◆ Rubyの基礎 ◆

3 さまざまなRuby処理系

一般

笹田耕一（Heroku, Inc.）

広がるRuby処理系

プログラミング言語Rubyのプログラムを実行する最初のRubyインタプリタは，1993年から松本行弘らにより開発が開始され，1995年にオープンソースソフトウェア（OSS）としてネットニュースにてソースコードの配布が開始された．このインタプリタはrubyというコマンド名で知られており，Rubyインタプリタといえば，おそらく多くの人がこのコマンドを想像する．開発した松本のニックネームがMatzであるため，これをMatz Ruby Interpreter，略してMRIと呼ばれたり，C言語で実装されていることからCRubyとも呼ばれる．本稿では，このRubyインタプリタをMRIと呼ぶ．

現在でもRuby処理系としてはMRIが有名だが，最近はMRI以外にもRuby処理系が存在し，実際に利用されている．汎用のRuby処理系としては，Javaで記述され，Java仮想マシン（JVM）上で動作するJRubyや，Ruby自体でRuby処理系を記述するRubiniusがある．汎用ではなく，特定の目的に向けたRuby処理系も提案されている．このように，プログラミング言語Rubyは，MRIだけでなく，用途に応じてさまざまな処理系を利用することができるようになってきている．本稿では，さまざまなRuby処理系について紹介する．

MRI：オリジナルRuby処理系

MRIは，開発から20年以上経過した今でも活発に開発が続けられており，2015年8月の最新安定版のバージョンは2.2.2である．長い間，Ruby処理系はMRIしか存在しなかったため，MRIがRubyのリファレンス実装として扱われてきた．つまり，MRIの機能が拡張されれば，それが新しいRuby仕様として扱われてきた．たとえば，ISO/IEC 30170にまとめられた言語仕様は，当時安定版であったMRI 1.8に搭載されている機能をベースにまとめたものとなっている．

MRIはC言語で実装されており，C言語でRuby処理系の拡張ライブラリを記述することができる．スレッドは当初はグリーンスレッドとして実装されたが，現在はRubyスレッド1つにつきOSなどが提供するネイティブスレッドを1つ持つ1対1実装となっている．ただし，インタプリタに1つ存在するジャイアントロックを持ったスレッドのみ実行するよう制御しているため，スレッド同士は並行に動作するが，並列計算機上で同時並列に実行は行わない．

MRIはOSSとして開発されているため，筆者を含む多くの開発者によるさまざまな品質向上の努力が行われてきた．ここでは，特に筆者が中心となって導入した大きな改善を2つ紹介する．まず，MRI 1.9において，抽象構文木（AST）を辿るインタプリタを，バイトコード実行型仮想マシン（VM）に変更した[1]．JVMと同様に，スタックマシン型のVMを設計し，ASTをこのVMのバイトコードへ変換し実行することで，言語VM向けの既知の最適化，たとえばピープホール最適化，スタックキャッシングなどの導入が可能になった．また，VMのメンテナンスコストを抑えるため，VM記述言語を定義し，そこからVM自体のC言語プログラムを自動的に生成するようにし，開発コストの低下に貢献した．

ガーベージコレクション（GC）は，MRI 開発当初より保守的マーク＆スイープを用いており，高いオーバヘッドと最悪停止時間が長いという 2 つの問題があった．そこで，MRI 2.1 において世代別 GC を導入し，オーバヘッドの問題を解決し，さらに MRI 2.2 でインクリメンタル GC とすることで，最悪停止時間が長いという問題を解決した．どちらの技法も，既知ではあったが，導入に必要となるライトバリア（WB）が MRI に含まれていないため，導入することができなかった．我々は，WB で適切に保護されたオブジェクトと，WB で保護されないオブジェクトを区別することで，世代別インクリメンタル GC を実現する手法を開発し，MRI に導入した[2]．この手法では，過去に開発された WB について考慮していない拡張ライブラリとの高い互換性も実現している．

リリースされた MRI はネイティブコードへのコンパイラを備えていないが，これを行うため，いくつか研究が行われている[3), 4)]．

MRI 以外の汎用の Ruby 処理系

MRI 以外にも Ruby 処理系がいくつも開発されている．その中でも，実際に Ruby on Rails といった複雑な Ruby プログラムを利用できる MRI 以外の Ruby 実装である JRuby と Rubinius を紹介する．

JRuby[☆1] は Java で実装された Ruby 処理系であり，JVM の優れたコンパイラや GC などの実装技術を用いることで，高い性能を実現している．Java 環境との親和性も高く，Java のライブラリを Ruby から気軽に呼ぶための独自拡張が行われている．JRuby は Ruby プログラムを JVM のバイトコードへ変換することで，変換済みコードの JVM による，さらなる最適化を行う．この変換には色々とコツがあるようで，JVM バイトコードへのコンパイラは何度も書き換えられ，実装が進化している．さらに，Oracle Lab では JVM 上で動作するプログラムをさらに高速化する GraalVM，Truffle フレームワークが研究されており，JRuby にこれを適用するプロジェクトが進んでいる[☆2]．Java との親和性から，Java の資産を持つ多くの企業で利用されている．

Rubinius[☆3] は Ruby で Ruby を実装するというメタサーキュラーインタプリタとなっており，C++ 言語で記述されたコア機能以外，Ruby で実装されている．そのため，Ruby を用いて処理系自身の拡張も容易であり，インタプリタ自身に手を入れる必要がある機能拡張が比較的容易である．たとえば，メソッド（関数）フレームをまたいだローカル変数のアクセスといった処理を Ruby で記述することができる．独自のバイトコード仮想マシンを定義しており，Ruby プログラムをコンパイルし実行する．この仮想マシンは，LLVM を用いた JIT コンパイル機能を備えており，機械語に変換する．

JRuby も Rubinius も，Ruby スレッドをネイティブスレッド（JRuby の場合は Java のスレッド）に 1 対 1 に割り付け，それらを同時並列に実行することができる．これを実現するため，両処理系には適切な細粒度排他制御を導入している．MRI では，プログラムを同時並列に動かすためにはプロセスを複数立ち上げる必要があるが，メモリを多く消費するという問題がある．JRuby や Rubinius では，複数スレッドを用いることで，より省メモリで同時並列に処理を実行することができ，たとえば複数のリクエストに対して，それぞれ同時並列に処理する Web アプリケーションサーバを効率良く実行することができる．

ただし，Ruby オブジェクトを複数スレッドで共有することになるため，Ruby レベルでの適切なマルチスレッドプログラミングが必要となる．既存の Ruby で記述された Ruby プログラムには，マルチスレッド対応をしていないものも多いため，慎重に用いる必要がある．このように，マルチスレッドを適切に行うのは困難である，という立場から，MRI

☆1 http://jruby.org/

☆2 JRuby+Truffle - a High-Performance Truffle Backend for JRuby, https://github.com/jruby/jruby/wiki/Truffle

☆3 http://rubini.us/

では並行に実行するスレッドの同時実行を禁止している．MRIでは，オブジェクトをすべて共有して並列に実行するスレッドの代わりに，共有を制限した，プログラミングが容易な並列計算モデルの導入を目指している．

特定用途のためのRuby処理系

特定の目的のために開発されたRuby処理系もある．これらの処理系はMRIとの互換を目指しているが，必要性や実現の困難さからサブセットであったり，独自拡張を追加している．

mruby[☆4]は松本が一からRubyを再実装し，計算機リソースが制限された環境でも動作するようコンパクトな実装を目指しているRuby処理系である．MRIには多くの組込みクラスやメソッドが存在するが，mrubyではそれらの多くを取り外し可能としている．Rubyプログラムは事前にバイトコードにコンパイルすることもできるため，それを利用する場合はパーサ自体も取り外しすることができる．独自のバイトコード処理系を実装しているが，MRIと異なり，レジスタマシン型である．組込み環境以外にも，他の言語で開発されたアプリケーションにRuby処理系を組み込むことも目指しており，たとえばWebサーバであるapacheに組み込み，簡単な処理をRubyで記述できるようにするといった提案もある[5]．mrubyの詳細は本特集6「Rubyを使った組込みソフト開発―mrubyによる組込みシステム開発―」を参照していただきたい．

RubyMotion[☆5]はMac OS XやiOS，Android OS上で動作するネイティブアプリケーションをRubyで開発するための独自のインタプリタおよび開発ツールチェインである．ほぼ，Rubyの機能が使えるが，キーワード引数に順序を持つ，という点が異なる（MRIは，順序を気にしない）．これは，Mac OS XやiOSなどで利用するライブラリ群が，キーワードの順序によって呼び出し先を変えるObjective-Cのインタフェースで提供されているためである．また，これらの環境では任意のプログラムを実行するためのインタプリタの搭載を禁止していることがあるため，任意のRubyプログラムを実行するeval()メソッドは提供していない．提供する環境ごとに個別の処理系を搭載することでRubyによるクロスプラットホーム開発を実現している．また，開発環境で動作する対話環境から，対象端末を操作する仕組みも有している．これを実現するためには通常のクロスコンパイルでは不十分であるため，たとえばAndroid端末向けには，開発環境でRubyプログラム片をARM機械語に変換し，端末へ送り込み実行する，という仕組みを実現している．

MagLev[☆6]は，オブジェクトデータベース（OODB）を扱う処理系の上に構築されたRuby処理系である．OODBを扱うSmalltalk処理系であるGemStone/S上にRuby処理系が構築されており，OODBに手続き，スレッド，継続などを含めたRubyオブジェクトを永続化オブジェクトとして保存できる．Rubyを直接ネイティブコードへ変換するコンパイラを有しており，高速に動作する．

RubyをJavaScriptに変換し，RubyでWebブラウザのプログラミングを目指すOpal[☆7]というプロジェクトもある．Webアプリケーション開発の重要性から，Webブラウザで動作するJavaScriptを変換対象とするプログラミング言語が数多く提案されており，Opalはその1つである．性能のために，できる限りJavaScriptにそのまま変換するようにしており，たとえばRubyでは変更可能である文字列オブジェクトはJavaScriptの仕様にあわせ，不変なオブジェクトとして存在する．しかし，定義されていないメソッドの呼び出しをフックするmethod missingといった，JavaScriptにない機能だがRubyでよく利用される機能は，Opal側のコード変換によりサポートされている．Webアプリケーシ

☆4 http://www.mruby.org/
☆5 http://www.rubymotion.com/
☆6 http://maglev.github.io/
☆7 http://opalrb.org/

ョン開発フレームワークである Ruby on Rails 上では，Opal を利用してブラウザ側プログラミングも Ruby で行う仕組みもサポートされている．

進化する Ruby 処理系

駆け足でさまざまな Ruby 処理系を紹介した．

複数の処理系で Ruby の互換性を保つことは相互運用可能性を保つために重要である．Ruby の仕様を自然言語でまとめた ISO/IEC 30170 の目的の 1 つである．また，RubySpec という，Ruby の挙動をチェックする網羅テストを用意するプロジェクトがあり，MRI の更新にあわせて追従するよう努力している．

Ruby ではないが，Ruby から派生した興味深い言語および処理系も出現している．Elixir[☆8] は，Erlang をベースに設計された関数型プログラミング言語で，Erlang の機能を Ruby の文法に似た形で利用することを実現している．Erlang が持つパターンマッチや並行処理の書きやすさといった特徴は受け継ぎながら，Ruby のような言語でプログラミングを行うことができる．Elixir 処理系は Erlang 処理系である BEAM で実装されている．Crystal[☆9] は，Ruby に型推論および静的型付けを導入したプログラミング言語で，一見すると Ruby と見分けが付かない見た目となっている．Crystal の処理系は，事前にすべて Crystal プログラムを LLVM を用いてネイティブコードへ変換する．コンパイル時に型を解決しているため，高速なコードを生成可能である．

以前は，Ruby は文法が複雑なので，Ruby 処理系は MRI しか存在し得ない，と言われたこともあったが，紹介したように，現在ではさまざまな派生が生まれている．Ruby というプログラミング言語自体の魅力も一因だろうが，難しいから挑戦したい，と思わせる部分もあるのかもしれない．Ruby 処理系の実装を解説する書籍も出版されている[6), 7)] ため，さらなる挑戦を期待したい．

☆8 http://elixir-lang.org/
☆9 http://crystal-lang.org/

参考文献

1）笹田耕一，松本行弘，前田敦司，並木美太郎：Ruby 用仮想マシン YARV の実装と評価，情報処理学会論文誌（PRO），Vol.47, No.SIG2（PRO28），pp.57-73 (Feb. 2006).
2）笹田耕一，松本行弘：Ruby におけるライトバリアのないオブジェクトを考慮した世代別インクリメンタル GC の実装，情報処理学会論文誌（PRO），Vol.8, No.1, pp.12-12 (2015).
3）芝　哲史，笹田耕一，平木　敬：CastOff : Ruby 用コンパイラのライブラリとしての実装，情報処理学会論文誌プログラミング（PRO），Vol.5, No.3, pp.1-22 (Aug. 2012).
4）井出真広，倉光君郎 : Ruby 向けトレース方式 Just-in-time コンパイラの設計と実装，情報処理学会論文誌プログラミング（PRO），Vol.8, No.1, pp.1-10 (2015).
5）松本亮介，岡部寿男：mod mruby：スクリプト言語で高速かつ省メモリに拡張可能な Web サーバの機能拡張支援機構，インターネットと運用技術シンポジウム 2013 論文集，pp.79-86（2013）.
6）青木峰郎 著，まつもとゆきひろ 監修 : Ruby ソースコード完全解説，インプレス (2002).
7）Pat Shaughnessy 著，島田浩二・角谷信太郎 共訳 : Ruby のしくみ―Ruby Under a Microscope，オーム社 (2014).

（2015 年 8 月 7 日受付）

笹田耕一（正会員） ko1@heroku.com

2007 年東京大学大学院情報理工学系研究科にて博士（情報理工学）．2006 年同助教，2008 年同講師．2012 年より Heroku, Inc. にて Ruby 処理系（MRI）の開発に従事．ACM 会員．