Rust intro

(for Java Developers)

JFokus 2017 - #jfokus

Hi!

- Computer Engineer
- Programming
- Electronics
- Math <3 <3
- Physics
- Lego
- Meetups
- Animals
- Coffee
- Pokémon
- GIFs

CODE MINER

OSS Projects:

- https://github.com/hannelita/neo4j-cassandra-connector
- https://github.com/hannelita/neo4j_doc_manager

Disclaimer

This is a session about Rust Features :)

Disclaimer

This is not a Rust intro tutorial

Some theory

Some subjects that may cause discussion. Views are on my own.

GIFs :)

Disclaimer

There are some references for introductory Rust Content

Language peace <3

Agenda

- **What is Rust?**
- Why is Rust interesting?
- Rust structure quick overview
- Borrow
- Lifetime
- Feature Comparison
- The sense of safety
- Rust downsides

What is Rust?

'Rust is a general purpose programming language, compiled, strong and static typed, sponsored by Mozilla Research. It is designed to be a "safe, concurrent, practical language", supporting functional and imperative-procedural paradigms.'

https://en.wikipedia.org/wiki/Rust_(programming_language)#cite_note-FAQ_-_The_Rust_Project-10

Is it yet another language that runs on top of the JVM?

No. Rust is not 'yet another language that runs on top of the JVM'.

Agenda

- **What is Rust?**
- **Why is Rust interesting?**
- Rust structure quick overview
- Borrow
- Lifetime
- Feature Comparison
- The sense of safety
- Rust downsides

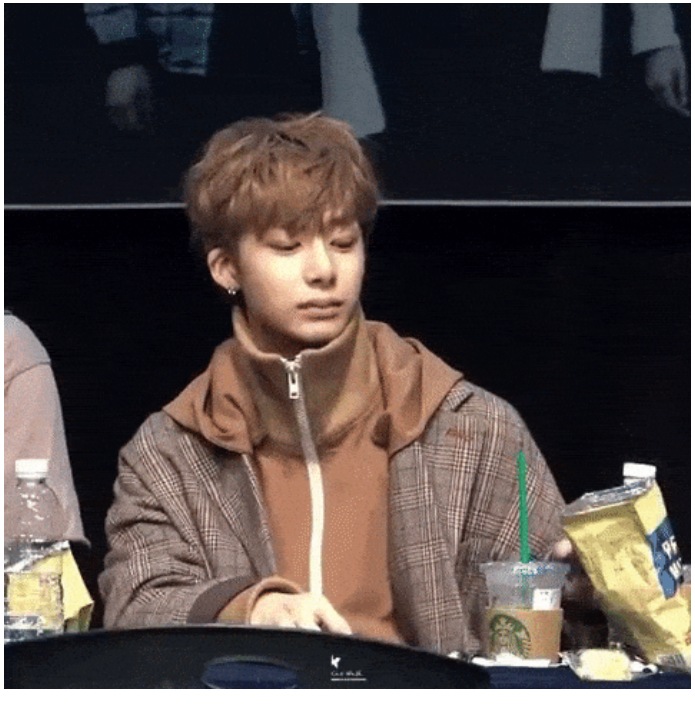
Embedded systems

Sometimes they are so restrictive that you can't use Java.

Which language do you choose?

Source - http://www.diva-portal.org/smash/get/diva2:215157/FULLTEXT01

C and C++



C and C++

- It's hard to debug
- It can be difficult to maintain the code
- Manual memory allocation

It may be inconvenient.

What are we looking for in terms of language?

Good choices

- No manual memory management
- Strong and Static Typed
- Compiled
- Fast
- Reduce number of runtime errors
- Active community
- Good amount of frameworks and libraries
- Open Source

Meet Rust!

Rust - features

- Memory safe, data race free
- A compiler that blocks lots of runtime errors
- Interface with C/C++
- Generics
- Polymorphism
- By Mozilla and an amazing community

Rust - it meets the requirements

- No manual memory management √
- Strong and Static Typed √
- Compiled √
- Fast √
- Reduce number of runtime errors √
- Active community √
- Good amount of frameworks and libraries √
- Open Source √

Bonus

- About the same level of verbosity as Java :)
- Rust Compiler is also verbose to explain the errors to you

More about Rust

- No VM
- No GC
- No manual malloc/free
- No seg faults

Agenda

- **What is Rust?**
- **Why is Rust interesting?**
- **Rust structure quick overview**
- Borrow
- Lifetime
- Feature Comparison
- The sense of safety
- Rust downsides

Quick view at Rust

```rust
fn main() {
    fizzbuzz_to(100);
}

fn is_divisible_by(lhs: u32, rhs: u32) -> bool {
    if rhs == 0 {
        return false;
    }
    lhs % rhs == 0
}

fn fizzbuzz(n: u32) -> () {
    if is_divisible_by(n, 15) {
        println!("fizzbuzz");
    } else if is_divisible_by(n, 3) {
        println!("fizz");
    } else if is_divisible_by(n, 5) {
        println!("buzz");
    } else {
        println!("{}", n);
    }
}

fn fizzbuzz_to(n: u32) {
    for n in 1..n + 1 {
        fizzbuzz(n);
    }
}
```

Limited type inference. Explicit type declaration for function parameters and return.

(same as in Java)

Macros

source: http://rustbyexample.com/fn.html

Quick view at Rust

```rust
fn main() {
    let _immutable_binding = 1;
    let mut mutable_binding = 1;
    println!("Before mutation: {}", mutable_binding);
    // Ok
    mutable_binding += 1;
    println!("After mutation: {}", mutable_binding);

    // Error!
    _immutable_binding += 1;
    // FIXME ^ Comment out this line
}
```

Immutability by default

source: http://rustbyexample.com/variable_bindings/mut.html

Quick view at Rust

```rust
fn is_odd(n: u32) -> bool {
    n % 2 == 1
}

fn main() {
    println!("Find the sum of all the squared odd numbers under 1000");
    let upper = 1000;
    let mut acc = 0;
    for n in 0.. {
        let n_squared = n * n;
        if n_squared >= upper {
            break;
        } else if is_odd(n_squared) {
            acc += n_squared;
        }
    }
    println!("imperative style: {}", acc);
    let sum_of_squared_odd_numbers: u32 =
        (0..).map(|n| n * n)              // All natural numbers squared
             .take_while(|&n| n < upper) // Below upper limit
             .filter(|n| is_odd(*n))     // That are odd
             .fold(0, |sum, i| sum + i); // Sum them
    println!("functional style: {}", sum_of_squared_odd_numbers);
}
```

High Order Functions

source: http://rustbyexample.com/fn/hof.html

Other features - Tuples, Enums, Structs, Traits.

Traits are similar to Java 8 Interfaces


Yeah, I have a lot of questions.

How do we achieve the 'No Seg Faults' feature?

Agenda

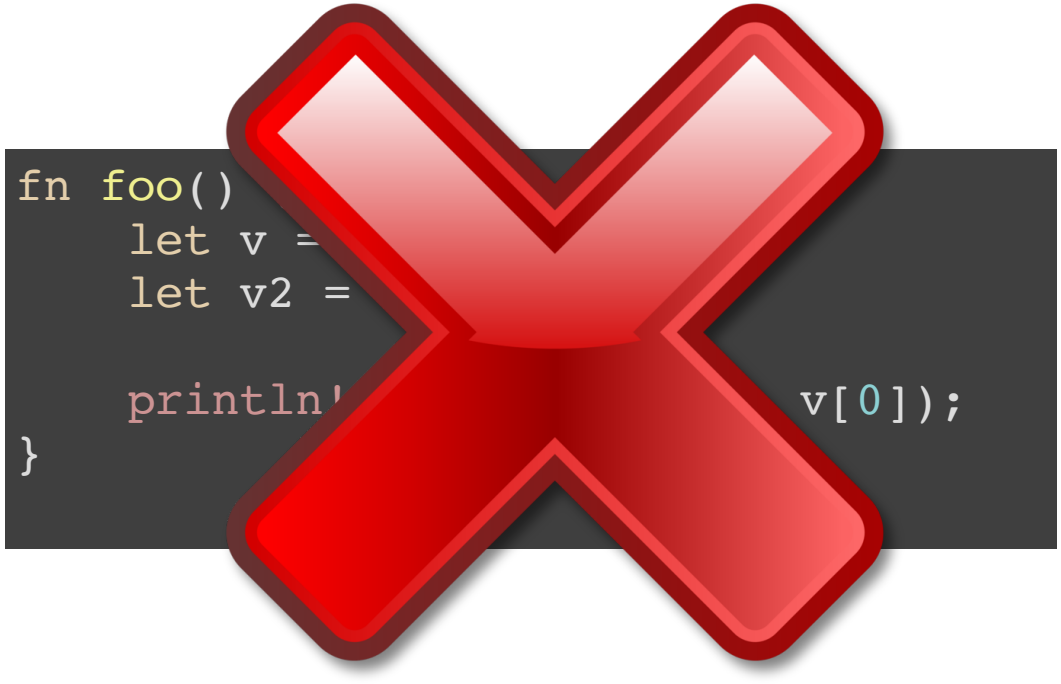
- **What is Rust?**
- **Why is Rust interesting?**
- **Rust structure quick overview**
- **Borrow**
- Lifetime
- Feature Comparison
- The sense of safety
- Rust downsides

Variable bindings own the values in Rust

```
fn foo() {
    let v = vec![1, 2, 3];
    let v2 = v;

    println!("v[0] is: {}", v[0]);
}
```

Variable bindings own the values in Rust



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

What?

It may sound unpractical, but by having this model, Rust prevents several memory errors.

Rust allows you to share some references with a feature called '**borrowing**'

Borrowing

```
fn main() {
    fn sum_vec(v: &Vec<i32>) -> i32 {
        return v.iter().fold(0, |a, &b| a + b);
    }
    fn foo(v1: &Vec<i32>, v2: &Vec<i32>) -> i32 {
        let s1 = sum_vec(v1);
        let s2 = sum_vec(v2);
        s1 + s2
    }

    let v1 = vec![1, 2, 3];
    let v2 = vec![4, 5, 6];

    let answer = foo(&v1, &v2);
    println!("{}", answer);
}
```

&

It is similar to Read-Writers lock

- Many readers at once **OR** a single writer with exclusive access
- Read only do not require exclusive access
- Exclusive access do not allow other readers

Rust uses a similar model at compile time.

(More info: https://users.cs.duke.edu/~chase/cps210-archive/slides/moresync6.pdf)

It is similar to Read-Writers lock

- Many readers at once **OR** a single writer with exclusive access
- Read only do not require exclusive access
- Exclusive access do not allow other readers

Rust uses a similar model at compile time.

T: Base type; owns a value
&T: Shared reader
&mut T: Exclusive writer

(Note: I am not considering another Rust feature called Copy)

It is similar to Read-Writers lock

- Many readers at once **OR** a single writer with exclusive access
- Read only do not require exclusive access
- Exclusive access do not allow other readers

Rust uses a similar model at compile time.

T: Base type; owns a value

&T: Shared reader — Immutable reference

&mut T: Exclusive writer — Mutable reference

(Note: I am not considering another Rust feature called Copy)

About exclusive writers

```rust
fn main() {
    let mut x = 5;
    let y = &mut x;

    *y += 1;

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

Rust compiler says: "error: cannot borrow `x` as immutable because it is also borrowed as mutable println!("{}", x);"

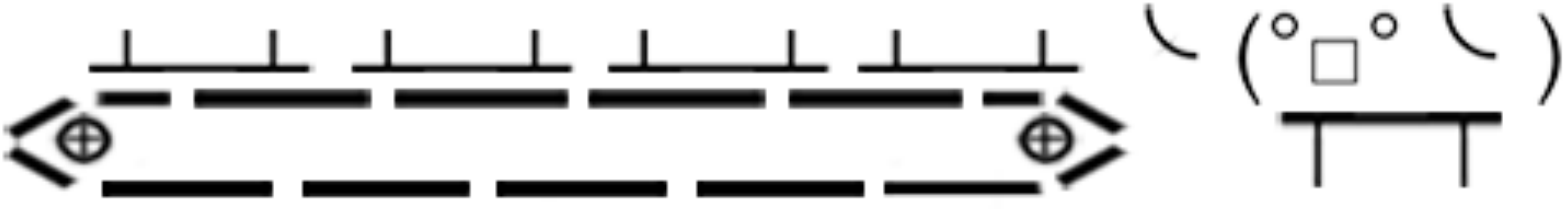
Top issues that borrowing prevents:

- Iterator invalidation
- Data race problems
- Use after free

BTW, how do I free a variable in Rust? Since there is no GC, how should I clean up the memory?

Also, I could easily mess up with borrowing by freeing a variable that I lent to a function.

You don't have to handle that manually. At least, not explicitly.



#jfokus @hannelita

Agenda

- **What is Rust?**
- **Why is Rust interesting?**
- **Rust structure quick overview**
- **Borrow**
- **Lifetime**
- Feature Comparison
- The sense of safety
- Rust downsides

In Rust, every reference has some lifetime associated with it.

```
fn lifetimes() {

    let a1 = vec![1, 2, 3]; //                  +
    let a2 = vec![4, 5, 6]; //             +    |
                            //             |    |
    let b1 = &a1;           //        +    |    |
    let b2 = &a2;           //   +    |    |    |
    foo(b1);                //   |    |    |    |
    foo(b2);                // 'b2  'b1  'a2  'a1
                            //   |    |    |    |
}
```

You can explicit lifetimes in Rust

```
fn explicit_lifetime<'a>(x: &'a i32) {
}
```

Or even multiple lifetimes:

```
fn multiple_lifetimes<'a, 'b>(x: &'a str, y: &'b str) -> &'a str {
}
```

By the end of a lifetime, a variable is free.

Top issues that lifetime system prevents:

- GC is not necessary
- Another mechanism to avoid dangling pointers
- No manual malloc nor free

Okay, so is Rust always safe?

Agenda

- **What is Rust?**
- **Why is Rust interesting?**
- **Rust structure quick overview**
- **Borrow**
- **Lifetime**
- **Feature Comparison**
- The sense of safety
- Rust downsides

Rust has a good Generics resource, with Traits and Closures

http://huonw.github.io/blog/2015/05/finding-closure-in-rust/

Comparison - Java and Rust Features

Classes


```
public class MyClass {
    private in number = 42;
    private MyOtherClass c =
      new MyOtherClass();


    public int count() {
      ..
    }
}
```

```
struct MyClass {
    number: i32,
    other: MyOtherClass,
}

impl MyClass {
    fn myMethodCountHere(&self) -> i32 {
        ...
    }
}
```

Primitive types

Primitive types

Interfaces

```java
public interface MyInterface {

        void someMethod();

        default void someDefault(String str){
                //implementation
        }

}
```

```rust
trait Animal {
    fn noise(&self) -> &'static str;
    fn talk(&self) {
        println!("I do not talk to humans");
    }
}

struct Horse { breed: &'static str }

impl Animal for Horse {
  fn noise(&self) -> &'static str {
    "neigh!"
    // I can't mimic horse sounds
  }

  fn talk(&self) {
      println!("{}!!!!", self.noise());
  }
}

impl Horse {
  fn move(&self) {
    //impl
  }
}
```

Generics

```java
public class MyGeneric<T> {
  //impl
}

public class NotGeneric {
  public static <T extends Comparable<T>> T maximum(T x, T y) {
    //awesome
  }
}
```

```rust
trait Traverse<I> {
  // methods
}

struct Bag<T> {
  //struct
}

impl<T> Bag<T> {
    //impl
}
```

Rust Generics

```
fn general_method<T: MyTrait>(t: T) { ... }

fn general_method<T: MyTrait + AnotherTrait + SomeRandomTrait>(t: T)
```

(Trait bounds: use it for the good and for the evil)

Quick mention

Arrays

Mutability

Agenda

- **What is Rust?**
- **Why is Rust interesting?**
- **Rust structure quick overview**
- **Borrow**
- **Lifetime**
- **Feature Comparison**
- **The sense of safety**
- Rust downsides

Rust is pretty safe not only because of borrowing and lifetimes

You can call C/C++ functions from Rust. But C/C++ is not safe.

unsafe

```
fn main() {
    let u: &[u8] = &[49, 50, 51];

    unsafe {
        assert!(u == std::mem::transmute::<&str, &[u8]>("123"));
    }
}
```

Explicit calls for unsafe.

So, is Rust perfect?

Agenda

- **What is Rust?**
- **Why is Rust interesting?**
- **Rust structure quick overview**
- **Borrow**
- **Lifetime**
- **Feature Comparison**
- **The sense of safety**
- **Rust downsides**

Top Rust complaints

- Learning curve is not too fast
- Lots of new concepts
- Pretty new language

Top Rust complaints

- Learning curve is not too fast
- Lots of new concepts
- Pretty new language

Top Rust responses to these problems

- Great docs and learning resources
- The community is active and willing to help
- The community is building lots of tools and libraries

Bonus #1:

How can you describe Rust type system?

Answer: Somewhat static, strongly typed. There is a huge research project to describe Rust type system

https://www.ralfj.de/blog/2015/10/12/formalizing-rust.html

Bonus #2: Performance

mandelbrot

source	secs	KB	gz	cpu	cpu load
Rust	**2.01**	28,256	1007	7.97	100% 100% 100% 100%
Java	5.89	89,504	796	23.08	98% 98% 98% 99%

k-nucleotide

source	secs	KB	gz	cpu	cpu load
Rust	9.44	152,620	1641	23.98	91% 38% 91% 36%
Java	8.02	467,004	1802	25.57	76% 98% 73% 74%

source: https://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=rust&lang2=java

Bonus #3:

Free GIF!

References

- https://www.youtube.com/watch?v=Q7lQCgnNWU0
- https://www.quora.com/Why-do-programming-languages-use-type-systems
- http://blogs.perl.org/users/ovid/2010/08/what-to-know-before-debating-type-systems.html
- http://lucacardelli.name/papers/typesystems.pdf
- https://www.ralfj.de/blog/2015/10/12/formalizing-rust.html
- http://jadpole.github.io/rust/type-system
- https://wiki.haskell.org/Typing
- https://gist.github.com/Kimundi/8391398
- https://www.smashingmagazine.com/2013/04/introduction-to-programming-type-systems/
- http://pcwalton.github.io/blog/2012/08/08/a-gentle-introduction-to-traits-in-rust/
- https://llogiq.github.io/2016/02/28/java-rust.html

References - Rust Intro

- https://doc.rust-lang.org/book/
- https://doc.rust-lang.org/reference.html
- https://doc.rust-lang.org/nomicon/
- Rust And Pokémons - http://slides.com/hannelitavante-hannelita/rust-and-pokmons-en#/
- Rust Type System - http://slides.com/hannelitavante-hannelita/rust-type-system-pybr12#/ (Python Brasil 2016 closing keynote)

Special Thanks

- Rust Community - https://www.meetup.com/Rust-Sao-Paulo-Meetup/ and @bltavares
- B.C., for the constant review
- JFokus Team

Thank you :)

Questions?

hannelita@gmail.com
@hannelita