毎日の開発に役立つ Railsプラグイン づくりの秘訣

@a_matsuda

最初にお知らせ

- 3つ

RubyKaigi 2019

Apr 18th (Thu) - 20th (Sat)

Fukuoka International Congress Center, Fukuoka, Japan

Keynote Speakers:

Yukihiro "Matz" Matsumoto

nagachika

Jeremy Evans

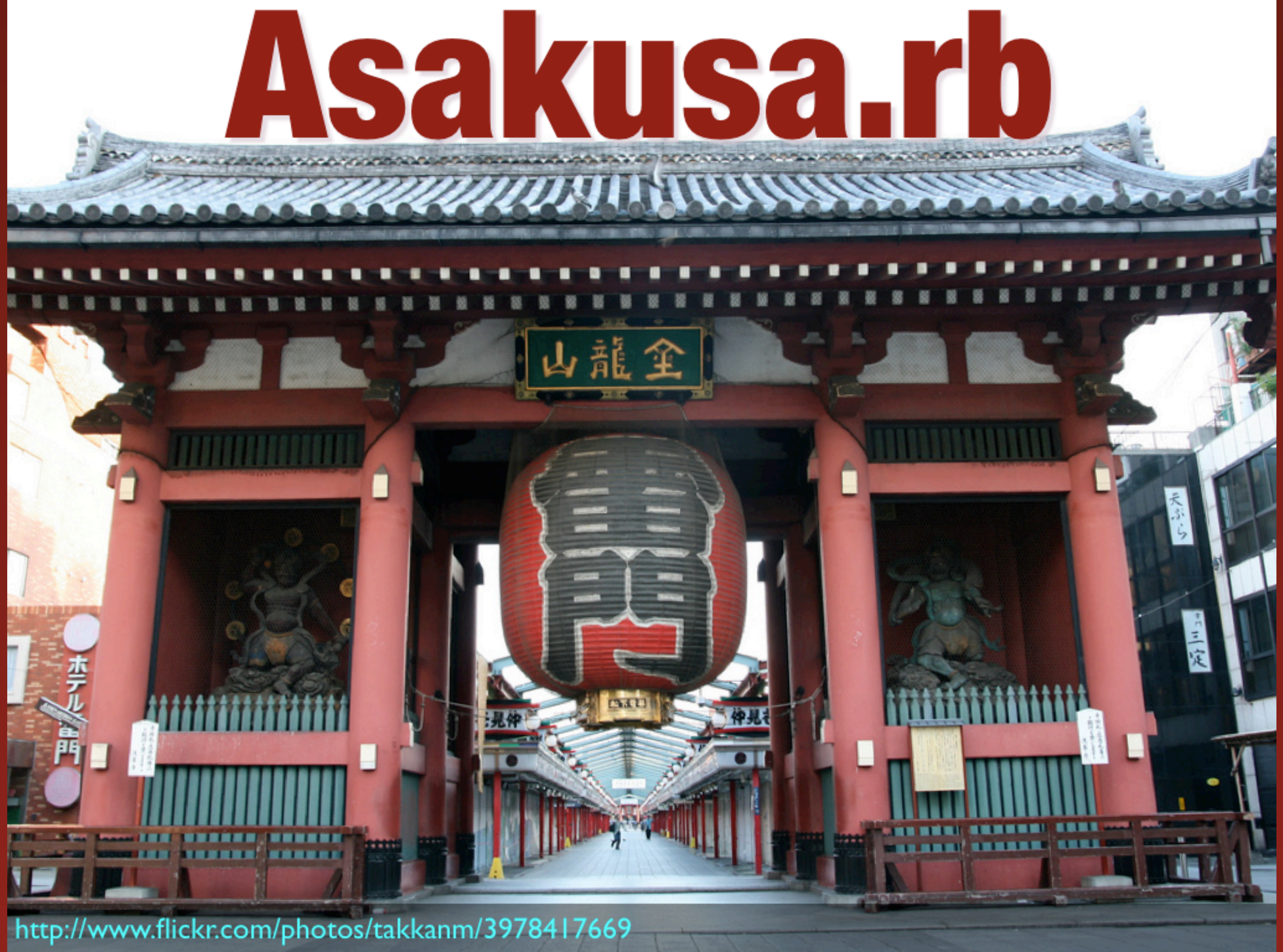
RubyKaigiのLTのプロポーザル受付中！

明後日まで！

RejectKaigiのプロポーザルも募集中！

今月いっぱい！

Asakusa.rb



http://www.flickr.com/photos/takkanm/3978417669

Asakusa.rb 花見やります！

来週の土曜日！

今日話すこと

- 比較的最近作ったgemの話
- あるいは
- 人はなぜgemを作るのか

今日話すこと

- それから、

今日話すこと

- テストコードが開発をドライブする話

今日話さないこと

- action_args
- active_decorator
- database_rewinder
- gem_src
- i18n_generators
- jb
- kaminari
- motorhead
- rfd
- stateful_enum
- string_template
- traceroute
- ...

Question

何か実用レベルのgemを自分で作って
リリースしてみたことがある人 ✋

いま手を挙げてない人たちに
言いたい

Why Don't You Create Your Own Gems?

なぜgemを作らないのか？

- または、人はなぜgemを作るのか？

僕の場合は、ほぼ100％

「現場の問題を解決するため」

Case #1

現場の問題

- テストはもちろん書かなくっちゃあいけない
- でもE2Eテスト書くのめんどい
- 項目が増えてきたりするとつらい

きっかけ

きっかけ

きっかけ

- 新規プロジェクトを始めるにあたって、E2Eテストとかあんまり書いたことないっていう若いメンバーも居た
- 誰でもテストがサクサクっと書けるといいなぁ、と思って作ってみた

heavens_door

このgemを作るにあたって気をつけたこと

- とにかく単機能にする

ブラウザの操作を記録してCapybaraのシナリオを作るプラグイン

- すごく直感的

使い方はREADMEに貼ったアニメgifを見ればすぐわかるので、説明は省略

heavens_door

- https://github.com/amatsuda/heavens_door

ところで、

ご記憶の方も何名かいらっしゃった
ようですが、

実は、このコンセプトは全然
新しくなくて、

はるか昔にhocus_pocusで一度やったことの焼き直し

Case #2

hocus_pocus

- 歴史の狭間に埋もれたオーパーツ
- 超多機能なブラウザ内統合開発環境みたいなものを目指した

これは、何か解決したい問題があったというよりは、
作れそうな気がしたから作り始めてみた、という感じ

hocus_pocusの構造

- main_appにmountされたhocus_pocus エンジンがさらに4つのサブmountable engine をmountしてるという無駄に複雑な構成
- 当時世界初だったはずだし、たぶん未だに そんなengineは他に無いのではないか

hocus_pocusの思想

- "resource missing" driven development
- ページ内にWikiNameを書いたら、新しいWikiページへのリンクになってくあの感じ

とか2019年に言っても、あんまり
伝わらなそうなので

およそ7年ぶりにDEMO

7年前

Sapporo RubyKaigi 2012

We Code.

hocus_pocus

- https://github.com/amatsuda/hocus_pocus

9年前に作り始めたやつ

- まったく流行らなかった
- というか完成すらしなかった

hocus_pocusのその後

- 壮大な構想は色々あったけど、
  やる気が持続せずにdiscon

もうちょい完成度が高くなったらドキュメント
とかもちゃんと書こう、と思ってた

- けど、そんな日は来なかった

ここから得た教訓

- 現場の問題ドリブンじゃない趣味の開発はモチベーションの維持が難しい
- 小さく作って早めにリリース、がやっぱり大事
- ドキュメンテーション大事
- そもそも使い方が難しすぎるツールは使ってもらえない

小さく作って早めにリリース、がやっぱり大事

- 思いついたアイデアはすぐに実装できるサイズじゃないと、なんだったか忘れる

ドキュメンテーション

- 大作なgemはドキュメントを書く障壁が高い
- そういう意味でも機能は少ないほうが良い

そもそも使い方が難しすぎるツールは使ってもらえない

- コンセプトが特殊だったりすると、伝わらずに終わる

hocus_pocusでの失敗を活かして heavens_doorでやってみたこと

アニメgif

アニメgifすごい

「言葉」でなく「心」で
理解できる！

でも、このメソッドが適用できるRubyのライブラリはそんなに多くないかも？

- 実際、自分が作った他のライブラリでこのメソッドが活用できるかどうか考えてみたけど、実際なかなかチャンスはなさそう
- てなわけで、RubyのライブラリよりもJSとかのライブラリの方が⭐を集めやすそう、っていう話はあるんだよなぁ

ところで、このブラウザ内でアプリを開発していくというコンセプト

- 2019年になって、あの”conductor”が盛り上がりを見せる機運
- これからまだまだ可能性のある分野なのでは

そこで、ブラウザで何かするやつを

もう1つ

Case #3

現場の問題

- ER図が見たい！

現場でER図見たいですよね？

- たとえばちょっとエンタープライズっぽい案件に途中から参画した時とか

そこでerd

- Railsアプリにmountして使うengine
- 親アプリのER図をブラウザ内に表示
- ついでに操作

操作？

- create tableとかalter tableとか
- migrationを作成、実行
- migrationの書き方を覚えられない/思い出せない人におすすめ
- migration が「言葉」でなく「心」で(ry

DEMO

Status of erd

- ぜんぜん未完成
- UIがいろいろビミョー
- もっとブラッシュアップしたい気持ちはあるけどJS書くのしんどい
- coffeeで雑に書いてdecafしたコードがもはやメンテ不能レベル
- 誰かJSが得意な方、助けてください！！！！(切実)

erd

- https://github.com/amatsuda/erd

さて、話は変わって、ブラウザ上でエンプラっぽい図を表示するengineをもう1つ

Case #4

現場の問題

- 今度は画面遷移図が見たい！

これも現場の切実な問題

- routes.rb を見ればリソースの一覧はわかる
- でもつながりは見えない

やっぱ画面遷移図

コンセプト

- 画面遷移図って昔はみんな手描きしてたものだけど、考えてみたらこれって生成できるんじゃないか？

roundabout

DEMO

やってること

- E2Eテスト内で実行された画面遷移を全て記録
- その結果を図示

これが何を表しているのか

- 実際に実装されててテストされている画面遷移を記録
- ただの routes.rb の図示とは違う

テストがちゃんと書かれているかの確認になる

- ある意味カバレッジの表示の一種のようなものと言えるかも

このgemのよさ

- 書いたテストが形を変えた資産になるのがよい

次はもう1つ、テスト資産を活かすやつ

をご紹介

Case #5

現場の問題

- Railsアップグレードでデグレしたくない

still_life

still_lifeでできること

- テストを走らせた際に、テスト中にrenderされたHTMLを全部ファイルに書き出す

すごい力技なプラグイン

目的

- アプリケーションコードの書き換え前後に、ユーザーに届けているものが変化していないことを確認できる

特徴

- 画面を踏んで回るだけで良くて、アサーションを書かなくて良いので楽チン

どういう時に使うのか

- 特にRailsのアップグレードが想定用途
- 一般的に各種リファクタリングの支援になるはず

そもそもの元ネタ

- 2011年に、クックパッドのRailsのバージョンを上げる時にセコンさんが書き捨てたスクリプト

https://speakerdeck.com/a_matsuda/the-recipe-for-the-worlds-largest-rails-monolith?slide=129

We do something like this

```
RSpec.configure do |config|
  config.include(
    Module.new do
      def save_response_body
        target = defined?(response) ? response : page
        if target.body.present?
          pathname = Rails.root.join("tmp/SOME_DIRECTORY/
#{example.location.gsub(?:, ?-)}.html")
          pathname.parent.mkpath
          pathname.open('w') {|file| file.puts target.body }
        end
      end
    end
  )
  config.after(type: :controller) { save_response_body }
  config.after(type: :request) { save_response_body }
  config.after(type: :feature) { save_response_body }
end
```

cookpad

まさに現場の問題を的確に解決する職人技

gem化にあたって、以下の全面サポートを追加

- Capybara 3
- RSpec 3
- test-unit
- minitest

皆さんの現場でももうじき Rails 5.2 -> 6.0 のアップグレード業が始まると思いますが、

- かなり有用かもしれないので、どうぞご活用ください

ところで、このライブラリの主な能力は
アプリをこじ開けて動作を記録するスタンド

- この能力って、これこそ
  "heavens_door" じゃあねえか！

と思ったけど、その名前のgemは既に取られてしまっていたのであった

名前重要

- gemの名前つけるの難しい……

これを8年越しでgem化したきっかけ

- 自作テンプレートエンジンの動作検証のために欲しくなった
- 某インターネット家計簿のRailsアップグレードの際にこれの存在を思い出した

Money Forward

「自作テンプレートエンジン」

Case #6

Himl

Himl

- きっかけは、永和システムマネジメントで毎月やってるOSSパッチ会

のあとの飲み会

永和システムマネジメント

kamipoさんの愚痴

- kamipo: 会社でHaml使ってるけど、HamlとかSlimとか文法が謎すぎて覚えられへんわ。ERBで良くない？
- me: いやー、Haml超いいっすよ。ERBに戻るとかムリムリ。

しかし、あらためて、Hamlの良さってなんだろう？

- 確かに、あの記号たちを駆使した文法自体はHamlの良さの本質ではないかも
- Hamlの良さは、タグを機械が閉じてくれるおかげで、コンパイルされたHTMLがvalidなことが保証されるところ

💡 じゃあ、Hamlの良さをERBに注入してやれば良いのでは？

よさそう、かもしれない

ので作ってみた

Himl

- https://github.com/amatsuda/himl

今回の現場は…

- 神田のとある居酒屋

Himlの開発を通じてわかったこと

居酒屋だって現場なんや！

まとめ

人はなぜgemを作るのか？

- 問題を解くため

問題はたくさん現場に転がってる

こういう問題の解き方もいいんじゃないでしょうか？

- なかなか楽しいですよ

Why Don't You Create Your Own Gems?

- gem作ろーぜ！

end

me

- Name: Akira Matsuda
- Twitter: @a_matsuda
- GitHub: amatsuda