Python 解释器

1. 做什么

语法文件讲了什么

- 学会阅读正则表达式
- Lexer: NAME,NUMBER,STRING... 匹配词法 终止符（叶结点）
- Parser: test,testlist,atom,and_test... 匹配语法 可展开

第二次机考第三题`4285. 乔鲁诺乔巴拿的梦想`，大写字母A->parser，小写字母a->lexer

antlr做了什么

字符流-->Token流-->语法树（多叉树）

```
ANTLRInputStream input(ifs);
Python3Lexer lexer(&input);     //字符流->Token流
CommonTokenStream tokens(&lexer);
tokens.fill();
Python3Parser parser(&tokens);
tree::ParseTree* tree=parser.file_input();  //Token流->语法树
EvalVisitor visitor;
visitor.visit(tree);    //遍历语法树
```

语法树的每一个结点都有对应于语法文件中对应parser的类型

```
term: factor (muls_op factor)*;
```

```
antlrcpp::Any visitMuls_op(Python3Parser::Muls_opContext *ctx) override;

antlrcpp::Any visitTerm(Python3Parser::TermContext* ctx) override;

antlrcpp::Any visitFactor(Python3Parser::FactorContext* ctx) override;
```

term->visitTerm

...

一堆xxx_test通过控制语法树的结构隐式处理了运算符优先级

结点无环，结点类型可能有环：testlist->test->.....->atom->test->.....

我们需要做什么

实现Evalvisitor，通过实现 *Evalvisitor中* 继承了Python3BaseVisitor后 *重载的函数* 对树进行一次DFS

2.代码细节

ctx是什么

```
Python3Parser::TermContext* ctx
```

<一个parser>Context 代表着对应类型结点所拥有的所有信息，包括它自身与以它为头结点的子树的信息

常用

```
ctx->getText() // return std::string
ctx-><某个parser>() // 语法文件中这个parser只出现一个 return parserContext的类型*
ctx-><某个parser>() // 此parser出现多个，return std::vector<parserContext的类型*>
连续调用
ctx->or_test()->and_test()[0]->not_test()[0]->comparison() // ComparisonContext*
```

怎么访问子结点

```
ctx-><某个parser>()
ctx-><某个parser>()[0]
```

的注意：返回的vector是现场构造的，若使用迭代器遍历不可使用

```
for(auto iter=ctx->test().begin();iter!=ctx->test().end();++iter)
```

此种形式

antlrcpp::Any 是什么

antlr为实现万能返回类型定义的类

利用指针存储对应类型的值（Any本质为封装好的void指针）

使用方法：

```
if(any_value.is<T>) // determine if any_value can be converted to T type
T orz=any_value.as<T> // convert any_value to a T type value
```

3.实例解释

- 自动访问

  ```
  flow_stmt: break_stmt | continue_stmt | return_stmt;
  ```

  当我们不需要特别地做些什么时

  ```
  antlrcpp::Any EvalVisitor::visitFlow_stmt(Python3Parser::Flow_stmtContext*
  ctx) {
    return visitChildren(ctx);
  }
  ```

- 手动访问

```
arith_expr: term (addsub_op term)*;
addsub_op: '+'|'-';
```

假设visitTerm返回Object

```
Object result = visitTerm(ctx->term()[0]);
for (int i = 0, end_n = ctx->addsub_op().size(); i < end_n; ++i) {
  if (ctx->addsub_op()[i]->getText() == "+") {
       result += visitTerm(ctx->term()[i + 1]);
  }
}
```

4. 自己修改g4文件

`antlr4 -visitor -Dlanguage=Cpp Python3.g4`

修Python3Lexer.h:

- 开头加入

```
#include "Python3Parser.h"
#include <regex>
```

修Python3Lexer.cpp

- 16行

```
Python3Lexer::~Python3Lexer() {
    delete _interpreter;
    for(auto token:tokens){
        delete token;
    }
}
```

- 115行`Python3Lexer::NEWLINEAction()`函数删去

```
default:
    break
```