CloudWeGo

Volo - 高性能 Rust RPC 框架

Rust & Volo 101

刘捷（基础架构-服务框架）

CloudWeGo 开源团队出品

2022/10/13

个人介绍

刘捷

- 毕业于南京大学
- CloudWeGo-Volo Maintainer
- Rust 语言爱好者
- 人生哲理是多倾听自己的声音

目录

CONTENT

PREFACE

Mark Russinovich ✓
@markrussinovich

Speaking of languages, it's time to halt starting any new projects in C/C++ and use Rust for those scenarios where a non-GC language is required. For the sake of security and reliability. the industry should declare those languages as deprecated.

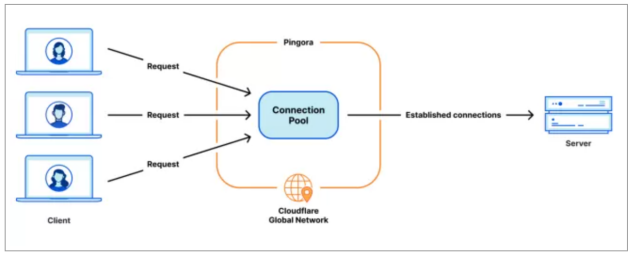
Cloudflare Ditches Nginx For In-House, Rust-Written Pingora

Written by Michael Larabel in Linux Networking on 15 September 2022 at 02:56 PM EDT. 99 Comments

Cloudflare has long relied upon Nginx as part of its HTTP proxy stack but now has replaced it with their in-house, Rust-written Pingora software that is said to be serving over one trillion requests per day and delivering better performance while only using about a third of the CPU and memory resources.

Cloudflare has "outgrown" Nginx and ended up creating their own HTTP proxy stack. Cloudflare found that Nginx's worker process architecture was hitting drawbacks, particularly around CPU resources. Nginx also proved to be difficult to extend to their needs.

Cloudflare engineers have been developing Pingora from scratch as an in-house solution. The Rust programming language was chosen for its memory safety while still delivering C-like performance. Cloudflare also implemented their own HTTP library for Rust in order to suit all of their different needs. Pingora employs a multi-threaded architecture rather than multi-process.



Cloudflare Pingora diagram.

As for the performance benefits with Pingora:

> *In production, Pingora consumes about 70% less CPU and 67% less memory compared to our old service with the same traffic load.*

Elon Musk ✓
@elonmusk

Replying to @jack and @gdb

I'm a fan of Rust. Clearly scales well, given that Discord uses it.

For max performance, however, nothing beats tight C with a customized compiler on specialized hardware. Important for max frame rate on vehicle inference computer.

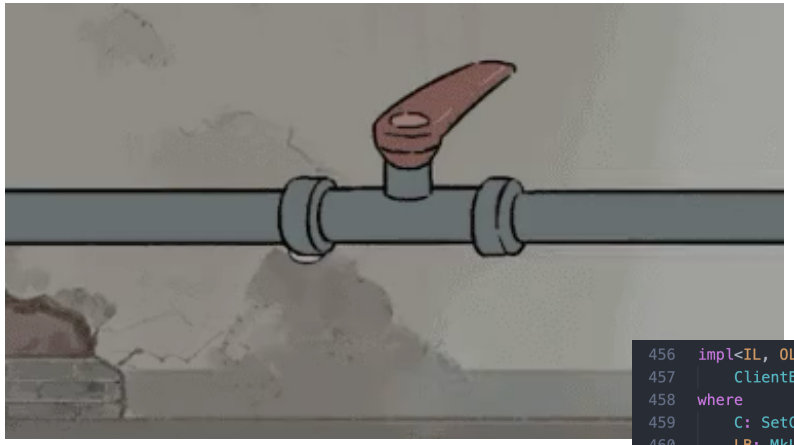
WHY GENERAL CATALYST INVESTED IN NEON

July 27, 2022 • BY QUENTIN CLARK

NEON

General Catalyst is excited to be working with Nikita Shamgunov, Heikki Linnakangas, Stas Kelvich, and the whole team at Neon. Once every so many years, foundational technology changes create opportunity in the database industry to build a transformational product and important company. Neon is doing this by building on the cloud's separation of storage and compute power, and applying it to OLTP databases to offer a serverless Postgres that we believe has the potential to scale and perform like nothing before it.

PREFACE



编译器斗争

```rust
impl<IL, OL, C, Req, Resp, MkE: MkEncoder + 'static, MkD: MkDecoder + 'static,
    ClientBuilder<IL, OL, C, Req, Resp, MkE, MkD, LB>
where
    C: SetClient<Req, Resp>,
    LB: MkLbLayer<IL::Service>,
    LB::Layer: Layer<IL::Service>,
    <LB::Layer as Layer<IL::Service>>::Service:
        Service<ClientContext, Req, Response = Option<Resp>> + 'static + Send
    <<LB::Layer as Layer<IL::Service>>::Service as Service<ClientContext, Req>
        Into<crate::Error>,
    Req: EntryMessage + Send + 'static + Size + Sync + Clone,
    Resp: EntryMessage + Send + 'static,
    IL: Layer<MessageService<Resp, MkE, MkD>>,
    IL::Service:
        Service<ClientContext, Req, Response = Option<Resp>> + Sync + Clone + Send + 'static,
    <IL::Service as Service<ClientContext, Req>>::Error: Send + Into<crate::Error>,
    OL: Layer<
        BoxCloneService<
            ClientContext,
            Req,
            Option<Resp>,
            <LB::Layer as Layer<IL::Service>>::Service as Service<ClientContext, Req>>::Error,
        >,
    >,
    OL::Service: Service<ClientContext, Req, Response = Option<Resp>> + 'static + Send + Clone,
    <OL::Service as Service<ClientContext, Req>>::Error: Send + Sync + Into<crate::Error>,
{
```

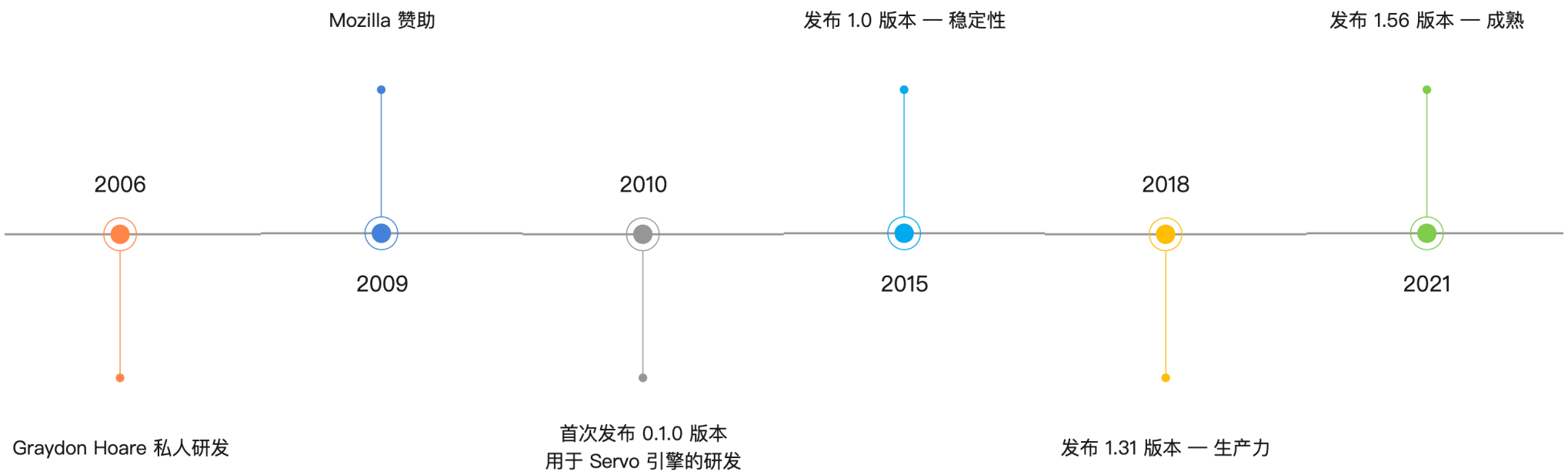
泛型

```rust
#[macro_export]
macro_rules! newtype_index {
    // ---- public rules ----

    // Use default constants
    ($(#[$attrs:meta])* $v:vis struct $name:ident { .. }) => (
        $crate::newtype_index!(
            // Leave out derives marker so we can use its absence to ensure it comes first
            @attrs        [$(#[$attrs])*]
            @type         [$name]
            // shave off 256 indices at the end to allow space for packing these indices into enums
            @max          [0xFFFF_FF00]
            @vis          [$v]
            @debug_format ["{}"]);
    );

    // Define any constants
    ($(#[$attrs:meta])* $v:vis struct $name:ident { $($tokens:tt)+ }) => (
        $crate::newtype_index!(
            // Leave out derives marker so we can use its absence to ensure it comes first
            @attrs        [$(#[$attrs])*]
            @type         [$name]
            // shave off 256 indices at the end to allow space for packing these indices into enums
            @max          [0xFFFF_FF00]
            @vis          [$v]
            @debug_format ["{}"]
                          $($tokens)+);
    );

    // ---- private rules ----

    // Base case, user-defined constants (if any) have already been defined
    (@derives      [$($derives:ident,)*]
     @attrs        [$(#[$attrs:meta])*]
     @type         [$type:ident]
     @max          [$max:expr]
     @vis          [$v:vis]
     @debug_format [$debug_format:tt]) => (
```

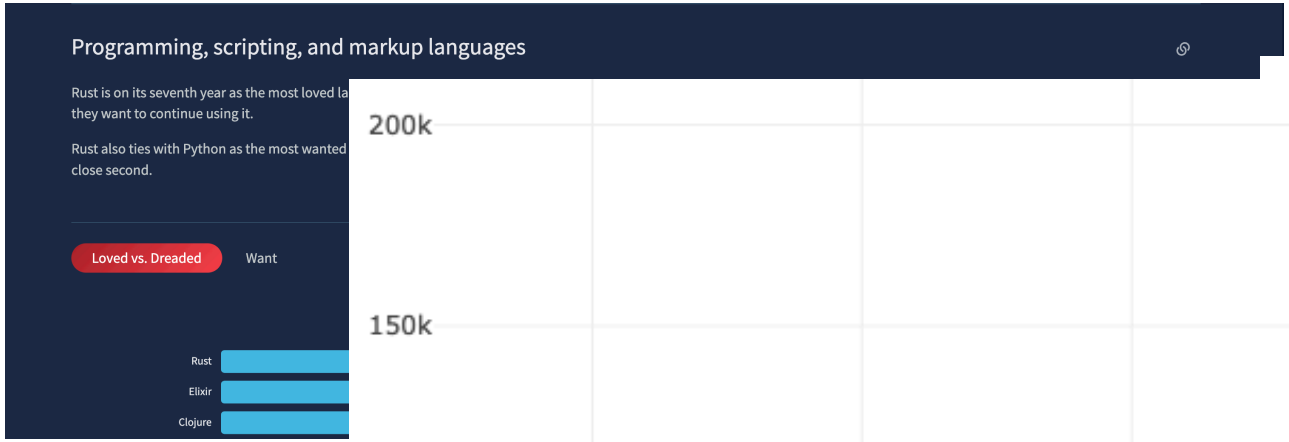
宏编程

01 Why Rust

Rust 发展


2006
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
发布 1.56 版本 — 成熟

Rust in 2022



Platinum

aws

r/rust subreddit subscribers

.0: Fast, Safe and

ime is now at 1.0! This means that all
lly ready to use in production.

uction-ready more than a year ago. But
ly engine. We wanted to have a super
wanted to feel really confident when
.

you, a number of us in the Bytecode
uction ourselves for the past year. And
uction environments, providing a stable
d wins.

Assembly

Rust 2024...the year of everywhere?

Sep 22, 2022

I've been thinking about what "Rust 2024" will look like lately. I don't really mean the edition itself — but more like, what will Rust feel like after we've finished up the next few years of work? I think the answer is that Rust 2024 is going to be the year of "everywhere". Let me explain what I mean. Up until now, Rust has had a lot of nice features, but they only work *sometimes*. By the time 2024 rolls around, they're going to work *everywhere* that you want to use them, and I think that's going to make a big difference in how Rust feels.

Why Rust

高性能

1. 基于 RAII 的内存管理

—— 没有垃圾回收

2. 严格的约束

—— 编译器极致优化

3. 零成本抽象

—— 没有运行时

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%

Benchmark Game: regex-redux

可靠性-内存安全

1. 所有权

—— 二次释放问题

```
error[E0382]: borrow of moved value: `s1`
  --> src/bin/client.rs:27:27
   |
24 |     let s1 = String::from("hello");
   |         -- move occurs because `s1` has type `String`, which does not implement the `Copy` trait
25 |     let s2 = s1;
   |              -- value moved here
26 |
27 |     println!("{}, world", s1);
   |                           ^^ value borrowed here after move
   |
```

2. 生命周期

—— 悬垂指针问题

```
error[E0597]: `x` does not live long enough
  --> src/bin/client.rs:27:13
   |
27 |         r = &x;
   |             ^^ borrowed value does not live long enough
28 |     }
   |     - `x` dropped here while still borrowed
29 |     println!("r: {}", r);
   |                        - borrow later used here
```

可靠性-并发安全

Link: The Rustonomicon

Send：实现 Send 的类型可以在多线程间转移所有权

Sync：实现 Sync 的类型可以在多线程间共享引用

推论：U 实现 Sync 当且仅当 &U 实现 Send

```rust
pub unsafe auto trait Send {}

pub unsafe auto trait Sync {}
```

```rust
pub fn spawn<F, T>(f: F) -> JoinHandle<T>
where
    F: FnOnce() -> T,
    F: Send + 'static,
    T: Send + 'static,
```

生产力

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

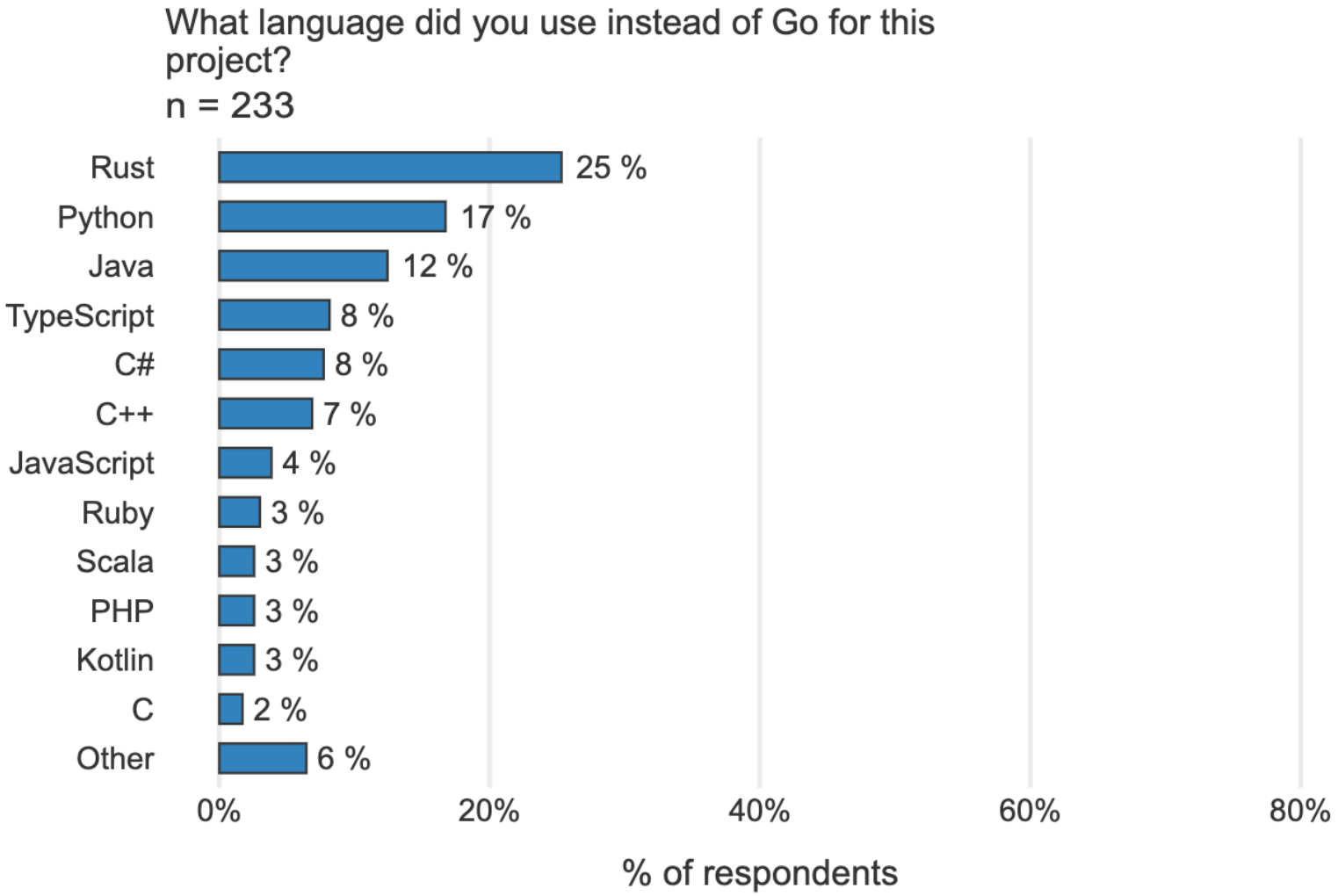
信任别人的代码

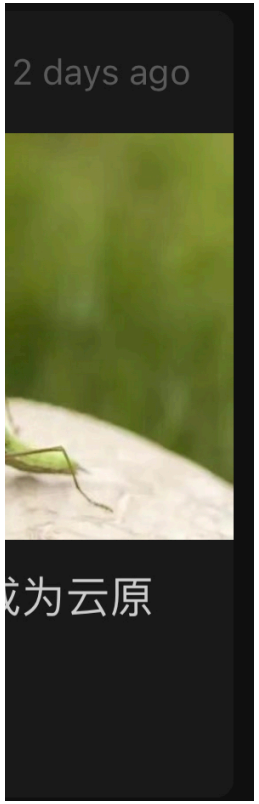
BENEFITS

Go ❤️ Rust

Language C

Language
Creation Date
Created at
Notable software written in language
Key workloads
Developer adoption
Most loved
Most wanted





Rust 安装

环境变量：

RUSTUP_DIST_SERVER="https://rsproxy.cn"

RUSTUP_UPDATE_ROOT="https://rsproxy.cn/rustup"

安装：

Linux/MacOS： curl --proto '=https' --tlsv1.2 -sSf https://rsproxy.cn/rustup-init.sh | sh

Windows： https://static.rust-lang.org/rustup/dist/i686-pc-windows-gnu/rustup-init.exe

验证：

rustc —version

可以看到按照以下格式显示的最新稳定版本的版本号、对应的 Commit Hash 和 Commit 日期：

rustc x.y.z (abcabcabc yyyy-mm-dd)

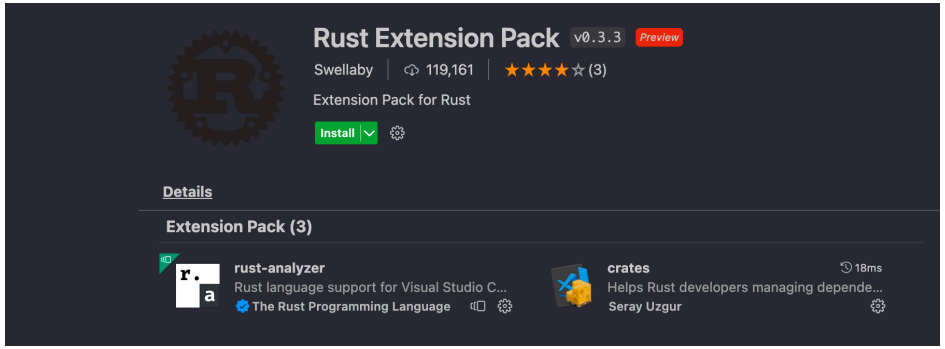
环境配置

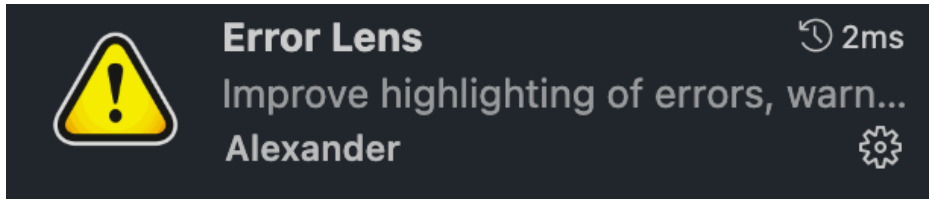
配置文件：~/.cargo/config

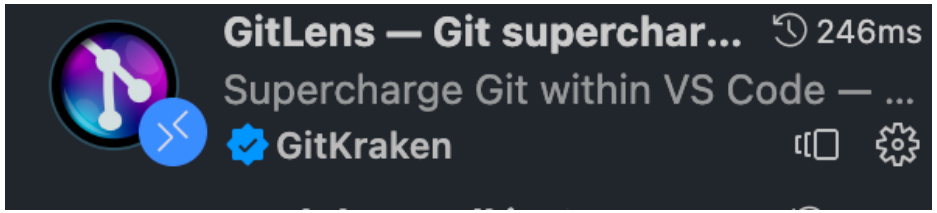
```
[source.crates-io]
replace-with = 'rsproxy'

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

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

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

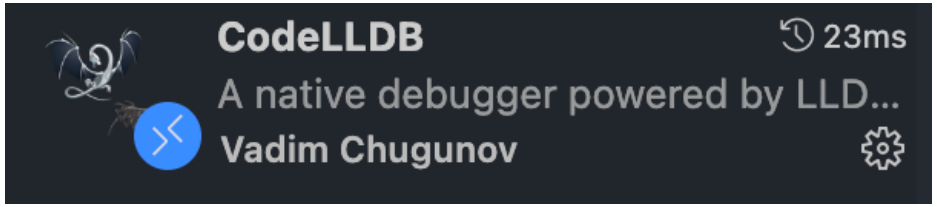
IDE-VS Code

扩展

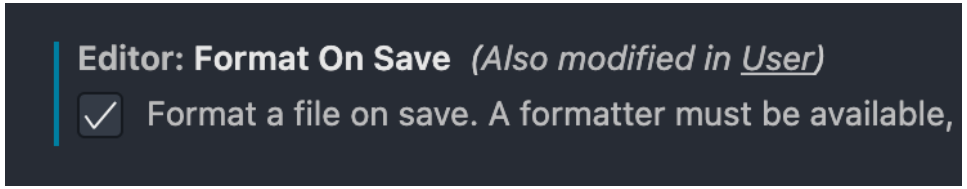


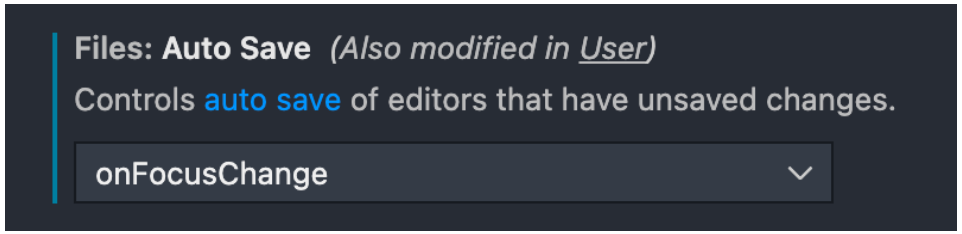


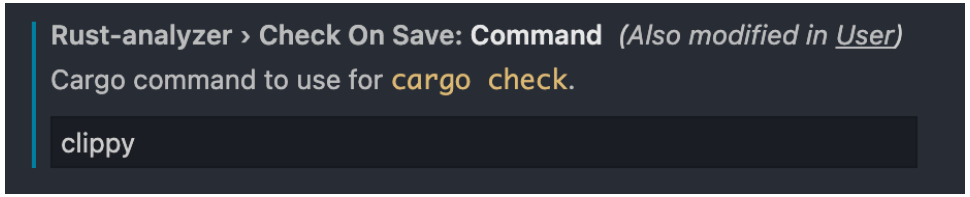


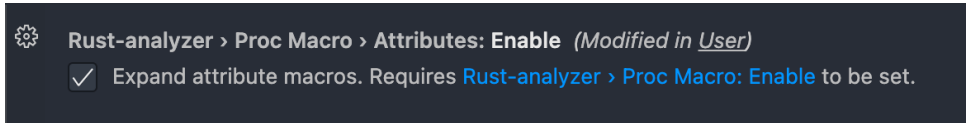


设置



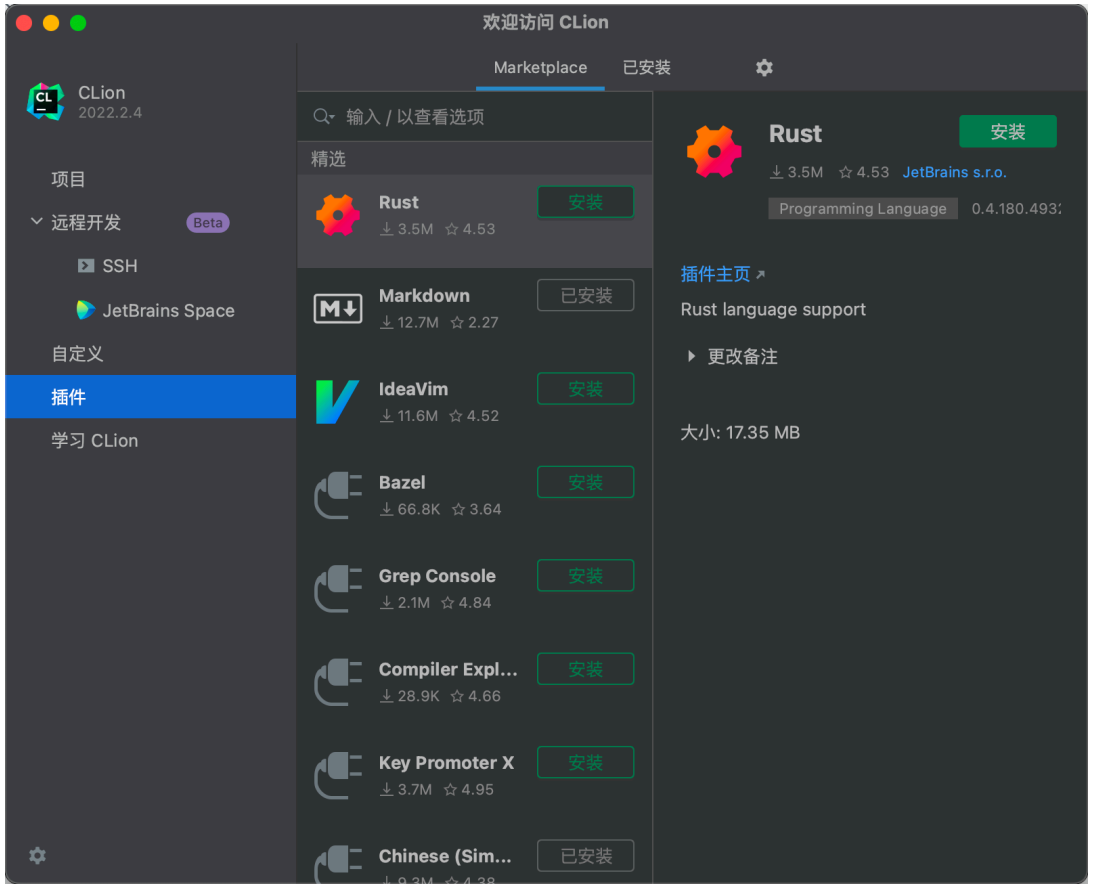




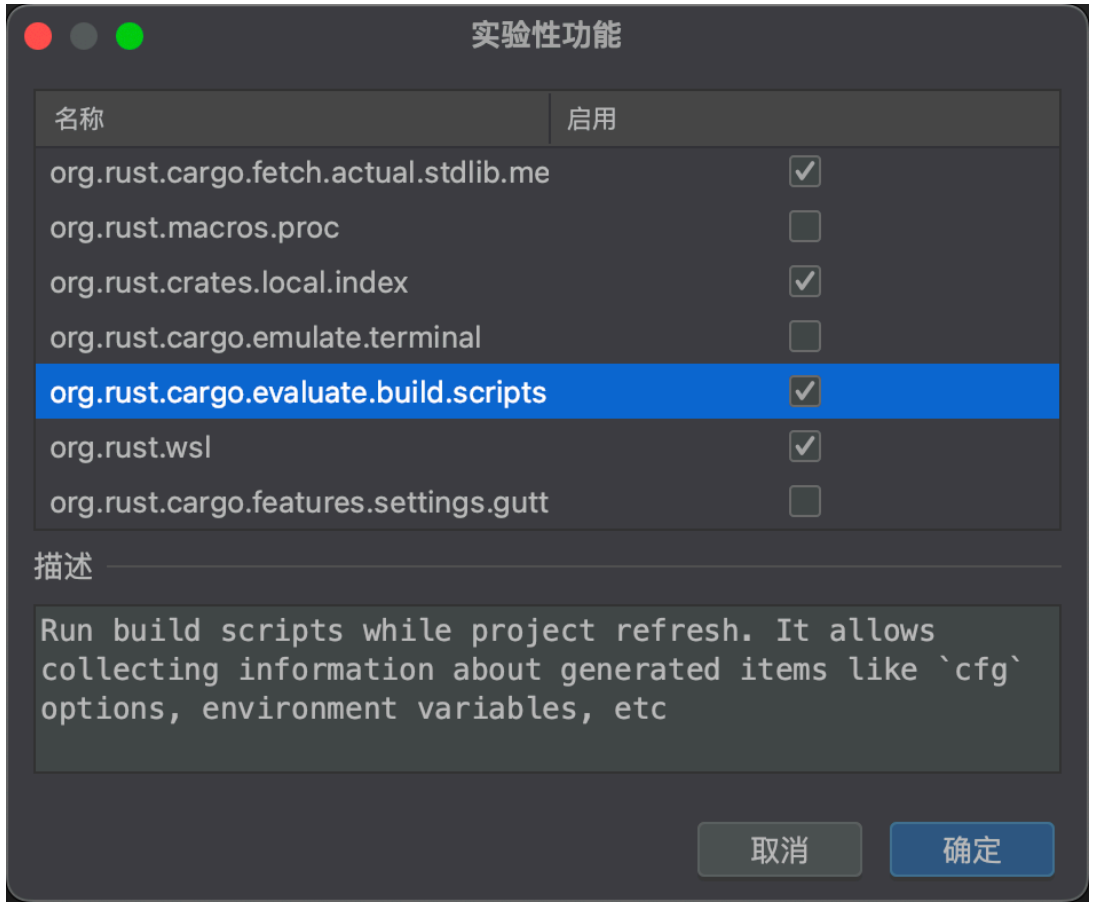


IDE-Clion

安装 Rust 插件



连按两次 Shift 搜索 Experimental
打开 Experimental Features

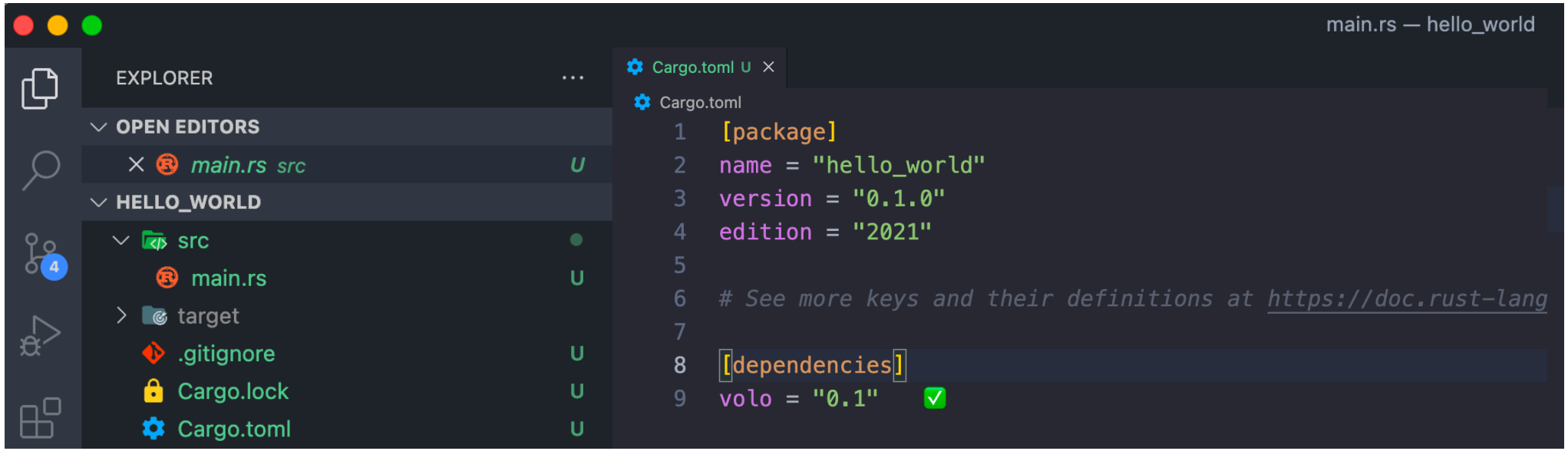


Hello World

创建新项目：cargo new hello_world

使用VS Code打开：code hello_world

跑起来：cargo run



学习资源

- The Book：https://doc.rust-lang.org/book/
- Rust By Example：https://doc.rust-lang.org/rust-by-example/
- Rustlings：https://github.com/rust-lang/rustlings
- Async Rust：https://rust-lang.github.io/async-book/
- 汇总：https://github.com/ctjhoa/rust-learning

02 Why Volo

生态现状

- **Thrift：**没有生产可用的 Async Thrift 实现
- **gRPC：**Tonic 实现服务治理功能偏弱，易用性不够强
- **抽象：**没有基于 GAT 和 TAIT 的设计

GAT & TAIT

Generic Associated Types

```
1 trait Foo {
2   type Bar<T>;
3
4   type Baz<'a>;
5 }
```

Type Alias Impl Trait

```
1 type Qux = impl Foo;
2
3 struct __Foo_alias;
4 type Qux = __Foo_alias;
```

一个 Timeout 中间件

- 代码量减少一倍多
- 逻辑更加清晰简单

```rust
#[derive(Debug, Clone)]
pub struct Timeout<T> {
    inner: T,
    timeout: Duration,
}

impl<S, Request> Service<Request> for Timeout<S>
where
    S: Service<Request>,
    S::Error: Into<crate::BoxError>,
{
    type Response = S::Response;
    type Error = crate::BoxError;
    type Future = ResponseFuture<S::Future>;

    fn poll_ready(&mut self, cx: &mut Context<'_>) -> Poll<Result<(), Self::Error>>
{
        match self.inner.poll_ready(cx) {
            Poll::Pending => Poll::Pending,
            Poll::Ready(r) => Poll::Ready(r.map_err(Into::into)),
        }
    }

    fn call(&mut self, request: Request) -> Self::Future {
        let response = self.inner.call(request);
        let sleep = tokio::time::sleep(self.timeout);

        ResponseFuture::new(response, sleep)
    }
}

pin_project! {
    #[derive(Debug)]
    pub struct ResponseFuture<T> {
        #[pin]
        response: T,
        #[pin]
        sleep: Sleep,
    }
}

impl<T> ResponseFuture<T> {
    pub(crate) fn new(response: T, sleep: Sleep) -> Self {
        ResponseFuture { response, sleep }
    }
}

impl<F, T, E> Future for ResponseFuture<F>
where
    F: Future<Output = Result<T, E>>,
    E: Into<crate::BoxError>,
{
    type Output = Result<T, crate::BoxError>;

    fn poll(self: Pin<&mut Self>, cx: &mut Context<'_>) -> Poll<Self::Output> {
        let this = self.project();

        match this.response.poll(cx) {
            Poll::Ready(v) => return Poll::Ready(v.map_err(Into::into)),
            Poll::Pending => {}
        }

        match this.sleep.poll(cx) {
            Poll::Pending => Poll::Pending,
            Poll::Ready(_) => Poll::Ready(Err(Elapsed(()).into())),
        }
    }
}
```

```rust
#[derive(Clone)]
pub struct Timeout<S> {
    inner: S,
    duration: Duration,
}

#[service]
impl<Cx, Req, S> Service<Cx, Req> for Timeout<S>
where
    Req: 'static + Send,
    S: Service<Cx, Req> + 'static + Send,
    Cx: 'static + Send,
    S::Error: Send + Sync + Into<BoxError>,
{
    async fn call(&mut self, cx: &mut Cx, req: Req) -> Result<S::Response, BoxError> {
        let sleep = tokio::time::sleep(self.duration);
        tokio::select! {
            r = self.inner.call(cx, req) => {
                r.map_err(Into::into)
            },
            _ = sleep => Err(std::io::Error::new(std::io::ErrorKind::TimedOut, "service time
out").into()),
        }
    }
}
```

Why Volo

易用性

- 基于 GAT 和 TAIT，降低编写中间件 Service 难度
- 提供了命令行工具生成项目默认 Layout
- 提供了 IDL管理能力
- ……

扩展性

- 基于 Service 的抽象

```rust
pub trait Service<Cx, Request> {
    /// Responses given by the service.
    type Response;
    /// Errors produced by the service.
    type Error;

    /// The future response value.
    type Future<'cx>: Future<Output = Result<Self::Response, Self::Error>> + Send +
'cx where
        Cx: 'cx,
        Self: 'cx;

    /// Process the request and return the response asynchronously.
    fn call<'cx, 's>(&'s mut self, cx: &'cx mut Cx, req: Request) -> Self::Future<'cx>
    where
        's: 'cx;
}
```

- 基于 RPC 元信息的控制

```rust
for (addr, _) in picker.zip(0..self.retry + 1) {
    call_count += 1;
    if let Some(callee) = cx.rpc_info_mut().callee_mut() {
        callee.address = Some(addr.clone())
    }

    match self.service.call(cx, req.clone()).await {
        Ok(resp) => {
            return Ok(resp);
        }
        Err(err) => {
            tracing::warn!("[VOLO] call endpoint: {:?} error: {:?}", addr,
err);    }
    }
}
```

高性能

- （非公平对比）在和 Kitex 相同的测试条件（限制 4C）下，Volo 极限 QPS 为 35W

- 同时，我们内部正在验证基于 Monoio（CloudWeGo 开源的 Rust Async Runtime）的版本，极限 QPS 可以达到 44W

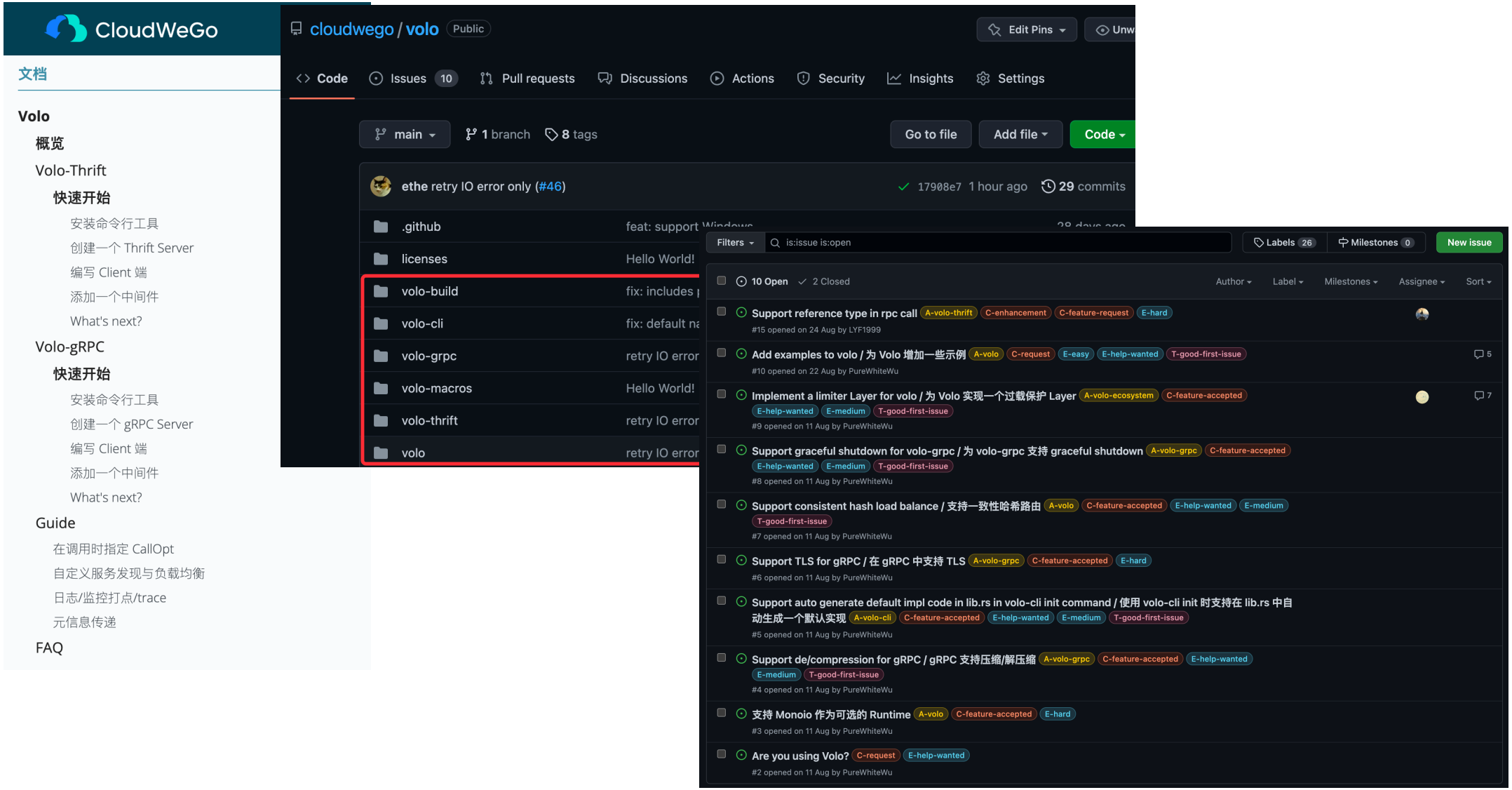
实际业务收益

- 业务 A（Proxy 类）：
  - CPU：-39.68%
  - MEM：-77.78%
  - P99：-76.67%
  - AVG：-70%
- 业务 B（有大量业务逻辑）：
  - CPU：-43.75%
  - MEM：-69%
  - P99：-43.22%
  - AVG：-47.81

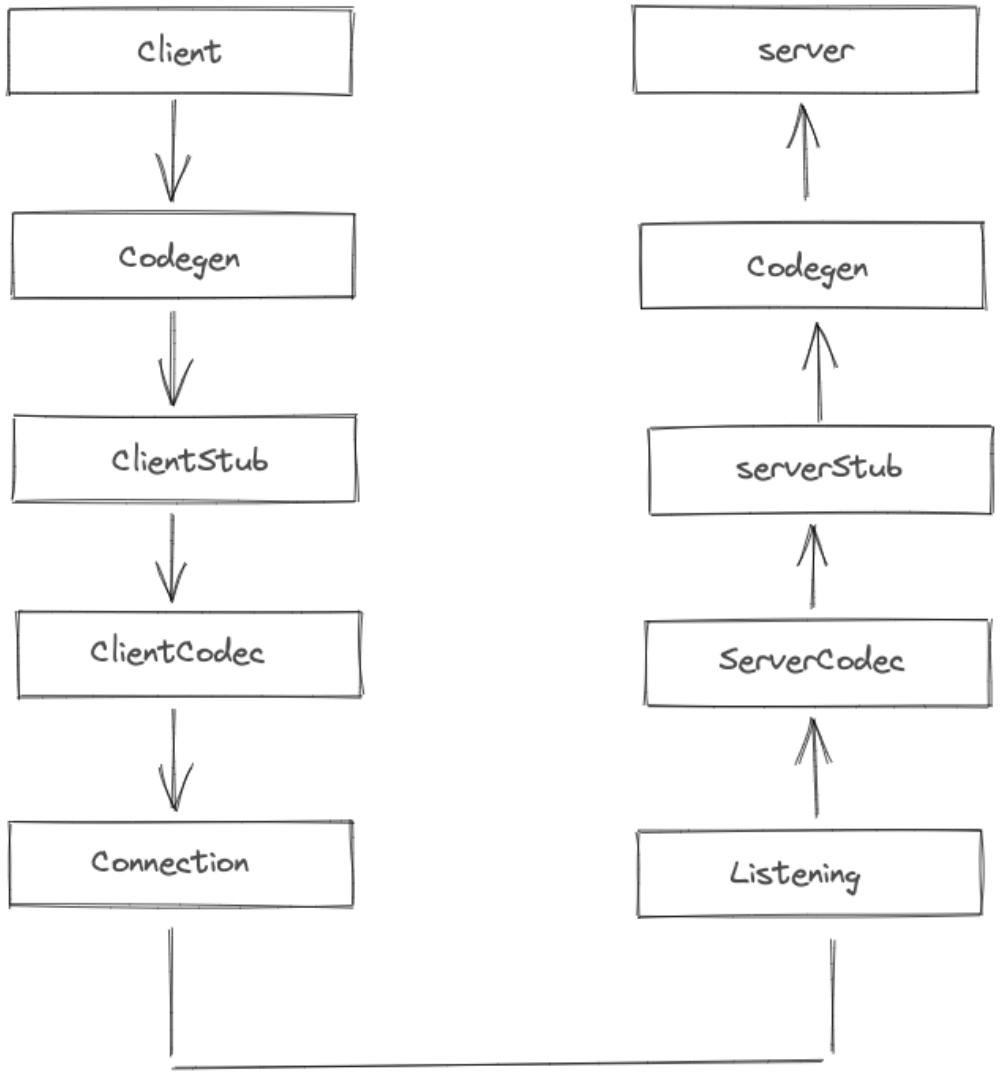
相关生态

- **Volo：** RPC 框架的名字，包含了 Volo-Thrift 和 Volo-gRPC 两部分
- **Volo-rs 组织：**Volo 的相关生态
- **Pilota：**Volo 使用的 Thrift 和 Protobuf 编译器及编解码的纯 Rust 实现
- **Motore：**Volo 参考 Tower 设计的，使用了 GATs 和 TAIT 的中间件抽象层
- **Metainfo：**Volo 用于进行元信息透传的组件，定义了一套元信息透传的标准

Volo


CloudWeGo
文档
Volo
概览
Volo-Thrift
快速开始
安装命令行工具
创建一个 Thrift Server
编写 Client 端
添加一个中间件
What's next?
Volo-gRPC
快速开始
安装命令行工具
创建一个 gRPC Server
编写 Client 端
添加一个中间件
What's next?
Guide
在调用时指定 CallOpt
自定义服务发现与负载均衡
日志/监控打点/trace
元信息传递
FAQ
cloudwego / volo
Public
Edit Pins
Code
Issues 10
Pull requests
Discussions
Actions
Security
Insights
Settings
main
1 branch
8 tags
Go to file
Add file
Code
ethe retry IO error only (#46)
17908e7 1 hour ago
29 commits
.github
licenses
volo-build
volo-cli
volo-grpc
volo-macros
volo-thrift
volo
Hello World!
retry IO error
Filters
is:issue is:open
Labels 26
Milestones 0
New issue
10 Open
2 Closed
Author
Label
Milestones
Assignee
Sort
Support reference type in rpc call
#15 opened on 24 Aug by LYF1999
Add examples to volo / 为 Volo 增加一些示例
#10 opened on 22 Aug by PureWhiteWu
Implement a limiter Layer for volo / 为 Volo 实现一个过载保护 Layer
#9 opened on 11 Aug by PureWhiteWu
Support graceful shutdown for volo-grpc / 为 volo-grpc 支持 graceful shutdown
#8 opened on 11 Aug by PureWhiteWu
Support consistent hash load balance / 支持一致性哈希路由
#7 opened on 11 Aug by PureWhiteWu
Support TLS for gRPC / 在 gRPC 中支持 TLS
#6 opened on 11 Aug by PureWhiteWu
Support auto generate default impl code in lib.rs in volo-cli init command / 使用 volo-cli init 时支持在 lib.rs 中自动生成一个默认实现
#5 opened on 11 Aug by PureWhiteWu
Support de/compression for gRPC / gRPC 支持压缩/解压缩
#4 opened on 11 Aug by PureWhiteWu
支持 Monoio 作为可选的 Runtime
#3 opened on 11 Aug by PureWhiteWu
Are you using Volo?
#2 opened on 11 Aug by PureWhiteWu

Demo


Client
Codegen
ClientStub
ClientCodec
Connection
server
Codegen
serverStub
ServerCodec
Listening

THANKS

欢迎关注 CloudWeGo

Q&A

CloudWeGo