# 代码搜索引擎－设计文档

## 1. 概述

该模块的功能旨在进行相似代码的检索，即用户输入一个代码片段作为查询，系统从后台数据库返回与该查询相似性较高的代码，按相似度从高到低进行排列，展示给用户。

搜索引擎后台所采用的代码相似性匹配方法有三种，分别是原始字符串的精确匹配、词法分析后的Token串匹配、语法分析后的语法树相似性匹配。

下面我们先依次介绍三种匹配方式，然后再说明我们如何将三种方式的搜索结果进行结合。

## 2. 字符串精确匹配

该方法直接对源代码进行简单的预处理，然后进行字符串的精确匹配即可。

预处理过程如下：

将代码中的所有字符，根据附录中的“符号替换词典”进行替换，并加上“punc\_”前缀，所有字母组成的单词加上“char\_” 前缀，然后用空格隔开组成一个长字符串。例如：

a = b + c 替换为： char\_a punc\_BN char\_b punc\_BF char\_c

该方法处理样例如下：

源代码：

# 这是一个注释

for i in range(10):

print('abcde')

预处理后字符串：

char\_for char\_i char\_in char\_range punc\_AW char\_10 punc\_AX punc\_B0 char\_print punc\_AW punc\_QT char\_abcde punc\_QT punc\_AX

注意：忽略了代码中的注释语句。

将查询代码和数据库中代码转换之后的字符串进行精确匹配，相同则返回，不同则不返回。

## 3. 词法分析 Token 串匹配

词法分析（lexical analysis）是计算机科学中将字符序列转换为标记（token）序列的过程。

该方法对源代码进行词法分析，旨在忽略代码中变量名、数值大小这类区别。

进行词法分析的程序或者函数叫作词法分析器（lexical analyzer，简称 lexer）。

我们使用Python内置的词法分析器（token模块）:

(https://docs.python.org/3/library/tokenize.html).

可以直接使用指令：python3 -m tokenize -e (+ filename) 对源代码进行词法分析，其中filename为代码文件所在路径。

我们对词法分析结果进行一些预处理，首先同样是将代码中的所有字符，根据附录中的“符号替换词典”进行替换，并加上“punc\_”前缀，然后将代码中的关键字统一加上“key\_”前缀，然后将词法分析中得到的Token标识符替换为较短的唯一标识符（根据附录中的”Token替换词典“，目的是使得最终得到的字符串长度较小）。例如：

a = b + c 替换为： AO punc\_BN AO punc\_BF AO

> 注意： 词法分析将代码中的所有变量名标记为“NAME”，我们用“AO”缩写来进行表示。

该方法处理样例如下：

源代码：

# 这是一个注释

for i in range(10):

print('abcde')

处理后字符串：

key\_for AO key\_in key\_range punc\_AW AR punc\_AX punc\_B0 key\_print punc\_AW AU punc\_AX

最后，将查询代码和数据库中代码转换之后的字符串进行精确匹配，相同则返回，不同则不返回。

## 4. 语法树相似性匹配

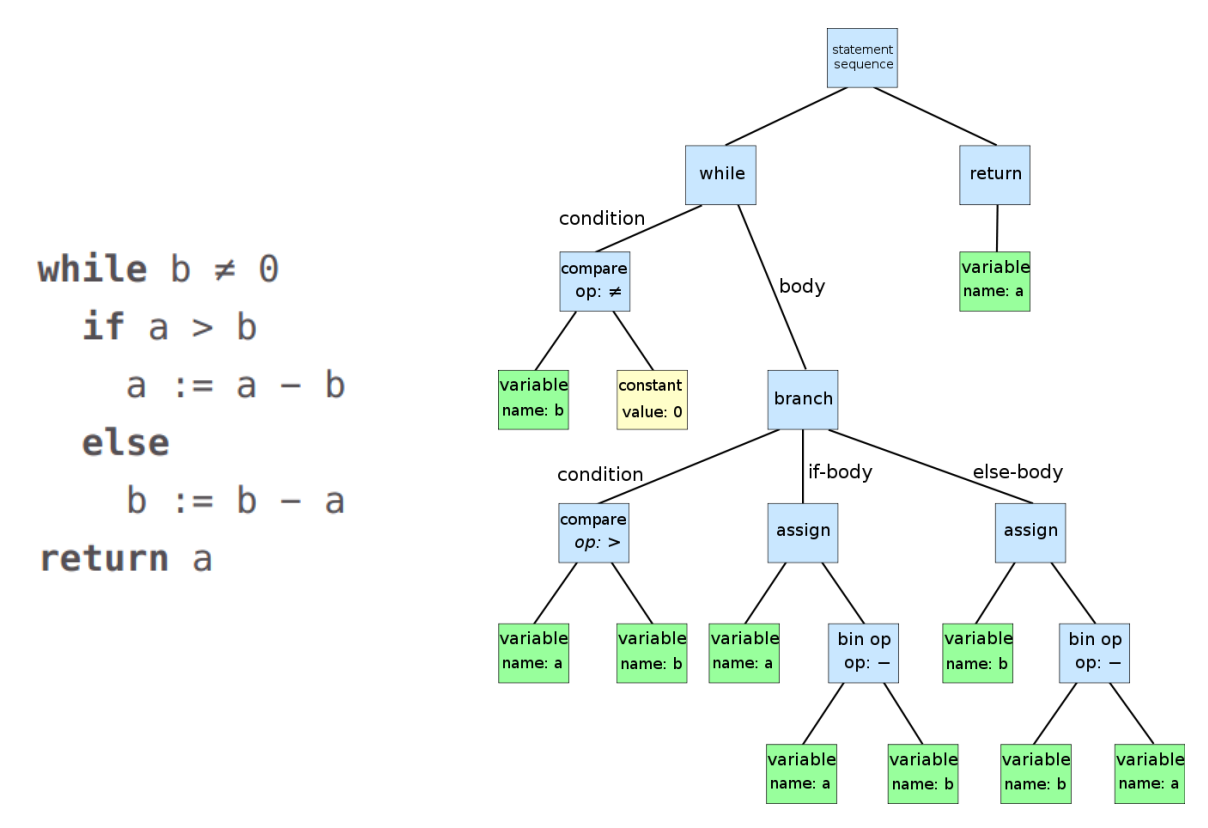
语法分析器基于特定语言的语法，将Token序列（由词法分析器生成）转换成一个抽象语法树。

抽象语法树（Abstract Syntax Tree，AST），或简称语法树（Syntax tree），是源代码语法结构的一种抽象表示。它以树状的形式表现编程语言的语法结构，树上的每个节点都表示源代码中的一种结构。之所以说语法是 “抽象” 的，是因为这里的语法并不会表示出真实语法中出现的每个细节。比如，嵌套括号被隐含在树的结构中，并没有以节点的形式呈现。

具体概念请参考维基百科[抽象语法树]

(https://www.wikiwand.com/zh-hans/%E6%8A%BD%E8%B1%A1%E8%AA%9E%E6%B3%95%E6%A8%B9)，以及其他相关资料。

下面例子中，左边为源代码，右边是其对应的抽象语法树：



在进行语法分析时，我们采用的工具如下：

1. Python ast : Python自带的语法分析器模块，通过parse()函数生成抽象语法树，并提供对抽象语法树的遍历。链接如下：

[https://docs.python.org/3.7/library/ast.html](https://docs.python.org/3.7/library/ast.html).

1. C++ Clang : 提供了C、C++等语言的语法分析器。链接如下：

[https://clang.llvm.org/](https://clang.llvm.org/).

在进行语法分析得到语法树之后，我们采用论文*Similarity Evaluation on Tree-structured Data*中提出的方法进行语法树的特征提取，将树的表达转化成为一个字符序列。转换方法如下：

1. 先将语法树转换成二叉树，采用左孩子右兄弟的转换方法。

2. 然后对二叉树的所有叶结点进行结点填充，使得每个叶节点都拥有其左孩子和右孩子结点，填充结点的值可以统一为任意值。

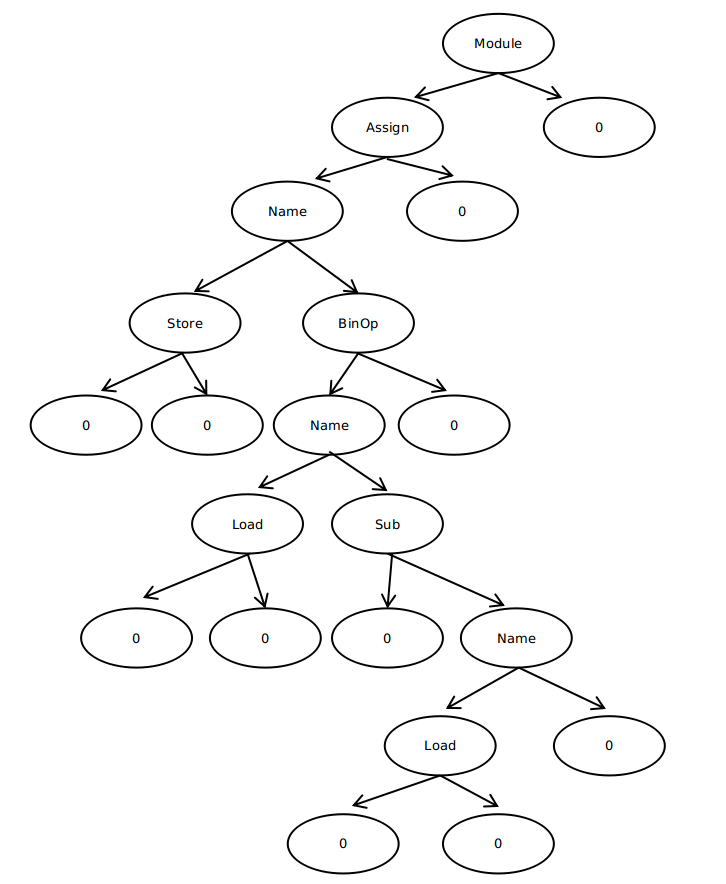
3. 取出转换之后二叉树中的每个两层子树，当做一个特征，将该两层子树结构中的三个结点值用下划线“\_”连接起来，作为最终树的字符串表达中的一项。

4. 将该语法树中的所有特征项用空格隔开，作为最终的树的字符串表达。

例如下面这行简单代码：

a = a - b

利用Python AST模块进行语法分析，得到的语法树结构(已经转换为二叉树)如下：



其中的0结点为叶子结点的填充结点，按照上面的特征提取过程中的第3步，进一步得到整个语法树的字符序列为：

Module\_Assign\_0 Assign\_Name\_0 Name\_Store\_BinOp Store\_0\_0 BinOp\_Name\_0 Name\_Load\_Sub Load\_0\_0 Sub\_0\_Name Name\_Load\_0 Load\_0\_0

其中，该字符序列中的每一项为二叉树中的每个两层子树，将该两层子树结构中的三个结点值用下划线“\_”连接起来得到的字符串。然后将所有子项用空格隔开，组成整个语法树的字符序列。

所以，我们将判断两段代码的相似性，转化为判断语法树的相似性，最后转化为判断两个字符序列之间的相似性。

## 5. 检索方案结合

我们最终采用的搜索方式，是将上述三种检索方式结合起来。

我们将原始字符序列、词法分析Token序列、语法分析序列三种解析结果合在一起，通过ElasticSearch，建立索引。

即数据库中的所有代码，都存在以上三种解析方式的索引。

实际使用过程中，传来一个查询query，我们会对该查询执行三种解析，得到三种不同的查询串，再用每一种查询串进行ElasticSearch的查询，然后将三种查询串搜索的结果进行合并，采取的合并策略是：原始字符串精确匹配的结果，优先级最高，排在最前面；词法分析的Token串查询的结果，优先级次之，依次排在后面；语法树的匹配结果，排在字符串结果和词法分析结果的后面。与此同时，在合并的过程中，进行去重，即字符串精确匹配的结果，可能也在词法分析串搜索中也会出现，则去除词法分析中的这些返回结果。

注意：我们采用的搜索结合方式，针对查询较短代码时，返回的结果中通常是很多精确匹配的结果，以及词法分析串匹配的结果；而针对查询的代码较长时，精确匹配的结果往往很少，此时返回的应该大部分是语法结构的匹配结果。

注意：在前端查询界面，建议给予用户一定的使用提示，提示用户查询时，尽量输入符合基本语法的代码片段（例如，简单的括号要配对起来），这样才能返回较好的搜索结果。

## 附录

1. 符号替换对应词典

{

"'": 'QT',

'"': 'DQT',

'(': 'AW',

')': 'AX',

'[': 'AY',

']': 'AZ',

':': 'B0',

',': 'BD',

';': 'BE',

'+': 'BF',

'-': 'BG',

'\*': 'BH',

'/': 'BI',

'|': 'BJ',

'||': 'DBJ',

'&': 'BK',

'&&': 'DBK',

'<': 'BL',

'>': 'BM',

'=': 'BN',

'.': 'BO',

'%': 'BP',

'{': 'BQ',

'}': 'BR',

'==': 'BS',

'!=': 'BT',

'<=': 'BU',

'>=': 'BV',

'~': 'BW',

'^': 'BX',

'<<': 'BY',

'>>': 'BZ',

'\*\*': 'CD',

'+=': 'CE',

'-=': 'CF',

'\*=': 'CG',

'/=': 'CH',

'%=': 'CI',

'&=': 'CJ',

'|=': 'CK',

'^=': 'CL',

'<<=': 'CM',

'>>=': 'CN',

'\*\*=': 'CO',

'//': 'CP',

'//=': 'CQ',

'@': 'CR',

'@=': 'CS',

}

2. Token 替换词典

{

'COMMENT': 'AB',

'NL': 'AC',

'ENCODING': 'AD',

'ASYNC': 'AE',

'value': 'AF',

'name': 'AG',

'AWAIT': 'AH',

'DEDENT': 'AI',

'ELLIPSIS': 'AJ',

'ENDMARKER': 'AK',

'ERRORTOKEN': 'AL',

'INDENT': 'AM',

'N\_TOKENS': 'AN',

'NAME': 'AO',

'NEWLINE': 'AP',

'NT\_OFFSET': 'AQ',

'NUMBER': 'AR',

'OP': 'AS',

'RARROW': 'AT',

'STRING': 'AU',

'tok\_name': 'AV',

'LPAR': 'AW',

'RPAR': 'AX',

'LSQB': 'AY',

'RSQB': 'AZ',

'COLON': 'B0',

'COMMA': 'BD',

'SEMI': 'BE',

'PLUS': 'BF',

'MINUS': 'BG',

'STAR': 'BH',

'SLASH': 'BI',

'VBAR': 'BJ',

'DOUBLEVBAR': 'DBJ',

'AMPER': 'BK',

'DOUBLEAMPER': 'DBK',

'LESS': 'BL',

'GREATER': 'BM',

'EQUAL': 'BN',

'DOT': 'BO',

'PERCENT': 'BP',

'LBRACE': 'BQ',

'RBRACE': 'BR',

'EQEQUAL': 'BS',

'NOTEQUAL': 'BT',

'LESSEQUAL': 'BU',

'GREATEREQUAL': 'BV',

'TILDE': 'BW',

'CIRCUMFLEX': 'BX',

'LEFTSHIFT': 'BY',

'RIGHTSHIFT': 'BZ',

'DOUBLESTAR': 'CD',

'PLUSEQUAL': 'CE',

'MINEQUAL': 'CF',

'STAREQUAL': 'CG',

'SLASHEQUAL': 'CH',

'PERCENTEQUAL': 'CI',

'AMPEREQUAL': 'CJ',

'VBAREQUAL': 'CK',

'CIRCUMFLEXEQUAL': 'CL',

'LEFTSHIFTEQUAL': 'CM',

'RIGHTSHIFTEQUAL': 'CN',

'DOUBLESTAREQUAL': 'CO',

'DOUBLESLASH': 'CP',

'DOUBLESLASHEQUAL': 'CQ',

'AT': 'CR',

'ATEQUAL': 'CS',

'DOUBLEQUOTE': 'DQT',

'QUOTE': 'QT',

}