Web 2.0 on Rails

前田 修吾

ネットワーク応用通信研究所

自己紹介

名前

前田 修吾

所属

ネットワーク応用通信研究所(NaCl)Rubyアソシエーション

Railsとの出会い

- 2005年4月
- 仕事のため
  - SOAPがメイン

Railsとの関わり(1)

- パッチ
  - 文字コード関連
  - 悲観的ロック
  - Bug fix
  - パフォーマンスチューニング

Railsとの関わり(2)

- 「RailsによるアジャイルWebアプリケーション開発」監訳

本日のテーマ

- Rails
- Ajax
- REST

Rails

- Ruby on Rails
  - Rubyの
  - Rubyによる
  - Rubyのための
  - Web Application Framework

Railsのコンセプト

- DRY
  - Don't Repeat Yourself
- CoC
  - Convention over Configuration
- 生産性 > 柔軟性

DRY

- 重複を排除する
  - コピペは悪
- プログラマの常識
  - でも実践は難しい

CoC

- 設定より規約
- 命名規約により設定を省略
  - ファイル名をクラスから推測
  - テーブル名をクラス名から推測
- 使いやすいデフォルト
  - 設定で上書きも可能

生産性 > 柔軟性

- 思い切った割り切り
- 柔軟性よりも生産性を重視
- 80%のWebアプリケーションを効率的に書ければよい

Railsの特徴

- オールインワン
- 自動生成
- プラグイン

オールインワン

- MVCをセットで提供
  - 密結合
- その他のユーティリティ
  - テストのサポートなど

自動生成

- scaffold
  - シンプルなCRUD
  - いじりやすい

プラグイン

- サードパーティ製のプラグイン
- フレームワークの機能を拡張
- Railsに取り込まれることも

コンポーネント

- ActiveRecord
- ActionView
- ActionController

ActiveRecord

- モデルを担当
- O/Rマッパー
- PofEAAの同名のパターンに由来

対応

テーブル	クラス
行	オブジェクト
列	属性

モデルの例

```
class Product < ActiveRecord::Base
end
```


ActionView

- ビューを担当
- デフォルトのビューはeRuby

ビューの例

```
名前: <%= h(@product.name) %><br/>
価格: <%= h(@product.price) %><br/>
```


ActionController

- コントローラを担当
- アクション
  - メソッドでリクエストを処理
- インスタンス変数をビューから参照することができる

コントローラの例

```
class ProductsController
  def show
    @product = Product.find(params[:id])
  end
end
```


デモ

- 製品カタログ

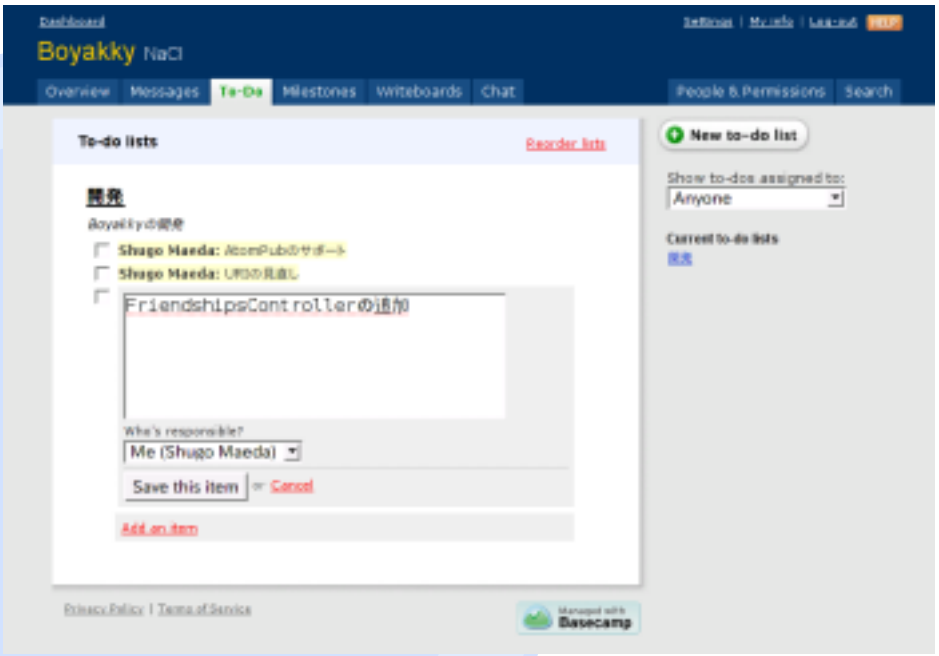
Web 2.0 on Rails

- Basecamp
- @nifty TimeLine
- アバウトミー
- Twitter

Basecamp

- 37signals社
  - DHH(=Rails作者)
- プロジェクト管理

Basecampの画面



TimeLine

- ソーシャルタイムライン
- 年表の共有

TimeLineの画面

アバウトミー

- 自分発見プロフィール
- みんなに質問

アバウトミーの画面

Twitter

- What are you doing?
  - 今してることをポスト
  - 実際は任意の短いメッセージ
- ゆるいチャット風
  - 返事がなくてもイタくない
  - ということにしたい

Twitterの画面

Why Rails?

- なぜWeb 2.0にRailsが適しているのか?
  - 変化に対する柔軟性
  - キーテクノロジーのサポート

変化に対する柔軟性

- 絶えず変化する仕様
  - 永遠のベータ
- 変化への対応
  - 軽量な開発サイクル
  - DRY

キーテクノロジー

- キーテクノロジーのサポート
  - Ajax
  - REST
  - OpenID

サンプル

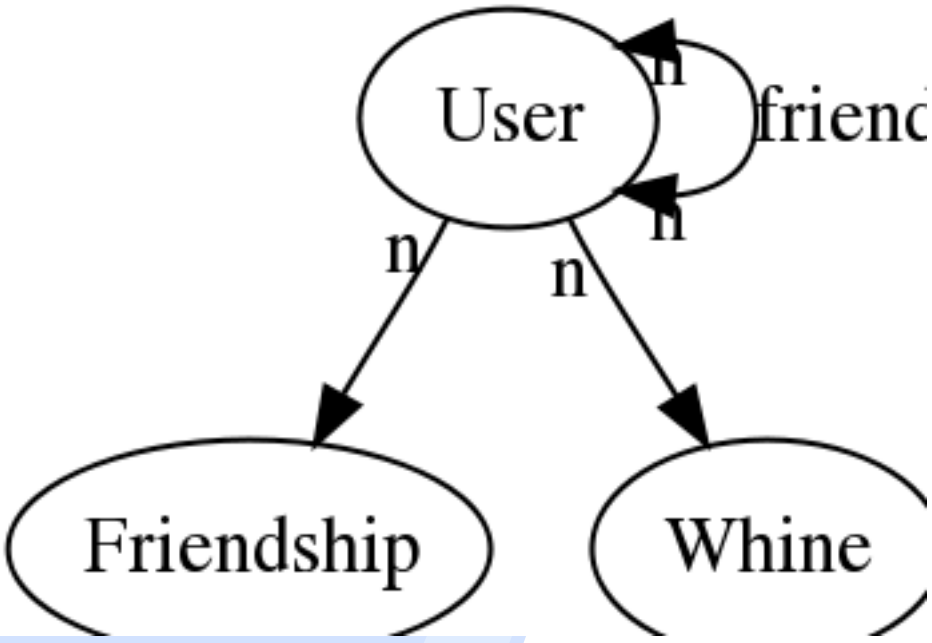
- Boyakky
  - Twitterもどき
  - Boyakky = ぼやき
- Railsのバージョン
  - 2.0 Preview Release

Boyakkyの画面

デモ

- 実際の動作

モデル



User

- ユーザを表現
- usersテーブルに対応
  - テーブル名は小文字・複数形

テーブル定義

```
create_table "users", :force => true do |t|
  t.column :login,              :string
  t.column :email,              :string
  t.column :crypted_password,   :string,
           :limit => 40
  t.column :salt,               :string,
           :limit => 40
  t.column :created_at,         :datetime
  t.column :updated_at,         :datetime
  t.column :remember_token,     :string
  t.column :remember_token_expires_at,
           :datetime
end
```


クラス定義

```
class User < ActiveRecord::Base
  has_many :whines, :include => :user,
           :order => "whines.created_at desc"
  has_many :friendships
  has_many :friends, :through => :friendships
end
```


Whine

- ぼやきを表現
- User/Whine = 1対多

テーブル定義

```
create_table :whines do |t|
  t.integer :user_id
  t.text :body

  t.timestamps
end
add_index :whines, :user_id
```


クラス定義

```
class Whine < ActiveRecord::Base
  belongs_to :user
  attr_protected :user_id
  validates_presence_of :body
end
```


Friendship

- 友達関係(双方向)を表現
- User/User = 多対多

テーブル定義

```
create_table :friendships do |t|
  t.integer :user_id
  t.integer :friend_id

  t.timestamps
end
add_index :friendships, [:user_id, :friend_id],
          :unique => true
```


クラス定義

```
class Friendship < ActiveRecord::Base
  belongs_to :user
  belongs_to :friend, :class_name => "User"
end
```


コントローラ

- 3つのコントローラを使用
  - AccountController
  - WhinesController
  - FriendshipsController
- WhinesControllerがメイン

Account

- ログイン処理
- 自動生成
  - acts_as_authenticated

Whines

- ぼやきの操作
  - ホーム(home)
  - 一覧(index)
  - 表示(show)
  - 作成(create)
  - 削除(destroy)

Friendships

- 友達関係の操作
  - 一覧(index)
  - 作成(show)
  - 削除(destroy)

ホーム画面

- /home/
- ログイン後に遷移
- ぼやき送信フォーム
- 自分と友達のぼやき一覧

homeアクション

```
def home
  # ビューで使用する変数の設定
  @whine = Whine.new
  @whines = current_user.whines_with_friends(:limit => 20)
end
```


ぼやき一覧の取得

```
class User < ActiveRecord::Base
  def whines_with_friends(options = {})
    user_ids = friendships.collect { |i| i.friend_id }
    user_ids.push(id)
    opts = options.merge(:conditions => [
                           "user_id in (?)",
                           user_ids
                         ],
                         :order => "whines.created_at desc",
                         :include => :user)
    return Whine.find(:all, opts)
  end
end
```


生成されるSQL

```
SELECT whines.`id` AS t0_r0, whines.`user_id` AS t0_r1,
       whines.`body` AS t0_r2, whines.`created_at` AS t0_r3,
       whines.`updated_at` AS t0_r4, users.`id` AS t1_r0,
       users.`login` AS t1_r1, users.`email` AS t1_r2,
       users.`crypted_password` AS t1_r3, users.`salt` AS t1_r4,
       users.`created_at` AS t1_r5, users.`updated_at` AS t1_r6,
       users.`remember_token` AS t1_r7,
       users.`remember_token_expires_at` AS t1_r8
  FROM whines
  LEFT OUTER JOIN users ON users.id = whines.user_id
  WHERE (user_id in (2,3,4,1)) ORDER BY whines.created_at desc
```


home.html.erb

```
<% remote_form_for(@whine,
                   :before => show_indicator,
                   :complete => hide_indicator) do |f| %>
  <%= f.text_area(:body, :cols => 50, :rows => 2) %>
  <%= f.submit("送信") %>
<% end %>

<div id="whines">
<% for whine in @whines %>
  <%= render(:partial => "whine", :object => whine) %>
<% end %>
</div>
```


_whine.html.erb

```
<div id="whine_<%= h(whine.id) %>" class="whine">
  <%= image_tag("emoticon_unhappy.png") %>
  <%= link_to(h(whine.user.login),
              :controller => "whines",
              :user_login => whine.user.login) %>
  <%=h whine.body %>
  <span class="date">
    <%=h whine.created_at.strftime("%Y/%m/%d %H:%M:%S") %>
  </span>
  <% if whine.user == current_user %>
  <%= link_to_remote(image_tag("bin.png", :alt => "削除"),
                     :url => whine,
                     :method => :delete,
                     :confirm => "本当に削除しますか?",
                     :before => show_indicator,
                     :complete => hide_indicator) %>
  <% end %>
  <%= link_to('permalink', whine) %>
</div>
```


remote_form_for

- モデルを作成・編集するためのフォームを生成
- remote_はAjaxによるリクエスト送信を意味する
  - remote_が付かないform_forは、普通にリクエストを送信

Ajax

- ここでAjaxの話を
- Ajax = Asynchronous JavaScript + XML
  - by Jesse James Garrett

Ajaxの要件

- 標準に準拠した表示
- 動的な表示・対話
- データ交換・操作
- 非同期なデータ取得
- これらすべての結合

Ajaxの要素技術

- XHTML/CSS
- Document Object Model
- XML/XSLT
- XMLHttpRequest
- JavaScript

Ajaxの特徴

- 画面遷移の制約からの解放
  - 画面の一部だけを更新
- リッチなUI
  - 視覚効果
  - ドラッグ&ドロップ

Boyakkyの場合

- ぼやきの書き込み後の処理
  - 古典的なWebアプリケーション
    - 画面遷移して全画面更新
  - Ajaxなアプリケーション
    - 新しい書き込みを追加するだけ

createアクション

- フォームの送信先の処理

```
def create
  @whine = Whine.new(params[:whine])
  @whine.user = current_user
  @whine.save
end
```


createのビュー

- HTMLの代りにJavaScriptを返す
  - ブラウザがJavaScriptを実行
- ただし、JavaScriptは書かない

RJS

- Ruby-Generated JavaScript
  - JavaScriptの自動生成
- Rubyで記述
  - DOM操作もRubyで

create.rjs

```
if @whine.valid?
  page.insert_html(:top, :whines,
                   :partial => "whine",
                   :object => @whine)
  page["whine_#{@whine.id}"].visual_effect(:highlight)
  page[:whine_body].value = ""
  page[:whine_body].focus
else
  page[:whine_body].visual_effect(:highlight,
                                  :startcolor => "#FF8888")
end
```


page変数

- JavaScriptGenerator
- page.insert_html()
  - 要素の内容にHTML断片を挿入
- page[要素のid]
  - 要素プロキシを返す

要素プロキシ

- 要素を操作するコードを生成

```
# 要素をハイライト表示
page[name].visual_effect(:highlight)
# 要素のvalue属性に空文字列を設定
page[name].value = ""
# 要素にフォーカスを移動
page[name].focus
```


生成されたコード

```
$("whine_7").visualEffect("highlight");
$("whine_body").value = "";
$("whine_body").focus();
```


インジケータ

- リクエストの処理中であることを表示
  - 視覚的フィードバックが重要
  - 処理しているのがわからないと、何度もクリックしてしまう

インジケータ画像

```
<%= image_tag ”indicator.gif”,
              :id => ”indicator”,
              :style => ”display: none” %>
```


ajaxload.info

- http://www.ajaxload.info/

:before/:complete

- Ajax処理前/処理後に実行する処理を指定

```
<% remote_form_for(@whine,
     :before => show_indicator,
     :complete => hide_indicator) do |f| %>
```


update_page

- JavaScriptを生成

```
def show_indicator
  return update_page { |page|
    page[:indicator].show
  }
end
```

- クライアントサイドで完結

link_to_remote

- remote_form_forと同様にAjaxリクエストを送信
- フォームではなくリンク

link_to_remoteの例

```
<%= link_to_remote(
      image_tag("bin.png", :alt => "削除"),
      :url => whine,
      :method => :delete,
      :confirm => "本当に削除しますか?",
      :before => show_indicator,
      :complete => hide_indicator) %>
```

- 「:method => :delete」?

REST

- ここでRESTの話を
- REST = Representational State Transfer
  - by Roy Fielding

Architectural style

- RESTはArchitectural style
- Architectural style
  - アーキテクチャのパターン
  - MVCとかクライアントサーバとか

Web

- WebはRESTの一実装形態
- これ以降はWebにおけるRESTの話

リソース

- RESTはリソース指向
- リソースはURIで識別

リソースの操作

- 基本はCRUD
- リクエストメソッドで識別

操作	メソッド
CREATE	POST
READ	GET
UPDATE	PUT
DELETE	DELETE

再びlink_to_remote

```
<%= link_to_remote(
      image_tag("bin.png", :alt => "削除"),
      :url => whine,
      :method => :delete,
```

- RESTful
  - URIでリソースを識別
  - メソッドで操作を識別

DELETE?

- ブラウザでDELETEを発行?
  - 普通はできない
- POSTで代用
  - パラメータで本来のメソッドを指定
    - _method=delete
    - PUTも同様

routes.rbの設定

```
# config/routes.rb
ActionController::Routing::Routes.draw do |map|
  map.resources :users
end
```


対応関係

メソッド	URI	アクション
GET	/users	index
POST	/users	create
GET	/users/1	show
PUT	/users/1	update
DELETE	/users/1	destroy

Webサービス

- RESTfulなWebサービス
  - リソースのRemix
- RESTfulでもAPIがばらばらだと使いにくい

AtomPub

- Atom Publishing Protocol
  - RFC5023
  - リソースを操作(CRUD)するためのプロトコル
  - データフォーマットとしてAtom(RFC4287)を採用

リソース

- コレクションリソース
  - フィード
- メンバリソース
  - エントリ

フィードの例

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xml:lang="ja" xmlns="http://www.w3.org/2005/Atom">
  <id>tag:localhost:whines</id>
  <link type="text/html" rel="alternate" href="http://localhost:3000"/>
  <title>Whines of all</title>
  <updated>2007-10-29T02:22:44+09:00</updated>
  <entry>
    <id>tag:localhost:3000:Whine10</id>
    <published>2007-10-29T02:22:44+09:00</published>
    <updated>2007-10-29T02:22:44+09:00</updated>
    <title>テスト</title>
    <author>
      <name>shugo</name>
    </author>
    <content type="text">テストです。</content>
  </entry>
  <entry>
  ...
  </entry>
</feed>
```


エントリの例

```
<?xml version="1.0" encoding="UTF-8"?>
<entry xml:lang="ja" xmlns="http://www.w3.org/2005/Atom">
  <id>tag:localhost:3000:Whine11</id>
  <published>2007-10-29T03:24:31+09:00</published>
  <updated>2007-10-29T03:24:31+09:00</updated>
  <link type="text/html" rel="alternate" href="http://localhost:3000/whines/11"/>
  <title>テスト</title>
  <author>
    <name>shugo</name>
  </author>
  <content type="text">テストです。</content>
  <link rel="edit" href="http://localhost:3000/whines/11.atom"/>
</entry>
```


フィードの取得

- 全ユーザのぼやき一覧
- 指定したユーザのぼやき一覧

リクエスト

```
GET /whines/ HTTP/1.1
Host: example.org
```


レスポンス

```
HTTP/1.1 200 OK
Content-Type: application/atom+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0" encoding="UTF-8"?>
<feed xml:lang="ja" xmlns="http://www.w3.org/2005/Atom">
  <id>tag:localhost:whines</id>
  <link type="text/html" rel="alternate" href="http://localhost:3000"/>
  <title>Whines of all</title>
  <updated>2007-10-29T02:22:44+09:00</updated>
  <entry>
    <id>tag:localhost:3000:Whine10</id>
    <published>2007-10-29T02:22:44+09:00</published>
    <updated>2007-10-29T02:22:44+09:00</updated>
    <title>テスト</title>
    <author>
      <name>shugo</name>
    </author>
    <content type="text">テストです。</content>
  </entry>
  <entry>
  ...
  </entry>
</feed>
```


indexアクション

```
def index
  if @user
    @whines = @user.whines.find(:all, :limit => 20)
  else
    @whines = Whine.find(:all,
                         :order => "created_at desc",
                         :limit => 20)
  end
  respond_to do |format|
    format.html
    format.atom
  end
end
```


respond_to

- Accept:やURIの拡張子によってビューを切替え
  - index.html.erb
  - index.atom.builder
- Content-Typeも設定

Builderテンプレート

- 拡張子は.builder
- ブロックによりXMLの入れ子構造を表現

index.atom.builder

```
atom_feed(:language => "ja") do |feed|
  name = @user.nil? ? "all" : @user.login
  feed.title("Whines of " + name)
  feed.updated(@whines.first.updated_at)
  for whine in @whines
    feed.entry(whine) do |entry|
      whine_atom_entry(entry, whine, :feed => true)
    end
  end
end
```


whine_atom_entry

```
def whine_atom_entry(entry, whine, options = {:feed => false})
  if !options[:feed]
    entry.id("tag:#{request.host_with_port}:#{whine.class}#{whine.id}")
    entry.published(whine.created_at.xmlschema)
    entry.updated(whine.updated_at.xmlschema)
    entry.link(:rel => 'alternate', :type => 'text/html',
               :href => whine_url(whine))
  end
  entry.title(truncate(whine.body, 20))
  entry.author do |author|
    author.name(whine.user.login)
  end
  entry.content(whine.body, :type => "text")
  entry.link(:rel => "edit", :href => whine_url(whine) + ".atom")
end
```


エントリの作成

- Basic認証
- リクエストボディでAtomエントリを送信

リクエスト

```
POST /whines/ HTTP/1.1
Host: example.org
Authorization: Basic c2h1Z286dGVzdA==
Content-Type: application/atom+xml;type=entry
Content-Length: nnn

<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title></title>
  <updated>2007-10-29T01:22:00Z</updated>
  <author><name>Shugo Maeda</name></author>
  <content>AtomPubのテスト</content>
</entry>
```


レスポンス

```
HTTP/1.1 201 Created
Date: Sun, 28 Oct 2007 18:24:31 GMT
Location: http://localhost:3000/whines/11
Content-Type: application/atom+xml;type=entry
Content-Length: 604

<?xml version="1.0" encoding="UTF-8"?>
<entry xml:lang="ja" xmlns="http://www.w3.org/2005/Atom">
  <id>tag:localhost:3000:Whine11</id>
  <published>2007-10-29T03:24:31+09:00</published>
  <updated>2007-10-29T03:24:31+09:00</updated>
  <link type="text/html" rel="alternate" href="http://localhost:3000/whines/11"/>
  <title>AtomPub&#12398;&#12486;&#12473;&#12488;</title>
  <author>
    <name>shugo</name>
  </author>
  <content type="text">AtomPub&#12398;&#12486;&#12473;&#12488;</content>
  <link rel="edit" href="http://localhost:3000/whines/11.atom"/>
</entry>
```


parse_atom

- beforeフィルタでAtomエントリをハッシュに変換

```
before_filter :parse_atom

def parse_atom
  if /^application¥/atom¥+xml/.match(request.env["CONTENT_TYPE"])
    xml = REXML::Document.new(request.raw_post)
    params[:whine] = {
      :body => xml.elements["entry/content"].text
    }
  end
  return true
end
```


createアクション

```
def create
  @whine = Whine.new(params[:whine])
  @whine.user = current_user
  @whine.save
  respond_to do |format|
    format.js
    format.atom do
      if @whine.valid?
        headers["Content-Type"] = "application/atom+xml;type=entry"
        headers["Location"] = whine_url(@whine) + ".atom"
        render :action => "show", :status => 201
      else
        headers["Content-Type"] = "text/plain"
        render :text => "validation failed¥n", :status => 400
      end
    end
  end
end
```


show.atom.builder

```
xml.instruct!
xml.entry("xml:lang" => "ja",
          "xmlns" => "http://www.w3.org/2005/Atom") do |entry|
  whine_atom_entry(entry, @whine)
end
```


curlによるテスト

```
$ curl -D - -u shugo:test --basic -s -X POST ¥
    -H 'Accept: application/atom+xml' ¥
    -H 'Content-Type: application/atom+xml;type=entry' ¥
    --data-binary @entry.atom ¥
    http://localhost:3000/whines/
```


まとめ

- RJSでJavaScriptを書かずにAjaxを実現
- map.resourcesでRESTfulに
- WebサービスはAtomPubで

参考文献

- Railsによるアジャイルwebアプリケーション開発
  - Dave Thomasほか
  - ISBN: 978-4-274-06696-2
- Ajax on Rails
  - Scott Raymond
  - ISBN: 978-4-87311-332-6

参考サイト

- RJS を使ってみる
  - http://jp.rubyist.net/magazine/?0014-RubyOnRails
- REST入門
  - http://yohei-y.blogspot.com/2005/04/rest_23.html

参考サイト(2)

- Atom Publishing Protocol 日本語訳
  - http://www.ricoh.co.jp/src/rd/webtech/rfc5023_ja.html

本日の講演資料

- スライドPDF
  - http://shugo.net/tmp/web2.0-on-rails.pdf
- ソースコード
  - http://shugo.net/tmp/boyakky.zip

おわり

ご静聴ありがとうございました