Hotwire
ことはじめ

2022/12/13
株式会社万葉

資料作成：依光奏江　デモ作成：鳥井雪

本セミナーの目的

- Railsは知っているけどHotwireはよく知らない、というような方向けにHotwireの初歩的な内容について解説させていただきます
- Hotwireを構成する技術要素の簡単な解説と、それぞれのデモを行い、雰囲気を掴んで頂くことが目的です
- Hotwireの導入の利点や使い所なども紹介

アジェンダ

1. Hotwireとは？
2. Hotwireを構成するもの
3. Hotwire詳解

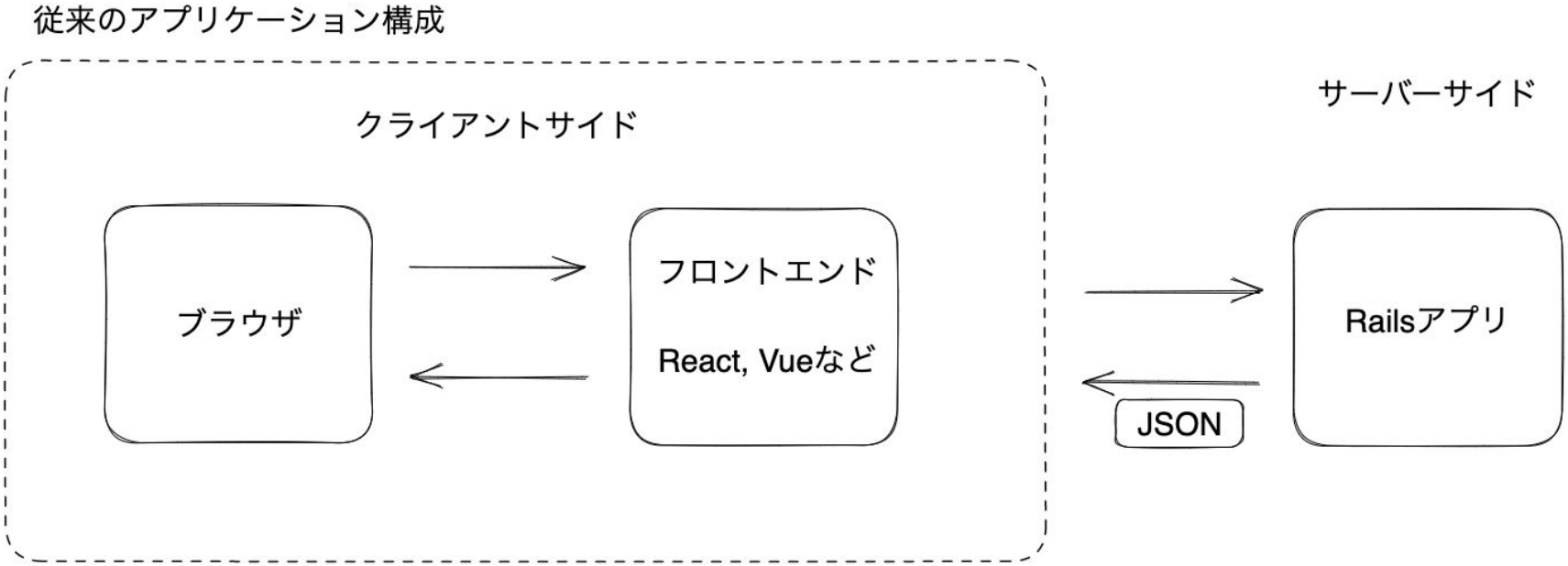
Hotwireとは？

Hotwireについて

- Rails7から標準で組み込まれるようになったフロントエンドのフレームワーク
- 「HTML OVER The Wire」が名前の由来で、HTMLをサーバから送る方式を採用している
- Rails専用ではない(Laravelで利用できるパッケージもある)

主要なフロントエンドフレームワークとの違い

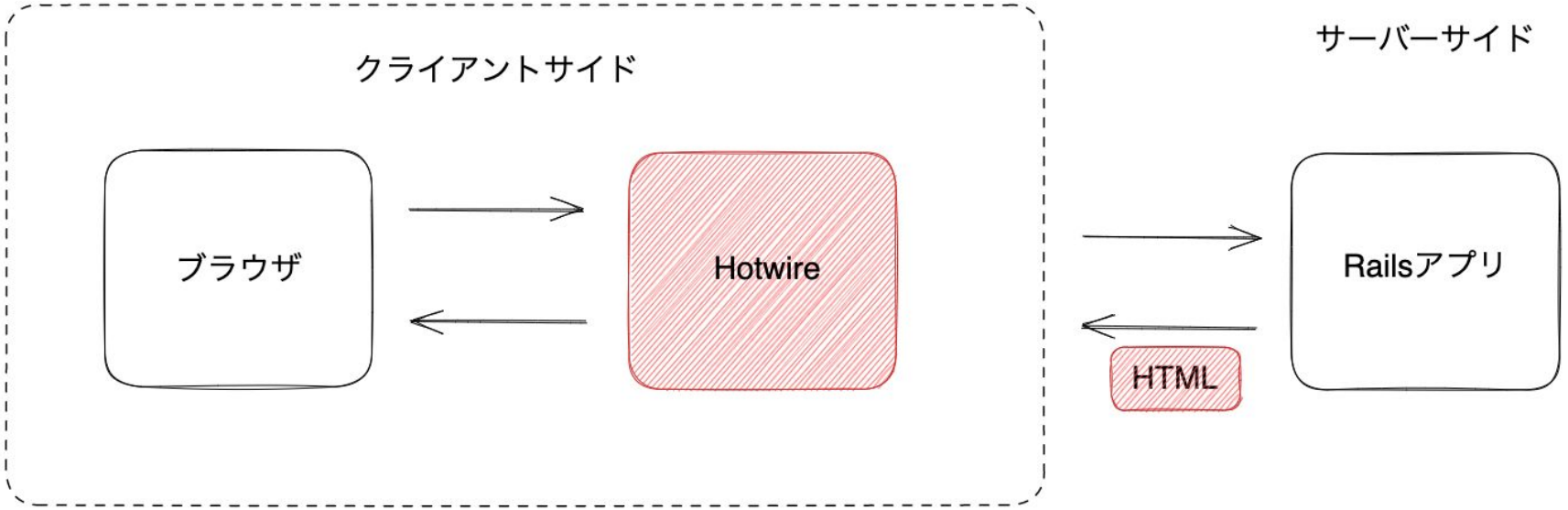
- ReactやVueの場合、フロントエンドフレームワークがHTMLのレンダリングを受け持ち、サーバーサイドはAPIサーバーとしてJSONを返す



主要なフロントエンドフレームワークとの違い

- サーバーサイドはJSONではなくHTMLを送信し、基本的にHotwireはそれを適切に配置していくだけ

Hotwireを利用したアプリケーション構成



Hotwireの利点

- Railsの提供する機能に乗れるので、フロントエンドのシステムを別途構築・メンテナンスするコストが省ける
- Railsエンジニアにとって、他の大きなフロントエンドのフレームワークよりも学習コストが低い
- SPA風の実装で、ユーザーの体験を向上することを手軽に実現できる

Hotwireの欠点

- 大掛かりなJavaScriptフレームワークより、実現できる範囲は制限される
- 新しい機能のため、まだ知見がすくない

Hotwireを構成するもの

Hotwireの構成技術

- Hotwireは3つの技術から構成されている
  - Turbo
  - Stimulus
  - Strada

Turbo とは？

- Hotwireの中心機能のJavaScriptライブラリ
- ページ遷移やフォームの送信などを高速化したり、SPAのような動作を実現するために必要な機能を提供
- 4つの技術から構成されている
  - Turbo Drive
  - Turbo Frames
  - Turbo Streams
  - Turbo Native

Stimulus とは？

- JavaScriptのフレームワーク
- Turboである程度インタラクティブな動作は実現できるが、それでもJavaScriptを書きたくなった時に使う
- 他のフロントエンドのフレームワークとは違い、HTMLのレンダリング等には関与せず、すでにあるHTMLを操作することに特化している
  - JavaScriptで状態を持たず、MutationObserverを利用してDOMの変更を監視して、要素が出現したら自動でアタッチするので、要素探索もしなくてよい

Hotwireの構成要素

機能	サーバー側からのレスポンス	画面の更新対象
これまでのRails	すべてのHTML (CSS、JS、HTML)	すべて
TurboDrive	すべてのHTML	body要素 head要素(差分のみマージ)
TurboFrames	メインテンプレートのHTML	<turbo-frame>タグで囲った範囲
TurboStreams	TurboStreamテンプレートのHTML	任意
Stimulus	-	-

Hotwire詳説

Turbo Drive

TurboDrive とは？

- Turbolinksの進化版
  - Turbolinksとは、Rails4からデフォルトで採用されていた画面遷移を高速化するgem
- 主にページ遷移周りに作用する
- Turbolinksとの違い
  - 機能的には殆ど変わっていない
  - リンクだけでなくフォームも監視している
  - クラス名やイベント名等も変わったりしている

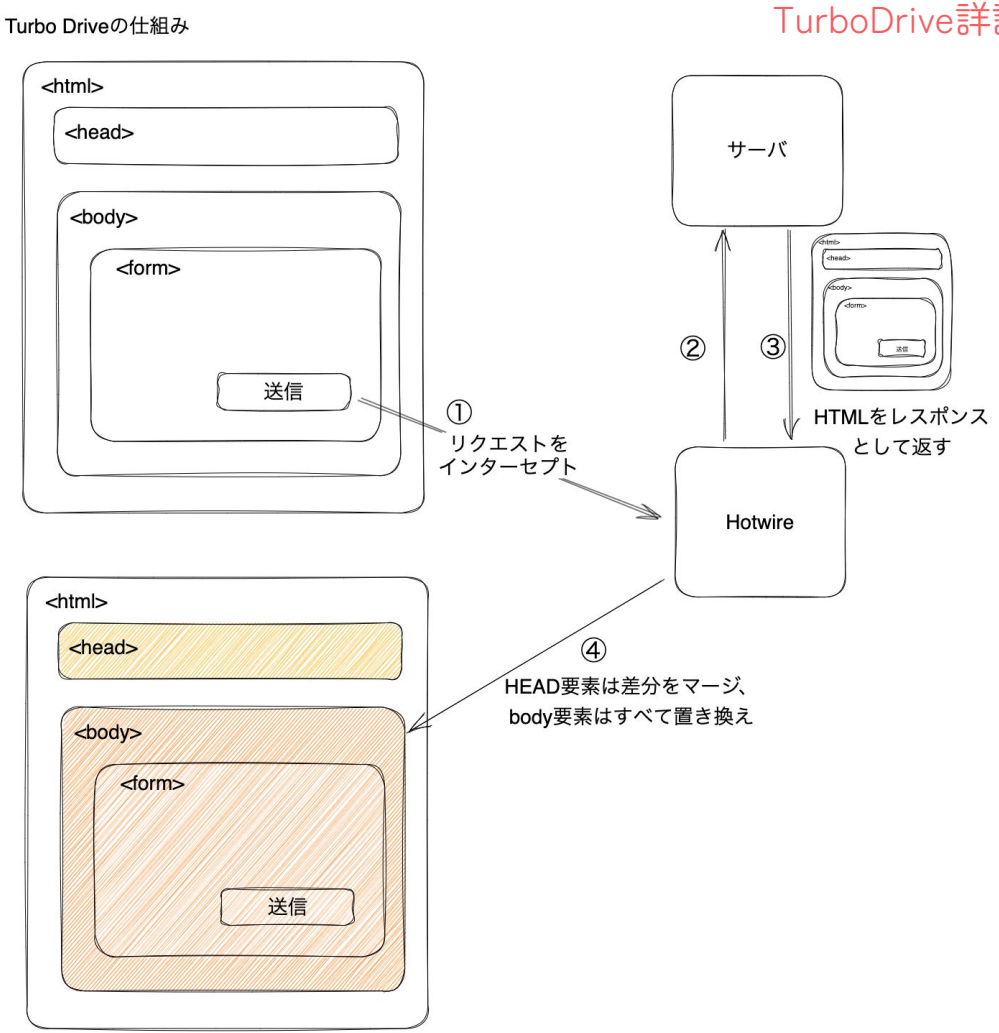
TurboDrive とは？

- 基本的にTurbolinksとやっていることは同じで、headタグを差分だけマージ、bodyタグは置き換えをすることで、画面遷移が起きたように見せかけている
  - headタグの差分だけをマージすることで、CSS、JSの再読み込みが発生しなくなるため、読み込み速度が向上する
- 上記の仕組みを使って、クライアントサイドでルーティングしたり状態管理すること無く、SPAと同等な高速な画面遷移を実現している

TurboDriveのしくみ

- 同じドメインへのリンクまたはフォームを監視し、HistoryAPIを利用してURLを変更し、fetchリクエストに差し替えてHTMLのレンダリングを行う
- レンダリングの際にbody要素は完全に置き換えるが、head要素は一部をマージし、その他はそのまま維持する(ここでJS、CSSの再読み込みが発生しないので画面遷移が高速になっている)

TurboDriveのしくみ



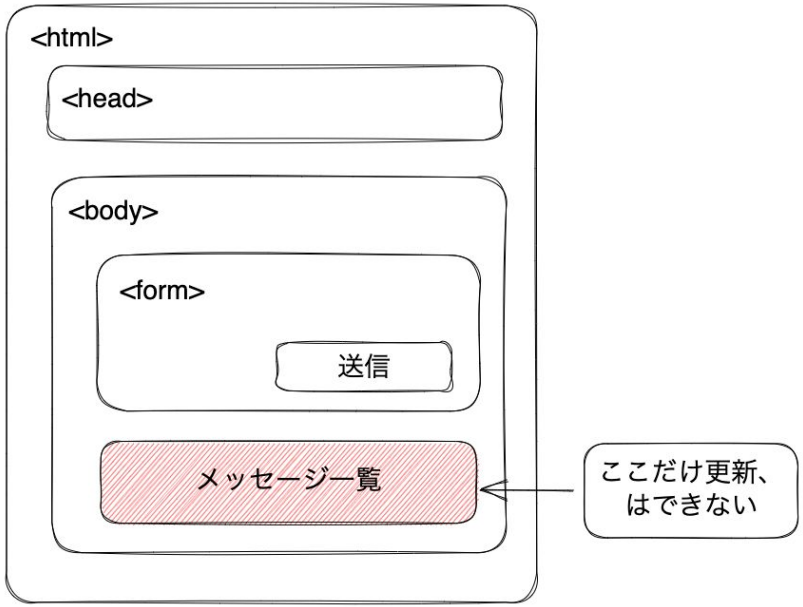
その他のポイント

- HistoryAPIを使うことで、画面遷移は発生しないけども、ブラウザの戻る、進むボタンは使える
- キャッシュを利用して画面表示を行っているので高速化になっている
- 通常のページキャッシュのほか、画面読み込みの間キャッシュを表示する機能(プレビュー)もある

Turbo Drive デモ

TurboDriveでできないこと

- bodyタグを置き換えすることしかできないため、ページ内の一部のみの更新というようなことはできない



Hotwireの構成要素

機能	サーバー側からのレスポンス	画面の更新対象
これまでのRails	すべてのHTML (CSS、JS、HTML)	すべて
TurboDrive	すべてのHTML	body要素 head要素(差分のみマージ)
TurboFrames	メインテンプレートのHTML	<turbo-frame>タグで囲った範囲
TurboStreams	TurboStreamテンプレートの HTML	任意
Stimulus	-	-

Hotwire詳説
Turbo Frames

TurboFrames とは？

- Driveの部分置換版
- TurboFramesでできること
  - 一部の要素を置き換え
  - ページの一部の遅延読み込み

TurboFramesの基本的な使い方

app/views/emotions/index.html.erb

```
<body>
  <ol class="emotion-list w-100">
    <%= turbo_frame_tag "emotions" do %>
      <%= render partial: "emotion", collection: @emotions %>
    <% end %>
  </ol>
</body>
```

app/views/layouts/_sidebar.html.erb

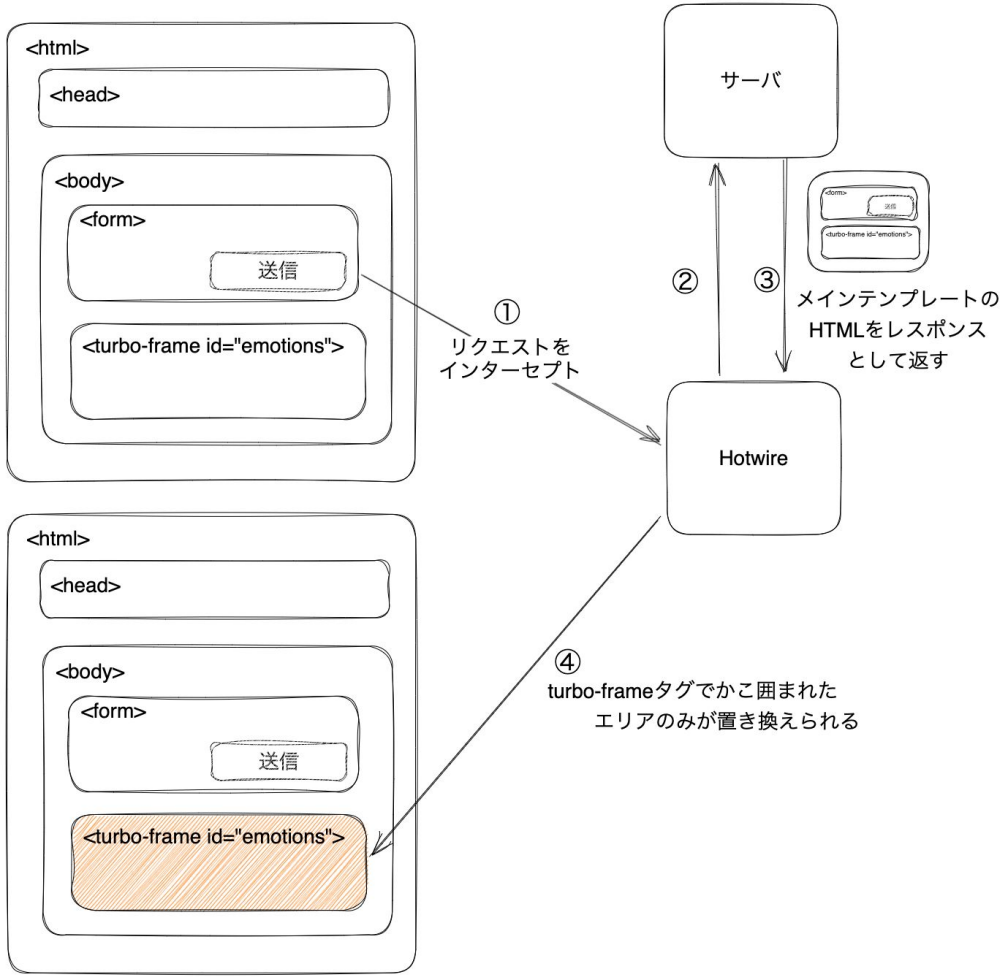
```
<%= link_to root_path, class: "nav-link", data: { turbo_frame: "emotions" }
do %>
  <%== show_icon(Emotion.icons[:love], width: 30, height: 30) %>
<% end %>
```

- TurboDriveはbody要素すべてを置き換えるが、TurboFramesでは指定した一部分のエリアのみ置き換える
- <turbo-frame> で囲まれたエリアが対象
- <turbo-frame>タグ内のリンクなら、特に指定なしで置き換えられる
- タグ外の場合がdata-turbo_frame属性で対象を指定する
- コード例だと、サイドバーのリンクをクリックすると、囲まれたエリアのemotions一覧が置き換えられる

TurboFramesのしくみ

- fetchまではTurboDriveと変わらない
- サーバからはメインテンプレート(index.html.erbなど)のHTMLのみが返ってくる

Turbo Framesの仕組み



Turbo Framesデモ

TurboFramesの少し便利な使い方

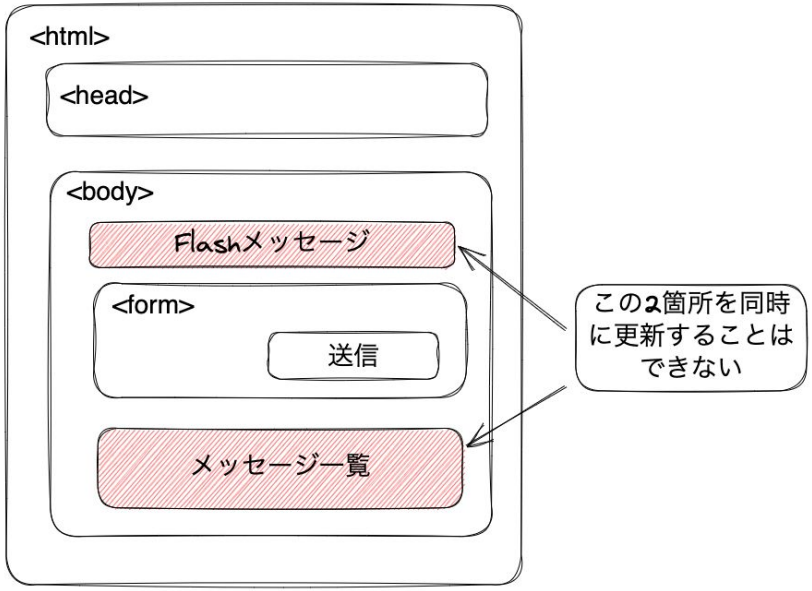
- 遅延読み込み
  - turbo-frameタグにsrc属性をつけると、ページロード後にsrcに指定したURLにTurboFrameリクエストし、遅延読み込みする

```
<body>
  <div id="navigation">メッセージ一覧</div>
  <turbo-frame id="messages" src="/messages"
loading="lazy">
    <img src="spinner.gif">
  </turbo-frame>
</body>
```

- turbo-frameタグ内にコンテンツを置いておくと、レスポンスが返されるまで表示される
- loading＝”lazy” を指定するとページロード時ではなく、スクロールなどで表示される時に遅延読み込みされる

TurboFramesでできないこと

- ページ内の複数箇所の更新
- リンクやフォーム送信以外をトリガーにした更新



Hotwireの構成要素

機能	サーバー側からのレスポンス	画面の更新対象
これまでのRails	すべてのHTML (CSS、JS、HTML)	すべて
TurboDrive	すべてのHTML	body要素 head要素(差分のみマージ)
TurboFrames	メインテンプレートのHTML	<turbo-frame>タグで囲った範囲
TurboStreams	TurboStreamテンプレートのHTML	任意
Stimulus	-	-

休憩

Hotwire詳説
Turbo Streams

TurboStreams とは？

- TurboDriveやTurboFramesより柔軟にDOM操作できるもの
- TurboFramesでできること
  - 複数箇所の更新(追加・更新・削除)
  - WebSocketやSSEと組み合わせてリアルタイム更新

複数要素の更新について

- TurboFramesは1箇所の要素の置き換えしかできなかったが、TurboStreamsでは複数箇所の更新ができる
- TurboFramesはturbo-frameタグ内のリンクやフォームのリクエストを自動で置き換え処理しているが、TurboStreamsではどんな操作(追加・更新・削除)をするのか明示的に指示する必要がある
- GETリクエストは扱えない

TurboStreamsの使い方(複数要素の更新)

- メッセージの更新と合わせてFlashメッセージも表示する

app/views/emotions/index.html.erb

```
<div id="flash"></div>
```

app/views/emotions/_new.html.erb

```
<%= turbo_frame_tag "emotion_form" do %>
  <h4 class="mb-3">こころのこえをつぶやく</h4>
  <div>
    (中略)
  </div>
<% end %>
```

app/controllers/emotions_controller.rb

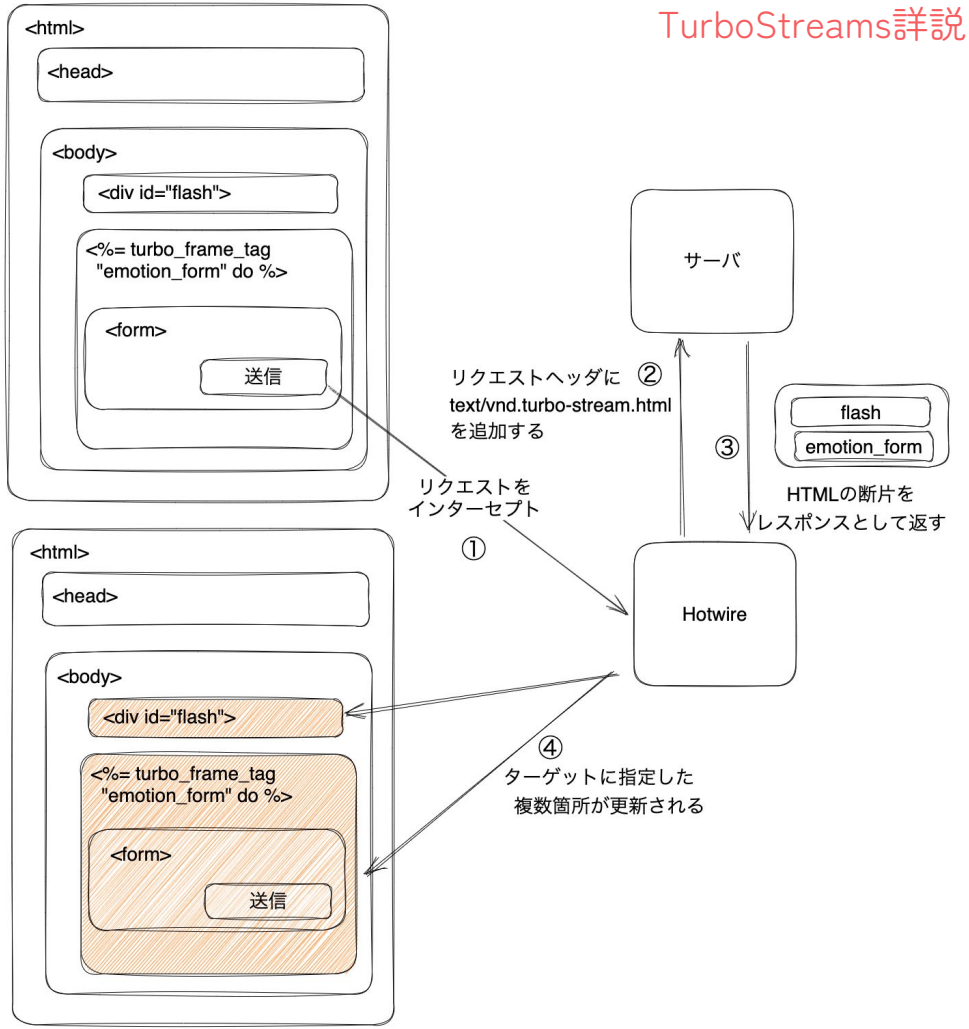
```
def create
  @emotion = Emotion.new(emotion_params)
  if @emotion.save
    flash.now.notice = "こころのこえをつぶやきました。"
  end
end
```

app/views/emotions/create.turbo_stream.erb

```
<%= turbo_stream.update "flash" do %>
  <%= render partial: "layouts/flash" %>
<% end %>
<%= turbo_stream.replace "emotion_form" do %>
  <%= render partial: "new", locals: {emotion: @emotion.persisted? ?
Emotion.new : @emotion} %>
<% end %>
```

TurboStreamsのしくみ

- 具体的な仕組み
  - インターセプトした際にリクエストヘッダにTurboStreamsのフォーマットを追加する
  - レスポンスのHTMLの断片を指定の場所に更新する



Turbo Streamデモ
(複数要素の更新)

WebSocketなどを利用した自動更新

- 自分の画面だけをリアルタイム更新するものと、他のプロセスにも即時反映するブロードキャスト方式がある
- ブロードキャスト方式をRailsで利用する場合は、WebSocketと組み合わせる(ActionCableを使う)
- ActionCableを使うのでRedisも必要

TurboStreamsの使い方(自動更新)

- turbo_stream_fromメソッドでTurboがemotionsストリームのサブスクライブを開始する
- ブロードキャスト処理をモデルに記述

app/models/emotion.rb

```
class Emotion < ApplicationRecord
  (略)
  after_create_commit -> { broadcast_prepend_to("emotions") }
  after_destroy_commit -> { broadcast_remove_to("emotions") }
end
```

app/views/emotions/index.html.erb

```
<%= turbo_stream_from "emotions" %>
```

Turbo Streamデモ
(自動更新)

Hotwireの構成要素

機能	サーバー側からのレスポンス	画面の更新対象
これまでのRails	すべてのHTML (CSS、JS、HTML)	すべて
TurboDrive	すべてのHTML	body要素 head要素(差分のみマージ)
TurboFrames	メインテンプレートのHTML	<turbo-frame>タグで囲った範囲
TurboStreams	TurboStreamテンプレートの HTML	任意
Stimulus	-	-

Hotwire詳説

Stimulus

Stimulus とは？

- JavaScriptのフレームワーク
- Turboである程度インタラクティブな動作は実現できるが、それでもJavaScriptを書きたくなった時に使う
- 状態をDOMに持つので、サーバーサイドレンダリングと相性が良い
  - JSフレームワークは、JSの個別に作ったオブジェクトに状態を持つことが多い
  - Stimulusは状態をDOMと紐づけて持つ
- 「控えめなJavaScript」なので、他ライブラリとほとんど干渉しない

Stimulusの基本的な使い方

app/views/emotions/_new.html.erb

```
<%= form_with model: emotion, local: true, html: { data: {controller:
"emotions", action: "emotions#submit"}, id: "form" } do |form| %>
  <%= form.radio_button :icon, :love, data: {emotions_target: "icon"} %>
  <%= form.text_field :name, data: {emotions_target: "name"} %>
<% end %>
```

app/javascript/controllers/emotions_controller.js

```
import { Controller } from "@hotwired/stimulus"

export default class extends Controller {
  static targets = [ "name",  "icon" ]

  submit() {
    const isAllInput = this.nameTarget.value !== ""
    this.element.requestSubmit()
  }
}
```

- フォーム送信時に処理する例
- Stimulusはcontroller、action、target要素から成り立つ
- data-controller属性: HTMLにコントローラーを割り振る
- data-action属性: イベントハンドリング
- data-target属性: DOMへの参照

Stimulusデモ

まとめ

まとめ

- 技術的構成
  - HotwireはTurbo、Stimulus、Stradaの3つの技術で構成されている
  - Turboはお手軽にSPA風アプリを実現できる
  - StimulusはTurboでは手が届かないところをフォローしてくれる

機能	サーバー側からのレスポンス	画面の更新対象
TurboDrive	すべてのHTML	body要素 head要素(差分のみマージ)
TurboFrames	メインテンプレートのHTML	<turbo-frame>タグで囲った範囲
TurboStreams	TurboStreamテンプレートのHTML	任意
Stimulus	-	-

まとめ

- 利点
  - フロントエンドのシステムを別途構築・メンテナンスするコストが省ける
  - Railsエンジニアにとっては学習コストが低い
  - SPA風の実装で、ユーザーの体験の向上が手軽に実現できる
- 欠点
  - リッチなUIの提供は難しい
  - 新しい機能のため情報が少ない

結論

欠点・利点のバランスをみて使い所を決めてゆくのがおすすめ