神奈川大学理学部　情報科学科　馬谷研究室

CPython 処理系の中間言語インタプリタの調査

小川慶 201903005

1　はじめに

本調査では CPython を使用する。CPython とは C 言語を使って書かれた Python 処理系であり、最もよく使われている Python 実装で、オリジナルの Python 処理系となっている。そこで用いられている中間言語インタプリタについて調査を進めていく。本調査の目的は CPython 処理系の中間言語インタプリタがどのように動作しているのかを調査し、行われている事象を明確にすることと、それにより理解を深めることにある。主に行うことは本題でもある CPython 処理系の中間言語インタプリタにおける知識の確認である。それを踏まえ、文献をもとに調査を行っていく。

2　前提

まず、コードオブジェクトは機械語による命令を並べたプログラムであり、そこにはバイトコードの形で個別の操作のリストが含まれている。CPython インタプリタはマーシャリングされた.pyc ファイルまたはコンパイラから取得したコードオブジェクトを評価し実行する。CPython でのコードの実行は、評価ループと呼ばれる中央のループの中で行われる。本調査では、その評価ループについて詳しく解説する。

また、評価ループにたどり着く以前に、Python のコードがどのような工程を踏んで抽象的な構文木に解析され、さらに、コードオブジェクトにコンパイルされていくかは次節でおおまかに説明していく。

3　CPython

ここでは、Python 言語の標準の処理系である CPython でソースコードを抽象構文木に解析される方法と、抽象構文木の構造について説明する。

まず、テキストで書かれたプログラムは読み取られ、字句解析器に渡される。渡されたプログラムは字句解析器を使用して構文木を作成する。その構文木を構文解析器に渡し、抽象構文木と呼ばれる木構造のデータを作成する。抽象構文木はコンパイラに渡されると、そこで複雑な構造をトレースし、制御フローグラフを作成する。そして、この制御フローグラフがアセンブラによってバイトコードに変換されることによってプログラムが実行可能になる。

今回は、プログラムが実行可能になったその後、実際に実行が行われるフェーズである中間言語インタプリタについて調査していく。

4　評価ループ

前述とおり、CPython でのコードの実行は、評価ループの中で行われ、そこでは、コードオブジェクトを受け取り、一連のフレームオブジェクトに変換する。

まず、スレッドステートの構築が行われる。フレームが実行される前には、スレッドにリンクされる必要がある。CPython は一つのインタプリタ内で多くのスレッドを同時に走らせることができる。インタプリタの状態には、それらのスレッドのリンクリストが含まれる。CPython は常に少なくとも一つのスレッドを持ち、各スレッドはそれ自身の状態を持つ。

そして、フレームオブジェクトの構築が行われる。コンパイルされたコードオブジェクトは、フ

レームオブジェクトに挿入される。フレームオブジェクトは Python の型であるため、C と Python の両方から参照することができる。フレームオブジェクトには、コードオブジェクトの命令を実行するために必要な他のランタイムデータも含まれる。そのデータには、ローカル変数、グローバル変数、組み込みモジュールが含まれている。

フレームオブジェクトの構築が完了すると、フレームスタックと呼ばれるスタックで実行が行われる。コードオブジェクトには実行されるバイトコードのバイナリーエンコーディングに加え、変数のリストとシンボルテーブルも含まれている。ローカル変数とグローバル変数は、関数、モジュール、ブロックがどのように呼び出されたかに基づいて、実行時に決定される。また、`_PyEval_EvalFrameDefault()` というデフォルトのフレーム評価関数がある。これは CPython にバンドルされている唯一のオプションとなっている。そして、`PyEval_Eval FrameEx()` という関数により、インタプリタが設定したこのフレーム評価関数が呼び出される。CPython で実行されるものはすべてこのフレーム評価関数を通過する。これらの関数がすべてをまとめており、コードを有効なものにしている。

5 バリュースタック

次に、変数が作成、変更され、コンパイルされたコードオブジェクトのバイトコード操作によって使用される場所となる、バリュースタックと呼ばれる概念について紹介していく。

評価ループの内部では、バリュースタックが作成される。これは、ポインタのリストであり、変数のような値であったり、関数（Python ではオブジェクト）への参照であったり、その他の Python オブジェクトであったりする。また、評価ループ内のバイトコード命令は、バリュースタックから入力を受け取る。ここで、バイトコード操作を例をもって以下に示す。

例えば、Python で `or` 文を挿入したとする。

```
if left or right:
    pass
```

コンパイラはこの `or` 演算を `BINARY_OR` 命令にコンパイルする。

```
static int
binop (struct compiler *c, operator_ty op)
{
    switch (op) {
    case Add:
        return BINARY_ADD;
    ・・・
    case BitOr:
        return BINARY_OR;
```

評価ループでは、`BINARY _ OR` の場合、バリュースタックから 2 つの値、左演算と右演算を取り、それらの 2 つのオブジェクトに対して `PyNumber_Or` を呼び出す。

```
・・・
case TARGET (BINARY_OR): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Or (left, right);
    Py_DECREF(left);
    Py_DECREF (right);
    SET_TOP (res);
    if (res == = NULL)
        goto error;

    DISPATCH();
}
```

結果である `res` は、現在のトップ値を上書きしてスタックのトップとして設定される。

6 まとめ

本研究では CPython 処理系の中間言語インタプリタの調査を行った。コードの実行は、評価ループと呼ばれる中央のループの中で行われており、コンパイルされた Python コードと、その下にある C 拡張モジュール、ライブラリ、およびシステムコールの間のインターフェイスになっている。

参考文献

[1] Anthony Show, CPython Internals: Your Guide to the Python 3 Interpreter, Real Python, 2021.