国際研究論叢　23（3）：153～162，2010

研究ノート

Ruby on Rails の指導の導入部分に関する考察

石　川　高　行*

Beginning Teaching Ruby on Rails

Takayuki Ishikawa *

Abstract

Ruby on Rails is the first spread, full-stack web application framework in the world. David Heinemeier Hansson, a Danish programmer, built Rails in 2004, and his book *Agile Web Development with Rails* is revised up to 3rd edition now. This paper makes a comparison with the tutorial part, the beginning of this book and other Rails textbooks.

Rails is an implementation of Model-View-Controller architecture. The tutorials of Rails textbooks have differences in teaching order according to the authors' viewpoint; Model-driven development or View-Controller driven development.

"Has-many" relationship is one of the important topics in teaching web application development. The tutorial in Hansson's book doesn't contain this topic, but some other textbooks include it in their tutorials.

Key Words

Ruby on Rails, web application, teaching programming, comparison of textbooks

1　はじめに

現実の社会の様々な業務を効率化し新しい価値を生み出すため、ICT は次々と新しい物を生み出し、標準化してきた。第２次世界大戦後に computer が作られ、network が整備され、database が標準化され、web application が普及してきた。そして、2004年に David Heinemeier Hansson（以下 DHH と略）によって作られた Ruby on Rails（以下 Rails と略）は、世界で初めて普及した full-stack web application framework として、従来の web application 開発を劇的に効率化し、またこれまで web application 開発に縁がなかった人々を web application の世界へと惹きつけている。

＊いしかわ　たかゆき：大阪国際大学現代社会学部准教授〈2009.12.22受理〉

本論は、この Rails を初めて学ぶ学習者に対し導入部分をどうやって指導するか、という点を理論的に考察することを目的とする。特に、Rails の作成者である DHH が著者となっている『Rails によるアジャイル Web アプリケーション開発』[1]（以下 AWDwR と略）も最近第３版が発行されたため、これやその他の Rails 書籍を比較することで、論を進める（一部、どの書籍にも掲載されていない点の考察も行う）。教授行為に関する研究は原則として実践によって理論の良し悪しを判断すべきものであるが、Rails が生まれてまだ５年しか経っていないため一部の仕様が固まりきっていないにも関わらず、Rails は広く普及し、大学における Rails 指導が求められているため、実践の前段階としての理論的考察にも一定の意味があるものと思われる。

２　前提

ここでは、個別の事項の比較・考察を行う前に、議論の範囲を定める。

２．１　「Ruby on Rails の指導」の範囲

情報教育に携わる者が「Ruby on Rails の指導」と言った際、この言葉がどこまでの範囲を指しているのかは定義次第である。さしあたって本論では、IP address や HTML など network の基礎を知っていて SQL 文など relational database（以下 RDB と略す）の基礎を知っている程度を「Ruby on Rails の指導」の入口とし、Rails によって基本的な web application を構築することができる程度を出口とする。

なお、web application を構築する際には data modeling や database 設計が欠かせないが、こういった教育内容については「Ruby on Rails の指導」に続く単元によって指導されることを想定している。つまり、「Ruby on Rails の指導」には data modeling や database 設計の内容を含めない。

また筆者は、入口に関して Rails を指導する際に同時に HTML や SQL 文を指導することもある程度可能であると考えている。そのため、前出の入口の定義はあまり厳密なものではなく、実際に指導案を作る際にはもっと間口を広くする（HTML や SQL 文を知らない者でも Rails を学ぶことができるようにする）予定である。

２．２　開発環境準備の教育と開発の教育を分離する

市販されている Rails 関連の書籍の大部分が、Rails の導入（installation）手順の説明に多くの頁を割いている。書籍は独習者を想定して書かれているものであるから導入の説明に多くの頁が割かれることは仕方がないことであるが、Rails の導入方法を説明すること（開発環境準備の教育）と Rails を用いた開発方法を説明すること（開発の教育）とは別物であり、Rails の指導の本質は後者にある[2]。この Rails の指導では、大学において既に開発環境（Rails 及び Eclipse などの環境）が準備された状態を想定して、後者（開発の教育）のみを扱う。

３　いくつかの事項の比較・考察

ここでは、relationship の実装を除くいくつかの事項について書籍を比較・考察する。relationship の実装については、次節で取りあげる。

３．１　Model 駆動開発と View/Controller 駆動開発

MVC 構造を用いた web application 開発には、Model（以下 M と略）を考えた後に View/Controller（以下 VC と略）を充実させていく順序（M 駆動開発）と、VC を考えた後に M を充実させていく順序（VC 駆動開発）がある。これは、web application を「database（M）の上に VC を被せたもの」と見なすか「実現したい作業に合わせて VC があり、その手段として M をつけたもの」と見なすかの２つの見方に対応している。

この２つは、Ruby on Rails の書籍においては M と VC のどちらを先に教えるかという違いとなって現れる。AWDwR 第３版における最初の application は、p.32 （第４章）で

```
ruby script/generate controller Say
```

と入力して “Say” controller を作った上で、この controller 内に手作業で “hello” action を作り、これに合わせて hello.html.erb を作り、browser 上で表示させている。明らかに VC 駆動開発であり、この時点で M は存在していない。初めて M が登場する章は第６章（p.62）である。

```
ruby script/generate scaffold product title:string
description:text image_url:string
```

これに対し、『はじめての Ruby on Rails ２』[3] のように、scaffold でいきなり M（そして同時に VC）を作ってしまう書籍もある（第３章 p.74）。

```
ruby script/generate scaffold member name:string
acc:string
```

同じ著者の書籍でも、『はじめての Ruby on Rails』[4]では、Rails ２.X[5] ではなく Rails １.１.２ を用いているため、scaffold によって MVC 全てを同時に生成することができず、M から作っている（第４章 p.99）。VC は後から作成している（p.101）。

```
ruby script/generate model flower
```

指導の順序として M と VC のどちらからがよいか、については、web application についての先述の２つの見方に帰するものであり、ここでは論じない。

3.2 scaffold plugin と scaffold generator

M の内容を指定するとその CRUD[6] を担う VC を自動で生成してくれる scaffold には、plugin として実装されているものと generator との２種類がある。plugin の場合、

```
Class FlowerController < ApplicationController
  Scaffold :flower
end
```

の「scaffold :flower」のように１行書き加えるだけで、M に合わせた VC を実行時に実現してくれる。ただし、VC の中身がどのように実装されているかを知ることは難しく[7]、教育向けではない。

一方、scaffold generator は flowers_controller.rb[8] や show.html.erb[9] というような具体的な VC files を生成してくれる。学習者は、生成された VC files の内容を読むことで、どのように記述するとどのように処理されるかを学ぶことができる。AWDwR 第３版 pp.402-405 にも scaffold generator によって生成された VC files の内容がそのまま掲載されている点からも、これが教育上有益であることが伺える[10]。

3.3 なるべく自動生成に頼ること

これは Rails の哲学である Convension over Configuration（設定より規約、以下 CoC と略）に関わる話である。前出の『はじめての Ruby on Rails ２』では、p.21 に次の記載がある。

> 「Ruby on Rails １」では、たとえば、
>
> ```
> ruby script/generate scaffold member
> ```
>
> とすれば、表「members」に関する至れりつくせりなアプリケーションが自動で作られましたが、「Ruby on Rails ２」では、
>
> ```
> ruby script/generate scaffold member name:string
> info:text
> ```
>
> のように、
>
> 表「members」の列「name」の値はテキスト・フィールドから、列「info」の値は複数テキスト領域から入力させてネ
>
> という説明をつけてやらなければならなくなりました。
>
> いや、もしかすると、本書の出たころには、また「Ruby on Rails １」のような簡単なコマンドで何もかもやってくれるように戻っているかもしれません。

しかし、この記述は正しくない。「表『members』に関する至れりつくせりなアプリケーションが自動で作られ」るためには、

```
ruby script/generate model member name:string info:
text
```

とした後に scaffold generator を利用するか、または

```
ruby script/generate model
```

とした後に手作業で migration file[11] を設定した後に scaffold generator を利用するか、のどちらかである必要がある[12]。著者は、まるで Rails 1.X よりも Rails 2.X の方が不便になったように記述しているが、実際には手間は軽減しており、改善である。この著者は前著『はじめての Ruby on Rails』p.99 において

```
ruby script/generate model flower
```

と記述しているため、M generator に options をつける前者の方法を知らなかったのではないかと思われる。この書籍に限らず、Rails 1.X の頃はこういった記述をしている書籍が多かった。

3.4 Rails の版（version）の指定

Rails は進歩が速く、書籍が対象としている版（version）と学習者が用いる版が異なると挙動が異なることが多い。このことが原因で学習に苦労した、という話は様々な blogs で見かける。

最大の原因は、その書籍で利用している Rails の版が明示されていないものが多いことにある。「本書のサンプルファイルは Rails 2.0 では動作しません。」[13]という記述がある書籍や、Rails を install する際に版を指定する書籍[14]もあるが、これらは少数派である。

Rails の版による挙動の違いは、書籍の説明によって防ぐことができるものもある。例えば、Rails による project 作成の第一歩は、以下の命令である（project の名前を「abc」とする場合）。

```
rails abc
```

このとき、Rails 2.0.1 以前なら MySQL へ接続する設定が自動的に生成され、Rails 2.0.2 以降なら SQLite 3 へ接続する設定が自動的に生成される。つまり、Rails 2.0.1 を対象として書かれた書籍に従って Rails 2.0.2 以降で「rails abc」などと入力した場合、MySQL 向けの設定が見つからず学習者は困ることになる。しかし、DBMS を指

定する option を明示することで、Rails の版に左右されずに目的の DBMS 向けの設定が自動的に生成される。こういった方法を記述している書籍もある[15]。

同様の問題に、web server として WEBrick を使うか mongrel を使うか、という問題もある。通常、Rails は WEBrick を web server として起動するが、新しい版の Rails では mongrel が存在する場合には WEBrick よりも mongrel を優先して起動する。これを避けるには、WEBrick を使うことを明示してやればよい。

3.5 test 手法を最初から教えるべきか

Rail の話ではないが、C 言語に `gets` という関数がある。この関数には致命的な欠陥があり、実際の programming では使ってはいけない関数とされている。こういった関数は、教育の場では使ってもよいだろうか。多くの人は「使ってはいけない」と答えるだろう。学習者によい programming 習慣を身につけさせるには、悪いものは使ってはいけない。

では逆に、良い習慣は何であっても早くから身につけさせるべきだろうか。Rails は、programming の際に test（unit test や function test など様々）を行うことを奨励している。「test first」と言って実際の code を書くよりも先に test を作ることを望ましいと考える開発者も少なくない。

しかし、Rails の書籍では、最初から test を作らせるものはない。これは、現場ではよいとされる習慣であっても学習の場では必ずしも最初から身につけさせる必要はない、という考えが主流であることを物語っている。これは、AWDwR 第３版の前半の大部分を占める tutorial（行うべき作業を順に Task A, Task B, …と名付けられている）において test が（ABC 順ではなく）Task T と名付けられてその他の tasks より後に書かれていること、また「開発の過程でアプリケーションの信頼性を少しずつ高めていけるよう、テストもインクリメンタルに書くのが理想です。テストは常時行うべきであり、前の章のあとにするものではないので、この章の作業は（タスク H ではなく）タスク T と呼ぶことにします。」[16]と書かれていることから、身につけるべき良い習慣であっても指導上の順序は後の方に配置するという意図がわかる。

3.6 “Skinny Controller, Fat Model”

これは、書籍に記載されていることではなく、とある blog の記事であるが、INTERNET 上で割と頻繁に話題になるものであるため、ここで取り上げる。

Rails では、その自由度の高さから、MVC のどの部分においても実際の処理内容を記述することが文法的に可能である[17]。だからといって、処理内容をどこに書いても良いというわけではない。このことに触れた記事が、「Skinny Controller, Fat Model」[18]である。ここでは、一定の条件を満たす人物を一覧表示するという処理について、V で行った例、C で行った例、M で行った例、の３例を挙げている。もちろん、どの例も意図通り正しく動くのだが、記事中に

> MVC has been successful for many reasons, and some of those reasons are “readability”, “maintainability”, “modularity”, and

“separation of concerns”.

と書かれている通り、可読性や保守性などを考えればそれぞれの処理は然るべき場所に（この場合は M に）書くことが望ましい、ということがこの記事の趣旨である。

実は、望ましい programming 習慣のような文法外の事柄は、多くの書籍で重視されていない。言い換えれば、文法事項さえ説明してしまえばそれで書籍の役割は終わりだ、と感じさせられる書籍はかなり多い。学習者に示すべき文法外の事項には一体どのようなものがあるか、を整理することも、今後の Rails 教育（ひいては programming 教育にも）に求められることであろう。

４　relationship の実装

４．１　relationship の種類

relationship とは、ある表と別の表との関連である。relationsip をしっかりと扱えるかどうかが、MS-Excel のような表計算 software と database との決定的な違いの１つであり、学習者は、relationship を学ばないことには database を学ぶ意義を充分に感じないだろう。

relationship には、１対１、１対多、多対多、の３種類がある。１対１の relationship が使われる状況は、手が加えられない legacy table に項目を追加したいなど特殊な状況に限られるため、ここでは検討しない。また、多対多の relationship は、１対多の relationship の応用である上に交差表（junction table）というものを用意しないといけないため、以下、１対多の relationship のみを検討することとする。

４．２　書籍での１対多の relationship の扱い

Rails の書籍の多くは、書籍の前半で tutorial を用意し、web application を実装していく様子を通じて Rails の基礎的な概念や手順を説明する。そして、書籍の後半で発展的な内容を扱う。

AWDwR 第３版では、tutorial（第５章～第14章）の中では１対多の relationship に関する記述はなく、第19章「Active Record その２: テーブル間のリレーションシップ」において触れられている。

『ライド･オン･Rails』[19]では、tutorial は１.３節にあり、１対多の relationship は２.２.６「ActiveRecord」において触れられている。

上記２冊はいずれも１対多の relationship を発展的内容として扱っていると言える。これに対し、『はじめての Ruby on Rails ２』では tutorial の中で１対多の relationship を扱っている（この書籍は、全編が tutorial であるという、他とは少々異なる構成となっている）。以下、この書籍の記述を見ていこう。

p.74 において、表「members」を以下のように生成している。

```
ruby script/generate scaffold member name:string
acc:string
```

この書籍では、このように scaffold generator によってできた１つの表とその CRUD VC を「小アプリケーション」と呼んでいる。２つ目の「小アプリケーション」として、表「places」を以下のように生成している。

```
ruby script/generate scaffold place name:string
info:text
```

その後、表「members」と表「places」とを１対多の relationship で結ぶため、p.171 で以下のように説明している。

> 小アプリケーション「places」で､小アプリケーション「members」を利用する場合、表「places」に表「members」の「id」列の値を入力する列を１つ作っておけばよい

のです。かつ、その場合、

> その列名を「member_id」にしておくと、楽だ

という決まりです。

この説明は、私が今までに読んだ Rails の書籍の中では最も丁寧なものである。なお、「member_id」列は適切な設定を行うことで foreign key（外部キー）となる。

４．３　table 構成の変更

この『はじめての Ruby on Rails』において「member_id」列を増やす操作は、残念なことに Rails の migration 機能を利用するのではなく SQLite３ の CUI で以下の SQL 文（使われることが少ない DDL の一種である ALTER 文）を直接入力している。

```
alter table places add member_id integer;
```

Rails の migration 機能は database 操作を抽象化して DBMS 毎の操作の違いを吸収することも目的の１つであるため、DRY 原則から考えても SQL 文を直接入力するより migration 機能を利用する方が望ましいと言える。

しかし、この migration 機能を用いても、Rails の初学者には負担が大きい。なぜなら、SQL 文を直接入力するよりはましであるものの、migration 機能は database の状態（どの migration file まで適用したか）を意識しなくてはならないからである。例えば、20091201000002_create_places.rb という migration file の内容を変更してもそれだけでは database には反映されず、rake db:migrate:redo step=XX という操作が必要となるからである（「XX」の部分は他の migration files に左右される）。

migration 機能を database に適用する場合は常に rake db:migrate:reset という操作を行う、という方法もある。この方法は、database がどのような状態であっても必

ず migration files の内容通りに設定されるので、初学者にとっては便利な方法である（この方法だけ覚えておけばよい）。ただし、これまでに入力した情報（tuples, records）が消えてしまうという難点がある。

以上の点を考慮すると、relationship については設計の段階でしっかり考えておき最初から scaffold generator で「ruby script/generate scaffold place name:string info:text member_id:integer」のように member_id を用意しておくことが初学者にとって最も負担が軽くなるものと思われる。

5　おわりに

2005年頃には種類も少なかった Rails の書籍も、現在（2009年）にはかなりの種類が出版されるようになった。しかし、Rails 自体の進歩が速すぎるため、どの書籍も出版の数ヶ月後（遅くとも半年後）には時代遅れとなっている（学習者が書籍の記載通りに操作してもその通りにならないようになっている）。比較・検討には、こういった出版年月の違いを切り分け、なるべく Rails 指導の本質的な部分を取りあげたつもりである。

本論では書籍（と blog）のみを取りあげたが、Rails 指導を考える際には開発環境の善し悪しの考察も欠かせない。これについては、別の機会に取りあげることとしたい。

文末注・引用文献

1 Sam Ruby, Dave Thomas, David Heinemeier Hansson, et al. 共著、前田修吾監訳『Rails によるアジャイル Web アプリケーション開発』第3版（オーム社 , 2009年）。原書 : Agile Web Development with Rails。頭文字をとって「AWDwR」と略される。

2 「Ruby on Rails 学習環境提供サービス」http://hitachisoft.jp/products/ruby_paas/ というものもあり、そこには「お客様専用の仮想サーバ上に構築済みの Ruby on Rails 実行環境と Rails アプリケーション群をセットにし、PaaS 型でご提供します。」とある。環境準備が Rails 指導の本質ではないことを端的に表す一例と言えるだろう。

3 清水美樹『はじめての Ruby on Rails ２』（工学社 , 2008年）。

4 清水美樹『はじめての Ruby on Rails』（工学社 , 2006年）。

5 Rails ２.X は、Rails ２.０.０ 以降で Rails ３.０.０ より前のものを表す。2009年12月現在、Rails ２.０.０ ～ ２.３.５ である。

6 Create, Read, Update, Delete のこと。database 内の tuple（row, record）に対する基本４操作。

7 事実上、scaffold generator によって生成される実装を理解していないと、自前の VC を作ることができない。

8 flower に関する C を納めた file。

9 １件の flower を詳細表示する際の表示内容を集めた（V を納めた）file。

10 また、Rails の scaffold generator が生成する内容は、Rails の版（version）によって微妙に異なる。これは新しい版の仕様に合わせた内容を生成するからである。初学者でなくても、Rails の新しい版が公開された時に scaffold generator の出力を見て新しい仕様について学ぶことは多い。

11 databse に table を適切に定義したり削除したりするための file。

12 前者の方法だと、migration file だけでなく test case や fixture などもある程度自動生成してくれるため、後者の方法よりも良い。

13 arton『10日でおぼえる Ruby on Rails 入門教室』（翔泳社 , 2008年）p.xxiv。

[14] 前出『はじめよう Ruby on Rails』p.26。
[15] 前出『はじめよう Ruby on Rails』p.46。
[16] AWDwR p.185。
[17] 自由度が高いことは、必ずしも良いこととは限らない。
[18] http://weblog.jamisbuck.org/2006/10/18/skinny-controller-fat-model
[19] 吉田和弘 , 馬場道明『ライド・オン・Rails』（ソフトバンククリエイティブ , 2006年）。