

乱弹: CLI 中的编译原理

墨泪

2020-03-29

大纲

- iRedis 中的输入流解析
- 编译原理是什么
- iRedis 中的编译原理
- Q&A

iRedis の 输入流

核心疑问: iRedis 读取输入后, 是怎么进行后续处理的?

```
iredis 1.5.0 (Python 3.7.6)
redis-server 5.0.7
Home: https://iredis.io
Issues: https://iredis.io/issues
127.0.0.1:6379> set foo bar xx
(nil)
127.0.0.1:6379> get foo
(nil)
127.0.0.1:6379> set foo bar nx
OK
127.0.0.1:6379> get foo
"bar"
```

```
test_utils.py ×
tests > unittests > test_utils.py

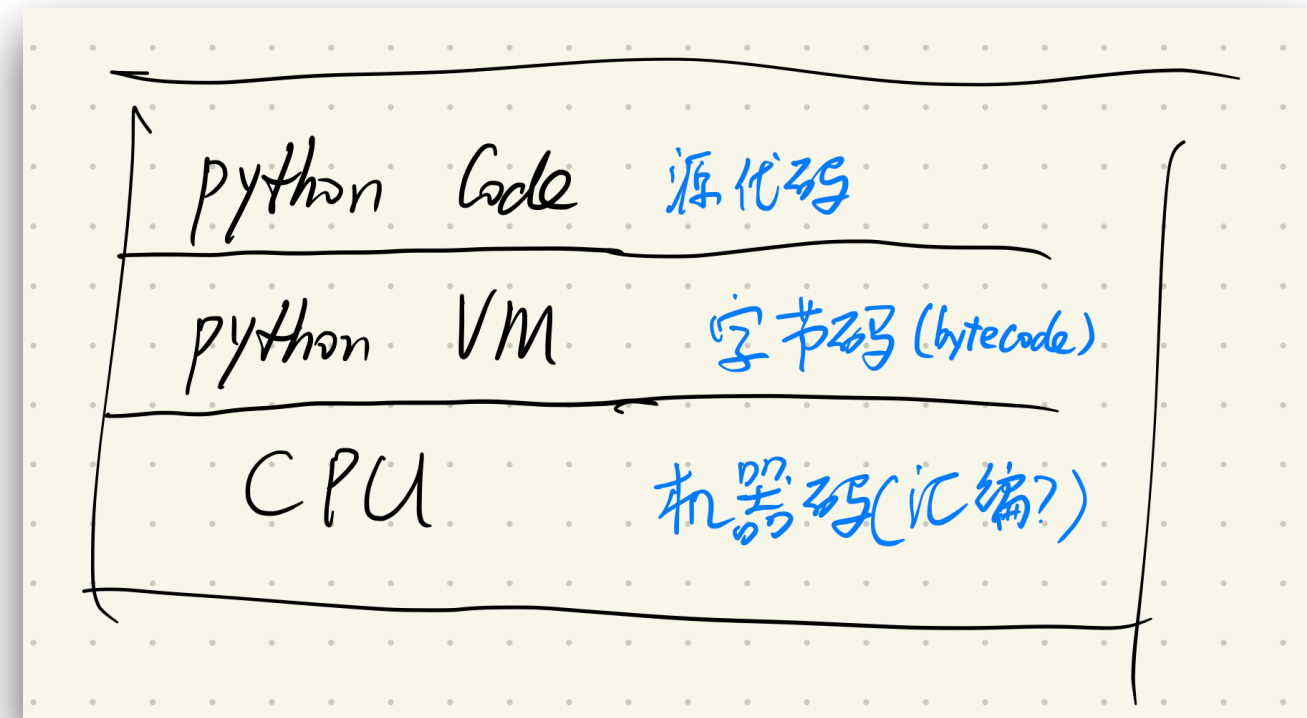
38 @pytest.mark.parametrize(
39     ["test_input,expected",
40     [
41         ("hello world", ["hello", "world"]),
42         ("'hello world'", ["hello world"]),
43         ('"hello"world"', ["helloworld"]),
44         (r'"hello"world"', [r"hello\world"]),
45         (r'"\\', [r"\\"]),
46         (r"\\", [r"\\"]),
47         (r"\\abcd ef", [r"\\abcd", "ef"]),
48         # quotes in quotes
49         (r'"'"'hello'world'"'"', ['hello'world']),
50         (r'"'"'hello'world'"'"', ["hello'world"]),
51         (r'"'"'hello\'world'"'"', ["hello'world"]),
52         (r'"'"'hello\'world'"'"', ['hello"world']),
53         (r"''", [""]), # set foo "" is a legal command
54         (r"''", [""]), # set foo "" is a legal command
55         (r"\\", [r"\\"]), # backslash are legal
56         (r"\\hello\\", [r"\\hello\\"]), # backslash are legal
57     ],
58 )
59 def test_stipe_quote_escaple_in_quote(test_input, expected):
60     assert list(strip_quote_args(test_input)) == expected
61
```

Case Num	Input	-> Output	-> Expected	Status
Case 0:	"hello world"	-> ["hello", "world"]	-> ["hello", "world"]	Success
Case 1:	"'hello world'"	-> ["hello world"]	-> ["hello world"]	Success
Case 2:	"hello\"world\""	-> ["helloworld"]	-> ["helloworld"]	Success
Case 3:	"hello\\\"world\\\""	-> ["hello\\world"]	-> ["hello\\world"]	Success
Case 4:	"\\\"\\\"\\\""	-> [r"\\\"\\\"\\\""]	-> [r"\\\"\\\"\\\""]	Success
Case 5:	"\\\"\\\"\\\""	-> [r"\\\"\\\"\\\""]	-> [r"\\\"\\\"\\\""]	Success
Case 6:	"\\\"abcd ef"	-> [r"\\\"abcd", "ef"]	-> [r"\\\"abcd", "ef"]	Success
Case 7:	" 'hello\"world' "	-> ["hello\"world"]	-> ["hello\"world"]	Success
Case 8:	" \"hello'world\" "	-> ["hello'world"]	-> ["hello'world"]	Success
Case 9:	" 'hello\\'world' "	-> ["hello'world"]	-> ["hello'world"]	Success
Case 10:	" \"hello\\\"world\" "	-> ["hello\"world"]	-> ["hello\"world"]	Success
Case 11:	"'"	-> [r"'""]	-> [r"'""]	Success
Case 12:	"\\\""	-> [r"\\\""]	-> [r"\\\""]	Success
Case 13:	"\\\"\\\""	-> [r"\\\"\\\""]	-> [r"\\\"\\\""]	Success
Case 14:	"\\\"hello\\\""	-> [r"\\\"hello\\\""]	-> [r"\\\"hello\\\""]	Success

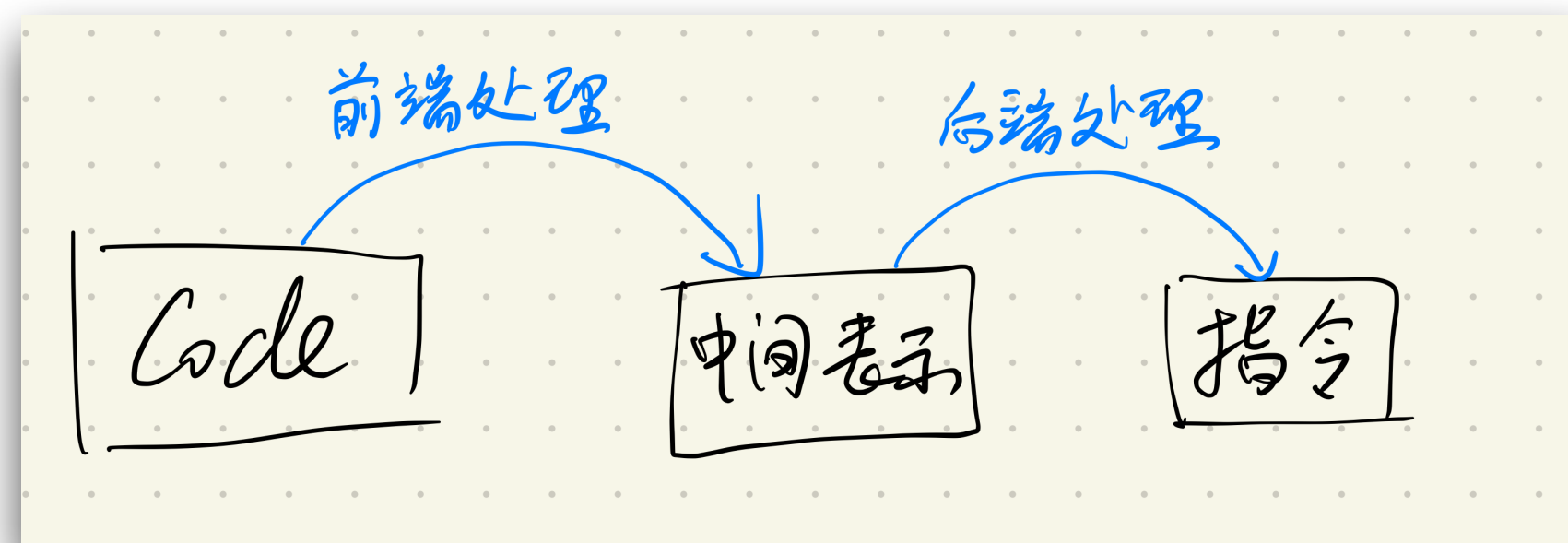
C926 J4:	"/p6jfo/"	-> ["/p6jfo/"]	-> ["/p6jfo/"]	Success
C926 J3:	"/" /	-> ["/" /]	-> ["/" /]	Success
C926 J5:	"/" /	-> ["/" /]	-> ["/" /]	Success
C926 J7:	"/" /	-> ["/" /]	-> ["/" /]	Success
C926 J8:	"/p6jfo/" /	-> ["/p6jfo/" /]	-> ["/p6jfo/" /]	Success

编译原理简介

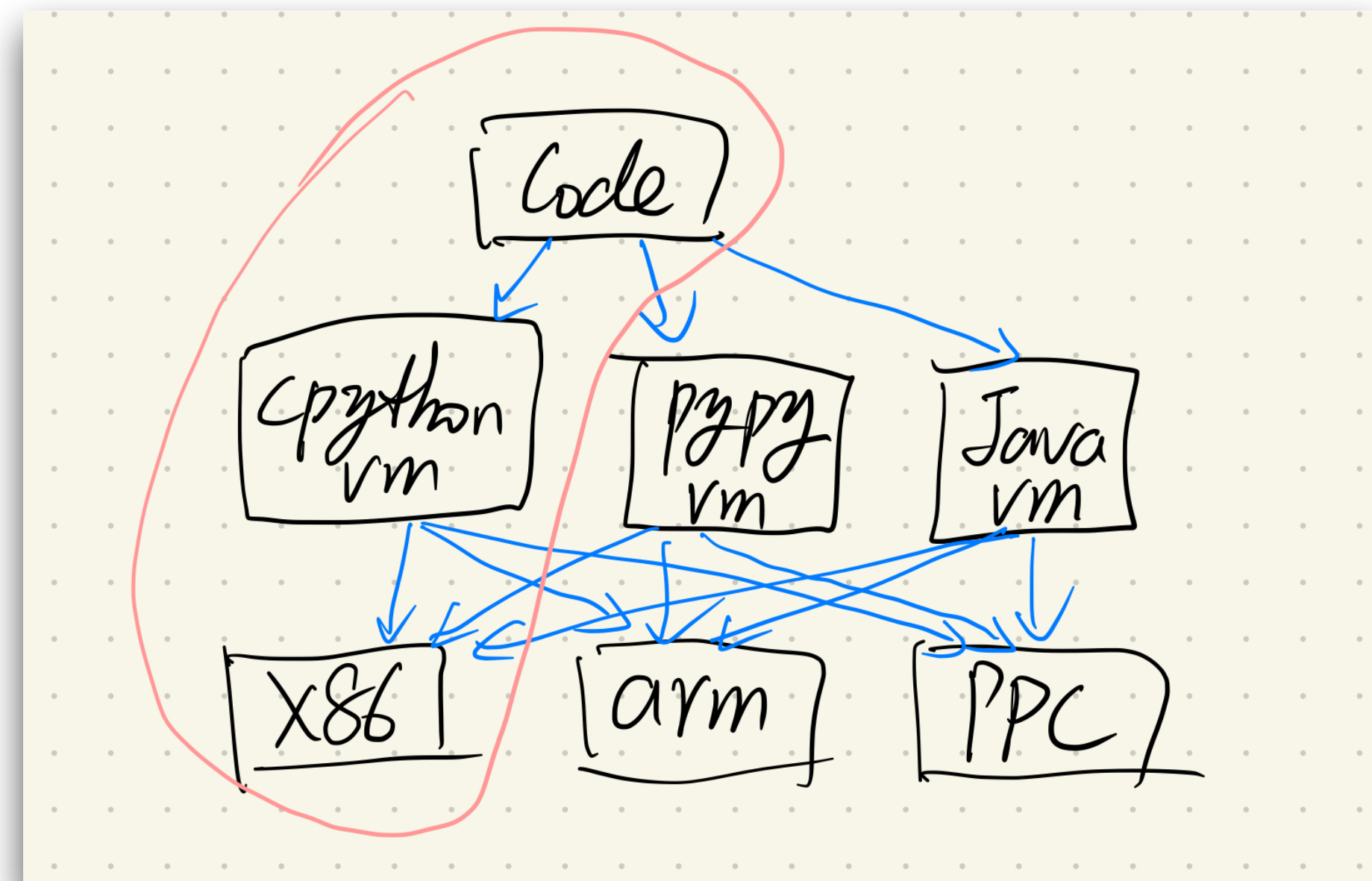
- 编译原理: 是一种“翻译”过程的系统化抽象原理



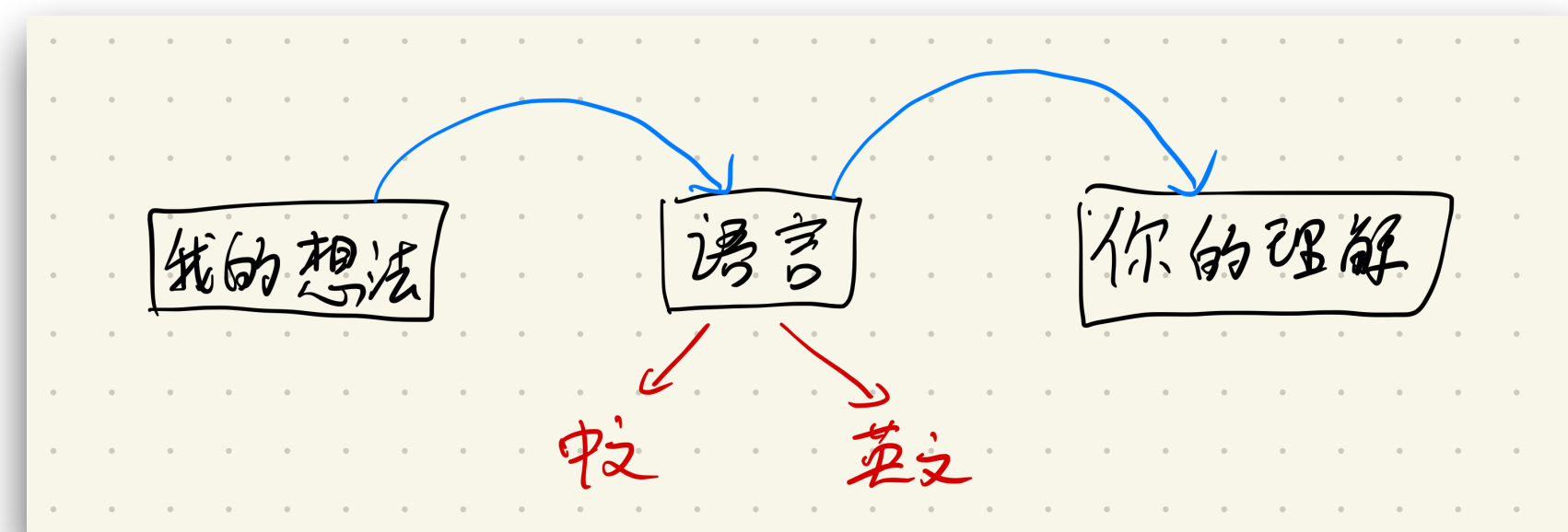
Python 语言编译架构



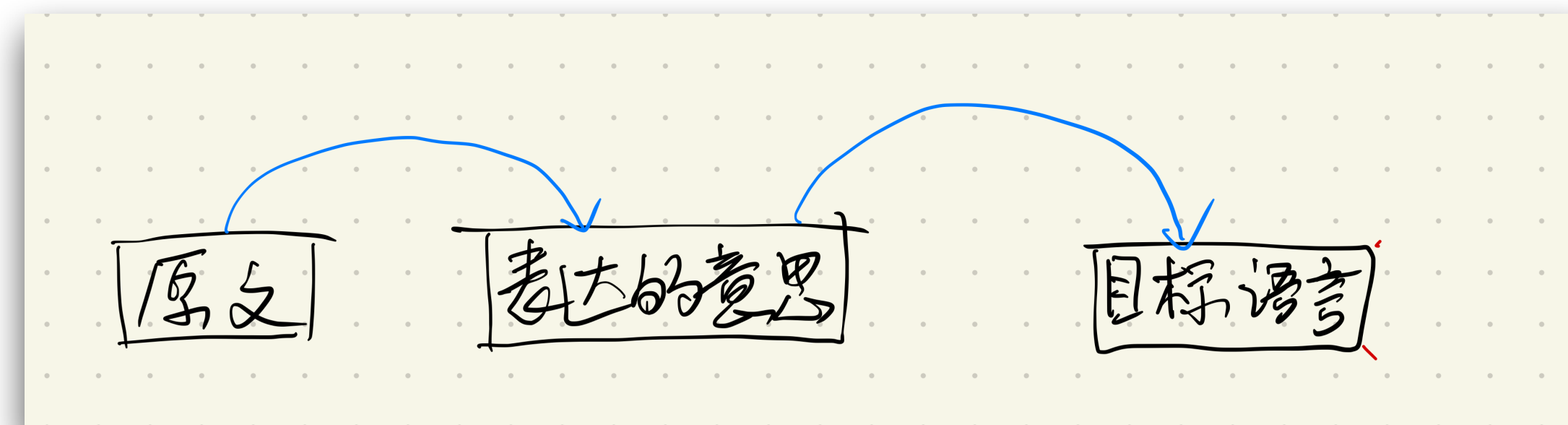
编译原理抽象过程



生活中的编译原理

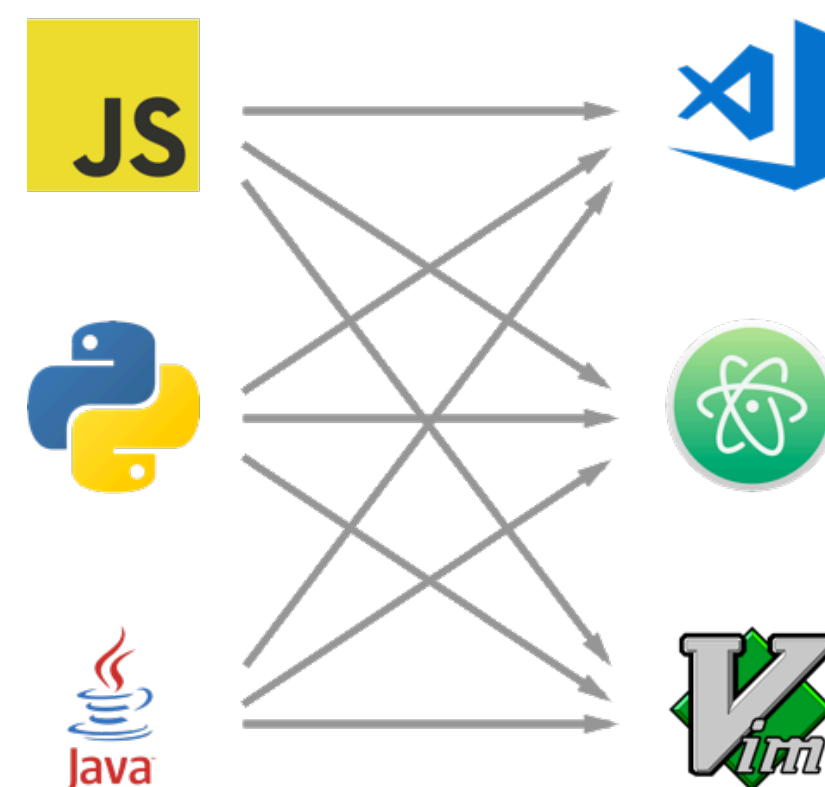


人和人之间的沟通交流

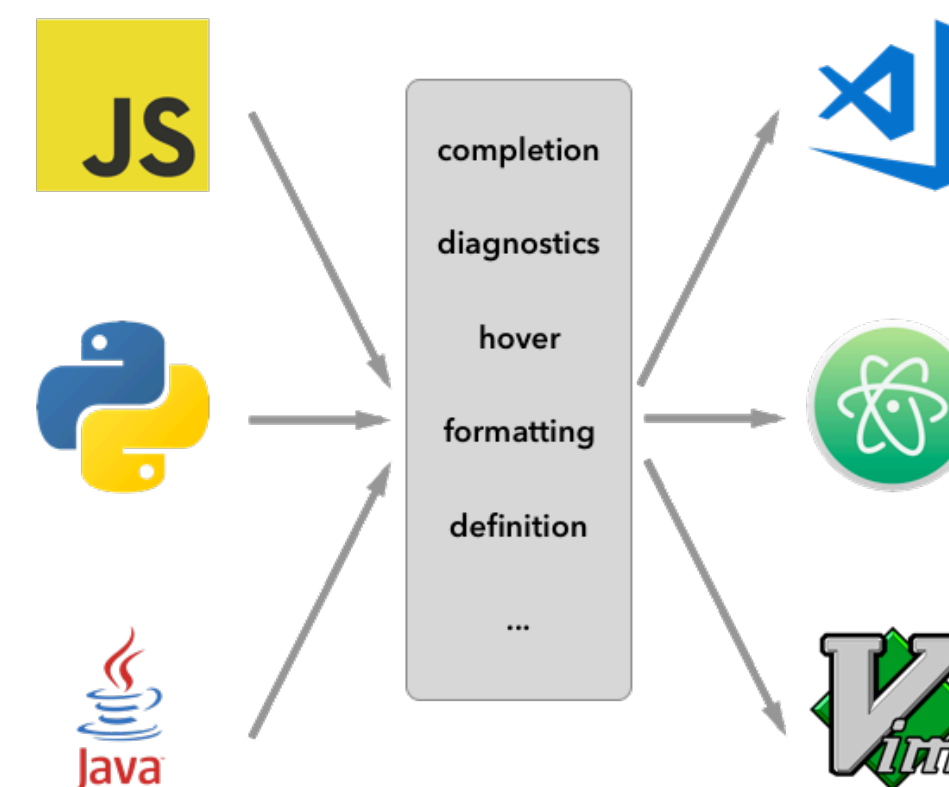


语言和语言之间的翻译

NO LSP



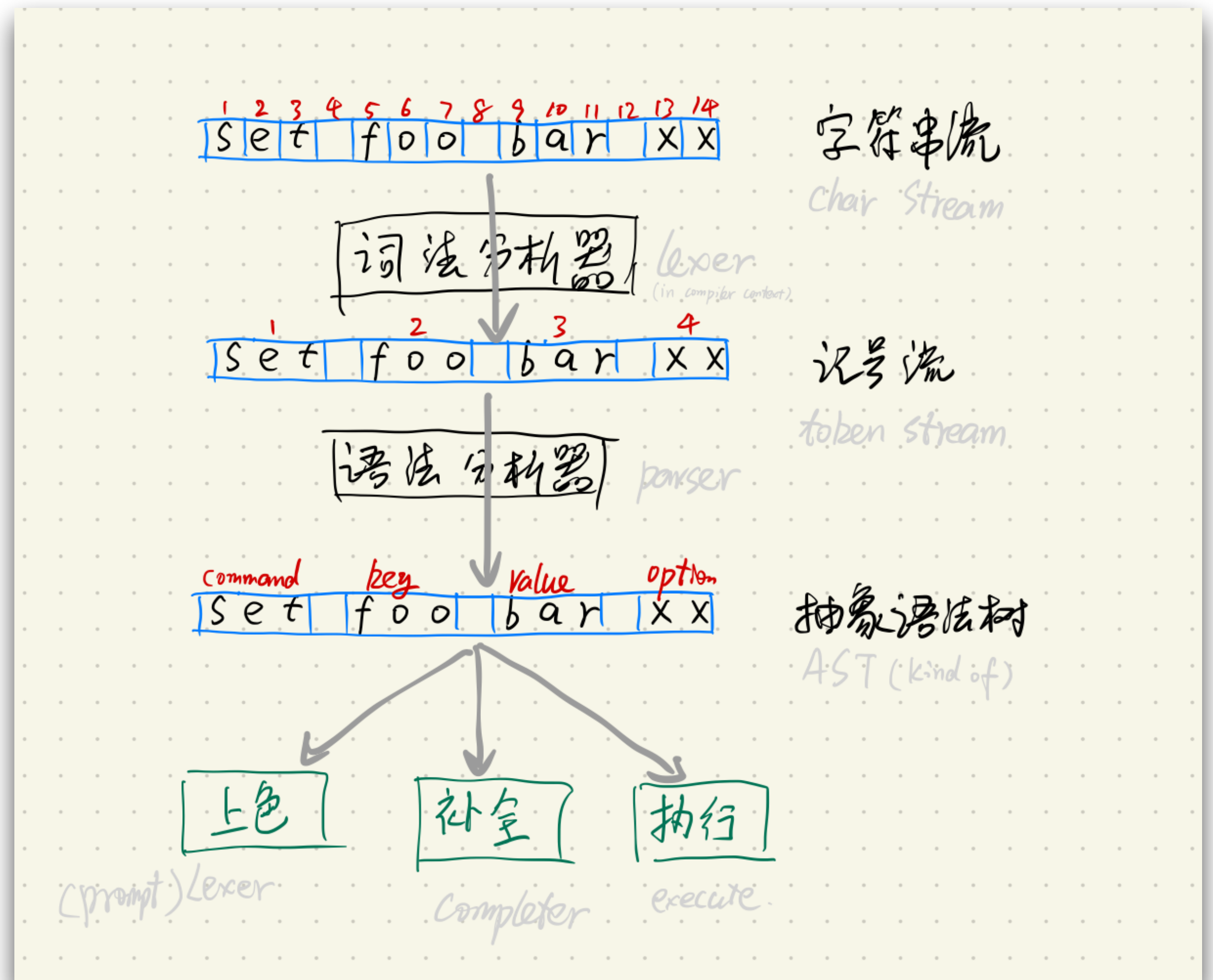
LSP



Language Server

iRedis 中的编译原理

```
iredis 1.5.0 (Python 3.7.6)
redis-server 5.0.7
Home: https://iredis.io
Issues: https://iredis.io/issues
127.0.0.1:6379> set foo bar xx
(nil)
127.0.0.1:6379> get foo
(nil)
127.0.0.1:6379> set foo bar nx
OK
127.0.0.1:6379> get foo
"bar"
```



iRedis 命令解析实现

```
utils.py x
iredis > utils.py > strip_quote_args

47 def strip_quote_args(s):
48     """
49     Given string s, split it into args. (Like bash paring)
50     Handle with all quote cases.
51
52     Raise ``InvalidArguments`` if quotes not match
53
54     :return: args list.
55     """
56     word = []
57     in_quote = None
58     pre_back_slash = False
59     for char in s:
60         if in_quote:
61             # close quote
62             if char == in_quote:
63                 if not pre_back_slash:
64                     yield "".join(word)
65                     word = []
66                     in_quote = None
67             else:
68                 # previous char is \, merge with current
69                 word[-1] = char
70         else:
71             word.append(char)
72             # not in quote
73         else:
74             # sperator
75             if sperator.match(char):
76                 if word:
77                     yield "".join(word)
78                     word = []
79             # open quotes
80             elif char in ['"', "'"]:
81                 in_quote = char
82             else:
83                 word.append(char)
84             if char == "\\" and not pre_back_slash:
85                 pre_back_slash = True
86             else:
87                 pre_back_slash = False
88
89     if word:
90         yield "".join(word)
91     # quote not close
92     if in_quote:
93         raise InvalidArguments("Invalid argument(s)")
94
```

缓冲区
状态机
引号处理
状态机转换

- 按字符读入缓冲区
- 空格在 'in_quote' 状态机下的处理
- 转义符在各种状态下的处理
- 最终完成了记号流的输出

为什么不用 Shlex

- 场景不同: iRedis 在每一个字符键入时进行解析
- 标准不同: redis-cli 的解析标准和 bash 不一致

```
>>> shlex.split("set foo 'bar ")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/System/Library/Frameworks/Python
    return list.lex)
  File "/System/Library/Frameworks/Python
    token = self.get_token()
  File "/System/Library/Frameworks/Python
    raw = self.read_token()
  File "/System/Library/Frameworks/Python
    raise ValueError, "No closing quotation
ValueError: No closing quotation
```

不允许未关闭的引号字符

```
[ruohan.chen@Ruohans-15Inch-MBP /Users/ruohan.chen]
$ redis-cli
127.0.0.1:6379> set key \hello\
OK
127.0.0.1:6379> get key
"\hello\"
127.0.0.1:6379>

ValueError: No closing quotation
>>> shlex.split("\hello")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/System/Library/Frameworks/Python.framework/Versions/2.7
    return list.lex)
  File "/System/Library/Frameworks/Python.framework/Versions/2.7
    token = self.get_token()
  File "/System/Library/Frameworks/Python.framework/Versions/2.7
    raw = self.read_token()
  File "/System/Library/Frameworks/Python.framework/Versions/2.7
    raise ValueError, "No escaped character"
ValueError: No escaped character
```

不允许转义符号后为空

The image shows two terminal windows. The top window is a Redis CLI session. The first command is `127.0.0.1:6379> set key \abc`, where `\abc` is boxed in red. The response is `OK`. The second command is `127.0.0.1:6379> get key`, where the response `'\\abc'` is boxed in red. A red arrow points from the boxed `\abc` in the first command to the boxed `'\\abc'` in the second command. To the right of this arrow, there is handwritten red text: `还有\"`. The bottom window is a Python REPL session. The first command is `>>> shlex.split(r\"abc\")`, where `r\"abc\"` is boxed in red. The response is `['abc']`, where `'abc'` is boxed in red. A red arrow points from the boxed `r\"abc\"` to the boxed `'abc'`. Below this, there is a second `>>>` prompt followed by a cursor, and handwritten red text `没了` (meaning 'nothing more' or 'ended').

```
127.0.0.1:6379> set key \abc
OK
127.0.0.1:6379> get key
'\\abc'
127.0.0.1:6379>
```

```
>>> shlex.split(r\"abc\")
['abc']
>>>
```

无意义转义时的理解不同

演示环节

- 介绍 SLY(Sly Lex Yacc)
- 基于 SLY 写的 iRedis “编译器” 走读

Q&A