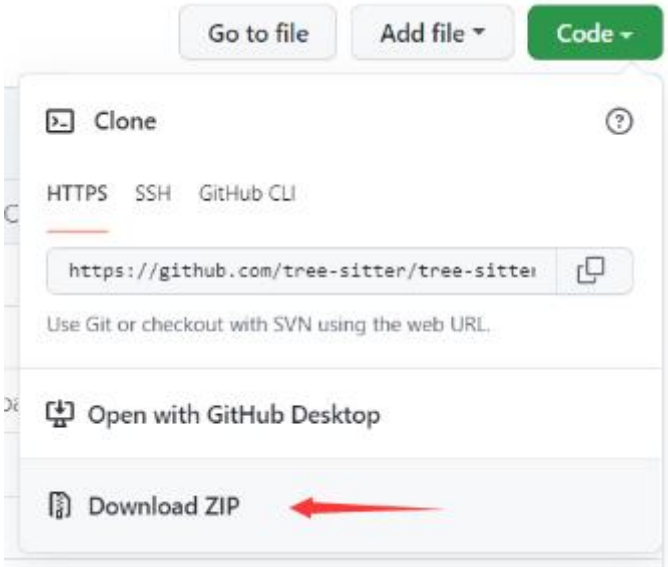
Tree-sitter 生成 AST

1. 安装 tree-sitter 库

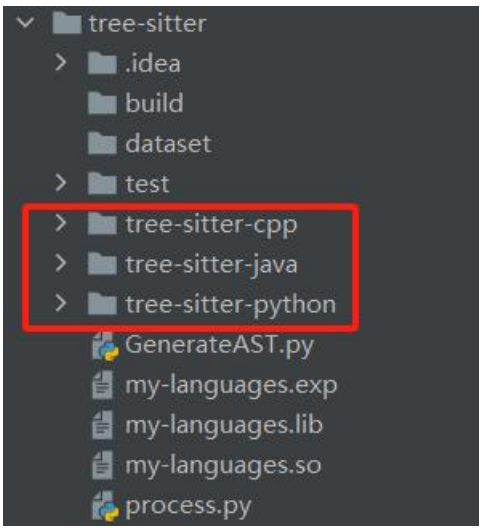
在 pytorch 环境中，用 anaconda 安装 tree-sitter 库：
pip install tree-sitter -i http://pypi.douban.com/simple --trusted-host pypi.douban.com

2. 下载语言解析器

在 https://github.com/tree-sitter 下载对应语言的解析器。以 java 为例，在 https://github.com/tree-sitter/tree-sitter-java 下载 tree-sitter-java。



下载好的文件 tree-sitter-java 放在代码的同级目录下



3. 设置要解析的编程语言

Python 使用 tree-sitter 生成 AST 的时候，设置对应的语言：

```
from tree_sitter import Language, Parser

Language.build_library(
  # Store the library
  'my-languages.so',

  # Include one or more languages
  ['tree-sitter-java']
)

JAVA_LANGUAGE = Language('my-languages.so', 'java')
parser = Parser()
parser.set_language(JAVA_LANGUAGE)
```

4. 测试程序

输出 SBT 格式的 AST：

```
# -- coding:utf-8 --
from tree_sitter import Language, Parser

Language.build_library(
  # Store the library in the `build` directory
  'my-languages.so',

  # Include one or more languages
  ['tree-sitter-java']
)

JAVA_LANGUAGE = Language('my-languages.so', 'java')
parser = Parser()
```

```
parser.set_language(JAVA_LANGUAGE)

src = 'class Test {public String extractFor(Integer id){LOG.debug(\"Extracting method with ID:{}\", id);return requests.remove(id);}}';

print(src)

tree = parser.parse(bytes(src,'utf8'))
cursor = tree.walk()
ast = ''

def make_move(cursor, move):
 # 递归遍历该树，把每个节点的信息保存起来，包括结点的类型、涉及范围的代码行起始位置、终止位置。
 # cursor: 即当前光标的位置（即节点的位置），通过 cursor.node 即可获取当前节点
 # move: 把 move 参数作为当前节点的移动方向
 # str: ast 树 SBT 遍历的结果
 global ast

 type = cursor.node.type
 if('identifier' in type or 'literal' in type):
  type=type+'_'+(str)(cursor.node.text)[1:]
 # type=type+' '

 if (move == "down"):
  ast=ast+'('+type
  if (cursor.goto_first_child()):
   make_move(cursor, "down")
  elif (cursor.goto_next_sibling()):
   ast=ast +')' + type
   make_move(cursor, "right")
  elif (cursor.goto_parent()):
```

```
            ast = ast + ')' + type
            make_move(cursor, "up")
        elif (move == "right"):
          ast=ast+'('+type
          if (cursor.goto_first_child()):
            make_move(cursor, "down")
          elif (cursor.goto_next_sibling()):
            ast = ast + ')' + type
            make_move(cursor, "right")
          elif (cursor.goto_parent()):
            ast = ast + ')' + type
            make_move(cursor, "up")
        elif move == "up":
          ast=ast+')'+type
          if (cursor.goto_next_sibling()):
            make_move(cursor, "right")
          elif (cursor.goto_parent()):
            make_move(cursor, "up")


    make_move(cursor, "down")
    print(ast)
```

5. 节点的 label

tree-sitter 解析器里的 grammar.js 文件里定义了各个节点，src 文件夹下的 parser.c 里面有列举节点的 label:

1. identifier
2. decimal_integer_literal
3. hex_integer_literal
4. octal_integer_literal
5. binary_integer_literal
6. decimal_floating_point_literal
7. hex_floating_point_literal
8. true
9. false

10. character_literal
11. string_literal
12. text_block
13. null_literal
14. boolean_type
15. void_type
16. this
17. super
18. line_comment
19. block_comment
20. program
21. cast_expression
22. assignment_expression
23. binary_expression
24. instanceof_expression
25. lambda_expression
26. inferred_parameters
27. ternary_expression
28. unary_expression
29. update_expression
30. array_creation_expression
31. dimensions_expr
32. parenthesized_expression
33. class_literal
34. object_creation_expression
35. field_access
36. array_access
37. method_invocation
38. argument_list
39. method_reference
40. type_arguments
41. wildcard
42. dimensions
43. switch_expression
44. switch_block
45. switch_block_statement_group
46. switch_rule
47. switch_label
48. block
49. expression_statement
50. labeled_statement
51. assert_statement
52. do_statement
53. break_statement
54. continue_statement

55. return_statement
56. yield_statement
57. synchronized_statement
58. throw_statement
59. try_statement
60. catch_clause
61. catch_formal_parameter
62. catch_type
63. finally_clause
64. try_with_resources_statement
65. resource_specification
66. resource
67. if_statement
68. while_statement
69. for_statement
70. enhanced_for_statement
71. marker_annotation
72. annotation
73. annotation_argument_list
74. element_value_pair
75. element_value_array_initializer
76. module_declaration
77. module_body
78. requires_module_directive
79. requires_modifier
80. exports_module_directive
81. opens_module_directive
82. uses_module_directive
83. provides_module_directive
84. package_declaration
85. import_declaration
86. asterisk
87. enum_declaration
88. enum_body
89. enum_body_declarations
90. enum_constant
91. class_declaration
92. modifiers
93. type_parameters
94. type_parameter
95. type_bound
96. superclass
97. super_interfaces
98. type_list
99. permits

100. class_body
101. static_initializer
102. constructor_declaration
103. constructor_body
104. explicit_constructor_invocation
105. scoped_identifier
106. field_declaration
107. record_declaration
108. annotation_type_declaration
109. annotation_type_body
110. annotation_type_element_declaration
111. interface_declaration
112. extends_interfaces
113. interface_body
114. constant_declaration
115. variable_declarator
116. array_initializer
117. annotated_type
118. scoped_type_identifier
119. generic_type
120. array_type
121. integral_type
122. floating_point_type
123. formal_parameters
124. formal_parameter
125. receiver_parameter
126. spread_parameter
127. throws
128. local_variable_declaration
129. method_declaration
130. type_identifier

Python

1. identifier
2. ellipsis
3. escape_sequence
4. type_conversion
5. integer
6. float
7. true
8. false
9. none
10. comment
11. module
12. import_statement

13. import_prefix
14. relative_import
15. future_import_statement
16. import_from_statement
17. aliased_import
18. wildcard_import
19. print_statement
20. chevron
21. assert_statement
22. expression_statement
23. named_expression
24. return_statement
25. delete_statement
26. raise_statement
27. pass_statement
28. break_statement
29. continue_statement
30. if_statement
31. elif_clause
32. else_clause
33. match_statement
34. case_clause
35. for_statement
36. while_statement
37. try_statement
38. except_clause
39. finally_clause
40. with_statement
41. with_clause
42. with_item
43. function_definition
44. parameters
45. lambda_parameters
46. list_splat
47. dictionary_splat
48. global_statement
49. nonlocal_statement
50. exec_statement
51. class_definition
52. parenthesized_list_splat
53. argument_list
54. decorated_definition
55. decorator
56. block
57. expression_list

58. dotted_name
59. tuple_pattern
60. list_pattern
61. default_parameter
62. typed_default_parameter
63. list_splat_pattern
64. dictionary_splat_pattern
65. as_pattern
66. not_operator
67. boolean_operator
68. binary_operator
69. unary_operator
70. comparison_operator
71. lambda
72. assignment
73. augmented_assignment
74. pattern_list
75. yield
76. attribute
77. subscript
78. slice
79. call
80. typed_parameter
81. type
82. keyword_argument
83. list
84. set
85. tuple
86. dictionary
87. pair
88. list_comprehension
89. dictionary_comprehension
90. set_comprehension
91. generator_expression
92. parenthesized_expression
93. for_in_clause
94. if_clause
95. conditional_expression
96. concatenated_string
97. string
98. interpolation
99. format_specifier
100. format_expression
101. await
102. positional_separator

103. keyword_separator
104. as_pattern_target
105. case_pattern

C++

1. identifier
2. preproc_directive
3. preproc_arg
4. ms_restrict_modifier
5. ms_unsigned_ptr_modifier
6. ms_signed_ptr_modifier
7. primitive_type
8. number_literal
9. escape_sequence
10. system_lib_string
11. true
12. false
13. null
14. comment
15. auto
16. this
17. nullptr
18. literal_suffix
19. raw_string_literal
20. translation_unit
21. preproc_include
22. preproc_def
23. preproc_function_def
24. preproc_params
25. preproc_call
26. preproc_if
27. preproc_ifdef
28. preproc_else
29. preproc_elif
30. parenthesized_expression
31. preproc_defined
32. unary_expression
33. call_expression
34. argument_list
35. binary_expression
36. function_definition
37. declaration
38. type_definition
39. linkage_specification
40. attribute_specifier

41. attribute
42. attribute_declaration
43. ms_declspec_modifier
44. ms_based_modifier
45. ms_call_modifier
46. ms_unaligned_ptr_modifier
47. ms_pointer_modifier
48. declaration_list
49. parenthesized_declarator
50. abstract_parenthesized_declarator
51. attributed_declarator
52. pointer_declarator
53. abstract_pointer_declarator
54. function_declarator
55. abstract_function_declarator
56. array_declarator
57. abstract_array_declarator
58. init_declarator
59. compound_statement
60. storage_class_specifier
61. type_qualifier
62. sized_type_specifier
63. enum_specifier
64. enumerator_list
65. struct_specifier
66. union_specifier
67. field_declaration_list
68. field_declaration
69. bitfield_clause
70. enumerator
71. parameter_list
72. parameter_declaration
73. attributed_statement
74. labeled_statement
75. expression_statement
76. if_statement
77. switch_statement
78. case_statement
79. while_statement
80. do_statement
81. for_statement
82. return_statement
83. break_statement
84. continue_statement
85. goto_statement

86. comma_expression
87. conditional_expression
88. assignment_expression
89. pointer_expression
90. update_expression
91. cast_expression
92. type_descriptor
93. sizeof_expression
94. subscript_expression
95. field_expression
96. compound_literal_expression
97. initializer_list
98. initializer_pair
99. subscript_designator
100. field_designator
101. char_literal
102. concatenated_string
103. string_literal
104. placeholder_type_specifier
105. decltype
106. class_specifier
107. virtual_specifier
108. virtual_function_specifier
109. explicit_function_specifier
110. base_class_clause
111. dependent_type
112. template_declaration
113. template_instantiation
114. template_parameter_list
115. type_parameter_declaration
116. variadic_type_parameter_declaration
117. optional_type_parameter_declaration
118. template_template_parameter_declaration
119. optional_parameter_declaration
120. variadic_parameter_declaration
121. variadic_declarator
122. reference_declarator
123. operator_cast
124. field_initializer_list
125. field_initializer
126. default_method_clause
127. friend_declaration
128. access_specifier
129. abstract_reference_declarator
130. structured_binding_declarator

131. ref_qualifier
132. trailing_return_type
133. noexcept
134. throw_specifier
135. template_type
136. template_method
137. template_function
138. template_argument_list
139. namespace_definition
140. namespace_alias_definition
141. namespace_definition_name
142. using_declaration
143. alias_declaration
144. static_assert_declaration
145. concept_definition
146. condition_clause
147. for_range_loop
148. co_return_statement
149. co_yield_statement
150. throw_statement
151. try_statement
152. catch_clause
153. co_await_expression
154. new_expression
155. new_declarator
156. delete_expression
157. type_requirement
158. compound_requirement
159. requirement_seq
160. constraint_conjunction
161. constraint_disjunction
162. requires_clause
163. requires_expression
164. lambda_expression
165. lambda_capture_specifier
166. lambda_default_capture
167. fold_expression
168. parameter_pack_expansion
169. destructor_name
170. dependent_name
171. qualified_identifier
172. operator_name
173. user_defined_literal
174. field_identifier
175. namespace_identifier

176. simple_requirement
177. statement_identifier
178. type_identifier