神奈川大学理学部　情報科学科　馬谷研究室

Lox インタプリンタで用いられている Visitor パターンの調査

高橋疾風 201903020

1　はじめに

本調査では Lox を使用する. そこで用いられている Visitor パターンについて調査を進めていく. 本調査の目的は Lox インタプリンタで Visitor パターンがどのように用いられているかを調査し、用いられている事象を明確にし、理解を深め、Visitor パターンは、コンパイラ構築において広く用いられている. 特定のプログラムのシンタックスツリーが与えられると、そのツリー上で動作する多くのビジターが記述される. 今後 Visitor パターンを用いたプログラムを作りたいと考えている為、Lox インタプリンタで用いられている Visitor パターンにおける知識の確認と、それにおける Visitor パターンについて考える. その為に文献を用いて調査を行う.

2　Lox 言語

まず Lox について説明する.Lox は命令型、動的型付け言語で特徴として、動的な型付けが可能であり、ガベージコレクション、参照カウントといった自動メモリ管理がある. この言語で使用されている Visitor パターンについて調査する.

3　Visitor パターン

Visitor パターンとはデザインパターンの一つであり、データ構造と処理を分離することが出来る. データ構造の中に多くの要素が格納されており、その各要素に対して、何らかの処理をしたい時、処理のコードはどこに書けばいいか、データ構造を表しているクラスの中に書くとすると新しい処理が必要になるたびに、データ構造のクラスを修正しなければならないといった問題を解決するために、Visitor パターンが使われている.Visitor パターンでは、「処理」を訪問者である Visitor オブジェクトに記述することで、処理の追加を簡単にする. 処理対象となる Acceptor オブジェクトは、Visitor オブジェクトを受け入れる accept() メソッドを実装している必要がある.Visitor パターンは、例えばコンパイラ構築において広く用いられている. 特定のプログラムのシンタックスツリーが与えられると、そのツリー上で動作する多くのビジターが記述される. ここでは式の評価と文の実行を調査していく. この二つはプログラミングにおいてよく使われているので、Visitor パターンでの使い方も紹介する.

4　式の評価

Lox では、値はリテラルによって作成され、式によって計算され、変数に格納される. visitor メソッドの戻り値の型は、Java コードで Lox 値を参照するために使用するルートクラスである Object になる.Visitor インターフェイスを満たすには、パーサーが生成する四つの表現木クラスそれぞれに visitor メソッドを定義する必要がある. 以下のクラスは Visitor であることを宣言している.

```
package com.craftinginterpreters.lox;
class Interpreter implements
Expr.Visitor<Object> {
}
```

ここで visitor メソッドを使い、リテラル、単項式、グループ化、二項演算子比較演算子を制作していくことで visitor インターフェイスを満たせる. リテラルの構文木ノードを実行時の値に変換する. そして、return expr.value; で返す.

return evaluate(expr.expression); でグループ化式自体を評価するために、式を再帰的に評価してそ

れを返す. 単項式は、まず、オペランド式を評価する. そして、その結果に単項演算子そのものを適用する. 評価がどのように再帰的にツリーを走査するのかがわかるようになる. 二項演算子は評価するオペランドが二つある. 算術演算と同じです。唯一の違いは、算術演算子がオペランドと同じ型（数値や文字列）の値を生成するのに対して、比較演算子は常にブール値を生成することである. 等式は数字を必要とする比較演算子とは異なり、等値演算子はあらゆる型のオペランドをサポートし、混合型もサポートする。Lox に 3 が ”three ”より小さいかどうかを尋ねることはできないが、それと等しいかどうかを尋ねることはできる.

5 文の実行

新しい Visitor インターフェース Stmt.Visitor を実装する. インタープリタが実装するインターフェースのリスト追加する.

```
class Interpreter implements

Expr.Visitor<Object>,

Stmt.Visitor<Void> {
```

ステートメントは値を生成しない為、visitor メソッドの戻り値は Object ではなく Void になる.2 つのステートメントタイプがあり、それぞれに対して visitor メソッドが必要.2 つの新しい構文木がある為、2 つの新しい visitor メソッドになる.1 つ目は、宣言文のためのものであり、もし変数に初期化子があれば、それを評価する。そうでない場合は、別の選択をすることになる.

式文は既存の evaluate() メソッドで内部式を評価し、その値を破棄する. そして、NULL を返す.

```
 @Override
public Void visitExpressionStmt
(Stmt.Expression stmt) {

evaluate(stmt.expression);
        return null;
}
@Override
public Void visitPrintStmt
(Stmt.Print stmt) {
  Object value =
    evaluate(stmt.expression);
  System.out.println(stringify(value));
    return null;
 }
```

式の値を破棄する前に、 stringify() メソッドで文字列に変換し、s 標準出力にダンプしてる. こうすることによって、インタープリタは文を訪問できるようになる.

6 まとめ

Visitor パターンは受け入れる側に処理を追加することなく、処理を追加することができるパターンであり、Visitor パターンを使う場合としては、集合に、型（クラス）が異なる要素が含まれている。要素の型（クラス）に応じて、行う処理が異なる。ある要素から、次の要素に辿る際、次の要素の型（クラス）が分からない。といった時に使用すると有用である. また、式の評価、文の実行だけでなく様々な用途がある為、これからも引き続き調査を続けていき理解を深めて、プログラムの知識を広げていき、Visitor パターンを用いたプログラムを作っていきたい.

参考文献

[1] Bob Nystrom, craftinginterpreters a tree-walk interpreter
http://craftinginterpreters.com/a-tree-walk-interpreter.html