◆ Rubyの基礎 ◆

2 Rubyの言語仕様と標準化

一般

中田育男

Rubyの標準化の意味

Rubyは2011年3月にJIS規格[1]，2012年4月にはISO規格[2]となり，名実ともにプログラム言語として世界に認められた．それにより，官公庁などで調達するソフトウェアもRubyで記述することが可能になった．今までRubyの言語仕様を記述したドキュメントには例題による記述にとどまるものが多く，厳密な仕様が書かれていなかったので，「仕様は処理系☆1に聞け」と言われ，仕様が分からなかったら，実際に処理系にかけてその結果で判断すればいいとされていた．Rubyの規格文書はRuby言語の基本的な部分の厳密な仕様を文書化したものである．今後開発されるRuby系の言語は，この規格をもとにすることができる．実際に，組込みシステム向けの軽量Ruby（mruby[3]）はこの規格文書をもとに開発されている．また，この規格の仕様記述をもとにRubyの操作的意味論の研究[4]も行われている．

ここでは，Rubyの言語の特徴と，その仕様がどのように記述されているかを解説し，今後の課題について述べる．

Rubyの標準化の過程

Rubyの言語仕様の国際規格化の作業は，2008年にIPA（情報処理推進機構）に設置されたRuby標準化検討WG（ワーキンググループ）で行われた．Rubyの言語仕様は処理系で決められていたから，その処理系の内容を熟知している人が原案を作成し，それをWGで検討して修正するという方法で，**図-1**の方針で進められた．（2）は新しいバージョンも規格に適合していると言えるようにするためである．（3），（4）は早期に標準化するためであり，（4）は日本主導で行うためである．

（1）広く使われているRuby 1.8をベースとする
（2）Ruby 1.9以降と互換性のないものは規定しない
（3）最初はRubyの核の部分と必要最小限のライブラリだけとする
（4）まずJIS規格とし，それの英語版をfast trackでISO規格とする

図-1　Ruby標準化の方針（2008年策定）

原案の一応の形ができたところで，Rubyコミュニティからのコメントによる修正を施し，規格文書として提案した．提案後も規格文書の審議者からのコメント，WG内での再検討などによって修正を繰り返した．修正の議論は主として電子メールで行ったが，その数は8,000通を超えている．

構文規則の記述法

Rubyはプログラマが楽しくプログラミングできるように設計されている，と言われている．プログラマが書きたくなるようないろいろな書き方が許されている．しかし，それは言語の文法規則を一般には複雑にする．たとえば，メソッド呼び出しの実引数を括弧で囲まなくてもよいので，メソッド呼び出しの式があたかもコマンドか変数参照のように見える．これはプログラムの見かけの表現の幅を広げている．しかし，その文法規則を書くのは簡単ではない．

言語仕様は構文規則と意味規則によって規定される．

☆1 Rubyの処理系はいくつかあるが，もとになっているのはMRI（Matz Ruby Implementation）である．以下，ここでは「処理系」はMRIを示すものとする．

否定先読み
《括弧なし実引数》::［先読み∉{“{”}］《実引数リスト》

禁止
《単一変数代入式》::《変数》［《行終端子》禁止］“=”《演算子式》

必須
《分離子》::“;” |［《行終端子》必須］

除外
《複数行コメント行》::《コメント行》ー《複数行コメント終了行》

自然語
《ソース文字》::
　［ISO/IEC 646 の国際基準版で規定されている任意の文字］

図 -2　生成規則の特殊記法

〖self〗	現在の実行主体を表すオブジェクトのスタック
〖クラスモジュールリスト〗	現在のクラス，モジュールまたは特異クラスの定義の入れ子状態を表しているリストのスタック
〖省略時可視性〗	メソッドの可視性のスタック
〖局所変数束縛集合〗	局所変数束縛の集合のスタック
〖定義時メソッド名〗	呼び出されたメソッドの定義名のスタック
〖ブロック〗	メソッド呼出し時に渡された《ブロック》のスタック
〖大域変数束縛集合〗	大域変数束縛の集合

図 -3　実行環境

構文規則は一般には生成規則と呼ばれるもので記述される．生成規則としては比較的表現力の強い，右辺に正規表現の形を許すものが使われたが，それだけではRuby の複雑な構文規則を記述しきれない．そこでいくつかの記法を導入した．それを**図 -2** に示す．

「《》」で囲まれているのは非終端記号である．［先読み ∉{“{”}］は，そこで先読みした記号が集合 {“{”} に入っていない，すなわち “{” でないことを表している．この例は，《括弧なし実引数》は “{” で始まらない《実引数リスト》であると読める．［《行終端子》禁止］は，そこに《行終端子》（改行）があってはならないことを，［《行終端子》必須］は，そこに《行終端子》がなければならないことを表している☆2．この 2 つの例では，代入式の等号の前には改行があってはならないことと，文と文の間の《分離子》は “;” か《行終端子》であることを表している．記号「ー」は除外を意味し，この例は，《複数行コメント行》は《複数行コメント終了行》以外の《コメント行》であると読める．最後の例は「[]」で囲まれた中に自然語で表現したものである．

以上の特殊記号を使うことでほとんどの構文規則は生成規則の形で記述されたが，生成規則だけでは記述しきれない場合には自然語で記述されている．たとえば，メソッド呼び出しの括弧が省略できて，しかも変数の宣言がないから，式の中に現れた局所変数識別子が変数参照であるか，括弧なし引数なしのメソッド呼び出しであるかは生成規則だけでは規定できない．処理系では，その識別子がそれまでに変数参照として使われていれば，変数参照であるが，そうでなければメソッド呼び出しであるとしているので，そのことは自然語で記述している．また，《配列リテラル》の構文規則は 1 つの生成規則では規定できない．たとえば「%w((ab cd)(ef))」のように「%w」の直後が左括弧で始まるものはそれに対応する右括弧までが《配列リテラル》の範囲であるが，その値は括弧構造とは関係なく，空白で要素が区切られたもので，この例では [“(ab”, “cd)(ef)”] である．規格文書では空白で区切られることを生成規則で規定し，括弧構造は自然語で記述している．ただし，改訂版では自然語はやめて，両方の生成規則を併記することが検討されている．

☆2 Ruby では，特に意味を持たない《行終端子》はプログラムの中でいろいろなところに入れられるが，それを生成規則で表現すれば，右辺にたくさん《行終端子》を書くことになるので，規格文書ではそれを基本的に省略している．

意味規則の記述法

意味規則には，プログラムの字面から決まる静的意味規則と，実行時に決まる動的意味規則があるが，Ruby では動的に決まるものが多い．動的意味規則は，Ruby の処理系（インタプリタ）が持つ実行時の情報を抽象化した**図 -3** の実行環境を使って記述されている．

〖大域変数束縛集合〗以外はスタックであり，プログラムの実行（規格文書では「処理系による評価」という）に応じてスタック要素の積み／降ろし

や内容の変更／参照が行われる.〚self〛には評価中のクラスやモジュールや特異クラス（以下ここではクラス類という）のオブジェクトやメソッド呼び出しの際のレシーバなどが積まれる.〚クラスモジュールリスト〛はクラス類のリストのスタックであり，トップのリストは，先頭が現在評価中のクラス類，あるいは現在実行中のメソッドが定義されているクラス類で，それに続いて，そのクラス類を囲むクラス類がネストの内側から順に並んでいるリストである．たとえば，式の中に《定数識別子》が現れた場合，その定数の値は，まず，このトップのリストの先頭から順に探して求める．それで見つからない場合は，リストの先頭のクラス類がインクルードしているモジュールの中を探す．それで見つからなければ，先頭のクラスのスーパクラスで探す，といったことが行われる.

言語仕様の明確化

完成した規格文書には，まだいくつかの問題点はあるが，いままで例題でしか説明されていなかった複雑な仕様も一般的な形で正確に記述されている．たとえば，メソッド呼出しの形とその意味は，19 個の生成規則と，それに対応した 11 個の大項目，40 個の中項目，26 個の小項目などからなる意味規則で記述しつくされている.

文と式の区別も明確に書かれた．文と式は Ruby の本 [5] では明確に区別して使われてはいないが，規格文書では《文》と《式》という非終端記号で明確に区別されている.《式》は《文》として使うことができる（《文》から《式》が生成される）が,《式》を書くべきところに《式》以外の《文》を書くことはできない.通常 if 文と呼ばれるものは《if 式》という《式》であるが,

《文》if《式》

という形のものは《if 修飾文》という《文》である.制御フローを変更する文と考えられているものも《式》であり，通常 return 文と呼ばれるものは《return 式》である. したがって,《return 式》を式の中に書くこともできる．また, 右辺が《演算子式》である代入は《代入式》であるが，右辺が《括弧なしメソッド呼出し》である代入は《代入文》である.

この規格文書作成の過程で，いままで仕様として決まらなかったものが決まったり，実装上の欠陥が判明して修正したこともある．たとえば，特異クラスを表す英語について，eigenclass が使われたり，singleton class が使われたりという用語の揺れがあったが，規格文書では singleton class とし，それが正式名称になった．また，Kernel モジュールの instance_eval メソッドの中で super を呼んだ場合の動作が実装上考慮されていなかったことが判明し，その場合にはエラーとするように処理系が修正され，そのことが《super 式》の意味規則に書き足された．ただし，Ruby 1.8 では修正されていないので，規格文書ではその場合の動作は未規定としている．また，オフィシャル・ドキュメント [6] の Module.constants の記述に誤りがあることが判明し修正された.

現規格の問題点

現規格では，今まで例題を使っての説明しかなかった言語仕様についても，一般的な形で正確に定義している．しかし，まだいくつかの問題がある.

◆ 現規格の欠陥

生成規則の数は 360 ほどあり，目視でそれらに欠陥がないかを確かめるのは困難である．そこで生成規則にできるだけ忠実な形でパーサを記述することを考えて，Scala 言語のパーサ・コンビネータで記述してみた [7]．生成規則では順序を規定していないものでも，プログラムとして書けば順序が規定されてしまうから，それで完全なパーサが記述できたわけではないが，その過程で生成規則の中の 3 つの欠陥を見つけることができた．現規格にはその欠陥がまだ残っている．もう 1 つその過程で，括弧なし実引数なしのメソッド呼び出しの局所変数識別子を生成する生成規則がないことに気がついた．処理系では，局所変数識別子をとりあえず局所変数参照としてから，それがメソッド呼び出しかどうかをチェックしているので，生

```
factorial = 1
2.upto(n) { |x| factorial *= x }
print factorial
```

図 -4　ブロックの例

成規則もそうしてしまっていた．しかし言語仕様の生成規則としては，局所変数識別子がメソッド呼び出しとして生成されるものが必要である．それを入れると文法が曖昧になるが，その解決法は別途記述すればよい．

意味規則に曖昧な点があると指摘されている[4]のはブロックの実行環境についてである．**図 -4** のプログラムでは，2 というオブジェクトの upto メソッドを呼び出すとき，ブロック { |x| factorial *= x } が渡され，upto メソッドからそのブロックが呼び出される．そのときのブロックの実行環境はそのブロックを渡すメソッド呼び出しが書かれている場所（図 -4 の 2 行目）の実行環境であるが，そのことが少し曖昧に書かれている．

◆ バージョンによって異なる機能は定義していない

バージョン 1.8 をベースとして，1.9 以降では互換性のない機能を規定してないので，どちらのバージョンもこの規格に適合していると言える点は良いのであるが，そのために未規定としている機能が多すぎるのが問題である．特に，文字データの扱いが 1.8 とそれ以降で違うので，現規格では ASCII 文字しか扱っていない．日本が提案した規格なのに，日本語のような文字が扱われていないのは残念である．

◆ 基本的なライブラリしか規定されていない

Ruby には膨大なライブラリがあり，それによってプログラムが容易に作成できるようになっているが，今回の標準化では言語の核の部分を規定するのに重点を置き，ライブラリについては，入門書のプログラム程度の簡単なプログラムが書けるための必要最小限のものだけしか規定していない．現規格に書かれているものは，クラスが，Object, Module, Class, NilClass, TrueClass, FalseClass, Numeric, Integer, Float, String, Symbol, Array, Hash, Range, Regexp, MatchData, Proc, Struct, Time, IO, File, Exception（と Exception のいくつかのサブクラス），モジュールが，Kernel, Comparable, Enumerable だけである．メソッドも基本的なものに限られている．

今後の課題

Ruby はバージョン 1.8 から 2.0 となって完成版と言えるようなものになっている．たとえば，文字データは 1.8 では単なるバイト列であったが，2.0 ではバイト列とエンコーディングの組として表現される．Ruby の特徴であるブロックも，1.8 ではその引数は，通常のメソッド呼び出しの引数とは異質のものがあったが，2.0 では，それらは統一されている．前記のような問題点を解消し，完成版の規格を作成するのが今後の課題である．

とりあえず，急いで，必要最小限の規格を作成し，その後で充実したものを作ろうという作戦で，現規格を作成したのであるが，その後，資金源がなくなったために作業は停滞し，現在は Ruby アソシエーションに設置された新規格作成のための WG で，GitHub 上での改訂作業が細々と続けられているだけである．日本で開発された言語の，日本で作成した規格案として恥ずかしくないものが早期に作成されることが望ましいのであるが．

参考文献

1）JIS X 3017 : 2011「プログラム言語 Ruby」（2013 年，ISO/IEC 30170 に合わせて改定）
2）ISO/IEC 30170:2012, Information technology – Programming languages – Ruby
3）http://www.mruby.org
4）Ueno, K., Fukasawa, Y., Morihata, A. and Ohori, A. : The Essence of Ruby, In Programming Languages and Systems, Lecture Notes in Computer Science, Vol.8858, pp.78-98（2014）.
5）David Flanagan, まつもとゆきひろ 著，卜部昌平 監訳，長尾高弘 訳：プログラミング言語 Ruby，オライリー・ジャパン（2009）.
6）http://ruby-doc.org/
7）https://github.com/inakata/ruby_scala

（2015 年 6 月 18 日受付）

中田育男（名誉会員） inakata@jcom.zaq.ne.jp

1960 年東京大学大学院数物系研究科修士課程修了．同年日立製作所入社．1979 年から 2008 年まで筑波大学，図書館情報大学，法政大学の各教授．2008 年 Ruby 標準化 WG 委員長．