エロサイトを支える技術

Ruby on Rails + AngularJS

今日はWWDCのkeynoteが
あるので早く帰って寝たい

自己紹介

- twitter : @kenchan0130_aki
- github : kenchan0130
- 脱エンジニア志望エンジニア

Pornfolio

開発の経緯

エロサイトって無駄な広告が多い

エロサイトって無駄な広告が多い

わざわざ自分で良質の動画を探すのが面倒

エロサイトって無駄な広告が多い

わざわざ自分で良質の動画を探すのが面倒

わざわざ自分で良質の動画を探すのが面倒

エロサイトって無駄な広告が多い

わざわざ自分で良質の動画を探すのが面倒

わざわざ自分で良質の動画を探すのが面倒

スマホ最適化されている動画まとめサイトがない

エロサイトって無駄な広告が多い

わざわざ自分で良質の動画を探すのが面倒

わざわざ自分で良質の動画を探すのが面倒

スマホ最適化されている動画まとめサイトがない

もっとスタイリッシュで見やすいサイト無いのか

じゃあ作るか

開発体制

- バックエンド 1人　→ Rails担当
- フロントエンド 1人 → Angular担当
- デザイン 1人　→ GitとSass使ってね☆

開発体制

- バックエンド 1人　→ Rails担当
- フロントエンド 1人 → Angular担当
- デザイン 1人　→ GitとSass使ってね☆

Rails周りを含め

話していければと

大きく分けて

・Ruby on Railsについて（導入まで）

・実際にPornfolioでは何をどうしてるのか

・Ruby on Railsについて（導入まで）

・実際にPornfolioでは何をどうしてるのか

そもそも
Ruby on Rails
とは

Ruby on Rails

- Ruby環境で使用できる、
  Webアプリケーションフレームワーク
- 開発がとにかく「楽」「速い」
- 現在の最新版は 4.1.1 (release
  2014/05/07)
- 同日3系も 3.2.18 もリリースされました
- 3系もまだメンテしていくみたいです

その他Rubyのフレームワーク

- Sinatra (Amon2的なやつ)
- Ramaze
  (map '/' 的なのをクラスに書いてルーティングする)
- Padrino (Sinatraを強化したやつ)
- Pakyow (Viewファーストらしい 知らない・・・)
- Halcyon (JSONベースらしい 知らない・・・)

Railsには

重要な設計哲学がある

設計哲学

- DRY (Don’t Repeat Yourself)
  - 同じ記述を繰り返さない
- CoC (Convention over Configuration)
  - 設定よりも規約

DRY (Don’t Repeat Yourself)

```
1 package GesuGirls::DB::Schema;
2 use strict;
3 use warnings;
4 use Teng::Schema::Declare;
5
6 table {
7     name 'users';
8     pk 'id';
9     columns qw(
10        id
11        nickname
12        password
13        email
14        photo
15        comment
16        sex
17        created_at
18        updated_at
19    );
20 };
21
22 table {
23    name 'friends';
24    pk qw(from_user_id to_user_id);
25    columns qw(
26        from_user_id
27        to_user_id
28    )
29 };
30
```

例えば

- データベースのスキーマ定義を設定ファイルとして記述する必要がない
- データベースにテーブルを作成するだけでRails側が勝手に認識

CoC (Convention over Configuration)

モデル	app/models/user.rb
ビュー	app/views/users/*
コントローラー	app/controllers/ user_controller.rb

例えば

- usersのテーブルを読み込むにはUserクラスを利用する
- users（複数形）とUser（単数形）が規約により結ばれている

Railsに必要な環境

環境

- Ruby
  - [3.1以下] そんなものは捨ててしまってOK
  - [3.2.x] Ruby 1.8.6以上 （Ruby 1.9.3推奨）
  - [4.x] Ruby 1.9.3以上（Ruby 2.0.0推奨）
- Database
- Node.js 安定版 (therubyracerで代用化)

Rails4での
トレンド/変更点

トレンド

- HTML5対応
  - ヘルパーもHTML5対応
- RESTfulなインターフェイス
  - 統一感のあるURLを設計できる
- 控えめなJavaScript
  - HTMLの中にJSの直書きは最低限にする

Rails3からの変更点

- ActiveRecord 4.xの採用
  - 色々変わってます
  - 特に、whereがActiveRecord::Relationを返す様になった
- テストディレクトリの変更
- concernsディレクトリの導入
- Strong Parametersの導入 （多分いちばん厄介）
- Jbuilderの導入（Javaのアレではないです）
- [4.1.0] Action Pack Variantsの導入
  - しかし、日本では文字コードの問題で jpmobile がいいみたい

Rails3からの変更点

- ActiveRecord 4.xの採用
  - 色々変わってます
  - 特に、whereがActiveRecord::Rel
- テストディレクトリの変更
- concerns

などなど

sの導入 （多分いちばん厄介）

derの導入（Javaのアレではないです）

- [4.1.0] Action Pack Variantsの導入
  - しかし、日本では文字コードの問題で jpmobile がいいみたい

とりあえず

Railsアプリを作成してみる

```
~ % ruby -v
ruby 2.1.1p76 (2014-02-24 revision 45161) [i686-linux]
~ % gem install rails --no-doc --no-ri
Fetching: minitest-5.3.4.gem (100%)
Successfully installed minitest-5.3.4
Fetching: tzinfo-1.2.1.gem (100%)
Successfully installed tzinfo-1.2.1
Fetching: i18n-0.6.9.gem (100%)
Successfully installed i18n-0.6.9
Fetching: activesupport-4.1.1.gem (100%)
Successfully installed activesupport-4.1.1
Fetching: erubis-2.7.0.gem (100%)
```

gemでrailsの最新版をインストール

```
~ % rails new railbook -d mysql
      create
      create  README.rdoc
      create  Rakefile
      create  config.ru
      create  .gitignore
      create  Gemfile
      create  app
      create  app/assets/javascripts/application.js
      create  app/assets/stylesheets/application.css
      create  app/controllers/application_controller.rb
      create  app/helpers/application_helper.rb
      create  app/views/layouts/application.html.erb
      create  app/assets/images/.keep
      create  app/mailers/.keep
      create  app/models/.keep
      create  app/controllers/concerns/.keep
      create  app/models/concerns/.keep
```

rails newでアプリケーションを作成

```
railbook % rails s
=> Booting WEBrick
=> Rails 4.1.1 application starting in development on http://0.0.0.0:3000
=> Run `rails server -h` for more startup options
=> Notice: server is listening on all interfaces (0.0.0.0). Consider using 127.0.0.1 (--binding option)
=> Ctrl-C to shutdown server
[2014-06-02 06:00:57] INFO  WEBrick 1.3.1
[2014-06-02 06:00:57] INFO  ruby 2.1.1 (2014-02-24) [i686-linux]
[2014-06-02 06:00:57] INFO  WEBrick::HTTPServer#start: pid=2515 port=3000
^C[2014-06-02 06:01:00] INFO  going to shutdown ...
[2014-06-02 06:01:00] INFO  WEBrick::HTTPServer#start done.
```

rails serverで簡易サーバー立ち上げ

これでもう君もrails厨

まぁこれだけだとアレなので...

若干ファイル構成について解説

/railbook	…… アプリケーションルート
├── /app	…… アプリケーションのメインフォルダ
│ ├── /assets	…… アセット（JavaScript、スタイルシート、画像などのリソース）
│ │ ├── /images	…… 画像ファイル
│ │ ├── /javascripts	…… JavaScript（CoffeeScript）ライブラリ
│ │ └── /stylesheets	…… CSS（SCSS）ライブラリ
│ ├── /controllers	…… コントローラクラス
│ │ ├── /concerns	…… コントローラ共通モジュール
│ │ └── application_controller.rb	…… アプリケーション共通のコントローラ
│ ├── /helpers	…… ビューヘルパー
│ │ └── application_helper.rb	…… アプリケーション共通のビューヘルパー
│ ├── /mailers	…… ActionMailer実装クラス
│ ├── /models	…… モデルクラス
│ │ └── /concerns	…… モデル共通モジュール
│ └── /views	…… ビュースクリプト
│ └── /layouts	…… レイアウト
│ └── application.html.erb	…… アプリケーション共通のレイアウト
├── /bin	…… コード生成や開発サーバの起動に利用するヘルパースクリプト
├── /config	…… アプリケーション本体、ルーティングなどの設定情報
│ ├── /environments	…… 環境単位の設定ファイル
│ ├── /initializers	…… 初期化ファイル
│ └── /locales	…… 辞書ファイル
├── /db	…… データベース本体やスキーマ情報、マイグレーションファイルなど
├── /lib	…… 自作のライブラリなど
│ ├── /assets	…… 自作ライブラリに関連するアセット
│ └── /tasks	…… タスク関連
├── /log	…… ログの出力先
├── /public	…… 公開フォルダ
├── /test	…… テストスクリプトなど
├── /tmp	…… 一時ファイル
├── /vendor	…… サードパーティのコード
│ └── /assets	…… サードパーティから提供されるアセット
├── config.ru	…… アプリケーションのエントリポイント
├── Gemfile	…… 必要なgemファイルを定義
├── Rakefile	…… ターミナルから実行可能なジョブ
└── README.rdoc	…… readmeファイル

http://www.buildinsider.net/web/rubyonrails4/0201

/railbook	…… アプリケーションルート
├── /app	…… アプリケーションのメインフォルダ
│　　├── /assets	…… アセット（JavaScript、スタイルシート、画像などのリソース）
│　　│　　├── /images	…… 画像ファイル
│　　│　　├── /javascripts	…… JavaScript（CoffeeScript）ライブラリ
│　　│　　└── /stylesheets	…… CSS（SCSS）ライブラリ
│　　├── /controllers	…… コントローラクラス
│　　│　　├── /concerns	…… コントローラ共通モジュール
│　　│　　└── application_controller.rb	…… アプリケーション共通のコントローラ
│　　├── /helpers	…… ビューヘルパー
│　　│　　└── application_helper.rb	…… アプリケーション共通のビューヘルパー
│　　├── /mailers	…… ActionMailer実装クラス
│　　├── /models	…… モデルクラス
│　　│　　└── /concerns	…… モデル共通モジュール
│　　└── /views	…… ビュースクリプト
│　　│　　└── /layouts	…… レイアウト
│　　│　　└── application.html.erb	…… アプリケーション共通のレイアウト

├── /bin	…… コード生成や開発サーバの起動に利用するヘルパースクリプト
├── /config	…… アプリケーション本体、ルーティングなどの設定情報
│　　├── /environments	…… 環境単位の設定ファイル
│　　├── /initializers	…… 初期化ファイル
│　　└── /locales	…… 辞書ファイル
├── /db	…… データベース本体やスキーマ情報、マイグレーションファイルなど
├── /lib	…… 自作のライブラリなど
│　　├── /assets	…… 自作ライブラリに関連するアセット
│　　└── /tasks	…… タスク関連
├── /log	…… ログの出力先
├── /public	…… 公開フォルダ
├── /test	…… テストスクリプトなど
├── /tmp	…… 一時ファイル
├── /vendor	…… サードパーティのコード
│　　└── /assets	…… サードパーティから提供されるアセット
├── config.ru	…… アプリケーションのエントリポイント
├── Gemfile	…… 必要なgemファイルを定義
├── Rakefile	…… ターミナルから実行可能なジョブ
└── README.rdoc	…… readmeファイル

ここで一息

質問とかあれば

・Ruby on Railsについて（導入まで）

・実際Pornfolioでは何をどうしているか

ぶち当たった問題

↓

解決方法

の流れでお話できればと思ってます

サーバー構成

- アプリケーションサーバー 1台
  - nginx + unicorn
- データベースサーバー 1台
  - MySQL 5.5

よし、サーバー借りよう

“問題発生”

アダルトサイトを載せていい
レンタルサーバーがほとんどない。。。

- VPSを提供している日本の会社・サービスはほとんどない
- あってもスペックがひどかったりする

解決策

- 「GMOクラウド」を選択
  - アダルトOKかつ低コストで始めることが可能
  - 申し込みをして使える様になるまで2日くらい必要
- AWSのEC2もアダルトは問題ない
  - スケールが楽であるが、サービスが回っていない間の
    初期投資額が大きくなってしまう
- ちなみに、DBサーバーはさくらVPSのSSDプランを選択

アプリケーション構成

- クライアントサイドにAngularJSを使用
- RailsはJSONを吐くAPIサーバー
- ViewとControllerの密結合を分離することができる

よし、

$ rails g controller hoges

だ！

“問題発生”

- Controllerをgenerateすると勝手にviewができてしまう
- js (coffee)とかcss (scss)とかいらない
- APIサーバーだからerbとかhamlをrenderしたくない

解決策

- [案1] rails-apiというgemを入れることでviewを作らなくて済む
  - rails new railbook → rails-api new railbook
- [案2] jbuilderを使ってjsonテンプレートに値を打ち込むようにする

解決策

- [案2]を採用
  - jbuilder使ってみたかった興味本位
- 以下のコマンドでjbuilderを作成して、いらないhelperとassetsを作らいないようにできる
  - routes.rbに「resources :tests, format: ‘json’」を書けばjbuilderでrenderしてくれる

```
railbook % rails g controller tests -e jbuilder --no-assets --no-helper
      create  app/controllers/tests_controller.rb
      invoke  jbuilder
  Plural version of the model detected, using singularized version. Override with --force-plural.
      create    app/views/tests
      create    app/views/tests/index.json.jbuilder
      create    app/views/tests/show.json.jbuilder
      invoke  test_unit
      create    test/controllers/test_controller_test.rb
```

データベース設計

- modelをgenerateするとmigrationファイルもできるので、じゃんじゃかDDL書かける

中間テーブルも楽々!

楽々・・・・orz

“問題発生”

- 中間テーブルなのに勝手にauto incrementな主キーidを作成してしまう
- ついでにhogehoge_idとかunsignedにしたい

解決策

- create_tableメソッドにid: falseのハッシュを与えるとid作らないで済む

- activerecord-mysql-unsigned というgemを入れることでunsignedが使えるようになる

- ちなみに、railsの規則に則ると、例えば、usersテーブルとpostsテーブルの中間テーブル名は posts_users となる

```
def change
  create_table :post_content_details_xvideos, :id => false do |t|
    t.integer :post_content_detail_id, :limit => 8, :null => false, unsigned: true
    t.integer :xvideos_id, :limit => 8, :null => false, unsigned: true
  end
  add_index :post_content_details_xvideos, :post_content_detail_id
  add_index :post_content_details_xvideos, :xvideos_id
end
```

xvideosやアゲサゲのデータ取得

- APIなんかは提供されてるわけもなく、
  スクレイピングを決行

- スクレイピングには nokogiri というgemを使用

スクレイピングでサムネイ
ルとvideoのid取り放題や!

“懸念事項発生”

- まぁ普通にサイト叩いているので、レスポンスが相手のサーバー次第（少なくとも4、５秒以上）はかかってしまう

- こちらの処理も含めると6秒かかってしまう場合も

- 並列に回したらIP BANされる可能性がある

解決策

- sidekiq というgemを使用する
  - これはtaskをredisに保存して、裏で回してくれるやつ
  - app/workers/hoge_woker.rbを作成して performメソッドの中に書いたものが HogeWorker.perform_async()でtaskがstackされる
  - 動画のURLを入れた瞬間にこの処理を回すようにして、ユーザーを待たせないようにする

さあdeployだ！

- unicorn を使ってプロセス動かそうかな
- capistrano というのは勝手にdeployしてくれる優れものらしい
- githubのリポジトリを指定しておけば後はよしなにやってくれる

$ cap development deploy

“問題発生”

- githubのリポジトリのrootにrailsのrootが無いとちゃんと動かない
- 色々path変えたりしたけど、gemfileのpathが異なったりと2日くらいを無駄にしてしまった

解決策

- 手動deployしました
- unicornのrakefileとか作って
  rake unicorn:start ENV=productuon
  したら動くようにしました
- ついでに sidekiq のrakefileも作って動くようにしました

“再び問題発生”

- unicornをstartさせると何故か
  redisのオブジェクトねーぞ馬鹿野郎
  ってエラーが出てくる
- 4つの子ワーカーを立ち上げる設定していたため、
  親でinitializeしたときに作ったRedisの設定とか
  が子からは参照できない
- 公式リファレンスには何も書いてない・・・辛い

解決策

```
# graceful restart用の設定 (Masterプロセスがシームレスに切り替わる)
before_fork do |server, worker|
  old_pid = "#{server.config[:pid]}.oldbin"
  # oldプロセスがいたら終了する
  if File.exists?(old_pid) && old_pid != server.pid
    begin
      sig = (worker.nr + 1) >= server.worker_processes ? :QUIT : :TTOU
      Process.kill(sig, File.read(old_pid).to_i)
    rescue Errno::ENOENT, Errno::ESRCH
    end
  end

  Redis.current.quit if defined?(Redis)
end


after_fork do |server, worker|
  defined?(ActiveRecord::Base) and ActiveRecord::Base.establish_connection
  Redis.current = Redis.new(:host => redis_config[:host], :port => redis_config[:port]) if defined?(Redis)
  Sidekiq.configure_client do |config|
    config.redis = { :url => "redis://#{redis_config['host']}:#{redis_config['port']}",
                     :namespace => "sidekiq" }
  end
end
```

解決策

```
# graceful restart用の設定 (Masterプロセスがシームレスに切り替わる)
before_fork do |server, worker|
  old_pid = "#{server.config[:pid]}.oldbin"
  # oldプロセスがいたら終了する
  if File.exists?(old_pid) && old_pid != server.pid
    begin
      sig = (worker.nr + 1) >= server.worker_processes ? :QUIT : :TTOU
      Process.kill(sig, File.read(old_pid).to_i)
    rescue Errno::ENOENT, Errno::ESRCH
    end
  end

  Redis.current.quit if defined?(Redis)
end


after_fork do |server, worker|
  defined?(ActiveRecord::Base) and ActiveRecord::Base.establish_connection
  Redis.current = Redis.new(:host => redis_config[:host], :port => redis_config[:port]) if defined?(Redis)
  Sidekiq.configure_client do |config|
    config.redis = { :url => "redis://#{redis_config['host']}:#{redis_config['port']}",
                     :namespace => "sidekiq" }
  end
end
```

forkする前にredisを一時的に止める

解決策

```
# graceful restart用の設定 (Masterプロセスがシームレスに切り替わる)
before_fork do |server, worker|
  old_pid = "#{server.config[:pid]}.oldbin"
  # oldプロセスがいたら終了する
  if File.exists?(old_pid) && old_pid != server.pid
    begin
      sig = (worker.nr + 1) >= server.worker_processes ? :QUIT : :TTOU
      Process.kill(sig, File.read(old_pid).to_i)
    rescue Errno::ENOENT, Errno::ESRCH
    end
  end

  Redis.current.quit if defined?(Redis)
end


after_fork do |server, worker|
  defined?(ActiveRecord::Base) and ActiveRecord::Base.establish_connection
  Redis.current = Redis.new(:host => redis_config[:host], :port => redis_config[:port]) if defined?(Redis)
  Sidekiq.configure_client do |config|
    config.redis = { :url => "redis://#{redis_config['host']}:#{redis_config['port']}",
                     :namespace => "sidekiq" }
  end
end
```

forkしたらもっかいredisオブジェクトを生成する

よし！あとはwebマスターツー
ル使ってindex貼ってコンテンツ
ごとにSEOとか強化していこう

“問題発生”

- AngularJSはGoogleさんが作ったんだからajaxとかよしなにやってくれると思った私が馬鹿だった
- {{app.title}} みたいに、angularのままindex貼りやがった

解決策

- ドメインのあとに「!#」をつけるとGoogleのロボットは「_escaped_fragment_」に変換してくれる
  - 例えば
    pornfolio.jp/!#posts/10
    は
    pornfolio.jp/_escaped_fragment_=posts/10
- _escaped_fragment_があったらPhantom.jsでpornfolio.jp/!#posts/10を叩いた結果をGoogleさんに返す様にする

その他Tips

- 広告はdocument.writeする糞仕様なので、
  ソースたどって取得しているimgタグをrails側で取りに行く
- nginx側でモバイル判定して呼び出すhtmlを変えている
- コントローラーやモデルで使うモジュールは concerns に記載
- ランキング処理などの定期的に実行するもには whenever というgemを使用
- BULK INSERTには activerecord-import というgemを使用
- ドキュメント作成には apipie-rails というgemを使用
- 初期値を入れるために seed-fu というgemを使用

まとめ

- Railsはすぐに開発ができ、gemもたくさん揃っているので、開発速度が速い
- いろんな問題にぶち当たってるものは大体他の人もぶち当たってる (stack overflowなんかにある)
- できないと思ったら自分で書いてしまったほうが速いことが結構ある
- 今回は触れてないが、メンテナンス・チューニングが大変

おつかれさまでした