小学生でもわかる

Ruby on Rails入門

by mitakalab

小学生でもわかる**Ruby on Rails**入門

by

Ryo Suzuki

Powered by OpenBook

§1 Ruby on Railsをはじめよう

1.1 Ruby on Railsをはじめよう

Ruby on RailsはRubyのWebフレームワークです。

ここでは、Railsの基本を学びましょう。

Railsってなに？

Railsはカンタンに説明すると、Webアプリケーションをカンタンに作れるためのツールだと思ってください。

Webアプリケーションを作ることが出来れば、非常にできることの範囲が広がります。

たとえば、iOSアプリやAndroidアプリの裏側にはWebアプリケーションがありますし、ありとあらゆるものの土台にWebアプリケーションがあります。

Webアプリケーション自体は、ほぼあらゆる言語で開発することができます。

しかし、プログラミング言語自体はWebアプリケーションだけを作るために設計されているわけではないので、効率がわるい点があります。

そこで、Webアプリケーションを開発するために設計されたものをWebアプリケーションフレームワーク、通称フレームワークと言います。

フレームワーク自体はプログラミング言語ではありません。フレームワークはある特定のことに特化した機能を集めたものです。

その理由は、最も先進的で安定しており、豊富なRubyGemsライブラリがあることで他のフレームワークに比べプロトタイプを作るスピードを早めることができるためです。

また、導入事例としては、Twitter、Github、国内だとCookPadなど非常にたくさんの導入事例があります。

また、Ruby自体たいへん素晴らしい言語であり、Rubyを使うことでプログラミングの理解を早く進められることができる点も、ひとつの理由です。

Railsの導入

Macの人はMacの環境構築を、Ubuntuの人はUbuntuの環境構築を参考にしてRubyをインストールしてください。

RailsはRubyGemsの一つとして公開されています。

Gemをインストールするようにしてインストールすることができます。

```
gem install rails
```

とすれば、最新のRailsがインストールされます。

実際に動か新柄読み進めるもよし、流れだけよんで、あとでRailsを試してみるもよし。

それでは、なにはともあれ、Railsの仕組みを学んできましょう。

§2 Railsがウェブページを表示する仕組み

2.1 Railsがウェブページを表示する仕組み

Railsに限らず、Webページを表示するためには、本質的にHTMLさえあれば表示できます。

HTMLというのはみなさんも、名前だけは聞いたことがあるかもしれませんが、HyperText Markup Languageの略で、マークアップという意味は、カンタンにいうとただ、ワードなどのように文字の大きさや色を変えているものだと理解してください。

すなわち、HTMLはプログラミング言語ではなく、ただ表示する文書を変えているだけです。

具体的には

```
This is Header
This is Text
```

これは、単純な文書ですが

```
<h1>This is Header</div>
<p>
  This is Text
</p>
```

とすると、HTMLだということになります。

基本的には、この文書の形を整えるHTMLだけでWebページ自体は作れてしまいます。事実世の中にあるすべてのWebページはHTMLを表示しているだけなのです。

それでは、なぜRuby on Railsのようなものが必要なのでしょうか？

その必要性を考えるために、たとえばTwitterのようなWebサービスを作りたい場合を考えてみましょう。

Twitterでは、たとえばRyo Suzukiというユーザーのプロフィールページを表示する場合は、次のようなHTMLを作ればいいわけでした。

```
<h1>Ryo Suzuki</h1>
<p>ryooopan</p>
<ul>
  <li>Location : Kanagawa, Japan</li>
  <li>About    : Hello, I'm Ryo.</li>
</ul>
```

これで全く問題ないわけですが、それではこのサービスにShohei Aokiという別のユーザーもいた場合次のようなHTMLも作らないといけません。

```
<h1>Shohei Aoki</h1>
<p>moyahima</p>
<ul>
  <li>Location : Tottori, Japan</li>
  <li>About    : Nice to meet you.</li>
</ul>
```

もしも、100人ユーザーがいたら、100個の別々のHTMLを作らなければなりません。

これは現実的ではありません。

しかし、HTMLはプログラミング言語ではないので、ただ文章を表示することしか出来ません。

そこで、必要となってくるのが、Rubyをはじめとしたプログラミング言語です。

すなわち、HTMLはテンプレートとして使い、それにRubyで中身を入れていくようにすれば、100人のユーザーのプロフィールを一つのHTMLだけを使って表示することが可能になります。

次のHTMLを見てください。

```
<h1><%= @user[:name] %></h1>
<p><%= @user[:username] %></p>
<ul>
  <li>Location : <%= @user[:location] %></li>
  <li>About    : <%= @user[:about] %></li>
</ul>
```

さきほどと少し変わっています。まず、具体的な名前やユーザー名が直接書かれているわけではなく、@user[:name]や@user[:username]に置き換わっています。

この@user[:name]などに当たる部分が、Rubyにあたるものです。すなわち、ユーザーの情報をRuby側で@userという箱（インスタンス変数と言います）につめ込んで、HTMLに渡してやればよい、という事です。

ここで、

```
<%= ... %>
```

で囲まれた部分はHTMLにRubyから渡したデータを埋め込んでいるところです。

Ruby側でユーザー情報を @user という変数に値をつめて、HTMLに渡せばそれぞれのページでプロフィールを表示することが可能になるわけです。

ここでででてきた、Webページを表示する部分、すなわちHTMLの部分をRailsでは「ビュー」と呼びます。

ビューの役割は非常にシンプルで、「HTMLを表示する」こと。ただ、それだけです。

しかし、これだけでは、ページを表示できません。

ビューに渡している@userという情報をどこかで、自分で設定しなければならないからです。

したがって、Ruby側で次のようにユーザー情報を入れます。

```
@user = Hash.new
@user[:name] = 'Ryo Suzuki'
@user[:username] = 'ryooopan'
@user[:location] = 'Kanagawa, Japan'
@user[:about] = 'Hello, I am Ryo.'
```

ここで@userという箱に値を詰めました。これではじめて、ビューで同じように表示できるようになりました。

この@userに情報をつめてビューに渡す役割を担っているのものを、Railsでは「コントローラー」と呼びます。

コントローラーの役割は、カンタンにいうと「ビューに値を渡す」ことです。

逆に言えば、Webページで表示したいことがひとつしかなければ、コントローラーは必要ありません。

ビューでページごとに表示する内容を変えたいときに、はじめてコントローラーが威力を発揮します。

> 注 このようなページごとに表示する内容を変えることを「動的なページ」と言います。

ここまでで、Railsを使ってWebページを作る方法が分かって来ました。

それでは、これを実際に作ってみましょう。

§3 実際にRailsアプリを作ってみる

それでは、Railsでアプリを作っていきましょう。

ここでは、先ほどのTwitterのようなアプリを mitakalab-twitter という名前でアプリを作ることにします。

Terminalで

```
rails new mitakalab-twitter
```

と打てば、自動でRailsアプリの型となるファイルがすべて自動で生成されます。

今作ったmitakalab-twitterのディレクトリに移動して、作られたファイルを見てみましょう。

```
cd mitakalab-twitter
ls
```

自動で作られたファイルはたくさんありますが、最初のステップで重要なのは次のファイルです。

```
- Gemfile
- app
    |- assets
  |- controllers
  |- helpers
  |- models
  |- views
- config
  |- routes.rb
- db
```

ここで出てきた app/controllers と app/views は先ほど出てきたビューとコントローラーが入るフォルダになります。

Railsでビューとコントローラーを作る場合は、Terminalで次のようなコマンドを入力します。

ここではユーザーの情報を表示するためのビューとコントローラーを作るので、usersとい

う名前をつけます。

```
rails g controller users index show
```

このindexというのは、ユーザーの一覧ページ。showというのは、ユーザーのプロフィールページです。

> 注 indexやshowは自分で自由に名前をつけることが可能ですが、あとで説明するようにindex、showというようにつけるのがよしとされています。 また、コントローラー名はusersというように複数形にするのがよしとされています。 こういう暗黙の約束事のようなものをRailsの規約と言います。 Railsはその名の通りレールに沿って作ればカンタンに作れる、という意味なので、出来る限りRailsが敷いたレールの上を走りましょう。

おそらく、次のようなファイルが自動でつくられていると思います。

```
app/controllers/users_controller.rb
app/views/users/index.html.erb
app/views/users/show.html.erb
```

ここでできた、users_controllerというのは、先ほど説明したまさにコントローラーです。

ビューの方もできていますね。

ここで注意深くみてみると、index.htmlではなくindex.html.erbとなっているのに気づいたでしょうか？

このerbはEmbed RuByの略。すなわちコントローラーから渡したrubyが埋め込まれているものなので、index.html.erbというようなファイルになっています。

実は、ここまででもうWebページが完成しています。

試しに、それを確認してみましょう。

Terminalで次のように入力するとサーバーが立ち上がります。

```
rails s
```

sはserverの略です。

このようにサーバーを起動させた状態で、Webブラウザで

```
http://0.0.0.0:3000/users/index
```

にアクセスしてみましょう。

次のようなWebページが表示されているのがわかるかと思います。

ここで表示されているのは、さきほど自動で生成された、app/view/users/index.html.erbです。

プロフィールページは

```
http://0.0.0.0:3000/users/show
```

にアクセスると確認することができます。

それでは、実際にビューとコントローラーを変更していきましょう。

§4 作ってわかるビューとコントローラー

4.1 作ってわかるビューとコントローラー

作ってわかるビューとコントローラー

それでは、まずはプロフィールページを先ほどのように、Twitterのようなプロフィールを表示したいので変更しましょう。

最初に言ったように、ページを変更するにはビューのHTMLを変えればいいだけです。

まずはコントローラーなどを使わずに、単純に app/views/users/show.html.erb を次のように編集してみましょう。

> 注 サーバーは起動したままで大丈夫なので、Terminalで別のタブを開くかSublimeTextで編集していきます。

```
<h1>Ryo Suzuki</h1>
<p>ryooopan</p>
<ul>
  <li>Location : Kanagawa, Japan</li>
  <li>About    : Hello, I'm Ryo.</li>
</ul>
```

ページを再読み込みして、次のような画面に変わっていれば成功です。

しかし、これだけでは、Railsを使っている意味がありません。ビューに表示する情報をコントローラーを使って動的に変えてはじめてWebアプリケーションになります。

なので、次に app/views/users/show.html.erb を次のように編集しましょう。

```
<h1><%= @user[:name] %></h1>
<p><%= @user[:username] %></p>
<ul>
  <li>Location : <%= @user[:location] %></li>
  <li>About    : <%= @user[:about] %></li>
</ul>
```

これで、ブラウザを再読み込みすると、次のようなエラーが表示されてしまいました。

Railsではうまくいかない場合は、このようにエラーを出してどこのファイルのどこにエラーが出ているかを丁寧に教えてくれます。

したがって、このエラーを注意深く読むことで、カンタンにバグを発見できるようになっています。

今のエラーは

```
undefined method `[]' for nil:NilClass
```

となっています。

これはすなわち@userから情報を引っ張ろうとしているけど、@userに情報が入っていないということ。

それもそのはずで、まだ@userという変数を定義していませんでした。

次に、コントローラーで@userに情報を埋め込まないといけません。

それでは、やってみましょう。

app/controllers/users_controller.rb を開くと次のようになっています。

```
class UsersController < ApplicationController
  def index
  end

  def show
  end
end
```

今は、 app/views/users/show.html.erb に @user という値を渡したいので、showを変更します。

```
def show
  @user = Hash.new
  @user[:name] = 'Shohei Aoki'
  @user[:username] = 'moyahima'
  @user[:location] = 'Tottori, Japan'
  @user[:about] = 'Nice to meet you.'
end
```

これで、もう一度再読み込みをすると、次のようにコントローラーから渡された値が表示できました。

しかし、これではまだまだです。

Twitterでは https://twitter.com/ryooopan というURLにアクセスすると、Ryo Suzukiのプロフィールが表示され、 https://twitter.com/moyahima というURLにアクセスすると、Shohei Aokiのプロフィールが表示されるようになっています。

しかし、今は、

```
http://0.0.0.0:3000/users/show
```

にアクセスすると、すべて同じ人のプロフィールになってしまっています。

これを

```
http://0.0.0.0:3000/users/show/ryooopan
```

にアクセスすると、ryooopanが

```
http://0.0.0.0:3000/users/show/moyahima
```

にアクセスすると、moyahimaがそれぞれ表示されるようにするためには、どうしたらよいのでしょうか。

§5 ルーティングという名の仕分け役：ここにアクセスされたらこれを表示しろ

5.1 ルーティングという名の仕分け役：ここにアクセスされたらこれを表示しろ

試しに、今

```
http://0.0.0.0:3000/users/show/ryooopan
```

にアクセスすると、次のようなエラーになっています。

何やらRouting Errorとなっています。

じつは、Railsではこのような仕組みが用意されていて、それはルーティングと呼ばれています。

どういうことかというと、ルーティングはユーザーがアクセスしたURLをもとに何を表示するかをすべて振り分けているのです。

すなわち、ルーティングの役割は、ユーザーがRyo Suzukiを表示しろというURLにアクセスしたら、その情報を表示するようにし、Shohei Aokiを表示しろというURLにアクセスしたら、その情報を表示するようにする、いわば「URLの仕分け」のようなことをしています。

したがって、実はURLにアクセスした時に最初に通るところがルーティングになるのです。

今までの流れを整理すると

```
ユーザーがURLにアクセス ⇒ ルーティングが仕分け ⇒ コントローラーが値を入れる ⇒ ビューがコントローラーから渡された値を表示
```

というこになります。

それでは、実際にはどのようにやればよいのでしょうか。

次にそれを見て行きましょう。

§6 作ってわかるルーティング

6.1 作ってわかるルーティング

ルーティングはconfig/routes.rbというところで設定されています。

実は、何も考えずに http://0.0.0.0:3000/users/index や http://0.0.0.0:3000/users/show にアクセスして表示していましたが、それは実はRailsが自動で作ってくれていたんですね。

今の時点で config/routes.rb は次のようになっていると思います。

```
MitakalabTwitter::Application.routes.draw do
  get "users/index"
  get "users/show"
  # The priority is based upon order of creation: first created -> highest priority.
  # See how all your routes lay out with "rake routes".
  ...
end
```

ここでは、先ほどのように

```
http://0.0.0.0:3000/users/show/ryooopan
```

でプロフィールページを表示するように変更したいので、次のように編集します。

```
MitakalabTwitter::Application.routes.draw do
  get "users/index"
  get "users/show/:username" => "users#show"
  # The priority is based upon order of creation: first created -> highest priority.
  # See how all your routes lay out with "rake routes".
  ...
end
```

ここでのポイントは、URLごとに変わる値（ここではryooopanやmoyahimaといった値）はコロンを付けて

```
"users/show/:username"
```

としている点です。

また、

```
... => "users#show"
```

はusersコントローラーのshowにいけ、という命令をしています。

ここまでで、

```
http://0.0.0.0:3000/users/show/ryooopan
```

にアクセスすると、Routing Errorは消え、無事にページが表示されました。

しかし、ryooopanにアクセスしているのに、Shohei Aokiのページが表示されてしまっています。

URLごとに表示するページを変えたいですね。

そういうときに、先ほど学んだビューとコントローラーの威力が発揮されてきます。

ビューとコントローラーをわざわざ分けたのは、ユーザーが表示はビューで、情報を変更するのはコントローラーで、というように分けれる点なのです。

それでは、いよいよ、プロフィールページを表示できるようにしていきましょう。

§7 ルーティングからコントローラーへ

7.1 ルーティングからコントローラーへ

ここで一度、config/routes.rb で書いたこと振り返ってみましょう。

```
get "users/show/:username" => "users#show"
```

ここで、:usernameはWebページを見る人がRailsに送っている情報です。

この情報のことを「パラメーター」と言います。

そして、実はこのパラメーターをコントローラーで受け取ることができます。

それはこのような形で、パラメーターを取得できるようになります。

```
params[:username]
```

ここは、はじめての人にちょっとわかりにくい部分なので、少し説明しますと、

```
http://0.0.0.0:3000/users/show/ryooopan
```

にアクセスすると、

```
params[:username] = "ryooopan"
```

という値が代入され、

```
http://0.0.0.0:3000/users/show/moyahima
```

にアクセスすると、

```
params[:username] = "moyahima"
```

という値が代入されます。

それでは、これを使って、コントローラーを変更してみましょう。

app/controllers/users_controller.rbのshowの部分を次のように編集します。

```
def show
  @user = Hash.new
  if params[:username] == 'ryooopan'
    @user[:name] = 'Ryo Suzuki'
    @user[:username] = 'ryooopan'
    @user[:location] = 'Kanagawa, Japan'
    @user[:about] = 'Hello, I am Ryo.'
  elsif params[:username] == 'moyahima'
    @user[:name] = 'Shohei Aoki'
    @user[:username] = 'moyahima'
    @user[:location] = 'Tottori, Japan'
    @user[:about] = 'Nice to meet you.'
  end
end
```

これで、それぞれのURLにアクセスすると、それぞれの情報を@userに入れて、ビューに渡すことができているはずです。

それでは、実際にブラウザでアクセスしてみましょう。

```
http://0.0.0.0:3000/users/show/ryooopan
```

にアクセスすると、

このようにRyo Suzukiのプロフィールが表示され

```
http://0.0.0.0:3000/users/show/ryooopan
```

にアクセスすると、

このようにShohei Aokiのプロフィールが表示されるようになりました！

これでかなりWebアプリケーションっぽくなって来ました。

とはいえ、ここではたと気づく人もいるかもしれません。

「もしも100人のプロフィールを作るとすると、コントローラーに100通りの情報を書かなければいけないのか...orz」

ちなみに、Twitterは現在ユーザーが1億人を超えていますので、もしこの方法で表示しよう

とする場合Twitter社は1億人の情報をコントローラーに書き込まなければなりません。

これは現実的ではありませんね。

そこで登場するのが、データベースと呼ばれるものです。

それでは、次にデータベースというものがWebアプリケーションを作る上で、なぜ必要なのかを説明していきましょう。

§8 データベースが必要な理由

8.1 データベースが必要な理由

データベースという言葉は聞いたことがある人もいるかもしれませんが、あまりピンと来ない人も多いのではないでしょうか？

ここでは一度Webをはなれて、身近な例のたとえば連絡先の管理という状況を考えてみましょう。

たとえば、友達に電話をかける時、その電話番号を覚えている人はいないと思います。

それでは、友達に電話をかけようと思った時に、正確に電話をかけるためにはどうしたらいいでしょうか。

まず、もっともカンタンな方法はどこかにメモを貼っておけばいいわけです。ポストイットなりに

```
Ryo Suzuki : 080-1234-5678
```

というようなメモを残しておけば間違えずにすみます。

しかし、小学校に入った途端友達が100人もできてしまった場合、電話のまわりがポストイットで埋まってしまいます。

そこで考えだされたのが、電話帳という画期的なシステムです。

今まで個別にメモを取っていたものを一箇所にまとめてしまえば、全員の連絡先はその電話帳にすべて入っているので探す必要がなく、また探すときもアルファベット順に並べておけば、すぐに見つけることができるようになります。

```
...
Ryo Suzuki  : 080-1234-5678
Shohei Aoki : 080-8765-4321
...
```

この電話帳のようなものをデータベースと呼びます。

データベースの役割は、「すべての情報をそこで一元管理することができる」ことと「見つけたい情報をすぐに見つけられるようにする」ことです。

話をRailsに戻しますと、ユーザーの情報はすべてユーザーのデータベースに入れておけば、表示したい場合はそこから情報を引っ張ってこれるようにできるわけです。

それでは、電話帳を作るためには、何が必要でしょうか。

次に電話帳を例にしてデータベースに必要なものを説明してきます。

§9 データベースに必要なもの：ノートと枠とその内容

9.1 データベースに必要なもの：ノートと枠とその内容

電話帳の例の場合、まず電話帳を書くためのノートが必要です。まずはノートを買ってきましょう。

買ってきたノートは真っ白なノートです。これを電話帳のように使うためには、何が必要でしょうか。

それは、まず次のように「どんな情報を入れる必要があるか」、という項目です。

その枠とは次のようなものです。

```
    id    |    name    |    phone    |    email    |    address
 ------------------------------------------------------------------
           |            |             |             |
          |            |            |            |
          |            |            |            |
          |            |            |            |
```

電話帳にこのような枠を書くようなイメージです。

このように枠を書いておけば、あとは新しい友達ができた時に、この枠の中に書いていけばいいわけです。

この枠のことを、Railsでは「スキーマ」といいます。

ここで買ってきたノートが1万ページくらいあるノートだとすると、これを電話帳だけに使うのはもったいないですね。実際、身の回りで管理したいものは電話帳だけではありません。

たとえば、読んだ本をまとめておくために、読書感想日誌のようなものを作りたい人もいるかもしれません。

その場合、同じノートの半分は、それに使おうと思うわけです。

読書感想日誌のスキーマは次のようになるかもしれません。

```
    id    |    book    |    author    |    date    |    note
-----------------------------------------------------------------
          |            |              |            |
          |            |              |            |
          |            |              |            |
          |            |              |            |
```

このように、大きな白いノートは保存しておきたい情報をすべてそこで管理するものになりました。

Webアプリケーションのデータベースもそれと同じです。

1万ページくらいある白いノートに当たるのが「データベース」、そこに最初に書き込む枠組みが「スキーマ」といいます。

```
データベース ： 1万ページの白いノート
スキーマ     ： 情報を入力するための共通の枠
```

ここまでで、データベースに必要なものが整いました。

次に、Railsの中でもイメージが難しいマイグレーションファイルについて説明していきましょう。

§10 マイグレーション：枠は直接書かずに、印刷する

10.1 マイグレーション：枠は直接書かずに、印刷する

さて、ここまでできて気をよくしてしまい、電話帳や読書日誌のデータベースを作ったと、学校で友達に漏らしてしまいました。

そうすると、うわさがうわさをよんで、みんなからオレのデータベースも作ってくれと頼まれてしまいました。

1万ページの白いノートにまた枠を全てのページに手作業で書くのは大変です。

もしも、PDFに枠が書いたものを保存しておけば、それを友達に渡して友達が印刷して使えるようにすればいくらでも友達に渡すことができるようになるわけです。

それと同じ仕組みが用意されています。

それがマイグレーションファイルと呼ばれるものです。（migrationは移住という意味です。）

Railsではデータベースを作るときに、この状況を最初から想定して、データベースのスキーマを直接書くのではなく、マイグレーションを書きます。

すなわち、先ほどの例の場合は、

ノートに直接枠組みを書く

のではなく

PDFファイルで枠組みを書いて、それを印刷してノートに貼る

という作業をします。

これをするメリットは、

1. 枠組みを直接書かないので、あとで変更することがラクに出来る

2. PDFという形でデータを作っておけば他の人が同じデータベースを作るときに、ラクに再現することができる

という点です。

このPDFに当たるものがマイグレーションファイルと呼ばれるものです。

そして、印刷することをマイグレートする、と呼びます。

> 注 migrationという名の通り、移住をしやすくするためのようなイメージです。 移住するときに家を立てなければなりませんが、家の設計図さえあれば、いくらでも同じ家を立てることができます。 その家の設計図がmigration fileです。しかし、このmigration fileだけでは、住めません。 家をたてるために、その設計図から家を実際に建てなければなりません。それがmigrateです。

ここまでで、データベースのほんのちょっと、なんとなくのイメージがつかめたかもしれません。

このほんのちょっとのイメージの状態でいいので、実際に作ってみましょう。

§11 作ってわかるデータベース

11.1 作ってわかるデータベース

Railsではデータベースを作るコマンドが用意されています。

Terminalで

```
rake db:create
```

と打つと、データベースが作られました。

データベースは実際にはどこにあるのかというと、db/development.sqlite3 というのがそれです。

> 注 データベースはテキストファイルではないので、emacsやSublimeTextなどで、中身を見ることはできません。

このデータベースにはまだまっさらで何も入っていません。

真っ白なノートを買ってきたところです。

次にやることは、ノートに直接枠を書く...ではなく、枠をPDFファイルに書くことでした。

それでは、実際にマイグレーションファイルを作りましょう。

ここで、電話帳からRailsで作っているTwitterアプリに頭を切り替えてください。

今欲しかったのはプロフィールを表示して、そのデータをデータベースに保存したい、というところでした。

それでは、データベースの枠は次のようになっていればいいですね。

```
    id    |    name    |    username    |    location    |    about
 ---------------------------------------------------------------------
          |            |                |                |
          |            |                |                |
          |            |                |                |
          |            |                |                |
```

このようにユーザーの情報を持っているひとかたまりをRailsではモデルと言います。

先ほどの例の場合、「電話帳モデル」とか「読書感想日誌モデル」などと言います。

それでは、実際にユーザーのモデルを作っていきましょう。

Terminalで

```
rails g model user name:string username:string location:string about:text
```

と入力します。

ここでstringというのはデータの種類で文字列を意味します。自己紹介のためのaboutは長い文章を入れるのでtextという長い文章を保存するためのデータの種類に設定してあります。

> 注 モデルはusersではなく、userという単数形であることに注意しましょう。 これもRailsの敷いたレールの一つです。

すると次のようなファイルが作られていると思います。

```
app/models/user.rb
db/migrate/20130630062417_create_users.rb
```

この app/models/user.rb がモデルのファイル。
db/migrate/20130630062417_create_users.rb がマイグレーションファイルです。

マイグレーションファイルは、この例の場合、2013年6月30日の6時24分17秒に作ったので、このようなファイル名になっています。

実際に中身を見て確認してみましょう。

db/migrate/20130630062417_create_users.rb をみてみると

```
class CreateUsers < ActiveRecord::Migration
  def change
    create_table :users do |t|
      t.string :name
      t.string :username
      t.string :location
      t.text :about

      t.timestamps
    end
  end
end
```

となっていてます。無事に作れているのがわかりますね。

しかし、このままでは、まだデータベースは使えません。

次にやることは、PDFファイルを印刷してノートに貼り付ける作業です。

これをRailsでは、次のようにやります。

Terminalで

```
rake db:migrate
```

と打つと、データベースの中にユーザーの枠がはじめて作られました。

§12 データベースの中にデータを入れる

12.1 データベースの中にデータを入れる

それでは、ユーザーのデータを実際に入れていきましょう。

とりあえず、今は試しにRyo SuzukiというユーザーとShohei Aokiというユーザーを入れることにしましょう。

それでは、db/seeds.rb というファイルがあると思います。

これは、初期データを入れるためのファイルなので、ここに書き込んでいきましょう。（seedというのは種という意味です。）

db/seeds.rb を次のように編集します。

```
@user = User.new
@user.name = 'Ryo Suzuki'
@user.username = 'ryooopan'
@user.location = 'Kanagawa, Japan'
@user.about = 'Hello, I am Ryo. I am from database!'
@user.save

@user = User.new
@user.name = 'Shohei Aoki'
@user.username = 'moyahima'
@user.location = 'Tottori, Japan'
@user.about = 'Nice to meet you. I am from database!'
@user.save
```

これを書き込んだら、Terminalで次のようにコマンドをうってください。

```
rake db:seed
```

これで晴れて、データベースにデータが入りました。

もしも、新しい人を追加したい場合は、同じようにして追加していけばいいわけです。

だいぶRailsの全容がつかめてきました。

それでは、次に実際にビューで表示する内容を、データベースからデータを取ってきて、

それを表示するようにしましょう。

§13 データベースのデータを表示する。

13.1 データベースのデータを表示する。

さて、ここまででデータベースの準備は完了です。

それでは、いよいよデータベースのデータを表示してみましょう。

表示する内容を決めるのはどこだったでしょうか？

そうコントローラーですね。

それでは、コントローラーを変更すればいいわけです。

今、app/controllers/users_controller.rb はこのようになっていると思います。

```
def show
  @user = Hash.new
  if params[:username] == 'ryooopan'
    @user[:name] = 'Ryo Suzuki'
    @user[:username] = 'ryooopan'
    @user[:location] = 'Kanagawa, Japan'
    @user[:about] = 'Hello, I am Ryo.'
  elsif params[:username] == 'moyahima'
    @user[:name] = 'Shohei Aoki'
    @user[:username] = 'moyahima'
    @user[:location] = 'Tottori, Japan'
    @user[:about] = 'Nice to meet you.'
  end
end
```

これを次のように編集してみましょう。

```
def show
  if params[:username] == 'ryooopan'
    @user = User.find_by(:username => 'ryooopan')
  elsif params[:username] == 'moyahima'
    @user = User.find_by(:username => 'moyahima')
  end
end
```

かなりスッキリしました。

これはデータはすべてデータベースに入っているので、わざわざここで書く必要はないからですね。

またデータベースは、検索ができるという点もポイントでした。（電話帳でアルファベット順にしていたことを思い出してください。）

なので、ユーザーモデルに入っているデータの中で、ユーザー名がryooopanの人を探す場合には、カンタンに次のように書けば取ってこれるわけです。

```
User.find_by(:username => 'ryooopan')
```

これは他にも、たとえば名前で探してくる場合は

```
User.find_by(:name => 'Ryo Suzuki')
```

とすればいいわけです。

idで探す場合は、同じように

```
User.find_by(:id => 1)
```

とできますが、idで探す場合がよく多いので、単純に

```
User.find(1)
```

として、idが1のユーザーを取ってこれるようになっています。

さて、これで準備が完了です。

ブラウザで

```
http://0.0.0.0:3000/users/show/ryooopan
```

にアクセスすると

と、このようにデータベースから取ってきたユーザー情報が表示できるようになりました！

素晴らしい進歩ですね。

しかし、少し頭のいい人は、ここでコントローラーで無駄が多いということに気づくはずです。

今、app/controllers/users_controller.rbはこのようになっていますが、

```
def show
  if params[:username] == 'ryooopan'
    @user = User.find_by(:username => 'ryooopan')
  elsif params[:username] == 'moyahima'
    @user = User.find_by(:username => 'moyahima')
  end
end
```

これも100人ユーザーいたら100人の場合分けをしないと行けないことになります。

これは単純に

```
def show
  @user = User.find_by(:username => params[:username])
end
```

にすれば同じ事ができてしまいます。

エレガントですね。

このようにRailsを使うことで、非常にクールなコードに仕上がりました。

Railsを使うメリットが少しずつわかってきたでしょうか？

さて、ここまではかなり順調に進んできましたが、よくよく考えると、Twitterアプリというからには、ユーザーがツイートをして、そのツイートを表示しなければ意味がありません。

こちらが用意した初期データのツイートを表示した所で、なにもおもしろくありません。

そこで、次に、ユーザーが実際にデータを入力し、それをデータベースに保存し、それを表示する、ということをやってみましょう。

パッと見ウルトラC的な難しさに見えますが、今まで使ったものを組み合わせるだけでカンタンにできてしまいます。

§14 ツイートを保存するためのビュー、コントローラー、モデルを作る

14.1 ツイートを保存するためのビュー、コントローラー、モデルを作る

今まではユーザーの情報だけを表示していましたが、ここからは、ツイートの情報も必要になって来ました。

表示するためにはビューとコントローラー、データを保存するにはモデルが必要でした。

さて、復習も兼ねて、やってしまいましょう。

まず、ビューとコントローラーをつくる方法は、

```
rails g controller tweets index show new
```

でした。

今回は新しいツイートを入力する画面のためにnewを追加しています。

次に、データを保存するためにモデルが必要でした。

モデルを作る方法は、

```
rails g model tweet title:string content:text
```

ここでは、ツイートのタイトルと文章を保存することにしましょう。

さて、これでマイグレーションファイルができました。

これを実際にデータベースに反映するためには

```
rake db:migrate
```

とすれば完了です。

ここまで行けば、イメージ的には

```
@tweet = Tweet.new
@tweet.title = 'This is first tweet title'
@tweet.content = 'I am making how to make twitter app.'
@tweet.save
```

とすれば、データが保存できそうです。

この中をユーザーが入力した値にすればいいのだということがわかります。

ここまでわかれば、ユーザーがツイートを打ち込む画面を次に作っていきましょう。

§15 ツイートを入力するためのフォームを作る

15.1 ツイートを入力するためのフォームを作る

さて、それではまずツイートの新規作成画面は app/views/tweets/new.html.erb でした。

それでは

```
http://0.0.0.0:3000/tweets/new
```

にアクセスすると、次のような画面になっていると思います。

今までと違い、今度はユーザーがツイートを入力する画面を作りたいわけでした。

そのためにはどうしたらいいのでしょうか。

それはフォームというものを使います。

これははじめてなので答えを先に言ってしまうと、 app/views/tweets/new.html.erb を次のように編集してみましょう。

```
<%= form_for Tweet.new do |f| %>
  <%= f.label :title %>
  <%= f.text_field :title %>
  <%= f.label :content %>
  <%= f.text_area :content %>
  <%= f.submit %>
<% end %>
```

このような形でRailsではカンタンにフォームを作成できるようになっています。

> 注 このようなものをヘルパーと言います。 この例の場合は、フォームヘルパーと呼ばれるものです。

さて、これで再度アクセスすると、次のようなエラーが出ています。

エラーには

```
undefined method `tweets_path' for #<#<Class:0x007fdd2300a638>:0x007fdd230097d8>
```

と出ています。

これはどういうことかというと、フォームにデータを入れてもらっても、それをどこに送ればいいのか書かれていない、というエラーです。

このようのなURLの設定は、どこを変更すればよかったのでしょうか。

そう、ルーティングです。

config/routes.rb を次のように編集しましょう。

```
MitakalabTwitter::Application.routes.draw do
  get "tweets/index"
  get "tweets/show"
  get "tweets/new"
  post "tweets" => "tweets#create"

  get "users/index"
  get "users/show/:username" => "users#show"
  ...
end
```

これは、ツイートのフォームを入力して、送信ボタンを押したらtweetsコントローラーのcreateにいきなさいと支持していることになります。

さて、ここまで変更して再読み込みすると、

おっ！それっぽいフォーム画面が出て来ました。

それでは早速入力してみましょう。

これで、意気揚々とCreate Tweetボタンを押すと、無常にもエラーが出て来ました。

エラーの内容は

```
The action 'create' could not be found for TweetsController
```

となっています。

それもそのはずで、先ほどルーティングでtweetsコントローラーのcreateに飛ばせと命令したにもかかわらず、コントローラーにcreateを作っていませんでした。

まずは、createアクションを作りましょう。

app/controllers/tweets_controller.rbを次のように編集します。

```
class TweetsController < ApplicationController
  def index
  end

  def show
  end

  def new
  end

  def create
  end
end
```

これで、createアクションを定義出来ました。

フォームで入力された情報はこのtweets#createに飛ばされることになるのです。

そう考えると、このcreateの中で、ユーザーから送られてきた情報をデータベースに保存することが出来れば、いいのです！

それでは、ユーザーから送られた情報はどうやって、受け取ればよかったのでしょうか？

実は、もうすでに出てきています。

そう、それはパラメーターです。

```
http://0.0.0.0:3000/users/show/:username
```

のときに出てきた、

```
params[:username]
```

です。

これと同じ仕組みで、ユーザーが入力した情報をコントローラーで取得することができるのです。

ここで、パラメーターについて補足をすると、パラメーターはハッシュというデータの形で送られて来ることになっています。（ハッシュは辞書のような形をしているので、辞書形式とも言われます。）

どういうことかというと、先ほどのusernameの場合、パラメーターは

```
params = { :username => 'ryoooopan' }
```

という情報が送られてきているわけです。

今のtweetの場合、パラメーターは次のような情報が飛んできます。

```
params = { :tweet => { :title => "This is first tweet title", :content => "I am making how to make twitter app." }
```

少し複雑ですが、これを同じようにタイトルを取得するためには

```
params[:tweet][:title]
```

と、ツイートの内容を取得するためには

```
params[:tweet][:content]
```

とすればいいのだと、わかります。

さて、ここまでがわかれば、詳しいことはわからないにしても、とりあえずデータの保存はできそうです。

それは app/controllers/tweets_controller.rb のcreateを次のように編集すればいいからです。

```
def create
  @tweet = Tweet.new
  @tweet.title = params[:tweet][:title]
  @tweet.content = params[:tweet][:content]
  @tweet.save
end
```

これで、本当にデータをデータベースに入れることができます。

もう一度入力して、送信すると次のような画面になります。

これはcreateのビューを作っていないエラーなので、このようなエラーが出ていますが、きちんとデータは挿入できています。

createのビューは必要ないので、もしもデータを保存し終わったらツイートの一覧にリダイレクトさせるようにしましょう。

それは app/controllers/tweets_controller.rb を再度次のように編集します。

```
def create
  @tweet = Tweet.new
  @tweet.title = params[:tweet][:title]
  @tweet.content = params[:tweet][:content]
  @tweet.save
  redirect_to '/tweets/index'
end
```

ここまでやって、このようにtweets#indexが表示されれば、完了です。

§16 最後の残り：ツイートの一覧表示

16.1 最後の残り：ツイートの一覧表示

さて、ここまでで、ずいぶん多くのことを学びました。

ビュー・コントローラーから始まり、ルーティング、データベースやモデル、そしてフォームなどです。

基本的なWebアプリケーションを作るベースは閑静です。

最後のしめとして、保存したツイートのデータを一覧で表示できるようにして、終わりにしましょう。

一覧表示は、tweets#indexを変更すればよいのでした。

データベースからすべてのデータを取ってくるのは非常にカンタンです。

app/controllers/tweets_controller.rbを次のように編集します。

```
def index
  @tweets = Tweet.all
end
```

この

```
Tweet.all
```

はツイートモデルのデータをすべて取ってこい、ということです。

これをビューで表示させれば良いわけです。

app/views/tweets/index.html.erb にいきましょう。

今回は、ひとつだけのデータでなく、たくさんのデータが入っているのを表示させることになります。

どうしたらよいでしょうか。

これはプログラミングのforループと呼ばれるものを使えば、カンタンにできます。

Rubyのforループは少し変わっていて、たとえば1から10までの足し算をしろ、という時には次のようなプログラムを書きます。

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
sum = 0
numbers.each do |number|
  sum += number
end

puts sum
```

とすると、55が表示されます。

非常にエレガントですね。

これを利用します。

app/views/tweets/index.html.erb を次のように編集してみましょう。

```
<% @tweets.each do |tweet| %>
  <h1><%= tweet.title %></h1>
  <p><%= tweet.content %></p>
<% end %>
```

ここで、

```
<% ... %>
```

というものが出てきました。

これは、Rubyの埋め込んでいるということはわかると思います。

このイコールのないものは、「HTMLとして表示剃る必要のないもの」、という意味です。

さて、ここまで出来れば、もう一度

```
http://0.0.0.0:3000/tweets/index
```

にアクセスしてみましょう。

このように、すべてのツイートが表示できていたら完成です！

§17 この次のステップは？

17.1 この次のステップは？

ここまで学んできて、ある程度Railsの仕組みが理解できたでしょうか？

他にもまだまだやりたいことはたくさんあると思います。

- 画像をアップロードするためには、どうしたらいいの？
- ユーザーのログインをして、ユーザーごとにツイートを表示したい。
- 実際のTwitterと連動して、ここでつぶやいたことをTwitterにも表示させたい。

などなど。

ここから先は、無限です。

Mitakalabコミュニティで学ぶもよし、本で独学するもよし、Google先生にきくもよし。

自分だけのアプリを作るべくがんばってください。

参考までに、この次のステップで有用なものを挙げておきます。

- Railscasts：Railsの応用例がたくさん載っているサイトです。（英語・日本語）
- Cumiki：Railsにかぎらず、Railsの応用例をどうやって付け加えるかを、レゴを組み立てるような手順で学ぶことができます。（日本語）
- ドットインストール：Railsにかぎらず、基本的な点を動画を使ってわかりやすく解説されています。（日本語）
- Ruby Toolbox：RailsにGemを追加して機能を付け加えたいときに、どんなGemがあるのかをカテゴリーごとに見ることができます。（英語）

§18 補足

18.1 補足

これは、Railsが実際にどういうふうに動いているのか、という点に重点を置いているので、説明の流れ上、無駄に回り道をしながら説明したり、実際に使われているコードより汚い箇所があるので、最後に補足として説明しておきます。

まず、ルーティングについてですが、今回は、

```
get "tweets/index"
get "tweets/show"
get "tweets/new"
post "tweets" => "tweets#create"

get "users/index"
get "users/show/:username" => "users#show"
```

となっていますが、一般によく使われるのは、

```
resources :tweets
resources :users
```

というものです。

これはどういうことかというと、

```
resources :tweets
```

と書くと、次のようなルーティングを自動で設定してくれる便利な機能です。

```
get 'tweets' => 'tweets#index'
get 'tweets/:id' => 'tweets#show'
get 'tweets/new' => 'tweets#new'
get 'tweets/:id/edit' => 'tweets#edit'
post 'tweets' => 'tweets#create'
put 'tweets' => 'tweets#update'
delete 'tweets' => 'tweets#destroy'
```

また、このgetやpostについては説明しないで、途中「GETとかPOSTとかってなんだ？」と思った方もいるかもしれませんが、はじめのうちは

アクセスするURLの中に直接パラメーターを含めるものがGET、そうでないのがPOST

というようなイメージで考えておけば十分だと思います。

なので例としては

```
0.0.0.0:3000/users/show/ryooopan
```

のようにパラメーターがURLに含められているので、これはGET、というような感じです。

また、パラメーターについて、ついでに補足しておくと

app/controllers/tweets_controller.rbのなかでわかりやすさをかねてこのように書きましたが、

```
def create
  @tweet = Tweet.new
  @tweet.title = params[:tweet][:title]
  @tweet.content = params[:tweet][:content]
  @tweet.save
  redirect_to '/tweets/index'
end
```

これは、params[:tweet]がハッシュの形になっていて、データはハッシュを直接渡せばそのとおりに保存してくれるので、

```
def create
  @tweet = Tweet.create(params[:tweet])
  redirect_to '/tweets/index'
end
```

としても、同じように動きます。

協賛企業募集