Le langage Rust: concepts et exemples

François-Xavier Pineau[1]

[1]Centre de Données astronomiques de Strasbourg

04 avril 2019

Sommaire

Introduction

Exemples et Concepts

Conclusion

Introduction

Introduction

Exemples et Concepts

Conclusion

Origine

- Projet perso chez Mozilla
  - voir présentations de Graydon Hoare de 2010 et 2012
  - version 1.0 date du 15 mai 2015
- Motivations: programmation **concurrente**, **sûre/sécurisée**, statique et **pragmatique** (c.f. faq complémentaire de Rust)
- Rust combine:
  - contrôle de bas niveau: **performance**
  - fonctionalités de haut niveau: **confort d'utiliation**
    - abstractions zéro-coût
  - environnement **moderne**: Cargo, crates.io, doc markdown, tests
- Du neuf avec du vieux:
  - "PL design has >50 years of history"
  - "Most good ideas discovered in the first 20"
  - "we've tried hard to avoid incorporating new technology"

Caractéristiques

- **Communautaire** et **Open-source**: licences Apache / MIT
  - sponsorisé par Mozilla
- **Compilé**, sans ramasse miette, pas d'environement d'exécution
  - faible empreinte mémoire, faible temps de démarrage
  - vu de l'extérieur comme du C
  - => WebAssembly, Python, PostgreSQL, ...
- **Rapide** (comme C/C++), mais **sécurisé**
  - **Typage fort**, avec inférence de type et *shadowing*
  - pas de pointer pendouillant
  - pas de fuite mémoire (sauf en utilisant des concepts avancés)
- **Pragmatique**:
  - multi-paradigmes: procédural, OO, FP, actor (concurrence)
  - mécanismes de sécurité désactivable (*unsafe*)

Pourquoi pas?

- Langage préféré des développeurs sur *Stack Overflow* 3 années de suite (2016, 2017, 2018)
- Deployé sur des millions de machines (dans *Firefox*)
- Utilisé par: *Dropbox*, *npm*, *OVH*, *Cloudflare*, *Deliveroo*, ...
- Commence à être adopté dans la communauté du jeu vidéo
- Exemple de retour enthousiasmant
- Marketing vert: langage peu énergivore
  - systèmes embarqués

Motivations personnelles

- Quelques limites (dans mon usage) de Java
  - coût de lancement d'une JVM => démons
  - blocage lors d'un gros GC => perf non prédictibles
  - pas de pointeurs => création d'objets VS thread safety
  - pas de type primitif sur les collections
  - pas d'entier non signé
  - *mmap* limité à 2GB
  - ...
- Avec Rust, un même langage pour
  - exécution native **coté serveur** et/ou **CLI**
  - libraires utilisables dans des **applications Web**
  - appels depuis d'autres outils: **Python**, **PostgreSQL**, ...

Les contres

- Langage récent: syndrôme du paradoxe de l'oeuf et de la poule
- Courbe d'apprentissage raide au départ:
  - syntaxe riche: *println!()*, *read()?*, *&'static str*, ...
    - **mais** participe à la puissance du langage
  - nouveaux concepts à assimiler: *ownership*, *borrowing*, *lifetime*
    - **mais** assurent la sécurité
    - **et** permettent des optimisations
- Le *borrow checker* est intraitable
  - lire la doc avant de commencer!
  - apprendre à reconnaitre les erreurs
  - **mais** garantie la bonne marche du programme
  - **et** doc bien faite, avec de l'aide proposée par le compilateur

Tester/Installer Rust

- En ligne https://play.rust-lang.org/
- Installation https://www.rust-lang.org/install
  - `curl https://sh.rustup.rs -sSf | sh`
  - `mkdir mon_appli; cd mon_appli`
  - `cargo init`
  - `cargo build|run|test|doc|bench`
- Pour éviter les surprises sur les perfs:
  - `cargo build|run --release`

Ressources en ligne

- The book https://doc.rust-lang.org/book/
- Rust par l'exemple https://doc.rust-lang.org/rust-by-example/index.html
- Exercices:
  - https://github.com/crazymykl/rust-koans
  - https://github.com/rust-lang/rustlings
- The cargo book https://doc.rust-lang.org/cargo/
- Rust/WebAssembly
  - https://rustwasm.github.io/docs/book/
  - https://rustwasm.github.io/docs/wasm-bindgen/

Environnements de dévlpt.

- Editeurs de texte: Vim, Emacs, Gedit, ...
- IDEs: Eclipse, **IntelliJ**, Visual Studio Code, Atom

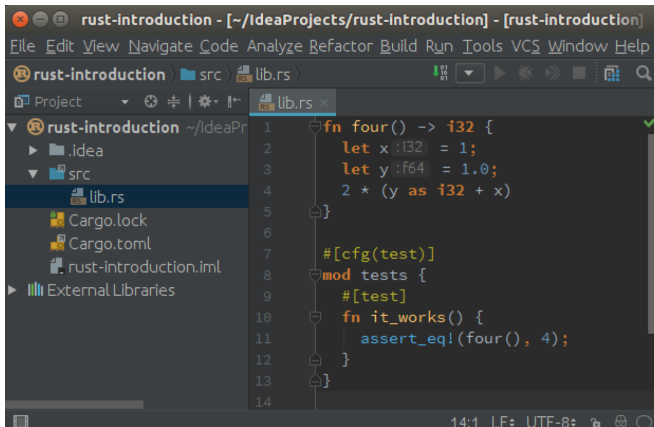

Figure 1: IntelliJ affiche les types inférés

Exemples et Concepts

Introduction

Exemples et Concepts

Conclusion

Macros 1/6

- Voir the book
  - métaprogrammation: du code qui écrit du code
- Macro déclarative ≈ *template* de code
  - fonction qui finit par un '`!`'
  - macro usuelles:
    - `format!`, `print!`, `println!`, `eprint!`, `eprintln!`, `write!`
    - `assert!`, `assert_eq!`, `debug_assert!`, `debug_assert_eq!`
    - `panic!`, `unreachable!`, `dbg!`

      ```
      let a = dbg!(ma_function(...));
      if dbg!(a == b) { ... }
      ```
  - corriger le code ci-dessous: lien playground

```
fn main() {
  let a = "Hello";
  let b = "word";
  println("{}, {}!", a, b);
}
```

Macros 2/6

- Exemple de macro déclarative

```
macro_rules! replace_attribute_if_some {
  ($sel:ident, $other:ident, $attr_name:ident) => {{
    if let Some(val) = $other.$attr_name {
      $sel.$attr_name  = val;
    }
  }};
}
[...]
fn update(&mut self, other: &MaStruct) {
  replace_attribute_if_some!(self, other, attr_1);
  replace_attribute_if_some!(self, other, attr_2);
  ...
  replace_attribute_if_some!(self, other, attr_n);
}
```

Macro 3/6

- Macro procédural = code qui génére du code
  - corriger le code ci-dessous: lien playground

```rust
struct Coo {
    lon: f64,
    lat: f64,
}

fn main() {
  let coo = Coo { lon: 3.5, lat: 4.5};
  println!("Debug: {:?}", coo);
  println!("Pretty debug: {:#?}", coo);
}
```

Macro 4/6

- L'exemple de la crate (librairie) `serde` (SERialize DEerialize)
  - corriger le code ci-dessous: lien playground

```
#[derive(Serialize, Deserialize)]
struct Coo {
    lon: f64,
    lat: f64,
}
fn main() -> Result<(), Error> {
    let data = r#"
        { "lon": 3.5, "lat": "4.5"}
    "#;
    let coo: Coo = serde_json::from_str(data)?;
    println!("{}", serde_json::to_string_pretty(&coo)?);
    Ok(())
}
```

Macro 5/6

- Example de choix de code à la compilation

```
#[cfg(all(any(target_arch = "x86",
              target_arch = "x86_64"),
          target_feature = "bmi2"))]
fn ij2h(i: u32, j: u32) -> u64 {
  #[cfg(target_arch = "x86")]
  use std::arch::x86::_pdep_u32;
  #[cfg(target_arch = "x86_64")]
  use std::arch::x86_64::_pdep_u32;
  unsafe {
    (_pdep_u32(i, 0x55555555_u32)
   | _pdep_u32(j, 0xAAAAAAAA_u32)) as u64
  }
}
```

Macro 6/6

- Crate StructOpt pour faire des CLI

```
#[derive(Debug, StructOpt)]
#[structopt(name = "ma_commande")]
/// Ma premiere commande en Rust
/// Exemple: ./ma_commande -l ra -b dec input.txt
struct Args {
  #[structopt(short = "l", long, default_value = "0")]
  /// Index or Name of the column containing the longitude
  lon: String,
  #[structopt(short = "b", long, default_value = "1")]
  /// Index or Name of the column containing the latitude
  lat: String,
  #[structopt(parse(from_os_str))]
  /// Input file (stdin if not present)
  input: Option<PathBuf>,
}
```

Pas de NULL!

- Mais un type Option (monade en prog. fonctionnelle)
  - corriger le code ci-dessous: lien playground

```
fn main() {
    let a: Option<u64> = Some(1);
    let b: Option<u64> = None;
    let a_str = match a {
        Some(val) => val.to_string().as_str(),
        None => "null",
    };
    println!("a: {}", a_str);
    if let Some(val) = b {
        println!("b: {}", val);
    }
    println!("b: {}", b.unwrap_or(-1));
}
```

Ownership 1/2

- Une valeur appartient à une seule variable à la fois (Owner)
- La valeur est nettoyée quand la variable sort du scope
  - corriger le code ci-dessous: lien playground

```
#[derive(Debug)]
struct Coo (f64, f64);

fn main() {
  let a1 = 1;
  let a2 = a1;
  let coo1 = Coo(3.5, 4.5);
  let coo2 = coo1;
  println!("{}, {:?}", a1, coo1);
  println!("{}, {:?}", a2, coo2);
}
```

Ownership 2/2

- Exemple de transfert de propriété à une fonction
  - corriger le code ci-dessous: lien playground

```rust
#[derive(Debug)]
struct Coo (f64, f64);

fn take_ownership(coo: Coo) -> Coo {
    if coo.0 > 0.0 { coo } else { Coo(-coo.0, coo.1) }
}

fn main() {
    let coo_1 = Coo(3.5, 4.5);
    let coo_2 = take_ownership(coo_1);
    println!("{:?}", coo_1);
}
```

Borrowing 1/2

- Transfert de propiété: pas la seule solution
- On peut préter/emprunter (borrow) une valeur via
  - soit un ou plusieurs pointeurs en lecture
  - soit un pointeur en écriture
- Corriger le code ci-dessous: lien playground

```
fn main() {
  let v1 = vec![0, 1, 2];
  let v2 = v1;
  println!("{:?}, {:?}", v1, v2);
}
```

Borrowing 2/2

- Corriger le code ci-dessous: lien playground

```
fn add(vec: &mut Vec<i32>, val: i32) {
  vec.push(val);
}
fn new_vec() -> &Vec<i32> {
  let v = vec![0, 1, 2, 3];
  &v
}
fn main() {
  let v = vec![0, 1, 2];
  add(v, 3);
  assert_eq!(v, new_vec());
}
```

Lifetimes

- `let my_str: &'static str = "toto";`
- Fonction qui retourne un pointeur => lifetime
  - corriger le code ci-dessous: lien playground

```
fn largest(v1: &Vec<i32>, v2: &Vec<i32>) -> &Vec<i32> {
  if v1.len() < v2.len() { v1 } else { v2 }
}

fn main() {
  let v1 = vec![0, 1, 2];
  let v2 = vec![0, 1, 2, 3];
  let largest = largest(&v1, &v2);
  println!("{:?}", largest);
}
```

Fonctionnel / parallélisme

Corriger le code ci-dessous: lien playground

```
use rayon::prelude::*;
fn main() {
  let v1 = ["-10.0", "14.0", "", "23.0", "NaN", "30.0"];
  let v2: Vec<f64> = v1.par_iter()
     .filter_map(|s| s.parse::<f64>().ok())
     .filter(|x| x.is_finite())
     .collect();
  let sum: f64 = v2.iter()
     .map(|x| x.to_radians().sin())
     .sum();
  println!("mean: {}", sum / v2.len());
}
```

Trait

Corriger le code ci-dessous: lien playground

```
trait Accumulable: Copy {
    fn identity() -> Self;
    fn add(self, rhs: Self) -> Self;
}

impl Accumulable for f32 {
    fn identity() -> Self { 0.0 }
    fn add(self, rhs: Self) -> Self { self + rhs }
}

struct Accumulateur<T> where T: Accumulable {
    n: i32,
    sum: T,
}
```

Trait

```
impl<T> Accumulateur<T> where T: Accumulable {
    pub fn new() -> Accumulateur<T> {
        Accumulateur { n: 0, sum: T::identity() }
    }
    pub fn add(&self, elem: T) { self.sum = self.sum.add(el
    pub fn n(&self) -> i32 { self.n }
    pub fn sum(&self) -> T { self.sum }
}
fn main() {
    let v = vec![1.0, 2.0, 3.0];
    let acc = Accumulateur::<f32>::new();
    for val in v.into_iter() {
        acc.add(val);
    }
    println!("n: {}, sum: {}", acc.n(), acc.sum());
}
```

Tests et Documentation

- Voir exemple de cdshealpix
- Corriger le code ci-dessous (en mode test): lien playground

```rust
/// Doc écrite en markdown
/// ```rust
/// use playground::is_positive;
/// assert!(is_positive(1));
/// assert!(is_positive(-1));
/// ```
pub fn is_positive(val: i32) -> bool { val >= 0 }

#[test]
fn test_is_positive() {
  assert_eq!(is_positive(1), true);
  assert_eq!(is_positive(-1), false);
}
```

Conclusion

Introduction

Exemples et Concepts

Conclusion

Conclusion

- Quelques points (biaisés) abordés ici
- Plein d'autres choses:
  - Erreurs récupérables / non-récupérables
  - `Enum`, `tuple`, `tuple struct`, `struct`
  - `type`, type associé, `PhantomData`
  - trait: `Deref`, `Drop`, ...
  - Pointeurs intelligents: `Box`, `Cell`, `Arc`, `Cow`, ...
  - Multi-threading, Future
  - ...
- Langage riche et souple, très bien documenté!
- **Sensation de liberté!!**
- Mais: être motivé pour passer les possibles frustrations du début!