

Ruby, Ruby on Rails勉強会

～環境構築から入門まで～

2011年5月14日
SEO片岡 乃村研究室

本日の流れ

- (1) 自己紹介
- (2) Ruby, Ruby on Railsの紹介
- (3) 作成物の紹介

休憩

- (4) Ruby, Railsのインストール
- (5) はじめてみようRuby on Rails
- (6) 作ってみよう商品管理システム

休憩

- (7) Rails tips べきべからず集
- (8) Railsに関するLightningトーク

Ruby

Rubyとは

- (1) オブジェクト指向のスクリプト言語
- (2) 1993年に開発開始
- (3) 既存の様々な言語を参考に設計

Lisp, Smalltalk, C, Perl etc

特徴

- (1) 純粹なオブジェクト指向
文字列や整数含め, 全てのデータがオブジェクト
- (2) 動的型付け
実行時に初めて変数の型が決まる
- (3) 演算子オーバーロード
演算子もメソッドとして定義されているため, 再定義が可能
- (4) ガベージコレクタ
ユーザによるメモリ管理が不要
- (5) インタプリタ
コンパイル無しで実行可能

RubyとRailsの歴史

- 1993年 開発開始
- 1995年 Ruby0.5公開
- 1996年 [ドキュメントの英訳](#)
- 1998年 英語のML開設
- 2000年 「Programming Ruby: A Pragmatic Programmer's Guide」
(Dave Thomas, Andy Hunt)
- 2004年 Ruby on Rails0.5 リリース
- 2006年 [Martin Fowler「私はRubyが大好きだ」](#)
- 2007年 Rubyアソシエーション発足
- 2010年 Rails3リリース(8月), Tiobe Index 10位(11月)
- 2011年 JIS規格に制定(3月)

Ruby on Rails

Ruby製のWebアプリケーションフレームワーク

(1) Railsの基本理念

(A) **Don't Repeat Yourself** (同じことを繰り返さない)

重複を排除する

(B) **Convention over Configuration** (設定より規約)

規約に従う事で面倒な設定を減らす

(2) Scaffoldによるコーディング量削減

(3) RESTfulなアプリケーション開発

(4) MVC(Model/View/Controller)アーキテクチャを採用

[利用事例1] 楽天

楽天市場(オンラインショッピング)における

大規模トランザクション処理, セキュリティ処理

[利用事例2] クックパッド

料理のレシピを投稿・検索できるWebサイト

Railsで開発, 運用

作例:LastNote

LastNoteとは

乃村研究室のGNグループが研究開発しているグループウェア

LastNote - index

LastNoteは、GNグループで開発しているWebベースのグループウェアです。

お知らせ

LastNote ver.3からLastNoteの各機能を利用するにはログインが必要です。
デフォルトでは各ユーザーのパスワードが設定されていません。
初回ログイン時は、パスワード欄に何も入力せずログインしてください。
ログイン後、個人ページのプロフィールからパスワードの設定を行ってください。

LastNote ver3.4をリリースしました。
詳細はリリースノートを参照してください。

リリースノート

LastNote Project Page

LastNote ver3

操作一覧

- 文書一覧
- 新規アップロード
- 未登録文書一覧
- フィルタの使い方

公開フィルター一覧

編集»

- New
- New議事録
- New(自分)
- 記録書(自分)
- AnT(自分)
- GN(自分)
- GN & LastNote
- 今日の資料
- 今日の記録書

個人フィルター一覧

編集»

- 計算機幹事資料
- GN議事録

"意見、要望は" [こちら](#)

検索: **検索**

現在有効なフィルター

追加:

<input checked="" type="checkbox"/> プロジェクト	いずれかである	<input type="button"/>
<ul style="list-style-type: none">SWLAB谷口研山内研乃村研AnTGNNewPOSSECTender		

適用 **保存»**

チェックされている文書を一括表示

一覧 (全 497件)

通番	資料番号(No.)	資料種別	形式	著者	タイトル	提出日付	詳細
1	gn 78-01	議事録		三原 優介	第77回GN検討打合せ議事録	2011年04月21日	
2	gn 78-02	打合資料		吉井 英人	TwitterBotプログラム仕様書	"	
3	gn 78-03	打合資料		木村 有祐	GNグループB4研修課題 Twitte	"	
4	gn 78-04	打合資料		三原 優介	スケジュール管理システムの全	"	
5	gn 78-05	打合資料		福田 大志	デスクトップブックマークの持	"	
6	gn 77-02	打合資料		長尾 武憲	引き継ぎ資料(修正版)	2011年03月16日	

作例:LastNote

【設計方針】

- (1) グループワークの基板となる各種モジュールを用意
 - (A) 文書管理
 - (B) 文献管理
 - (C) 名簿管理
 - (D) 部屋予約管理
 - (E) メーリングリスト管理
 - (F) インベントリ管理
- (2) 各サービスが疎に連携
 - (A) ユーザが環境を変えなくても付加価値
 - (B) 各種Webサービスと自由に連携可能

SEO: Ruby 関連事業

Rubyの普及啓発

- Ruby普及啓発冊子
「わたしたちのRuby」
- 中高生Ruby入門講座
- Ruby講師養成セミナー
- Ruby普及啓発セミナー



Ruby以外にもいくつかのイベントを開催

実際にRubyによりシステムを開発

SEO: 作成予定のシステム

- イベント告知
- 参加者の集計
- イベントの結果報告

ATNDのようなシステム

- 懇親会の情報
- 情報の公開・限定公開等の設定
- 複数人数の一括登録
- イベントの検索機能

企業・個人共に使いやすいシステム

Ruby on Railsのインストール (windows)

RailsInstaller

RailsInstallerとは

- (1) Railsの環境をワンクリックで構築してくれるインストーラ
- (2) インストールするもの
 - (A) Rails 3系 … Railsの最新系
 - (B) Ruby 1.8系 … Rubyの前世代系
 - (C) SQLite … Railsのデフォルトデータベース
 - (D) Git … Railsのプラグインを入れるのに使うバージョン管理システム
- (3) インストールに450MB以上の空き容量が必要

Railsの環境構築にかかる手間を大きく削減

RailsInstallerのダウンロード

<http://railsinstaller.org/>にアクセス

Ready to deploy your app? [Start a free trial at Engine Yard.](#)

RAILS INSTALLER GET UP & RUNNING WITH RAILS

STEP 1 ↓ クリックして
ダウントロード

DOWNLOAD the KIT

For Windows only. Mac and Linux versions coming soon.

Rails Installer has everything you need to hit the ground running. In one easy-to-use installer, you get all the common packages needed for a full Rails stack. Download it now and be writing (and running) Rails code in no time. Packages included:

STEP 2 □

WATCH the VIDEO



STEP 3 →

WHAT'S NEXT?

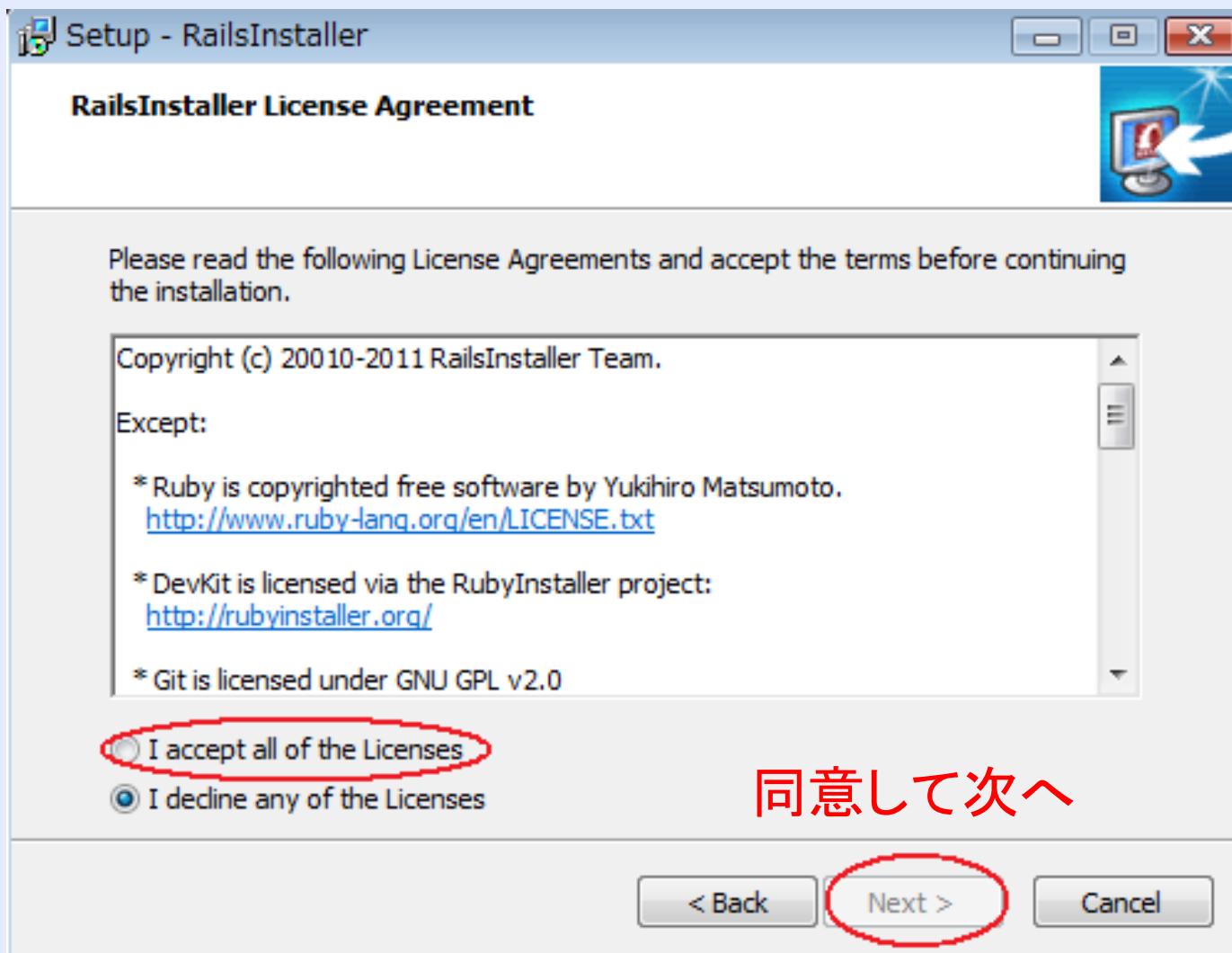
You've got everything installed. You've written (and ran) some code. If you find yourself asking *where do I go from here*, don't worry. You're not alone.

The answer is ***just about anywhere***, but here's a handful of good places to start. The official [Ruby on Rails site](#), [blog](#), [Twitter account \(@rails\)](#), and

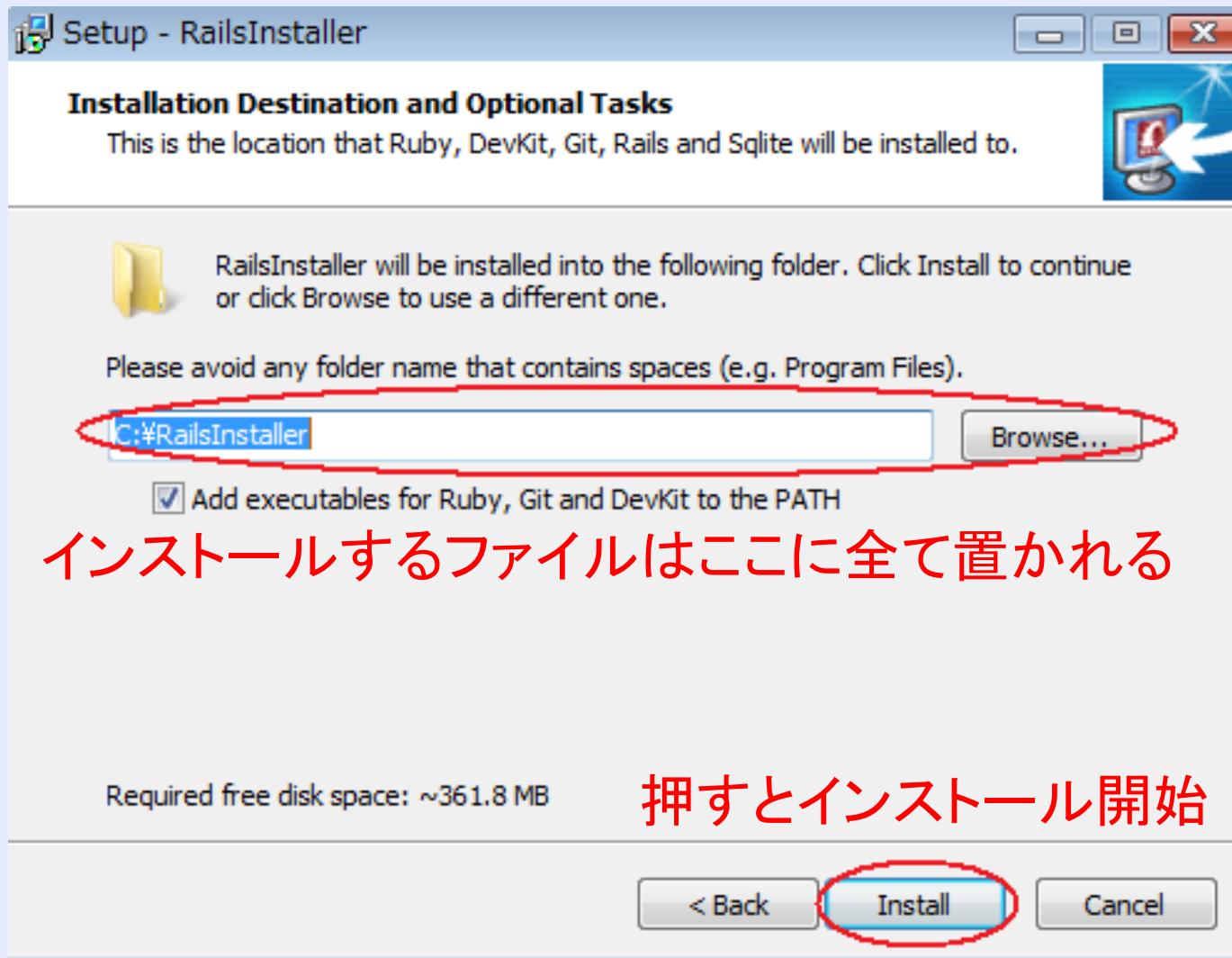
RailsInstallerのインストール方法



RailsInstallerのインストール方法



RailsInstallerのインストール方法



Ruby on Railsのインストール (Mac OS)

Mac OSへのインストール

(1) Rubygemsのバージョンの確認

```
> gem -v
```

(2) バージョン1.6.2へアップデート

```
> sudo gem install rubygems-update -v 1.6.2  
> sudo update_rubygems
```

(3) Railsのインストール

```
> sudo gem install rails sqlite3 --no-ri --no-rdoc
```

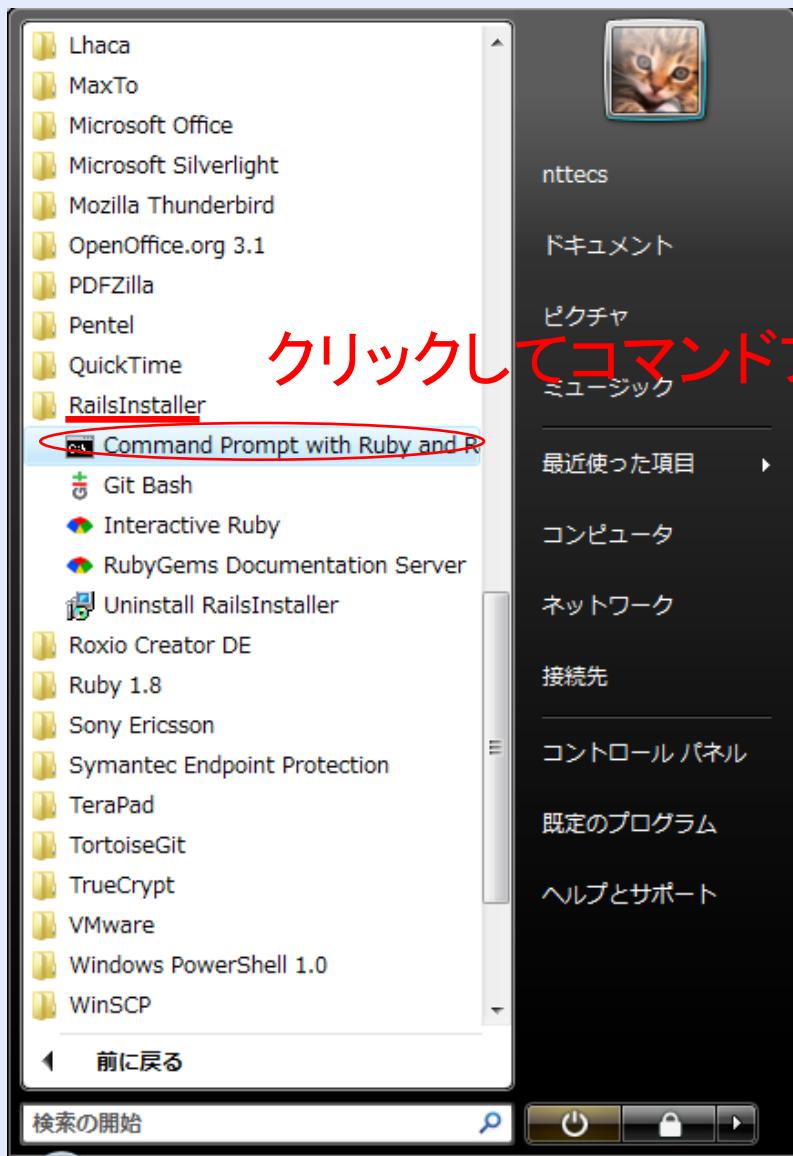
(4) Railsの確認

```
> rails -v
```

Rails 3.0.7 と表示されればOK

はじめてみようRuby on Rails

手順1:コマンドプロンプト起動(windows)



手順1:コマンドプロンプト起動(windows)

最初に起動するとき, gitに関する設定の入力を求められる

```
name >  
email >
```

- (1)今回はgitを直接利用しないので適当に設定して問題ない
- (2)後で変更可能

コマンドプロンプトを起動すると, C:\\$Sites(windows)にいます

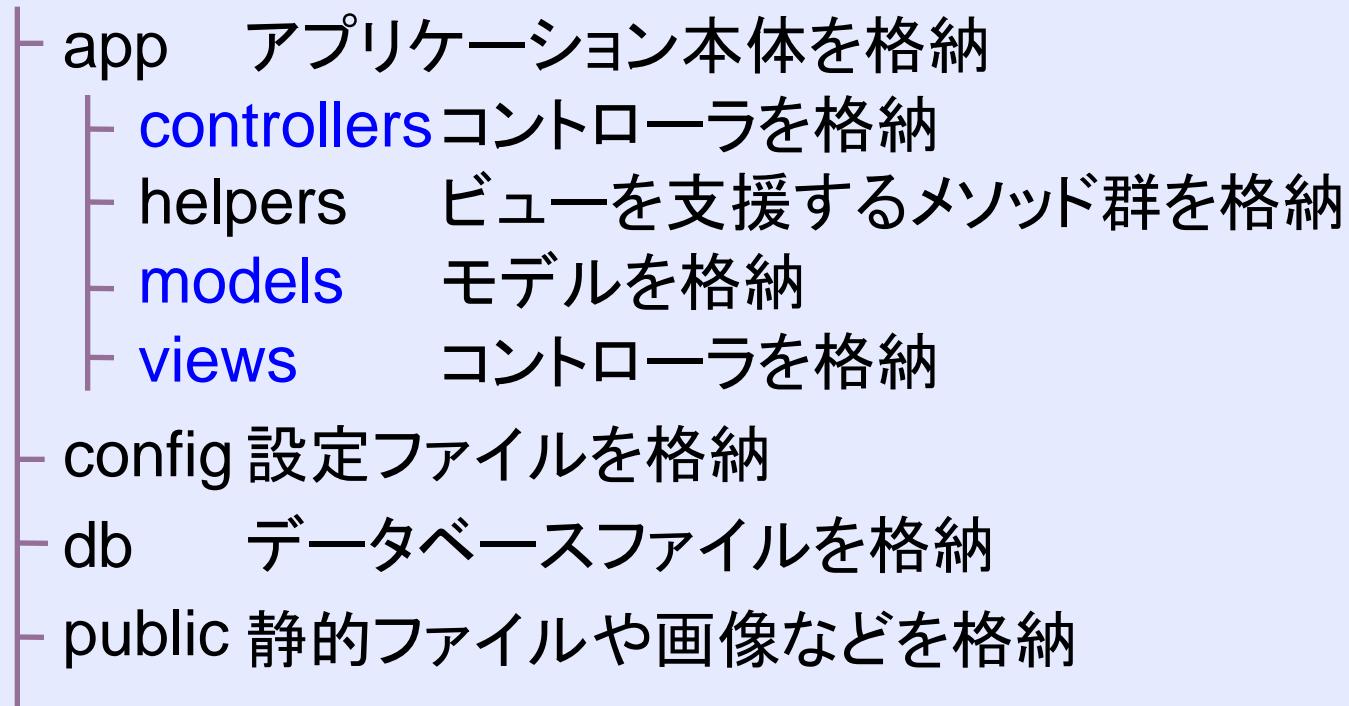
手順2: アプリケーションの土台作成

(1) アプリケーションを作成

```
>rails new depot
```

depot以下にアプリケーションのファイルが生成

生成される主要なディレクトリ



手順3: アプリケーションの動作確認

(1) アプリケーションを起動

```
> cd depot  
> rails server
```

(2) 起動を確認

webブラウザで<http://localhost:3000>にアクセス

The screenshot shows a web browser displaying the 'Welcome aboard' page of a Ruby on Rails application. The page features a red 'RAILS' logo with a white train icon. The main content area includes the title 'Welcome aboard', a sub-header 'You're riding Ruby on Rails!', and a link 'About your application's environment'. Below this is a section titled 'Getting started' with the sub-header 'Here's how to get rolling:'. It lists three steps: 1. 'Use rails generate to create your models and controllers', with a note about running it without parameters; 2. 'Set up a default route and remove or rename this file', with a note about routes being set up in config/routes.rb; and 3. 'Create your database', with a note about running rake db:migrate. To the right of the main content is a sidebar titled 'Browse the documentation' containing links to 'Rails API', 'Ruby standard library', 'Ruby core', and 'Rails Guides'.

手順4: 静的なページ作成

(1) 表示させているページについて確認

depot/publicを見る

(2) Hello Worldページ作成

depot/public/hello.htmlを作成

以下を記述する

```
<html>
  <head>
    <title>Hello world !</title>
  </head>
  <body>
    Hello world !
  </body>
</html>
```

(3) <http://localhost:3000/hello>にアクセス

手順5:HelloWorldページを作成

- (1) depot/app/controllersにhello_controller.rbを作成
以下を記述する

```
class HelloController < ApplicationController
  def world
  end
end
```

- (2) depot/app/views以下で次のことをする
- (A) depot/app/views/helloディレクトリを作成
 - (B) depot/app/views/helloにworld.html.erbを作成
 - (C) world.html.erbの中身は先に作成したhello.htmlと同じ
- (3) config/routes.rbの以下の部分のコメントを外す

```
#match ':controller(/:action(/:id(.:format)))'
```

- (4) <http://localhost:3000/hello/world>にアクセス

手順6:ビューへのプログラムの埋め込み

(1) world.html.erbにプログラムを埋め込む

world.html.erbに赤の部分を追記

```
<html>
  <head>
    <title>Hello world !</title>
  </head>
  <body>
    Hello world ! <%= Time.now %>
  </body>
</html>
```

(2) <http://localhost:3000/hello/world>にアクセス

赤の部分<% %>はRubyのプログラムとして実行される

手順7:コントローラに変数を渡す

(6) controllerから値を渡す

(A) hello_controller.rbに赤の部分を追記

```
class HelloController < ApplicationController
  def world
    @time = Time.now
  end
end
```

(B) world.html.erbの赤の部分を変更

```
<body>
  Hello world ! <%= @time %>
</body>
```

(7) <http://localhost:3000/hello/world>にアクセス

controllerで定義した変数はviewで参照可能

コントローラーとビュー

Q. ビューでRubyプログラムを実行可能ならば、コントローラは必要ではないか？

A. ビューだけでもアプリケーションを作ることは可能しかし、コントローラとビューを使い分けることでコードを整理

コントローラの役割

- (A) リクエストに対してどう内部データを変更するか定義
- (B) リクエストに対して何をレスポンスとして返すか定義

ビューの役割

- (A) コントローラから与えられたデータを元に、どんな形でレスポンスを返すかを定義

モデル

コントローラは処理間で共通の内部データを扱う
これらの内部データはデータベースで管理

モデルの役割

- (1) モデルではデータベースの整合性を保証
 - (A) データのバリデートを記述
 - (B) データを操作する処理を記述
- (2) データベース構造の変化を吸収
 - (A) モデル間の関係を記述

MVC(Model View Controller)の扱い

モデル, ビュー, 及びコントローラをどう結びつけるのか

通常：それぞれの関係性を設定する必要がある

Rails：命名規約で自動的に関連があるとみなす

例) app/models/member.rb

メンバの情報を管理するモデル

app/controllers/members_controller.rb

メンバの情報を操作するコントローラ

app/views/members/list.html.erb

メンバの一覧表示画面のビュー

app/views/members/new.html.erb

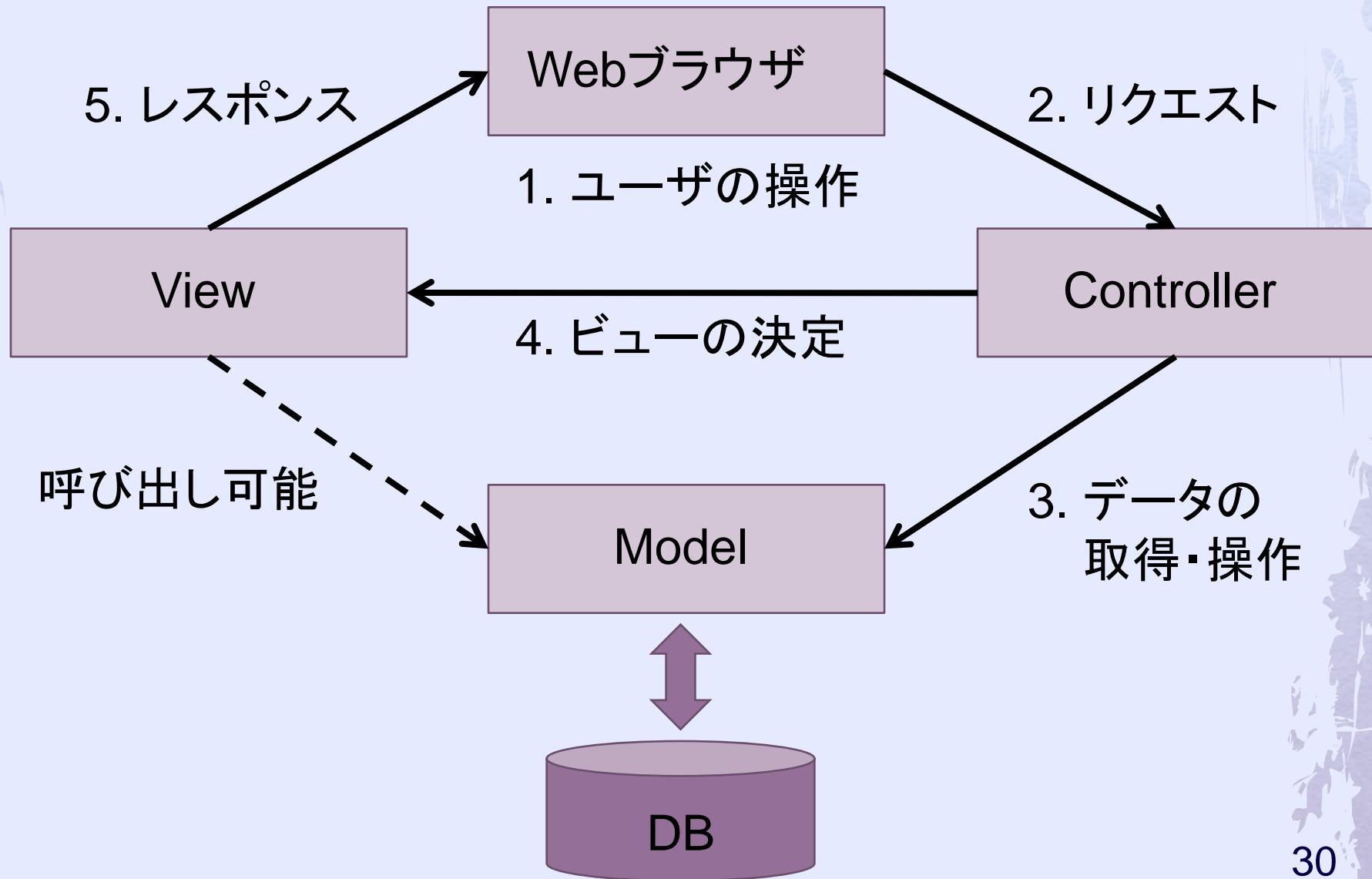
メンバの新規作成画面のビュー

Railsのプログラミング哲学

Convention over Configuration(CoC) 設定より規約

同じ名前のMVC一式を自動生成するジェネレータが存在

Railsの構成



作ってみよう商品管理システム

商品管理システム概要

機能

商品管理機能

- (1) 商品の一覧表示
- (2) 商品の詳細表示
- (3) 商品の新規登録
- (4) 商品の登録内容変更
- (5) 商品の削除
- (6) 商品コードでの参照

Listing products

Title	Description	Image url
RailsによるアジャイルWebアプリケーション 第3版 Railsを始めるならこれ！ただし、第3版ではRails3系まで未対応。		Show Edit Destroy
Rubyレシピブック 第2版 268の技	良く使います。	Show Edit Destroy
Railsレシピブック 183の技	Rails3には未対応。	Show Edit Destroy

[New Product](#)

商品管理システムで作るもの

商品 = Product

app/controllers

 products_controller.rb

Productを操作するコントローラ

app/models

 product.rb

Productのデータを扱うモデル

app/views

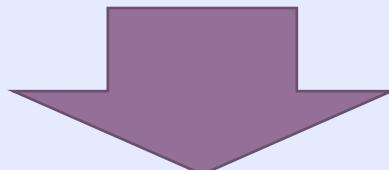
 products/index.html.erb

各種操作に対応するView

 products/show.html.erb

 products/new.html.erb

 products/edit.html.erb



一気に作ってくれるscaffold

Scaffoldを使ってみる

(1) productに関するファイル一式を作成

```
> rails generate scaffold product title:string  
description:text image_url:string
```

(2) 生成されたものを確認

(A) コントローラ, ビュー, モデルのファイル

app/controllers/products_controller.rb

app/models/product.rb

app/views/products/index.html.erb

app/views/products/show.html.erb

app/views/products/new.html.erb

app/views/products/edit.html.erb

app/views/products/_form.html.erb

新規作成と更新で利用するproductのフォーマット

Scaffoldを使ってみる

(2)生成されたものを確認

(B)マイグレーションファイル

db/migrate/[時刻]_create_products.rb

データベースにproductのスキーマを追加するファイル

(C)ヘルパーファイル

app/helpers/products_helper.rb

productに関するビューの共通処理を記述するファイル

(D)テストファイル

test/unit/product_test.rb

test/unit/helpers/products_helper_test.rb

productに関するテストを記述するファイル

(3)データベースにproductを反映

(2)(B)のマイグレーションファイルを利用する

```
> rake db:migrate
```

Scaffoldを使ってみる

(4) 自動生成された商品管理機能を触ってみる

> rails server

<http://localhost:3000/products>にアクセスする

The screenshot shows a simple web application interface. At the top, it says "Listing products". Below that is a table with four columns: "Title", "Desctiption", "Image", and "url". There is one row in the table. At the bottom, there is a link labeled "New Product".

Title	Desctiption	Image	url

New Product

商品登録をやってみる

モデルの仕組みについて

(1) app/models/product.rbを見てみる

Productクラスについてほとんど何も書かれていない
ただし、ProductはActiveRecordを継承している

(2) Productクラスを使ってみる

(A) コンソールを利用

```
> rails console
```

アプリケーションをコンソールで操作可能

[注意] windowsで文字化けせずにコンソールを使うには
準備が必要

Windowsでコンソール使う場合

Railsは文字列をutf-8として管理

→ Windowsのコンソールに正しく表示させるには準備が必要

[準備] config/initializers/windows.rbを作成

以下の内容を記述

```
if Bundler::WINDOWS
  require 'nkf'

  class << $stdout
    def write_with_conversion(str)
      write_without_conversion NKF.nkf('-sxm0', str.to_s)
    end
    alias_method_chain :write, :conversion
  end
end
```

モデルの仕組みについて

(2) Productモデルを使ってみる

(A) コンソールを利用する

```
> rails console
```

(B) Productモデルを操作

(a) 保存されているProductモデルのデータを一覧表示

```
irb(main) > Product.all
```

(b) 特定の条件に合うProductモデルを表示

```
irb(main) > Product.where(:title => “[タイトル]”)
```

[注意] データはutf-8のため、全角文字だと一致しない

(c) 特定の項目を表示

```
irb(main) > product = Product.first
```

```
irb(main) > product.title
```

データベースをオブジェクトとして扱える

モデルの仕組みについて

データベースの中身をオブジェクトとして見える仕組み
データベースとオブジェクトの連携が名前によって決まっている

(a) Product.all

productsテーブルの各行からProductオブジェクトを生成

(b) Product.where(:title => “[タイトル]”)

productsテーブルに対して, title 列が[タイトル]である行を検索し, 一致した行をProductオブジェクトを生成

(c) product = Product.first

product.title

productsテーブルの最初の行のtitle列を返す

RailsのO/RマッパーであるActiveRecordがデータベースに応じて動的にメソッドを生成する

マイグレーション

データベースの構造変更を管理する機能

例:今回のマイグレーションファイル

```
class CreateProducts < ActiveRecord::Migration
  def self.up
    create_table :products do |t|
      t.string :title
      t.text :desctiption
      t.string :image_url
      t.timestamps
    end
  end
  def self.down
    drop_table :products
  end
end
```

このマイグレーションファイル
を適用した時にデータベース
に加わる変更

すべてのマイグレーションファイルを適用 = 最終データベースの状態

マイグレーションを使ってみる

(1)商品にcodeの項目を追加

```
> rails generate migration add_code_to_product  
code:string
```

db/migrate/[日時]_add_code_to_product.rbを生成

(2)マイグレーションファイルを確認

```
class AddCodeToProduct < ActiveRecord::Migration  
  def self.up  
    add_column :products, :code, :string  
  end  
  def self.down  
    remove_column :products, :code  
  end  
end
```

マイグレーションを使ってみる

(3) マイグレーションファイルを編集
赤の部分を追記する

```
class AddCodeToProduct < ActiveRecord::Migration
  def self.up
    add_column :products, :code, :string
    Product.all.each do |product|
      product.code = product.id
      product.save
    end
  end
  def self.down
    remove_column :products, :code
  end
end
```

} すでに登録されている商品はデフォルトで code=idとなるよう設定

マイグレーションを使ってみる

(4) データベースに反映

```
> rake db:migrate
```

db/migrate/に未適用のマイグレーションファイルがあった場合
未適用のファイルを[日時]の小さい順から適用させていく

(5) Productモデルで操作できることを確認

```
> rails console
irb(main) > product = Product.first
irb(main) > product.code = "1111-aaaa"
```

追加した項目を利用

- (1) 新規作成と更新の入力フォームにcodeを追加

app/views/products/_form.html.erbの26行目に以下を追加

```
<div class = “field” >
    <%= f. label :code %><br />
    <%= f. text_field :code %>
</div>
```

コントローラは変更の必要なし

- (2) 一覧画面と詳細画面にcodeを追加

オブジェクトに対してcodeメソッドを使用

例)app/views/products/show.html.erbの17行目に以下を追加

```
<p>
    <b>Code: </b>
    <%= @product. code %>
</p>
```

ルーティングの設定

products/indexから任意のshowにアクセス

`http://localhost:3000/products/_1`

データベースのプライマリーキーを使ったURIではデータベースに同じ商品を再登録した際に使えなくなる

→ codeを利用したルーティングを追加

(1) 現在有効なURI一覧を表示

```
> rake routes
```

(2) Routeの設定

ルーティングはconfig/routes.rbに記述

(1)のルーティングは以下の記述で設定

```
resources :products
```

ルーティングの設定

(2) Routeの設定

config/routes.rbに赤の部分を追加

```
Depot::Application.routes.draw do
  resources :products
    match '/products/code/:code' => "products#show_by_code"
  end
```

(3) show_by_codeアクションの追加

app/controllers/products_controller.rbに以下を追加

```
def show_by_code
  @product = Product.where(:code => params[:code]).first

  respond_to do |format|
    format.html { render :action => "show" }
  end
end
```

(4) [http://localhost:3000/products/code/\[任意のcode\]](http://localhost:3000/products/code/[任意のcode])

バリデーションの設定

codeをユニークかつ必須にさせたい

(1) モデルにバリデーションを設定

app/models/product.rbに以下の内容を追加

```
class Product < ActiveRecord::Base
  validates :code, :presence => true, :uniqueness => true
end
```

(2) <http://localhost:3000/products/>にアクセス

Railsべきべからず集

モデルの変更は慎重するべし

[トラブル]

必要に応じて適当にモデルを作成してきた。命名規則の統一がで
きていなかったので、リファクタリングすると大変な工数に。

[原因]

CoCのために、モデルに関連する名前は、ビューとコントローラ内
に大量に使用されている

[対処]

マイグレーションがあるからと思って適当にモデルを作らない

テストしやすいコードを心がけるべし

[トラブル]

モデル名の変更をしようとした時、影響の範囲が予測できなかつたのでCucumber(capybara)のテストコードを用意しようとしたら、一部ページのテストコードが書けない

cucumber : 受け入れテスト用ツール

[原因]

JavaScriptを利用したページ遷移はテストができない

[対処]

テストを書くこと前提でコードを書く

- (1) JavaScriptを多用しすぎない(Unobtrusive JavaScript)
- (2) RESTを意識したページ遷移をする

Rails(Ruby)はテストコードで動作を保証しないと、スペルミスをしていても例外を発生させず動くことがよくある

国際化は多国語未対応でも使うべし

[トラブル]

表示されている項目名やボタン名がバラバラになる

(例) Productのcode => 「製品番号」「商品コード」「コード」

新規作成画面のsubmitボタン => 「作成」「登録」「Create」

これはテストコードを書く際にも問題となる

[原因]

様々なタイミングで様々な人がviewのファイルを追加していく

[対処]

国際化(i18n)を利用する

config/locales/translation_ja.yml

```
ja:  
  activerecord:  
    attributes:  
      product:  
        code: "商品コード"
```

ヘルスチェック用ページを用意するべし

[トラブル]

Rails3.0移行時にConfig内の設定によってエラーが多く発生した。
また、タイムゾーンがUTCとなっている問題が何度も発生した。

[原因]

Configの設定が正しいかどうかを確認(テスト)しづらい

[対処]

- (1) 設定を表示するページ(ヘルスチェック用ページ)を用意する
- (2) ヘルスチェック用ページをチェックするテストコードを書く

ビジネスロジックはモデルに書くべし

[トラブル]

- (1) コントローラのファイルがとても大きくなる
- (2) コントローラに手を加えるとデータの整合性が崩れやすい

[原因]

ActiveRecordを利用すると、データベースのオブジェクトのように扱えるため、商品の購入処理といった複数のモデルの複数のカラムを操作する処理をコントローラに書きがちになる

[対処]

データの整合性は、モデルが責任を持つのが正しいMVCである。ビジネスロジックにあたるものはモデルのメソッドとし、コントローラはこれを呼び出すのに徹する
(コントローラでモデルの要素を書き換えたりしない)

安いな継承をするべからず

[トラブル]

間違えて「提出資料」としてアップロードした文書を、「議事録」に変更することができなかった

[原因]

単一テーブル継承(STI)が便利なので、振る舞いの違いを安いに継承を利用して実装したこと

(「文書」クラスの中で、継承を利用して「議事録」や「提出資料」といった振る舞いの違うサブクラスを定義していた)

[対処]

「継承よりも委譲」の原則を守る

「文書」クラスとして生成し、委譲 Mix-in(プラグイン)を利用して振る舞いを変える

Lightningトーク

懇親会

場所：快食市場 うる寅地蔵

住所：岡山市北区津島新野1丁目5

電話番号：086 – 255 – 6688

時間：18:30 ~ 20:30

料金：3000円

予約者名：山口