PV281: Programování v Rustu

Obsah

AJAX (Asynchronous Javascript and XML)

Pojďme se podívat 10 let zpátky:

- místo celé stránky posíláme pouze vykreslené části
- není třeba překreslit celou stránku

Kdy se mi to bude hodit?

Instalace přes npm

```
npm install htmx.org
```

následně přidat do stránky `node_modules/htmx.org/dist/htmx.js`

Instalace přes CDN (unpkg)

```
<script src="https://unpkg.com/htmx.org@1.9.8" integrity="sha384-rgjA7mptc2ETQqXoYC3/zJvkU7K/aP44Y+z7xQuJiVnB/422P/Ak+F/AqFR7E4Wr" crossorigin="anonymous"></script>
```

Load data

```
<div hx-get="/products"></div>
```

Response je vykreslená v elementu, který provedl request.

Requesty

Atribut	Typ requestu
hx-get	Provede `GET` na URL
hx-post	Provede `POST` na URL
hx-put	Provede `PUT` na URL
hx-patch	Provede `PATCH` na URL
hx-delete	Provede `DELETE` na URL

Předání parametrů

```
<div hx-get="/example" hx-vals='{"myVal": "My Value"}'>Get Some HTML, Including A Value in the Request</div>
```

Target

```
<div hx-get="/products" hx-target="#product-results">
        Products
</div>
<div id="#product-results"></div>
```

Pro změnu cíle pro vykreslení je nutné použít `hx-target`.

Extended selector

Atributy (jako `hx-target`), které očekávají CSS selector, ve větši případů podporují rozšířenou syntaxi.

`closest <CSS selector>` najde nejbližšího rodiče elementu

`next <CSS selector>` najde element níže v DOMu (následující)

`previous <CSS selector>` najde element výše v DOMu (předcházející)

`find <CSS selector>` najde nejbližšího potomka elementu

Swapping

Hodnota	Popis
`innerHTML`	výchozí nastavení - nahradí obsah elementu (potomky)
`outerHTML`	nahradí celý element včetně potomků
`none`	bez vypsání odpovědi

Loading indikátor

```
<button hx-get="/click">
    Click Me!
    <img class="htmx-indicator" src="/spinner.gif">
</button>
```

Standardně se loading indikátor použije potomek s třídou `htmx-indicator`. Jeho `opacity` se nastaví na 1. Element jde vybrat pomocí `hx-indicator="#indicator"`.

View Transition

Využítím View Transitions API lze animatovat změny v rámci DOMu.

```
<div hx-get="/products" hx-target="#product-results" hx-swap="transition:true">
        Products
</div>
<div id="#product-results"></div>
```

Vykreslení JSONu

Vzhledem k potenciálním problémům s CORS je doporučené použít server jako proxy pro získání dat.

Pokud ale k tak potřebujete udělat request na klientu, tak lze použít klientské šablony.

Klientské šablony

```
<script src="https://unpkg.com/htmx.org/dist/ext/client-side-templates.js"></script>
<script src="https://unpkg.com/mustache@latest"></script>
```

Mustache šablona

```
<body>
  <div hx-ext="client-side-templates">
    <button hx-get="/user"
            hx-swap="innerHTML"
            hx-target="#content"
            mustache-array-template="foo">
      Load user data
    </button>

    <div id="content">Loading...</div>

        <!-- Tady bude sablona -->

  </div>
</body>
```

```
<body>
  <div hx-ext="client-side-templates">
    <!-- tlacitko a div z prechoziho slajdu -->

    <template id="foo">
      {{#data}}
      <p> {{name}} at {{email}} is with {{company.name}}</p>
      {{/data}}
    </template>
  </div>
</body>
```

Pozor na mixování s Askamou

Formulář

```
<form hx-post="/store">
    <input id="title" name="title" type="text"
        hx-post="/validate"
        hx-trigger="change"
    >
    <button type="submit">Submit</button>
</form>
```

Přidání parametru mimo form

```
<div>
    <button hx-post="/register" hx-include="[name='email']">
        Register!
    </button>
    Enter email: <input name="email" type="email"/>
</div>
```

Odeslaní dat jako JSON

```
<script src="https://unpkg.com/htmx.org/dist/ext/json-enc.js"></script>
```

Přidáním `hx-ext='json-enc'` je request odeslaný jako typ `applicaton/json` s převodem na JSON v těle.

```
<form hx-post="/api/store" hx-ext='json-enc'>
    <input id="title" name="title" type="text">
    <button type="submit">Submit</button>
</form>
```

Odeslání souboru

Periodicky se vyvolává `htmx:xhr:progress`. Odpovídá standardní `progress` události během uploadu.

```
<form hx-encoding='multipart/form-data' hx-post='/upload'
          _='on htmx:xhr:progress(loaded, total) set #progress.value to (loaded/total)*100'>
        <input type='file' name='file'>
        <button>
            Upload
        </button>
        <progress id='progress' value='0' max='100'></progress>
    </form>
```

Zpracování v Rustu

```
async fn save_files(
    MultipartForm(form): MultipartForm<UploadForm>,
) -> Result<impl Responder, Error> {
    for f in form.files {
        let path = format!("./tmp/{}", f.file_name.unwrap());
        log::info!("saving to {path}");
        f.file.persist(path).unwrap();
    }

    Ok(HttpResponse::Ok())
}
```

Zpracování v Rustu

```rust
#[actix_web::main]
async fn main() -> std::io::Result<()> {
    env_logger::init_from_env(env_logger::Env::new().default_filter_or("info"));

    log::info!("creating temporary upload directory");
    std::fs::create_dir_all("./tmp")?;

    log::info!("starting HTTP server at http://localhost:8080");

    HttpServer::new(|| {
        App::new()
            .wrap(middleware::Logger::default())
            .app_data(TempFileConfig::default().directory("./tmp"))
            .service(
                web::resource("/")
                    .route(web::get().to(index))
                    .route(web::post().to(save_files)),
            )
    })
    .bind(("127.0.0.1", 8080))?
    .workers(2)
    .run()
    .await
}
```

Klientská validace

Provádí se na formulářových prvcích. Jinde je třeba zapnout přes `hx-validate='true'`.

Skriptování je přes hyperscript. Pro instalaci:
přidat:

Klientská validace

```
<form hx-post="/test">
    <input _="on htmx:validation:validate
                  if my.value != 'foo'
                      call me.setCustomValidity('Please enter the value foo')
                  else
                      call me.setCustomValidity('')"
        name="example"
    >
</form>
```

CSS extensions

```
<script src="https://unpkg.com/htmx.org/dist/ext/class-tools.js"></script>
```

```
<div hx-ext="class-tools">
    <div classes="add foo"/> <!-- adds the class "foo" after 100ms -->
    <div class="bar" classes="remove bar:1s"/> <!-- removes the class "bar" after 1s -->
    <div class="bar" classes="remove bar:1s, add foo:1s"/> <!-- removes the class "bar" after 1s
                                                               then adds the class "foo" 1s after that -->
    <div class="bar" classes="remove bar:1s & add foo:1s"/> <!-- removes the class "bar" and adds
                                                                class "foo" after 1s  -->
    <div classes="toggle foo:1s"/> <!-- toggles the class "foo" every 1s -->
</div>
```

Multiswap

```
<script src="https://unpkg.com/htmx.org/dist/ext/multi-swap.js"></script>
```

```
<body hx-ext="multi-swap,preload">
  <header>...</header>
  <menu hx-boost="true">
    <ul>
      <li><a href="/page-1" hx-swap="multi:#submenu,#content" preload="mouseover" preload-images="true">Page 1</a></li>
      <li><a href="/page-2" hx-swap="multi:#submenu,#content" preload="mouseover" preload-images="true">Page 2</a></li>
    </ul>
    <div id="submenu">... submenu contains items by selected top-level menu ...</div>
  <menu>
  <main id="content">...</div>
  <footer>...</footer>
</body>
```

Práce s historii

```
<div hx-get="/account" hx-push-url="true">
  Go to My Account
</div>
```

Pozor - pokud měníte URL musíte i umět překreslit celou stránku.

Práce s historií

```
<div hx-get="/account" hx-push-url="/account/home">
  Go to My Account
</div>
```

A nebo chytře rozdělit url komponenty (partial view) a URL stránky.

Websocket

```
<div hx-ws="connect:wss:/chatroom">
    <div id="chat_room">
        ...
    </div>
    <form hx-ws="send:submit">
        <input name="chat_message">
    </form>
</div>
```

Šablony pomocí Askamy

Jeden ze šablonovacích enginů. Šablony jsou kompilované s typovou kontrolou.

```
[dependencies]
actix-web = "4"
askama = "0.10"

[build-dependencies]
askama = "0.10"
```

Alternativou může být např. crate Tera.

Rust kód šablony

```rust
use std::collections::HashMap;
use actix_web::{web, App, HttpResponse, HttpServer, Result};
use askama::Template;

#[derive(Template)]
#[template(path = "user.html")]
struct UserTemplate<'a> {
    name: &'a str,
    text: &'a str,
}

#[derive(Template)]
#[template(path = "index.html")]
struct Index;
```

```rust
async fn index(
    query: web::Query<HashMap<String, String>>
) -> Result<HttpResponse> {
    let s = if let Some(name) = query.get("name") {
        UserTemplate {
            name,
            text: "Welcome!",
        }
        .render()
        .unwrap()
    } else {
        Index.render().unwrap()
    };
    Ok(HttpResponse::Ok().content_type("text/html").body(s))
}
```

Dědičnost šablon

Definice bloků `title`, `head` a `content` pro použití v potomcích.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>{% block title %}{{ title }} - My Site{% endblock %}</title>
    {% block head %}{% endblock %}
  </head>
  <body>
    <div id="content">
      {% block content %}<p>Placeholder content</p>{% endblock %}
    </div>
  </body>
</html>
```

Child template

```
{% extends "base.html" %}

{% block title %}Index{% endblock %}

{% block head %}
  <style>
  </style>
{% endblock %}

{% block content %}
  <h1>Index</h1>
  <p>Hello, world!</p>
  {% call super() %}
{% endblock %}
```

Include

```
<main>
  {% include "item.html" %}
</main>
```

For

```
<h1>Users</h1>
<ul>
{% for user in users %}
  <li>{{ user.name|e }}</li>
{% endfor %}
</ul>
```

If

```
{% if users.len() == 0 %}
  No users
{% else if users.len() == 1 %}
  1 user
{% else %}
  {{ users.len() }} users
{% endif %}
```

If

```
{% match item %}
  {% when Some with ("foo") %}
    Found literal foo
  {% when Some with (val) %}
    Found {{ val }}
  {% when None %}
{% endmatch %}
```

Co je cookie?

Actix Cookie

```rust
use cookie::Cookie;

let cookie = Cookie::build("name", "value")
    .domain("www.rust-lang.org")
    .path("/")
    .secure(true)
    .http_only(true)
    .finish();
```

Actix Session

- umožňuje držet stav uživatele.
- session je postavená na cookie (Set-Cookie).
- je třeba řešit tak, aby řešení bylo škálovatelné (Redis vs vše v cookie)
- OWASP cheatsheat na práci se session: https://cheatsheetseries.owasp.org/cheatsheets/Session_Manageme

```rust
use actix_web::{App, cookie::{Key, time}, Error, HttpResponse, HttpServer, web};
use actix_session::{Session, SessionMiddleware, storage::RedisActorSessionStore};
use actix_session::config::PersistentSession;

#[actix_web::main]
async fn main() -> std::io::Result<()> {
    let secret_key = get_secret_key_from_config();
    let redis_connection_string = "127.0.0.1:6379";
    HttpServer::new(move ||
            App::new()
            .wrap(
                SessionMiddleware::builder(
                    RedisActorSessionStore::new(redis_connection_string),
                    secret_key.clone()
                )
                .session_lifecycle(
                    PersistentSession::default()
                        .session_ttl(time::Duration::days(5))
                )
                .build(),
            )
            .default_service(web::to(|| HttpResponse::Ok())))
        .bind(("127.0.0.1", 8080))?
        .run()
        .await
}
```

Výběr backendu

```
[dependencies]
# ...
actix-session = { version = "...", features = ["cookie-session"] }

# pro Actix Redis
actix-session = { version = "...", features = ["redis-actor-session"] }
```

Získání session

```rust
use actix_session::Session;

#[get("/")]
async fn index(session: Session) -> Result<impl Responder> {
    // access session data
    if let Some(count) = session.get::<i32>("counter")? {
        session.insert("counter", count + 1)?;
    } else {
        session.insert("counter", 1)?;
    }

    let count = session.get::<i32>("counter")?.unwrap();
    Ok(format!("Counter: {}", count))
}
```

Dotazy?

Děkuji za pozornost