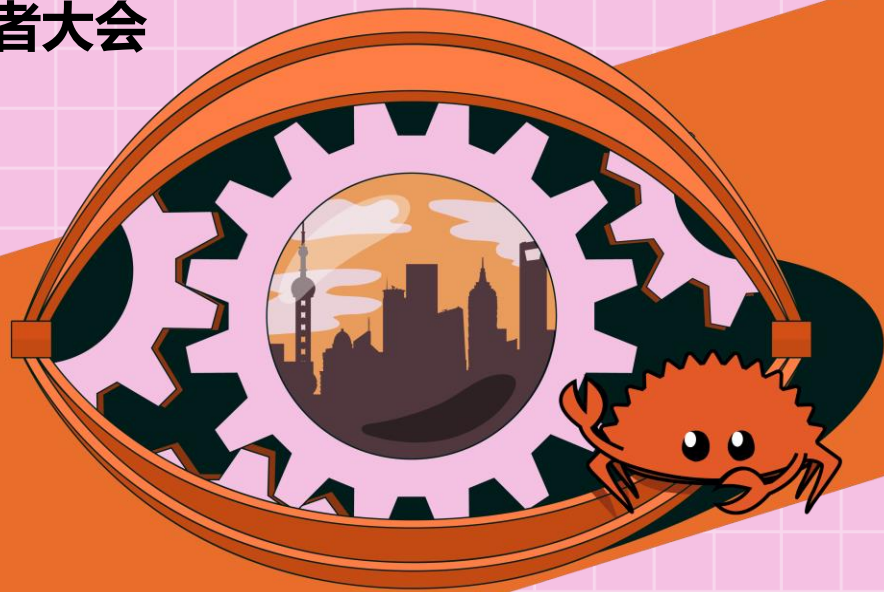


# RUST CHINA CONF 2023

第三届中国Rust开发者大会



6.17-6.18 @Shanghai

# Rust HTTP 协议栈在终端通信场景的实践

胡凯

[hukai45@huawei.com](mailto:hukai45@huawei.com)

华为 公共开发部 嵌入式软件能力中心



# 目录

## ● HTTP 协议介绍

什么是 HTTP 协议？

## ● Rust 与 HTTP 协议

介绍 Rust 与 HTTP 协议栈结合的业界实现。

## ● 终端 HTTP 通信场景浅析

终端场景下 HTTP 协议的主要使用场景，以及需要思考的问题。

## ● Rust 与终端 HTTP 通信场景结合

我们当前结合 Rust 和终端通信场景的实践的简单介绍。



## Part 01

# HTTP 协议介绍

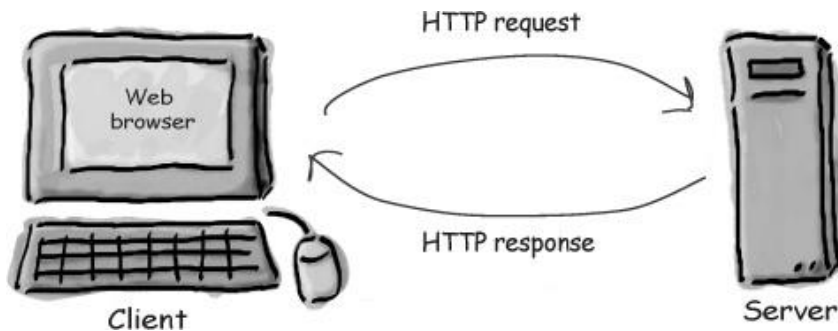
什么是 HTTP 协议?



## HTTP 协议介绍

HTTP 协议，即超文本传输协议（HyperText Transfer Protocol）是一种用于分布式、协作式和超媒体信息系统的应用层协议。

HTTP 是一个客户端（用户）和服务端（网站）之间请求和应答的标准。



## HTTP 协议介绍

HTTP 协议主要具有以下特点：

- ✓ 支持客户/服务器模式。
- ✓ 简单快速：客户向服务器请求服务时，只需传送请求方法、路径和请求头。HTTP 协议简单、HTTP 服务器的程序规模小，因而通信速度很快。
- ✓ 灵活：HTTP 支持传输任意类型的数据对象。
- ✓ 无连接：HTTP 限制每次连接只处理一个请求，节省传输时间。(在 HTTP/1.1 之后变更)
- ✓ 无状态：HTTP 协议对于事务处理没有记忆能力，每个请求/应答之间相互独立。



# HTTP 协议介绍

HTTP 协议的版本演化如下:

## HTTP/0.9

- 早期的 HTTP 协议
- 请求方法仅能使用 GET
- 响应仅含有文档内容, 且仅支持 html

GET	Content
request	response

## HTTP/1.0

- 新的请求方法
- 发送请求时附带版本信息
- 支持标头字段
- 支持请求上传内容
- 支持传输多种格式的内容。

Method	Status
Req-Line	Sta-Line
Headers	Headers
Content	Content
request	response

## HTTP/1.1

- 连接可以复用
- 管线化技术
- 支持响应分块
- 引入额外的缓存控制机制
- 引入内容协商机制

- 报文基本格式不再变化

## HTTP/2

- 二进制协议
- 支持多路复用
- 支持响应分块
- 支持标头压缩
- 支持服务端推送

- 报文基本格式不再变化

## HTTP/3

- 基于 UDP 连接

- 报文基本格式不再变化

## HTTP 协议介绍

HTTPS 协议 = HTTP 协议 + TLS

超文本传输安全协议（英语：HyperText Transfer Protocol Secure，缩写：HTTPS）是一种通过计算机网络进行安全通信的传输协议。HTTPS经由HTTP进行通信，但利用SSL/TLS来加密数据包。





## Part 02

# Rust 与 HTTP 协议

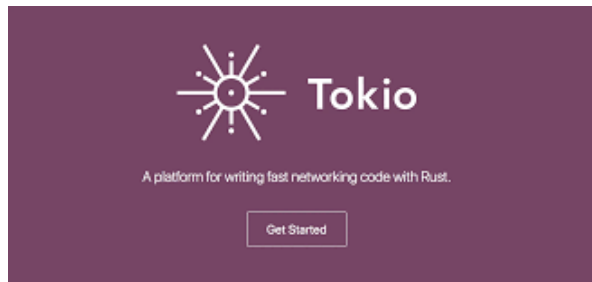
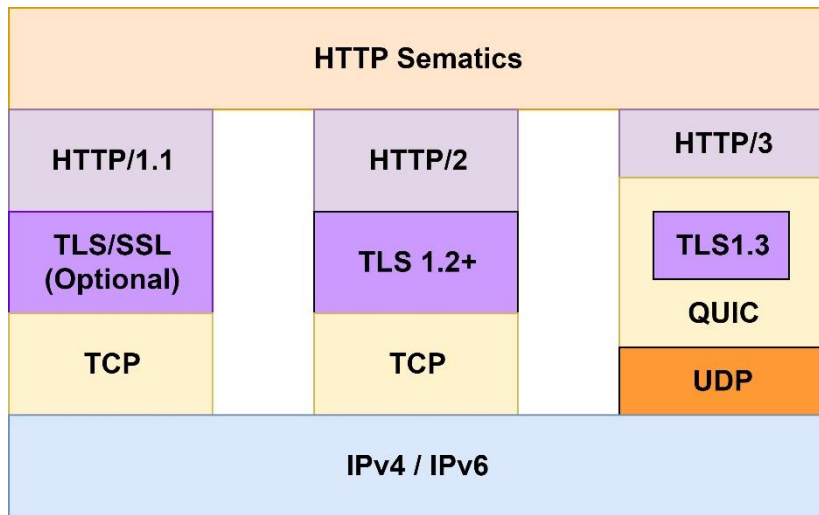
介绍 Rust 与 HTTP 协议栈结合的业界实现



## Rust 与 HTTP 协议

Rust 对于 HTTP 协议有良好支持：


HTTP 协议是以 TCP\TLS\UDP 等各种连接为基础的，非常依赖于高性能的 IO 操作。  
利用 Rust 异步实现 HTTP 协议和各种应用程序能得到十分可观的性能提升，并且能降低用户编码的难度。



# Rust 与 HTTP 协议

借助于 Rust 异步能力的热门 Rust HTTP 协议库或应用库：

## Hyper




### hyperium

217 followers <https://hyper.rs> [@hyper\\_rs@fosstodon.org](#) Verified

[Overview](#) [Repositories 13](#) [Projects 2](#) [Packages](#) [People 5](#)

Pinned




#### hyper

Public

An HTTP library for Rust

Rust 12k 1.4k




#### http

Public

Rust HTTP types

Rust 944 253




#### h2

Public

HTTP 2.0 client & server implementation for Rust.

Rust 1.2k 215

## request




### request

Public

An easy and powerful Rust HTTP Client

Rust 7.6k 832

## actix-web



### actix-web

Public

Actix Web is a powerful, pragmatic, and extremely fast web framework for Rust.

Rust 17.7k 1.5k

## axum

### axum

Public

Ergonomic and modular web framework built with Tokio, Tower, and Hyper

Rust 10,560 709 20 13 Updated 1 hour ago

## Rust 与 HTTP 协议

hyper

crates.io v1.0.0-rc.3 docs passing license MIT CI passing chat 1894 online

A fast and correct HTTP implementation for Rust.

hyper 是一个高效且功能完整的 HTTP 协议底层库，可以支持高层的应用软件使用 HTTP 协议。

- ✓ Rust 异步实现
- ✓ 支持 HTTP/1.1 和 HTTP/2
- ✓ 支持 Client 和 Server
- ✓ 高性能
- ✓ 高可扩展性

## Rust 与 HTTP 协议

### reqwest

crates.io v0.11.18 docs passing license MIT OR Apache-2.0 CI passing

An ergonomic, batteries-included HTTP Client for Rust.

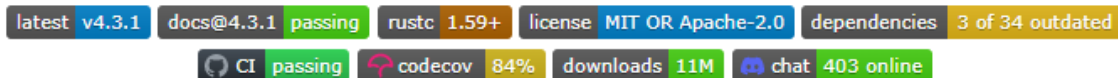
reqwest 是基于 hyper 实现的高性能、易用的 HTTP 客户端库。

- ✓ Rust 异步实现
- ✓ 支持明文、JSON、Multipart 的 body 类型
- ✓ 支持 HTTP Proxy
- ✓ 支持 HTTPS
- ✓ 支持 Cookies

## Rust 与 HTTP 协议

### Actix Web

Actix Web is a powerful, pragmatic, and extremely fast web framework for Rust



actix\_web 是一个强大、实用且速度极快的 Rust 网络框架。

- ✓ Rust 异步实现
- ✓ 支持 HTTP/1 和 HTTP/2
- ✓ 支持 HTTPS
- ✓ 支持消息路由
- ✓ 支持 body 自动解压缩
- ✓ 支持 multipart

## Rust 与 HTTP 协议

以上 Rust HTTP 库主要支持的场景特点:

- 并发量、吞吐量需求较高
- 网络环境稳定
- 不太需要体现交互界面
- 不太关注资源使用

比较适合构建浏览器、大型 WEB 服务器等。



## Part 03

# 终端 HTTP 通信场景浅析

探讨终端场景下 HTTP 协议的主要使用场景，以及需要思考的问题

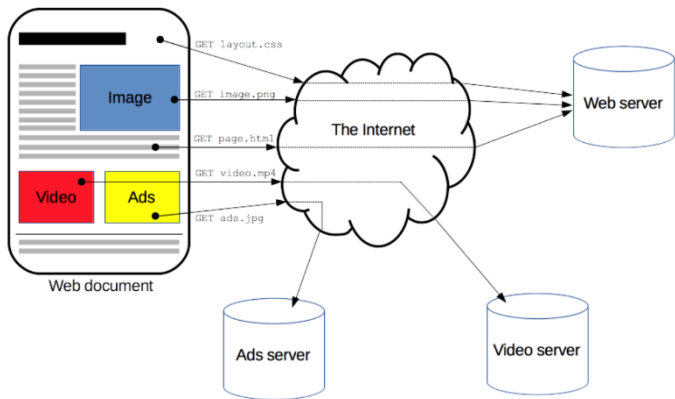




## 终端 HTTP 协议场景浅析

在终端上大多数使用 HTTP 协议的应用，主要是运用 HTTP 客户端的能力，向指定网址发起请求来获取服务器上的资源。

例如使用浏览器 APP 访问网页，使用视频 APP 观看视频和直播，电商 APP 浏览商品页面等。



## ■ 终端 HTTP 协议场景浅析

终端的网络环境特点：

- 弱网环境：移动端网络整体处于弱网环境，网络时延较高。
- 网络不稳定：移动端网络经常受到用户或者环境影响而产生波动。
- 流量限制：移动端网络流量受到用户的限制。
- 设备资源有限：移动端设备CPU、内存等资源较少。

## ■ 终端 HTTP 协议场景浅析

从用户使用的角度出发：

- 下载进度：对于一些涉及上传或下载的应用软件，进度显示能够给用户及时性的反馈。
- 速度和流量限制：受到资费和网络状况的影响，传输速度和流量需要提供给用户设置。
- 暂停和重试：网络传输需要提供给用户控制启动和暂停的控制手段。
- 功耗：网络传输需要消耗终端设备资源，需要尽可能平衡功耗和传输速度。
- 性能表现：网络传输不能影响到和用户直接交互的前台应用的表现。

## Part 04

# Rust 与终端 HTTP 通信场景结合

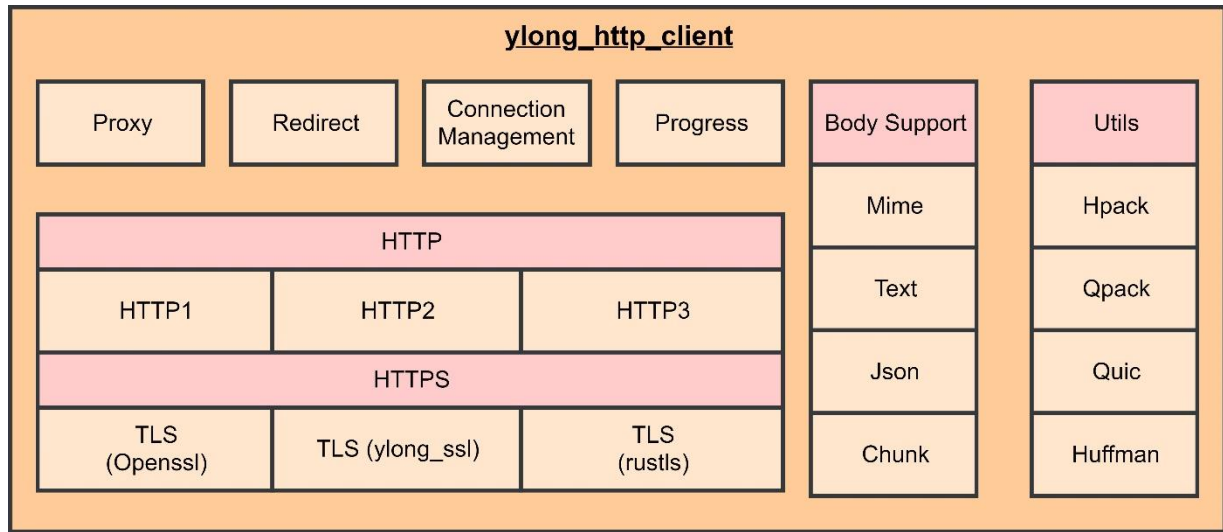
我们当前结合 Rust 和终端通信场景的实践的简单介绍



## Rust 与终端 HTTP 通信场景结合

我们 Ylong HTTP 客户端库当前实现的基础功能：

- ✓ 支持同步逻辑和异步逻辑
- ✓ 支持 HTTP/1.1、HTTP/2、HTTP/3 协议及其组件
- ✓ 支持 HTTPS
- ✓ 支持客户端代理
- ✓ 支持自动重定向
- ✓ 支持连接管理和复用
- ✓ 支持进度显示
- ✓ 支持发送 Multipart/ Chunk 格式 body

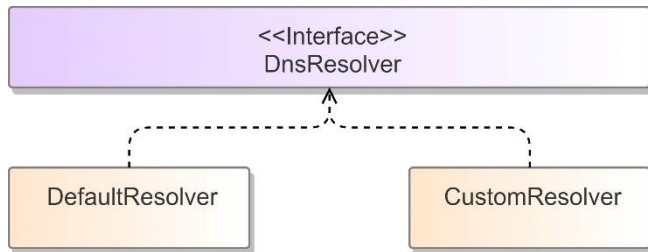


## Rust 与终端 HTTP 通信场景结合

针对弱网环境的处理举例：

- 支持上层自定义 DNS 解析逻辑：利用 Rust 的 trait 实现继承关系，方便用户根据自身需求自定义 DNS 解析器。DNS 操作也是 IO 操作，可以使用 Rust 异步 IO 逻辑来提升性能。

```
/// Trait for customizing DNS resolution.
pub trait DnsResolver: Send + Sync {
    fn resolve(&self, name: Name) -> Resolving;
}
```

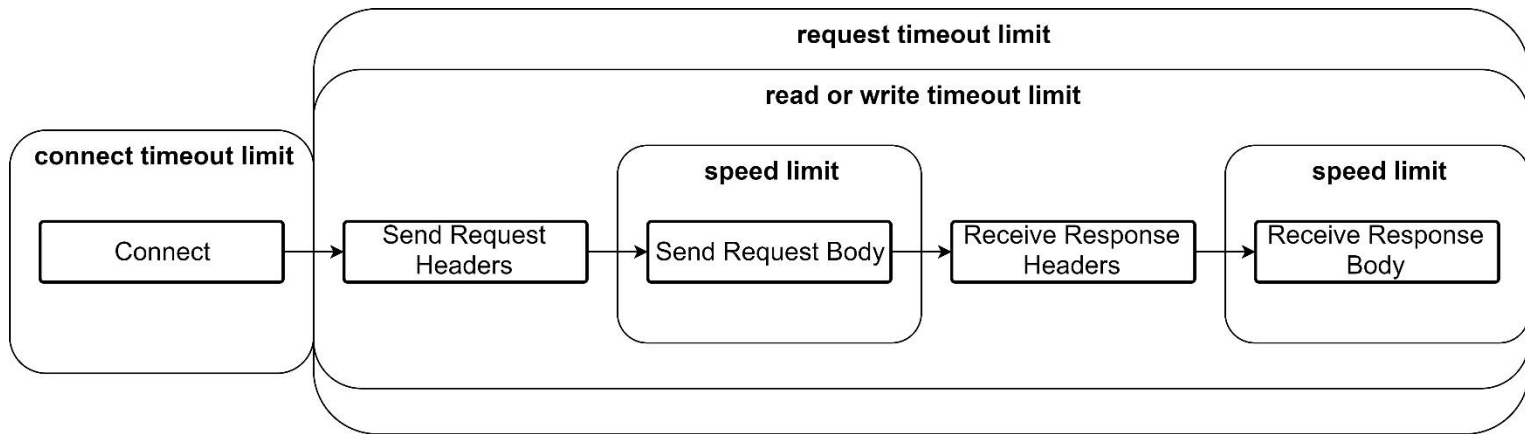


- 支持高版本的 HTTP 协议，如 HTTP/2、HTTP/3。利用 IO 复用机制或 UDP 连接可以提高弱网环境下的传输性能。

## Rust 与终端 HTTP 通信场景结合

针对网络不稳定场景：

- 支持用户设置速度限制范围：给消息 body 读取逻辑增加速度的上下限，以及时根据网络变化做出操作。
- 支持用户设置连接和请求的超时时间：给请求的各个区间设置定时器，以及时检测网络变化。



## Rust 与终端 HTTP 通信场景结合

提供用户界面表现的相关接口：

- 暂停、停止、重试、显示回调：利用 Rust 闭包、trait 实现下载操作回调。方便上层在传输过程中操作传输行为。

```
/// Downloader is used for download a response body.
pub struct Downloader<T> {
    operator: T,
    body: Response,
    config: DownloadConfig,
    info: Option<DownloadInfo>,
}
```

```
// DownloadOperator provides callback functions.
pub trait DownloadOperator {
    fn poll_download(
        self: Pin<&mut Self>,
        cx: &mut Context<'_>,
        data: &[u8],
    ) -> Poll<Result<usize, HttpClientError>>;

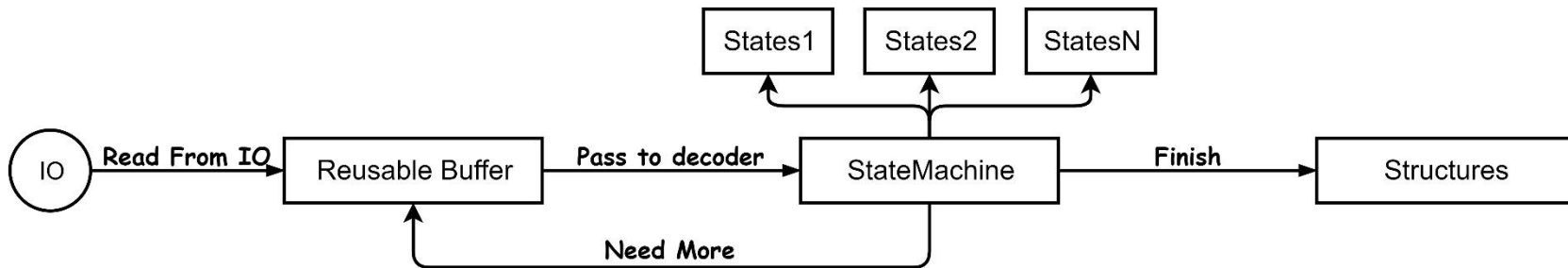
    fn poll_progress(
        self: Pin<&mut Self>,
        _cx: &mut Context<'_>,
        _downloaded: u64,
        _total: Option<u64>,
    ) -> Poll<Result<(), HttpClientError>> {
        Poll::Ready(Ok(()))
    }
}
```



## Rust 与终端 HTTP 通信场景结合

功耗和性能表现：

- 使用 Rust 异步 IO 可以充分利用线程资源，带来稳定的性能表现。
- 针对 HTTP 协议层的解析逻辑进行优化，使用状态机和可复用内存减少运行内存占用。
- 管理和复用已有连接，减少连接的反复创建。



# Thank you!

