Ambient IT
128 Boulevard Macdonald
75019 Paris

Mail : formation@ambient-it.net
Tél : +33 (0)1 40 34 11 53

AMBIENT-IT

Formation Rust

Durée 3 jours (21 heures)

Présentation

Rust est un nouveau langage de programmation Open Source créé par Mozilla Research, conçu pour aider les développeurs à concevoir des applications ultra-rapides et sécurisées, utilisant avantageusement la puissance offerte par les processeurs multi-core modernes. Il permet d'éviter les erreurs de segmentation et assure la sécurité des threads, le tout avec une syntaxe élégante. Il a de très bonnes performances via sa gestion de la mémoire faite au niveau du compilateur et une compilation native.

Rust offre un haut niveau d'abstractions, garanti la sécurité de la mémoire, des threads sans problème de concurrence, des templates : trait-based generics, du pattern matching, l'inférence de type, et un binding C efficace, le tout avec runtime minimal !

Rust est donc un langage de programmation axé sur la sécurité, la vitesse et la concurrence. Sa conception vous permet de créer des programmes qui ont la performance et le contrôle d'un langage de bas niveau tel que le C, mais avec les abstractions puissantes d'un langage de haut niveau (C#, Java...). Ces propriétés rendent Rust adapté aux programmeurs qui ont de l'expérience dans des langages comme C/C++ mais qui recherchent une alternative plus sûre, ainsi que ceux qui utilisent des langages tels que Python mais qui recherchent des moyens d'écrire un code qui fonctionne plus efficacement sans sacrifier l'expressivité.

La devise de Rust est :Performance, Sécurité, Productivité, choisissez-en trois !

À travers cette formation, vous apprendrez à utiliser tous les concepts de Rust à savoir : borrowing, ownership, lifetimes, mutabilité, génériques et bien d'autres choses afin de développer efficacement vos prochains projets !

Comme dans toutes nos formations, celle-ci vous présentera la toute dernière version de Rust Programming Language, Rust 1.53.

Objectifs

- Découvrir, comprendre et utiliser le langage Rust et son écosystème dans vos futurs projets
- Utiliser la programmation concurrente et la bonne gestion d'erreur avec Rust
- Migrer efficacement vos applications C ou C ++ vers un environnement Rust

Public visé

Développeurs, Architectes

Pré-requis

- Connaissances de base d'un système Unix
- Connaissance de programmation (C, C#, Java, C#)

POUR ALLER PLUS LOIN

- Découvrez le couplage de Rust et de la technologie de Microsoft WebAssembly avec notre Formation WebAssembly avec Rust

Programme de la formation Rust

Introduction à Rust

- Téléchargement et installation de l'environnement Rust
    - rustup, cargo, rustc, crates.io
- Mon premier programme
- Ecrire vos tests unitaires : unit tests
- Command-line arguments
- Ecrire son web-serveur

Les bases de la programmation concurrente

- Définition : Mandelbrot
- Parsing pair : command-line arguments
- Mapping pixels : complex numbers
- Tracer : Plotting the set
- Ecriture d'un fichier image
- Mandelbrot : Concurrent program
- Exécuter : Mandelbrot plotter
- Sécurité

Types basique : Basic types

- Machine types
    - Integer, Floating-point, bool, Characters types

- Tuples
    - Pointer types
    - References
    - Boxes
    - Raw pointers
- Arrays, Vectors, and Slices
- String types

Ownership

- Ownership
- Moves : operations, control flow, indexed content
- Copy types : exception
- Rc & Arc: shared ownership

Les conditions

- if, else if et else
- Les loops (loop, for et while)

References

- References as values
- Rust references VS C++ references
- Assignation, comparaison, Null
- References to slices and trait objects
- Passing & Returning
- Structs
- Lifetime parameters
- Sharing VS mutation

Expressions

- Blocks & semicolons
- Declarations
- if and match, if let
- Loops
- Return expressions
- Function & method calls
- Fields & elements
- Reference operators
- Arithmetic, bitwise, comparison, and logical operators
- Assignment, Type casts, Closures
- Precedence & Associativity

Gestion d'erreur

- Panic
- Aborting, Result, Catching, Ignoring
- Result type aliases
- Propagating errors
- Manipuler plusieurs types d'erreurs
- Custom error type

Tools : Crates and modules

- Crates : Build profiles
- Modules : separate files
- Visibility
- Paths & imports
- The standard prelude
- Items : building blocks
- Library
- Répertoire src/bin
- Attributes
- Tests & documentation
- Integration tests & Unit test
- Benchmarks, Fuzzing
- Doc-tests
- Specifying dependencies
- Versions
- Cargo.lock
- Publication : crates.io
- Workspaces

Structs

- Named-field structs
- Tuple-like, Unit-like structs
- Struct layout
- Définir : methods implémentation
- Generic structs
- Lifetime parameters
- Héritage : Traits for struct types
- Mutabilité interne

Enums & patterns

- enums : Rich data structures using
- Generic enums
- Patterns
    - Literals, variables, wildcards
    - Reference, Tuple & struct patternsMatching multiple, guards, @ patterns

- Remplir : binary tree

Traits & Generics

- Trait objects, layout
- Drop, Sized, Clone, Copy
- Deref & DerefMut, Default
- AsRef & AsMut
- Borrow & BorrowMut
- From & Into
- Borrow & ToOwned
- Self in traits
- Subtraits
- Static methods
- Associated types (or, how iterators)
- Generic traits (or, how operator overloading)
- Buddy traits (or, how rand::random())
- Reverse-engineering bounds

Surcharge d'opérateurs

- Operator overloading
- Arithmetic and bitwise operators
- Unary, Binary operators
- Affectation : Compound assignment operators
- Egalité : Equality tests
- Ordered comparisons
- Index & IndexMut
- Autre opérateurs

Closures

- Capturing variables
- Closures : Borrow, steal, function & types
- Performance
- Sécurité
- FnOnce & FnMut
- Callbacks

Iterateurs

- Iterator & IntoIterator traits
    - iter & iter_mut methods
    - drain methods

- Adapters
  - Map & filter
  - filter_map & flat_map
  - scan
  - take & take_while, skip & skip_while
  - peekable, fuse
  - Reversible iterators and rev
  - inspect, chain, enumerate, zip
  - by_ref, cloned, cycle
- Consuming iterators
  - Accumulation : count, sum, product
  - max, min, max_by, min_by
  - max_by_key, min_by_key
  - Comparaison : item sequences
  - any & all
  - position, rposition, & ExactSizeIterator
  - fold, nth, last, find
  - collect & FromIterator
  - Extend trait, Partition
- Implémenter son propre itérateur

Collections

- Vec<T>
- Accessing elements
- Iteration
- Growing & shrinking vectors
- Joining, Splitting, Swapping
- Sorting & searching
- Comparing slices
- Random elements
- Règles : Rust rules out invalidation errors
- VecDeque<T>, LinkedList<T>, BinaryHeap<T>
- HashMap<K, V> and BTreeMap<K, V>
- Entries
- Map iteration
- HashSet<T> and BTreeSet<T>
- Set iteration
- Hashing & custom hashing algorithm

Strings & Text

- ASCII, Latin-1, Unicode , UTF-8
- Conversions : characters, integers, digits
- String and str
- Inspection
- Appending, inserting, Removing text
- Conventions for searching and iterating
- Patterns : Searching & replacing
- Iterating over text, Trimming
- Case conversion for strings
- Parsing other types from strings
- Converting other types to strings
- Borrowing as other text-like types
- Strings as generic collections

- Formatting values, text, numbers, custom...
- Debugging
- Regular expressions
- Basic Regex use & lazily
- Normalization forms, unicode-normalization crate

Input & output

- Readers & Writers : Files, Directories, Seeking
- Binary data, compression, serialization
- OsStr & Path
- Path & PathBuf methods
- Filesystem access functions
- Platform-specific features
- Pile réseau : Networking

Concurrency

- Fork-join parallelism, spawn & join, Error
- Sharing immutable data, Rayon, Mandelbrot
- Channels, Sending, Receiving values
- Pipeline, Channel features & performance
- Thread safety: Send and Sync
- Iterator channel
- Partager : Shared mutable state
- Mutex in Rust : mut & Mutex
- Deadlock, Poisoned mutexes
- Multi-producer channels
- Read-write locks (RwLock)
- Condition variables (Condvar)
- Atomics, Global variables

Macros

- Expansion, Repetition, Debugging, Error
- Utilisation : Json, Fragment types, Recursion, Traits with macros
- Bonnes pratiques : Scoping
- Import and export

Unsafe code

- Unsafe blocks
- Exemple : ASCII string type
- Unsafe functions, traits, Raw pointers
- Exemple: RefWithFlag
- Nullable pointers

- Type sizes & alignments
- Pointer arithmetic
- Moving into and out of memory
- Exemple : GapBuffer
- Interop : Foreign functions, calling function lib C & C++
- Common data representations
- Déclaration : foreign functions & variables
- Raw interface libgit2, Safe interface libgit2

Sociétés concernées

Cette formation s'adresse aux entreprises, petites ou grandes, souhaitant former ses équipes à une nouvelle technologie informatique avancée.

Méthodes pédagogiques

Stage Pratique : 60% Pratique, 40% Théorie. Support de la formation distribué au format numérique à tous les participants.

Organisation

Le cours alterne les apports théoriques du formateur soutenus par des exemples et des séances de réflexions, et de travail en groupe.

Validation

À la fin de la session, un questionnaire à choix multiple permet de vérifier l'acquisition correcte des compétences.

Sanction

Une attestation sera remise à chaque stagiaire qui aura suivi la totalité de la formation.