おつかれさまです、相澤です。
今日から Ruby on Rails による Web アプリケーションの開発手法を連載します。
質問や突っ込みなどはこのメールに返信してください。

Day1 目次

Rails とは

Ruby on Rails[1]はオープンソースの Web アプリケーションフレームワークです。
Ruby による柔軟な記述、メタプログラミングを活用して MVC アーキテクチャに基づく Web アプリケーションを高速に開発することが可能です。

Rails の哲学は主に下記の 2 つです。

- CoC（Conversion over Configuration：規約に従うことによって設定を不要とする）
- DRY（Don't Replay Yourself：同じことを繰り返さない）

Rails では Rails の作法に従ってコードを記述することで、従来必要だった設定ファイルをほとんど書くことなく、また必要な情報はアプリケーションのただ一箇所に記述することで必要な処理を実現することが可能です。
その代表的な例が Rails の OR マッピングで使用される ActiveRecord です。
ActiveRecord では

- テーブル名をモデルクラスの複数形[2]とする
- 主キーは id というカラムで整数型の自動インクリメント属性とする

という 2 つの規約を守ることでデータベースとの連携をほとんどコーディングすることなく実現することが可能です。
また、テーブルにどんなカラムが存在するかはデータベースのスキーマ情報を参照するため従来のドメインオブジェクトのようにテーブルカラムと一対のフィールドを定義しなくても、それが存在するかのごとく取り扱うことができます。

このように Rails を用いることで従来の開発で必要だった設定ファイル（xml で記述されることが多い）を書く必要が無くなり、開発者はアプリケーションの機能の本質部分だけを実装することに集中することができます。
だたし、Rails による高い生産性を得るためには Rails の作法（規約）について十分に習熟している必要があり、Rails の作法によらない動作を実現しようとする場合にはとたんに開発が難しくなるという欠点もあります。

Rails による開発の基本戦略は第一に **Rails の規約に則って開発する**ことが成功のカギとなります。
Rails の作法を習得するために最も効果的な方法は、Rails を使用して実際にアプリケーションを構築することです。
それでは実際に Rails を使用して簡単なアプリケーションを作成してみましょう。

[1] 単に Rails と呼ばれることも多いです。RoR と表記されることもあります。
[2] 日本人にはちょっとわかりにくいですが、複数形の不規則変化も提供されます。例えば Peason クラスに対応するテーブル名は people です。

STEP0. 開発環境の準備

Ruby 本体のバージョンが 1.8.6 以上である必要があります。Ruby のバージョンを確認するためには下記のコマンドを実行します。（もちろん環境変数の設定も忘れずにおこなってください）

```
ruby -v
```

また、Rails を使用するためにはいくつかの Ruby のライブラリがあらかじめインストールされている必要があります。これらはいずれも RubyGems によってオンラインでインストールすることが可能です。
Rails 本体は下記のコマンドでインストールできます。

```
gem install rails
```

rails のインストールをおこなうと、依存する以下のライブラリも同時にインストールされます。

表 1 Rails の依存ライブラリー覧

ライブラリ名	説明
ActiveSupport	Rails アプリケーションで使用されるさまざまなコア拡張ライブラリ
ActionPack	Rails アプリケーションの View、Controller 部分を実現するライブラリ
ActiveRecord	Rails 標準の OR マッピングのためのライブラリ
ActiveResource	外部リソースをシームレスに取り扱うためのライブラリ
ActionMailer	メールに関する操作をおこなうためのライブラリ

rails が正しくインストールされたことを確認するには以下のコマンドを実行します。

```
rails -v
```

なおこの連載ではとくに言及がない限り Rails 2.3.2 をインストールしたものとして進めていきます。
また、Rails でアプリケーションを構築する場合にさまざまな一連のタスクを実行するために Rake というライブラリが必要です。Rake をインストールするには下記のコマンドを実行します。

```
gem install rake
```

rake が正しくインストールされたことを確認するには以下のコマンドを実行します。

```
rake --version
```

RubyGems のバージョンが古い場合には上記のコマンドが失敗したり、Rails が上手く動かなかったりします。RubyGems のバージョンは 1.3.1 以上のものを使用してください。RubyGems のバージョンを確認するためには下記のコマンドを実行します。

```
gem -v
```

古い RubyGems を更新するには下記のコマンドを実行します。

```
gem install rubygems-update
```

さらにその後、下記のコマンドを実行することで RubyGems を更新することが可能です。

```
update_rubygems
```

また 1.3.1 以上のバージョンの RubyGems を最新のものに更新するためには下記のコマンドを実行します。

```
gem update --system
```

さらにデータベースとの接続をおこなうために、対応する RDBMS 用のドライバが必要です。この連載では RDBMS に SQLite3 を利用しますので、下記のコマンドで SQLite3 のドライバをインストールしてください。

```
gem install sqlite3-ruby
```

Debian 系の Linux ではあらかじめ ruby-dev と libsqlite3-dev がインストールされていないとインストールに失敗します。これらのライブラリは下記のコマンドでインストールします。

```
apt-get install ruby-dev
apt-get install libsqlite3-dev
```

Windows 環境では提供されている gem の都合により最新版のドライバをインストールすることができません。Windows 環境で作業をする場合は下記のコマンドでバージョン 1.2.3 をインストールします。

```
gem install sqlite3-ruby --version '=1.2.3'
```

sqlite3-ruby が正常にインストールされれば Rails からのテーブル操作は可能ですが直接 DB ファイルの中身をみるときのために SQLite3 本体もインストールしておくと便利です。
SQLite3 本体は下記のサイトからダウンロードしてください。
・SQLite Download Page
http://www.sqlite.org/download.html

Windows 環境用にコンパイルされたものは sqlite-3_6_14.zip です。ページの中ほどにリンクがあります。
zip ファイルを解凍すると sqlite3.exe が取得できますこれをパスの通ったフォルダにコピーすればインストールは完了です。正常に起動することを確認するにはコマンドプロンプトで以下のコマンドを実行します。

```
sqlite3 --version
```

なお、Debian 系の Linux 環境では下記のコマンドを実行するとインストールされます。

```
aot-get install sqlite3
```

STEP0 のおさらい
下記の開発環境が正常にインストールされていることを確認しましょう。

ツール	バージョン	インストール	確認方法
ruby	1.8.6 以上	ActiveScript Ruby をダウンロードしてインストールし、環境変数 PATH を設定する	`ruby -v`
rails	2.3.2	gem install rails	`rails -v`
rake	0.8.4	gem install rake	`rake --version`
gem	1.3.1 以上	gem install rubygems-update したのち update_rubygems `または gem update --system	`gem -v`
sqlite3-ruby	1.2.3 以上	gem install sqlite3-ruby または gem install sqlite3-ruby –version '=1.2.3'（Windows）	`gem list`
sqlite3	3.6.1	http://www.sqlite.org/download.html から sqlite-3_6_14.zip をダウンロードし、解凍されたファイル sqlite3.exe を PATH がとおっているフォルダにコピーする	`sqlite3 --version`

STEP1. Rails アプリケーションの雛形を作成する

rails コマンドを用いて任意の場所にアプリケーションの雛形を作成することができます。
この連載では作業フォルダを C:¥work にしたものとしてすすめていきます。異なるフォルダを作業フォルダにした場合には適宜読み替えてください。

まずコマンドプロンプトで C:¥work に移動して下記のコマンドを実行します。

```
rails todo
```

todo の部分がプロジェクト名になります。アプリケーションの挙動には影響しないので何でも構いません。
実行結果は以下のようになり、雛形となるフォルダやファイルが自動的に作成されます。

```
C:\work>rails todo
      create
      create  app/controllers
      create  app/helpers
      create  app/models
      create  app/views/layouts
      （中略）
      create  log/server.log
      create  log/production.log
      create  log/development.log
      create  log/test.log
```

work の中に todo というディレクトリが作成され、その中に雛形の構成が作成されていることを確認してください。以下の説明ではこの todo ディレクトリのパスを RAILS_ROOT と表記します。RAILS_ROOT 直下に作成される各ファイル・ディレクトリの役割は下記のとおりです。

表 2 Rails のディレクトリ構成

ディレクトリ名	説明
app	Rails アプリケーションの本体が格納される場所
config	Rails アプリケーションの設定ファイルが格納される場所
db	Rails アプリケーションが使用する DB スキーマ情報などが格納される場所
doc	Rails アプリケーションのドキュメント類が格納される場所
lib	Rails アプリケーションで使用するライブラリを格納する場所
log	サーバーの実行ログなどが格納される場所
public	静的な HTML ファイルなどが格納される場所
Rakefile	Rails アプリケーションを開発するために必要な一連の自動実行タスクを定義しているファイル
README	Rails アプリケーションの説明を記述するファイル
script	Rails アプリケーションを開発するために必要なスクリプト類が格納される場所
test	Rails アプリケーションのテストスクリプトが格納される場所
tmp	サーバー起動時の一時ファイルが格納される場所
vendor	機能拡張のためのプラグインが格納される場所

STEP1 のおさらい

下記のコマンドを実行し、Rails アプリケーションの雛形を作成します。

```
rails todo
```

STEP2. Rails アプリケーションを作成する

それでは rails アプリケーションの雛形ができたので、さっそく ToDo 管理をおこなう簡単なアプリケーションを作成してみます。

Rails で最も簡単にアプリケーションを作成するためには scaffold という機能を使用するのが一般的です。
RAILS_ROOT で下記のコマンドを実行します。

```
ruby script\generate scaffold todo title:string due:date priority:integer
```

上記のコマンドは script ディレクトリにある generate スクリプトの scaffold 機能を実行するものです。
generate スクリプトで scaffold を実行するときの書式は下記のとおりです。

```
generate scaffold モデル名（単数形）[カラム名:型] ...
```

モデル名は DB のテーブルに紐付くモデルクラスの名前を指定します。Rails の作法に従う場合、モデル名は名詞の単数形にします。今回は ToDo リストを作成するのでモデル名は todo としました。このモデルクラスに関連付けられるテーブル名は自動的に todos（モデル名の複数形）になります。

モデル名に引き続いて指定するのはそのモデル（≒テーブル）にもたせるカラムに関する情報です。とりあえず ToDo 管理に必要になりそうな 3 つのカラム（title、due、priority）を指定しています。カラム名の:（コロン）の右側にはそのカラムの型を指定します。今回は title は文字列型、due は日付型、priority は整数型としました。

なお、このときに指定できる型の一覧は下記のとおりです。

表 3 generate スクリプトで指定できる型一覧

指定する型	DB への反映
string	文字列型
text	テキスト型
integer	整数型
float	浮動小数点型
decimal	固定小数点型
datetime	日付時刻型
timestamp	タイムスタンプ型
time	時刻型
date	日付型
binary	バイナリ型
boolean	真偽値型

ここまでの作業で Ruby 側のアプリケーション構築は完了です。（これだけです！）あとは今作成したテーブル情報を DB に反映させる必要があります。
どのようなテーブルが DB に作成されるかを確認するためには RAILS_ROOT¥db¥migrate に作成された 20090507171612_create_todos.rb というファイルをみます。数字の部分は generate スクリプトを実行した時間のタイムスタンプです。

20090507171612_create_todos.rb

```
class CreateTodos < ActiveRecord::Migration
  def self.up
    create_table :todos do |t|
      t.string :title
      t.date :due
      t.integer :priority

      t.timestamps
    end
  end

  def self.down
    drop_table :todos
  end
end
```

テーブル情報は Ruby の DSL（Domain Specific Language：ドメイン固有言語）としての機能を生かして Ruby コードとして記述されています。これは SQL の DDL（Data Definition Language：データ定義言語）に相当する Ruby スクリプトで、マイグレーションファイルと呼ばれます。

今回の場合 CreateTodos というクラスに2つのクラスメソッド[3]up と down が定義されました。up は DB へのマイグレーションファイルの反映を進めるときに実行され、down はマイグレーションファイルをロールバックするときに実行されます。

SQL を理解していれば up メソッドや down メソッドが何をしているか理解できますよね？up メソッドは SQL の

```
  create table todos (
    title string,
    due date,
    priority integer
  );
```

とほぼ同義ですが、Rails の規約により暗黙につくられるカラムを含めると完全に等価な SQL は下記のようになります。

```
  create table todos (
    id integer primary key autoincliment, -- 主キーは id という名前で整数型の自動インクリメント
    title string,
    due date,
    priority integer,
    created_at timestamp, -- レコード作成タイムスタンプがセットされる
    updated_at timestamp  -- レコード最終更新タイムスタンプがセットされる
  );
```

[3] 正確には特異メソッドですが、簡単のため連載の中ではクラスメソッドと表記します。

同様に down メソッドは

```
  drop teble todos;
```

と等価です。up や down の中で使用されている create_table メソッドや drop_table メソッドは ActiveRecord::ConnectionAdapters::SchemaStatements モジュールに定義されています。メソッドの詳細は下記の URL を参照してください。

・Rials API/Module
ActiveRecord::ConnectionAdapters::SchemaStatements
http://api.rubyonrails.org/classes/ActiveRecord/ConnectionAdapters/SchemaStatements.html

STEP2 のおさらい
下記のコマンドを実行し、ToDo リストアプリケーションを作成します。

```
ruby script\generate scaffold todo title:string due:date priority:integer
```

STEP3. マイグレーションファイルをマイグレートする

さて、それではこのマイグレーションファイルを DB に反映させてみましょう。マイグレーションファイルを DB に反映させることをマイグレートするといいます。
マイグレートをするには Rake を用いるのが簡単です。RAILS_ROOT で下記のコマンドを実行します。

```
C:\work\todo>rake db:migrate
(in C:/work/todo)
==  CreateTodos: migrating ====================================================
-- create_table(:todos)
   -> 0.0160s
==  CreateTodos: migrated (0.0160s) ===========================================
```

マイグレートに成功すると上のような実行結果となります。このとき実行されたのはさきほどの up メソッドです。
ちなみに今おこなったマイグレートを取り消したい場合（ロールバックしたい）は以下のコマンドを実行します。

```
C:\work\todo>rake db:rollback
(in C:/work/todo)
==  CreateTodos: reverting ====================================================
-- drop_table(:todos)
   -> 0.0000s
==  CreateTodos: reverted (0.0160s) ===========================================
```

このときに実行されたのは down メソッドです。

rake に指定した db:migrate や db:rollback のことを Rake タスクといいます。Rake タスクは RAILS_ROOT¥Rakefile に定義されており、自分で追加することも可能です。現在使用できる Rake タスクの一覧をみるには以下のコマンドを実行します。

```
C:\work\todo>rake --task
(in C:/work/todo)
rake db:abort_if_pending_migrations       # Raises an error if there are pe...
rake db:charset                           # Retrieves the charset for the c...
rake db:collation                         # Retrieves the collation for the...
rake db:create                            # Create the database defined in ...
       （中略）
rake tmp:clear                            # Clear session, cache, and socke...
rake tmp:create                           # Creates tmp directories for ses...
rake tmp:pids:clear                       # Clears all files in tmp/pids
rake tmp:sessions:clear                   # Clears all files in tmp/sessions
rake tmp:sockets:clear                    # Clears all files in tmp/sockets
```

各 Rake タスクの詳細な機能は出力結果の右側のコメントをみてください。

スキーマ情報を rollback してしまったので忘れずに `rake db:migrate` を実行しておいてください。
なんとこれで Rails アプリケーションは完成です。

STEP3 のおさらい

下記のコマンドを実行し、マイグレーションを実行します。

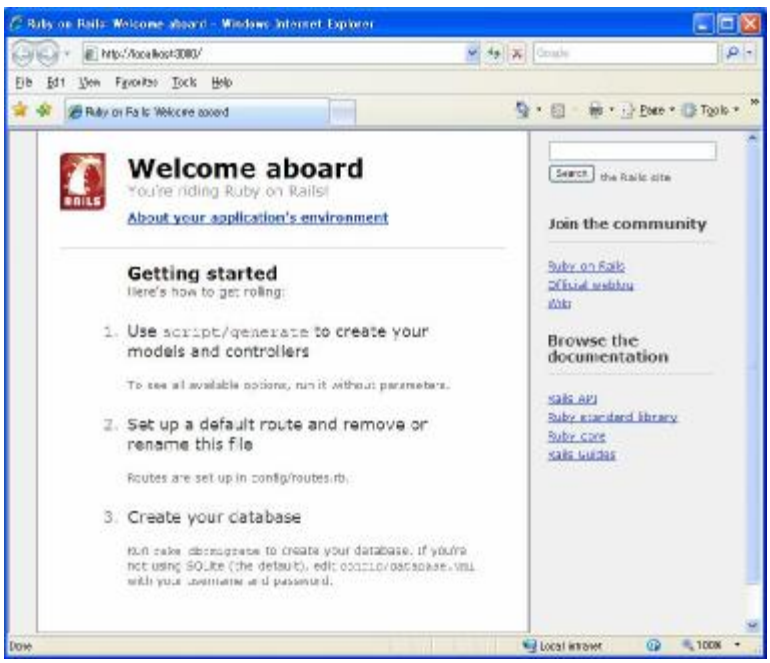
```
rake db:migrate
```

STEP4. サーバーを起動し動作確認をする

それでは実際に Rails アプリケーションを動作させて動きを確認してみましょう。
Rails にはアプリケーションサーバーの起動スクリプトもそろっています。とくに何もしない状態では Ruby 標準添付の WEBRick サーバーをテスト用として用います。WEBRick サーバーは本番用のサーバーとしては力不足ですが機能レベルの単体テストをおこなうには十分です。
サーバーを起動するには RAILS_ROOT で以下のコマンドを実行します。

```
C:\work\todo>ruby script\server
=> Booting Mongrel
=> Rails 2.3.2 application starting on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
```

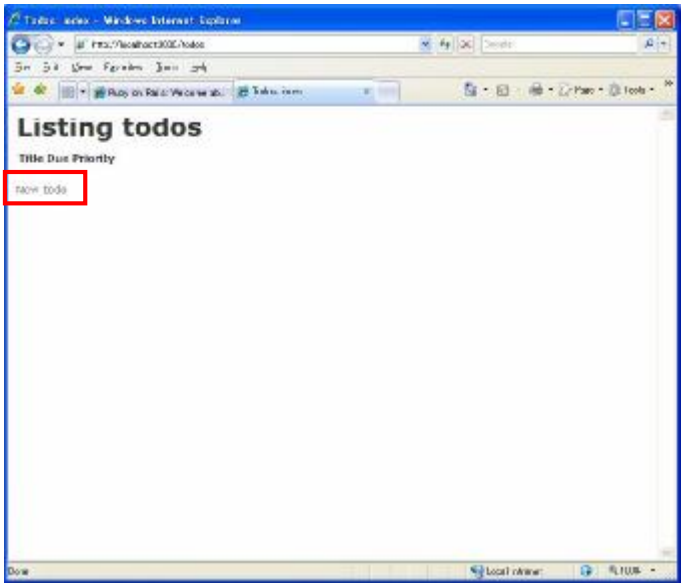
サーバーが localhost の 3000 番ポートで起動しました。ブラウザで http://localhost:3000 にアクセスしてみましょう。



"Welcome aboard"というタイトルのページが表示されましたか？これが Rails のデフォルトのトップ画面です。

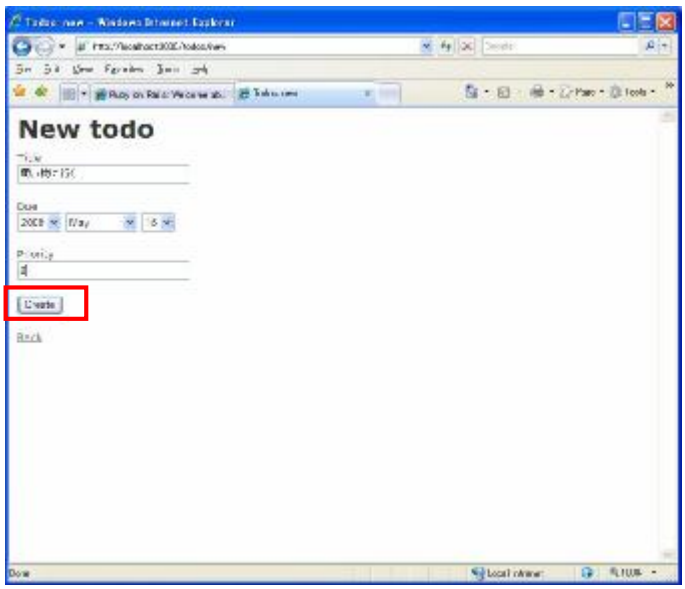
それでは先ほど作成した ToDo 管理アプ理ケーションにアクセスしてみます。以下の URL をブラウザのアドレスバーに入力してください。

http://localhost:3000/todos



"Listing todos"というタイトルのページが表示されていれば成功です。todos テーブルに対する CRUD 操作を備えた Web アプリケーションが既に動作しています。ためしにいくつかの ToDo を入力してみましょう。新しい ToDo を作成するには"New todo"というリンクをクリックします。

"New todo"という画面が表示されたはずです。Due の入力欄が日付を入力するセレクトボックスになっているのに気がついたでしょうか。これはスキーマ情報を作成するときに日付型を指定したので Rails が自動的に生成したものです。



なにか適当な ToDo を入力して「Create」ボタンを押すと ToDo の作成が成功した旨のメッセージが表示されますね。Back リンクをクリックして"Listing todos"画面に戻ると、確かに ToDo が作成されています。

"Listing todos"画面の ToDo の横に表示される Show で表示、Edit で編集、Destroy で削除ができます。削除の時にはちゃんと「Are you sure?」という確認のポップアップまで表示されます。気が利いていますね。

STEP4 のおさらい

下記のコマンドを実行し、サーバーを起動します。

```
ruby script\server
```

http://localhost:3000/todos をブラウザで開き、アプリケーションの挙動を確認します。

ToDo アプリケーションの動作を一通り確認したら今回は終了です。起動したサーバーを停止するにはサーバーを起動したコマンドプロンプトで Ctrl-C を押します。Windows 環境ではまれにサーバーの停止ができないことがあります。その場合はタスクマネージャーを起動して ruby.exe のプロセスを強制終了してください。

次回はこの ToDo リストを改造していきます。