Introduction

Rust a commencé comme projet personnel de Graydon Hoare en 2006, un employé chez Mozilla, Mozilla va ensuite sponsoriser ce projet et compte l'utiliser pour programmer un futur moteur de navigation prévu pour être plus rapide que Gecko (utilisé pour Firefox).

Vers 2011, Rust se compile soi-même. Ainsi le compilateur Rust est écrit lui même en Rust. Rust va ensuite gagner en popularité et être utilisé dans de plus en plus de grosses entreprises tel que Alphabet (Google), Meta (Facebook), Microsoft, Discord, Amazon, Dropbox. Et un support pour Rust a également été ajouté dans le kernel Linux.

La documentation

Rust maintient 3 documentations officielles pour les débutants : une globale, une par exemples, et une autre basé sur des petits défis à faire en Rust. La pluspars des documentations se font via un site qui a exactement la même forme que celui ci et les exemples de codes peuvent être directement exécuté et modifié dans le navigateur.

La documentation des librairies est standardisées et est écrite directement dans le code, et les exemples d'utilisations écrits dans la documentation peuvent être testé automatiquement au même moment que la génération pour avoir une documentation toujours à jour.

L'utilisation de cette forme de documentation est très agréable car on peut cliquer d'un type à l'autre pour voir comment ça fonctionne, les différents champs, méthodes, fonctions et traits. On peut aussi voir le code source d'une partie de la librairie directement dans la documentation.

L'absence de null (et Options et Result)

Rust n'a pas de valeur "null" pour indiquer l'absence d'une valeur. A partir d'une déclaration de fonction ou de méthode, on peut directement voir si la fonction pourrait ne rien retourner et le compilateur nous force à traiter le cas où une erreur surviendrais ou que la valeur serait None.

Le but du compilateur est de tester un maximum de cas possibles pour empécher des mauvaises surprises plus tard.

Les erreurs Rust et les bonnes pratiques (clippy)

Le système d'erreurs de Rust est magnifique et super clair. Les erreurs sont colorées, avec des inforamations sur comment les résoudres ainsi que des liens vers la documentation de Rust en cas de soucis de compréhension.

Pour appliquer les bonnes pratiques en Rust il y a aussi un outil appellé "clippy" (lui aussi par défault) qui peut dans beaucoup de cas améliorer la qualité du code automatiquement et résoudre certaines erreurs automatiquement également.

La capacité de créer ses propres types

On peut aussi créer ses propres types en Rust comme des classes dans d'autres languages mais en beaucoup plus flexibles (on peut implémenter de nouvelles méthodes sur n'importe quel type). Le but est ainsi de faire en sorte que des cas problèmatiques qui ne sont pas censé exister, ne soit simplement pas représentable dans le programme. (voir le chapitre sur struct et enum)

La standardisation du style de code

Le style de code à employer est standardisé en rust et peut être automatiquement appliqué par les IDE ou par la commande `cargo fmt` ce qui permet d'avoir un code beaucoup plus compréhensible et uniforme.

La magie des macros

Les macros permettent de faire encore plus de tests au moment de la compilation ce qui permet par exemple à la librairie `sqlx` (pour intéragir à des bases de données mysql, postgres ou sqlite) de tester les requêtes au moment de la compilation sur une base de donnée de test ou encore à la librairie `yew` (pour faire du webassembly) de tester la validité du code HTML.

Rust fonctionne quasi partout

Rust fonctionne sur toutes les plateformes, même parfois anciennes, Rust fonctionne aussi sur des systèmes embarqués et même dans le navigateur comme remplacement de Javascript (avec webassembly) et Webassembly est parfois utilisé aussi comme pont avec d'autres architectures, plus obscures.

Le gestionaire de libraries

Je dis librarie depuis le début mais dans Rust on appelle ça des *crates*, et il se trouve que l'on peut explorer les differents crates sur crates.io et leur documentation sur docs.rs. On peut ainsi

facilement trouver des librairies et les ajouter à son projet en faisant `cargo add <nom de la lib>` (ce qui va automatiquement ajouter la version actuelle de la librarie dans la liste des dépendences)

Rapidité, efficacité et optimisation

Rust n'a pas de *garbage collector* comme beaucoup d'autres languages pour gérer la mémoire. A la place, Rust utilise un systeme ingénieux appelé *borrow checker*, qui vérifie au moment de la compilation pour faire en sorte que le code soit le plus optimisé possible.

Stabilité et durabilité

Rust est fait pour que ce qui est programmé en Rust soit toujours compatible avec les futures versions et soit également très durable (autant au niveau écologique que temporel). C'est nottament dû à l'efficacité et l'optimisation citée plus haut.

Revision #1
Created 27 April 2023 06:10:24 by SnowCode
Updated 27 April 2023 06:12:49 by SnowCode