

Ruby on Rails

インストールから簡単なアプリの構
築まで

何故、Ruby on railsなの？

それは、そこに山があるから

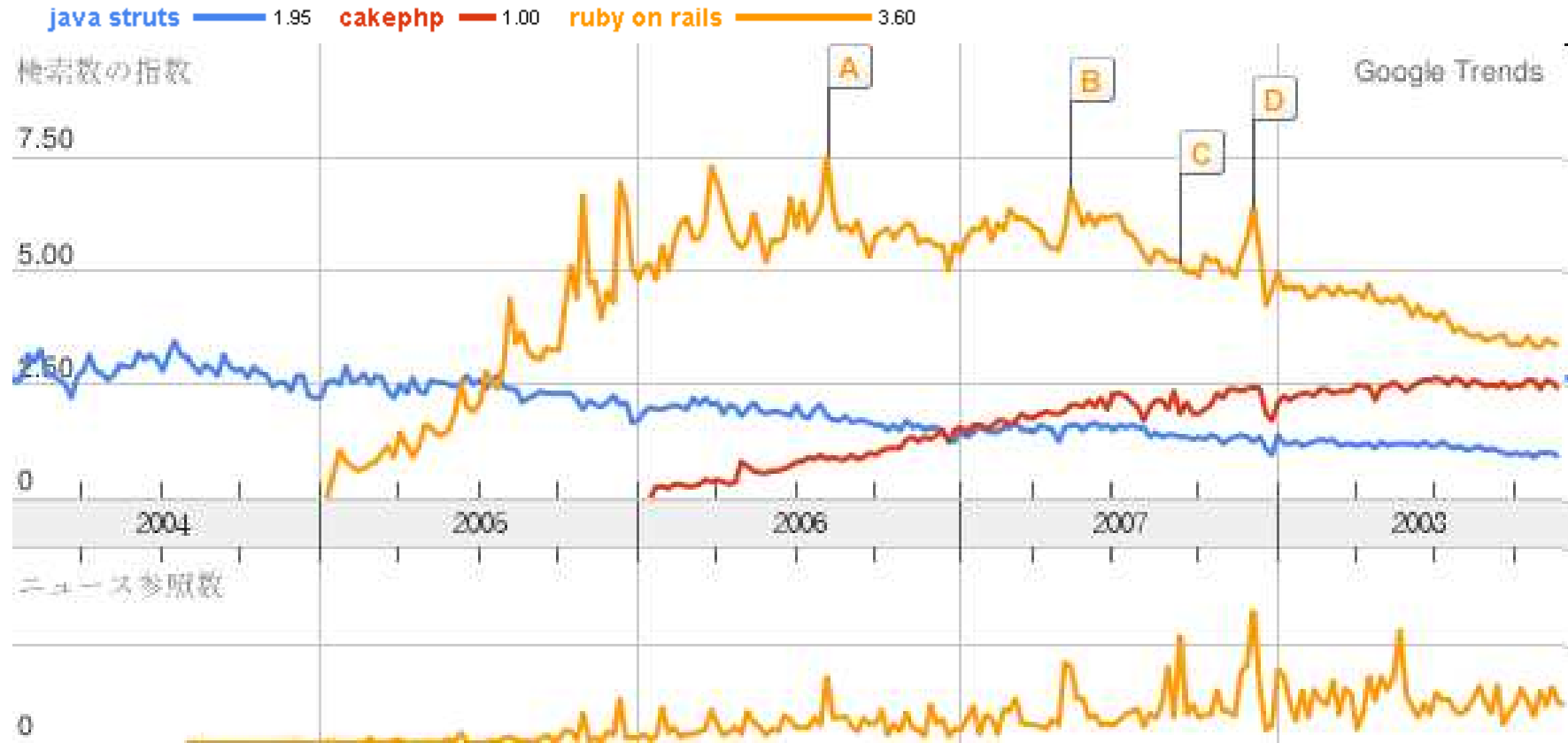
ではなくて、以下の理由からです。

- シェアが高い
 - 多くの人々が改良しているので枯れている
 - ドキュメントが豊富
 - 需要がある
- railsの思想を学ぶことにより優れた設計を身につけたい。
- たまには新しい言語を試したい。。。 (おい

本当にシェアが高いの？

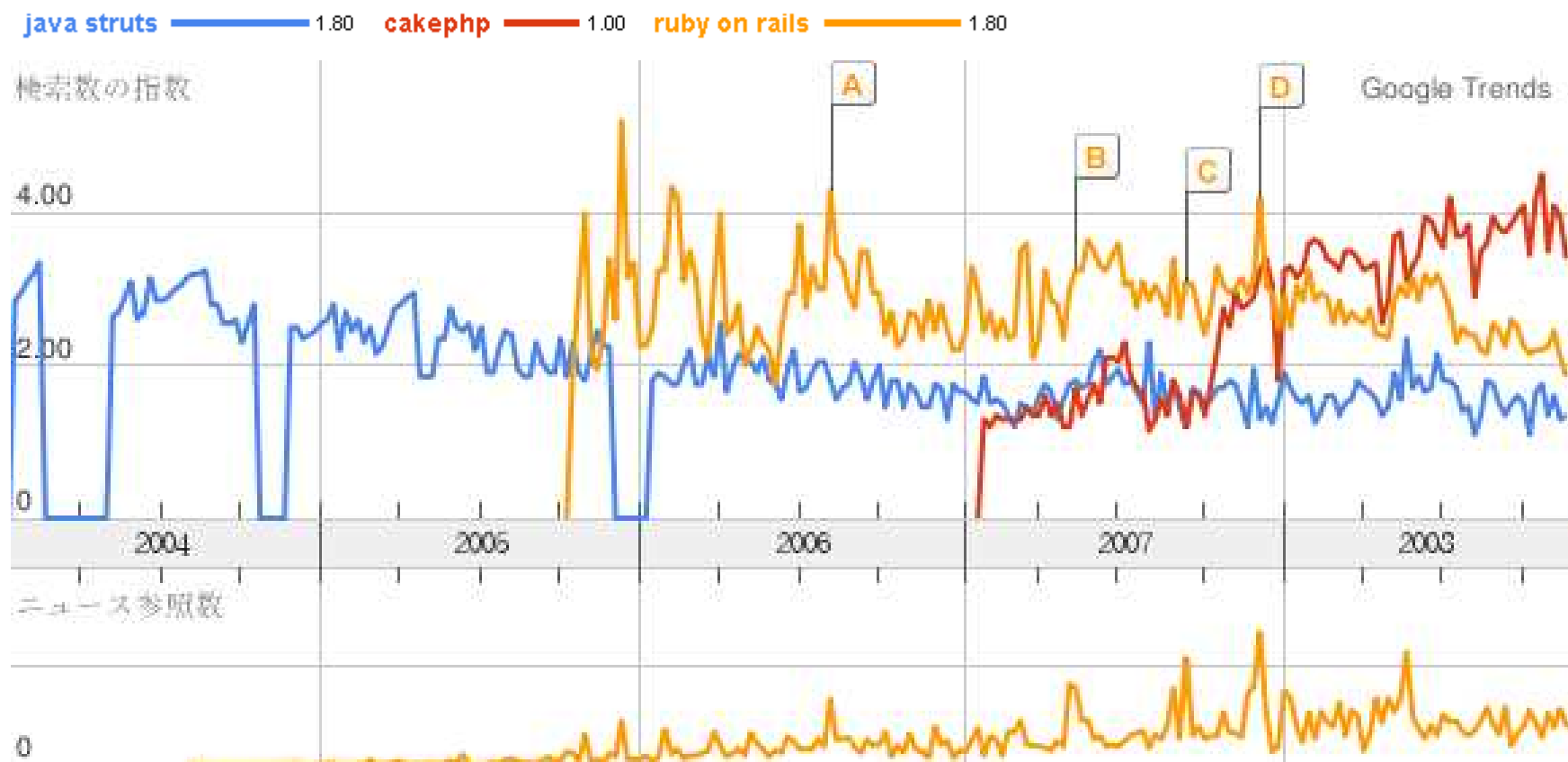
各言語の代表的なフレームワークを
Google Trendで調べてみた。

まずは世界



railsが数多く検索されCakePHPが
追い上げています。

次に日本



CakePHPがrailsを抜いて上昇中

もうrailsに決定しているので
見なかったことに。。。(おい

CakePHPと比較し

- 翻訳された日本語ドキュメントが豊富
 - railsはWebドキュメントが分散されている
- php使いは多い
- ほぼ全てのサーバーは標準でphpが使える
- railsの欠点であるViewで使用するテンプレートをデザイナーさんが使い慣れた物に切り替えすることが出来る。

ということでCakePHPが良いのかも

次に、お詫びを。。。(汗

Linux系に絞って話をします。

Windowsの方ごめんなさいm(____)m

まずはrubyのインストール

インストールしようと思ったら
ruby1.8と1.9があるでは
あーりませんか！

どう違うんだろう。。。。

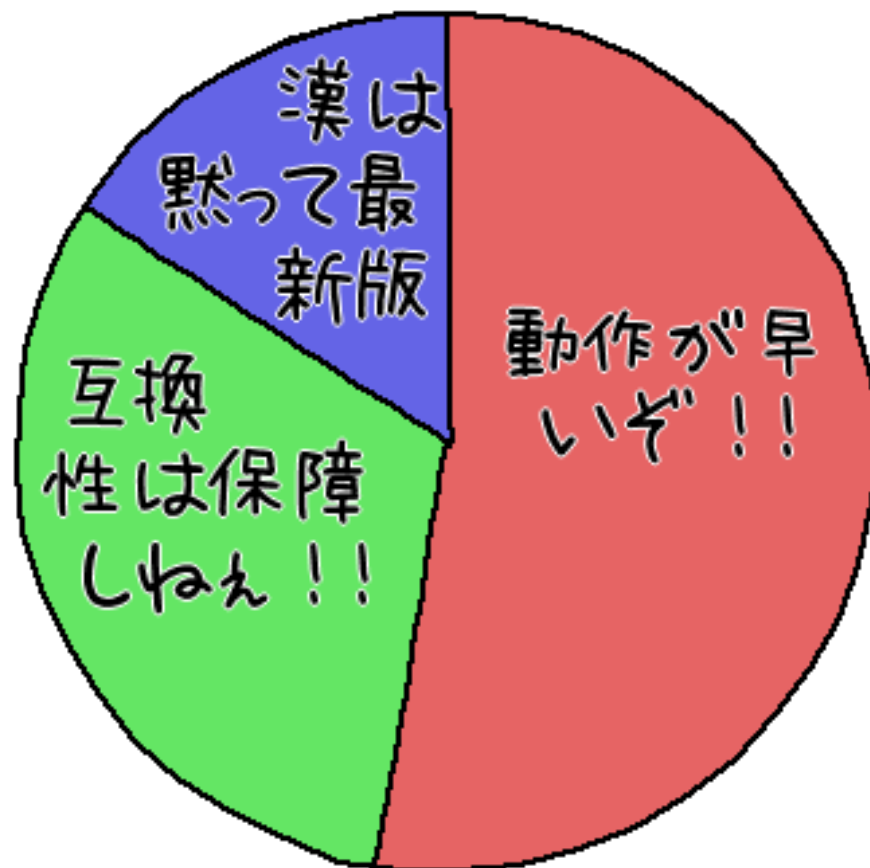
調べてみた。

Google [検索表](#)

ウェブ全体から検索 日本語のページを検索

ウェブ

ruby1.9の主張を円グラフに。。。。



真面目に変更点をリストにすると 以下の通り

- YARVによる処理速度の向上
- M17N(多言語化)のサポート
- 軽量なスレッドを実現するFiberの導入
- RubyGems(ライブラリ管理)の標準付属
- Rake(make ruby版)の標準付属

人柱になっている余裕が無いので
今回はruby1.8を使います。

ひっぱいとして
ごめんなさいm(____)m

補足として安定 & 高速動作を
お求めの方にはJRubyという
JavaVM上で動く物がございます。。。
JRubyはJavaのライブラリも
使用出来るようでございます。

また、使用メモリを33%削減した
Ruby Enterprise Edition
もございます。

とりあえずrubyをインストール

Ubuntu

```
# apt-get install ruby  
# apt-get install ruby-devel  
# apt-get install rubygems
```

Fedora

```
# yum install ruby  
# yum install ruby-devel  
# yum install rubygems
```

ここでrubygemsについて解説

phpに置けるpear

perlに置けるCPAN

debian/ubuntu/Vineに置けるapt

Fedoraに置けるyum

のような物です。。。

gemを使ってrailsをインストール

```
# gem install rails ¥  
  --include-dependencies
```

railsとその依存しているものを
全てインストールしてくれます。

次にrailsアプリを呼び出して
くれる実行環境をインストール

むむむ。。。。

人生いろいろ

サーバーもいろいろ

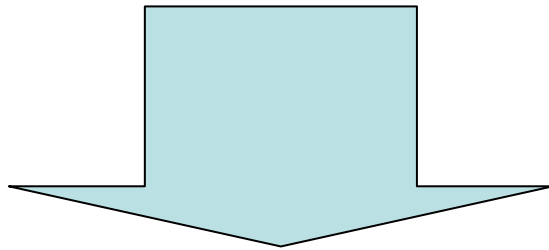
実行環境を調べてみる

主な実行環境

- apache cgi
- apache+FastCGI
- Webrick
- Mongrel
- thin
- Ebb
- apache+Passenger(mod_rails)

apache cgi

- とにかく遅い！！



問題外???

apache+FastCGI

- 動作速度はプロセス常駐型で○
- 設定が面倒

Webrick

- ruby標準添付
- 設定簡単
- cgiほどではないが遅い
- テスト環境のみで使用するのがよさげ

Mongrel

- gemで即導入
- railsの実行が早い
- 通常の処理は遅い
- 他のWebサーバーと組み合わせる必要有り
- 作者がグレた。。。

thin

- gemで即導入
- Mongrelよりrailsの実行が早い
- 通常のリクエストの処理は遅い
- 他のWebサーバーと組み合わせて使用する必要あり

Ebb

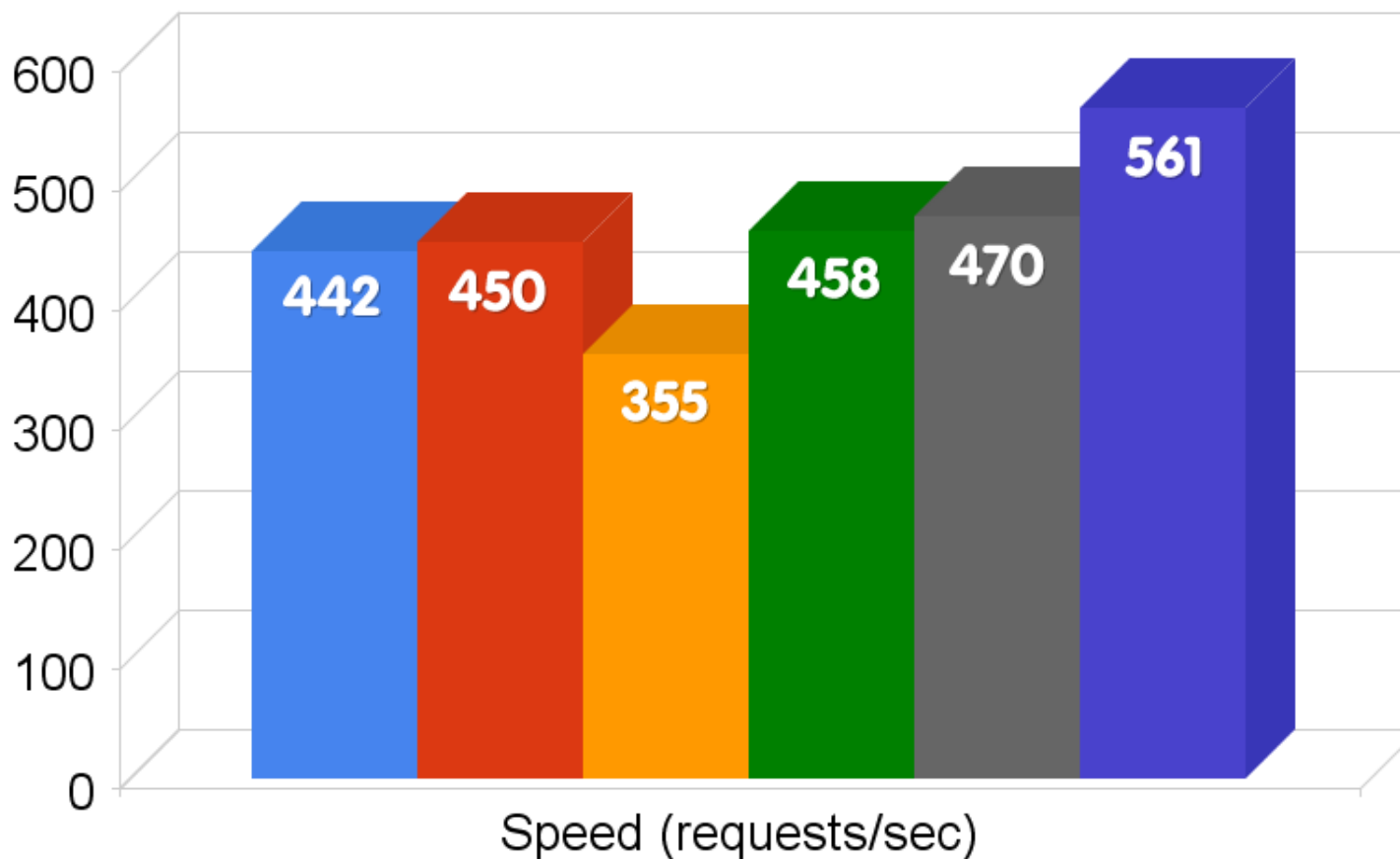
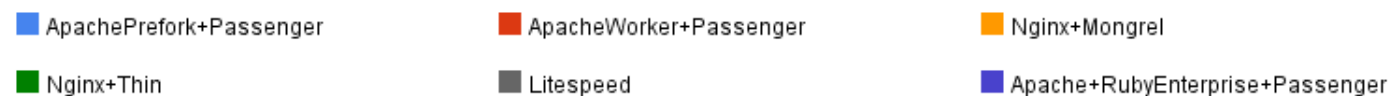
- gemで即導入
- thinよりrailsの実行が早い
- 通常のリクエストの処理は遅い
- 他のWebサーバーと組み合わせて使用する必要あり

apache+Passenger

- gemで即導入
- railsの実行が早い
- apacheモジュールとして動作
- apacheで完結出来る
- Windowsは未対応

Passengerのベンチ(大本営発表)

Speed comparison



実行環境比較結論

調べた結果、動作速度は以下の順

Ebb>= apache+Passenger

>thin>Mongrel>Webrick>cgi

(FastCGIは比較が発見できず)

実使用に耐えられる環境の構築のしやすさはapache+Passengerが群を抜いているとのことで、apache+Passengerを導入することに

前準備としてapacheインストール

Ubuntu

```
# apt-get install apache2
```

```
# apt-get install apache2-threaded-dev
```

```
# apt-get install g++
```

Fedora

```
# yum install httpd
```

```
# yum install httpd-devel
```

```
# yum install gcc-c++
```

passengerのインストール(1)

Ubuntu、fedora共通

```
# gem install passenger ¥  
  --include-dependencies
```

Ubuntuのみ

```
# export ¥  
  PATH=$PATH:/var/lib/gems/1.8/bin/
```

Ubuntu、fedora共通

```
# passenger-install-apache2-module
```

passengerのインストール(2)

```
Welcome to the Phusion Passenger Apache 2 module installer, v2.0.3.
```

```
This installer will guide you through the entire installation process. It shouldn't take more than 3 minutes in total.
```

```
Here's what you can expect from the installation process:
```

1. The Apache 2 module will be installed for you.
2. You'll learn how to configure Apache.
3. You'll learn how to deploy a Ruby on Rails application.

```
Don't worry if anything goes wrong. This installer will advise you on how to solve any problems.
```

```
Press Enter to continue, or Ctrl-C to abort.
```

```
1
```

1を選択

passengerのインストール(3)

```
Checking for required software...
* GNU C++ compiler... found at /usr/bin/g++
* Ruby development headers... found
* OpenSSL support for Ruby... found
* RubyGems... found
* Rake... found at /var/lib/gems/1.8/bin/rake
* Apache 2... found at /usr/sbin/apache2
* Apache 2 development headers... found at /usr/bin/apxs2
* Apache Portable Runtime (APR) development headers... found at /usr/bin/apr-1-
config
* fastthread... found
* rack... found
-----
-----
```

必要ソフトウェアのチェック
インストールされていないものが
あればインストール

passengerのインストール(4)

```
The Apache 2 module was successfully installed.  
Please edit your Apache configuration file, and add these lines:  
  
LoadModule passenger_module /var/lib/gems/1.8/gems/passenger-2.0.3/ext/apache  
2/mod_passenger.so  
PassengerRoot /var/lib/gems/1.8/gems/passenger-2.0.3  
PassengerRuby /usr/bin/ruby1.8  
  
After you restart Apache, you are ready to deploy any number of Ruby on Rails  
applications on Apache, without any further Ruby on Rails-specific  
configuration!  
  
Press ENTER to continue.
```

apacheの設定ファイルに
赤丸の内容を追加します。

apacheの設定(Fedora)

```
# cd /etc/apache2/conf.d
```

```
# touch passenger.conf
```

```
# <<好きなエディタ>> passenger.conf
```

さっきの内容をコピー&ペースト

```
# /etc/init.d/apache2 restart
```

確認

```
# apache2ctl -M | grep passenger
```

passenger_moduleが含まれていたらOK

apacheの設定 | Ubuntu(1)

```
# cd /etc/apache2/mods-available  
# touch passenger.conf passenger.load  
# <<好きなエディタ>> passenger.load
```

以下の行を追加

```
LoadModule passenger_module /var/lib/gems/1.8/gems/passenger-  
2.0.3/ext/apache2/mod_passenger.so
```

```
# バージョンによってパスが異なるため注意
```

apacheの設定 | Ubuntu(2)

<<好きなエディタ>> passenger.conf

以下の行を追加

```
<IfModule mod_passenger.c>
```

```
  # バージョンによってパスが違うので注意
```

```
  PassengerRoot /var/lib/gems/1.8/gems/passenger-2.0.3
```

```
  # Ruby Enterprise Editionを使用する場合は以下のパスを変更
```

```
  #PassengerRuby /usr/bin/ruby1.8
```

```
  PassengerRuby /opt/ruby-enterprise/bin/ruby
```

```
</IfModule>
```

apacheの設定 | Ubuntu(3)

a2enmodでpassengerモジュールを追加(ただリンクを張るだけですが。。。)し、apache再起動

```
# a2enmod
```

Which module would you like to enable?

Your choices are: actions alias asis auth_basic auth_digest authn_alias
authn_anon authn_dbd authn_dbm authn_default authn_file authnz_ldap
authz_dbm authz_default authz_groupfile authz_host authz_owner
authz_user autoindex cache cern_meta cgi cgid charset_lite dav dav_fs
dav_lock dbd deflate dir disk_cache dump_io env expires ext_filter
file_cache filter headers ident imagemap include info ldap log_forensic
mem_cache mime mime_magic negotiation passenger proxy proxy_ajp
proxy_balancer proxy_connect proxy_ftp proxy_http rewrite setenvif spelling
ssl status substitute suexec unique_id userdir usertrack version vhost_alias

Module name? **passenger**

This module is already enabled!

```
# /etc/init.d/apache2 restart
```

apacheのバーチャルホストにrails
プロジェクトのルートディレクトリを
登録して終了です。

```
<VirtualHost *:80>
```

```
ServerName サーバー名
```

```
DocumentRoot railsプロジェクトのパス
```

```
#もしDocumentRootがrailsプロジェクトの
```

```
#ルートディレクトリで無い場合は以下で指定
```

```
# RailsBaseURI /rails
```

```
</VirtualHost>
```

環境が整ったところで
railsの説明

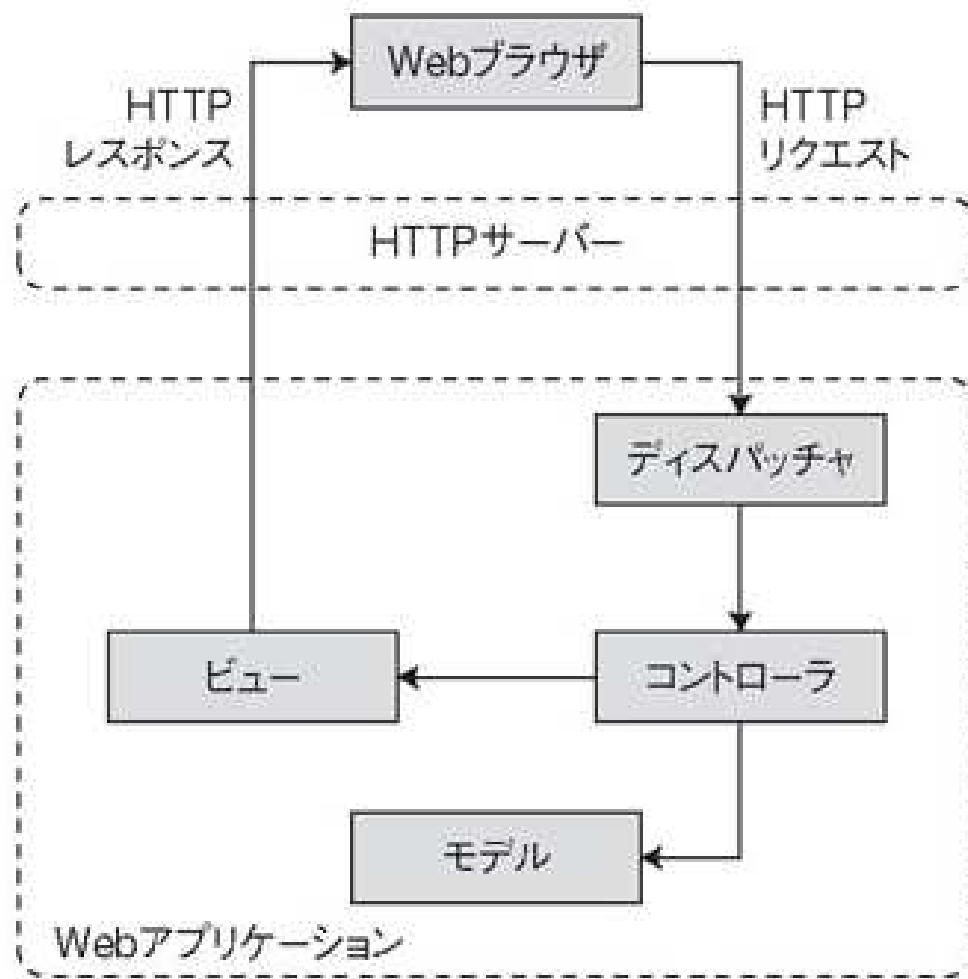
まず以下のコマンドで
railsプロジェクトを作成します。

\$ rails プロジェクト名

するとrailsアプリの実行に必要な
ファイルがコピーされます。

次にMCVに沿って話をします。
(あれ？順序が。。。)

railsのMVCモデル



railsのモデル規則を簡単に説明

- 主キーはオートナンバー
- 主キーのカラム名は”id”(推奨)
- テーブル名は複数形(推奨)
- リレーションを張るカラム名は”単数形_id”(推奨)
- レコード作成日を自動入力したいならばcreated_on
- レコード作成日時を自動入力したいならばcreated_at
- レコード更新日を自動入力したいならばupdate_on
- レコード更新日時を自動入力したいならばupdate_at
- テーブルを使用するためのクラス名は単数形
(entries→Entry)

まだまだありますが、命名規則によって処理を書かなくても良い仕組みがいっぱいです。

railsのモデルで何をするの？

データベースを完全隠蔽化し、どんなDBでも共通した手順を提供します。

モデルには何を書くの？

app/models/モデル名.rbにモデルの制御を書きます。

- 各テーブル間のリレーションの記述
 - 変更されたらメールする、ロギング等のトリガー動作
- などなど

データベースの設定

データベースは
プロジェクト/config/database.yml
で設定します。

database.ymlの例

```
production:  
  adapter: mysql  
  database: redmine_development  
  host: localhost  
  username: root  
  password:  
  encoding: utf8
```

この設定だけで殆どの場合DBの違いを意識することなくDBを扱えます。

テーブルの定義

テーブルの定義は直接DBに行うことも出来ますが
プロジェクト/db/migrate/
の中のファイルでやると、バージョン
を戻したり、DBの再構築が簡単に出来て便利です。

db/migrate/のファイル説明

バージョン_foo_hogehoge.rb

というファイル命名規則になっています。
ファイルの先頭のバージョンでDB構造
を戻したりすることが出来ます。

db/migrate/VERSION_create_テーブル名.rb の設定例

```
class CreateEntries < ActiveRecord::Migration
```

```
  def self.up
```

```
    create_table :entries do |t|
```

```
      t.column :title, :string, :null => false
```

```
      t.column :name, :string, :default => "名無し", :null => false
```

```
      t.column :mail, :string, :null => false
```

```
      t.column :body, :text, :null => false
```

```
      t.column :created_on, :datetime, :null => false
```

```
    end
```

```
  end
```

self.upメソッドはバージョンアップ時実行

self.downはバージョンダウン時実行

```
  def self.down
```

```
    drop_table :entries
```

```
  end
```

```
end
```

テーブル作成

以下でDBにテーブルを作成出来ます。

```
$ rake db:migrate
```

バージョンを指定するには以下

```
$ rake db:migrate VERSION=バージョン
```

ダウングレードするには

```
$ rake db:migrate:down VERSION=バージョン
```

環境を指定するにはRAILS_ENV =環境名

先ほどの例ではentriesテーブル
が作成されます。

テーブル作成例

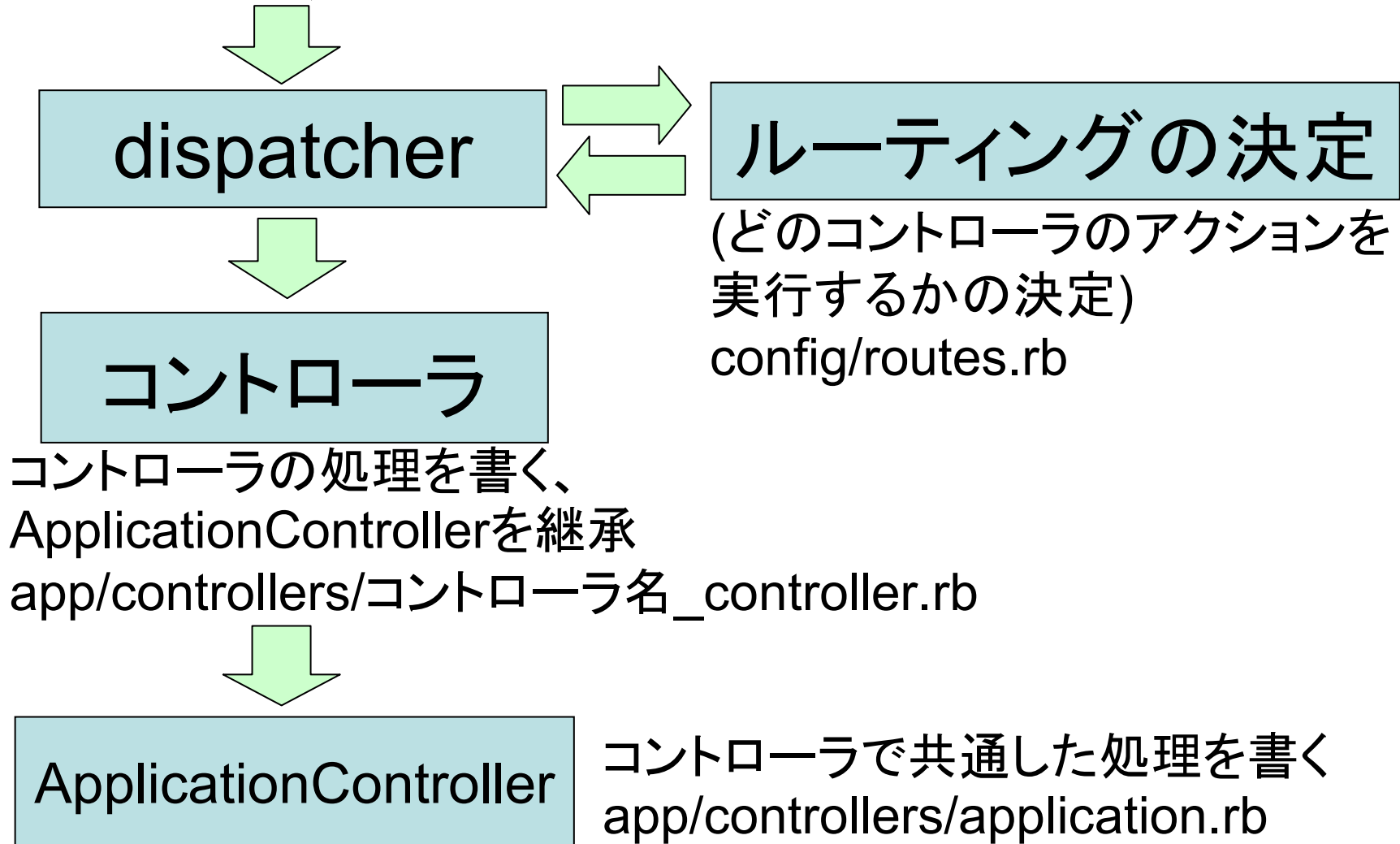
以下ではVERSION=0001でproduction環境のDBを構築しています。

```
$ rake db:migrate VERSION=0001 RAILS_ENV=production
(in /home/fuku/rails/hello)
== CreateEntries: migrating
=====
=====
-- create_table(:entries)
   -> 0.0257s
== CreateEntries: migrated (0.0583s)
=====
```

次にコントローラをざっくり説明

コントローラとは、ユーザーからの入力を受け、ビューやモデルを操作するオブジェクトです。

コントローラの構造をざくっと説明 HTTPリクエスト



とりあえずコントローラの作成

bbsコントローラの雛形の作成をします。

```
$ script/generate controller bbs
exists app/controllers/
exists app/helpers/
create app/views/bbs ← Viewを保存するディレクトリ
exists test/functional/
create app/controllers/bbs_controller.rb ←コントローラ本体
create test/functional/bbs_controller_bbs.rb
create app/helpers/bbs_helper.rb ←ビューで使用するヘルパー
```

一緒にViewに関するものも作成されます。

基本となるindexアクションを作成

```
class BbsController < ApplicationController
  def index
    @entries = Entry.find(:all)
  end
end
```

action名

モデルオブジェクトの頭は大文字

entryモデル

全てのレコード

レコードはallなので配列で返る

findメソッドで
entriesテーブルをselect

これだけでentriesテーブルの中身を
@entries配列に全て取り出す処理の完成

次にViewの作成

script/generateのオプションにView
が無い。。。

以下のファイル名でViewを
手動で作成します。

app/views/コントローラ名/アクション名.rhtml

例ではbbsコントローラのindexアクションのViewなのでファイル名は以下

app/views/bbs/index.rhtml

Viewを記述

```
<html>
<head>
  <title>テスト掲示板</title>
  <meta http-equiv="Content-Style-Type" content="text/css" />
  <%= stylesheet_link_tag "style.css" %>
</head>
<body>
  <h1>テスト掲示板</h1>
  <% @entries.each do |@entry| %>
    <dl>
      <dt>タイトル</dt>
      <dd><%=h @entry.title %> </dd>
      <dt>名前</dt>
      <dd><%=h @entry.name %> </dd>
      <dt>メールアドレス</dt>
      <dd><%=h @entry.mail %> </dd>
      <dt>本文</dt>
      <dd><%=h @entry.body %> </dd>
    </dl>
  <% end %>
</body>
</html>
```

\$ script/server -e 環境名
でrails実行環境を起動すると以下のように



テスト掲示板

タイトル	testtitle
名前	testname
メールアドレス	test@name.jp
本文	aaaaaaaaaaa

タイトル	testtitle2
名前	testname
メールアドレス	test@name.jp
本文	aaaaaaaaaaa

passenger(mod_rails)
ではproduction環境で
動作します。

これではいくら簡単な物と言っても
寂しいので投稿機能を足します。

app/controllers/bbs_controller.rb
に以下のアクション(メソッド?)を追加

```
def add_entry
  @entry = Entry.new(params[:new_entry])
  if @entry.save
    redirect_to(:action => "index")
  else
    render_text "登録できませんでした。"
  end
end
```

app/views/bbs/index.rhtml

に以下を追加

```
<%= form_tag :action=>"add_entry" %>
<dl>
  <dt>タイトル</dt>
  <dd><%= text_field("new_entry", "title", "size"=>16) %></dd>
  <dt>名前</dt>
  <dd><%= text_field("new_entry", "name", "size"=>16) %></dd>
  <dt>メールアドレス</dt>
  <dd><%= text_field("new_entry", "mail", "size"=>32) %></dd>
  <dt>本文</dt>
  <dd><%= text_area("new_entry", "body", "cols"=>60,
"rows"=>10) %></dd>
</dl>
  <input type="submit" value="Add" />
</form>
```

こんな感じになりました。

テスト掲示板

タイトル	testtitle
名前	testname
メールアドレス	test@name.jp
本文	aaaaaaaaaa

タイトル	testtitle2
名前	testname
メールアドレス	test@name.jp
本文	aaaaaaaaaa

タイトル	<input type="text"/>
名前	<input type="text"/>
メールアドレス	<input type="text"/>
本文	<input type="text"/>

<input type="text"/>
<input type="text"/>
<input type="text"/>
<input type="text"/>

Add

個人的まとめ

- Railsの真髄はどこに何が記述されているかが明確になることと見つけたり
- 規約に法ることによって処理を省略し楽できます。
- 開発効率は開発言語+テンプレートエンジンよりDBを意識しなくなる分早くなる
- 関数呼び出しを主体にする言語の使い手にとってはrailsよりruby言語の特徴に戸惑います。