CloudWeGo

# Rust 语言简介

## Rust 开发实训

字节跳动

2023/09/02

# 目录

CONTENT

# 01 课程简介

# 学前准备

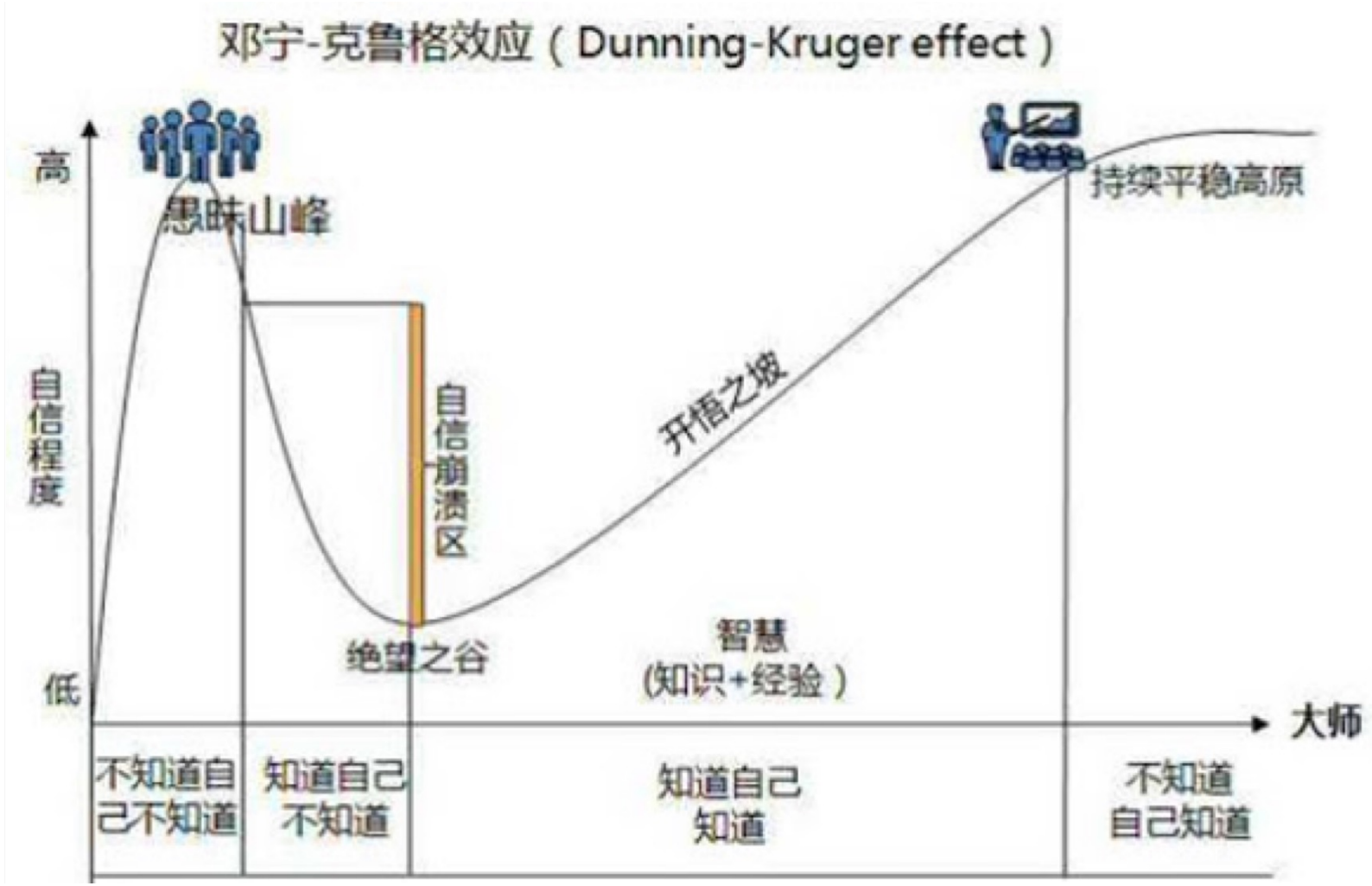
对你的期望：

- 有至少一门编程语言经验
- 了解基础的 Linux 命令使用及相关知识
- 有操作系统、数据结构基础更好
- 有能力访问 github 等开源网站

# 学习方法

- 夯实计算机基础知识
- 学会“知识屏蔽”：把部分知识当黑盒
- 学会类比已有的知识
- 多阅读标准库和优秀开源项目源码
- 以 Rust 的方式来思考
- 一万小时定律：多写！多写！再多写！

# 邓克效应

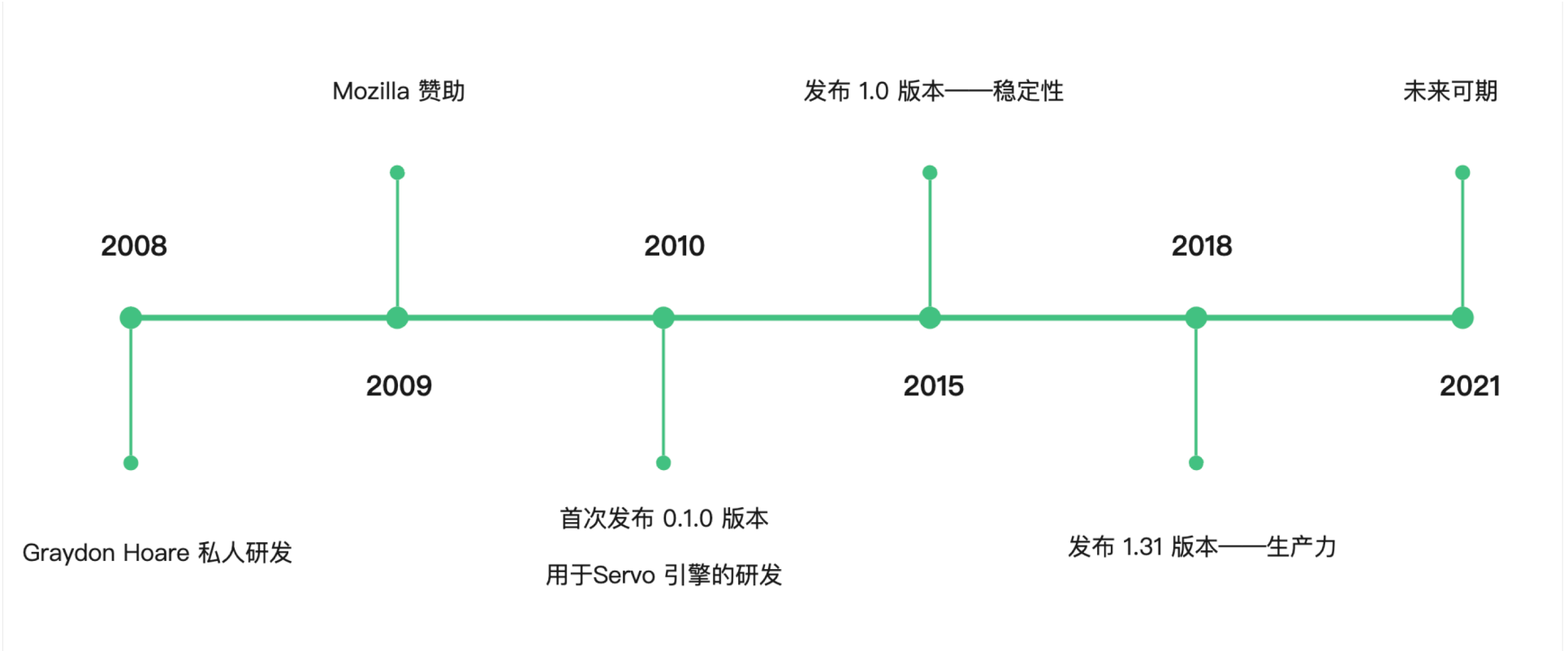

邓宁-克鲁格效应（Dunning-Kruger effect）
高
自信程度
低
愚昧山峰
自信崩溃区
绝望之谷
开悟之坡
持续平稳高原
智慧
(知识+经验)
大师
不知道自己不知道
知道自己不知道
知道自己知道
不知道自己知道

# 预期收获

- 掌握 Rust 基础和异步相关理论知识
- 独立完成 Rust 中等复杂项目的设计和开发

# 02 为什么要学习 Rust?

# Rust 历史


2008
Graydon Hoare 私人研发
2009
Mozilla 赞助
2010
首次发布 0.1.0 版本
用于Servo 引擎的研发
2015
发布 1.0 版本——稳定性
2018
发布 1.31 版本——生产力
2021
未来可期

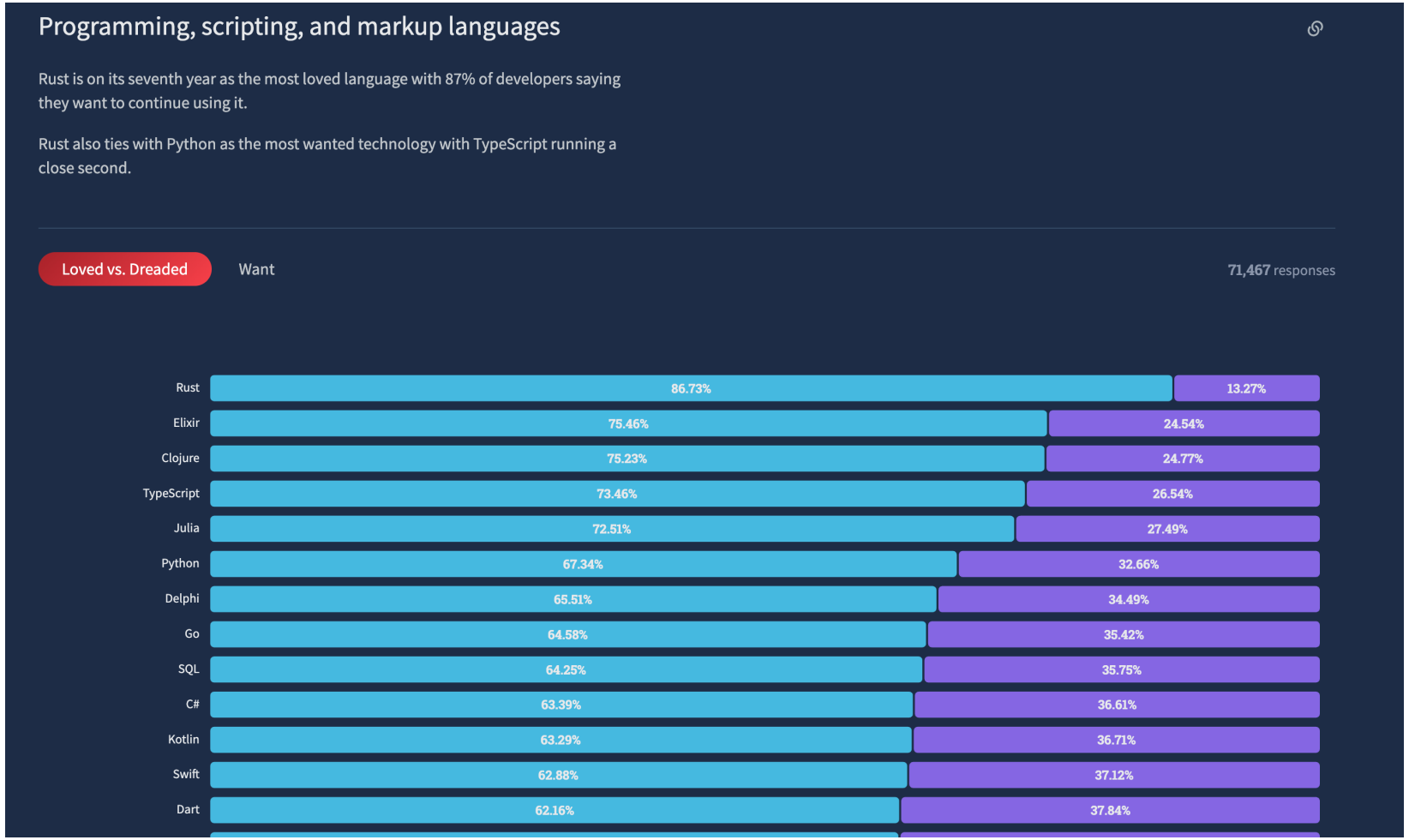
# Rust 2024: scaling empowerment

Rust's goal is to **empower everyone to build reliable and efficient software**. Success requires not only designing and implementing a great language with great libraries and great tools, but also maintaining a great and supportive community.

Our focus for Rust 2024 is to **scale empowerment** in many different ways. As we grow, we face increasing challenges in how we can scale the ways in which we empower people to an increasing number of people. This roadmap presents three general themes we plan to focus on:

- **Flatten the (learning) curve**: scaling to new users and new use cases
  - Make Rust more accessible to new and existing users alike, and make solving hard problems easier.
- **Help Rust's users help each other**: scaling the ecosystem
  - Empower library authors so they can---in turn---empower their users.
- **Help the Rust project scale**: scaling the project
  - Develop processes to scale to the needs and use cases of a growing number of users; evaluate and finish projects we've started.

# StackOverflow 开发者调研



# 企业应用

| | | | |
|---|---|---|---|
| 字节跳动 | 全方位应用：服务端、客户端、虚拟化、安全、前端、WASM 等等 | 蚂蚁金服 | 时序数据库、安全 OS |
| Google | Chrome、Android、Fuchsia，内部 Rust 开发者 1000+ | Meta | 服务端 |
| Microsoft | Azure 云、Windows 内核 | Github | 代码搜索引擎 |
| 华为 | 全方位应用 | Amazon | AWS |
| Linux | Rust for Linux | … | |

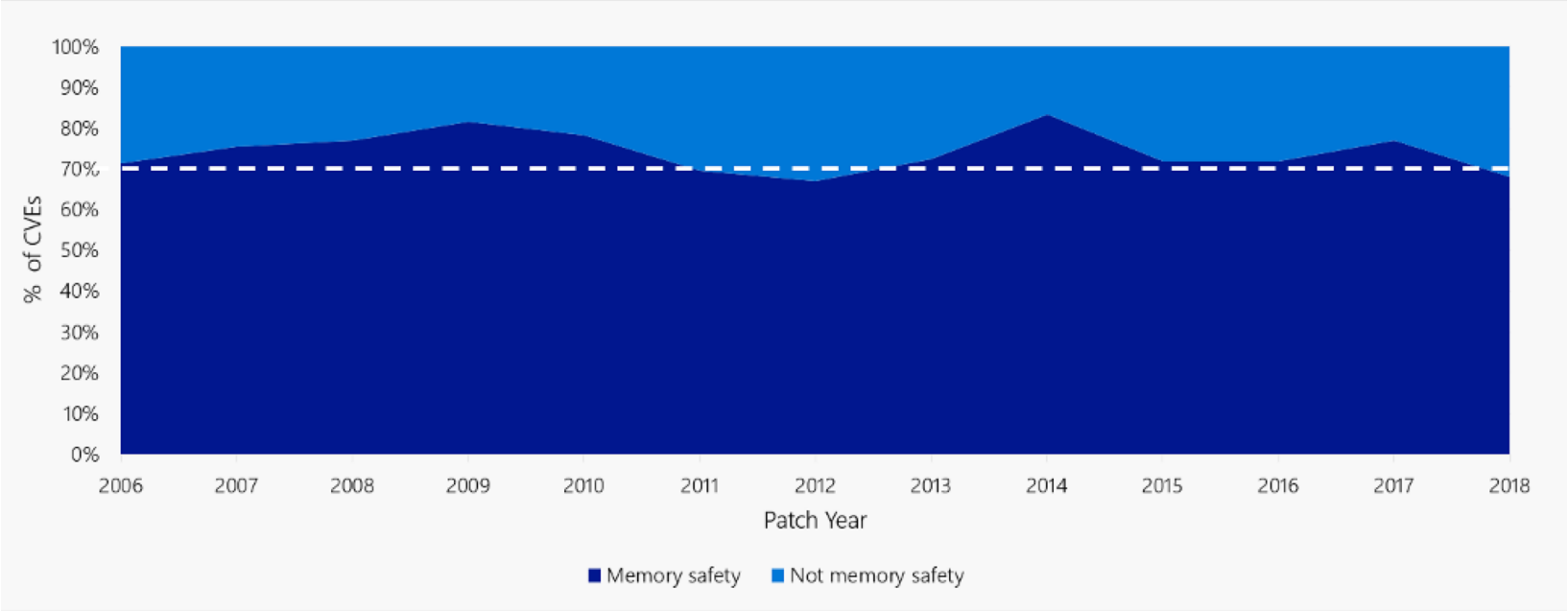
# Rust 特性

# Benchmark

| source | secs | mem | cpu load |
|---|---|---|---|
| Rust | 0.77 | 147,384 | 55% 59% 55% 91% |
| Go | 3.85 | 324,200 | 27% 19% 20% 91% |
| C gcc | 0.8 | 152,172 | 52% 99% 48% 53% |
| C clang | 3.12 | 103,044 | 100% 0% 1% 0% |
| C++ g++ | 1.1 | 203,924 | 63% 77% 71% 100% |

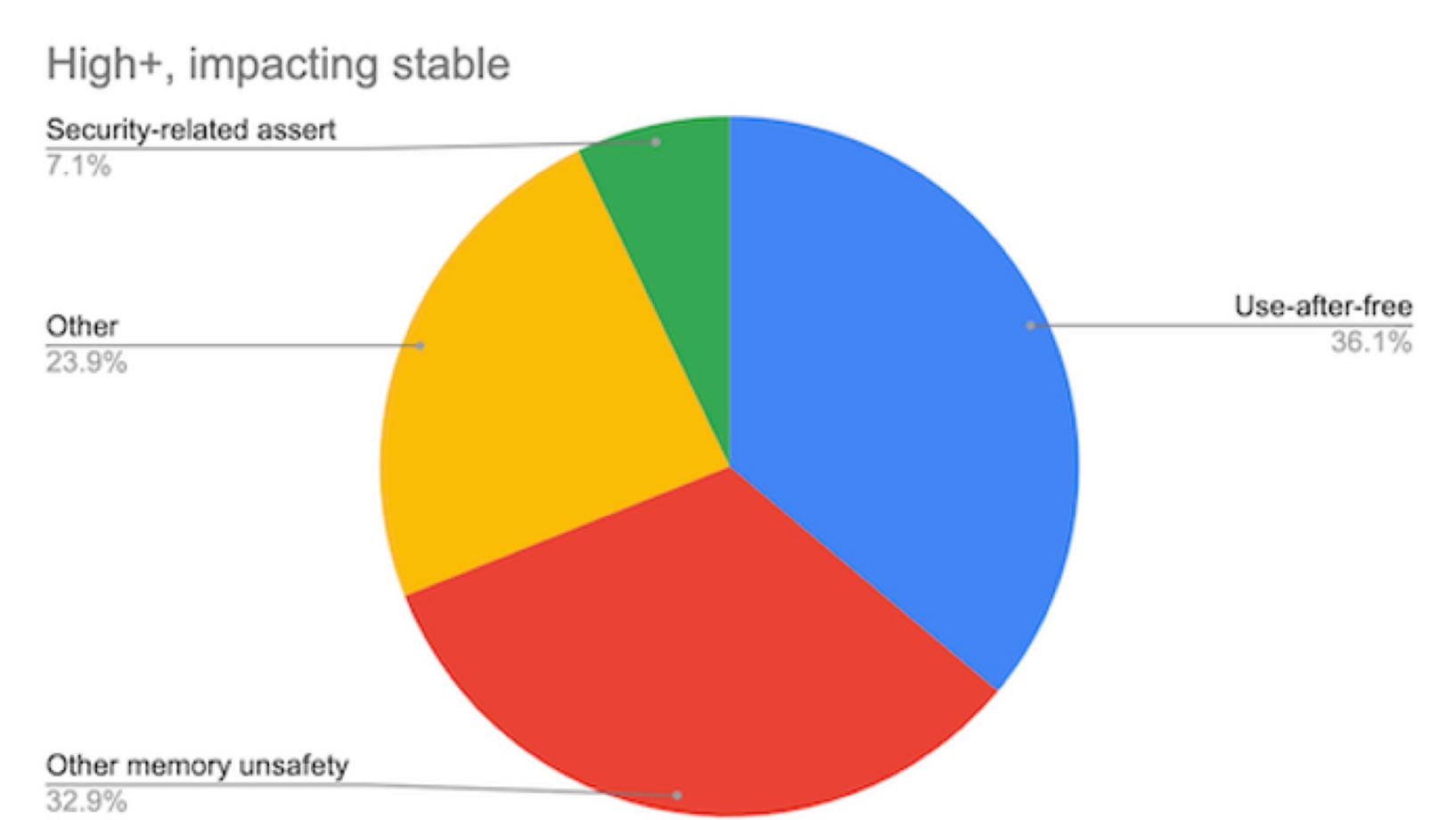
# 安全

微软：70% 的安全问题是内存安全导致的



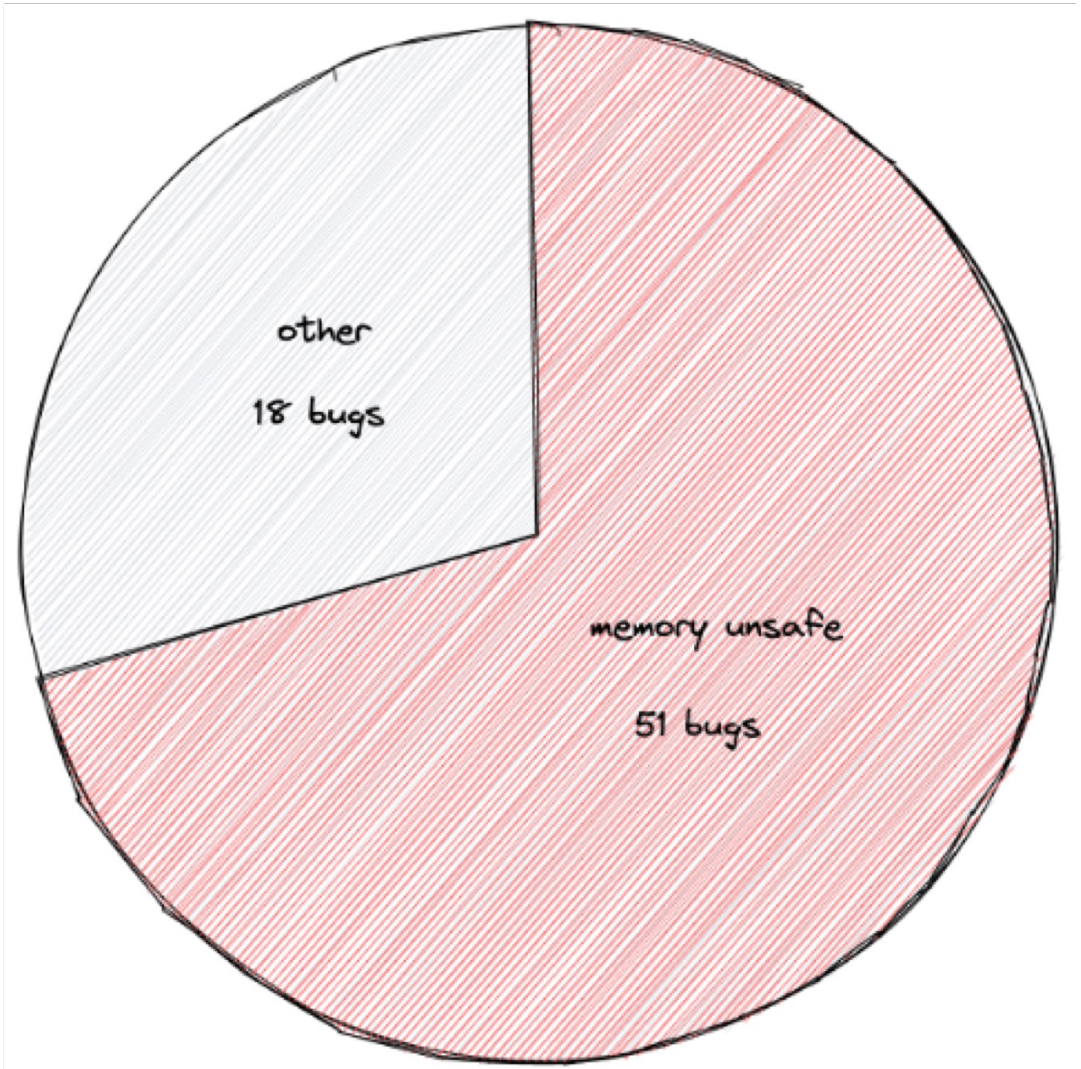
# 安全

## Chromium: 同样也是70%



# 安全

Firefox css component: 73%



# 安全：和我有什么关系？

对于生产产品，没有安全，一切都没有意义！

# 想一想

```
int print(int *a) {
    printf("%d", *a);
}
```

# 想一想

```
int* add(int a, int b) {
    int c = a + b;
    return &c;
}
```

## 想一想

```
Conn* accept_conn();
void handle_conn(Conn *conn);
void close_conn(Conn *conn);

int event_loop() {
    while (1) {
        try {
            Conn *conn = accept_conn();
            handle_conn(conn);
            close_conn(conn);
        } catch(...) {
            std::cout << "exception" << std::endl;
        }
    }
}
```

# 协作

工程实践出来的语言：

- 智能的编译器
- 完善的文档
- 齐全的工具链
- 成熟的包管理

信任别人的代码

# Rust 是什么

- 新时代的全能型系统级语言：上得了业务，下得了内核
- C/C++ 无损 FFI
- 安全、性能强、协作好、开发效率高
- 生态蓬勃发展

# Rust 能带给你的

- Rust 让你的编程经验更有价值
- 丰富的生态，优秀的工程实践
- 基于安全和性能之上的软件大厦

# 03 Rust 基础概念

# 安装 Rust

## https://rustup.rs/

**rustup** is an installer for
the systems programming language **Rust**

Run the following in your terminal, then follow the
onscreen instructions.

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

You appear to be running Unix. If not, display all supported installers.

Need help?
Ask on #beginners in the Rust Discord
or in the Rust Users Forum.

rustup is an official Rust project.

other installation options · component availability · about rustup

# 安装 Rust

## https://rsproxy.cn/

RsProxy.cn

### 用法

crates.io Mirror

~/.cargo/config:

```
[source.crates-io]
replace-with = 'rsproxy'

[source.rsproxy]
registry = "https://rsproxy.cn/crates.io-index"

[registries.rsproxy]
index = "https://rsproxy.cn/crates.io-index"

[net]
git-fetch-with-cli = true
```

Rustup Mirror

~/.zshrc or ~/.bashrc:

```
export RUSTUP_DIST_SERVER="https://rsproxy.cn"
export RUSTUP_UPDATE_ROOT="https://rsproxy.cn/rustup"
```

Install Rust:

```
# export the env above first
curl --proto '=https' --tlsv1.2 -sSf
https://rsproxy.cn/rustup-init.sh | sh
```

# 安装 Rust

```
export RUSTUP_DIST_SERVER="https://rsproxy.cn"
export RUSTUP_UPDATE_ROOT="https://rsproxy.cn/rustup"
```

```
# export the env above first
curl --proto '=https' --tlsv1.2 -sSf https://rsproxy.cn/rustup-init.sh | sh
```

# 安装 Rust

~/.cargo/config

```
[source.crates-io]
replace-with = 'rsproxy'

[source.rsproxy]
registry = "https://rsproxy.cn/crates.io-index"

[registries.rsproxy]
index = "https://rsproxy.cn/crates.io-index"

[net]
git-fetch-with-cli = true
```

# Rustup

Rustup 工具链管理工具：

切换工具链：rustup default stable/beta/nightly

更新：rustup update

# Hello Rust


cargo new hello-rust


chapter-02
├── Cargo.lock # Cargo 依赖版本锁定文件
├── Cargo.toml # Cargo 主要设置文件
└── src
    └── main.rs # Rust 程序入口

# Hello Rust

```rust
use std::io; // 使用标准库中的 io 这个模块

fn main() {
    println!("Hello, world!");
    println!("Please input a number: ");
    let mut input = String::new(); // 在这里我们创建了一个新的 String，用来接收下面的输入
    io::stdin()
        .read_line(&mut input) // 读取一行
        .expect("Failed to read input!"); // 比较粗暴的错误处理
    println!("Your raw input is: {:?}.", input); // 打印输入的原始内容
    let number: i64 = input.trim().parse().expect("Input is not a number!"); // trim 把前后的空格、换行符这些空白字符都去掉，parse 将输入的字符串解析为 i64 类型，如果解析失败就报错
    println!("Your input is: {}.", number); // 打印 parse 之后的 i64 数字
}
```

# Hello Rust

```
$ cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
     Running `target/debug/hello-rust`
Hello, world!
Please input a number:
1024
Your raw input is: "1024\n".
Your input is: 1024.
```

```
$ cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
     Running `target/debug/hello-rust`
Hello, world!
Please input a number:
abcd
Your raw input is: "abcd\n".
thread 'main' panicked at 'Input is not a number!: ParseIntError { kind: InvalidDigit }',
src/main.rs:11:44
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

# Hello Rust


$ cargo clean
$ cargo build -v
Running `rustc --crate-name hello-rust --edition=2021 ......


$ rustc src/main.rs
$ ./main
Hello, world!
Please input a number:
1024
Your raw input is: "1024\n".
Your input is: 1024.

# Cargo

Cargo 包管理工具：

- 新建一个项目：cargo new
- 格式化代码：cargo fmt
- 检查 && Lint：cargo check && cargo clippy
- 编译：cargo build (--release)

# Rustc

Rustc 编译器：

- 用来编译 rust 代码
- 一般通过 Cargo 调用

# 变量声明及数据类型

```rust
use std::io;

fn main() {
    println!("Hello, world!");
    println!("Please input a number: ");
    let mut input = String::new(); // 这一行就是变量声明，let mut 代表声明的是一个可变的变量，如果不加 mut 默认声明的就是一个不可变的变量
    io::stdin()
        .read_line(&mut input) // 这里我们拿到了 input 这个变量的可变引用，也因此在声明的时候需要声明称可变变量，关于引用我们会在后续章节中详细介绍
        .expect("Failed to read input!");
    println!("Your raw input is: {:?}.", input);
    let number: i64 = input.trim().parse().expect("Input is not a number!"); // 这里的 number 是不可变的，类型我们指定为了 i64，在这里指定的原因是为了告诉编译器 parse 需要解析成什么类型，因为 String 可以 parse 成多个类型，所以在这里 Rust 编译器无法自行推断出来
    println!("Your input is: {}.", number);
}
```

# 变量声明及数据类型

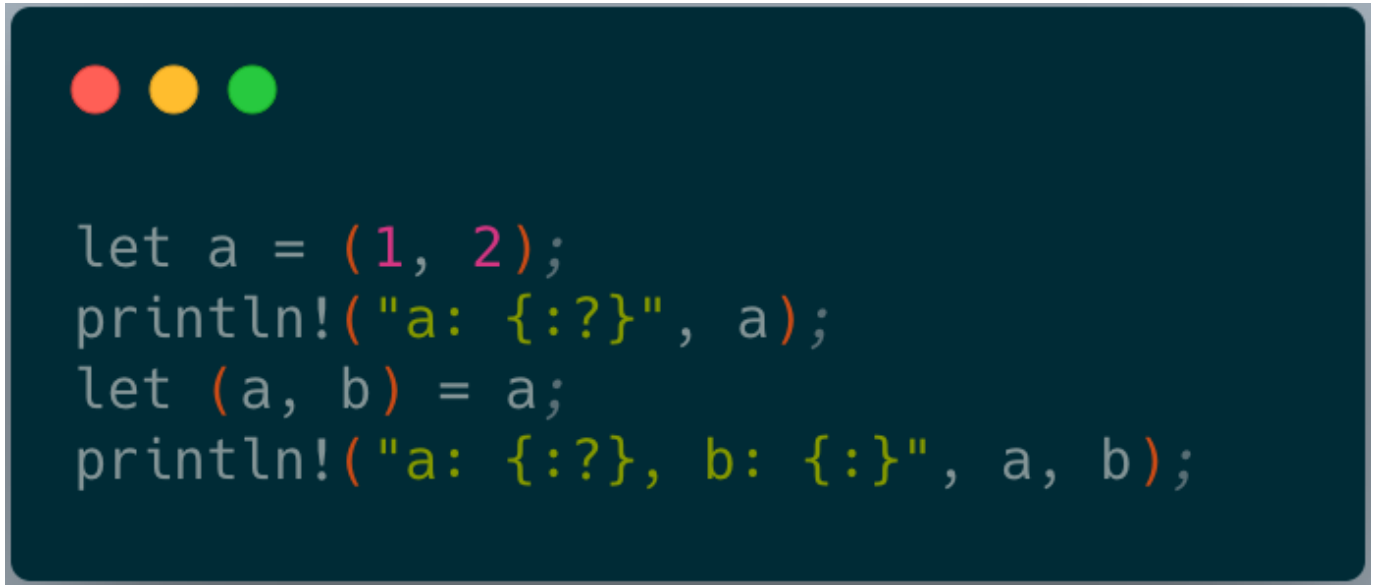
```rust
use std::io;

fn main() {
    println!("Hello, world!");
    println!("Please input a number: ");
    let input = String::new(); // 把这里的 mut 去掉，会发生什么？
    io::stdin()
        .read_line(&mut input)
        .expect("Failed to read input!");
    println!("Your raw input is: {:?}.", input);
    let number: i64 = input.trim().parse().expect("Input is not a number!");
    println!("Your input is: {}.", number);
}
```

```
$ cargo check
    Checking hello-rust v0.1.0
error[E0596]: cannot borrow `input` as mutable, as it is not declared as mutable
 --> src/main.rs:8:20
  |
8 |         .read_line(&mut input)
  |                    ^^^^^^^^^^ cannot borrow as mutable
  |
help: consider changing this to be mutable
  |
6 |     let mut input = String::new();
  |         +++

For more information about this error, try `rustc --explain E0596`.
error: could not compile `hello-rust` due to previous error
```

# 元组



```rust
let a = (1, 2);
println!("a: {:?}", a);
let (a, b) = a;
println!("a: {:?}, b: {:}", a, b);
```

# 元组

| Length | Signed | Unsigned |
| --- | --- | --- |
| 8-bit | `i8` | `u8` |
| 16-bit | `i16` | `u16` |
| 32-bit | `i32` | `u32` |
| 64-bit | `i64` | `u64` |
| 128-bit | `i128` | `u128` |
| arch | `isize` | `usize` |

| Number literals | Example |
| --- | --- |
| Decimal | `98_222` |
| Hex | `0xff` |
| Octal | `0o77` |
| Binary | `0b1111_0000` |
| Byte (`u8` only) | `b'A'` |

# 其它标量类型和复合类型

除了整型之外，Rust 还有以下标量类型：

- 浮点数：f32 / f64
- bool
- char：这个比较特殊，Rust 中一个 char 占 4 字节，存放的是一个 UTF-32，而不像 C/C++ 那样本质上是个 u8

除了标量类型之外，Rust 还支持两种复合类型：

- 元组 tuple：let a = (1, 2); let (a, b) = (1, 2)
- 数组 array: let a = [1, 2, 3]; let a = [0; 5] // 这个声明中 0 是默认值，5 是长度，等价于 let a = [0, 0, 0, 0, 0]

# 函数

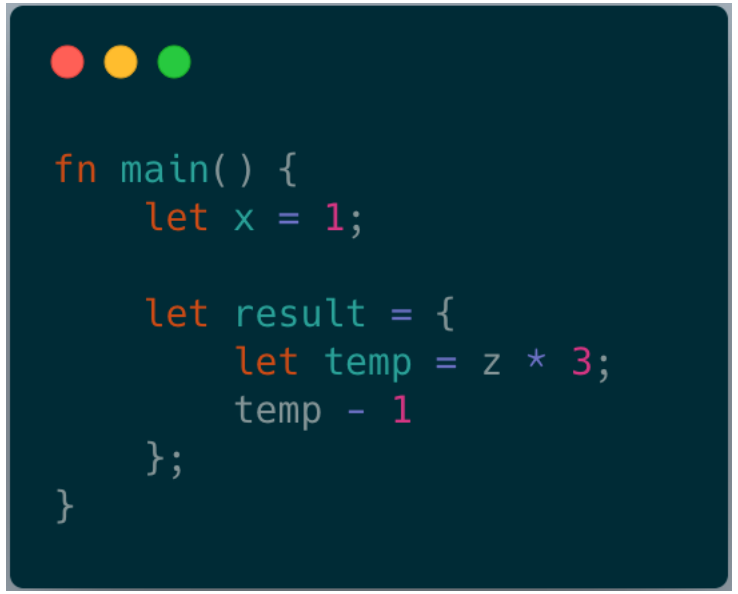
```rust
fn greet() {
    println!("Welcome!");
}

fn say_hello(name: String) {
    println!("Hello, {}", name);
}

fn main() {
    greet();

    let name = "Alice".to_string();
    say_hello(name);
}
```

# 语句和表达式



```rust
fn main() {
    let x = 1;

    let result = {
        let temp = z * 3;
        temp - 1
    };
}
```

# 函数返回值

```rust
fn add(a: i32, b: i32) -> i32 {
    a + b
}

fn pass_exam(score: u32) -> bool {
    if score < 60 {
        return false;
    }
    true
}

fn main() {
    let num1 = 5;
    let num2 = 7;

    let sum = add(num1, num2);
    println!("The sum of {} and {} is {}.", num1, num2, sum);
}
```

# 函数返回值

```rust
fn add(a: i32, b: i32) -> i32 {
    a + b;
}
```

```
error[E0308]: mismatched types
 --> src/main.rs:1:27
  |
1 | fn add(a: i32, b: i32) -> i32 {
  |    ---                    ^^^ expected `i32`, found `()`
  |    |
  |    implicitly returns `()` as its body has no tail or `return` expression
2 |     a + b;
  |          - help: remove this semicolon to return this value

For more information about this error, try `rustc --explain E0308`.
error: could not compile `playground` due to previous error
```

# 控制流

```
if condition1 {
    // code to execute if condition1 is true
} else if condition2 {
    // code to execute if condition1 is false and condition2 is true
} else {
    // code to execute if both conditions are false
}
```

```
fn main() {
    let temperature = 20;

    let weather = if temperature >= 25 {
        "hot"
    } else {
        "cool"
    };

    println!("The weather today is {}.", weather);
}
```

# 控制流

```rust
fn main() {
    let temperature = 20;

    let weather = if temperature >= 25 {
        "hot"
    } else {
        25
    };

    println!("The weather today is {}.", weather);
}
```

```
error[E0308]: `if` and `else` have incompatible types
 --> src/main.rs:7:9
  |
4 |       let weather = if temperature >= 25 {
  |  ___________________-
5 | |         "hot"
  | |         ----- expected because of this
6 | |     } else {
7 | |         25
  | |         ^^ expected `&str`, found integer
8 | |     };
  | |_____- `if` and `else` have incompatible types

For more information about this error, try `rustc --explain E0308`.
error: could not compile `playground` due to previous error
```

# 控制流

```
loop {
    // code to execute repeatedly
}
```

```
let mut counter = 0;

loop {
    counter += 1;
    if counter < 5 {
        continue;
    }
    println!("Hello, world!");
    if counter >= 5 {
        break;
    }
}
```

# 控制流

```rust
fn main() {
    let target = 10;
    let mut sum = 0;
    let mut counter = 1;

    let result = loop {
        sum += counter;

        if sum >= target {
            break counter; // The value of counter will be returned from the loop as a result
        }

        counter += 1;
    };

    println!("The first number whose sum of all previous numbers is greater than or equal to {} is {}.",
target, result);
}
```

# 控制流

```
while condition {
    // code to execute while the condition is true
}
```

# 控制流

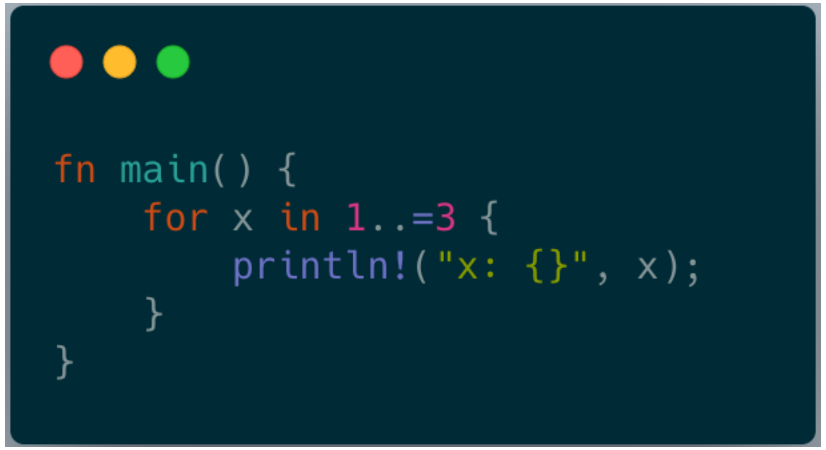
```rust
fn main() {
    let numbers = [1, 2, 3, 4, 5];
    let mut index = 0;

    while index < numbers.len() {
        println!("The value is: {}", numbers[index]);
        index += 1;
    }
}
```

```rust
fn main() {
    let numbers = [1, 2, 3, 4, 5];

    for number in numbers {
        println!("The value is: {}", number);
    }
}
```

# 控制流



```rust
fn main() {
    for x in 1..=3 {
        println!("x: {}", x);
    }
}
```

# 控制流

```rust
fn main() {
    let x = 1;

    'outer: loop {
        let mut y = 1;

        'inner: loop {
            if y == 3 {
                y += 1;
                continue 'inner; // Skips to the next iteration of the 'inner loop
            }

            println!("x: {}, y: {}", x, y);

            y += 1;

            if y > 5 {
                break 'outer; // Breaks out of the 'inner loop
            }
        }
    }
}
```

# 控制流

```rust
fn main() {
    let x = 1;

    let z = 'outer: loop {
        let mut y = 1;

        'inner: loop {
            if y == 3 {
                y += 1;
                continue 'inner; // Skips to the next iteration of the 'inner loop
            }

            println!("x: {}, y: {}", x, y);

            y += 1;

            if y > 5 {
                break 'outer y; // Breaks out of the 'inner loop
            }
        }
    };
    println!("z: {}", z);
}
```

# 控制流

```rust
fn main() {
    'outer: for x in 1..=3 {
        for y in 1..=5 {
            if y == 3 {
                continue 'outer; // Skips the current iteration of the 'outer loop
            }

            if x == 2 && y == 4 {
                break 'outer; // Breaks out of the 'outer loop
            }

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

# String

```c
void print(char *s) {
    printf("%s\n", s);
}

int main() {
    char *s = (char *)malloc(10);
    print(s);
    return 0;
}
```

# String

```c
void print(char *s) {
    printf("%s\n", s);
}

int main() {
    char *s = (char *)malloc(10);
    print(s);
    return 0;
}
```

# 所有权

- Rust中每个值都有一个所有者
- 一个值同时只能有一个所有者
- 当所有者离开作用域范围，这个值将被丢弃

# 所有权

```rust
fn main() {
    // 规则 1：Rust中每个值都有一个所有者
    let s1 = String::from("hello"); // s1 是值 "hello" 的所有者

    {
        // 规则 2：一个值同时只能有一个所有者
        let s2 = s1; // 所有权从 s1 转移到 s2（s1 不再有效）
        // println!("{}", s1); // 这会导致编译时错误，因为 s1 不再有效
        println!("{}", s2); // 这是允许的，因为 s2 现在拥有该值
    } // s2在此处超出范围

    // 规则 3：当所有者离开作用域范围，这个值将被丢弃
    // 由于 s2 超出范围，因此为值 "hello" 分配的内存在此处自动释放。
}
```

# 所有权的转移

```rust
fn takes_ownership(s: String) {
    println!("Received string: {}", s);
} // s 离开作用域，被丢弃

fn gives_ownership() -> String {
    String::from("hello")
} // 返回了String的所有权

fn main() {
    let s = String::from("hello");
    takes_ownership(s); // s转移到了函数内，不再可用

    // s 不再可用

    let s = gives_ownership(); // s 获得了返回值的所有权
}
```

# 不可变借用

```rust
fn main() {
    let s1 = String::from("hello");
    let len = calculate_length(&s1);
    println!("The length of '{}' is {}.", s1, len);
}

fn calculate_length(s: &String) -> usize {
    s.len()
}
```

# 可变借用

```rust
fn main() {
    let mut s = String::from("hello");
    change(&mut s);
    println!("The updated string is: {}", s);
}

fn change(s: &mut String) {
    s.push_str(", world!");
}
```

# 借用规则

- 对于一个变量，同时只能存在一个可变引用或者多个不可变引用

```rust
fn main() {
    let mut s = String::from("hello");

    // 多个不可变引用是允许的
    let r1 = &s;
    let r2 = &s;
    println!("r1: {}, r2: {}", r1, r2);
    // 在这里，多个不可变引用是允许的，因此打印不会引发错误。

    // 一个可变引用
    let r3 = &mut s;
    println!("r3: {}", r3);
    // 在这里，只有一个可变引用，因此打印不会引发错误。

    // 不允许同时存在可变引用和不可变引用
    // let r4 = &s; // 这会导致编译时错误
    // println!("r3: {}, r4: {}", r3, r4);
    // 如果取消注释上面两行，同时存在可变引用和不可变引用将导致编译时错误。
}
```

# 切片（slice）

```rust
let s = String::from("hello");
let len = s.len();

//以0开始时，0可以省略
let slice = &s[0..2];
let slice = &s[..2];

//以最后一位结束时，len可以省略
let slice = &s[3..len];
let slice = &s[3..];

//同时满足上述两条，那么两头都可以省略
let slice = &s[0..len];
let slice = &s[..];
```

# struct

```
struct Point {
    x: f64,
    y: f64,
}

let point = Point {
    x: 3.0,
    y: 4.0
};

let mut point = Point { x: 3.0, y: 4.0 };
point.x = 5.0;
point.y = 6.0;
```

# struct

```rust
struct User {
    username: String,
    email: String,
    age: u8,
}

fn main() {
    let user1 = User {
        username: String::from("Alice"),
        email: String::from("alice@example.com"),
        age: 18,
    };

    let user2 = User {
        username: user1.username,
        email: String::from("alice2@example.com"),
        age: user1.age,
    };

    // 等价写法
    let user2 = User {
        email: String::from("alice2@example.com"),
        ..user1
    };
}
```

# struct

```
struct Color(u8, u8, u8);

let red = Color(255, 0, 0);

struct Marker;

fn main() {
    let marker = Marker;
}
```

# struct

```rust
struct Person {
    name: String,
}

impl Person {
    fn new(name: String) -> Self {
        Self { name }
    }

    // Immutable method
    fn greet(&self) {
        println!("Hello, my name is {}", self.name);
    }

    // Mutable method
    fn change_name(&mut self, new_name: String) {
        self.name = new_name;
    }

    // Consuming method
    fn get_name_and_consume(self) -> String {
        self.name
    }
}

fn main() {
    let mut person = Person::new(String::from("Alice"));

    // Call the immutable method
    person.greet();

    // Call the mutable method
    person.change_name(String::from("Bob"));
    person.greet();

    // Call the consuming method
    let name = person.get_name_and_consume();
    println!("The person's name was: {}", name);

    // The following line would cause a compile error because `person` has been consumed
    // person.greet();
}
```

# enum

```rust
enum Color {
    Red,
    Yellow,
    Green,
}

struct Price(u64);

enum FruitBox {
    Apple(Color),
    Pear(Color, Price),
}

fn main() {
    let _ = FruitBox::Apple(Color::Red);
    let _ = FruitBox::Pear(Color::Green, Price(50));
}
```

```rust
enum Color {
    Red,
    Yellow,
    Green,
}

enum FruitBox {
    Apple(Color),
    Pear{color: Color, price: u64},
}

fn main() {
    let _ = FruitBox::Apple(Color::Red);
    let _ = FruitBox::Pear{color: Color::Green, price: 50};
}
```

# match

```
fn eat_or_sell(fruit_box: FruitBox) {
    match fruit_box {
        FruitBox::Apple(_) => eat_apple(),
        FruitBox::Pear(_, price) => sell_pear(price),
    }
}
```

```
fn eat_or_sell(fruit_box: FruitBox) {
    match fruit_box {
        FruitBox::Apple(_) => eat_apple(),
        _ => do_nothing(),
    }
}
```

# if let

```rust
fn eat_if_is_apple(fruit_box: FruitBox) {
    if let FruitBox::Apple(_) = fruit_box {
        eat_apple();
    }
}
```

```rust
fn do_a_lot_thing_if_is_apple(fruit_box: FruitBox) {
    let FruitBox::Apple(apple_color) = fruit_box else {
        return;
    };

    do_things_1(apple_color);
    do_things_2();
    do_things_3();
    do_things_4();
}
```

# 包管理

```
cargo add rand
```

```
[dependencies]
rand = "0.8.5"
```

```
use rand::Rng;

fn main() {
    let gen = rand::thread_rng().gen::<i64>() % 2;
    if gen == 0 {
        println!("Hello");
    } else {
        println!("World");
    }
}
```

# 模块

```
$ cd src/
$ touch apple.rs
$ touch pear.rs

# directory layout
src
├── main.rs
├── apple.rs
└── pear.rs
```

```rust
// in main.rs
mod apple;
mod pear;
use rand::Rng;

fn main() {
    let gen = rand::thread_rng().gen::<i64>() % 2;
    if gen == 0 {
        apple::eat_apple();
    } else {
        pear::eat_pear();
    }
}

// in apple.rs
pub fn eat_apple() {
    println!("I eat apple");
}

// in pear.rs
pub fn eat_pear() {
    println!("I eat pear");
}
```

# 模块

```
$ cd src/
$ mkdir orange
$ touch orange/mod.rs
$ touch orange/eat.rs

# directory layout
src
├── main.rs
└── orange
    └── mod.rs
    └── eat.rs
```

```rust
// in orange/mod.rs
pub mod eat;

// in orange/eat.rs
pub fn eat_orange() {
    println!("I eat orange");
}

// in main.rs
mod apple;
mod orange;
mod pear;
use rand::Rng;

fn main() {
    let gen = rand::thread_rng().gen::<i64>() % 2;
    if gen == 0 {
        apple::eat_apple();
    } else {
        pear::eat_pear();
    }

    orange::eat::eat_orange();
}
```

# 文件内模块

```rust
mod fruit {
    mod apple {
        pub fn eat_apple() {
            println!("I eat apple");
        }
    }

    pub(crate) mod pear {
        pub fn eat_pear() {
            println!("I eat pear");
        }
    }

    mod orange {
        pub fn eat_orange() {
            println!("I eat orange");
        }
    }
}
```

# 容器：Vec

```rust
fn init_vec() {
    let _: Vec<i32> = Vec::new();
    let _: Vec<i32> = Vec::with_capacity(16);
    let _ = vec!["hello", "world"];
}

fn this_also_works() {
    let mut v = Vec::new(); // 这里可以先不指定类型
    v.push("str");          // 在这里，编译器会通过元素的类型确定 v 的类型，为 Vec<&str>
}

fn this_also_works2() {
    let _ = Vec::<String>::new();
}
```

# 容器：Vec

```rust
fn get_by_index() {
    let mut v = Vec::with_capacity(2);
    v.push(1);
    v.push(2);

    println!("{}", v[1]); // 输出 2

    println!("{}", v[2]); // 这里程序会直接 panic 退出
}

fn get_by_index() {
    let mut v = Vec::with_capacity(2);
    v.push(1);
    v.push(2);

    println!("{:?}", v.get(1)); // 输出 Some(2)

    println!("{:?}", v.get(2)); // 输出 None
}

fn modify_by_index() {
    let mut v = Vec::with_capacity(2);
    v.push(1);
    v.push(2);

    v[0] = 100;
    println!("{}", v[0]); // 会输出 100
}
```

# 容器：Vec

```rust
fn will_consume_v() {
    let v = vec!["Hello", "World"];
    for elem in v {
        println!("{elem}");
    }

    println!("{:?}", v); // 这行将会报错，因为我们在第三行的 for 已经将 v 给消耗掉了
}

fn willnot_consume_v() {
    let v = vec!["Hello", "World"];
    for elem in &v {
        println!("{elem}");
    }

    println!("{:?}", v); // 这行会正常输出
}

fn modify_elem() {
    let mut v = vec!["Hello", "World"];
    for elem in &mut v {
        *elem = "elem"
    }

    println!("{:?}", v); // 这行会输出 ["elem", "elem"]
}
```

# Option

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

```c
// C
struct User {
    long long int id;
    char *name;
};

// Go
type User struct {
    id int64
    name string // or *string
}

// Rust
struct User {
    id: i64,
    name: Option<String>,
}
```

# Result

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

```rust
#[derive(Debug)]
enum Version { Version1, Version2 }

fn parse_version(header: &[u8]) -> Result<Version, &'static str> {
    match header.get(0) {
        None => Err("invalid header length"),
        Some(&1) => Ok(Version::Version1),
        Some(&2) => Ok(Version::Version2),
        Some(_) => Err("invalid version"),
    }
}

let version = parse_version(&[1, 2, 3, 4]);
match version {
    Ok(v) => println!("working with version: {v:?}"),
    Err(e) => println!("error parsing header: {e:?}"),
}
```

# 错误传播

```rust
enum MyError {
    IOError(io::Error),
    RequestError(reqwest::Error),
    OtherError,
}

impl From<io::Error> for MyError {
    fn from(err: io::Error) -> MyError {
        MyError::IOError(err)
    }
}

impl From<reqwest::Error> for MyError {
    fn from(err: reqwest::Error) -> MyError {
        MyError::RequestError(err)
    }
}

async fn foo(url: &str) -> Result<(), MyError>{
    let text = reqwest::get(url)
        .await?
        .text()
        .await?;

    let mut file = File::create("file.txt")?;
    file.write_all(text.as_bytes())?;

    Ok(())
}
```

# panic 与 unwind

Rust panic 的实现机制有两种：unwind 和 abort。

- unwind: 方式会一层一层退出函数调用栈，正常析构栈上的资源。
- abort: 方式会直接退出程序。

# Result 和 panic!

- 错误在运行时没有机会/能力恢复，用 panic!
- 错误属于开发阶段应该解决而非运行时的错误（比如，数组访问越界、API 用法错误等），用 panic!
- 错误是预期中（可能会出现）的，用 Result

# 04 课后作业

# 练习：开发一个自己的 find 工具

```toml
[package]
name = "myfind"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
regex = "1"

# 推荐的 release 模式配置，性能最好
[profile.release]
opt-level = 3
debug = true
debug-assertions = false
overflow-checks = false
lto = true
panic = 'unwind'
incremental = false
codegen-units = 1
rpath = false
```

# 练习：开发一个自己的 find 工具

```rust
use regex::Regex;
use std::env;
use std::fs;
use std::path::Path;
use std::process;

fn main() {
    let args: Vec<String> = env::args().collect();

    // 参数 1：搜索目录；参数 2：要搜索的正则表达式。
    if args.len() < 3 {
        eprintln!("使用方式：{} <目标目录> <要搜索的正则表达式>", args[0]);
        process::exit(1);
    }
    // 思考一下：如果用户输入的参数太多，应该怎么样？

    let pattern = &args[2];
    let regex = match Regex::new(pattern) {
        Ok(re) => re,
        Err(err) => {
            eprintln!("无效的正则表达式 '{}': {}", pattern, err);
            process::exit(1);
        }
    };

    match find(&args[1], &regex) {
        Ok(matches) => {
            if matches.is_empty() {
                println!("未找到匹配项。");
            } else {
                println!("找到以下匹配项：");
                for file in matches {
                    println!("{}", file);
                }
            }
        }
        Err(error) => {
            eprintln!("发生错误：{}", error);
            process::exit(1);
        }
    }
}
```

```rust
fn find<P: AsRef<Path>>(root: P, regex: &Regex) -> Result<Vec<String>, Box<dyn std::error::Error>> {
    let mut matches = Vec::new();
    walk_tree(root.as_ref(), regex, &mut matches)?;
    Ok(matches)
}

fn walk_tree(
    dir: &Path,
    regex: &Regex,
    matches: &mut Vec<String>,
) -> Result<(), Box<dyn std::error::Error>> {
    // 如果不是，应该怎么办呢？
    if dir.is_dir() {
        for entry in fs::read_dir(dir)? {
            let entry = entry?;
            let path = entry.path();
            if path.is_dir() {
                walk_tree(&path, regex, matches)?;
            } else if let Some(filename) = path.file_name().and_then(|s| s.to_str()) {
                if regex.is_match(filename) {
                    matches.push(path.to_string_lossy().to_string());
                }
            }
        }
    }
    Ok(())
}
```

# 进阶

- 尝试将该代码重构到多个模块中
- 尝试增加新的功能，比如 –v/--verbose 参数输出所有遍历到的文件
- 尝试同时支持匹配多个正则/给输出结果去重排序
- 尝试使用 tracing 库打日志
- 尝试支持同时搜索多个 path
- 尝试支持命令行彩色输出
- 尝试一切你想要的东西，毕竟是你自己的 find

# 好帮手

- Google/StackOverflow
- docs.rs
- chatgpt：不推荐使用，需要自己学习，自己思考。等进阶为高阶工程师了，再用。

# 05 大作业

# 大作业

Mini - Redis：

尝试使用异步 Rust 实现一个简单的 Redis RPC 服务。

进阶：

1. 支持 AOF 机制
2. 支持 leader/follower 主从架构
3. 支持 redis cluster 集群

# THANKS

CloudWeGo