5分でつくる！

Funicular on Railsチャットアプリ

hasumikin

仙台Ruby会議03

2026-11-21

self.inspect

- Hitoshi HASUMI
  - @hasumikin (GitHub & Twitter)
- PicoRubyの作者
- mrubyとmruby/cのコミッタ
- ANDPAD, inc.



今日の話

Part 1. 5分でつくるチャットアプリ

Part 2. Funicularの特長たくさん

フニクリ・フニクラ🎶



https://ja.wikipedia.org/wiki/ヴェスヴィオ

Part 1

5分でつくるチャットアプリ

Demo：チャットアプリ

いまやったこと

```
rails new chat_demo
echo 'gem "funicular"' >> Gemfile
bundle install
bin/rails funicular:install
npm install                          # jsdom for client tests
bin/rails generate funicular:chat
bin/rails db:migrate
bin/rails funicular:compile
bin/rails server
```

JavaScriptはゼロ。JavaScriptのビルドツールもゼロ

ジェネレータがつくったもの（抜粋）

```
app/models/funicular_chat_message.rb                # Rails
app/controllers/funicular_chat_controller.rb        # Rails
app/channels/funicular_chat_channel.rb              # Rails
app/views/funicular_chat/show.html.erb              # Rails
------------------------------------------------------------
app/funicular/components/funicular_chat_component.rb # UI
app/funicular/initializer.rb                         # UI
test/funicular/client/funicular_chat_component_picotest.rb
```

- 保存と配信はRailsの仕事
- 状態を持つ画面はFunicularの仕事

コンポーネント（抜粋）

```ruby
class FunicularChatComponent < Funicular::Component
  def initialize_state
    { name: "Rails developer", body: "", messages: [], connected: false }
  end

  def component_mounted
    Funicular::HTTP.get("/funicular_chat/messages") do |response|
      patch(messages: response.data)
    end

    @consumer = Funicular::Cable.create_consumer("/cable")

    @consumer.subscriptions.create(channel: "FunicularChatChannel") do |message|
      patch(messages: state[:messages] + [message])
    end
  end

  def render
    div(class: styles.shell) do
      (...)
    end
  end
end
```

Part 2

Funicularの特長たくさん

Funicular とは

- SPAフレームワーク
  - Virtual DOMと差分更新（ReactJSみたいなやつ）
  - Rubyクラスベースのコンポーネント
  - JavaScriptは書かない（書きたければ書ける）
- **funicular gem**（RailsプラグインのCRubyGem）
  - picoruby.wasm同梱（CDNも使える）
- 実装はピュアRuby（約1万行）
  - RailsがピュアRubyであるように

小さい

	CRuby.wasm + stdlib	PicoRuby.wasm + Funicular
Binary size (brotli)	31.0 MB (8.6 MB)	2.7 MB (0.8 MB)
Kernel#sleep	NO	YES
ASYNCIFY	YES	NO

- PicoRubyはマイコン出身なので、512 KBのRAMでも十分に動く

最近のトピック: ピュアRuby + AOT

- FunicularはピュアRuby（C拡張なし）
- VMの上で動くRubyはJavaScriptより遅い
- picoruby-aot: Rubyの実装をビルド時にC拡張へコンパイル
  - 型推論はSpinel（by Matz）
  - ソースはピュアRuby のまま
- Funicularのdiff計算（200 件のリスト）
  - **未AOT: 1.06 ms → AOT: 0.50 ms**
- Rubyで書き、WASMネイティブ（に近い）速度
- Funicular次バージョンでリリース

コンポーネント: JSXなし、Ruby DLS

```ruby
class CounterComponent < Funicular::Component
  def initialize_state
    { count: 0 }
  end

  def render          # required override
    div do
      p { "Current count: #{state[:count]}" }
      button(onclick: :increment) { "Increment" }
    end
  end

  def increment       # user-defined
    patch(count: state[:count] + 1) # kicks Differ & render
  end
end
```

Component#patchが再描画

```
Component#patch
  └─ Component#component_will_update # If callback exists
     @state.merge(normalized_state)
     Component#re_render
       └─ patches = VDOM::Differ.diff(@vdom, new_vdom)
          VDOM::Patcher.new.apply(@dom_element, patches)
     Component#component_updated     # If callback exists
```

（特別なことは何もない）

状態（**props**）は親から子へ流れ、イベントは**patch**で遡上

ルーティング: Rails風

```ruby
# app/funicular/initializer.rb
Funicular.start(container: 'app') do |router|
  router.get('/login',              to: LoginComponent, as: 'login')
  router.get('/chat/:channel_id', to: ChatComponent, as: 'chat_channel',
                                    constraints: { channel_id: /^\d+/ })
  router.get('/settings',           to: SettingsComponent, as: 'settings')
  router.set_default('/login')
end
```

- **config/routes.rbと同じ書き味**
- **as:からroutes.settings_pathができる**
- **:channel_idはprops[:channel_id]に入る**

ActionCable: Rails側はそのまんま

```ruby
# app/channels/chat_channel.rb (Rails)
class ChatChannel < ApplicationCable::Channel
  def subscribed
    stream_from "chat_#{params[:channel_id]}"
  end

  def speak(data)   # @subscription.perform("speak", message: "Hi")
    ActionCable.server.broadcast("chat_#{params[:channel_id]}", ...)
  end
end
```

- **`Funicular::Cable`** は ActionCableと完全互換
- 切れたら自動で再接続
- タブ（ブラウザ）が隠れたら休み、再表示とともに再開

フロントエンドDB: SQLite3

```ruby
# config/initializers/funicular.rb
Funicular.configure do |config|
  config.local_database = true
  config.user_key = ->(controller) { controller.request.session[:user_id]&.to_s }
end
```

- SQLite3をPicoRuby.wasmに組み込んだ（sqlite3.wasmじゃない）
- FunicularがSQLを書く
- 永続化はIndexedDB
- ログインユーザごとに異なるデータベースファイル

クライアント専用モデル

```ruby
# app/funicular/models/draft.rb
class Draft < Funicular::Model
  storage :local do
    migrate 1 do |t|
      t.integer :channel_id, null: false
      t.text    :body
      t.timestamps
      t.index   :channel_id
    end
  end
  def self.store(channel_id, body)     # upsert; sync, no blocks
    draft = local.find_by(channel_id: channel_id)
    draft ? draft.update(body: body) : create(channel_id: channel_id, body: body)
  end
end
```

- サーバに送らないデータ: 下書き、設定
- マイグレーションをモデルに書く（もっとよい方法あれば提案カモン）

Component#watch

```
class DraftListComponent < Funicular::Component
  def initialize_state
    { drafts: [] }
  end
  def component_mounted
    watch(:drafts) { Draft.local.order(:updated_at) }   # reactive
  end
  def render
    ul do
      state[:drafts].each { |draft| li(key: draft.id) { draft.body } }
    end
  end
end
```

- テーブルが変わるたびに再評価して再描画
- 書き込むタブはひとつだけ（ほかのタブは読むだけ）

SSR とハイドレーション

```ruby
class HomeController < ApplicationController
  def index
    @ssr = Funicular::SSR.render(path: "/blog", state: { posts: blog_posts })
  end
end
```

```erb
<%= funicular_app_container(@ssr ? @ssr[:html] : "") %>
<%= funicular_state_tag(@ssr[:state]) if @ssr %>
<script type="application/x-mrb" src="<%= asset_path('app.mrb') %>">
```

- サーバがHTMLを返す: 最初の表示が速い。SEO対策
- ブラウザは同じ状態からDOMを引き継ぐ
- クライアント側のコードは追加なし

同じRubyが3つの処理系で動く

Where	Runtime	Purpose
Browser	PicoRuby.wasm	The app
Rails process	CRuby	SSR
Node.js + jsdom	PicoRuby.wasm	bin/rails test

- FunicularのDifferやComponentなどの実装が共通
- SSRとSPAのDOM構築ロジックの同一性を**完全保証**
  （ただしCRubyとmrubyの非互換だけは要注意）

プラグイン: gem で配る

```ruby
# Gemfile
group :funicular do
  gem "funicular-datepicker"
  gem "funicular-image-uploader"
end
```

- プラグインはただのCRubyGem
- Bundler のグループに入れるだけで登録完了
- アプリと同じ**app.mrb**にコンパイルされる
- npmでなくRubyGemsのエコシステム

テスト: bin/rails testに統合

```ruby
# test/funicular/client/login_component_picotest.rb
class LoginComponentTest < Funicular::Testing::DOMTest
  def test_empty_submit_shows_validation_error
    mount LoginComponent
    submit "form"
    assert_text "Please enter username and password"
  end
end
```

- **bin/rails test** → Node.js → jsdom → PicoRuby.wasm
- モデルのテストもコンポーネントのテストも1つのコマンドで
- アサーション数はRailsのレポートにまとまる

デバッグ: コンポーネントの枠

```ruby
# app/funicular/initializer.rb
Funicular.debug_color = "pink"    # outlines every component
```

- 開発環境ではコンポーネントごとに色つきの枠
- マウスを乗せるとクラス名が出る

デバッグ: ブラウザで binding.irb

```ruby
def handle_submit(data)
  binding.irb   # pauses here, in the browser
  Comment.create(post_id: state[:post]["id"], body: data["body"]) { ... }
end
```

- Chrome拡張: **PicoRuby Debugger**
  - REPL、ステップ実行、ローカル変数、コールスタック
  - コンポーネントの状態とツリー
  - 未成熟なのでコントリビュータ募集

Funicular: 向いていない人

- 最大限の性能がほしい: wasmとVMのオーバーヘッド
  - AOTには（ほどほど）ご期待ください
- 巨大で複雑な状態を管理したい: ベストプラクティスが**まだ**ない（だれか頼む）
- 高い信頼性が要る: まだ誰も使っていない（hasumikin以外）
- Hotwire/Turbo で足りている？ ならそのままでよいです

Funicular: 向いている人

Rubyを愛するひと全員

まとめ

まとめ

Rails	Funicular (browser)
rails generate	funicular:chat
routes.rb	router.get(...) / routes.*_path
ActionCable channel	Funicular::Cable
ERB view	Component#render (+ SSR on CRuby)
SQLite3 on the server	SQLite3 in the browser
Bundler group	Plugins (Rubygems)
bin/rails test	*_picotest.rb on jsdom
ActiveRecord validations	(RubyWorld Conference 2026にて)

予告

RubyWorld Conference 2026

Object-REST Mapper

```ruby
Post.all(category: "tech")     # GET    /api/posts?category=tech
Post.find(123)                 # GET    /api/posts/123
Post.create(title: "Hi")       # POST   /api/posts
post.update(title: "Edited")   # PATCH  /api/posts/123
post.destroy                   # DELETE /api/posts/123
```

- ActiveRecord はテーブルを Ruby のオブジェクトにする
- Object-REST Mapper はリソースを Ruby のオブジェクトにする

統合されたバリデーション

```rb
# app/models/comment.rb (Rails)
class Comment < ApplicationRecord
  validates :body, presence: true
end

# app/controllers/api/schema_controller.rb (Rails)
# スキーマにバリデーション情報が乗ってくる
render json: Funicular::Schema.build(
       Comment, attributes: { "body" => { type: "string", readonly: false } })

# app/funicular/models/comment.rb (Funucular)
class Comment < Funicular::Model; end
```

- ブラウザで先に検証: 不正ならリクエストを送らない
- サーバの検証結果も同じ **errors** オブジェクト

続きは RubyWorld Conference 2026 で

- 12月3日と4日（のDay2）
  - **REST Rebooted: What Funicular on Rails Carries**
  - RESTの6つの制約を復習
  - バックエンドとフロントエンドの真の統合

Star押してくれ〜🌟

github.com/picoruby/funicular

ドキュメントとチュートリアル👀

picoruby.org/funicular