BRAD CYPERT

Clojure

LispのエレガントさをJVMのパフォーマンスで

Clojureは、JVMベースの汎用的かつ動的な関数型プログラミング言語です。スクリプト言語の手軽さとインタラクティブ開発を、マルチスレッド・プログラミング向けの効率的で安定したインフラストラクチャと組み合わせている点が特徴です。筆者は、Clojureで初めてLisp方言と言われる言語に触れました。よく耳にする不満の中には丸括弧の多用があるものの、言語自体については、美しく書くことができ、エレガントに読むことができるものだと感じました。Clojureによる、おなじみのHello Worldコードは次のようになります。

```
(ns demo.hello-world)
; これはコメントです
(println "Hello World")
```

このコードを実行するためには、ClojureのJARファイルが必要です。このJARファイルは、こちらから、またはMavenから取得できます。JARファイルをダウンロードしたら、先ほどのコードをhello.cljに保存して、次を実行します。

```
java -cp clojure.jar clojure.main hello.clj
> Hello World
```

以降のソースの例で、式の下にあるコメントは、その式からの出力または戻り値を示しています。上記の例では、demo.hello-worldという新しいネームスペースを定義し、そのネームスペースでHello Worldを出力しています。とても簡単なサンプルですね。

これは貴重な第一歩ですが、文を出力するだけでは大半の言語の力を示すことにはなりません。Clojureも同様です。もう少し難しい問題を扱う方が適切でしょう。そこで、整数のリストに含まれるすべての整数を2倍して合計

を求める場合を考えてみます。これは、単純なマップ/リデュース問題だと気づいた方もいらっしゃるかもしれません。Clojureでは、まさにその考え方を構文で表現できます。

```
(ns demo.hello-lists)

(def my-list '(1 2 3 4 5))

(reduce
  +
  (map #(* % 2) my-list))
; 30
```

Clojureでは、キーワードという考え方がサポートされています。これをJavaなどの他の言語における予約キーワードという考え方（staticやfinalなど）と混同しないようにしてください。

上記のコードは、いくつかのシンプルな考え方を実際に示したものです。

- まず、値1、2、3、4、5が格納された新しいリストを定義します。Clojureのリストは、一重引用符の後に、リストに格納する値を丸括弧のペアで囲んで記述することによって定義できます。
- 次に、リデュース操作を行う式を記述します。reduceは、関数とリストを受け取ります。ここでは、+記号にバインドされている関数と、map関数から返されるリストを渡しています。
- 最初の行で定義したリストを使い、リストの各値に対してそれぞれを2倍した値をマップします。mapの最初の引数には、匿名関数を渡しています。%は、そこに引数が渡されることを示すプレースホルダです。複数の引数がある場合は、%1や%2のようにインデックスを付けて記述できます。

ほとんどの関数型言語と同様に、Clojureも問題に対して大まかなアプローチをとっています。関数型言語は、多くの場合、個々の手順に対して処理を行うコードが必要になるというよりは、リストの反復処理といった実装の詳細は気にせずに、解決しようとしている問題を説明するコードを書く方法だとみなされています。

Java開発者のためのClojure

Clojureは、コンパイル言語であるものの、強力なスクリプト言語のように感じられる、ある種のダイナミックさをJVMにもたらすものです。Clojureのすべての機能は、コンパイル時だけでなく、実行時にもサポートされます。さらに、Clojureを使う場合、お気に入りのJavaライブラリを使えるという快適さを捨てる必要はありません。ClojureはJavaの相互運用性を完全にサポートしているため、既存のJavaコードを活用できます。つまり、完全にコードを書き直さなくてもClojureを試してみることができるということでもあります。JavaコードとともにClojureコードを使うこ

とができ、その逆も可能です。

基本は不変性

関数型プログラミング言語であるClojureには、データ構造の不変性と永続性に関する豊富な機能が搭載されています。可変状態が必要な場合は、Clojureのソフトウェア・トランザクション・メモリ・システムとリアクティブ・エージェント・システムを活用して、クリーンかつ正確なマルチスレッド設計を実現できます。

マルチスレッドと言えば、Clojureでは標準ライブラリの一部としてcore.asyncという非同期ツールキットが提供されています。Clojureは大まかにGoの同時実行性モデルに基づいているため、Clojureを使うことによってチャネルを使いこなせるようになります。Clojureでチャネルを使うと、チャネルからの値の取り出しと、チャネルへの値の格納という主に2つのことができます。

Clojureの非同期処理を深く理解するためには、1冊の本まではいかなくとも、1つの記事が必要になるほどです。ここでは非同期処理について深く取り上げることはしませんが、Clojureの非同期ライブラリはスレッドよりもエレガントな抽象化を行う方法であることは強調しておきたいと思います。

ツールとコミュニティ

Clojureコミュニティの先頭に立っているのは、Clojureの発案者で、CognitectのCTOであるリッチ・ヒッキー氏です。ヒッキー氏はベテランのJava開発者で、評判の高い講演を数多く行っています。中には、有名な「Simple Made Easy」という講演もあります。実際、ライブラリの構築、初心者の援助、創造力の育成などにおいて、コミュニティ全体でこの「Simple Made Easy」という考え方が取り入れられています。この考え方を意識すると、Clojureコミュニティで「フレームワーク」が避けられる傾向にあることに気づくでしょう。初心者が大きなフレームワークを提示されることはあまりないでしょう。それよりも、ニーズに合わせていくつかのライブラリを組み合わせるべきだという助言を受けることの方がはるかに多いはずです。Clojure言語の初心者、大規模なフレームワークを使ってきた方は特に、このことに対して不安を感じるかもしれません。しかし、少しばかり時間をかけて努力すれば、解決方法を探していた問題を解決できるだけでなく、Clojure自体を深く理解できることでしょう。

非常によく使用されているのは、Clojureの事実上の標準ビルド・ツールであるLeinengenです。このツールは、コミュニティが構築したもので、JavaでGradleが解決したものと同様の問題を解決することを狙ったものですが、実際はそれ以上のことを行ってくれます。Leinengenには、プロジェクトをすぐに開始できるテンプレート・プロジェクトが搭載され、プロジェクトのアドオン（プロファイルと呼ばれます）のサポートも提供されています。たとえば、PostgreSQLがサポートされたluminusテンプレートを土台にして新しいプロジェクトを作りたい場合、ターミナ

ルを開いて次のコマンドを実行すればいいだけです。

```
lein new luminus myapp +postgres
```

覚えておいていただきたいのは、各テンプレートには独自のプロファイルが含まれているため、別のテンプレートを使う場合は、postgresプロファイルをベースにできない場合があることです。

Leinengenは、build.gradleファイルやpackage.jsonファイルに相当するproject.cljファイルを作成してくれます。このファイルの中で、依存性の定義、プロジェクトのメタデータの設定、Leinengenプラグインの利用、ビルド・パイプラインの作成を行うことができます。

また、Leinengenは、豊富な機能を持つRead-Eval-Print Loop（REPL）も備えています。PythonやNodeを使って開発を行っている方なら、Python統合開発学習環境（IDLE）やNode REPLでさまざまなアイデアを試してみたことがあるでしょう。LeinengenのREPLも同じようなものです。Leinengenをインストールすると、コマンドラインからlein replを実行するだけで、新しく思いついたことをClojureで簡単にテストできます。クリーンなClojure環境が開き、そこで新しいアイデアを試してみることができます。

Clojureでのテスト

Clojureには、clojure.testというネームスペースを持つ軽量テスト・ライブラリが付属しています。このライブラリの中核をなすのが、任意の式のアサーションを定義できるisマクロです。Clojureでは、2つのパターンでテスト・ケースを記述できます。1つ目は、with-testマクロを使ってソース・コード内にインラインでテスト・ケースを記述する方法です。2つ目は、deftestマクロを使って別のネームスペースにテストを配置する方法です。シンプルな算術式のテストを次のようにして容易に行うことができます。

```
(ns demo.hello-tests)

(deftest math-tests
  (is (= 7 (+ 4 3)))
```

テストとは本来、単にコンピュータのもっとも基本的な処理の境界値をテストするものではなく、ビジネス・ロジックをテストするものです。もう少し的確な例として、独自のネームスペースで実行されるテスト・ケースが考えられま

す。次に例を示します。

```
(ns app.test.modules.users
  (:require [clojure.test :refer :all]
            [app.modules.users :refer [get-user]]))

(deftest user-tests
  (testing "Fetching a user"
    (let [user (get-user 1)]
      (is (some? (:email user))))))
```

上記のコードは、今まで紹介してきたものの中で一番複雑であるため、詳しく見てみることにします。ここでは、新しいネームスペースを設定して1件のテストを定義しています。そのテストでは、キーに1を指定してget-userを呼び出したときに、電子メールが定義されているデータ構造が返されることを確認しています。なお、get-userは、「email」というキーとメールアドレスの値で構成される、キーと値のペアを返すことに注意してください。

ここでは、以下のことを行っています。

- 新しいネームスペースapp.test.modules.usersを定義します。
- このネームスペースでは、いくつかのインポートを行います。これは、clojure.testとapp.modules.usersにアクセスする必要があるためです。ここではclojure.test内の関数をすべて参照していることにお気づきになると思いますが、app.modules.usersで行っているように、テストしようとするもののみを公開することも可能です。
- user-testsという新しいテストを定義します。
- testingマクロを使って新しいスコープを作成し、そのテスト・コンテキストにFetching a userというラベルを設定します。
- let式を使い、(get-user 1)で得られたキーと値のペアを、userという名前のローカル・バインディングにバインドします。
- 最後に、isマクロでアサーションを定義します。このマクロの本体には、some?関数が含まれています。この関数は、渡された値がnil（nullを表す値）でない場合にtrueを、それ以外の場合はfalseを返します。その後、先ほどバインドした、ユーザーを表すデータ構造で:emailキーの値を検索します。

コロンの意味

Clojureでは、キーワードという考え方がサポートされています。これをJavaなどの他の言語における予約キーワードという考え方（staticやfinalなど）と混同しないようにしてください。キーワードは、マップやパターン・マッチングの条件をわかりやすく記述する優れた方法です。また、出力される際には、コロンを除いた文字列として評価されます。Clojureでコードを書く場合、頻繁にキーワードを使うことになるでしょう。キーワードは、記述するコードの意図の表現や宣言に便利で、ユーザー・データのマップのキーや、リストの値を定義する際に使用できます。

```
(ns demo.hello-keywords)

(def brads-details
 {:name "Brad"
  :languages '(:clojure :java :javascript)})
```

キーワードは、Javaの列挙型とよく比較されます。列挙型と同じように使用されることもありますが、コンパイル時または実行時に定義されるキーワードに要件はないことについても触れておくべきでしょう。事実、「keyword」関数を使って、実行時に文字列からキーワードを作成することもできます。先ほどの例では、name、languages、clojure、java、javascriptというキーワードが使われており、これらのキーワードはすべてコロンで始まっていることにお気づきになると思います。

関数型プログラミングという考え方

Clojureの大部分は、関数型プログラミング言語という考え方に基づいています。そのため、Clojureはコードを一連の数式として扱い、状態やデータの変更を避けようとします。Clojureは、関数プログラミングでとてもよく使われる手法によってこのパラダイムにアプローチしています。すなわち、Clojureでは不変の永続データ構造と第一級関数群が使われています。このアプローチとJavaのマップを比較してみます。まずは、Javaのコードを示します。

```
import java.util.HashMap;

HashMap<String, Integer> map = new HashMap<>();
map.put("java", 7);
System.out.println(map);
map.put("java", 8);
```

```
System.out.println(map);
```

ここで何をしているかは容易におわかりいただけるでしょう。最初にマップを出力した際は、javaという名前のキーおよび値7が出力されます。マップのjavaキーに新しい値を設定してから出力した際には、javaという名前のキーおよび値8のマップが出力されます。Clojureでは、この操作は次のようになります。

```
(ns demo.hello-immutability)
(def map {:java 7})
(println map)
(assoc map :java 8)
(println map)
```

このClojureコードでは、両方とも{:java 7}が出力されます。これは、Clojureのマップが不変であるためです。Clojureでは、ほとんどのデータ構造が不変です。(assoc map :java 8)という行で実際に返されるのは、元のマップを複製して作られた{:java 8}という新しいマップです。並列プログラミングには、デッドロックやデータ競合といったありがちな落とし穴があります。そのような落とし穴を避けることができるようにするために、Clojureの中核には不変性という考え方が据えられています。

Clojureの関数は第一級です。すなわち、関数をデータ構造に格納することや、引数として渡すことが可能です。実際、関数の適用（関数をリスト内のすべての値に適用するという考え方）という処理を行っているのは、別の関数とリストを引数として受け取る関数です。たとえば、applyを使って、指定したリスト内のそれぞれの値に関数を適用することができます。パラメータには、匿名関数や、ネームスペースまたはローカル・バインディングにすでにバインドされている関数を使うことができます。最初に示した、値を2倍する例は、次の2つの方法のいずれでも記述できます。

```
(ns demo.hello-apply)

(def double [x] (* x 2))

(apply double (list 1 2 3 4))
; (2 4 6 8)
```

```
(apply #(* % 2) (list 1 2 3 4))
; (2 4 6 8)
```

Lispやメタ言語を使ったことのある方なら、関数型プログラミング言語の分割代入というエレガントな仕組みを期待することでしょう。これは、キーと値のペアやリスト（ネストしている場合もあります）に格納されたデータから複数の値を抽出する際に便利な方法です。分割代入の本質は、データ構造に含まれる、使用またはバインドしたい一部のデータを宣言的に定義できることにあります。分割代入はClojureですぐに使用できるため、複雑なデータ構造をはるかに簡単に扱うことができます。キーワードのリストを受け取り、その最初のキーワードを文字列として返す関数は、次のように記述できます。

```
(ns demo.hello-transform)

(defn first-keyword->string
  [my-list]
  (name
    (first my-list)))
```

関数名のわかりやすさに注目してください。特殊文字（ハイフンや大なり記号など）を関数名に使用できるため、簡潔でわかりやすい関数名を付けることができます。データをある構造から別の構造に変換するような場合には、特にそれが当てはまります。この関数は、1つのパラメータmy-listを受け取り、引数を文字列に変換するname関数を呼び出します。この場合、引数はmy-listの最初の項目になります。

このコードは、分割代入を使って次のように書き換えることもできます。

```
(ns demo.hello-transform)

(defn first-keyword->string
  [[head tail]]
  (name head))
```

このコードこそ、とてもわかりやすいものだと思いませんか? Clojureでは、リスト、マップ、ベクターなどの多くのもの

に対して分割代入を行うことができます。特に、リストやマップなどの大規模なデータ構造がどのくらい使い回されるかを考慮すると、Clojureでの開発において分割代入はすぐに標準的な手法となります。

マクロ・システム

他のLisp言語から色濃く継承した特徴として、先ほども触れたClojureのマクロ・サポートが挙げられます。キャリアの大半でJavaを使ってきたという方にとって、マクロという概念はおなじみではないかもしれません。マクロはコンパイル時に評価され、コードに展開されます。つまり、コンパイラ拡張やCのプリプロセッサと同じようなものです。

マクロは非常に強力な機能で、Clojureではとてもよく使用されます。実は、Clojureの中核をなす構成要素の多くは単なるマクロです。たとえば、Web開発でとてもよく使用されるCompojureというライブラリでは、URLルートを定義し、そのルートに一致する着信リクエストを関数にディスパッチできるようになっていますが、その際にマクロが使われています。コードベースにCompojureへの依存性を追加すると、次のような構文でルートを容易に定義できるようになります。

```
(ns demo.hello-compojure)

(def get-user-by-id
  [id]
  (str "getting user for id: " id))

(defroutes my-routes
  (GET "/user/:id" [id] get-user-by-id)
```

この例では、Compojureを使ってWebサーバーの/user/1へのルートを設定しています。curlまたはWebブラウザを使ってこのリソースをリクエストすると、getting user for id: 1が返されます。実際には、get-user-by-idをもう少し意味のある処理で置き換える必要がありますが、マクロの威力はこの例からわかっていただけるでしょう。マクロを使うと、わかりやすく簡潔な表記法を使ってたくさんのコードを書くことができます。

この例では、Clojureを使ってHTTPサーバーを抽象化するRingというWebサーバーを使っています。筆者がClojureを使った経験のほとんどは、Web中心の事例です。その際に、Ringによる抽象化を使用しています。Ringは長い間使われており、Clojureコミュニティの多くのメンバーがサポートおよびコミットを行ってきています。

まとめ

その構文と長所から、Clojureはすばらしく強力な言語であることが証明されています。筆者もClojureを使ったコーディングが大好きです。触れるべきことはまだたくさんありますが、本記事では、皆さんがこの言語の魅力を感じ、試したいと思っていただけると筆者が考える一部の要点のみを紹介しました。Clojureについてもっと学んでみたいという方は、ホーム・サイト、筆者がClojureに目覚めた無料のオンライン・チュートリアル、または筆者の記事をご覧ください。</article>

Brad Cypert：米国ケンタッキー州ルイビル在住のソフトウェア・エンジニア。Clojure、EdTech、JVMに情熱を傾けている。LinkedIn、Carfaxなどの大手技術企業で働いた経験を持ち、空いた時間には、より快適にポッドキャストを聞けることを目指すWebアプリケーションをClojureとTypeScriptで構築する自身のプロジェクト、Poriosに取り組んでいる。