Dinamizando el cliente con Frankt

Cristian Álvarez Belaustegui

¿Qué es Frankt?

Eventos desde el servidor

Envío de información desde servidor

El modelo estándar HTTP el servidor siempre responde a peticiones iniciadas por el cliente

- **Polling**: el cliente solicita nueva información del servidor periódicamente
- **Long polling**: el servidor no responde a la petición hasta que puede enviar datos
- **Server-Sent Events**: el servidor mantiene una conexión abierta por la que va enviando los datos

Websockets

Permiten sustituir los mecanismos que hemos visto antes.

- Proporcionan una **comunicación bidireccional cliente-servidor**
- Sólo es necesario mantener una única conexión
- **Utiliza la infraestructura HTTP existente**
  - Usa los puertos 80 (ws) y 443 (wss)
  - Suporta los proxies HTTP y otros intermediarios
- El soporte está bastante extendido
  - IE 10, Edge, Firefox, Chrome, Safari 7, iOS 6.1, Android 4.4
  - **95% de soporte**

Phoenix

Channels (objetivo)

Abstracción para trabajar con conexiones de larga duración entre cliente y servidor

- Por defecto usa WebSockets
- Enviar y recibir mensajes entre el cliente y el servidor
- El cliente y el servidor pueden ser al mismo tiempo emisores y receptores
- Los mensajes se organizan en “topics”
- Un mensaje puede tener múltiples receptores

Channels (funcionamiento)

- El cliente se conecta a un socket usando un protocolo de transporte y se une a uno o más topics
- Se crea un proceso por cliente y por topic. Puede mantener el estado entre los distintos mensajes.
- Una vez conectado los mensajes que llegan se envían al proceso correspondiente. Si se hace un broadcast el mensaje se envía a todos los clientes conectados al mismo topic

Frankt

Filosofía

Ejecutar acciones en el cliente usando la lógica de negocio que ya tenemos en el backend.

- Fina capa sobre los channels de Phoenix
- Poca carga en las plantillas
- Extensible

Historia

- **25-mayo-2017**: @vortizhe y @odarriba hacen la implementación y el DSL inicial de Frankenstein en Acutario
- **26-septiembre-2017**: @belaustegui “integra” Frankenstein en Bizneo
- **21-diciembre-2017**: @vortizhe extrae Frankenstein a una librería independiente. Vive en [acutario/frankt]
- **5-enero-2018**: integración de la librería Frankt en Bizneo
- **4-junio-2018**: soporte para configuración por canal y gestión de errores
- **21-junio-2018**: soporte para plugs en las acciones de Frankt

Ejemplo de formulario simple (front)

```
<!-- index.html.eex -->
<%= form_for @conn, nil,  [id: "greet-form", as: :greet], fn f -> %>
  <%= text_input f, :name %>
  <%= submit "Greet!", data: [frankt: [action: "greeting:greet", target: "#greet-form"]] %>
<% end %>
<%= render "_greeting.html" %>

<!-- _greeting.html.eex -->
<div id="greeting">
  <%= if assigns[:name], do: "Hello, #{assigns[:name]}" %>
</div>
```

Ejemplo de formulario simple (back)

```
defmodule Frankt.TestApplicationWeb.FranktHandlers.Greeting do
  import Phoenix.Channel
  import Frankt.Handler

  alias Frankt.TestApplicationWeb.GreetView

  def greet(%{"greet" => %{"name" => name}}, socket) do
    push(socket, "replace_with", %{
      html: render(socket, GreetView, "_greeting.html", name: name),
      target: "#greeting"
    })
  end
end
```

Ejemplo de eventos desde el servidor

```
# En consola, en una tarea de Mix, en un proceso, etc.

alias Frankt.TestApplicationWeb.Endpoint
alias Frankt.TestApplicationWeb.GreetView

Endpoint.broadcast!(topic, "replace_with", %{
  html: Phoenix.View.render_to_string(GreetView, "_greeting.html", name: name),
  target: "#greeting"
})
```

Ejemplo de envío automático (front)

```
<!-- index.html.eex -->
<%= form_for @conn, nil, [id: "filter-form", as: :filters], fn form -> %>
  <% frankt_data = [action: "filter:filter", target: "#filter-form", auto: true] %>

  <%= text_input form, :name, placeholder: "Name", data: [frankt: frankt_data] %>
  <%= select form, :gender, gender_options(), prompt: "Any", data: [frankt: frankt_data] %>
<% end %>

<%= render "_table.html", assigns %>
```

Ejemplo de chat por @odarriba (front)

```
<!-- index.html.eex -->
<%= form_for @conn, nil,  [id: "chat-form", as: :chat],  fn f -> %>
 <%= label f, :sender %>
 <%= text_input f, :sender %>

 <%= label f, :message %>
 <%= textarea f, :message %>

 <%= submit "Send!", data: [frankt: [action: "chat:send", target: "#chat-form"]] %>
<% end %>
```

Ejemplo de chat por @odarriba (back)

```
defmodule Frankt.TestApplicationWeb.FranktHandlers.Chat do
 import Frankt.Handler

 alias Frankt.TestApplicationWeb.Endpoint
 alias Frankt.TestApplicationWeb.ChatView

 def send(%{"chat" => %{"sender" => sender, "message" => msg}}, socket) do
   Endpoint.broadcast(socket.topic, "append", %{
     html: render(socket, ChatView, "_message.html", sender: sender, message: msg),
     target: "#chat"
   })
 end
end
```

Próximos pasos

Publicar la primera versión de Frankt en hex.pm

Pero antes:

- Tests de integración y aplicación de muestra
- Mejoras en la documentación
  - Manual
  - Guías
  - Ejemplos

Alternativa: Drab

Permite controlar el frontend desde Phoenix sin tener que escribir Javascript.

- Lleva tiempo funcionando (publicada el **14 de enero de 2017**)
- Mucha funcionalidad
- **Bastante carga conceptual propia**
  - Commanders
  - Peek
  - Poke
  - Formato de plantillas

¿Dudas? ¿Ruegos?
¿Preguntas?

Gracias