

项目目录介绍：

“Rust_Project”目录：为5月初学习Rust语法的笔记，该目录下的子目录src_youtube_backup为学习的源码；

“rustlings”目录：rustlings 4.5.0练习；

“rust-quiz”目录：rust-quiz练习；

“学习记录文档”目录：一些解决问题的策略和方案；

“OS实验”：部分lab截图；

2021.07.02

回顾看过的Rust和rcore项目；

2021.07.03

由于在MacOS上，vm(virtual machine)的机器不能跟host(host machine)机器共用一个代理，导致需要重复在vm上连一个新的代理，为了方便（偷懒）一步到位地解决网络问题，通过几个月的不懈努力，终于解决了这个心头之恨；配置网络，在得益于大佬萧洛元的帮助，成功解决了vm不能和走host一个代理端口的问题；

2021.07.05

在中国大学慕课上，开始观看《计算机组成与设计：RISC-V》课程；

2021.07.07

《计算机组成与设计：RISC-V》课程观看结束；

感受：全英文课件，可以学习到原设计者的原汁原味的（Untranslated）知识；

2021.07.08

观看riscv汇编英文的介绍课程；

学习视频来源：<https://www.youtube.com/channel/UC8t99gp5IN-FTf5rGVaRevw>

感受：与arm和x86不同，riscv架构讲究高性能，单单从汇编指令的精简性，就能看出riscv架构在这几种架构优势点；riscv的指令集具有简洁高效，开源开放，是其主要的优势；

2021.07.08

Rustlings开始；

Rustlings版本： 4.5.0 (20210708当前的最新版本)

2021.07.10

Rustlings结束；

感受：有了前面知识学习的铺垫，发现Rust并不是很难，甚至觉得比C++更好用（编译器自带的纠错功能真好用/狗头）；

自己的答案md写到第9个内容modules就放弃在md里写了，直接在rustlings里的源文件里写注释了；

答案参考这个网站，总结归纳做得非常好，很简洁的一个归纳；

<https://blog.frankel.ch/start-rust/3/>

部分思路来自外国开发者Théo Pomies对于github上的rustlings4.5.0的解析，链接如下：

https://github.com/theopomies/rustlings_solutions

```
use std::str::FromStr;
#[derive(Debug)]
struct Person {
    name: String,
    age: usize,
}

// Steps:
// 1. If the length of the provided string is less than 2, return an Err.
// 2. Split the given string on the comma.
// 3. Only 2 elements should be returned.
// 4. Extract the first element from the vector and parse it into a name.
// 5. Extract the other element from the vector and parse it into an age.
// 6. If while extracting the name and age, there's something like "4".parse::<error>, return an Err.
// If everything goes well, then return Ok(Person { name, age })
```

The code is compiling, and the tests pass! 🎉

You can keep working on this exercise, or jump into the next one by removing the `I AM NOT DONE` comment:

12 | }
13 | // I AM NOT DONE
14 | // Steps:
15 | // Successfully tested exercises/conversions/from_str.rs

The code is compiling, and the tests pass! 🎉

You can keep working on this exercise, or jump into the next one by removing the `I AM NOT DONE` comment:

12 | }
13 | // I AM NOT DONE
14 | // Steps:
15 | // Successfully tested exercises/conversions/from_str.rs

All exercises completed! 🎉

You made it to the Fe-nish line! 🎉

We hope you enjoyed learning about the various aspects of Rust!
If you noticed any issues, please don't hesitate to report them to our repo.
You can also contribute your own exercises to help the greater community!

Before reporting an issue or contributing, please read our guidelines:
<https://github.com/rust-lang/rustlings/blob/main/CONTRIBUTING.md>

此图留念rustlings完结；

2021.07.11

Rust-quiz完结；

You have answered 33 of 33 questions correctly.

[https://dtolnay.github.io/rust-quiz/1](https://dtolnay.github.io/rust-quiz/)

2021.07.12

由于本人经常使用的编程语言是C++，经陈渝老师推荐，开始阅读[《A Guide to Porting C and C++ code to Rust》](#)；

2021.07.15

看完[《A Guide to Porting C and C++ code to Rust》](#)；

由于本人有Python、Java、C++的使用经验，对于Rust的语法，让我眼前一亮，其结合了不同语言的优缺点，作为一门安全的语言，其编译器的纠错提醒功能让我十分喜欢，希望今后能多用这门语言做出点事情；

2021.07.16

强推CSAPP这本书及本书作者配套的视频课；

回顾CSAPP的笔记，发现这本书及本书作者配套的视频课，让我真的受益匪浅，能让我深层次的理解计算机为什么这样设计，历代不同指令集的发展史，对比新兴起的RISCV指令集，更让我看到了RISCV指令集的活力以及无穷的潜力；

下图为5月~6月观看CSAPP视频课的笔记；

Dynamic Range (Positive Only)

Eg.: (IEEE, 8 bits)

$$2^{8-15} = 1 - L$$

.....1111

$$\text{Bias} = 2^3 - 1 = 7$$

$$V = (-1)^S M 2^E$$

$$n : E = \text{Exp} - \text{Bias}$$

$\rightarrow +10^{\circ} \rightarrow +\text{positive}$

$\times -1) = -1$

Negative

$$\text{Norm.} \quad 0 \quad 0000 \quad 001 \quad -6$$

$$(0 * 2^{-6})$$

$$(1 * 2^{-2}) * 2^{-6} = 1/256 = 1/412$$

Closest to zero

- too

far

$$0 \quad 0000 \quad 110 \quad -6$$

$$(2^1 + 2^2) * 2^{-6} = 6/412$$

$$7/512$$

$$1/255$$

$$1/254$$

$$1/253$$

$$1/252$$

$$1/251$$

$$1/250$$

$$1/249$$

$$1/248$$

$$1/247$$

$$1/246$$

$$1/245$$

$$1/244$$

$$1/243$$

$$1/242$$

$$1/241$$

$$1/240$$

$$1/239$$

$$1/238$$

$$1/237$$

$$1/236$$

$$1/235$$

$$1/234$$

$$1/233$$

$$1/232$$

$$1/231$$

0: represent negative
1: represent positive

$2^8 = 16$

✓ 2. $\exp \approx 000\ldots 0$, $\text{frac} \neq 000\ldots 0$,

① Numbers closest to 0.0

② Equispaced

Special Values

Condition: $\exp = 111\ldots 1$

1. Case: $\exp = 111\ldots 1$, $\text{frac} = 000\ldots 0$
 - ① Represents value ∞
 - ② Operation that overflows
 - ③ Both positive or negative
- E.g.: $1.0/0.0 = 1.0/-0.0 = +\infty$, $1.0/-0.0 = -\infty$

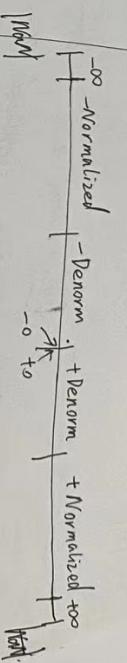
2. Case: $\exp = 111\ldots 1$, $\text{frac} \neq 000\ldots 0$

① Not-a-Number (NaN)

② Used for no answer's questions.

③ E.g., $\sqrt{-1}$, $\infty - \infty$, $\infty \times 0$.

Visualization: Floating Point Encodings



2021.07.17

看RISC-V开发者手册《RISC-V-Reader-Chinese-v2》；

其实这本书我5月初就开始阅读了，但是完全掌握书上的内容，短时间内似乎无法实现；

关于各个不同寄存器的相关用途，目前只能记住常用的寄存器的相关用法，其他生僻的内容，只能靠CTRL+F来查阅相关文档；

2021.07.18

发现两个月前在gitbook下提的建议被采纳了；



beinglida 发表于 大约 2 个月前



在macos上使用github desktop，直接clone，会出现跟BraveY一样的错误；复制终端代理命令（临时），

如下：（不同的代理客户端会有不同的端口号）

```
export https_proxy=http://127.0.0.1:7890 http_proxy=http://127.0.0.1:7890  
all_proxy=socks5://127.0.0.1:7890
```

,

然后clone到本地后，就可以正常跑了

运行命令

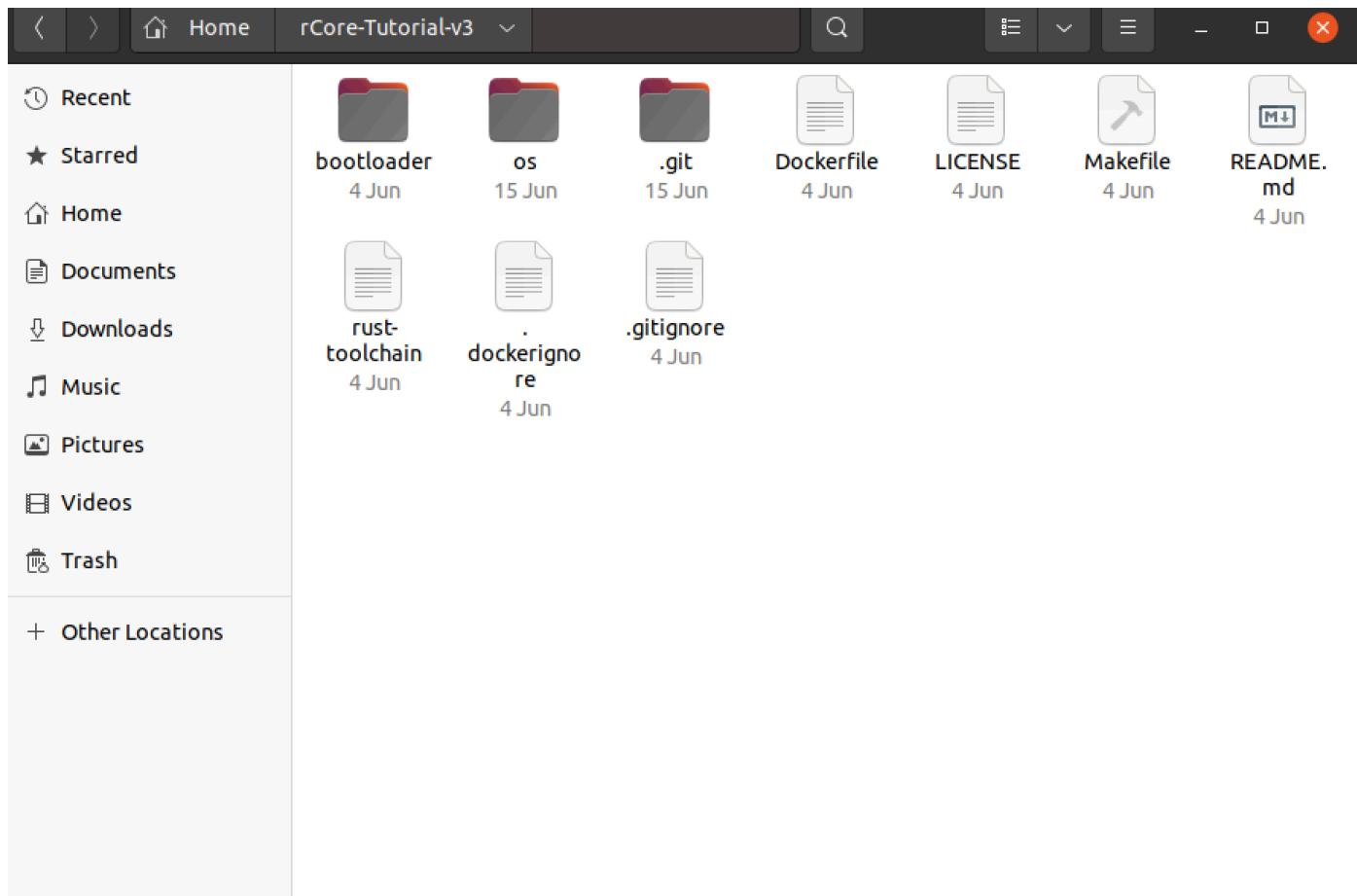
```
1 export RUSTUP_DIST_SERVER=https://mirrors.ustc.edu.cn/rust-static  
2 export RUSTUP_UPDATE_ROOT=https://mirrors.ustc.edu.cn/rust-static/rustup  
3 curl https://sh.rustup.rs -sSf | sh
```

或者也可以通过在运行前设置命令行中的科学上网代理来实现：

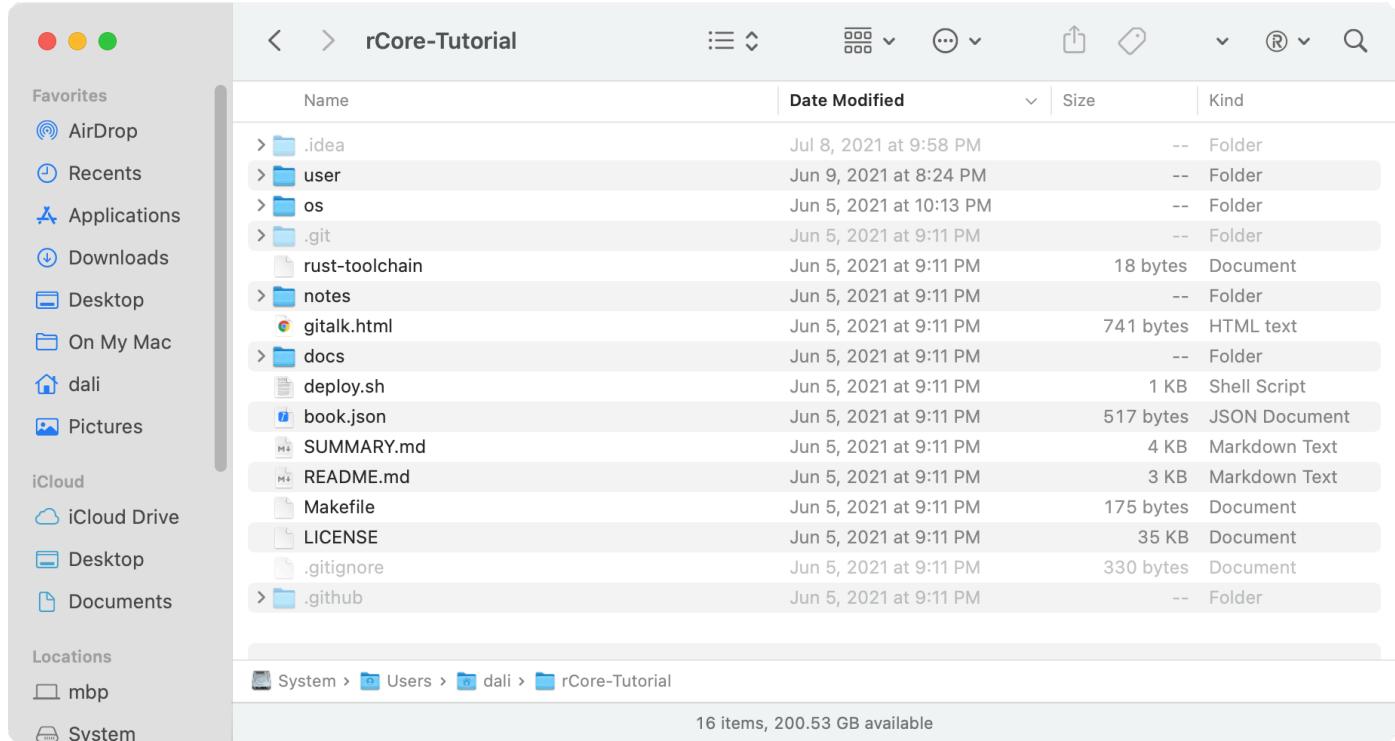
运行命令

```
1 # e.g. Shadowsocks 代理  
2 export https_proxy=http://127.0.0.1:1080  
3 export http_proxy=http://127.0.0.1:1080  
4 export ftp_proxy=http://127.0.0.1:1080
```

把rCore-Tutorial部署到Linux：



把rCore-Tutorial部署到MacOS:



MacOS上启动rcore, 至此, lab0完结;

```

qemu-system-riscv64
-zsh      ¶ 1      qemu-system-riscv64      ¶ 2
Compiling xmas-elf v0.7.0
Compiling hashbrown v0.8.2
Compiling virtio-drivers v0.1.0 (https://github.com/rcore-os/virtio-drivers#2b3c6cf)
Compiling rustc_version v0.2.3
Compiling bare-metal v0.2.5
Compiling aho-corasick v0.7.18
Compiling buddy_system_allocator v0.4.0
Compiling rcore-fs-sfs v0.1.0 (https://github.com/rcore-os/rcore-fs#6df6cd24)
Compiling regex v1.5.4
Compiling riscv-target v0.1.2
Compiling riscv v0.6.0 (https://github.com/rcore-os/riscv#b6c469f0)
Compiling os v0.1.0 (/Users/dali/rCore-Tutorial/os)
Finished dev [unoptimized + debuginfo] target(s) in 14.30s

OpenSBI v0.9

    /--\   /---- ---- ----
   |  | | | \_ -- _ -- - -- | ( _ | |_) || |
   |  | | | | '_ \ / _ \ ' \ \_ \ \_ \| _ < | |
   |  | _| | |_) | _/ | | |_) | |_) || |_
 \_ \_ /| . / \_ \_ | | | | _ / | _ / | _ |
   | |
   |_|

Platform Name          : riscv-virtio,qemu
Platform Features       : timer,mfdeleg
Platform HART Count    : 1
Firmware Base          : 0x80000000
Firmware Size           : 100 KB
Runtime SBI Version     : 0.2

Domain0 Name            : root
Domain0 Boot HART        : 0
Domain0 HARTs             : 0*
Domain0 Region00         : 0x0000000080000000-0x000000008001ffff ( )
Domain0 Region01         : 0x0000000000000000-0xffffffffffff (R,W,X)
Domain0 Next Address      : 0x0000000000000000
Domain0 Next Arg1        : 0x0000000008700000
Domain0 Next Mode          : S-mode
Domain0 SysReset          : yes

Boot HART ID              : 0
Boot HART Domain           : root
Boot HART ISA                : rv64imafdcu
Boot HART Features           : scounteren,mcounteren,time
Boot HART PMP Count          : 16
Boot HART PMP Granularity : 4
Boot HART PMP Address Bits: 54
Boot HART MHPM Count          : 0
Boot HART MHPM Count          : 0
Boot HART MIDELEG           : 0x0000000000000222
Boot HART MEDELEG           : 0x000000000000b109

```

2021.07.19

Lab1:

```

dali@mbp lab0 % make run
make: *** No rule to make target `run'. Stop.
dali@mbp lab0 % cd os
dali@mbp os % ls

```

```
dali@mbp os % ls
Cargo.toml      Makefile      src
dali@mbp os % make run
Compiling os v0.1.0 (/Users/dali/rcore/labs/lab0/os)
Finished dev [unoptimized + debuginfo] target(s) in 0.63s
```

OpenSBI v0.9



```
Platform Name          : riscv-virtio,qemu
Platform Features       : timer,mfdeleg
Platform HART Count    : 1
Firmware Base          : 0x80000000
Firmware Size           : 100 KB
Runtime SBI Version     : 0.2

Domain0 Name            : root
Domain0 Boot HART        : 0
Domain0 HARTs             : 0*
Domain0 Region00         : 0x0000000080000000-0x000000008001ffff ()
Domain0 Region01         : 0x0000000000000000-0xffffffffffff (R,W,X)
Domain0 Next Address      : 0x0000000000000000
Domain0 Next Arg1         : 0x0000000087000000
Domain0 Next Mode          : S-mode
Domain0 SysReset          : yes

Boot HART ID              : 0
Boot HART Domain           : root
Boot HART ISA                : rv64imafdcu
Boot HART Features          : scounteren,mcounteren,time
Boot HART PMP Count         : 16
Boot HART PMP Granularity   : 4
Boot HART PMP Address Bits: 54
Boot HART MHPM Count        : 0
Boot HART MHPM Count        : 0
Boot HART MIDELEG           : 0x00000000000000222
Boot HART MEDELEG           : 0x000000000000b109
```

lab没做完，后续有精力再研究研究；

2021.07.23

玩硬件，熟悉树莓派RISCV开发板、树莓派ARM开发板、某空气质量检测仪（芯片为MXCHIP EMW3080）的电路的基本结构，观看EMW3080的开发者手册，以及Windows的调试软件（SecureCRT）、硬件（J-LINK）、驱动（J-LINK）；

观看了某大神关于智能设备的教程，希望能把该功能应用于riscv板子上，链接如下：

<https://github.com/a2633063>

阅读了《物联网安全百科》，熟悉相关软硬件安全策略，未曾接触过此类知识，里面有关于软硬件调试的方法：

<https://iot-security.wiki/hardware-security/firmware/reverse.html>

2021.07.24

找到一些no_std的基于rust语言的gba项目，后续会把这些项目合并移植到zcore上；

arm-embedded（可以移植）：

<https://crates.io/crates/gba/versions>

[no_std]的（现成轮子）：

<https://github.com/ryankurte/rust-gba>

2021.07.25

今天才开始学习git命令；

项目没能成功push到仓库；

月初的时候的遗留文件未能成功清理；

2021.07.26

萧洛元的指点下，发现了codechina仓库push权限关闭了，在codechina上打开权限后，成功git push成功；

现在把我从5月初到7月底的大部分工作传到本仓库；