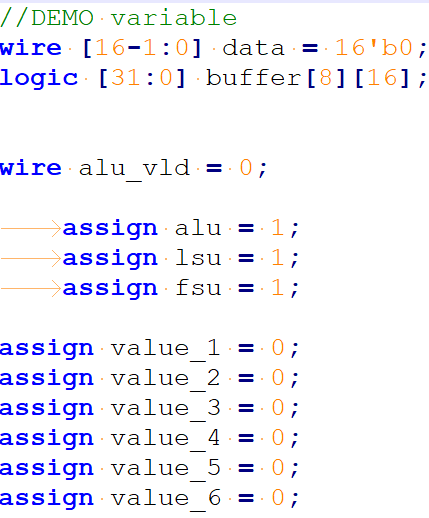


3.对于变量作用域为就近原则，在循环和分支内部，可以访问外部变量，但是如果内部再次定义同名变量，则内部变量覆盖外部变量；同时，内部定义的变量在外部不可见。

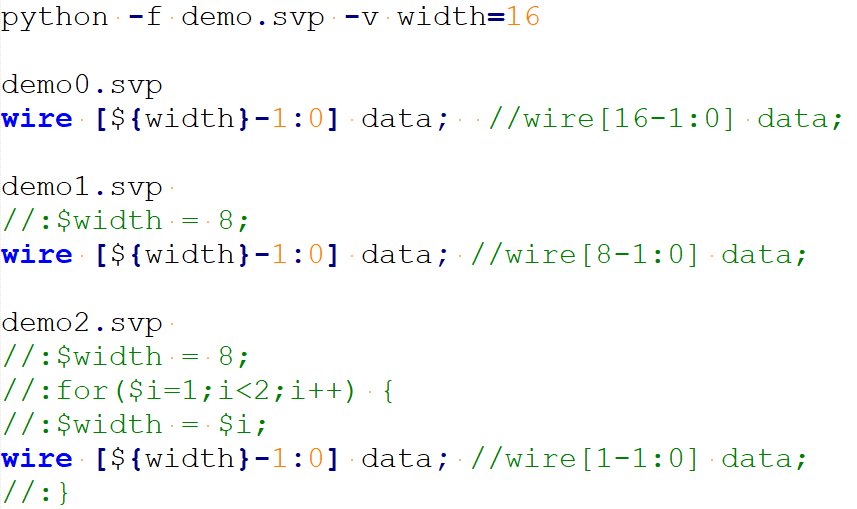
**2.2.2 语法说明**

1.变量声明语句

对于命令语句中的变量声明，可以定义的变量类型为数值，字符串，列表均可，也可进行变量之间的运算，声明后的分号不做强制要求，在声明时通过美元符($)表示。在代码生成时，如果使用之前声明的变量，需通过美元符加花括号的形式(${})体现，脚本会自动收集关键字并进行替换。

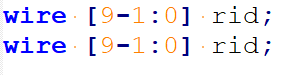
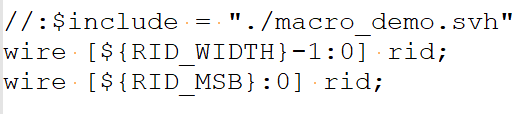
需要注意的是，对于变量的使用存在就近原则，即代码生成时会使用相同名字最近的变量进行替换，如图，demo0中使用全局变量，demo1中使用文件内变量，demo2中使用块语句内的变量。

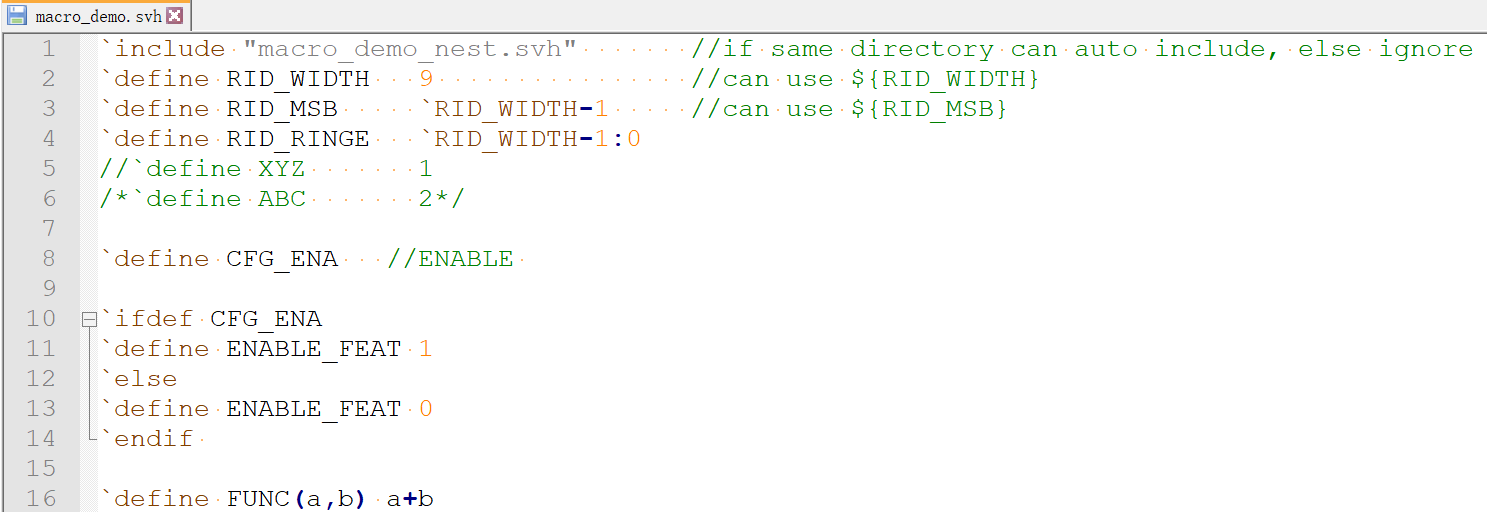


另外，在变量定义时，禁止使用命令关键字作为变量名，即对于提供的命令语句，if/elsif/else/for/include，禁止定义该变量。

2.宏文件导入语句

命令语句支持SystemVerilog以及Verilog兼容的宏定义文件导入，对于导入文件的内容，代码生成器会自动解析宏文件的内容，并自动将宏定义为可用变量。





脚本虽支持直接传入宏文件，自动产生可用变量的声明，但是由于SystemVerilog/Verilog语法灵活多边形，部分情况脚本可能无法正确处理，如下做详细说明。

1.对于宏文件中的内容，脚本会自动忽略注释。

2. 如果verilog中的宏文件存在嵌套（如macro\_demo\_nest.svh），对于相同路径下的宏文件嵌套include，脚本也会自动对嵌套宏文件的内容进行解析，并自动加入到可用变量中，如果无法找到嵌套宏文件，则会自动忽略。

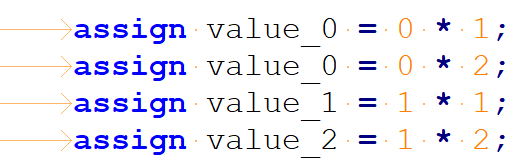
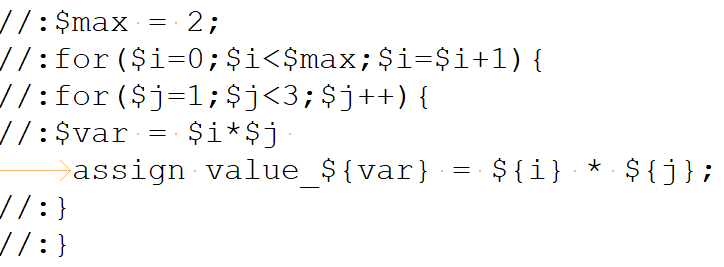
3. 如果文件中的宏存在一定关联，如上图中RID\_WIDTH与RID\_MSB为例，脚本会自动替换RID\_MSB为9-1，但由于关联程度可能极为复杂，且可能存在多层嵌套甚至死锁，脚本无法保证一定处理成功。建议用户不使用非直接定义的宏作为变量。

4. 对于宏文件中的ifdef/else/endif等，脚本会在相同路径下寻找合法分支，如上图中的CFG\_ENA，脚本会自动处理变量ENABLE\_FEAT为1，但是如果存在跨文件的`ifdef，脚本可能无法处理成功，用户需谨慎使用该变量。

如果用户使用include命令语句，要求脚本自动定义变量，建议用户仅使用单一define的变量；另外，防止宏文件中的变量和自定义变量冲突，且大多情况下，宏为大写字母，所以建议用户的自定义变量声明为小写字母，从而避免声明重复，如果存在冲突，脚本依然按照就近定义的原则使用，另外，建议路径使用带环境变量的绝对路径。

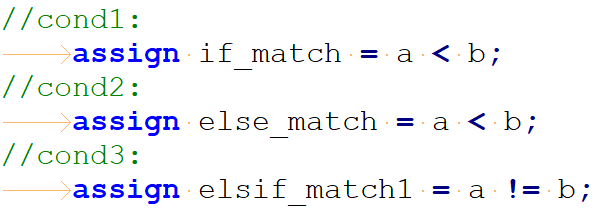
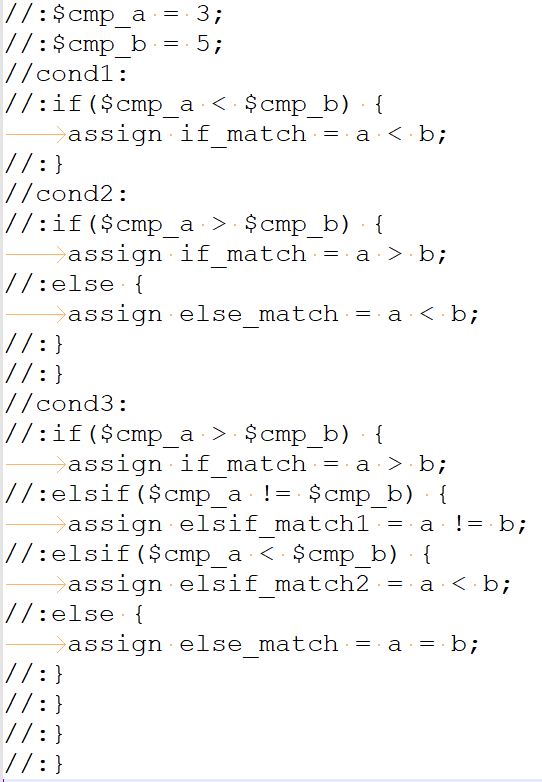
3.for循环语句

命令语句支持for循环，for循环语句中的的变量可以自定义，也可以使用之前定义过的内容，另外，对于循环的步进处理，除了支持i=i+x,i=i-x等基本语法外，对自加1的情况进行可语法优化，支持i++/i--等语句。同时也支持for循环与if条件之间的嵌套处理。

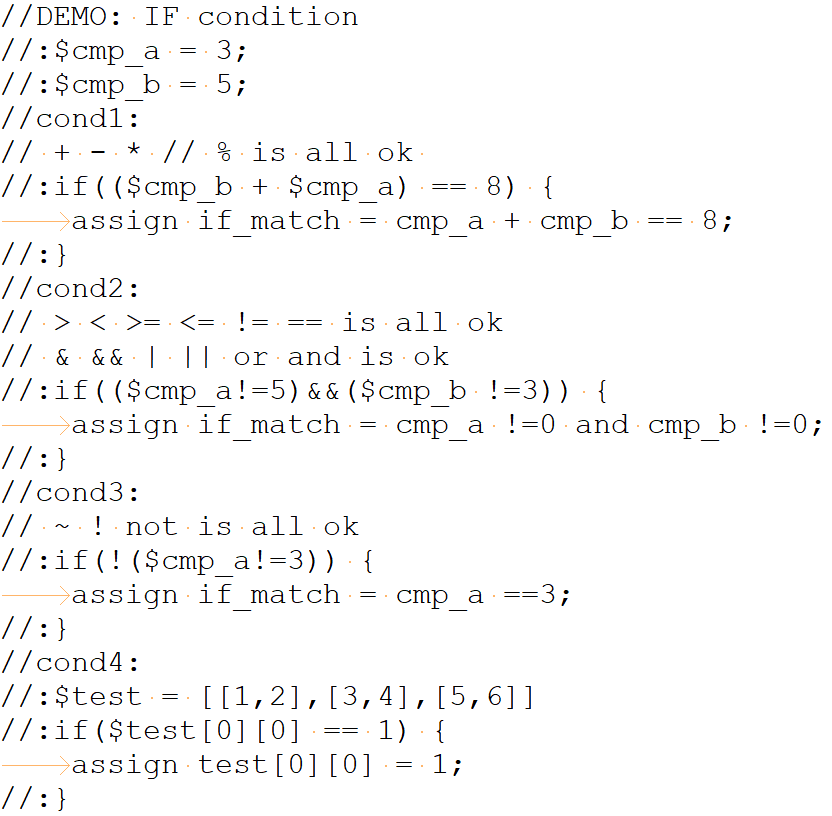
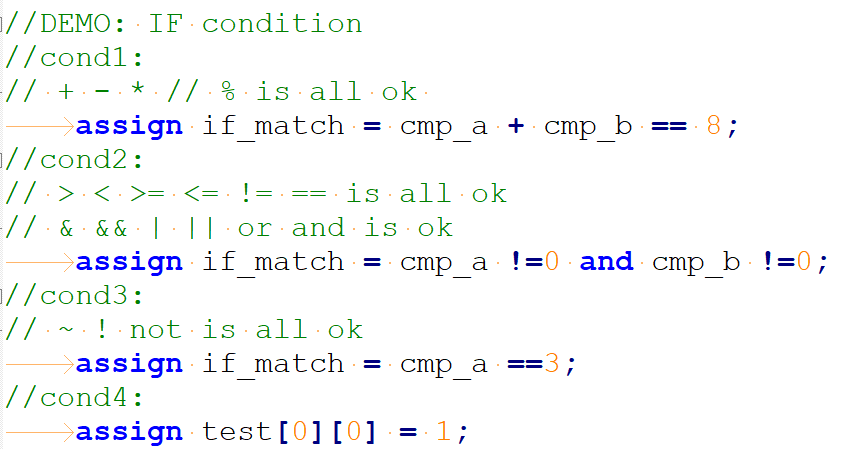


4.if条件语句

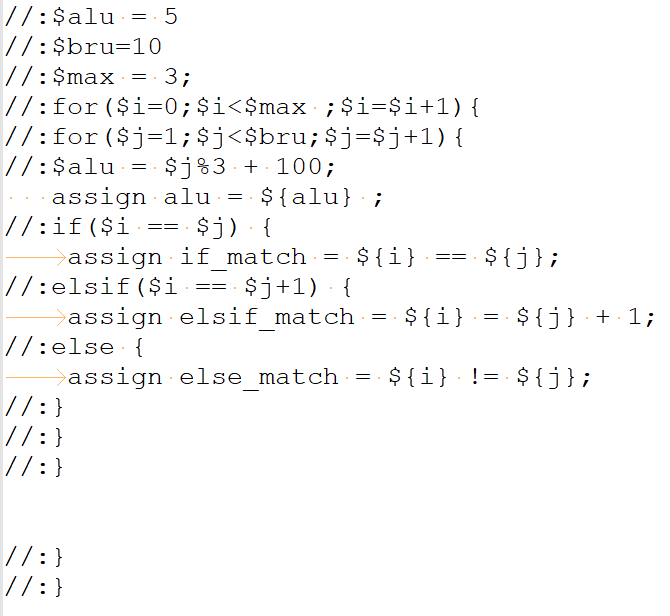
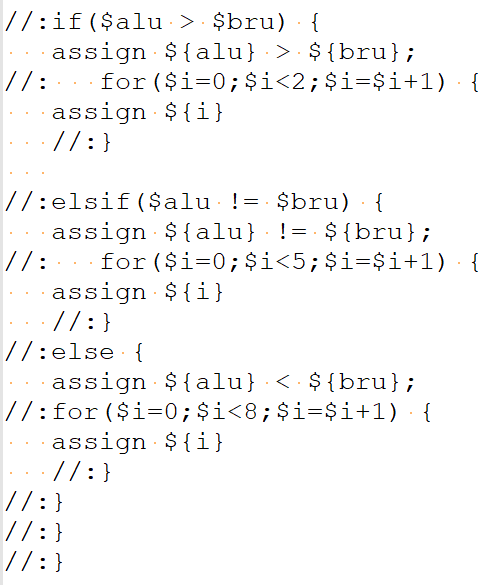
命令语句支持if判断，包括if语句，if/else语句，if/elsif/else语句的判断，当在一个if块内，多个elsif分支均满足时，只有第一个match条件的分支会被执行，并生成代码。同时也支持if条件与for循环之间的嵌套处理。

对于if条件块中的条件代码，比较方式支持所有python语句支持的算数运算(+ - \* // %)以及逻辑运算(& | ! && || and or not > >= < <= == !=)。值得注意的是，由于Python语法不支持&&，||等逻辑运算，脚本中会自动对此进行替换为and or等内容。另外对于除法，大多情况需要使用”//”整除操作。其他运算则使用python自带运算即可。

对于for与if语句，支持之间进行嵌套使用，并进行代码生成。

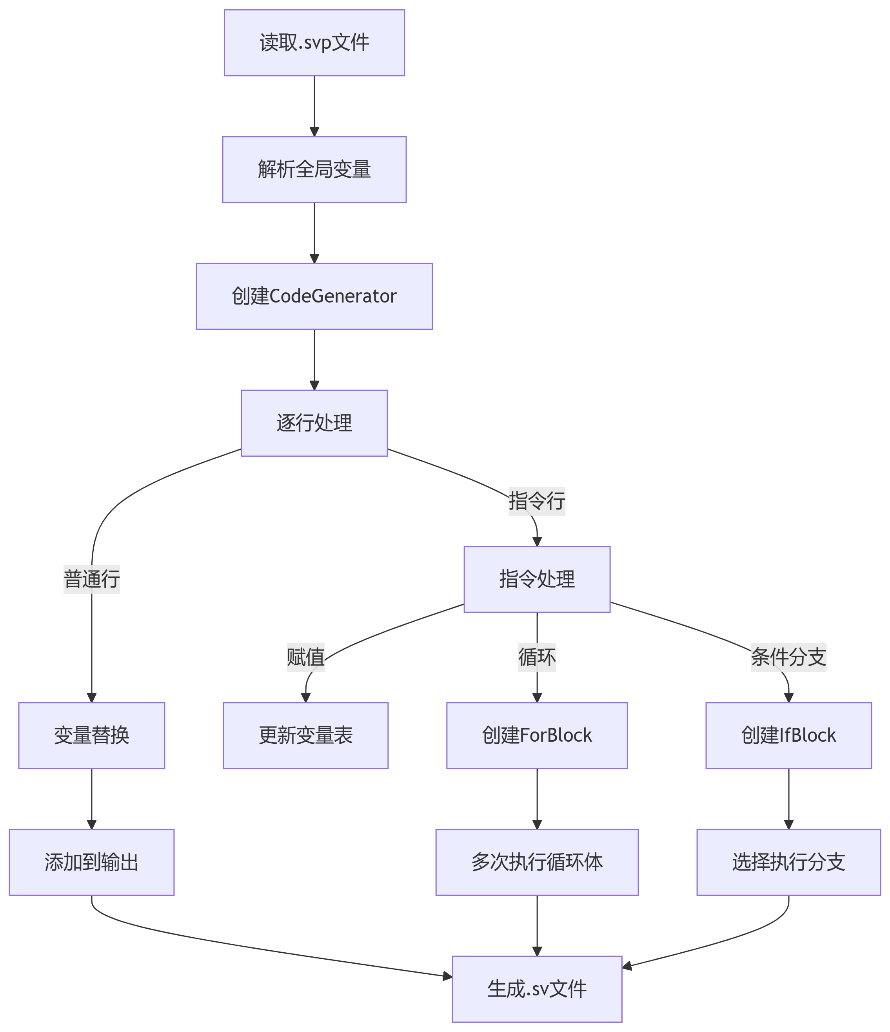
 

# 原理介绍

3.1 基本原理

通过Python的内置函数eval，将字符串作为代码执行并返回结果，包括执行数学表达式，转换数据结构以及调用变量，实现代码的生成。

脚本中含有代码生成器类，用于进行源文件的变量替换，同时执行生成代码。对于循环语句以及分支语句，通过按照条件递归调用代码生成器，实现嵌套生成。结构框图如下图所示。



3.2 代码说明

**3.2.1** main函数

1.利用argparse解析命令行参数，获取目录位置，文件位置，debug条件，并作自检查

2.通过parse\_vars()解析全局变量，根据脚本传递的参数，获取全局变量,分割变量名以及对应的值，存入变量字典中

3.例化代码生成器，生成代码，将输出内容进行写出

**3.2.2** CodeGenerator

1.eval\_expr运用python语法进行表达式计算求值，如果直接求值失败，可能由于存在变量，将变量替换后再次求值计算

该类时核心代码的生成引擎，用于变量的存储，查找更新，对各个变量进行管理，同时负责命令的分发处理

eval\_expr:用作表达式的求值

replace\_vars:用作变量的替换，对于普通代码行，如果存在${var}等变量，进行替换

process\_assignment:用作处理变量赋值的命令，对于命令行，如果存在变量的声明，则解析变量名与表达式，并解析为python字面量。如果解析失败，则可能存在python语法处理，则进行表达式求值，完成后更新变量词典

process\_lines:用作处理每一个代码行，进行命令的分发，对于每一行代码，如果为命令行中的变量赋值，则对变量进行处理，如果为循环命令，则例化ForBlock，如果为分支命令，则例化IfBlock处理，如果为普通行，则进行变量替换后，加入到output中，准备输出。

generate：对文件中的每一行代码进行处理并输出

**3.2.3** ForBlock

parse：解析循环头，包括循环初值，条件，步进等内容，然后创建子生成器同时继承父作用域的变量。之后执行循环，检查条件是否满足，如果满足，则执行循环体，创建新的代码生成器进行代码生成，之后执行步进，迭代循环；如果存在命令的嵌套，通过新的代码生成器进行递归处理。

normalize\_step\_expr：将步进表达式归一化，对于i++等归一为i=i+1

step:执行步进语句，更新循环变量的值

**3.2.4** IfBlock

收集所有的分支块，包括if/elsif/else，之后按顺序求值分支条件是否满足，执行第一个满足条件的分支，创建子生成器，并处理该分支内的所有行，如果存在命令的嵌套，通过新的代码生成器进行递归处理。处理后跳过其他的分支。

3.3 维护说明

此设计实现了灵活可扩展的模板引擎，通过递归生成器结构支持任意深度的嵌套控制流，同时保持清晰的变量作用域管理。

