◆ Rubyの基礎 ◆

1 20年目のRubyの真実

一般

松本行弘☆1（Rubyアソシエーション） 笹田耕一（Heroku, Inc.）

「Rubyの真実」を振り返って

オブジェクト指向プログラミング言語Rubyは1993年から開発を開始し，1995年に最初にオープンソースソフトウェアとしてリリースされた．Rubyは，今や世界中で多くの人が利用するプログラミング言語となった．

松本は2003年に本誌にて「Rubyの真実」[1]という，Rubyについての解説記事を寄稿した．この記事は，Rubyを紹介するのではなく，Rubyはどのような発想から開発されたものかをまとめた．基本的に，「Rubyの真実」で述べたことは，今でもほぼ通用する☆2．

ただ，それから10年以上経ち，新たな知見が得られた．本稿は，「Rubyの真実」をもとに，何が変わって，何が変わらなかったのかについて考察する．そのため，「Rubyの真実」を先に読んでおくことをお勧めする．なお，本稿は笹田が松本にインタビューを行い，その内容をまとめるという形で執筆した．このインタビュー全文についてはWebサイト☆3に掲載しているので，興味があれば参照されたい．

一貫性と驚き最小の法則

「Rubyの真実」では，「プログラミング言語の役割」において，マン・マシン・インタフェースの一種であり，ユーザインタフェースの原則の多くが適用され，特に一貫性，簡潔性，柔軟性が重要である，と述べた．これらが重要であるという意見はあまり変わらないが，現在では一貫性について，別の意見を持っている．

一貫性について，一貫していることを目標にしないようにした．一貫性を理由とせず，一貫していることで得られる価値や，その変更による影響範囲を考慮して検討することとした．たとえば，このクラスAにある機能は，似たクラスBにもあるべきだ，という意見については，明確なユースケースがあり，価値があると判断できれば導入するが，利用しないことが明らかである場合は棄却することにしている．

一貫性と関連する言葉に，「驚き最小の法則」（Principle of least astonishment）という言葉があり，以前は重視していたが，現在は利用しないように注意している．この言葉は，Ruby利用者にとって，利用に際して驚きが少ないことが良いデザインだ，という見方だ．しかし，Rubyの開発が20年以上進められ，Ruby自体の開発に携わる人数も増えた．このとき，驚き最小の法則がプログラミング言語デザインの議論において，邪魔であることが分かった．というのも，異なるバックグラウンドのある者が，独自の「驚き」をもとに議論をするためだ．特に，自分の慣れ親しんだプログラミング言語とRubyの挙動の齟齬を問題視することが多い．そのため，「驚き」という感覚をもとに言語デザインの決定を行わないように方針を転換した．現在は，ユースケースなどをもとに定性的・定量的な議論を行い，最終的な判断を行うようにしている．

2003年の時点では，新しい機能の導入など，言

☆1 普段はRuby開発者として「まつもとゆきひろ」を名乗っている．
☆2 本稿執筆のために，12年ぶりに「Rubyの真実」を再読した松本曰く「この人とは強く共感できる」．
☆3 https://www.ipsj.or.jp/magazine/magfree.html

語設計の提案や判断は主に松本が行っていた．現在は，多くの人々が改善の意見を述べ，松本が最終的に取捨選択をするようになった．方針の変化は，この進め方の違いが大きく影響している．

見かけの重要性

「Ruby の真実」では,「見かけの重要性」において，プログラミング言語において最も大事なのは複雑さの制御であり，見かけの単純さの演出により記述の複雑度を下げることは重要であると述べた．そのため，ミニマリスト的立場からは冗長に見える機能を追加した．これは，Ruby の複雑さを線形に増加させる．しかし，独自の構文を導入することができるマクロは，Ruby の複雑度を指数的に上げると考えたため採用しなかった．複雑な問題を解決するためのプログラミング言語の拡充は，さらなる複雑さの導入につながるが，これを注意深く制御することが重要である．

Ruby はドメイン特化言語（DSL）のベース言語としての人気が高い（DSL と Ruby については，本特集 5「Ruby による Domain Specific Language の実際」を参照されたい）．現在の Ruby の人気は，この DSL としての書きやすさもあると思われる．

この DSL としての Ruby の筋の良さは，当初からプログラムの見た目について強くこだわった結果として大成した．これは，もともと松本が Ruby をソフトウェアの設定ファイル（たとえば，Emacs などエディタの設定ファイル）にしたとき，それらしくなるように設計したためである．

まず，メソッド呼び出しにおいて，曖昧性がない個所ではなるべく引数を囲む括弧を外すことができるようにした．その代わり，パーサの実装はその分とても複雑になった．この工夫により，ある程度プログラムが英語のように見えるようになった．また，コード片をメソッドの引数として簡単に渡すためのブロックを導入した．ブロックの導入により，コードの取り扱いの柔軟性が大きく向上した．

この括弧をなくす，ブロックを導入する，という工夫により，完璧ではないが，ある程度 Ruby プログラムを設定ファイルのように書くことができるようになった．マクロを用いれば，さらに柔軟に記述を変更することができる．しかし，プログラムの複雑度を指数的に増加し，プログラムを把握することを大きく困難にする可能性がある．この 2 つの工夫があれば，マクロを用いて行いたいことの多くができると考え，マクロを導入しなかった．そして，現在 Ruby を用いて多くの DSL が生まれ，利用されている．

コミュニティ

Ruby にかかわるコミュニティは，2003 年から大きく変化した．「Ruby の真実」では，コミュニティについて，主に日本国内の Ruby 言語およびインタプリタ開発コミュニティについての言及しかないが，現在は世界規模のユーザコミュニティ，および国際的な Ruby 開発コミュニティがあり，Ruby の世界を広げた立役者となっている．

Ruby のユーザコミュニティを世界に広げた要因として，ここでは 2 つ挙げる．RubyGems の開発と Ruby on Rails の成功だ．RubyGems とは，Ruby のライブラリをパッケージ化し，手元の環境でそのライブラリを容易に利用するためのツールだ．RubyGems によって，Ruby 利用者が必要とする機能に簡単にアクセスすることができるようになった．Ruby on Rails は，2004 年から David Heinemeier Hansson（DHH）が開発した Web アプリケーションフレームワークで，世界中で人気を誇っている．この 2 つの成功により，Ruby は国際的なユーザコミュニティを得ることになった．

国際的なコミュニティにあって，松本は言語開発者という立場から，コミュニティのリーダーとしての役割を期待されることが多くなった．その立場として，公平公正であること，そして出会いを大切にすることを心がけている．たとえば，Ruby に関するふとした思いつきについても，それが既出のアイディアであっても「3 年勉強してから出直してこい」

というような反応をせず，真面目に対応するようにしている．3年前にRubyに出会えなかったのは発言者のせいではないし，時間を理由に答えないのはフェアではないためだ．

出会いについては，たとえば2000年ごろ，英語で最初に出版されたRubyの書籍の執筆者であるDave Thomasと英語で多くの議論を行った．その結果，彼らの書籍によりRubyが世界中に広まったのは事実である．また，2001年に行われた国際カンファレンスで，前述のDHHと出会い，Rubyについて大いに議論した．これがきっかけとなり，Ruby on Railsが生まれたのかもしれない．これらの姿勢はファンを大事にする，と言い換えることもできる．

2003年には予想できなかった点として，現在のソーシャルコーディングという流行がある．GitHubといったソースコードリポジトリサービスにオープンソースでソースコードを公開しておき，そのソースコードを他者が発見し，改良し，フィードバックする，というプロセスを，システムのサポートを加えてとても速く回すことで，ソフトウェアの進化を強力に促進するものだ．

さらに，この流行の中で，プロモーションが重要であることが分かった．現在松本が開発している別のプログラミング言語であるStreemは，文法定義しかない状態で，広く公開するつもりはなく，バックアップ代わりにGitHub上に送信した結果，「Rubyの松本の新言語」ということで注目が集まり，未実装の機能を有志が実装するなど，多くの協力を得ることができた．未完成のまま放置されるソフトウェアが少なくない中，十分な注目を受けることで，ソフトウェア開発コミュニティにより開発を進めていけることが分かった．

Ruby処理系とその課題

Ruby処理系は，2003年の状況から大きく発展している．松本が開発を開始し，リファレンス実装となっているRuby（CRubyやMRIとも呼ばれる）のほかに，Javaで開発されたJRuby，多くをRuby自身で記述したRubiniusがその他のRuby処理系として有名である．また，ISO/IEC 30170に準拠した組込み向けに開発されているmrubyも登場した．

「Rubyの真実」の「Rubyの実装とその課題」および「将来の実装」には，2003年時点でのいくつかの課題や目標が述べられているが，それらの多くは解決している．

「インタプリタの初期化」，「インタプリタの構造化」はアプリケーション組込みも目指すmrubyによって解決された．「ネイティブスレッド対応」は，Ruby 1.9によって解決した．ただし，ジャイアントロックを用いて同時に実行するRubyプログラムが動作するスレッドは，たかだか1つに制限している．これは，完全に並行・並列に実行したときに生じるかもしれないバグを，なるべく排除するための処置である．

「インタプリタのバイトコード化」については，Ruby 1.9で実現した．また，「世代別ガーベージコレクタ」についてはRuby 2.1で実現した．現在最新版であるRuby 2.2では，加えてインクリメンタルGCも実装している．これらの工夫により，2003年時点で最新版であったRuby 1.8よりも，大きく性能が向上している．

mrubyについては本特集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，他の処理系については同じく本特集3「さまざまなRuby処理系」を参照されたい．

今後の課題

2003年に予測できなかったことの1つに，並列計算機のコモディティ化がある．マルチコア，メニーコアマシンにおいては，並列プログラミングを行わなければ十分に計算機を使い切ることができない．現在，ジャイアントロックで複数スレッドを同時並列に実行できないRubyで並列プログラミングを行うためには，Rubyプロセスを複数立ち上げ，それらが協調して実行していく必要がある．JRubyや

Rubinius といった別実装では，生成したスレッドを並列計算機上で同時に実行することができる．しかし，状態をすべて共有するスレッドプログラミングは根本的に難しい．容易に並行・並列プログラミングを行うための，新しい並行・並列計算モデルの導入が必要である．

また，メモリ消費が問題になることがある，というのも 2003 年当時では想定していなかった事実である．Ruby が Web アプリケーション開発に利用されることになり，Ruby で大規模なアプリケーションを記述するようになったためである．さらに，これを複数プロセスで実行しようとすると，メモリ消費が増大する．クラウド環境での一般的な利用形態は，計算機リソースの利用にあたり，時間と規模に応じて課金される形式であるが，メモリ消費が多ければ，それが利用料に反映するため，メモリ消費を抑えてほしい，という要求が多くある．メモリハードウェアの単価は 2003 年に比べて大幅に下がっているのに，計算機の利用形態の変化によって，この課題が以前よりも重要になっているのは興味深い．

そのほかの課題として，Ruby の適用範囲の拡張が考えられる．2003 年当時は Ruby はテキスト処理を行うためのスクリプト言語という立ち位置であったが，現在は Ruby 人気の起爆剤となった Ruby on Rails の影響から，Ruby は Web アプリケーションを書くための言語という捉え方をされることが多い．この評判を維持するだけでは，この先に進むことができず，新しい計算機の活用領域に対応できなくなるため，この固定観念を打ち破り，新たな Ruby の活躍する領域を作っていきたい．たとえば，データ解析に利用するための SciRuby プロジェクトなどに期待している．

Ruby は 20 年以上経つプロジェクトであり，多くの Ruby アプリケーションが生まれている．そのため，互換性を堅持するような，保守的な判断に陥りがちとなる．しかし，そのような姿勢でプログラミング言語を開発していくと，目立った新機能を提案できず，コミュニティが衰退してしまうため，これからも新しい挑戦を続けていきたい．

読者への挑戦状

日本のコンピュータサイエンスの場において，さまざまな試みが行われ，プログラミング言語も多く開発されてきた．しかし，そのような多くの日本発のプログラミング言語の中で，世界中で広く利用されているのは Ruby のみである．2003 年に松本が執筆した「Ruby の真実」では，その状況に一石を投じるつもりで，当時の知見と課題をまとめた．しかし，現在もその状況は変わっていない．最近では Google 社による Go 言語や，Apple 社による Swift 言語など，企業による新言語が注目されているが，CoffeeScript や Clojure，Elixir といった個人によるプログラミング言語も登場しており，個人だからできない，というわけでは，少なくとも今のところはなっていない．次の日本発のプログラミング言語が，やはり松本が現在開発している Streem では，笑うに笑えない．

次の 10 年こそ，新しいプログラミング言語が日本から，そしてこの記事の読者から，出てきてほしいと切に願う．

参考文献

1）松本行弘 : Ruby の真実，情報処理，Vol.44, No.5, pp.515-521（May 2003）．

（2015 年 7 月 21 日受付）

松本行弘　matz@ruby.or.jp

1990 年筑波大学第三学群情報学類卒業．（財）Ruby アソシエーション理事長，米 Heroku Chief Architect, Ruby など兼務．プログラミング言語の設計と実装に興味を持つ．

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．