The Rust Programming Language

Tim Hegemann

January 13, 2017

Outline

Facts and Features

The Rust Programming Language

- Invented by Graydon Hoare at Mozilla
- Open Source (Apache 2.0 / MIT dual licence)
- Stable Release (Rust 1.0) in 2015
- Current Version 1.14 (Dec 22, 2016)
- Multi-paradigm
- Compiled
- Most loved Programming Language 2016

Key Features

- Zero-cost abstractions
- Move semantics
- Guaranteed memory safety
- Threads without data races
- Trait-based generics
- Pattern matching
- Type inference
- Efficient C bindings

Projects using Rust

- `rustc` - The Rust compiler
- Servo browser engine
- Mozilla Firefox
- Redox OS experimental Operation System
- Magic Pocket Dropbox Petabyte Storage System
- OpenDNS

Syntax and Semantics

Outline

Hello world!

```
fn main() {
    println!("Hello world!");
}
```

Hello world!

```rust
fn main() {
    println!("Hello world!");
}
```

```
$ rustc helloworld.rs
$ ./helloworld
Hello world!
```

Variable bindings

```
let x = 3;
```

Variable bindings

```
let x = 3;
```

```
let (x, y) = (3, 14);
```

Variable bindings

```
let x = 3;
```

```
let (x, y) = (3, 14);
```

```
let y: i32 = 14;
```

Variable bindings

```
let x = 3;
```

```
let (x, y) = (3, 14);
```

```
let y: i32 = 14;
```

```
let z: u64;
```

Mutability

```
let x = vec![1, 2, 3];
x.push(4);
```

Mutability

```
let x = vec![1, 2, 3];
x.push(4);
```

```
error: cannot borrow immutable local variable `x` as mutable
 --> variable.rs:3:5
  |
2 |     let x = vec![1, 2, 3];
  |         - use `mut x` here to make mutable
3 |     x.push(4);
  |     ^ cannot borrow mutably

error: aborting due to previous error
```

Functions

```rust
fn main() {
    let x = 7;
    println!("Square of {} is {}", x, square(x));
}

fn square(x: i32) -> i32 {
    x * x
}
```

Functions

```rust
fn main() {
    let x = 7;
    println!("Square of {} is {}", x, square(x));
}

fn square(x: i32) -> i32 {
    x * x
}
```

```
$ rustc square.rs
$ ./square
Square of 7 is 49
```

Primitive Types

- Booleans: `bool`
- Unicode characters (UTF-32): `char`

Primitive Types

- Booleans: `bool`
- Unicode characters (UTF-32): `char`
- Numeric types
  - Singed / unsigned integers: `i8` / `u32`
  - Floting-point types: `f64`
  - Platform dependent: `isize` and `usize`

Primitive Types

- Booleans: `bool`
- Unicode characters (UTF-32): `char`
- Numeric types
  - Singed / unsigned integers: `i8` / `u32`
  - Floting-point types: `f64`
  - Platform dependent: `isize` and `usize`
- Arrays / Slices: `[1, 2, 3]`
- Tuples: `(u16, (char, bool))`
- Function pointers: `fn(f32) -> f32`

if

```
fn abs(i: i32) -> i32 {
    if i < 0 {-i} else {i}
}
```

if

```
fn abs(i: i32) -> i32 {
    if i < 0 {-i} else {i}
}
```

```
$ rustc abs.rs
$ ./abs
abs(42) is 42
abs(-17) is 17
abs(-33) is 33
```

Loops

```
loop {
    println!("loop forever!");
}
```

Loops

```
loop {
    println!("loop forever!");
}
```

```
while 1 > 0 {
    println!("loop forever!");
}
```

Loops

```
loop {
    println!("loop forever!");
}
```

```
while 1 > 0 {
    println!("loop forever!");
}
```

```
for _ in 0.. {
    println!("loop forever!");
}
```

The for-loop

```
for i in (0..4).rev() {
    println!("{} seconds left …", i);
}
```

The for-loop

```
for i in (0..4).rev() {
    println!("{} seconds left …", i);
}
```

Works for

- Ranges `0..10`
- Collections
- Anything that `impl`s `IntoIterator`

Match

```
fn print_as_ordinal(number: u32) {
    println!("{}{}", number,
             match (number % 10, number % 100) {
                 (1, 1) | (1, 21...91) => "st",
                 (2, 2) | (2, 22...92) => "nd",
                 (3, 3) | (3, 23...93) => "rd",
                 _                     => "th"
             });
}
```

Outline

Ownership

```c
#include <stdio.h>

int * create_vec(int len) {
    int res[len];
    for (int i = 0; i < len; ++i) {
        res[i] = i;
    }
    return res;
}

int main() {
    int * vec = create_vec(5);
    for (int i = 0; i < 5; ++i) {
        printf("%d, ", vec[i]);
    }
}
```

Ownership

```
$ gcc 1_segfault.c
1_segfault.c: In function ‘create_vec’:
1_segfault.c:8:9: warning: function returns address of
↪  local variable [-Wreturn-local-addr]
  return res;
         ^~~
$ ./a.out
segmentation fault (core dumped)
```

Ownership

```
fn create_vec(len: i32) -> Vec<i32> {
    (0..len).collect()
}

fn main() {
    let vec = create_vec(5);
    println!("{:?}", vec);
}
```

Ownership

```
$ rustc 1_segfault.rs
$ ./1_segfault
[0, 1, 2, 3, 4]
```

Ownership

```c
#include <stdio.h>

void fill(int *vec, int len) {
    for (int i = 0; i < len; ++i) {
        vec[i] = i;
    }
}

int main() {
    int vec[5];
    fill(vec, 5);
    for (int i = 0; i < 5; ++i) {
        printf("%d, ", vec[i]);
    }
    printf("\n");
}
```

Ownership

```
$ gcc 2_ownership.c
$ ./a.out
0, 1, 2, 3, 4,
```

Ownership

```
fn fill(mut vec: Vec<i32>, len: i32) {
    vec.extend(0..len);
}

fn main() {
    let mut vec = Vec::new();
    fill(vec, 5);
    println!("{:?}", vec);
}
```

Ownership

```
$ rustc 2_ownership.rs
error[E0382]: use of moved value: `vec`
 --> 2_ownership.rs:8:22
  |
7 |     fill(vec, 5);
  |          --- value moved here
8 |     println!("{:?}", vec);
  |                      ^^^ value used here after move
  |
  = note: move occurs because `vec` has type
  ↪  `std::vec::Vec<i32>`, which does not implement the
  ↪  `Copy` trait

error: aborting due to previous error
```

Outline

Borrowing

```
fn fill(vec: &mut Vec<i32>, len: i32) {
    vec.extend(0..len);
}

fn main() {
    let mut vec = Vec::new();
    fill(&mut vec, 5);
    println!("{:?}", vec);
}
```

Borrowing

```
$ rustc 3_borrowing.rs
$ ./3_borrowing
[0, 1, 2, 3, 4]
```

Borrowing

```
class Test {
    public static void main(String[] args) {
        List<Integer> vec = new ArrayList<>();
        for (int i = 0; i < 5; i++) {
            vec.add(i);
        }
        for (int i : vec) {
            if (i % 2 == 1) {
                vec.add(i);
            }
        }
        System.out.println(vec);
    }
}
```

Borrowing

```
$ javac Test.java
$ java Test
Exception in thread "main"
        java.util.ConcurrentModificationException
    at java.util.ArrayList$Itr
        .checkForComodification(ArrayList.java:901)
    at java.util.ArrayList$Itr.next(ArrayList.java:851)
    at Test.main(Test.java:9)
```

Borrowing

```
fn main() {
    let mut vec: Vec<_> = (0..5).collect();

    for i in &vec {
        if i % 2 == 1 {
            vec.push(*i);
        }
    }

    println!("{:?}", vec);
}
```

Borrowing

```
$ rustc 4_iterator.rs
error[E0502]: cannot borrow `vec` as mutable because it is
↪  also borrowed as immutable
 --> 4_iterator.rs:6:13
  |
4 |     for i in &vec {
  |              --- immutable borrow occurs here
5 |         if i % 2 == 1 {
6 |             vec.push(*i);
  |             ^^^ mutable borrow occurs here
7 |         }
8 |     }
  |     - immutable borrow ends here

error: aborting due to previous error
```

Borrowing

```
class Test {
    public static void main(String[] args) {
        ArrayList<Integer> vec = new ArrayList<>();
        for (int i = 0; i < 5; i++) {
            vec.add(i);
        }
        ArrayList<Integer> tmp = new ArrayList<>();
        for (int i : vec) {
            if (i % 2 == 1) {
                tmp.add(i);
            }
        }
        vec.addAll(tmp);
        System.out.println(vec);
    }
}
```

Borrowing

```
$ javac Test.java
$ java Test
[0, 1, 2, 3, 4, 1, 3]
```

Borrowing

```
fn main() {
    let mut vec: Vec<_> = (0..5).collect();
    let mut tmp = Vec::new();

    for i in &vec {
        if i % 2 == 1 {
            tmp.push(*i);
        }
    }

    vec.append(&mut tmp);

    println!("{:?}", vec);
}
```

Borrowing

```
$ rustc 5_join.rs
$ ./5_join
[0, 1, 2, 3, 4, 1, 3]
```

Lambdas

```
class Test {
    public static void main(String[] args) {
        List<Integer> vec = IntStream.range(0, 5)
                .boxed().collect(Collectors.toList());

        vec.addAll(vec.stream()
                .filter(i -> i % 2 == 1)
                .collect(Collectors.toList()));

        System.out.println(vec);
    }
}
```

Lambdas

```
fn main() {
    let mut vec: Vec<_> = (0..5).collect();

    let mut tmp = vec.iter().cloned()
        .filter(|i| i % 2 == 1).collect();

    vec.append(&mut tmp);

    println!("{:?}", vec);
}
```

Outline

Structs

- `struct`

```
struct Person {
    name: String,
    age: u16,
}
```

Structs

- `struct`

```
struct Person {
    name: String,
    age: u16,
}
```

- Touple `struct`

```
struct Point(i32, i32, i32);
```

Structs

- `struct`

```
struct Person {
    name: String,
    age: u16,
}
```

- Touple `struct`

```
struct Point(i32, i32, i32);
```

- Unit-like `struct`

```
struct Unicorn;
```

Structs

```
struct Person {
    name: String,
    age: u16,
}
```

```
let hawking = Person{
    name: "Stephen Hawking".to_string(),
    age: 75 };
```

Methods

```
impl Person {
    fn new(name: String, age: u16) -> Person {
        Person { name: name, age: age }
    }

    fn introduce(&self) {
        println!("My name is {}. I'm {} years old.",
                 self.name, self.age);
    }
}

fn main() {
    let name = "Stephen Hawking".to_string();
    let hawking = Person::new(name, 75);
    hawking.introduce();
}
```

Traits

```
interface Animal {
    void makeNoise();
    void eat();
}
```

Traits

```
interface Animal {
    void makeNoise();
    void eat();
}
```

```
trait Animal {
    fn make_noise(&self);
    fn eat(&self);
}
```

Traits

```java
class Cat implements Animal {
    public void makeNoise() {
        System.out.println("meow!");
    }
    public void eat() {
        System.out.println("mhh milk!");
    }
}
```

```rust
struct Cat;
impl Animal for Cat {
    fn make_noise(&self) {
        println!("meow!");
    }
    fn eat(&self) {
        println!("mhh milk!");
    }
}
```

Traits

```
class Dog implements Animal {
    public void makeNoise() {
        System.out.println("woof!");
    }
    public void eat() {
        System.out.println("mhh mailman...");
    }
}
```

```
struct Dog;
impl Animal for Dog {
    fn make_noise(&self) {
        println!("woof!");
    }
    fn eat(&self) {
        println!("mhh mailman...");
    }
}
```

Traits

```
static void feed(Animal animal) {
    animal.eat();
    animal.makeNoise();
}
```

Traits

```
static void feed(Animal animal) {
    animal.eat();
    animal.makeNoise();
}
```

```
fn feed<T: Animal>(animal: T) {
    animal.eat();
    animal.make_noise();
}
```

Traits

```
public static void main(String[] args) {
    Cat cat = new Cat();
    feed(cat);
    Dog dog = new Dog();
    feed(dog);
}
```

```
fn main() {
    let cat = Cat;
    feed(cat);
    let dog = Dog;
    feed(dog);
}
```

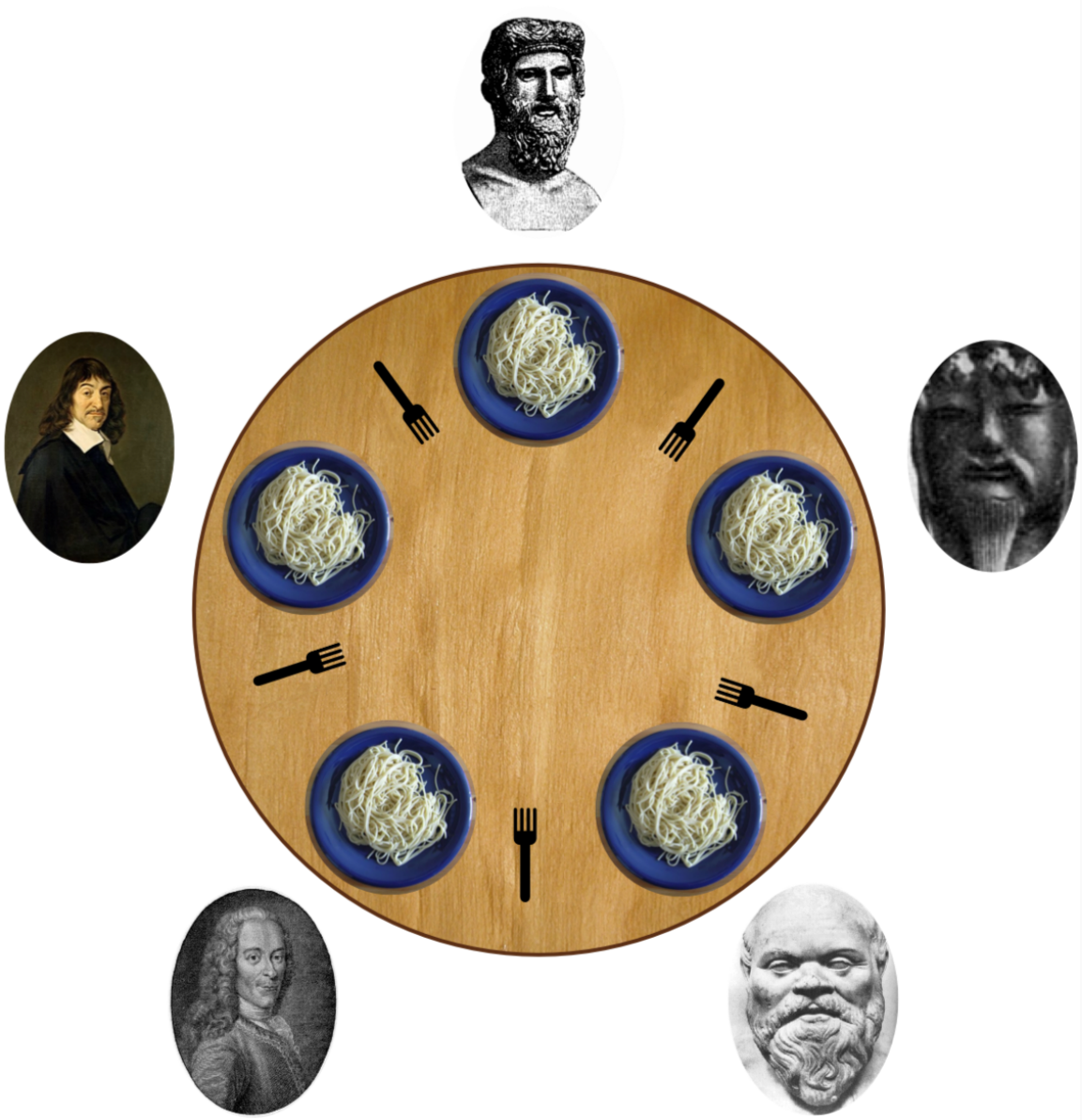
Traits

```
$ javac Traits.java
$ java Traits
mhh milk!
meow!
mhh mailman...
woof!
```

```
$ rustc traits.rs
$ ./traits
mhh milk!
meow!
mhh mailman...
woof!
```

Effective Rust

Dining Philosophers



Dining Philosophers

```
struct Table {
    forks: Vec<Mutex<()>>,
}
```

Dining Philosophers

```
struct Table {
    forks: Vec<Mutex<()>>,
}
```

```
struct Philosopher {
    name: String,
    left: usize,
    right: usize,
}
```

Dining Philosophers

```
impl Philosopher {
    fn eat(&self, table: &Table) {
        let _left = table.forks[self.left]
                .lock().unwrap();

        thread::sleep(Duration::from_millis(100));

        let _right = table.forks[self.right]
                .lock().unwrap();

        println!("{} is eating.", self.name);
        thread::sleep(Duration::from_millis(100));
        println!("{} is done eating.", self.name);
    }
}
```

Dining Philosophers

```
fn main() {
    let table = Arc::new(Table { forks: vec![
        Mutex::new(()), Mutex::new(()),
        Mutex::new(()), Mutex::new(()),
        Mutex::new(()),
    ]});

    let philosophers = vec![
        Philosopher::new("Plato", 0, 1),
        Philosopher::new("Konfuzius", 1, 2),
        Philosopher::new("Socrates", 2, 3),
        Philosopher::new("Voltaire", 3, 4),
        Philosopher::new("Descartes", 0, 4),
    ];
```

Dining Philosophers

```
    let handles: Vec<_> = philosophers
        .into_iter().map(|p| {
            let table = table.clone();

            thread::spawn(move || {
                p.eat(&table);
            })
        }).collect();

    for h in handles {
        h.join().unwrap();
    }
}
```

Dining Philosophers

```
$ rustc philosophers.rs
$ ./philosophers
Plato is eating.
Socrates is eating.
Konfuzius is eating.
Voltaire is eating.
Voltaire is done eating.
Socrates is done eating.
Konfuzius is done eating.
Plato is done eating.
Descartes is eating.
Descartes is done eating.
```

Advanced Topics

- Lifetimes
- Trait Objects
- Associated Types
- Operator Overloading
- Macros
- Conditional Compilation
- `unsafe` Rust
- Crates and Modules
- Foreign Function Interface
- Cargo
- Rustdoc
- …