アジャイル開発概要

Ruby On Rails

Ruby On Rails概要

- オールインワンフレームワーク
  ORマッパ、キャッシュ、テスト用DB、テストツール、テスト用サーバが含まれている。
- プログラミング言語にRubyを採用。
- 動的言語の開発スタイル。
  ソースコードを修正して、ブラウザの更新ボタンを押すだけ（コンパイルやデプロイが不要）。
- 設定よりも規約を重視する。
- Rubyを世界的に有名にした火付け役。

Ruby On Rails超入門

Ruby On Rails超入門

Ruby On Rails超入門

- インストール方法は、OSによって異なる。Linuxを用いるのが簡単。
- テスト用DBとしてsqliteが入っているが、機能が少ないので通常は本番用に別のDBを用意する。
- 本番用サーバは入っていないので、本番稼動時には別途用意する（Unicornなど）。

Ruby On Rails超入門

Hello World

Ruby On Rails超入門

- rails new helloを実行(helloの部分がアプリケーション名）。
  雛形が生成される。

```
shanai@ubuntu:~/rails$ rails new hello
      create
      create  README.rdoc
      create  Rakefile
      create  config.ru
      create  .gitignore
      create  Gemfile
      create  app
      create  app/assets/images/rails.png
      create  app/assets/javascripts/application.js
...
```

Ruby On Rails超入門

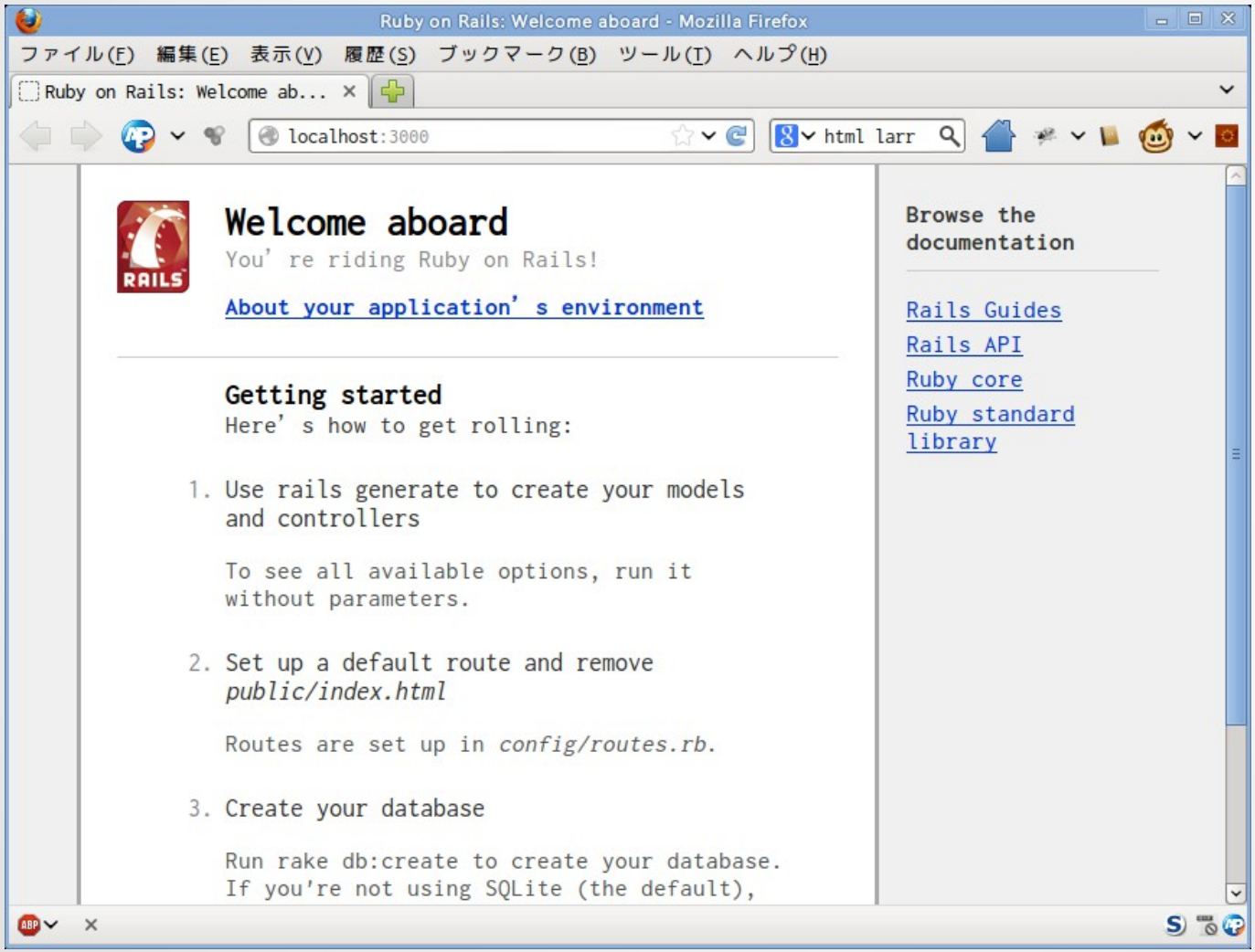
- サーバを起動。
  テスト用にWEBrickが入っている。

```
shanai@ubuntu:~/rails/hello$ rails server
=> Booting WEBrick
=> Rails 3.2.9 application starting in development on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server

[2013-03-24 17:04:51] INFO  WEBrick 1.3.1
[2013-03-24 17:04:51] INFO  ruby 1.9.3 (2012-11-10) [x86_64-linux]
[2013-03-24 17:04:51] INFO  WEBrick::HTTPServer#start: pid=19717 port=3000
```

Ruby On Rails超入門

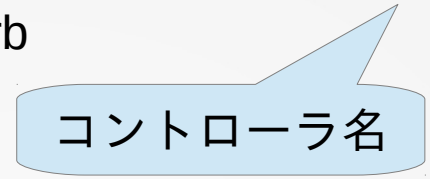
- http://localhost:3000を開く

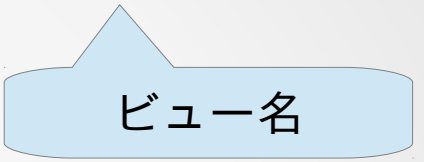


Ruby On Rails超入門

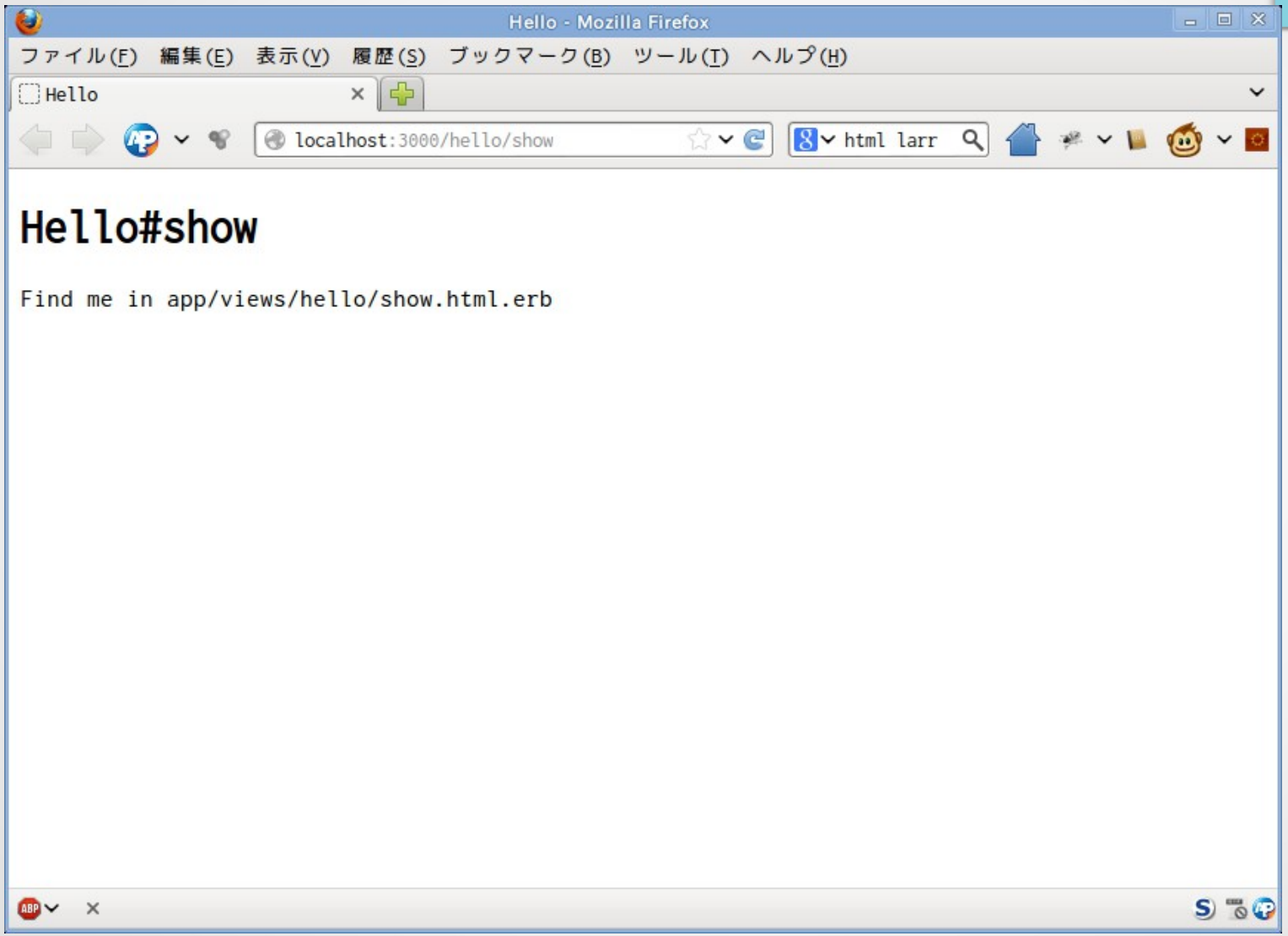
- コントローラとビューを作成する。

```
shanai@ubuntu:~/rails/hello$ rails generate controller hello show
      create  app/controllers/hello_controller.rb
       route  get "hello/show"
      invoke  erb
      create    app/views/hello
      create    app/views/hello/show.html.erb
      invoke  test_unit
      create    test/functional/hello_controller_test.rb
      invoke  helper
      create    app/helpers/hello_helper.rb
      invoke    test_unit
      create      test/unit/helpers/hello_helper_test.rb
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/hello.js.coffee
      invoke    scss
      create      app/assets/stylesheets/hello.css.scss
```





Ruby On Rails超入門



Ruby On Rails超入門

/hello/showがリクエストされた時の処理の流れ

config/routes.rb

```
get "hello/show"
```

規約により、HelloControllerのshowメソッドが呼び出される。

app/controllers/hello_controller.rb

```
class HelloController < ApplicationController
  def show
  end
end
```

規約により、show.html.erbが呼び出される。

app/views/hello/show.html.erb

```
<h1>Hello#show</h1>
<p>Find me in app/views/hello/show.html.erb</p>
```

Ruby On Rails超入門

- 動的コンテンツ

app/controllers/hello_controller.rb

```
class HelloController < ApplicationController
  def show
    @name = "Taro"
  end
end
```

テンプレート変数

app/views/hello/show.html.erb

```
<h1>Hello <%= @name%></h1>
```

Ruby On Rails超入門

- モデルを作成

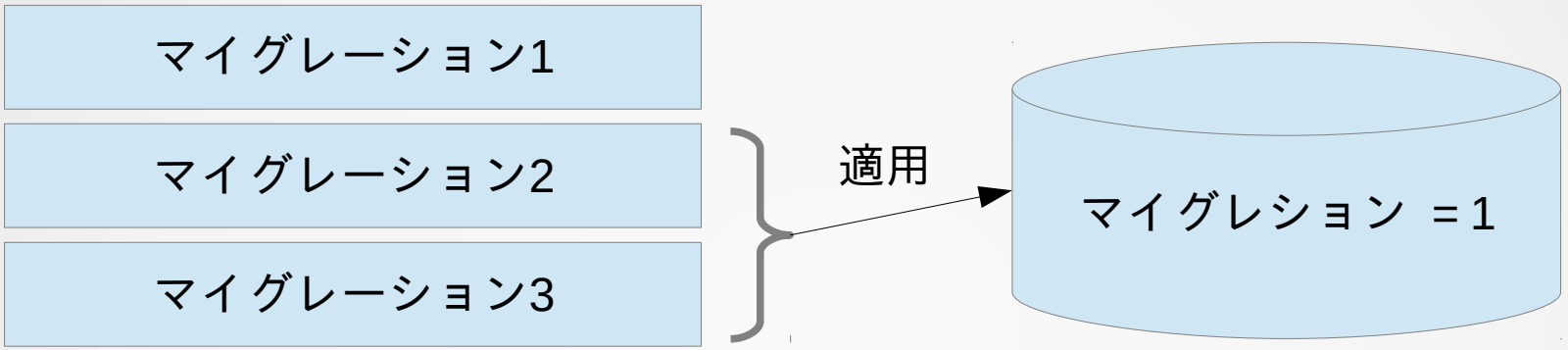
rails generate model person name:string age:integer

```
invoke  active_record
create    db/migrate/20130324101958_create_people.rb
create    app/models/person.rb
invoke    test_unit
create      test/unit/person_test.rb
create      test/fixtures/people.yml
```

Ruby On Rails超入門

- マイグレーション

db/migrate/




```
shanai@shanai-ThinkPad-X61:~/rails/hello$ rake db:migrate
==  CreatePeople: migrating
=============================================
-- create_table(:people)
   -> 0.0016s
==  CreatePeople: migrated (0.0017s)
=============================================
```

```
shanai@shanai-ThinkPad-X61:~/rails/hello$ rake db:migrate
shanai@shanai-ThinkPad-X61:~/rails/hello$
```

最新バージョンが記録されるので2回目は何も実行されない。

Ruby On Rails超入門

- フィクスチャ（テストデータ）

test/fixtures/people.yml

```
one:
  name: MyString
  age: 1

two:
  name: MyString
  age: 1
```

規約によりPeopleテーブルのデータとして用いられる。

yml(ヤムル）という形式。インデントによりデータ構造を表現する。

- 以下を実行するとロードされる

```
rake db:fixtures:load
```

Ruby On Rails超入門

config/routes.rb

```
get "hello/list"
```

app/controllers/hello_controller.rb

```
def list
  @person_list = Person.all
end
```

views/hello/list.html.erb

```
<table border="1">
  <tr>
    <th>名前</th>
    <th>年齢</th>
  </tr>

  <% @person_list.each do |person| %>
    <tr>
      <td><%= person.name%></td>
      <td><%= person.age%></td>
    </tr>
  <% end %>
</table>
```

一覧表示の処理

Ruby On Rails超入門

Ruby On Rails超入門

- Active Record
  - Rails標準のORマッパ
  - 表をクラスに、行をそのクラスのオブジェクトに見立てる
  - 基本的にモデル内にビジネスロジックを構築する（リッチなドメインモデル）

Ruby On Rails超入門

- 基本操作
  - findメソッド(主キーによる検索)
    @book = Book.find(2)
    @books = Book.find([2, 3, 5])

Ruby On Rails超入門

- 基本操作
  - find_by_xxxメソッド(主キー以外による検索。1件のみ)
    @book = Book.find_by_publisher('xxx')
    @book =
      Book.find_by_publisher_and_price('xxx', 100)
  - find_all_by_xxxメソッド(主キー以外による検索。全件)

メソッド名を見て動的に照会を実行している。

Ruby On Rails超入門

- 照会

```
@books = Book.where(:publisher => 'xxx').order('publisher desc')
```

遅延ロードされる

- パラメータ埋込み

```
@books = Book.where(’publisher = :publisher and price < :price',
   {:publisher => 'xxx', :price => xxx}
```

Ruby On Rails超入門

- スコープ

```
class Book < ActiveRecord::Base
  scope :cheap where('price < 1000')
end

@books = Book.cheap
```

抽出条件に名前を付けて再利用

- スコープ(引数付き)

```
class Book < ActiveRecord::Base
  scope :cheaper, lambda {
    |price| where('price < :price', price)
  }
end

@books = Book.cheaper(1000)
```

Ruby On Rails超入門

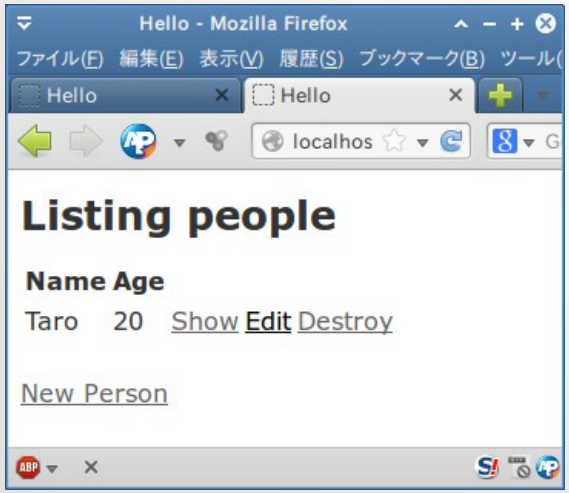
- スキャッフォルド
  CRUDアプリケーションの自動生成

```
rails generate scaffold people name:string age:integer
...
      invoke    test_unit
      create      test/unit/helpers/people_helper_test.rb
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/people.js.coffee
      invoke    scss
      create      app/assets/stylesheets/people.css.scss
      invoke  scss
      create    app/assets/stylesheets/scaffolds.css.scss

shanai@shanai-ThinkPad-X61:~/rails/hello$ rake db:migrate
==  CreatePeople: migrating
=============================================
-- create_table(:people)
   -> 0.0016s
==  CreatePeople: migrated (0.0017s)
```

Ruby On Rails超入門

http://localhost:3000/people



Ruby On Rails超入門

- 生成されたroutes.rb

config/routes.rb

```
resources :people
```

この1行のみ

rake routesで確認できる

```
shanai@shanai-ThinkPad-X61:~/rails/hello$ rake routes
     people GET    /people(.:format)          people#index
            POST   /people(.:format)          people#create
 new_person GET    /people/new(.:format)      people#new
edit_person GET    /people/:id/edit(.:format) people#edit
     person GET    /people/:id(.:format)      people#show
            PUT    /people/:id(.:format)      people#update
            DELETE /people/:id(.:format)      people#destroy
```

実際は、これだけのroutesが生成されている。

Ruby On Rails超入門

- リスト

http://localhost:3000/people

format省略時は、html

```
shanai@shanai-ThinkPad-X61:~/rails/hello$ rake routes
     people GET    /people(.:format)          people#index
            POST   /people(.:format)          people#create
 new_person GET    /people/new(.:format)      people#new
edit_person GET    /people/:id/edit(.:format) people#edit
     person GET    /people/:id(.:format)      people#show
            PUT    /people/:id(.:format)      people#update
            DELETE /people/:id(.:format)      people#destroy
```

app/controllers/people_controller.rb

拡張子かContent-Typeが使用される

```
def index
  @people = Person.all

  respond_to do |format|
    format.html # index.html.erb
    format.json { render json: @people }
  end
end
```

Content-Typeごとに処理方法を記載する。

Ruby On Rails超入門

app/views/people/index.html.erb

```
<table>
  <tr>
    <th>Name</th>
    <th>Age</th>
    <th></th>
    <th></th>
    <th></th>
  </tr>

<% @people.each do |person| %>
  <tr>
    <td><%= person.name %></td>
    <td><%= person.age %></td>
    <td><%= link_to 'Show', person %></td>
    <td><%= link_to 'Edit', edit_person_path(person) %></td>
    <td><%= link_to 'Destroy', person, method: :delete, data: { confirm: 'Are you sure?' }
%></td>
  </tr>
<% end %>
</table>
```

Ruby On Rails超入門

- 新規作成

app/views/people/index.html.erb

```
<%= link_to 'New Person', new_person_path %>
```

resourcesで生成されるヘルパ

- 以下のヘルパが自動生成される。

ヘルパ	生成されるurl
person_path	/person
person_path(id)	/person/:id
new_person_path	/person/new
edit_person_path(id)	/person/:id/edit

Ruby On Rails超入門

config/routes.rb

```
shanai@shanai-ThinkPad-X61:~/rails/hello$ rake routes
      people GET    /people(.:format)          people#index
             POST   /people(.:format)          people#create
  new_person GET    /people/new(.:format)      people#new
edit_person GET    /people/:id/edit(.:format) people#edit
      person GET    /people/:id(.:format)      people#show
             PUT    /people/:id(.:format)      people#update
             DELETE /people/:id(.:format)      people#destroy
```

app/controllers/people_controller.rb

```
def new
  @person = Person.new

  respond_to do |format|
    format.html # new.html.erb
    format.json { render json: @person }
  end
end
```

Ruby On Rails超入門

app/views/people/new.html.erb

```
<h1>New person</h1>

<%= render 'form' %>

<%= link_to 'Back', people_path %>
```

app/views/people/_form.html.erb

```
<%= form_for(@person) do |f| %>
  <% if @person.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@person.errors.count, "error") %> prohibited this person from being
saved:</h2>

      <ul>
      <% @person.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
      </ul>
    </div>
  <% end %>
```

Ruby On Rails超入門

```
  <div class="field">
    <%= f.label :name %><br />
    <%= f.text_field :name %>
  </div>
  <div class="field">
    <%= f.label :age %><br />
    <%= f.number_field :age %>
  </div>
  <div class="actions">
    <%= f.submit %>
  </div>
<% end %>
```

/people/newで表示されたので、/peopleへのPOST。

Ruby On Rails超入門

config/routes.rb

```
shanai@shanai-ThinkPad-X61:~/rails/hello$ rake routes
     people GET    /people(.:format)          people#index
            POST   /people(.:format)          people#create
 new_person GET    /people/new(.:format)      people#new
edit_person GET    /people/:id/edit(.:format) people#edit
     person GET    /people/:id(.:format)      people#show
            PUT    /people/:id(.:format)      people#update
            DELETE /people/:id(.:format)      people#destroy
```

Ruby On Rails超入門

app/controllers/people_controller.rb

```
def create
  @person = Person.new(params[:person])

  respond_to do |format|
    if @person.save
      format.html { redirect_to @person, notice: 'Person was successfully created.' }
      format.json { render json: @person, status: :created, location: @person }
    else
      format.html { render action: "new" }
      format.json { render json: @person.errors, status: :unprocessable_entity }
    end
  end
end
```

リクエストのパラメータから、Personを生成。

/person/:idへリダイレクト

フラッシュ記憶

Ruby On Rails超入門

config/routes.rb

```
shanai@shanai-ThinkPad-X61:~/rails/hello$ rake routes
      people GET    /people(.:format)          people#index
             POST   /people(.:format)          people#create
  new_person GET    /people/new(.:format)      people#new
edit_person GET    /people/:id/edit(.:format) people#edit
      person GET    /people/:id(.:format)      people#show
             PUT    /people/:id(.:format)      people#update
             DELETE /people/:id(.:format)      people#destroy
```

app/controllers/people_controller.rb

```
  def show
    @person = Person.find(params[:id])

    respond_to do |format|
      format.html # show.html.erb
      format.json { render json: @person }
    end
  end
```

Ruby On Rails超入門

app/views/people/show.html.erb

```
<p id="notice"><%= notice %></p>

<p>
  <b>Name:</b>
  <%= @person.name %>
</p>

<p>
  <b>Age:</b>
  <%= @person.age %>
</p>


<%= link_to 'Edit', edit_person_path(@person) %> |
<%= link_to 'Back', people_path %>
```

フラッシュ記憶の表示

Ruby On Rails超入門

- フラッシュ記憶域
  - リクエスト内で共有するデータは、コントローラで保持する（@付き変数）
  - しかしリダイレクトをまたいでデータを渡したい場合は、別リクエストになってしまうため、この方法は使用できない。
  - フラッシュ記憶域は、次のリクエストまでデータが保持される。

Ruby On Rails超入門

- 更新

app/views/people/index.html.erb

resourcesで生成されるヘルパ

```
<td><%= link_to 'Edit', edit_person_path(person) %></td>
```

ヘルパ	生成されるurl
person_path	/person
person_path(id)	/person/:id
new_person_path	/person/new
edit_person_path(id)	/person/:id/edit

Ruby On Rails超入門

- 更新

app/controllers/people_controller.rb

```
def edit
  @person = Person.find(params[:id])
end
```

app/views/people/edit.html.erb

```
<h1>Editing person</h1>

<%= render 'form' %>

<%= link_to 'Show', @person %> |
<%= link_to 'Back', people_path %>
```

Ruby On Rails超入門

- 更新

app/views/people/_form.html.erb

```
<%= form_for(@person) do |f| %>
  <% if @person.errors.any? %>
...
  <div class="actions">
    <%= f.submit %>
  </div>
<% end %>
```

/person/:id/editで表示されたため、/person/:idへのPOSTになる。

Ruby On Rails超入門

config/routes.rb

```
shanai@shanai-ThinkPad-X61:~/rails/hello$ rake routes
      people GET    /people(.:format)          people#index
             POST   /people(.:format)          people#create
  new_person GET    /people/new(.:format)      people#new
edit_person GET    /people/:id/edit(.:format) people#edit
      person GET    /people/:id(.:format)      people#show
             PUT    /people/:id(.:format)      people#update
             DELETE /people/:id(.:format)      people#destroy
```

Ruby On Rails超入門

app/controllers/people_controller.rb

新規の時と同様にリダイレクト。

```
def update
  @person = Person.find(params[:id])

  respond_to do |format|
    if @person.update_attributes(params[:person])
      format.html { redirect_to @person, notice: 'Person was successfully updated.' }
      format.json { head :no_content }
    else
      format.html { render action: "edit" }
      format.json { render json: @person.errors, status: :unprocessable_entity }
    end
  end
end
```

Ruby On Rails超入門

テスト

自動テスト

- Railsにはデフォルトで、Test::Unitが付属する。
- 自動テストが完備されていることがRailsの大きな特長。
- 他のツールも利用可能（RSpecなど)。
- 4種類のテストが可能
  - 単体テスト（クラス単位のテスト）
  - 機能テスト（画面単位のテスト）
  - 統合テスト（複数画面にまたがったテスト）
  - パフォーマンステスト（Rails 3.1から）
- Railsでの単体テストはDBアクセスを含む。

自動テスト

単体テスト

単体テスト

- test/unitの下に配置する。
- 基本的にクラスのメソッドを対象とする。

```
class Person < ActiveRecord::Base
  attr_accessible :age, :name

  def self.average_age(person_table)
    if person_table.empty?
      0
    else
      person_table.inject(0) do |sum, person|
        sum + person.age
      end / person_table.length
    end
  end
end
```

例：年齢の平均を求めるメソッド。

単体テスト

テストの例

```
require 'test_helper'

class PersonTest < ActiveSupport::TestCase
  test "average age test" do
    person1 = Person.new(:age => 100)
    person2 = Person.new(:age => 50)
    assert_equal 75, Person.average_age([person1, person2])
  end

  test "average age for empty array" do
    assert_equal 0, Person.average_age([])
  end
end
```

単体テスト

テストの実行

```
shanai@ubuntu:~/rails/hello$ rake test:units
Run options:

# Running tests:

..

Finished tests in 0.068915s, 29.0212 tests/s, 29.0212 assertions/s.

2 tests, 2 assertions, 0 failures, 0 errors, 0 skips
```

自動テスト

機能テスト

機能テスト

- test/functionalの下に配置する。
- 基本的に1リクエスト処理を対象とする。

テスト対象

```
class HelloController < ApplicationController
  def show
    @name = "Taro"
  end
end
```

テスト

```
  require 'test_helper'

  class HelloControllerTest < ActionController::TestCase
    test "should get show" do
      get :show
      assert_equal "Taro", assigns(:name)
      assert_response :success
    end
  end
```

getリクエストを生成

テンプレート変数@nameの内容を確認

200(OK)が返ることを確認

機能テスト

スキャッフォルドで生成したコード

```
def create
  @person = Person.new(params[:person])

  respond_to do |format|
    if @person.save
      format.html { redirect_to @person, notice: 'Person was successfully created.' }
      format.json { render json: @person, status: :created, location: @person }
    else
      format.html { render action: "new" }
      format.json { render json: @person.errors, status: :unprocessable_entity }
    end
  end
end
```

assert_differenceは、ブロックを実行して、引数の評価結果が変化することを確認する。

POSTリクエストを生成。

```
test "should create person" do
  assert_difference('Person.count') do
    post :create, person: { age: @person.age, name: @person.name }
  end

  assert_redirected_to person_path(assigns(:person))
end
```

リダイレクトされることを確認。

自動テスト

統合テスト

統合テスト

- test/integrationの下に配置する。
- 複数画面にわたるテストを実施する。
- rails generate integration_test テスト名で雛形が生成できる。

```
shanai@ubuntu:~/rails/hello$ rails generate integration_test people

      invoke  test_unit
      create    test/integration/people_test.rb
```

統合テスト

- テストデータの投入

test/fixtures/people.yml

```
one:
  id: 1
  name: Taro
  age: 20
```

test/integration/people_test.rb

```
require 'test_helper'

class PeopleTest < ActionDispatch::IntegrationTest
  fixtures :all
...
```

テストデータのロード指示

統合テスト

- 例：削除が正しく動作するかをテストする。

```
require 'test_helper'

class PeopleTest < ActionDispatch::IntegrationTest
  fixtures :all

  test "Delete model" do
    # List
    get '/people'
    assert_response :success
    assert_select "table" do
      assert_select "tr" do
        assert_select "th", {:text => "Name"}
        assert_select "th", {:text => "Age"}
      end
      assert_select "tr" do
        assert_select "td", {:text => "Taro"}
        assert_select "td", {:text => "20"}
      end
    end
```

ページ内のtableに、<th>Name</th>と<th>Age</th>があること、<td>Taro</td>、<td>20</td>があることを検証する。

統合テスト

```
    # Delete
    delete '/people/1'
    assert_redirected_to :controller => 'people', :action => 'index'
    follow_redirect!

    # List
    assert_response :success
    assert_select "table" do
      assert_select "tr" do
        assert_select "th", {:text => "Name"}
        assert_select "th", {:text => "Age"}
      end
      assert_select "tr" do
        assert_select "td", false
      end
    end
  end
end
```

deleteリクエストを投げて、リダイレクト先を確認し、リダイレクト先に飛ぶ。

データが0件になったので、<td>タグが無いことを検証。

統合テスト

- 統合テストはSeleniumなどのテストツールでも可能なので、予め守備範囲を決めておく。
- テストが重複しないように。
  テストが重複すると、コードを修正した時に修正しなければならないテストの量が増える。

まとめ

- オールインワンのフレームワーク
- Ruby対応
- データアクセスにはActiveRecordを使用する
- 設定より規約を重視する
- コマンドによるコード生成
- 自動テストツールを完備