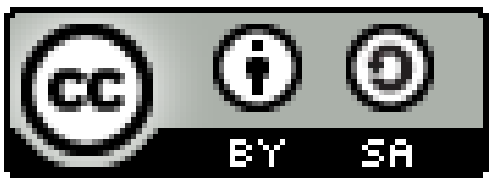
Le langage Rust

Matthieu Herrb

Conseil de service IDEA

http://homepages.laas.fr/matthieu/talks/rust-idea.pdf

Licence



Introduction

Langage C (et C++ dans une certaine mesure) en tant que langage système :

- proche du matériel
- efficace (taille du code + rapidité)
- gestion du multi-tâche (pthreads + primitives de synchronisation)

Mais gestion mémoire pas adaptée :

- débordements de buffers
- validité des pointeurs
- accès concurrents

$\rightarrow$ source de nombreuses vulnérabilités (70% des CVE / an)

Pourquoi Rust ?

Il faut que ça change !

- Rust est un langage de programmation sûr du point de vue des accès à la mémoire et bien adapté à la programmation concurrente.
- Développé par Mozilla, mais piloté par la communauté.
- Adopté par de nombreux grands groupes (Microsoft, Intel,...)
- Gestionnaire de paquets + écosystème Cargo

Oxydation $\rightarrow$ conversion de projets vers Rust

Caractéristiques du langage

- sécurité des accès mémoire dynamique, sans ramasse-miettes
- sécurité des accès concurrents
- performance (gains observés dans le code de Firefox)
- système de typage fort
- syntaxe « assez proche » du C
- interopérabilité avec le C
- sémantique assez déroutante aux premiers abords
  proche des langages fonctionnels (lisp, ocaml, Haskell)
- environnement riche (rustc, cargo, rustfmt, tests,...)

Hello, Idea !

`main.rs` :

```
1 fn main() {
2       println!("Hello, Idea!");
3 }
```

Compilation & exécution :

```
1 $ rustc main.rs
2 $ ./main
3 Hello, Idea!
4 $
```

Fibonacci - pattern matching + récursivité

```
1 // Recursive Fibonnaci
2 fn fib(n: u64) -> u64 {
3     match n {
4         0 => 1,
5         1 => 1,
6         _ => fib(n-1) + fib(n-2),
7     }
8 }
```

Variable - immuabilité

```
1 fn main() {
2     let x = 2; // immuable
3     x = 3;     // erreur
4 }
```

Variable mutable :

```
1 fn main() {
2     let mut x = 2; // mutable
3     x = 3;         // autorisé
4 }
```

Références

```
1 fn main () {
2     let v1 = vec![1, 2, 3]; // vecteur
3     let v2 = &v1;
4 }
```

Propriété (*ownership*)

- Une valeur a toujours un seul propriétaire (une variable).
- Si la valeur change de propriétaire, l'ancien propriétaire ne peut plus l'utiliser.
- Quand le propriétaire devient hors de portée, la valeur est libérée

```
1 fn main() {
2     let s1 = String::from("Hello");
3     let s2 = s1;
4     println!("{}, World", s1);   // erreur s1
5                                 // n'est plus accessible
6 }
```

Propriété - appel de fonction

- En passant une valeur en paramètre, on transfère sa propriété à la fonction

```
1 fn main() {
2     let s1 = String::from("test");
3     affiche(s1);    // transfère s1 à affiche
4     println!(s1);   // invalide
5 }
6
7 fn affiche(s: String) {
8     println!(s);
9 } // ici s est libérée (fin de la portée)
```

Propriété - transfert

Solution:

```
fn main() {
    let s1 = String::from("test");
    let s1 = affiche(s1);
    println!(s1);   // valide
}

fn affiche(s: String) -> String {
    println!(s);
    s       // rend s à l'appelant
}
```

Propriété - emprunt (*borrowing*)

Une référence permet d'emprunter une variable sans transférer la propriété

```
1 fn main() {
2     let s = String::from("un␣autre␣exemple");
3     let l = longueur(&s);
4     println!("la␣longueur␣de␣\"{}␣est␣{}\"", s, l);
5 }
6
7 fn longueur(s: &String) -> usize {
8     s.len()
9 }
```

```
5 $ ./t
6 la longueur de "un␣autre␣exemple" est 16
```

Hello Cargo

```
$ cargo new --bin hello
  Created binary (application) `hello` package
$ cd hello
$ cargo run
  Compiling hello v0.1.0 (/home/matthieu/prog/rust/hello
    Finished dev [unoptimized + debuginfo] target(s) in
     Running `target/debug/hello`
Hello, world!
$ emacs src/main.rs
$ cargo run
Hello, Idea!
```

Crates

Bibliothèques.

- dépendances listées dans `Cargo.toml`

`Cargo.toml`

```
[dependencies]
rand = "0.3.14"
```

`src/main.rs`

```
use rand::{thread_rng, Rng};

fn main() {
    let dice = thread_rng().gen_range(1, 7);
    println!("{}", dice);
}
```

Ressources

- The Rust Programming Language, Steve Klabnik and Carol Nichols, No Starch Press, 2018
- Rustlings, exercices de programmation en Rust.
- Rust: Systems Programmers Can Have Nice Things, Arun Thomas, EuroBSDCon 2019, Lillehammer, Norvège. (video)
- Corrode : traduction automatique de C vers Rust préservant la sémantique. Jamey Sharp (écrit en Haskell ...) .

Questions ?