◆ Rubyの応用 ◆

5 RubyによるDomain Specific Languageの実際

一般

田島暁雄（日本NCR（株））

本稿では，DSL（Domain Specific Language――ドメイン特化言語）用プログラミング言語としてのRubyに着目する．Rubyに向いた処理としてDSLの作成ということが言われている．実例を示してRubyのどのような点がDSL向きとされているかについて説明する．

最初にDSLという言葉の意味と歴史，DSL実装言語としてのRubyという見方の出自について説明する．それから具体的なRubyによるDSLの実装パターンを解説する．

RubyとDSLの歴史

ここでは，最初にDSLという言葉が持つ意味について説明する．次に，Rubyにおける DSLの位置付けと歴史，DSLという言葉が受容されるまでについて説明する．

◆ ミニ言語とDSL

元々Unix文化ではawkやシェルスクリプトなどのツールを利用して当意即妙な小さな言語（Little Language）を作って問題を解決するということが行われていた．例として『珠玉のプログラミング』（Jon Bentley著，小林健一郎訳，丸善出版，2014）を挙げることができる．

英語による最初のRuby解説書『Programming Ruby : The Pragmatic Programmer's Guide』 の著者のDavid ThomasとAndrew Huntが上梓した『達人プログラマー―システム開発の職人から名匠への道―』（Andrew Hunt, David Thomas著，村上雅章訳，ピアソン・エデュケーション（2000））に達人のアプローチとして問題領域に特化した「ミニ言語」[☆1]の作成を紹介している．

『達人プログラマー』ではミニ言語を次の2種類に分類する．

- スタンド・アローン型
  たとえばPerlの正規表現を利用して簡単にパースできる行指向言語を作り利用する．
- 埋め込み言語型
  問題領域を示す名前の関数を用意して，それを中心に記述することで，元のスクリプト言語自体をミニ言語として扱う．現時点での最も良い例は，C#におけるLINQのクエリ構文であろう．

一方のDSLはドメイン工学から生まれた．これは，その名が示すように特定の問題領域（ドメイン）に特化したプログラミング言語であり，ソフトウェア開発においてドメイン分析／ドメイン設計に注目が集まった1990年代に成立したソフトウェア実装のためのアイディアである．

ドメイン分析／設計では，ドメインをどのようにモデル化するかがソフトウェア開発の中核になる．ここで定義されたモデルをオブジェクト指向プログラミングのオブジェクトに写像できれば，分析／設計から実装までが連続する．途中に実装のための設計を経ないため，効率的であり，かつ実装時の解釈間違いによるバグの混入を防ぐこともできる．ところが，ドメインスペシャリストはその問題領域の専門家ではあるが，ソフトウェア開発の専門家であるとは限らない．このため，結局はモデル―オブジェ

☆1 どちらかというと「リトル言語」のほうが一般的だと思うが，本稿では以降『達人プログラマー』に合わせて「ミニ言語」と表記する．

クトの写像に非連続性が生じ，先に挙げた開発上の優位性を失うことになる．

このモデル―オブジェクトのギャップを解決するために考案されたものがDSLである．具体的には，プログラミングの専門家ではないドメインスペシャリストが最終的なソフトウェアの生成を行えるようにする簡易記述言語を意味する．

なお，同時期に成立したコンポーネント指向では，専門のプログラマが開発したコンポーネントをドメインスペシャリストが簡易なグルー言語で組み合わせて操作するという方法で同じ問題を解決しようとする．この時期はソフトウェアの規模が巨大化し，分析／設計と実装の乖離が目立つようになったことがこれらの技術を開発する動機となったと言えよう．このように，ドメイン工学におけるDSLは，分析／設計の完全性が重視されるシステム――金融などの大規模システム構築における実装のための役割を担うものであった☆2．

原書が2000年に出版された『ジェネレーティブプログラミング』（Krzysztof Czarnecki, Ulrich W. Eisenecker 著，津田義史，今関　剛，朝比奈勲 訳，翔泳社（2008））は，このタイプのDSLについて詳述し，固定・独立型，組込み型，モジュール組合せ型の3種類の実現方法について述べている．

適用分野と利用者層の断絶から，ミニ言語とDSLは互いに独立しているように見えるが，実体は同じものである．

◆ Rubyによるミニ言語Rakeの登場

Rubyは Unix 文化圏に属することから，21世紀初頭の時点ではミニ言語のためのツールとして利用されることは多かったと考えられる．しかし，当初Rubyは知る人のみが便利に利用する汎用のオブジェクト指向スクリプト言語という状態にとどまっていた．すでに『達人プログラマー』でミニ言語というアプローチを紹介し，『Programming Ruby：The Pragmatic Programmer's Guide』でRubyの文法やライブラリの解説を終えたDavid ThomasとAndrew Huntが，Rubyを利用したミニ言語について公に語ることはなかった．また，ミニ言語はその性格上，個々の開発者が必要に応じてその場その場で自分の問題を解決するために作るものなので公開する動機もなかったと言える．

この状況を変えたのが2003年に公開されたRakeである．

Rakeは，Rubyで開発されたMakeの代替ソフトウェアである．

Makeは特殊な書式を使って，ファイル間の依存関係と，依存関係が崩れた場合（具体的には作成時刻が逆転した場合）に実行するコマンドを定義したファイルを読み込んで実行するミニ言語（と同時にDSLの具体例でもある）の処理系で，主にビルドシステムとしてソフトウェアの構築（コンパイル，リンク，配布など）に使われる．

2003年3月15日，英語圏用に用意されているRubyのメーリングリスト「ruby-talk」にJim Weirichの「The onion truck strikes again ... Announcing rake」が投稿された☆3．

この投稿でJim Weirichは，Rubyの特徴をうまく説明しているため，かいつまんで紹介する．

複雑なソフトウェアの構築処理のためにMakefileと格闘していたJimは，ふとMakeの定義ファイルをRubyで記述することを考え付く．

リスト）

```
task "build" do
  java_compile (...args, etc ...)
end
```

task関数にbuildという文字列をターゲットとして登録しておけば，ビルドシステムは必要に応じてbuildというターゲット――doからendの

☆2 この意味でのDSLは，たとえばモデルからソフトウェアを生成するツールで知られるMDAのように当初意図していたほど期待する効果を得られず普及しなかったことと，実装をオフショア開発へアウトソースしてコストダウンすることが主流のシステム構築方法となったことで，フェーズアウトしたと考えられる．

☆3 http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-talk/66974．なおこの投稿はアイディアの披露でありRakeのリリースは5日後の投稿の「ANN Rake」である．

間に記述したブロック――を実行する．

続けてこのアイディアの長所として次の4点を挙げる．

- Makeの奇妙な文法を使わずに，Rubyの文法だけを使って記述できる
- Make以外のビルドシステム（例としてJava用のビルドシステムのantを挙げている）のようにXMLを編集したり読んだりしなくてもよい
- 実装ごとに方言があるMakeと異なりRubyにのみ依存しているためプラットフォーム非依存である
- Rubyの処理系があればほかのビルドシステムは不要である

ここで示されたアイディアの特徴は，ビルドシステム（たとえばMake）がそのシステム用の言語で記述された定義ファイル（たとえばMakefile）を読み込んで処理を実行するのでは「ない」点にある．つまり，ビルドシステムがRubyで記述された独立したプログラムであると同時に，定義ファイルもRubyで記述されたプログラムであるという点である．これは『ジェネレーティブプログラミング』での組込み型や『達人プログラマー』の埋め込み言語型に似ている．その一方で，ビルドシステムのプログラムと，定義ファイルのプログラムがあらかじめ分離されている点が異なる．結果として，使い捨てのミニ言語を超える再利用性が確保されている．

これが可能となった理由は，最初に示したJimの3行の例によく表れている．

この例にはRubyの以下の特徴がすべて盛り込まれている．

- 関数呼び出しの引数を括るカッコが不要である（1行目）
- 改行で終結する場合，式末尾のセミコロンなどの終端子が不要である
- 関数を引数とできる．この場合，通常の引数リストとは独立して末尾に`do ～ end`のブロックを記述する

最初と次の点は，定義ファイルとして記述するプログラムからプログラム然とした記号の排除に役立つ☆4．

そして3点目により，各定義ファイルで固有となる処理を簡潔な記法で関数パラメータ化できる．

特に最後の点は，当時Rubyほど簡潔に関数引数を記述できるALGOL系言語がほかに見当たらなかったことから重要である．

ここで挙げた3つの特徴は単に見た目が定義ファイルに見えるというだけではなく，カッコのマッチやカンマの抜け漏れといった文法エラーが生じにくいという点からも，定義ファイルらしさをもたらしている．

ここまで見たように，

- Unixのスクリプトプログラミングでは旧来からミニ言語を利用した開発の補助は一般的であった．
- DSLという概念と用語はドメイン工学に由来する．その本来の目的は設計と実装の統合，再利用性の確保，それに伴う開発コストの低減である．
- Rakeの登場は，ミニ言語とミニ言語処理系の両方がRubyプログラムであるという点で画期的であった☆5．またアイディアが公開されることで他の開発者にRubyの活用方法を広く知らしめた．

◆「Ruby による DSL」の確立

DSLという用語がRubyコミュニティで一般化したのは2005年である．

例として2005年のRubyConfでのRake作者のJim Weirichの講演「Creating Domain Specific Languages in Ruby」を挙げることができよう．2003年のRakeの公開時点ではまったく使っていなかったDSLという用語がここではタイトルとして使われている．

日本のRubyコミュニティのWeb情報誌である『Rubyist Magazine』で「DSL」の初出を調べると☆6，同年9月の第9号の巻頭言が見つかる☆7．

☆4 これはRakeの定義ファイルを記述するのもRubyプログラマだと前提していることと考え合わせると奇異である．しかし「気分として」納得できる．定義ファイルのアフォーダンスに従っていることが重要なのであろう．

☆5 ただしLISPコミュニティのようにマクロを自由に利用できるプログラミング言語の利用者にとってはなんら画期性はないと考えられる．

☆6 「Domain Specific Language」で調べると2005年7月の8号（http://magazine.rubyist.net/?0008-RLR）である．実際には「DSL」もこの時点で出ているが，いずれにしても2005年である．

☆7 http://magazine.rubyist.net/?0009-ForeWord

------- 引用開始 -------
最近 Ruby が 内部 DSL のホスト言語として取り上げられることがある（Martin Fowler's Bliki in Japanese― 言語ワークベンチ：ドメイン特化言語のキラーアプリ☆8）が，これを快適にしている文法上の理由の 1 つとして，メソッド名とその引数との間に使われる「カッコ」が省略可能である，という点が挙げられる．
--- 引用ここまで☆9---

結論を先に述べると，巻頭言で引用されている Martin Fowler の『言語ワークベンチ』という論考が DSL という用語の一般化と Ruby コミュニティへの普及に重大な役割を果たした．というのは，この論考の中で Ruby は C# などと並んで DSL 用の言語として大きく取り上げられていたからである．

これには Martin Fowler の立ち位置が大きい．彼は DSL が対象としたエンタープライズシステムのコンサルタントであると同時に Ruby などの Unix 文化に根ざした OSS コミュニティに対しても影響力を持っている．また，前年（2004 年）に公開されて Ruby の知名度をいっきに高めることになった Ruby on Rails の存在が，Martin Fowler が Ruby に着目するきっかけとなった可能性が高い．

いずれにしても，『言語ワークベンチ』の中でミニ言語が DSL の 1 実装形態として説明されたことによって，この論考後，元々それほど利用されていたわけではないミニ言語という呼び方ではなく，DSL という呼び方が一般化したのである．

Ruby での DSL の実例

実際に利用されている Ruby の DSL について見てみよう．

ここでは設定ファイルと定型処理の 2 つのカテゴリについて，それぞれ異なる 2 つの実装パターンを示す．

例 1）tDiary の設定ファイル

最初にきわめて原始的な DSL として tDiary という Web 日記システムの設定ファイルを示す．

リスト）サンプルの設定ファイル（tdiary.conf）の抜粋

```
# 日記より上位のコンテンツがある場合に指定
# @index_page には，あなたの日記の表紙（またはホームページのトップページ）
# の URL を指定してください．日記のヘッダ部分および，
# ページの先頭に埋め込まれます．
@index_page = 'http://www.example.net/~foo/'

# TITLE タグおよび携帯端末で使われる日記タイトル．
# HTML タグは使えません．
@html_title = 'hogehoge diary'
```

（https://github.com/tdiary/tdiary-core/blob/master/tdiary.conf.sample）

ここで利用されているプログラミング要素は設定ファイル（DSL）上のコメントと代入，プログラム側での eval である．

この例の重要な特徴は，非 Ruby プログラマにも設定可能だという点である．tDiary は 2001 年 1 月に公開され，日本では Web ホスティング業者が Ruby をインストールするきっかけとなったと称されるくらいに普及した Web 日記システムである．このため，Ruby に対する知識がまったくない非プログラマーが設置する例もあった．また拡張子が Ruby のプログラムを示す .rb ではなく，.conf として設定ファイルのアフォーダンスに従っている点にも注目したい．

tdiary.conf を DSL として挙げることができるのは，設定ファイル自体がインスタンス変数を設定するプログラムだという点と，プレーンな初期化ファイル（ini ファイルなど）と異なり，読み込み用ライブラリの利用やファイルのパースをせずに eval によって直接プログラム内にロードする点である．

☆8 http://bliki-ja.github.io/LanguageWorkbench/（原文は http://martinfowler.com/articles/languageWorkbench.html）
☆9 http://magazine.rubyist.net/?0009-ForeWord

例 2）Gemspec の仕様記述

次の例として，Ruby の特徴を活かした DSL の Gem（Ruby のライブラリ配布パッケージ）作成パラメータの定義を示す．

リスト）serverspec の gemspec（抜粋）

```
Gem::Specification.new do |spec|
  spec.name           = "serverspec"
  spec.version        = Serverspec::VERSION
  spec.authors        = ["Gosuke Miyashita"]
(略)
  spec.add_development_dependency "rake", "~>
  10.1.1"
end
```

（http://svn.ruby-lang.org/cgi-bin/viewvc.cgi/trunk/lib/rubygems/package_task.rb?view=markup）

ここで利用されているプログラミング要素はブロックに与えたオブジェクトのプロパティ設定である．Gemspec の定義は，Rake の Gem タスクへ与える定義ファイルの一部である．すでに説明したように Rake の定義ファイルはそれ自身が Ruby スクリプトである．

利用者が記述するのは Gem モジュールの Specification クラスのインスタンス生成メソッド new へ与えるブロックである．利用者はブロックへ与えられたパラメータ spec を操作して設定する．

この例ではブロック以外にも Ruby による DSL の作りやすさの特徴を見ることができる．Ruby では末尾が = で終わる単一パラメータのメソッド呼び出しを代入のように記述できる．そのため spec.name=(“serverspec”) というメソッド呼び出しを spec.name = “serverspec” として記述できるのである．なお，このようなオブジェクトの属性にアクセスするメソッドをアクセサメソッドと呼ぶ．

Ruby の機能からははずれるが，この例でもう 1 点特徴的なのは add_development_dependency メソッドに与える 2 つのパラメータである．例では "rake" と "~> 10.1.1" を引数として与えているが，Rake のバージョン 10.1.1 よりも大きいものが Gem の開発（具体的にはテストの実行）に必須であることが明確に示されている．工夫された DSL においては，パラメータとして利用する文字列の記述則や，被設定側オブジェクトのメソッド名（この例であれば name や version）の自明さも重要である．

例 3）バイト配列のフィールド分割処理用クラスの生成ライブラリ（HexStruct）

次に設定ファイルではなく，プログラム内に記述して複数のフィールドを持つバイナリデータを各フィールドに分割するためのクラスを生成するライブラリ HexStruct の利用例を示す．

リスト）4 つのフィールドを持つ定義

```
FixedSizeFrame = HexStruct.define {
  fixed_size_field :MsgCode, 2
  fixed_size_field :from_add, 3
  fixed_size_field :to_add, 3
  variable_size_field :data
}
```

（http://magazine.rubyist.net/?0011-CodeReview）

このリストは，それぞれ 2, 3. 3 バイトの固定長フィールドと可変長フィールドを持つデータ用クラス FixedSizeFrame を生成するコードである．

ここで利用しているのは，例 2 と同じくブロック内にユーザ定義を記述させる方法である．ただし例 2 では設定対象のインスタンスをブロックへパラメータとして与えているが，ここではモジュールの評価関数を利用することで記述を不要化している．そのため，ブロック内の記述は直接 fixed_size_field などのキーワードのように見えるメソッド名から書きはじめることができる．また，フィールド名を示す最初のパラメータにシンボル型を採用してクォーテーションの不要化，パラメータリストを囲むカッコの省略など，Ruby の特徴を活用してプログラムらしい見た目を抑制して宣言的に記述している．

例 4）Web アプリケーションの記述（Sinatra）

最後の例としてWebアプリケーションフレームワークのSinatraを利用したアプリケーションの記述例を示す.

リスト）Sinatra添付サンプル（streamを利用したレスポンス例）

```
class Stream < Sinatra::Base
  get '/' do
    content_type :txt

    stream do |out|
     out << "It's gonna be legen -\n"
      sleep 0.5
      out << " (wait for it) \n"
      sleep 1
      out << "- dary!\n"
    end
  end
end

run Stream
```

（https://github.com/sinatra/sinatra/blob/master/examples/stream.ru）

リストはWebサーバの / に対するクライアントからのgetリクエストに対して単純なテキストを返すWebアプリケーションである.

Sinatraは，継承を利用してユーザ固有の処理を記述する方式のクラスライブラリである．したがって，ユーザコードは派生クラスのプログラムとなる．しかしサンプルが示すように，その記述方法はリクエストメソッドに対応するメソッド（ここでは`get`）に対して処理対象のURI（ここでは`'/'`）とレスポンス生成用のブロックを与えるというものだ．ブロック内ではヘッダ変数を設定するための専用メソッド（ここでは`content_type`）も利用できるため，開発されるプログラムはDSLを利用したレスポンスの定義に近いものとなる.

DSL開発言語としてのRuby

本稿では4種類の例を挙げてRubyによるDSLの実装を示した.

ここで示した例のうち最初のものはevalを持つスクリプト言語であればほぼ同様なことができる．しかし，リフレクションを利用せずに単純なevalのみでselfの指定なしにインスタンス変数を設定できるのはRubyが採用したスコープ指定子 @ による効果である.

2番目以降の例ではブロック，パラメータリストのカッコの不要，アクセサメソッドといったRubyならではの文法がDSLを強力にサポートしていることが分かる．Rubyにはそれ以外にも3番目の例で触れたシンボル型や，例には出現しないがハッシュの => 記法や % を利用した文字列配列記法などの，設定ファイルや特殊な宣言的記法に見せかけることが可能な書き方が用意されている.

マクロを利用できるプログラミング言語であればより柔軟なDSLを作れる．しかしトレードオフとして元のプログラミング言語の文法を逸脱することになる．Rubyを利用したDSLは，Rakeの作者が強調したように「Rubyの文法だけを知っていればよい」ことが強みである.

RubyがDSL開発言語として優れているのは，自身が備える文法のみを利用して設定ファイルや宣言的な記述の列挙のような見かけが作れることなのである.

（2015年7月27日受付）

田島暁雄　artonx@yahoo.co.jp

流通業向けの比較的大規模なOLTPシステムの端末からセンターシステムまでの設計，開発に従事．なお，本稿の内容は著者個人の見解であり，所属先の立場，戦略，意見を代表するものではない.