=======================================================================

arthur 2019-07-18 11:56:09 767 收藏 13

分类专栏： 区块链

Rust官网:https://www.rust-lang.org/zh-CN/

Rust中文社区:https://rust.cc/

github地址:https://github.com/rust-lang/rust

docs.rs文档库:https://docs.rs/

crates.io仓库地址:https://crates.io/

Rust入门教程-中文版:https://kaisery.gitbooks.io/rust-book-chinese/content/

Rust高级教程-中文版:https://kaisery.github.io/trpl-zh-cn/

使用 Remix MetaMask 发币

arthur 2021-08-17 16:51:04

如何使用 remix 和 metamask ，在火链 heco 以太坊 eth 上部署合约，并且发 erc20 的代币，然后上 uniswap 或者 mdex，进行兑换和挖矿。

火链，以太坊教程是通用的，erc20 hrc20 也是一个东西，互相兼容的。

这里以 heco 测试链为主。

原文链接：

<https://blog.csdn.net/chen__an/article/details/119760592>

=======================================================================

Rust官网 & Rust程序设计语言，网页版：

<https://www.rust-lang.org/zh-CN/learn>

（包含学习课程与练习例子）

中文网页版：

<https://kaisery.github.io/trpl-zh-cn/title-page.html>

=======================================================================

[蒋古申](https://www.zhihu.com/people/huang-yi-zheng-27)

少睡懒觉多读书

**第一步：**

查阅 [Rust 官方文档](https://link.zhihu.com/?target=https%3A//www.rust-lang.org/learn/get-started)，安装配置好 Rust 的开发环境。

使用 [cargo new hello-world](https://www.zhihu.com/search?q=cargo+new+hello-world&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22answer%22%2C%22sourceId%22%3A915365379%7D)新建一个 Rust 项目，并且使用自己喜欢的开发工具打开（VSCode, Clion）。我个人比较喜欢 VSCode + Docker 用配置 [devcontainer](https://link.zhihu.com/?target=https%3A//code.visualstudio.com/docs/remote/containers) 的方式学习一门新的编程语言，这样的好处是可以不用考虑容器之外的开发环境，做到最高效、不踩坑。

**第二步：**

不要急于求成，先熟悉编程语言“周边工具”，这样可以大大提高你学习和使用 Rust 的效率。我推荐先去熟悉 cargo：Rust 的项目和依赖管理工具。具体的方法是直接看一本叫做 [The Cargo Book](https://link.zhihu.com/?target=https%3A//doc.rust-lang.org/cargo/index.html) 的书籍。

看 The Cargo Book 的同时在自己配置好的开发环境里面尝试使用 cargo ，通过实践来学习 cargo.toml 的作用和写法，命令，理解为什么 cargo.lock 在工程作为一个 lib 的时候要加入 .gitignore 而作为 command-line application 的时候又要加入 .git这类的东西 233……

**第三步：**

到了这一步才真正开始 Rust 的学习，关于任何编程语言的学习，都是从阅读官方文档开始。Rust 社区发行了一个简明的开源教程：[The Rust Programming Language](https://link.zhihu.com/?target=https%3A//doc.rust-lang.org/book/%23the-rust-programming-language) , 强烈推荐阅读。

如果不太适应英文教材的阅读，可以看这本书的中文翻译版本：[Rust 程序设计语言（第二版 & 2018 edition）简体中文版](https://link.zhihu.com/?target=https%3A//kaisery.gitbooks.io/trpl-zh-cn/content/)。

在阅读官方文档的时候，就可以根据上述的进行一些编程练习了。不过 Rust 的官方文档相对于其他编程语言文档来说，比较注重的是 Rust 的几个编程特性：零成本抽象，所有权特性

等，在阅读这些的时候不推荐自己瞎写练习，应该对比阅读文档和其他的开源 Rust 代码。

**第四步：**

在充分感受 Rust 的编程范式以后，应该进入一定量的练习阶段。

这个时候推荐学习的材料是：[Rust by Example](https://link.zhihu.com/?target=https%3A//doc.rust-lang.org/rust-by-example/), 直接通过在线修改代码阅读样例来学习。

快速过一遍 Rust by Example 之后，相信现在至少可以开始上手写一些 Rust 代码了，个人喜欢把以前的一些小项目用新学习的编程语言重写，例如实现一个最简单的 DBMS，通过这样的行为不仅可以加深对新学语言的熟练度，还可以通过与原有实现的对比，感受 Rust 语言特性。

**第五步：**

之后主要通过项目练手，阅读开源代码，逐步提高，推荐辅助阅读张汉东的《Rust 编程之道》。

编辑于 2019-11-29 15:14

=======================================================================

Medjed 男

1.安装vscode

2.安装rust，vscode安装rust-analyze插件，这个比rls好太多了。但是ra不支持一些检查（据我的经验，返回较大范围的生命周期的引用不会报错，但是cargo check会编译不过）

3.把std.rs和lib.rs这两个网站收藏到书签，并且放到显眼位置。

4.按部就班看the rust book或者rust权威指南或者rust examples，不要中途想按照其他语言那样写几个数据结构练手你会被编译器锤的。

5.**跟编译器学习rus**t。

6.斯坦福开了个rust语言课叫cs110l。里面除了练习外还有两个project，分别是写一个debugger和一个反向代理，可以做做。

7.做pingcap的talent plan

编辑于 2021-04-15 14:22

=======================================================================

华为云开发者社区

**对于不同类型初学者学Rust要突破的难点也是不一样的！**

此处，将初学者分为五大类来讨论。

**完全零基础的开发者：**掌握计算机基础体系知识结构，理解Rust语言和硬件/OS层的抽象，理解Rust语言核心概念、以及它的抽象模式，选择Rust语言的某个适用领域进行实操训练，通过实践来提升Rust语言的熟练度和理解深度，同时掌握领域知识。

**有C语言基础：**由于C语言开发者对高级语言的抽象不是很理解，所以着重了解掌握Rust所有权机制，包括所有权的语义，生命周期和借用检查。了解Rust语言的抽象模式，主要是类型和trait；以及Rust本身的的OOP和函数式语言特性。

**有C++基础：**C++开发者对于Rust语言的所有权有很好的理解能力，主要精力放在Rust的抽象模式和函数式语言特性上。

**有Java/Python/Ruby基础：**着重理解攻克Rust所有权机制、抽象模式、函数式编程语言特性。

**有Go基础：**Go语言开发者比较容易理解Rust的类型和trait抽象模式，但Go也是GC语言，所以所有权机制和函数式语言特性是他们的学习重点。

**有Haskell基础：**Haskell系的开发者对Rust语言函数式特性能很好的理解，主要攻克所有权机制和OOP语言特性。

所以，对于有一定基础的开发者来说，学习Rust语言要掌握的**几个关键概念有**：

**1、Rust所有权机制，包括所有权的语义，生命周期和借用检查**

所有权机制是Rust语言最核心的特性，它保证了在没有垃圾回收机制下的内存安全，所以对于习惯了GC的开发者，理解Rust的所有权是最关键的一环，切记这三点：

Rust中的每一个值都有一个被称为其**所有者 (owner)**的变量。

值有且只有一个所有者。

当所有者（变量）离开作用域，这个值将被丢弃。

这其中又涉及到生命周期和借用检查等概念，是相对比较难啃的一块硬骨头。

**2、Rust语言的抽象模式，主要是类型和trait**

trait借鉴了Haskell中的Typeclass，它是对类型行为的抽象，可以通俗地类比为其他编程语言里的接口，它告诉编译器一个类型必须提供哪些功能语言特性。使用时要遵循一致性，不能定义相互冲突的实现。

**3、OOP语言特性**

熟悉面向对象编程（OOP）的常见的四个特性：对象、封装、继承和多态，可以更好地理解Rust的一些特性，比如impl、pub、trait等等。

**4、函数式语言特性**

Rust语言的设计深受函数式编程的影响，看到函数式特性，数学不好的人可能会望而却步，因为函数式编程语言的最大特点是把运算过程尽量写成一系列嵌套的函数调用,在Rust中，掌握闭包和迭代器是编写函数式语言风格的高性能Rust代码的重要一环。

具体含义和代码说明不再赘述，大家可以参考**《Rust成语设计语言》**以及**《Rust编程之道》**。为了让大家能更通俗易懂地理解Rust语言相关概念，**特此附上华为云专家从自身业务经验出发学习总结的Rust资料，点击下载。**

最后，感谢张汉东老师提供的思维导图学习框架，让初学者能系统的了解Rust,走近这颗冉冉升起的新星。

发布于 2021-01-13 17:13

=======================================================================

　　学习Rust已经两周了，基本上是断断续续的在学，或者是在上下班坐公交时，或者是在ODC没事做时。现在已经学习了Rust程序设计语言的前5章，是时候做一个总结了。关于数据类型或者if else这种内容我就不在罗列了，我只说一说我感兴趣的部分。

变量和可变性(variable & mutability)

rust中变量默认是不可变的，也就是说下面的类似下面的赋值操作是非法的。

let a = 100;
a = 101;

 函数的隐式返回值

    如果函数不适用return语句进行返回的话，函数隐式的返回最后一个表达式。当然这种写法是可以选装的，你可以选择传统的return版的返回语句。

 理解语句（statement）和表达式（expression）的区别

　rust对于语句和表达式的区分比较严格，语句就是执行一些操作但并不产生结果的指令，表达式计算并产生一些值。

let mut num = 100; //一个赋值语句
num = num + 1; //语句
num + 1 //表达式

　　 函数调用是一个表达式。宏调用是一个表达式。我们用来创建新作用域的大括号（代码块），{}，也是一个表达式。

 break可以带变量

　　就下边那样，loop是rust中的循环，类似while(1)。

fn main() {
let mut counter = 0;
let result = loop {
counter += 1;
if counter == 10 {
break counter\*2;
}
};
println!("The Result is {}", result);
}

元组

　　元组(tuple)类型是一个复合类型，它可以包含多种不同类型。这种类型的形式是 " (type1, type2, type3 ... ... ) "。

let tuple1 = (1, "hello", 3.14);
let tuple2: (i32, string, f32) = (1, "hello", 3.14);

　　那么怎么去获取元组的元素呢。可以通过元组的解构（destructuring），通过解构元组我们可以把元组内的值赋给其他的变量。

let (x, y, z) = tuple1;

　　也可以使用点号(".")去直接访问元组内的元素

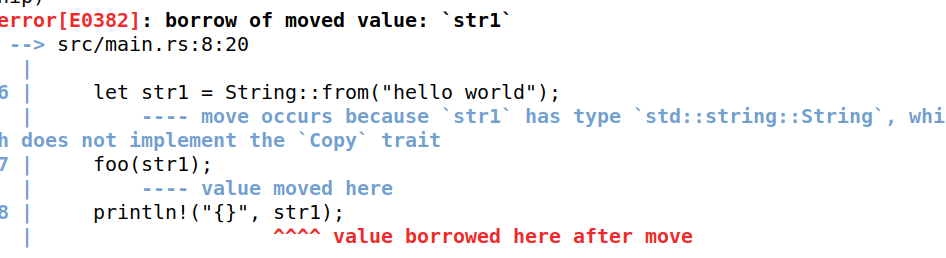
let x = tuple1.0;
let y = tuple1.1;
let z = tuple1.2;

所有权(ownership)

　　所有权是rust一个特点之一，也是rust内存管理的核心。每个值（可以理解是内存的一小块）都有一个所有者，这个所有者可以是变量。同时程序超出所有者的作用域时，所有者所有的值也会通过调用drop()函数，从而被释放。（注意drop不是通过运行时来调用的，而是在编译期就被插入了drop函数），这种做法类似于C++中的RAII机制。值得注意的是，所有权是可以被转移的，并且所有者在转移了值的所有权之后，他就会被rust认为是无效的所有者，再次使用该变量（所有者）会产生错误。

fn foo(str\_cp:String) {
println!("{}", str\_cp);
}
fn main() {
let str1 = String::from("hello world");
foo(str1);
println!("{}", str1);
}

　　在上边的例子中，我们在堆上创建了一个字符串"hello world"，该字符串的所有者是变量str1，值得注意的是，将变量作为函数的形参也会转移值的所有权，因此我们当7行的函数调用返回时，str1已经是一个无效的所有者，可怜的str1，白白丢了自己的地盘，但是他也不是什么都没有收获的，这种所有权的转移，也是有好处的。



 　　这里我生搬硬套了一种情况，尝试说明这种所有权转移的机制是如何防止double free的情况产生的。下面是一段和上述代码段类似的代码段，可以看到这里对同一内存块进行了两次free。那么rust是如何防止这种情况产生的呢？实际上在rust中17行的情况根本就不会出现，因为str1在此时已经是无效的了，没有free它的可能性。

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
void foo(char \*str)
{
printf("%s\n", str);
free(str);
}
int main(void)
{
while (1) {
char \*str1 = (char \*)malloc(512);
strncpy(str1, "hello world", 512);
foo(str1);
free(str1); //double free
printf("%s\n", str);
}
return 0;
}

=======================================================================