LinuxDay Torino 20181027

Rust, Introduzione in breve

Luca Barbato - lu_zero@gentoo.org

Rust

Cos'e` Rust?

> "Un linguaggio di sistema decisamente veloce che previene segfault, ed evita buona parte dei problemi che si hanno quando si scrive codice multithread"

- Programmazione generica
  - Implementata tramite `Trait`
  - Astrazione a costo 0
- Accesso e Gestione della memoria garantiti a compile-time
  - Semantica `move`
  - Thread senza `data race`
- Pattern matching
- ABI-compatibile con C

Programmazione Generica

Funzioni

```
fn takes_anything<T>(x: T) {
    // do something with x
}

fn takes_something_we_can_print<T : Display>(x: T) {
    println!("{}", x);
}

fn takes_something_thread_safe<T>(x: T)
    where T: Send + Sync {
    // do something with x
}
```

Programmazione Generica

Strutture

```
struct Point<T> {
    x: T,
    y: T,
}
```

Enum

```
enum Option<T> {
    Some<T>,
    None
}
```

Programmazione Generica

```
trait GetSet<Bar> {
    fn set(&mut self, &Bar);
    fn get(&self) -> Bar;
}

trait Pull {
    type T;

    fn pull(&mut self) -> Self::T;
}

fn foo<T>(foo: &mut T)
    where T: GetSet<i32>
{
    let f = foo.get();
    foo.set(&f);
}

fn pull<P: Pull>(p: &mut P) -> P::T {
    p.pull()
}
```

Gestione della memoria

Principi

- Le variabili sono comunemente immutabili
- E` possibile accedere in scrittura solo da una singola variabile per volta.
- Le variabili di default adottano una semantica `move` (a differenza dei linguaggi in cui la semantica predefinita e` `copy`)
- Una volta che una variabile esce dallo `scope` viene chiamata la sua `drop()`.

Gestione della memoria

Semantica move

```
struct Something {
    x: u8,
}

fn do_stuff() {
    let mut a = 0;
    let c = 1;
    let d = Something { x: c };
    let d_ = d;

    a = 42;
    // Errore, c e` immutabile.
    c = 2;

    // Errore, d non esiste piu`, e` diventato d_.
    println!("{:?} -> {:?}", d.x, d_.x);
}
```

Gestione della memoria

Tempo di vita

```
fn b<'a>(x: &'a [u8], y: &'a [u8]) -> &'a [u8] {
    if x.is_empty() && y.is_empty() {
        panic!("Entrambe vuote")
    } else if x.is_empty() {
        y
    } else {
        x
    }
}
```

Il compilatore a volte ha bisogno che il codice sia annotato per poter garantire che le variabili vivano abbastanza a lungo.

Codice di alto livello

- Iteratori (lazy)
- Closure

```
fn iteratori(inc: u8) -> Vec<u8> {
    let a = 0..10;

    println!("Vettore {:?}", a);

    let it = a.map(|&v| {
        v + inc
    }).filter(|&v| (v & 1) != 0);

    it.collect()
}
```

Codice di basso livello

```
#[inline(always)]
pub unsafe fn syscall1(n: usize, a1: usize) -> usize {
    let ret : usize;
    asm!("syscall" : "={rax}"(ret)
                   : "{rax}"(n), "{rdi}"(a1)
                   : "rcx", "r11", "memory"
                   : "volatile");
    ret
}
```

Interoperabilita` da C

```
extern crate libc;
use libc::c_int;

#[link(name = "libfoo")]
extern {
    fn foo(arg: c_int) -> c_int;
}

fn main() {
    let x = unsafe { foo(100) };
    println!("foo: {}", x);
}
```

Interoperabilita` verso C

```
#[repr(C)]
struct Point {
    x: i32,
    y: i32,
}

extern "C" fn new(new: *mut Point, x: i32, y: i32) {
    println!("I'm called from C with value {} {}", x, y);
    unsafe {
        *new = Point { x: x, y: y };
    }
}
```

Codice SIMD

```
#[target_feature(enable = "avx2")]
#[cfg(any(target_arch = "x86", target_arch = "x86_64"))]
unsafe fn hex_encode_avx2<'a>(
    mut src: &[u8], dst: &'a mut [u8],
) -> Result<&'a str, usize> {
    let ascii_zero = _mm256_set1_epi8(b'0' as i8);
    let nines = _mm256_set1_epi8(9);
    let ascii_a = _mm256_set1_epi8((b'a' - 9 - 1) as i8);
    let and4bits = _mm256_set1_epi8(0xf);
```

Toolchain

rustup

- Permette di installare toolchain rust in parallelo
- Simile a `RVM` o `pyenv`

```
$ curl https://sh.rustup.rs -sSf | sh
```

Toolchain

rustc

- Basato su LLVM
  - Cross Compilatore inerente: Dove funziona clang funziona rustc
  - Primitive SIMD
  - Strumentazione (ASAN, profiling, ..)
- Attenzione su warning ed errori **comprensibili**

Toolchain

cargo

- Package manager e build system
- Compila ed installa sorgenti da repository (e.g. `crates.io`)
- Test e benchmark
- Estensibile (e.g. integrazioni `wasm`)

```
$ cargo +nightly test
```

```
$ cargo web deploy
```

```
$ cargo build --target powerpc64le-unknown-linux-musl
```

```
$ cargo kcov
```

Perche` piace?

- Buone performance, spesso migliori di C a parita` di fatica impiegata.
- Meno problemi a cui dover badare, diverse tipologie sono semplicemente impossibili: il compilatore ti impedisce di fare quegli sbagli in partenza.
- Ottimi strumenti di sviluppo
- Buona integrazione con codice non-rust in ambo le direzioni

Dove ha piu` senso usarlo?

- Nessun linguaggio e` sempre **LA** soluzione ad ogni problema.
- Rust e` una buona soluzione ove vi sia necessita` di dover gestire concetti **complessi** ed al contempo avere buone performance.
- Rust rende molto facile scrivere codice multithread e le strutture dati fornite dalla libreria standard sono spesso ottimali per risolvere problemi comuni.

Dove viene usato con successo?

Piccoli programmi utili

Dato che in rust scrivere codice multithread e` decisamente semplice ci sono diverse reimplementazioni di utility quali `grep` o `find`.

ripgrep e` piu` veloce degli altri sostituti di *grep* e fornisce un buon numero di feature aggiuntive.
fd-find oltre ad essere veloce rompe completamente la compatibilita` con *find* per migliorarne l'usabilita`.
ion e` una *shell* posix-like ma non posix-compatibile con diverse caratteristiche, fra cui l'essere molto robusta a bachi come quello che ha portato a shellshock.

Parsing di formati

Rust ha molteplici librerie per fare parsing:

- pest
- nom
- serde

E sono state usate con successo per sostituire parser scritti in altri linguaggi ottenendo sia maggiore robustezza sia maggiori performance.

Firefox e, prossimamente, VLC utilizzano ed utilizzeranno demuxer scritti in rust in questo modo.

Browser web

Servo e` uno degli esempi di come rust renda piu` semplice gestire problemi complessi.

Diverse componenti di `Servo` sono gia` parte di `Firefox` e `Firefox Reality` grazie al progetto `Quantum`.

Sistemi operativi

Redox e` un sistema operativo completo scritto quasi interamente in rust, include anche una reimplementazione di `libc` scritta in rust, relibc.

Multimedia

Il Multimedia e` un altro ambito in cui vi e` la necessita` di avere velocita` e controllo su quanto si scrive rust-av e` il mio esperimento a riguardo e rav1e e` l'encoder AV1 che presentero` nella prossima mezz'ora.