Introduction To Rust

"The type safety of Haskell, the concurrency of Erlang, and the speed of C++"

About

github.com/jeanCarloMachado

twitter.com/JeanCarloMachad

jeancarlomachado.com.br

compufacil.com.br

Rust

Generic multiparadigm system's programming language.

Rust

Generic multiparadigm system's programming language.

Version 1.0 in May of 2015, 1.10 currently.

Rust

Generic multiparadigm system's programming language.

Version 1.0 in May of 2015, 1.10 currently.

Platforms:

- ARM
- Apple
- MIPS
- PC
- Raspberry PI
- TIVA
- Tessel 2
- Others

Who Built?

Mozilla Research team (2009)

- Graydon Hoare https://github.com/graydon
- Brian Anderson https://github.com/brson
- Patrick Walton https://github.com/pcwalton
- Alex Crichton https://github.com/alexcrichton

Built for

Patrick Walton: Rust should focus on the same domain as C++ does.

- Browser Engines
- Games
- IOT
- OS's

Who Uses?

- Servo: Browser engine
- Dropbox
- Coursera
- Skylight: New relic for rails
- Redox: Unix based OS
- Tikv: Distributed database
- Many more

In the wild

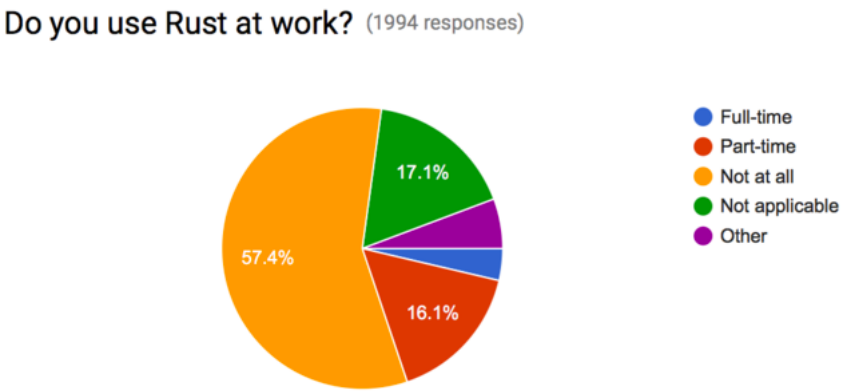

Figure 1:who uses

Interest

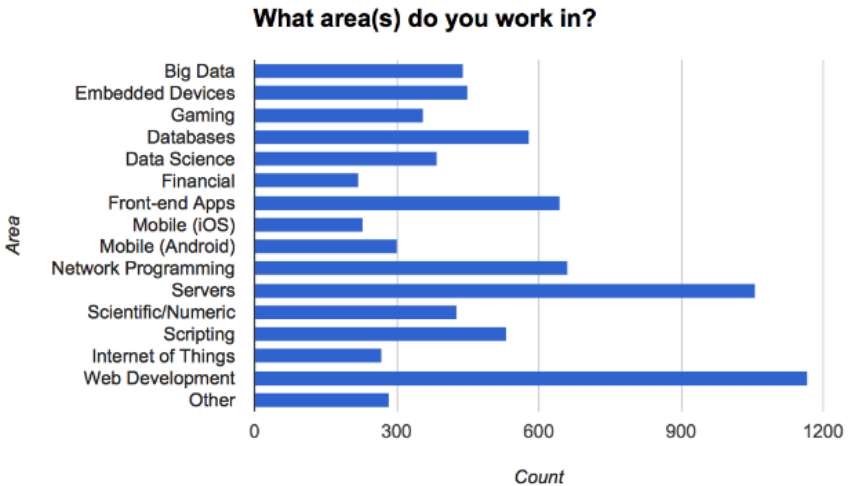

Figure 2:Interest

Strengths

- Security
- Performance
- Fine grained control

Weaknesses

- Fight the borrow checker
- Huge syntax
- Relatively new

Rust vs the world

Go

- fast compilation
- simplicity
- imperative foundations
- higher level (Haskell,OCaml,C#,F#)

Rust

- slow compilation
- long list of features
- functional foundations
- lower level (C++,C,D)

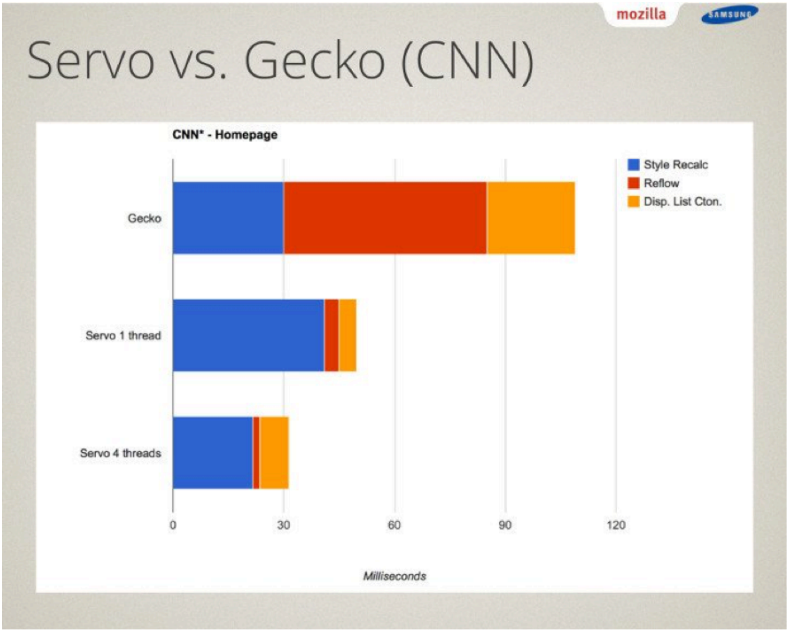

Figure 3:Servo performance

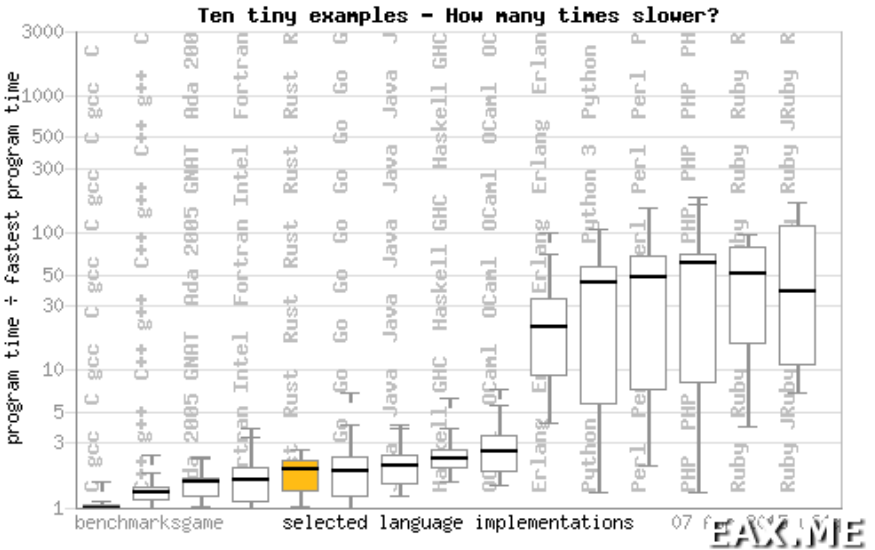

Figure 4:Performance

rustc

- Written in Rust
- Uses LLVM
- MIR
- Zero cost abstractions

Syntax

```rust
fn fib(n: u64) -> u64 {
    match n {
        0 | 1 =>  n,
        n => fib(n - 1) + fib(n - 2)
    }
}
```

```
fn gcd(mut n: u64, mut m: u64) -> u64 {
    assert!(n != 0 && m != 0);
    while m != 0 {
        if m < n {
            let t = m; m=n; n=t;
        }

        m = m % n;
    }
    n
}
```

Enums

```
enum Elements  {
    Square,
    Circle
}
```

Structs

```
struct Point3d {
    x: i32,
    y: i32,
    z: i32,
}
let mut point = Point3d { x: 0, y: 0, z: 0 };
```

Traits

```
struct Circle {
    x: f64,
    y: f64,
    radius: f64,
}
trait HasArea {
    fn area(&self) -> f64;
}
impl HasArea for Circle {
    fn area(&self) -> f64 {
        std::f64::consts::PI *
        (self.radius * self.radius)
    }
}
```

Generics

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

enum Result<T, E> {
   Ok(T),
   Err(E),
}
```

Generics

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

enum Result<T, E> {
   Ok(T),
   Err(E),
}
```

```
let foo: Option<f64> = Some(5.0f64);
let foo: Option<u8> = None;
let foo: Result<u8> = Ok(8);
let foo: Result<i16> = Err(9);
```

Generic Functions

```
fn min<T: Ord>(a: T, b: T) -> T {
    if a <= b { a } else  { b }
}
```

Generic Functions

```
fn min<T: Ord>(a: T, b: T) -> T {
    if a <= b { a } else  { b }
}
```

Inline specialization

```
min(10i8, 20) == 10; //T is i8
min(10, 20u32) == 10; //T is u32
min("abc", "xyz") == "abc"; //strings are Ord
```

Composite types dependency

```
use std::fmt::Debug;
fn foo<T: Clone + Debug>(x: T) {
    x.clone();
    println!("{:?}", x);
}
```

Refined implementation filter

```
impl<R: Num, C: Num> Add<Matrix<R, C>>
    for Matrix<R, C> {
...
}
```

Lazy evaluation

```
(1..)
.filter(|&x| x % 2 == 0)
.find(|x| *x  > 42)
.take(5)
.collect::<Vec<i32>>();
```

Imperative

```
let mut file = match File::open(file_path) {
    Ok(file) => file,
    Err(err) => return Err(err.to_string()),
};
let mut contents = String::new();
if let Err(err) =
    file.read_to_string(&mut contents) {
        return Err(err.to_string());
    }
let n: i32 = match contents.trim().parse() {
    Ok(n) => n,
    Err(err) => return Err(err.to_string()),
};
Ok(2 * n)
```

Functional

```
File::open(file_path)
.map_err(|err| err.to_string())
.and_then(|mut file| {
  let mut contents = String::new();
  file.read_to_string(&mut contents)
      .map_err(|err| err.to_string())
      .map(|_| contents)
})
.and_then(|contents| {
  contents.trim().parse::<i32>()
          .map_err(|err| err.to_string())
})
.map(|n| 2 * n)
```

Memory

Rust way

- No null pointer deferences.
- No dangling pointers
- No buffer overruns

Garbage collector

- Non deterministic
- Runtime checking
- Lack of control

Garbage collector

- Non deterministic
- Runtime checking
- Lack of control

Out of scope deallocation

Ownership

```
let v = vec![1, 2, 3];
let v2 = v;
println!("v[0] is: {}", v[0]);
```

Ownership

```
let v = vec![1, 2, 3];
let v2 = v;
println!("v[0] is: {}", v[0]);
```

```
error: use of moved value: `v`
println!("v[0] is: {}", v[0]);
```

Ownership

```
let v = vec![1, 2, 3];
let v2 = v;
println!("v[0] is: {}", v[0]);
```

```
error: use of moved value: `v`
println!("v[0] is: {}", v[0]);
```

```
let v2 = &v;
```

Borrowing

```
let v = vec![];
foo(&v);
fn foo(v: &Vec<i32>) {
      println!("{}", v); //works
      v.push(5); //fails
}
```

Borrowing

```
let v = vec![];
foo(&v);
fn foo(v: &Vec<i32>) {
     println!("{}", v); //works
     v.push(5); //fails
}
```

```
error: cannot borrow immutable borrowed content `*v` as mutable
v.push(5);
```

Borrowing Mutable

```
let mut x = 5;
{
    let y = &mut x;
    *y += 1;
}

println!("{}", x);
```

Borrowing Rules

- a borrowed value can't have greater lifetime
- one or more immutable references (&T) to a resource,
- exactly one mutable reference (&mut T).

Lifetimes

```
fn firsts<'a 'b> (x: &'a Vec<i32> y: &'b Vec<i32>)
    -> (&'a i32, &'b i32)  {
        return (&x[0], &y[0]);
}
```

Concurrency

Shared mutable state (root of all evil)

Concurrency

Shared mutable state (root of all evil)

4 feet tall

Concurrency

Shared mutable state (root of all evil)

4 feet tall

Other languages tries to solve the mutable.

Concurrency

Shared mutable state (root of all evil)

4 feet tall

Other languages tries to solve the mutable.

Rust solves the shared

Default thread

```
let thread1 = std::thread::spawn(|| {
    println!("foo");
    return 999;
});

let thread2 = std::thread::spawn(|| {
    println!("bar");
    return 666;
});


assert_eq!(try!(thread.join()), 999);
assert_eq!(try!(thread.join()), 666);
```

Shared thread

```
fn main() {
    let mut x = 1;
    let thread1 = std::thread::scoped(|| {x + 8});
    let thread2 = std::thread::scoped(|| {x + 27});
    assert_eq!(thread1.join() + thread2.join(), 37);
}
```

Shared mutable thread

```
//arc = safe reference count
//mutex = mutable data with locking
let data = Arc::new(Mutex::new(vec![1,2,3]));

for i in 0..3 {
    let data = data.clone();
    thread::spawn(move || {
        let mut data = data.lock().unwrap();
        data[0] += i;
    });
}

thread::sleep(Duration::from_millis(50));
```

Tests

```
fn prime_factors(mut num: i64) -> Vec<i64> {
    let mut result = vec![];
    let mut i  = 2;
    while num > 1 {
        while num % i == 0 {
            result.push(i);
            num /= i;
        }
        i +=1;
    }

    result
}
```

```rust
#[test]
fn prime_factors_of_48() {
    assert_eq!(prime_factors(48), [2, 2, 2, 2, 3]);
}
```

Tests

```
#[test]
#[should_panic(expected = "assertion failed")]
fn it_works() {
    assert_eq!("Hello", "world");
}
```

Tests as Documentation

```
//! ```
//! assert_eq!(4, adder::add_two(2));
//! ```
```

Macros

```
let x: Vec<u32> = vec![1, 2, 3];
```

Macros

```
let x: Vec<u32> = vec![1, 2, 3];
```

```
let x: Vec<u32> = {
    let mut temp_vec = Vec::new();
    temp_vec.push(1);
    temp_vec.push(2);
    temp_vec.push(3);
    temp_vec
};
```

Implementation

```
macro_rules! vec {
    ( $( $x:expr ),* ) => {
        {
            let mut temp_vec = Vec::new();
            $(
                temp_vec.push($x);
            )*
            temp_vec
        }
    };
}
```

Cargo

Package manager

run/build/test/bench/update/init

Libs

- rustup - croscompilling toolset
- clap - parse command line args
- iron - Web framework
- helix - ruby classes in Rust
- racer - auto-complete-er

Beyond

- Module system
- FFI Foreign functions interfaces
- Unsafe
- Much more!

Cool Resources

- Blog: https://blog.rust-lang.org/
- Rust by Example: http://rustbyexample.com/
- Rust Book: https://doc.rust-lang.org
- This week in Rust https://this-week-in-rust.org
- Writing an OS in Rust: http://os.phil-opp.com/

References

- Platforms:
  https://hacks.Mozilla.org/2015/05/diving-into-rust-for-the-first-time/
- Comparison: https://www.rust-lang.org/faq.html#other-languages
- Rust Programming Language:
  http://www.slideshare.net/jaejukim9/rust-programming-language
- State of Rust:
  https://blog.rust-lang.org/2016/06/30/State-of-Rust-Survey-2016.html
- Why Rust:
- Which programs are fastest:
  http://benchmarksgame.alioth.debian.org/u64q/which-programs-are-fastest.html
  http://www.oreilly.com/programming/free/files/why-rust.pdf

exit(0)