Framework Frontend

Comment lancer des applications Rust

Dans la vidéo suivante, nous allons utiliser des frameworks réalisés en Rust. Il est donc important d'installer le langage et d'en apprendre les bases. Attention, c'est un langage assez complexe. Ne vous inquiétez pas si vous ne comprenez pas tout : le compilateur vous expliquera vos erreurs lorsque vous écrirez du code.

Installer Rust



Ajouter la destination de compilation

Analysons le nom de cette destination de compilation :

- **WASM32** : WASM pour Webassembly et 32 pour dire que l'espace mémoire est sur 32 bits.
- Le premier **unknown** de la target wasm32-**unknown-**unknown concerne la source de la compilation, la machine qui a généré le binaire. Dans notre cas, elle est inconnue, premièrement parce que toutes les architectures peuvent compiler vers le WASM et aussi parce que la compilation n'est pas signée. Une fois le binaire créé, il n'est pas possible de savoir depuis quelle architecture le binaire a été compilé.
- Le deuxième **unknown** : wasm32-unknown-**unknown** concerne la destination de compilation, ici encore une fois, on ne sait pas dans quel contexte le binaire va être exécuté. Cependant, il existe des discussions en cours. Premièrement, vu les différents usages du WebAssembly, il est pertinent de vouloir différencier les targets en fonction des usages. Et pour cela, certains voudraient ajouter des destinations de compilation :
  - wasm32-npm-unknown (pour une version compatible avec npm)
  - wasm32-node-unknown

- wasm32-web-unknown

À noter qu'il existe une destination de compilation `wasm-wasi` qui est utilisée par le projet wasmtime. Elle est un standard du WASI et est adaptée à l'exécution côté serveur d'applications 100% WebAssembly. Elle a la particularité de pouvoir compiler en créant plusieurs fichiers binaires, contrairement à wasm32-unknown-unknown.

Installer trunk

Trunk est un outil de gestion d'applications très répandu sur le web. Il permet de lancer et d'orchestrer des outils qui interviennent pendant la création d'un projet web, allant de la compilation de SASS à la compression d'image et à la compilation de fichiers Rust en WebAssembly.

Il existe plusieurs méthodes pour l'installer. Maintenant que vous avez installé Rust, je vous recommande d'utiliser Cargo, car cette commande fonctionnera quel que soit votre système d'exploitation et, étant donné que Trunk est un outil spécifique à Rust, il est logique qu'il soit désinstallé lorsque vous désinstallerez Rust.

```
cargo install --locked trunk
```

Spécificité Mac

Pour les utilisateurs de la plateforme Apple Silicon, le module `wasm bindgen-cli` n'est pas installé correctement avec le tronc actuellement. Le paquet `wasm bindgen` est un outil qui permet de générer des liens (des ponts) entre le code Rust et du code JavaScript qui servira de connexion avec le navigateur.

La commande pour installer wasm bindgen-cli est la suivante :

```
cargo install --locked wasm bindgen-cli
```

Instruction spécifique a la version nightly

Il existe plusieurs versions de Rust, également appelées canaux de diffusion. Il y en a 3 principaux :

- Stable
- Beta
- Nightly

Rust ajoute des fonctionnalités en continu et n'a pas prévu de régressions. Il utilise le sémantique versionning. Aujourd'hui, aucun changement de version majeur n'est prévu.

La version stable de Rust est mise à jour toutes les 6 semaines. La version bêta représente la prochaine version stable.

Rust utilise une technique appelée "feature flag" pour déterminer si une fonctionnalité est active sur une version donnée. Si vous souhaitez utiliser les fonctionnalités non stables de Rust, vous devrez utiliser le canal Nightly et activer la fonctionnalité. C'est ce que fait le framework Leptos.

Commande pour installer le channel Nightly

```
rustup toolchain install nightly
```

Commande pour activer Rust Nightly

```
rustup override set nightly
```

Commande pour lister les channels installées

```
rustup toolchain list
```

Commande pour lister les channels installées

```
rustup toolchain list
```

En vérité, on liste les toolchains, mais je ne vais pas faire la distinction entre toolchain et channel.

Une fois passé à la version Nightly, je vous recommande de réinstaller la cible wasm32-unknown-unknown pour avoir la dernière version.

```
rustup target add wasm32-unknown-unknown
```

Démonstration

Type de rendu

Lors du développement d'une application frontend, il est important de prendre en compte ces différents enjeux lors du choix du type de rendu pour votre projet de développement afin de trouver l'équilibre approprié entre performance, charge sur les serveurs, répartition géographique des serveurs et SEO.

Le type de rendu désigne la façon dont les données sont présentées à l'utilisateur final, comment la page est construite et affichée. Il existe plusieurs solutions :

Le **Client side rendering** (CSR), dans lequel l'application est construite par le navigateur. Le serveur ne fait que renvoyer le code nécessaire au navigateur pour construire la page. C'est le code JavaScript qui va ensuite aller chercher la donnée et l'afficher dans la page.

Le **Server side rendering** (SSR) est un mode de rendu dans lequel le serveur va récupérer les données et construire la page avec ces données, puis l'envoyer au navigateur sous forme de package complet.

Le **Static site generation** (SSG) est le concept de générer des pages avec du rendu côté serveur pour des données peu souvent mises à jour, et de mettre ces pages en cache.

L'**incremental static regeneration** (ISR) est une forme de SSG dans laquelle on modifie les parties du cache sensibles aux changements plutôt que de régénérer tout le site.

Aujourd'hui, l'ISR est peu répandu et le CSR et le SSR ont des problèmes en termes de performance (SEO ou charge serveur). La génération d'un site statique est un processus qui peut être long, surtout si l'on veut construire des millions de pages. Heureusement, il existe Hugo (”https://gohugo.io/”) qui permet d'accélérer le processus de génération du site grâce à la gestion de la programmation concurrente du langage GO. Cependant, GO n'est pas un langage du web, et ici, on a dupliqué la manière de faire des rendus. Si cette logique pouvait être en WebAssembly, et donc disponible à la fois côté navigateur et côté serveur, on gagnerait énormément en performance et ouvrirait le web à de nouveaux usages. Pour cela, il faudrait néanmoins attendre que la gestion du DOM soit intégrée dans le WebAssembly. Ce qui n'est pas une priorité et qui risque de briser la nature du web et notamment la rétrocompatibilité. Cependant, peut-être verrons-nous arriver cette norme avec le HTML 7.

Ressources

https://docs.wasmtime.dev/wasm-rust.html

https://github.com/bytecodealliance/wasmtime/blob/main/docs/WASI-overview.md

https://github.com/rustwasm/team/issues/38

https://github.com/rustwasm/wasm-bindgen/issues/979

https://rust-lang.github.io/rustup/concepts/channels.html

Revision #4
Created 2023-05-05 13:12:38 UTC by Noé Larrieu-Lacoste
Updated 2023-05-05 13:21:45 UTC by Noé Larrieu-Lacoste