目錄

Project V

Project V 是一个工具集合，它可以帮助你打造专属的基础通信网络。Project V 的核心工具称为 `V2Ray`，其主要负责网络协议和功能的实现，与其它 Project V 通信。V2Ray 可以单独运行，也可以和其它工具配合，以提供简便的操作流程。

本站点主要包含了 V2Ray 的使用手册，以及其它 Project V 相关工具的介绍与链接。

主要特性

- 多入口多出口: 一个 V2Ray 进程可并发支持多个入站和出站协议，每个协议可独立工作。
- 可定制化路由: 入站流量可按配置由不同的出口发出。轻松实现按区域或按域名分流，以达到最优的网络性能。
- 多协议支持: V2Ray 可同时开启多个协议支持，包括 Socks、HTTP、Shadowsocks、VMess 等。每个协议可单独设置传输载体，比如 TCP、mKCP、WebSocket 等。
- 隐蔽性: V2Ray 的节点可以伪装成正常的网站（HTTPS），将其流量与正常的网页流量混淆，以避开第三方干扰。
- 反向代理: 通用的反向代理支持，可实现内网穿透功能。
- 多平台支持: 原生支持所有常见平台，如 Windows、Mac OS、Linux，并已有第三方支持移动平台。

赞助商

本站点由 GitBook 生成，托管于 GitHub。如需修改本站的内容，请向这个 Repo 提交 Pull Request。

更新日志

本页列出了常规版本的功能升级记录，未列出的版本通常为 bug 修复。

2019.03.01 v4.18

- 路由中的端口列表新增了混合格式。
- 更新了路由中 `geosite` 的说明，和 DNS 服务器中静态域名列表的说明。
- TLS 配置中添加了禁用系统 CA 的选项。
- 路由中添加了流量属性检测，可用于检测 URL。

2019.02.22 v4.17

- V2Ctl 中新增了 `tlsping` 工具。

2019.02.15 v4.16

使用相关:

- DNS 传出代理支持修改目标 DNS 服务器的设定。

开发相关:

- 'ext' 仓库已完全迁移到 'core'。#1541

2019.02.08 v4.15

- DNS 传出代理。

2019.02.01 v4.14

- DNS 支持了静态域名到域名的匹配。

2019.01.18 v4.13

- 更新了 Quic 库，与之前的版本不兼容。
- DNS 配置中新增了 `tag` 选项。

2019.01.11 v4.11

开发相关:

- 新增了 core.DialUDP
- internet.DialSystem 现在会调用 internet.ListenSystemPacket 去建立 UDP 连接。

2018.12.07 v4.8

使用相关:

- Bug 修复

开发相关:

- 新增了 API 文档
- 新增了 internet.RegisterListenerController 方法
- 新增了 internet.RegisterDialerController 方法

2018.11.30 v4.7

- Freedom 中新增了 UseIPv4 和 UseIPv6 模式。
- 传输方式新增了 QUIC。

2018.11.23 v4.6

- Freedom 的 `"UseIP"` 模式会根据出站所用的 IP 地址自动使用 IPv4 或 IPv6 目标地址。

2018.11.16 v4.4

- 路由中新增了负载均衡配置。
- 优化了内存使用效率。

2018.11.05 v4.1

- 配置文件格式简化。主要是传入传出代理和路由部分。旧版格式仍然可用，但推荐尽快升级到最新格式。
- 优化了 GeoIP 的性能，减少了内存占用，提升了匹配效率。

2018.11.02 v4.0

- 反向代理
- 新增了 ppc64 和 ppc64le 预编译包。

2018.10.12 v3.47

- 升级了自动构建工具

2018.09.28 v3.44

- Dokodemo-door 现已支持 Linux 的 TProxy。

2018.09.21 v3.43

- DNS 查询支持了按域名选择服务器的功能。

2018.09.14 v3.40

- 传输配置中新增了连接选项，可用于配置 VPN。
- 传输配置中新增了 TCP Fast Open 选项。

2018.09.03 v3.38

- mKCP 中新增了 WireGuard 伪装。
- 修复了 mips/mips64 中 softfloat 版本的编译问题。

2018.08.31 v3.37

- 优化了读取网络连接时的性能，见环境变量。
- VMess 在 ARM64 平台上将默认使用 AES-128-GCM 作为加密方式。
- 使用 Go 1.11 编译。

2018.08.24 v3.36

- 大幅提升了路由中子域名（`domain:`）匹配的效率。
- 路由中支持了完整域名匹配。

2018.08.07 v3.34

- 在大多数设备上有了更精确的内存控制

2018.07.27 v3.32

- 现在可以探测 BitTorrent 流量了。
- 路由配置中新增了 `protocol` 选项用于按流量类型进行路由选择。
- 路由配置中新增了 `geosite:speedtest` 用于适配所有的 Speedtest.net 公用服务器。

2018.07.20 v3.31

- 新增了 Domain Socket 传输方式 (感谢 @xiaokangwang)。

2018.07.13 v3.30

- 解决了一个可能会导致 mKCP 断流的问题。
- 感谢来自俄罗斯的朋友，我们现在有俄语文档了。

2018.07.06 v3.29

- 新增了 MTProto 代理协议。

2018.06.29 v3.27

- DNS 支持了 EDNS client subnet。
- DNS 的静态 IP 匹配现在支持子域名了。

2018.06.15 v3.26

- 新增了 Dragonfly BSD 的预编译版本。

2018.06.01 v3.24

- JSON 配置中新增了用户级别的缓存控制选项。
- mKCP 新增了 DTLS 伪装。
- TLS 现在默认只使用 TLS 1.3 推荐的加密套件。

2018.05.25 v3.23

- JSON 配置文件中，端口支持从环境变量加载。
- JSON 配置文件支持从指定文件加载 IP 和域名。

2018.04.20 v3.19

- 入站代理的流量统计。

2018.04.13 v3.17

- V2Ray 可直接加载来自 HTTP(s) 的配置文件。
- V2Ray 的 TLS 可使用 CA 证书自动为任意域名签发新的证书。
- HTTP/2 传输方式。

2018.04.06 v3.16

- 统计信息。开启方式略麻烦，请仔细看文档。
- Shadowsocks 入站协议现在可以只监听 UDP 端口而不监听 TCP 了。强烈建议不在同一端口上同时监听 TCP 和 UDP。

2018.03.02 v3.11

- VMess Inbound 提供了一个选项，用于禁止客户端使用不安全的加密方式。
- 提供了 ARMv7 编译版。
- 提供了不要求 FPU 的 MIPS 编译版。

2018.02.23 v3.10

- 日志格式修改，每条日志前添加了 Session ID，用于区分不同的代理请求。
- 修复了一些问题。

2018.02.16 v3.9

- 新年快乐!
- 修复了一些问题。

2018.02.09 v3.7

- 开发者预览：远程控制。
- 修复了一些问题。

2017.12.29 v3.5

- Geoip 支持名为“Private”的私有地址列表。

2017.12.22 v3.4

- Websocket 现在可以识别 X-Forwarded-For 并用做源地址。
- 支持 s390x 平台。

2017.12.08 v3.1

- 支持本地策略
- 支持从环境变量指定的路径加载配置文件

2017.12.01 v3.0

- 支持 Shadowsocks AEAD

2017.11.18 v2.50

- `v2ray` 现在会尝试使用 `v2ctl` 进行配置文件解析，请确保这两个文件在同一目录下；
- 路由中新增 `IPOnDemand` 策略进行同步 IP 解析；

2017.11.10 v2.47

- `geosite` 的数据现在放在 `geosite.dat` 文件中了。
- 一些修复。

2017.11.03 v2.46

- 路由中添加 `geosite:cn`，等效于现有的 `chinasites`，但更加灵活。
- 路由中添加 `geoip`。
  - 安装包中新增了 `geoip.dat` 文件，包含所有 `geoip` 信息，此文件必须和 `v2ray` 程序放置于同一路径下。由于安装脚本未及时更新，服务器端和部分客户端可能需要手动复制此文件。
- 安装包中新增 `v2ctl` 程序，可使用命令 `v2ctl verify /path/to/v2ray` 来验证 `v2ray` 程序签名的有效性。

2017.10.27 v2.44

- HTTP 代理中加入了 Basic Authentication 支持。
- 修复了一些 bug。

2017.10.06 v2.40

- 修复了导致内存泄漏的问题。

2017.09.29 v2.39

- 当远程服务器关闭连接时，入站代理现在会尽快断开与客户端的连接。
- 默认连接超时时间更改为 5 分钟。

2017.05.12 v2.27

- 路由中新增了“域名”匹配选项。

2017.05.02 v2.26

- 各种稳定性修复。

2017.04.28 v2.25

- 可以简单地探测 HTTP 和 HTTPS 流量并根据其内容进行转发。

2017.04.21 v2.24

- 修复了 Mux 的稳定性问题。
- 提升了总体的内存使用效率。

2017.04.15 v2.23

- 提升了 Mux 的性能。
- Mux 中可以配置并发连接数了。
- 提升了 HTTP 代理的性能。
- 移除了 connectionReuse 的配置（由 Mux 替代）。

2017.04.08 v2.22

- Mux.Cool 协议

2017.02.25 v2.20

- 安装包中可执行文件附带了 gpg 签名。
- Windows 安装包中带了新的 wv2ray.exe，运行时不会出现任何界面。

2017.02.18 v2.19

- 服务器端强制开启防重放攻击（Anti replay attack）机制。
  - 不会影响任何现有客户端。
- VMess AES-128-GCM / Chacha20-Poly1305 / None 加密方式升级。
  - 如正在使用其中任何一个加密方式，请同时升级客户端和服务器。
  - AES-128-CFB 不受影响。
- 使用 Golang 1.8 编译，加入对 32 位 MIPS 的支持。

2017.02.11 v2.18

- 清理了 WebSocket 相关的代码。
- 移除了 allowPassive 设置。现在的行为相当于 allowPassive = true。
- Bug 修复。

2017.02.04 v2.17

- Bug 修复。

2017.01.28 v2.16

- Freedom 中可以指定一个重定向地址，将所有数据发往这个地址。此选项可用于适配 Shadowsocks Obfs 或 KCPTun 等工具。
- VMess 的“不加密”选项性能优化，与之前版本不兼容。
- 新年快乐！

2017.01.16 v2.15

- mKCP 和 WebSocket 现已适用于所有的出站（入站）协议。

2017.01.09 v2.14

- Socks 5 出站协议。
- mKCP 新增微信视频通话伪装。

2017.01.02 v2.13

- 修复了一些问题。

2016.12.26 v2.12

- 修复了一些问题。

2016.12.19 v2.11

- JSON 配置文件可以写注释了。

2016.12.12 v2.10

- VMess 中增加了 AES-GCM 和 Chacha20-Poly1305 加密方式；

2016.12.05 v2.9

- 修复了与 Alpine Linux 的兼容性；

2016.11.28 v2.8

- Shadowsocks 服务器端现接受 OTA 设置，可强制开启或关闭 OTA 验证；
- 小修小补；

2016.11.21 v2.7

- 现在可以给主入站出站连接设置标识了；
- 一些小修小补；

2016.11.14 v2.6

- 可将一个出站协议发出的数据转发至另一出站协议；
- 路由可根据入站协议的标识来进行判断了；

2016.11.07 v2.5

- 新增 Shadowsocks 出站协议；
- 新增 TCP 连接中的 HTTP 头部伪装；

2016.10.24 v2.4

- 每个入站出站协议可以配置各自的 TCP / mKCP / WebSocket 设置了；
- 路由现可以跟据来源 IP 进行转发；

2016.10.17 v2.3

- 重构了配置文件相关的代码，引入了基于 Protobuf 的新格式；
- 增加了 OpenBSD 的二进程文件；
- 小修小补；

2016.09.19 v2.2

- 新增了 WebSocket 的载体（感谢 @xiaokangwang）；

2016.09.19 v2.1

- mKCP 性能提升；

2016.08.20 v2.0

- 一周年

2016.08.15 v1.24

- mKCP 新增了 BT 数据包伪装。
- 入站连接现在可以设置 allowPassive 来允许被动连接，如 IMAP。

2016.08.08 v1.23

- 优化了 mKCP 数据包。此版本的 mKCP 与之前版本不兼容，请同时升级客户端和服务器。
- mKCP 现在可以经过配置，伪装成视频通话的数据。

2016.08.01 v1.22

- 修复了 mKCP 中一个内存泄漏问题；
- 新增了 FreeBSD 的支持；

2016.07.25 v1.21

- 提升了 ChaCha20 的性能（感谢 aead@）；
- 修复了一些问题；

2016.07.18 v1.20

- KCP 中新增了 readBufferSize 和 writeBufferSize 选项；
- 修复了一些问题；

2016.07.11 v1.19

- 新增了 TLS 选项，平台支持的所有协议都可以开启 TLS；

- 修复了 KCP 的性能问题；
- 修复了一个可能导致 KCP 中 EOF 响应的问题；

2016.07.04 v1.18

- 修复了 KCP 中连接多过的问题；
- 降低了 KCP 对 CPU 的使用；
- 修复了一些其它的问题；

2016.06.19 v1.17

- 整合了 KCP（感谢 xiaokangwang、xtaci、skywind3000）
- 修复了一些小问题

2016.06.12 v1.16

- TCP 连接重用默认开启；
- Dokodemo-door 现在可以识别由 iptables 转发的数据了；
- Blackhole 可配置自动发送 HTTP 形式的拒绝数据；

2016.06.05 v1.15

- 增加了 TCP 连接重用选项，对于 HTTP 请求的性能有明显的提升；
- 支持监听指定的 IP 地址而非全部；
- 支持从指定的 IP 地址发出数据；
- 修复了 HTTP 代理中一个可能会导致内存耗尽的问题；

2016.05.29 v1.14

- 修复了 HTTP 代理中一个可能导致崩溃的问题；
- 安装脚本中增加了一些功能；
- DNS 中增加了静态路由功能；
- 官方服务器地址变动；

2016.05.16 v1.13

- 内置 DNS 服务，可搭配 chinasites 和 chinaip 进行更为精准的路由判断；
- 修复了一个 UDP 转发时的问题；

2016.05.01 v1.12.1

- 修复了 VMess 协议中的一个 bug。

2016.05.01 v1.12

- 再次尝试修复内存使用问题；
- 提升了 Shadowsocks 的性能；
- loglevel 增加了新的选项“none”表示不记录任何 log；

2016.04.18 v1.11

- 尝试修复内存使用问题。

2016.03.07 v1.10

- 修复了动态端口刷新时的性能问题。

2016.02.29 v1.9.1

- 修复了 Shadowsocks 中 OTA 的问题。

2016.02.29 v1.9

- Shadowsocks 支持了 ChaCha20 加密方式；
- 默认配置文件增加了更多的直连网站；
- 动态端口中现已可以自动创建帐号，无需事先指定；

2016.02.22 v1.8

- 更新了安装脚本 install-release.sh （感谢 netcookies@)：
  - 现在可以自动停止 V2Ray 进程，并在更新完成后自动运行 V2Ray；
  - install-release.sh 接受 --proxy 参数并从指定的代理下载 V2Ray；
- 使用 Go 1.6 编译，提升了 AES 加密的性能；
- 一些小修小补；
- 官方服务器 IP 更新，请重新下载安装包以获得最新的配置。

2016.02.08 v1.7

- 提升了 UDP 转发的性能；
- 提升了 Shadowsocks 的安全性；
- 修复了一些问题；
- 祝大家新年快乐！

2016.02.01 v1.6

- 服务器端支持 Shadowsocks；
  - 协议详情

2016.01.25 v1.5

- 修复了一个 VMess 中的安全性问题，导致了 1.5 和之前版本不兼容，请同时升级你的客户端和服务器端；
- 修复了一个路由不能正常工作的问题；
- 动态端口；
- 略微提升了性能；

2016.01.18 v1.4

- 更新了安装脚本，在 Debian / Ubuntu / CentOS 7 中可自动安装和更新 V2Ray；
- 修复了一个 VMess 的内存使用问题；

2016.01.11 v1.3

- Wiki 中更新了一些英语页面（感谢 chenxiaoqino）；
- Docker 配置文件（感谢 adoot）；
- HTTP 代理（感谢 adoot）；
- 路由中内置了常见的国内网站域名；
- VMess 配置中新增了 alterId 选项；
- 修复了若干小问题；

2015.12.14 v1.2

- 简洁且高效的国内 IP 路由；
- 错误日志可写入文件；
- 路由中支持正则表达式方式的域名匹配；
- 修复了一个 SOCKS 协议的兼容性 bug；

2015.12.07 v1.1

- 修复了一个 VMess 协议的 bug，也导致了 1.1 和 1.0 的应用程序不兼容，配置文件不受影响；
- 修复了一个 InboundDetourHandler 中的 bug；

2015.11.30 v1.0

V2Ray 1.0 正式版，包含以下功能：

- Socks 4 / 5 代理协议；
- 可以防止重放攻击的高速中继协议；
- 静态路由功能，用户可选择性屏蔽或代理指定的 IP 段或域名；

工作机制

单服务器模式

和其它的网络代理工具一样，你需要在一台配置了 V2Ray 的服务器，然后在自己的设备上安装 V2Ray 客户端，然后即可流畅地访问互联网。

graph LR; A(PC) -.- B(防火墙); B -.-> C(墙外网站); A --> D(V2Ray/VPS); D --> C; A --> E(墙内网站);
一个 V2Ray 服务器可同时支持多台设备，使用不同的代理协议访问。同时，经过合理的配置，V2Ray 可以识别并区分需要代理和不需要代理的流量，直连的流量不需要绕路。

桥接模式

如果你不想在每一台设备上都配置路由，你也可以设置一台中转服务器，用于接收客户端发来的所有流量，然后在服务器中进行转发判断。

graph LR; A(PC) -.-> B(防火墙); B -.-> C(墙外网站); A --> D(墙内VPS); D --> E(墙外VPS); E --> C; D --> F(墙内网站);

工作原理

在配置 V2Ray 之前，不妨先来看一下 V2Ray 的工作原理，以下是单个 V2Ray 进程的内部结构示意图。多个 V2Ray 之间互相独立，互不影响。

graph LR; A1(inbound) --> D(Dispatcher / Router / DNS); A2(inbound) --> D; A3(inbound) --> D; A4(inbound) --> D; D --> B1(outbound); D --> B2(outbound); D --> B3(outbound); D --> B4(outbound);

- 需要配置至少一个入站协议（Inbound）和一个出站协议（Outbound）才可以正常工作。协议列表见第二章节。
  - 入站协议负责与客户端（如浏览器）通信：
    - 入站协议通常可以配置用户认证，如 ID 和密码等；
    - 入站协议收到数据之后，会交给分发器（Dispatcher）进行分发；
  - 出站协议负责将数据发给服务器，如另一台主机上的 V2Ray。
- 当有多个出站协议时，可以配置路由（Routing）来指定某一类流量由某一个出站协议发出。
  - 路由会在必要时查询 DNS 以获取更多信息来进行判断。

具体的配置格式详见第二章节。

下载安装

平台支持

V2Ray 在以下平台中可用：

- Windows 7 及之后版本（x86 / amd64）；
- Mac OS X 10.10 Yosemite 及之后版本（amd64）；
- Linux 2.6.23 及之后版本（x86 / amd64 / arm / arm64 / mips64 / mips）；
  - 包括但不限于 Debian 7 / 8、Ubuntu 12.04 / 14.04 及后续版本、CentOS 6 / 7、Arch Linux；
- FreeBSD (x86 / amd64)；
- OpenBSD (x86 / amd64)；
- Dragonfly BSD (amd64)；

下载 **V2Ray**

预编译的压缩包可以在如下几个站点找到：

1. Github Release: github.com/v2ray/v2ray-core
2. Github 分流: github.com/v2ray/dist
3. Homebrew: github.com/v2ray/homebrew-v2ray
4. Arch Linux: packages/community/x86_64/v2ray/
5. Snapcraft: snapcraft.io/v2ray-core

压缩包均为 zip 格式，找到对应平台的压缩包，下载解压即可使用。

验证安装包

V2Ray 提供两种验证方式：

1. 安装包 zip 文件的 SHA1 / SHA256 摘要，在每个安装包对应的 `.dgst` 文件中可以找到。
2. 可运行程序（v2ray 或 v2ray.exe）的 gpg 签名，文件位于安装包中的 v2ray.sig 或 v2ray.exe.sig。签名公钥可以在代码库中找到。

Windows 和 **Mac OS** 安装方式

通过上述方式下载的压缩包，解压之后可看到 v2ray 或 v2ray.exe。直接运行即可。

Linux 发行版仓库

部分发行版可能已收录 V2Ray 到其官方维护和支持的软件仓库/软件源中。出于兼容性、适配性考虑，您可以考虑选用由您发行版开发团队维护的软件包或下文的安装脚本亦或基于已发布的二进制文件或源代码安装。

Linux 安装脚本

V2Ray 提供了一个在 Linux 中的自动化安装脚本。这个脚本会自动检测有没有安装过 V2Ray，如果没有，则进行完整的安装和配置；如果之前安装过 V2Ray，则只更新 V2Ray 二进制程序而不更新配置。

以下指令假设已在 su 环境下，如果不是，请先运行 sudo su。

运行下面的指令下载并安装 V2Ray。当 yum 或 apt-get 可用的情况下，此脚本会自动安装 unzip 和 daemon。这两个组件是安装 V2Ray 的必要组件。如果你使用的系统不支持 yum 或 apt-get，请自行安装 unzip 和 daemon

```
bash <(curl -L -s https://install.direct/go.sh)
```

此脚本会自动安装以下文件：

- `/usr/bin/v2ray/v2ray`： V2Ray 程序；
- `/usr/bin/v2ray/v2ctl`： V2Ray 工具；
- `/etc/v2ray/config.json`： 配置文件；
- `/usr/bin/v2ray/geoip.dat`： IP 数据文件
- `/usr/bin/v2ray/geosite.dat`： 域名数据文件

此脚本会配置自动运行脚本。自动运行脚本会在系统重启之后，自动运行 V2Ray。目前自动运行脚本只支持带有 Systemd 的系统，以及 Debian / Ubuntu 全系列。

运行脚本位于系统的以下位置：

- `/etc/systemd/system/v2ray.service`：Systemd
- `/etc/init.d/v2ray`：SysV

脚本运行完成后，你需要：

1. 编辑 /etc/v2ray/config.json 文件来配置你需要的代理方式；
2. 运行 service v2ray start 来启动 V2Ray 进程；
3. 之后可以使用 service v2ray start|stop|status|reload|restart|force-reload 控制 V2Ray 的运行。

go.sh 参数

go.sh 支持如下参数，可在手动安装时根据实际情况调整：

- `-p` 或 `--proxy`：使用代理服务器来下载 V2Ray 的文件，格式与 curl 接受的参数一致，比如 `"socks5://127.0.0.1:1080"` 或 `"http://127.0.0.1:3128"`。
- `-f` 或 `--force`：强制安装。在默认情况下，如果当前系统中已有最新版本的 V2Ray，go.sh 会在检测之后就退出。如果需要强制重装一遍，则需要指定该参数。
- `--version`：指定需要安装的版本，比如 `"v1.13"`。默认值为最新版本。
- `--local`：使用一个本地文件进行安装。如果你已经下载了某个版本的 V2Ray，则可通过这个参数指定一个文件路径来进行安装。

示例：

- 使用地址为 127.0.0.1:1080 的 SOCKS 代理下载并安装最新版本：`./go.sh -p socks5://127.0.0.1:1080`
- 安装本地的 v1.13 版本：`./go.sh --version v1.13 --local /path/to/v2ray.zip`

Docker

V2Ray 提供了两个预编译的 Docker image：

- v2ray/official: 包含最新发布的版本，每周跟随新版本更新；
- v2ray/dev: 包含由最新的代码编译而成的程序文件，随代码库更新；

两个 image 的文件结构相同：

- /etc/v2ray/config.json: 配置文件
- /usr/bin/v2ray/v2ray: V2Ray 主程序
- /usr/bin/v2ray/v2ctl: V2Ray 辅助工具
- /usr/bin/v2ray/geoip.dat: IP 数据文件
- /usr/bin/v2ray/geosite.dat: 域名数据文件

新手上路

在下载并安装了 V2Ray 之后，你需要对它进行一下配置。这里介绍一下简单的配置方式，只是为了演示，如需配置更复杂的功能，请参考后续的配置文件说明。

客户端

在你的 PC （或手机）中，你需要运行 V2Ray 并使用下面的配置：

```
{
  "inbounds": [{
    "port": 1080,  // SOCKS 代理端口，在浏览器中需配置代理并指向这个端口
    "listen": "127.0.0.1",
    "protocol": "socks",
    "settings": {
      "udp": true
    }
  }],
  "outbounds": [{
    "protocol": "vmess",
    "settings": {
      "vnext": [{
        "address": "server", // 服务器地址，请修改为你自己的服务器 ip 或域名
        "port": 10086,  // 服务器端口
        "users": [{ "id": "b831381d-6324-4d53-ad4f-8cda48b30811" }]
      }]
    }
  },{
    "protocol": "freedom",
    "tag": "direct",
    "settings": {}
  }],
  "routing": {
    "domainStrategy": "IPOnDemand",
    "rules": [{
      "type": "field",
      "ip": ["geoip:private"],
      "outboundTag": "direct"
    }]
  }
}
```

上述配置唯一要改的地方就是你的服务器 IP，配置中已注明。上述配置会把除了局域网（比如访问路由器）之外的所有流量转发到你的服务器。

服务器

然后你需要一台防火墙外的服务器，来运行服务器端的 V2Ray。配置如下：

```
{
  "inbounds": [{
    "port": 10086, // 服务器监听端口，必须和上面的一样
    "protocol": "vmess",
    "settings": {
      "clients": [{ "id": "b831381d-6324-4d53-ad4f-8cda48b30811" }]
    }
  }],
  "outbounds": [{
    "protocol": "freedom",
    "settings": {}
  }]
}
```

服务器的配置中需要确保 `id` 和端口与客户端一致，就可以正常连接了。

运行

- 在 Windows 和 macOS 中，配置文件通常是 V2Ray 同目录下的 `config.json` 文件。直接运行 `v2ray` 或 `v2ray.exe` 即可。
- 在 Linux 中，配置文件通常位于 `/etc/v2ray/config.json` 文件。运行 `v2ray --config=/etc/v2ray/config.json`，或使用 systemd 等工具把 V2Ray 作为服务在后台运行。

更多详见的说明可以参考白话文教程和配置文件说明。

命令行参数

V2Ray

V2Ray 的程序文件的命令行参数如下：

```
v2ray [-version] [-test] [-config=config.json] [-format=json]
```

> `-version`

只输出当前版本然后退出，不运行 V2Ray 主程序。

> `-test`

测试配置文件有效性，如果有问题则输出错误信息，不运行 V2Ray 主程序。

> `-config`

配置文件路径，可选的形式如下:

- 本地路径，可以是一个绝对路径，或者相对路径。
- `"stdin:"`：表示将从标准输入读取配置文件内容，调用者必须在输入完毕后关闭标准输入流。
- 以 `http://` 或 `https://` (均为小写)开头: V2Ray 将尝试从这个远程地址加载配置文件。

> `-format`

配置文件格式，可选的值有：

- `json`：JSON 格式；
- `pb` 或 `protobuf`：Protobuf 格式；

> 当 `-config` 没有指定时，V2Ray 将先后尝试从以下路径加载 `config.json`：
>
> - 工作目录（Working Directory）
> - 环境变量中 `v2ray.location.asset` 所指定的路径

V2Ctl

V2Ctl 是一个集合，它有若干个子命令组成。全局的命令行形式如下：

```
v2ctl <command> <options>
```

> `command`

子命令，有以下选项:

- `api`：调用 V2Ray 进程的远程控制指令。
- `config`：从标准输入读取 JSON 格式的配置，然后从标准输出打印 Protobuf 格式的配置。
- `cert`：生成 TLS 证书。
- `fetch`：抓取远程文件。
- `tlsping`：(V2Ray 4.17+) 尝试进行 TLS 握手。
- `verify`：验证文件是否由 Project V 官方签名。
- `uuid`：输出一个随机的 UUID。

V2Ctl Api

`v2ctl api [--server=127.0.0.1:8080] <Service.Method> <Request>`

调用 V2Ray 进程的远程控制指令。示例：

```
v2ctl api --server=127.0.0.1:8080 LoggerService.RestartLogger ''
```

V2Ctl Config

`v2ctl config`

此命令没有参数。它从标准输入读取 JSON 格式的配置，然后从标准输出打印 Protobuf 格式的配置。

V2Ctl Cert

`v2ctl cert [--ca] [--domain=v2ray.com] [--expire=240h] [--name="V2Ray Inc"] [--org="V2Ray Inc] [--json] [--file=v2ray]`

生成一个 TLS 证书。

> `--ca`

如果指定此选项，将会生成一个 CA 证书。

> `--domain`

证书的 Alternative Name 项。该参数可以多次使用，来指定多个域名。比如 `--domain=v2ray.com --domain=v2ray.cool` 。

> `--expire`

证书有效期。格式为 Golang 的时间长度。

> `--name`

证书的 Command Name 项。

> `--org`

证书的 Orgnization 项。

> `--json`

将生成的证书以 V2Ray 支持的 JSON 格式输出到标准输出。默认开启。

> `--file`

将证书以 PEM 格式输出到文件。当指定 `--file=a` 时，将生成 `a_cert.pem` 和 `a_key.pem` 两个文件。

V2Ctl Fetch

`v2ctl fetch <url>`

抓取指定的 URL 的内容并输出，只支持 HTTP 和 HTTPS。

V2Ctl TlsPing

`v2ctl tlsping <domain> --ip=[ip]`

向指定的域名发起 TLS 握手。

> domain

目标域名

> --ip

此域名的 IP 地址。如果未指定此参数，V2Ctl 将使用系统的 DNS 进行域名解析。

V2Ctl Verify

`v2ctl verify [--sig=/path/to/sigfile] <filepath>`

此命令用于验证一个文件是否由 Project V 官方签名。

> `--sig`

签名文件路径，默认值为待验证文件加入'.sig'后缀。

> `filepath`

待验证文件路径。

V2Ctl UUID

`v2ctl uuid`

此命令没有参数。每次运行都会输出一个新的 UUID。

资助 **V2Ray** 发展

V2Ray 是一个非营利项目，它的所有代码均公开，软件可以自由使用，不受限制。如果您喜爱本项目，可以通过下面的方式请作者喝一杯咖啡。

您的捐款是对 Project V 的无偿贡献，和 Project V 提供的技术和服务没有直接关联。对于每一份捐款，我们都会以邮件的方式确认。如果你不希望收到我们的邮件，请在捐款时留言注明。

对于单次捐赠的价值超过 $50，您可以从如下选项中指定一项:

- 为期一个月的首页广告 (您的名字，或 Logo);
- 按您的喜好修改我们的代码。限制：不可影响用户使用，不可影响文档。您的修改将保留一个月。

传统方式

> Paypal

可以通过 Paypal 内部转帐 或者信用卡的方式捐赠。

由于 Paypal 收取非常高的手续费，小于 1 美元的捐赠会变得没有意义。

> Patreon

Patreon 是一个定期投食的平台，即每月捐赠固定金额。一旦设定完毕，每个月都会收到我们的小礼物哦。

> 礼品卡

目前只支持美亚礼品卡，可在 Amazon 或者淘宝购买。请寄送至 `love@v2ray.com` 。

极客方式

相比起传统方式，加密货币更适合跨境支付。加密货币通常都是匿名持有，其交易也很难被跟踪。

由于加密货币天生的匿名特性，如果有必要的话，请事先联系项目组表明捐赠者的身份。

> 比特币 Bitcoin

地址: `3GctrB7R5sMhJ73N4AKo56Bdf9RE3RJsuM`

点此链接显示二维码

> 比特现金 Bitcoin Cash

地址: `15oATKUq5mEfuzasPnsJ58TjJU5SvDJK97`

点此链接显示二维码

> 以太坊 Ethereum

地址: `0x112ee71189704fe04cabed4aa045f4461c8c8696`

点此链接显示二维码。你也可以通过这个地址发送这些货币：OMG、REP、GNT、DGD

> EOS

地址: `EOS8Civdok4CBN3jCpsaGQijzesjKof1eyaRFuBU5mLMtWVkLsy8a`

> 莱特币 Litecoin

地址: `LVdeH2HkCgGRs8ZEpan7fkAEEPbiJ4McoR`

> 门罗币 Monero

地址: `48kA4NyLRCWQvB7U2A77G66Z25uWbyzmoZSYjxJfrMR1J4dRFW6fWFLDn3wirAqP8ySnR4rnvoXWxfkNFhrK5ZxY1WyBqKg`

> 瑞波币 Ripple

地址: `r439fPk8DzCf4nSxkpfodEuE2cG4KVZQHq`

不需要 Tag

> 其它

如果你想捐赠其它的加密货币，请联系我们: `love@v2ray.com` 。

寻求帮助

Project V 提供了多种方式进行交流。

> Project V 团队支持中文和英文，请选择你所熟悉的语言来提问，以避免一些不必要的误会。管理员会以问题发起者使用的语言来回复；如果提问者使用了其它的语言，则以英文回复。

Github Issue

我们使用几个不同的仓库进行不同类型的讨论。

- 代码问题
  - 仅用于讨论 V2Ray 的代码问题，比如 bug。
- 未来计划
- 常规讨论

Telegram 讨论组

Project V 提供了下列讨论组，用于讨论不同类型的问题。

- 使用
  - 仅用于讨论 Project V 使用相关问题。
  - 严禁任何不相关的话题。
- 日常
  - 用于其它不相关的话题。

> Project V 的所有讨论组都不可以发布 18 禁、政治相关、仇恨言论，一旦发现立即封禁。

另外请关注 Project V 官方公告 以获得最新资讯。

Twitter

ProjectV2Ray 用于常规的项目进展通知。

Email

如果你想和 V2Ray 的开发人员私下讨论一些问题，可以通过下列的邮箱联系。

`love@v2ray.com`：基于 Gmail，主要通信邮箱，基本每天都查看。

`v2ray@protonmail.com`：基于 Protonmail，端对端加密安全性高，但登录不方便，不能做到每天都收邮件。

> 由于工作繁忙，不能保证每封邮件都回复，请见谅。请尽量使用社区讨论以得到更快速的社区响应。

如果你需要发送加密信息，可以搜索 `love@v2ray.com` 相关联的 PGP 公钥，或者从下面的文本导入。

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Comment: GPGTools - https://gpgtools.org

mQINBFaPeDABEADAbh3mk58UbKKWndztFKchtTRzU2xFWWRHTdYaNv7Eoo06wrxu
3eglp0vn+16DIfru4H62TQMS/XSvxib90c4c1tQ4FndVSzv896/IplLKzdrtgn85
c9eEX4V5b/tKKUmyrG593A/oDdsrpwaIgbNJdzbfqh7WLYESAevRkFJmBZMgfObs
0pVO/dX6TqS8iV/ARDPbPAzuLosTWXkrRi2+JQmEOKePLsdypRMV9bcgymDA8N6w
EyGzHyZ2I4wAALtjHXipWFYSZ/4ZK9q9H8G0XV/pk9Y4OFLPGR7T3VCtMwVMhGZk
CgtcwGOUoE68a/bb9P4FowONnM7tqjM5ef30qNbwe8dMY5DvThffEqQ70LnNc9sj
OpJF6njxV9ktjjLa3imAB5AStfwX1mBTkQTwnlqAU4pVFcMD6/z+kRFTZGP9nbcY
cxu5Fg1VVBHHbpgWS39uiwzIBSWVfZj2iHOPcJd9SCZW5xvClcVb1KsU9UD+D88m
uWBp3TUmxf0f53Uo7Flka1a7MyAEhfcorYsaRioqtPxTZ1z7oYbPLRqLbwmln/YV
va8XIINQ0hI5phz9kly76ksUqYSz1DRe924/1BwLMs1VzeayB81tOctYd3I7I0aF
5bF+RLWk0mvYgDBhw/be03PMn17OAkmR6IKNsFcowByNiNaVCLqoE8YVqQARAQAB
tCFWaWN0b3JpYSBSYXltb25kIDxsb3ZlQHYycmF5LmNvbT6JAj0EEwEKACcFAlaP
eDACGwMFCQeGH4AFCwkIBwMFFQoJCAsFFgIDAQACHgECF4AACgkQV4XalIxIroqu
ORAAoZNV6LVaDIxjJgH7BFh0oDNEa7Qac+INxkqd62cAmN3vAEF0PlwUhcz80twY
WvSPMwiLg85wdg/TzM8Ps8umWT6d9rhpkWnlZWyRFYJBpxn6vOpg0OHsgZpJ2+IM
iREmjK4sYFvuJjGi9e/BT0Mh2+ObcJejwd/+w5vhKb5seva+hw9Fyq569Y2+KMWG
1M29n3c7NZ6+abYuynxjGdHKOzaTUEGfjrGLZqROD7gb+xtXhBjqKr6jnYlZBAo5
iEAgnnzbiuDoa6Au+PosB2XZQ+rkecu/3AvmTYZ3wcEJt3FC4Gf9jY2A16ypmArm
ak2cmLAlz9CIXllFHjqvAHF68UCx2ptxzG3MoWhDFqws09zKSpwLcLdiAWbw//MM
otzeekx3+b+fjMUa4Eyjg5CvBN0wv2IbEjti744b3HWXW9yOIDrsTrMd8iixfjAe
3rt9egqTJkLlE8e57Rqnd3E8GpM6LWx8uYXo1aSFxayjJvf+Rvb4VzDA0DHElD2Q
dRVXJ/mhd14RHDc2QtERcF69GZZ79GFLcTMcXXBlQMDsY0vwBJ56Gi/IGImz9JpT
swSlU05VSdosMcr01niYoA9BEQjm9Smgd/1b+8qZEAIfvad+SabIRiDacOpN3xSh
SZzLzIkPukbEp+5hzXt7x22w9M+D6W0Oo22hS8zfV1ZavUa5Ag0EVo94MAEQANoc
tlwqgv2T372ucpnalh6js8Y/0KkpaevFK6pguZMP6frLX1J05mt+YRUp2UADeqGt
kuGEiXFZ5cr+smngF0l0HbRkflnILV7wd1WJJmQnvtAbfJfO+tovawUhQH19dLkB
8KMULcgs0Hrf7QLwus6zs9q0ASu/KkDuQj9gruwWfxk4W5MOQZohk0JcJAZAaBy/
ojKxz/91YjP3JEEzZuLqLiOz4RK38UBi96u5NEkqrmwCe6SAtpCHsa0dpj6LYZ1y
1ITmYSUnRwgol/UTRNySkZ2PozX6YFXrdZYtuWir0VClw50tFFkIOCaoT3Q5FpCO
SSS8smLjT9AAr8U3OZNTfNCQtFSNo8FUBYhXmMgskRC7oeV+t6LJ3rdJxgdaFcA+
GWL4srMedqxmbmW+8CYGIImmurjTy+C9zAtH88NFMYDnG312IIo9Cse3LnYl7eO/
KBd3YNvCL096Yqd4B/31fFy74/HaGEfNH7jwLvaY4moRdTff4kBERj7u39UE/jG/
b4NXmnezUZ55qkYQahoPkUIMzF0tn5bOIh1DOg55Wu/ZYe7DQJVE9L6te6KTxALy
ycD4WtQaP58SZpxHhGzEjEicPsPE1PpyRoW+Dl8lL+ppMbCl9zAV++BjkNeL5xZN
dsCtx/jbh+44X9VtK3G+2sdfdC6TqRbW/lELc8P7ABEBAAGJAiUEGAEKAA8FAlaP
eDACGwwFCQeGH4AACgkQV4XalIxIrorw8RAAg4L9/s8eg90qhslfPjWzvMyi7RvV
9erHwMB1utxjFDHKknTds2CvnS5JzjpjiT4BT8ICD9cetSP3d7WRNT/XmveeJXjB
TCxmswrT4H451qsGsCg9eiKIV2is/GolHLnKAvEHICRGsFeCRSB2rNkgWSQ/YJIt
6vLP1SvwTNY/Y8tyChSbCDaUpvmn4sYf6Qop1Svmn299+22ZVUBp20szsNw233SJ
IZL0WxSdrMqTdVsZ70m6VEOjOSa15yLryZrwN9U7qbpe2MmuygI90TVGQK4nLZG+
2XH2YVtBNTl9ZiFdpwq8d7+eMGcStKYQcdy6IZDd3cOeb2sTMUG31HJS3efTmSrT
tZpOz8TDeNjami78zR5qcW1VmGYbxlZHEhQnpG3U4qWRUZsy5TOYIXKHqCNc3rYH
iixcKE3UcyHBF7XjdKgpKtsgZfKoDXFz3XATJEeW80GEG05GVMoD9Yvf96Q7lvC6
U83vL8mHNjuG60KvBk7A9grYaVFioqRvbTdPf41G+tJpYKrHSJXPGglxHnzvBfiD
WWJIBRTyaCCglZLZ/a1sCsT1DciDWQoqqH7DuK2YvMXG3IDaOfoKAE/uW31azT0B
X7R92mxQ+gCZgePwG4gOE/xAqHZh84VzdWZJ4cehPgHa/z/enqbQbiQAwTXld4Wt
PeRpd1GUXXeoGCw=
=To5T
-----END PGP PUBLIC KEY BLOCK-----
```

常见问题

一般问题

制作 **V2Ray** 的目的是什么?

对于已公开的合法信息，人民有自由获取的权利。同时，人民也有言论自由的权利。

是否反对管制?

不反对管制。事实上任何事物都需要管制，以减少大众的生活成本，比如奶粉。但监管需要有法可依、有据可循。民众都认可的监管方式也是合理的，可接受的。

Project V 由谁主导开发?

我们是一群爱好自由的人们。由于政策压力，我们倾向于保持匿名，尽可能地不透露任何身份信息。

V2Ray 使用相关

V2Ray 如何升级

- 重新下载安装包，或者
- 如果你使用安装脚本进行安装，重新运行安装脚本即可

V2Ray 闪退

- 如果你使用 Linux 并开启了 systemd，可以使用 `journalctl -u v2ray` 查看 V2Ray 退出时的日志;
- 一般情况可以手动运行 `v2ray -config=<config-file> -test` 来查看错误信息;

兼容性保证

- 配置文件向后兼容至少一个大版本，即 V2Ray 4.x 可以正常加载 3.x 的配置文件。
- 所有基于 Protobuf 的通信协议，如 Api，向后兼容至少一个大版本。
- 所有基于二进制的通信协议，如 Shadowsocks 和 VMess。当服务器版本不低于客户端版本时，保持永久兼容；当客户端版本超过服务器版本时，保持至少 12 个小版本的兼容性。

V2Ray 错误信息

VMess: Invalid User

可能的原因:

1. 客户端和服务器端的用户 ID 不匹配;
2. 客户端和服务器端的用户 alterId 不匹配;
3. 客户端与服务器的时间不匹配，误差不能超过90秒钟;

Shadowsocks: Unknown address type

可能的原因：

1. Shadowsocks 协议的加密方式或是密码不匹配；

Socks: Unknown Socks version: 67

可能的原因：

- 你开启的是 Socks 代理，但在浏览器中配置了 HTTP 代理

解决方案：

- 在 V2Ray 中配置一个 HTTP 入站代理，然后把浏览器的设置指向这个代理

其它软件错误

访问 **Google** 时，浏览器中显示证书无效

错误信息：攻击者可能会试图从 www.google.com 窃取您的信息（例如：密码、通讯内容或信用卡信息）。了解详情 NET::ERR_CERT_COMMON_NAME_INVALID

原因：你的 DNS 缓存可能已被污染。

解决方案：

- 需要在使用代理的情况下，清空缓存并重新抓取 DNS 数据
- 在 V2Ray 中可使用 `sniffing` 功能克服一部分 DNS 污染。

项目授权

Project V 使用以下方式进行授权

V2Ray

源代码以官方发布的安装包，使用 MIT 协议授权。包括以下代码仓库中的源代码及安装包：

- v2ray/v2ray-core
- v2ray/ext

官方网站

官方网站（v2ray.com）以知识共享署名 4.0 国际许可协议协议授权。

- 包括网站中所有可见的文字内容和图片。
- 包括 Project V 图标文件。
- 包括生成网站所使用到的源代码，即v2ray/manual。

软件截图和其它文件

第三方所创作的内容，其版权归其创作者所有。Project V 放弃对这些内容的所有权。

- 包括 Project V 使用过程中的截图。
- 包括运行 Project V 所需的配置文件。
- 包括 Project V 运行时产生的日志文件。

其它内容

未在上述提及的内容，其版权视具体情况而定。

配置文件

V2Ray 本身使用基于 Protobuf 的配置。由于 Protobuf 的文本格式不方便阅读，V2Ray 同时也支持 JSON 格式的配置。在运行之前，V2Ray 会自动将 JSON 转换为对应的 Protobuf。换言之，V2Ray 将来也可能会支持其它格式的配置。

以下介绍一下基于 JSON 格式的配置。

JSON，全称 JavaScript Object Notation，简而言之是 Javascript 中的对象（Object）。一个 JSON 文件包含一个完整的对象，以大括号“{”开头，大括号“}”结束。

一个 JSON 对象包含一系列的键值对（Key-Value Pair），一个键是一个字符串（String），而值有多种类型，常见的有字符串（String）、数字（Number）、布尔（Bool）、数组（Array）和对象（Object）。下面是一个简单的 JSON 对象示例：

```
{
  "stringValue": "This is a string.",
  "numberValue": 42,
  "boolValue": true,
  "arrayValue": ["this", "is", "a", "string", "array"],
  "objectValue": {
    "another": "object"
  }
}
```

V2Ray 的 JSON 格式支持注释，可使用“//”或者“/* */”来进行注释。在不支持注释的编辑器中可能被显示为“错误”，但实际上是可以正常使用的。

JSON 数据类型

这里介绍一下常用的数据类型，在之后其它的配置中会用到。

`boolean`：true | false

布尔值，只有 `true` 和 `false` 两种取值，不带引号。

`number`

数字，在 V2Ray 的使用中通常为非负整数，即 `0`、`53` …… 数字在 JSON 格式中不带引号。

`string`

字符串，由引号包含的一串字符，如无特殊说明，字符的内容不限。

`array`：[]

数组，由方括号包含的一组元素，如字符串数组表示为 `[string]`。

`object`：{}

对象，一组键值对。样例见本文上方的示例。

通常一个键值对的后面需要有一个逗号","，但如果这个键值对后面紧跟一个大括号"}"的话，则一定不能有逗号。

V2Ray 常用数据类型

> `map` : object {string:string}

一组键值对，其类型在括号内指出。每一个键和值的类型对应相同。

> `address` : string

字符串，表示一个 IP 地址或域名，形如：`"8.8.8.8"` 或 `"www.v2ray.com"`

> `address_port` : string

字符串，表示一个地址和端口，常见的形式如：`"8.8.8.8:53"`，或者 `"www.v2ray.com:80"`。在一部分配置中，地址部分可以省略，如 `":443"`。

配置生成器和模板

V2Ray 项目目前尚没有官方维护模板合集或配置文件生成工具，如有需要，可以前往 神一样的工具们 章节查阅部分收录。

配置文件格式

V2Ray 的配置文件形式如下，客户端和服务器通用一种形式，只是实际的配置不一样。

```
{
  "log": {},
  "api": {},
  "dns": {},
  "stats": {},
  "routing": {},
  "policy": {},
  "reverse": {},
  "inbounds": [],
  "outbounds": [],
  "transport": {}
}
```

> `log` : LogObject

日志配置，表示 V2Ray 如何输出日志。

> `api` : ApiObject

内置的远程控置 API，详见远程控制配置。

> `dns` : DnsObject

内置的 DNS 服务器，若此项不存在，则默认使用本机的 DNS 设置。详见DNS 配置

> `routing` : RoutingObject

路由配置

> `policy` : PolicyObject

本地策略可进行一些权限相关的配置，详见本地策略

> `inbounds` : [InboundObject]

一个数组，每个元素是一个入站连接配置。

> `outbounds` : [OutboundObject]

一个数组，每个元素是一个出站连接配置。列表中的第一个元素作为主出站协议。当路由匹配不存在或没有匹配成功时，流量由主出站协议发出。

> `transport` : TransportObject

用于配置 V2Ray 如何与其它服务器建立和使用网络连接。详见底层传输配置

> `stats` : StatsObject

当此项存在时，开启统计信息。

> `reverse` : ReverseObject

反向代理配置。

LogObject

```
{
  "access": "文件地址",
```

```
  "error": "文件地址",
  "loglevel": "warning"
}
```

> `access`: string

访问日志的文件地址，其值是一个合法的文件地址，如 `"/tmp/v2ray/_access.log"`（Linux）或者 `"C:\\Temp\\v2ray\\_access.log"`（Windows）。当此项不指定或为空值时，表示将日志输出至 stdout。

> `error`: string

错误日志的文件地址，其值是一个合法的文件地址，如 `"/tmp/v2ray/_error.log"`（Linux）或者 `"C:\\Temp\\v2ray\\_error.log"`（Windows）。当此项不指定或为空值时，表示将日志输出至 stdout。

> `loglevel`: "debug" | "info" | "warning" | "error" | "none"

错误日志的级别。默认值为 `"warning"`。

- `"debug"`：只有开发人员能看懂的信息。同时包含所有 `"info"` 内容。
- `"info"`：V2Ray 在运行时的状态，不影响正常使用。同时包含所有 `"warning"` 内容。
- `"warning"`：V2Ray 遇到了一些问题，通常是外部问题，不影响 V2Ray 的正常运行，但有可能影响用户的体验。同时包含所有 `"error"` 内容。
- `"error"`：V2Ray 遇到了无法正常运行的问题，需要立即解决。
- `"none"`：不记录任何内容。

InboundObject

入站连接用于接收从客户端（浏览器或上一级代理服务器）发来的数据，可用的协议请见协议列表。

```
{
  "port": 1080,
  "listen": "127.0.0.1",
  "protocol": "协议名称",
  "settings": {},
  "streamSettings": {},
  "tag": "标识",
  "sniffing": {
    "enabled": false,
    "destOverride": ["http", "tls"]
  },
  "allocate": {
    "strategy": "always",
    "refresh": 5,
    "concurrency": 3
  }
}
```

> `port`: number | "env:variable" | string

端口。接受的格式如下:

- 整型数值: 实际的端口号。
- 环境变量: 以 `"env:"` 开头，后面是一个环境变量的名称，如 `"env:PORT"`。V2Ray 会以字符串形式解析这个环境变量。
- 字符串: 可以是一个数值类型的字符串，如 `"1234"`；或者一个数值范围，如 `"5-10"` 表示端口 5 到端口 10 这 6 个端口。

当只有一个端口时，V2Ray 会在此端口监听入站连接。当指定了一个端口范围时，取决于 `allocate` 设置。

> `listen`: address

监听地址，只允许 IP 地址，默认值为 `"0.0.0.0"`，表示接收所有网卡上的连接。除此之外，必须指定一个现有网卡的地址。

> `protocol`: string

连接协议名称，可选的值见协议列表。

> `settings`: InboundConfigurationObject

具体的配置内容，视协议不同而不同。详见每个协议中的 `InboundConfigurationObject`。

> `streamSettings`: StreamSettingsObject。

底层传输配置

> `tag`: string

此入站连接的标识，用于在其它的配置中定位此连接。当其不为空时，其值必须在所有 `tag` 中唯一。

> `sniffing`: SniffingObject

尝试探测流量的类型

> `allocate`: AllocateObject

端口分配设置

SniffingObject

```
{
  "enabled": false,
  "destOverride": ["http", "tls"]
}
```

> `enabled`: true | false

是否开启流量探测。

> `destOverride`: ["http" | "tls"]

当流量为指定类型时，按其中包括的目标地址重置当前连接的目标。

AllocateObject

```
{
  "strategy": "always",
  "refresh": 5,
  "concurrency": 3
}
```

> `strategy`: "always" | "random"

端口分配策略。`"always"` 表示总是分配所有已指定的端口，`port` 中指定了多少个端口，V2Ray 就会监听这些端口。`"random"` 表示随机开放端口，每隔 `refresh` 分钟在 `port` 范围中随机选取 `concurrency` 个端口来监听。

> `refresh`: number

随机端口刷新间隔，单位为分钟。最小值为 `2`，建议值为 `5`。这个属性仅当 `strategy = random` 时有效。

> `concurrency`: number

随机端口数量。最小值为 `1`，最大值为 `port` 范围的三分之一。建议值为 `3`。

OutboundObject

出站连接用于向远程网站或下一级代理服务器发送数据，可用的协议请见协议列表。

```
{
  "sendThrough": "0.0.0.0",
  "protocol": "协议名称",
  "settings": {},
  "tag": "标识",
  "streamSettings": {},
  "proxySettings": {
    "tag": "another-outbound-tag"
  },
  "mux": {}
}
```

> `sendThrough`: address

用于发送数据的 IP 地址，当主机有多个 IP 地址时有效，默认值为 `"0.0.0.0"`。

> `protocol`: string

连接协议名称，可选的值见协议列表。

> `settings`: OutboundConfigurationObject

具体的配置内容，视协议不同而不同。详见每个协议中的 `OutboundConfigurationObject`。

> `tag`: string

此出站连接的标识，用于在其它的配置中定位此连接。当其值不为空时，必须在所有 tag 中唯一。

> `streamSettings`: StreamSettingsObject。

底层传输配置

> `proxySettings`: ProxySettingsObject

出站代理配置。当出站代理生效时，此出站协议的 `streamSettings` 将不起作用。

> `mux`: MuxObject

Mux 配置。

ProxySettingsObject

```
{
  "tag": "another-outbound-tag"
}
```

> `tag`: string

当指定另一个出站协议的标识时，此出站协议发出的数据，将被转发至所指定的出站协议发出。

V2Ray 协议列表

V2Ray 支持以下协议：

- Blackhole
- Dokodemo-door
- Freedom
- HTTP
- MTProto
- Shadowsocks
- Socks
- VMess

Blackhole

- 名称: `blackhole`
- 类型: 出站协议

Blackhole（黑洞）是一个出站数据协议，它会阻碍所有数据的出站，配合路由（Routing）一起使用，可以达到禁止访问某些网站的效果。

OutboundConfigurationObject

```
{
  "response": {
    "type": "none"
  }
}
```

> `response`: ResponseObject

配置黑洞的响应数据。Blackhole 会在收到待转发数据之后，发送指定的响应数据，然后关闭连接。待转发的数据将被丢弃。如不指定此项，Blackhole 将直接关闭连接。

ResponseObject

```
{
  "type": "none"
}
```

> `type`: "http" | "none"

当 `type` 为 `"none"` （默认值）时，Blackhole将直接关闭连接。当 `type` 为 `"http"` 时，Blackhole会发回一个简单的 HTTP 403 数据包，然后关闭连接。

DNS

- 名称: `dns`
- 类型: 出站协议

DNS 是一个出站协议，主要用于拦截和转发 DNS 查询。此出站协议只能接收 DNS 流量（包含基于 UDP 和 TCP 协议的查询），其它类型的流量会导致错误。

在处理 DNS 查询时，此出站协议会将 IP 查询（即 A 和 AAAA）转发给内置的 DNS 服务器。其它类型的查询流量将被转发至它们原本的目标地址。

DNS 出站协议在 V2Ray 4.15 中引入。

OutboundConfigurationObject

```
{
    "network": "tcp",
    "address": "1.1.1.1",
    "port": 53
}
```

> `network`: "tcp" | "udp"

(V2Ray 4.16+) 修改 DNS 流量的传输层协议，可选的值有 `"tcp"` 和 `"udp"`。当不指定时，保持来源的传输方式不变。

> `address`: address

(V2Ray 4.16+) 修改 DNS 服务器地址。当不指定时，保持来源中指定的地址不变。

> `port`: number

(V2Ray 4.16+) 修改 DNS 服务器端口。当不指定时，保持来源中指定的端口不变。

Dokodemo-door

- 名称: `dokodemo-door`
- 类型: 入站协议

Dokodemo door（任意门）是一个入站数据协议，它可以监听一个本地端口，并把所有进入此端口的数据发送至指定服务器的一个端口，从而达到端口映射的效果。

InboundConfigurationObject

```
{
  "address": "8.8.8.8",
  "port": 53,
  "network": "tcp",
  "timeout": 0,
  "followRedirect": false,
  "userLevel": 0
}
```

> `address`: address

将流量转发到此地址。可以是一个 IP 地址，形如 `"1.2.3.4"`，或者一个域名，形如 `"v2ray.com"`。字符串类型。

当 `followRedirect`（见下文）为 `true` 时，`address` 可为空。

> `port`: number

将流量转发到目标地址的指定端口，范围[1, 65535]，数值类型。必填参数。

> `network`: "tcp" | "udp" | "tcp,udp"

可接收的网络协议类型。比如当指定为 `"tcp"` 时，任意门仅会接收 TCP 流量。默认值为 `"tcp"`。

> `timeout`: number

入站数据的时间限制（秒），默认值为 300。

V2Ray 3.1 后等价于对应用户等级的 `connIdle` 策略

> `followRedirect`: true | false

当值为 `true` 时，dokodemo-door 会识别出由 iptables 转发而来的数据，并转发到相应的目标地址。详见传输配置中的 `tproxy` 设置。

> `userLevel`: number

用户等级，所有连接都会使用这个用户等级。

透明代理配置样例

V2Ray 中增加一个 dokodemo-door 的入站协议：

```
{
  "network": "tcp,udp",
  "timeout": 30,
  "followRedirect": true
}
```

配置 iptables：

```
# Create new chain
iptables -t nat -N V2RAY
iptables -t mangle -N V2RAY
iptables -t mangle -N V2RAY_MARK

# Ignore your V2Ray server's addresses
# It's very IMPORTANT, just be careful.
iptables -t nat -A V2RAY -d 123.123.123.123 -j RETURN

# Ignore LANs and any other addresses you'd like to bypass the proxy
# See Wikipedia and RFC5735 for full list of reserved networks.
iptables -t nat -A V2RAY -d 0.0.0.0/8 -j RETURN
iptables -t nat -A V2RAY -d 10.0.0.0/8 -j RETURN
iptables -t nat -A V2RAY -d 127.0.0.0/8 -j RETURN
iptables -t nat -A V2RAY -d 169.254.0.0/16 -j RETURN
iptables -t nat -A V2RAY -d 172.16.0.0/12 -j RETURN
iptables -t nat -A V2RAY -d 192.168.0.0/16 -j RETURN
iptables -t nat -A V2RAY -d 224.0.0.0/4 -j RETURN
iptables -t nat -A V2RAY -d 240.0.0.0/4 -j RETURN

# Anything else should be redirected to Dokodemo-door's local port
iptables -t nat -A V2RAY -p tcp -j REDIRECT --to-ports 12345

# Add any UDP rules
ip route add local default dev lo table 100
ip rule add fwmark 1 lookup 100
iptables -t mangle -A V2RAY -p udp --dport 53 -j TPROXY --on-port 12345 --tproxy-mark 0x01/0x01
iptables -t mangle -A V2RAY_MARK -p udp --dport 53 -j MARK --set-mark 1

# Apply the rules
iptables -t nat -A OUTPUT -p tcp -j V2RAY
iptables -t mangle -A PREROUTING -j V2RAY
iptables -t mangle -A OUTPUT -j V2RAY_MARK
```

Freedom

- 名称：`freedom`
- 类型：出站协议

Freedom 是一个出站协议，可以用来向任意网络发送（正常的） TCP 或 UDP 数据。

OutboundConfigurationObject

```
{
  "domainStrategy": "AsIs",
  "redirect": "127.0.0.1:3366",
  "userLevel": 0
}
```

> `domainStrategy`: "AsIs" | "UseIP" | "UseIPv4" | "UseIPv6"

在目标地址为域名时，Freedom 可以直接向此域名发出连接（`"AsIs"`），或者将域名解析为 IP 之后再建立连接（`"UseIP"`、`"UseIPv4"`、`"UseIPv6"`）。解析 IP 的步骤会使用 V2Ray 内建的 DNS。默认值为`"AsIs"`。

(V2Ray 4.6+) 当使用`"UseIP"`模式，并且出站连接配置中指定了`sendThrough`时，Freedom 会根据`sendThrough`的值自动判断所需的IP类型，IPv4 或 IPv6。

(V2Ray 4.7+) 当使用`"UseIPv4"`或`"UseIPv6"`模式时，Freedom 会只使用对应的 IPv4 或 IPv6 地址。当`sendThrough`指定了不匹配的本地地址时，将导致连接失败。

> `redirect`: address_port

Freedom 会强制将所有数据发送到指定地址（而不是入站协议指定的地址）。其值为一个字符串，样例：`"127.0.0.1:80"`, `":1234"`。当地址不指定时，如`":443"`，Freedom 不会修改原先的目标地址。当端口为`0`时，如`"v2ray.com:0"`，Freedom 不会修改原先的端口。

> `userLevel`: number

用户等级，所有连接都使用这一等级。

HTTP

- 名称：`http`
- 类型：入站协议

HTTP 是一个入站数据协议，兼容 HTTP 1.x 代理。

InboundConfigurationObject

```
{
  "timeout": 0,
  "accounts": [
    {
      "user": "my-username",
      "pass": "my-password"
    }
  ],
  "allowTransparent": false,
  "userLevel": 0
}
```

> `timeout` : number

从客户端读取数据的超时设置（秒），0 表示不限时。默认值为 300。 V2Ray 3.1 后等价于对应用户等级的 `connIdle` 策略。

> `accounts` : [AccountObject]

一个数组，数组中每个元素为一个用户帐号。默认值为空。

当 `accounts` 非空时，HTTP 代理将对入站连接进行 Basic Authentication 验证。

> `allowTransparent` : true | false

当为 `true` 时，会转发所有 HTTP 请求，而非只是代理请求。若配置不当，开启此选项会导致死循环。

> `userLevel` : number

用户等级，所有连接使用这一等级。

AccountObject

```
{
  "user": "my-username",
  "pass": "my-password"
}
```

> `user` : string

用户名，字符串类型。必填。

> `pass` : string

密码，字符串类型。必填。

在 Linux 中使用以下环境变量即可在当前 session 使用全局 HTTP 代理（很多软件都支持这一设置，也有不支持的）。

- `export http_proxy=http://127.0.0.1:8080/` (地址须改成你配置的 HTTP 入站代理地址)
- `export https_proxy=$http_proxy`

MTProto

- 名称: `mtproto`
- 类型: 入站 / 出站

MTProto 是一个 Telegram 专用的代理协议。在 V2Ray 中可使用一组入站出站代理来完成 Telegram 数据的代理任务。

目前只支持转发到 Telegram 的 IPv4 地址。

InboundConfigurationObject

```
{
  "users": [{
    "email": "love@v2ray.com",
    "level": 0,
    "secret": "b0cbcef5a486d9636472ac27f8e11a9d"
  }]
}
```

> `users`: [UserObject]

一个数组，其中每一个元素表示一个用户。目前只有第一个用户会生效。

UserObject

```
{
  "email": "love@v2ray.com",
  "level": 0,
  "secret": "b0cbcef5a486d9636472ac27f8e11a9d"
}
```

> `email`: string

用户邮箱，用于统计流量等辅助功能

> `level`: number

用户等级。

> `secret`: string

用户密钥。必须为 32 个字符，仅可包含 `0` 到 `9` 和 `a` 到 `f` 之间的字符。

使用此命令生成 MTProto 代理所需要的用户密钥: `openssl rand -hex 16`

OutboundConfigurationObject

```
{
}
```

样例配置

MTProto 仅可用于 Telegram 数据。你可能需要一个路由来绑定对应的入站出站代理。以下是一个不完整的示例：

入站代理：

```
{
  "tag": "tg-in",
  "port": 443,
  "protocol": "mtproto",
  "settings": {
    "users": [{"secret": "b0cbcef5a486d9636472ac27f8e11a9d"}]
  }
}
```

出站代理：

```
{
  "tag": "tg-out",
  "protocol": "mtproto",
  "settings": {}
}
```

路由：

```
{
  "type": "field",
  "inboundTag": ["tg-in"],
  "outboundTag": "tg-out"
}
```

然后使用 Telegram 连接这台机器的 443 端口即可。

Shadowsocks

- 名称：`shadowsocks`
- 类型：入站 / 出站

Shadowsocks 协议，包含入站和出站两部分，兼容大部分其它版本的实现。

与官方版本的兼容性：

- 支持 TCP 和 UDP 数据包转发，其中 UDP 可选择性关闭；
- 支持 OTA；
  - 客户端可选开启或关闭；
  - 服务器端可强制开启、关闭或自适应；
- 加密方式（其中 AEAD 加密方式在 V2Ray 3.0 中加入）：
  - aes-256-cfb
  - aes-128-cfb
  - chacha20
  - chacha20-ietf
  - aes-256-gcm
  - aes-128-gcm
  - chacha20-poly1305 或称 chacha20-ietf-poly1305
- 插件：
  - 通过 Standalone 模式支持 obfs

Shadowsocks 的配置分为两部分，`InboundConfigurationObject` 和 `OutboundConfigurationObject`，分别对应入站和出站协议配置中的 `settings` 项。

InboundConfigurationObject

```
{
  "email": "love@v2ray.com",
  "method": "aes-128-cfb",
  "password": "密码",
  "level": 0,
  "ota": true,
  "network": "tcp"
}
```

> `email`: string

邮件地址，可选，用于标识用户

> `method`: string

必填。可选的值见加密方式列表

> `password`: string

必填，任意字符串。Shadowsocks 协议不限制密码长度，但短密码会更可能被破解，建议使用 16 字符或更长的密码。

> `level`: number

用户等级，默认值为 `0`。详见本地策略。

> `ota`: true | false

是否强制 OTA，如果不指定此项，则自动判断。强制开启 OTA 后，V2Ray 会拒绝未启用 OTA 的连接。反之亦然。

当使用 AEAD 时，`ota` 设置无效

> `network`: "tcp" | "udp" | "tcp,udp"

可接收的网络连接类型，默认值为`"tcp"`。

OutboundConfigurationObject

```
{
  "servers": [
    {
      "email": "love@v2ray.com",
      "address": "127.0.0.1",
      "port": 1234,
      "method": "加密方式",
      "password": "密码",
      "ota": false,
      "level": 0
    }
  ]
}
```

> `servers`: [ServerObject]

一个数组，其中每一项是一个 ServerObject。

ServerObject

```
{
  "email": "love@v2ray.com",
  "address": "127.0.0.1",
  "port": 1234,
  "method": "加密方式",
  "password": "密码",
  "ota": false,
  "level": 0
}
```

> `email`: string

邮件地址，可选，用于标识用户

> `address`: address

Shadowsocks 服务器地址，支持 IPv4、IPv6 和域名。必填。

> `port`: number

Shadowsocks 服务器端口。必填。

> `method`: string

必填。可选的值见加密方式列表

> `password`: string

必填。任意字符串。Shadowsocks 协议不限制密码长度，但短密码会更可能被破解，建议使用 16 字符或更长的密码。

> `ota`: true | false

是否开启 Shadowsocks 的一次验证（One time auth），默认值为`false`。

当使用 AEAD 时，`ota` 设置无效。

> `level` : number

用户等级

加密方式列表

- `"aes-256-cfb"`
- `"aes-128-cfb"`
- `"chacha20"`
- `"chacha20-ietf"`
- `"aes-256-gcm"`
- `"aes-128-gcm"`
- `"chacha20-poly1305"` 或 `"chacha20-ietf-poly1305"`

Socks

- 名称：`socks`
- 类型：入站 / 出站

标准 Socks 协议实现，兼容 Socks 4、Socks 4a 和 Socks 5。

Socks 的配置分为两部分，`InboundConfigurationObject` 和 `OutboundConfigurationObject`，分别对应入站和出站协议配置中的 `settings` 项。

OutboundConfigurationObject

```
{
  "servers": [{
    "address": "127.0.0.1",
    "port": 1234,
    "users": [
      {
        "user": "test user",
        "pass": "test pass",
        "level": 0
      }
    ]
  }]
}
```

> `servers`: [ServerObject]

Socks 服务器列表，其中每一项是一个服务器配置。

ServerObject

```
{
  "address": "127.0.0.1",
  "port": 1234,
  "users": [
    {
      "user": "test user",
      "pass": "test pass",
      "level": 0
    }
  ]
}
```

> `address`: address

服务器地址。

> 仅支持连接到 Socks 5 服务器。

> `port`: number

服务器端口

> `users`: [UserObject]

用户列表，其中每一项一个用户配置。当列表不为空时，Socks 客户端会使用此用户信息进行认证；如未指定，则不进行认证。

UserObject

```
{
  "user": "test user",
  "pass": "test pass",
  "level": 0
}
```

> `user` : string

用户名

> `pass` : string

密码

> `level` : number

用户等级

InboundConfigurationObject

```
{
  "auth": "noauth",
  "accounts": [
    {
      "user": "my-username",
      "pass": "my-password"
    }
  ],
  "udp": false,
  "ip": "127.0.0.1",
  "userLevel": 0
}
```

> `auth` : "noauth" | "password"

Socks 协议的认证方式，支持 `"noauth"` 匿名方式和 `"password"` 用户密码方式。默认值为 `"noauth"` 。

> `accounts` : [AccountObject]

一个数组，数组中每个元素为一个用户帐号。默认值为空。此选项仅当 `auth` 为 `password` 时有效。

> `udp` : true | false

是否开启 UDP 协议的支持。默认值为 `false` 。

> `ip` : address

当开启 UDP 时，V2Ray 需要知道本机的 IP 地址。默认值为 `"127.0.0.1"` 。

> `userLevel` : number

用户等级，所有连接使用这一等级。

AccountObject

```
{
  "user": "my-username",
```

```
    "pass": "my-password"
  }
```

> `user` : string

用户名

> `pass` : string

密码

VMess

- 名称：`vmess`
- 类型：入站 / 出站

VMess 是一个加密传输协议，它分为入站和出站两部分，通常作为 V2Ray 客户端和服务器之间的桥梁。

VMess 依赖于系统时间，请确保使用 V2Ray 的系统 UTC 时间误差在 90 秒之内，时区无关。在 Linux 系统中可以安装 `ntp` 服务来自动同步系统时间。

VMess 的配置分为两部分，`InboundConfigurationObject` 和 `OutboundConfigurationObject`，分别对应入站和出站协议配置中的 `settings` 项。

OutboundConfigurationObject

```
{
  "vnext": [
    {
      "address": "127.0.0.1",
      "port": 37192,
      "users": [
        {
          "id": "27848739-7e62-4138-9fd3-098a63964b6b",
          "alterId": 4,
          "security": "auto",
          "level": 0
        }
      ]
    }
  ]
}
```

> `vnext`: [ServerObject]

一个数组，包含一系列的服务器配置

ServerObject

```
{
  "address": "127.0.0.1",
  "port": 37192,
  "users": []
}
```

> `address`: address

服务器地址，支持 IP 地址或者域名。

> `port`: number

服务器端口号。

> `users`: [UserObject]

一组服务器认可的用户

UserObject

```
{
```

```
  "id": "27848739-7e62-4138-9fd3-098a63964b6b",
  "alterId": 4,
  "security": "auto",
  "level": 0
}
```

> `id`: string

VMess 用户的主 ID。必须是一个合法的 UUID。

> `alterId`: number

为了进一步防止被探测，一个用户可以在主 ID 的基础上，再额外生成多个 ID。这里只需要指定额外的 ID 的数量，推荐值为 4。不指定的话，默认值是 `0`。最大值 `65535`。这个值不能超过服务器端所指定的值。

> `level`: number

用户等级

> `security`: "aes-128-gcm" | "chacha20-poly1305" | "auto" | "none"

加密方式，客户端将使用配置的加密方式发送数据，服务器端自动识别，无需配置。

- `"aes-128-gcm"`：推荐在 PC 上使用
- `"chacha20-poly1305"`：推荐在手机端使用
- `"auto"`：默认值，自动选择（运行框架为 AMD64、ARM64 或 s390x 时为aes-128-gcm加密方式，其他情况则为 Chacha20-Poly1305 加密方式）
- `"none"`：不加密

推荐使用 `"auto"` 加密方式，这样可以永久保证安全性和兼容性。

InboundConfigurationObject

```
{
  "clients": [
    {
      "id": "27848739-7e62-4138-9fd3-098a63964b6b",
      "level": 0,
      "alterId": 4,
      "email": "love@v2ray.com"
    }
  ],
  "default": {
    "level": 0,
    "alterId": 4
  },
  "detour": {
    "to": "tag_to_detour"
  },
  "disableInsecureEncryption": false
}
```

> `clients`: [ClientObject]

一组服务器认可的用户。clients 可以为空。当此配置用作动态端口时，V2Ray 会自动创建用户。

> `detour`: DetourObject

指示对应的出站协议使用另一个服务器。

> `default`: DefaultObject

可选，clients 的默认配置。仅在配合 `detour` 时有效。

> `disableInsecureEncryption`: true | false

是否禁止客户端使用不安全的加密方式，当客户端指定下列加密方式时，服务器会主动断开连接。默认值为 `false`。

- `"none"`
- `"aes-128-cfb"`

ClientObject

```
{
  "id": "27848739-7e62-4138-9fd3-098a63964b6b",
  "level": 0,
  "alterId": 4,
  "email": "love@v2ray.com"
}
```

> `id`: string

VMess 的用户 ID。必须是一个合法的 UUID。

> `level`: number

用户等级，详见本地策略

> `alterId`: number

与上文出站协议中的含义相同。

> `email`: string

用户邮箱地址，用于区分不同用户的流量。

`alterId` 取值的大小和流量特征没有必然联系。对于日常使用，`16` 以内的值已经够用了。

DetourObject

```
{
  "to": "tag_to_detour"
}
```

> `to`: string

一个入站协议的 `tag`，详见配置文件。指定的入站协议必须是一个 VMess

DefaultObject

```
{
  "level": 0,
  "alterId": 4
}
```

> `level`: number

用户等级，意义同上。默认值为 `0`。

> `alterId`: number

和 `ClientObject` 中的 `alterId` 相同，默认值为 `64` 。推荐值 `4` 。

本地策略

本地策略可以配置一些用户相关的权限，比如连接超时设置。V2Ray 处理的每一个连接，都对应到一个用户，按照这个用户的等级（level）应用不同的策略。本地策略可按照等级的不同而变化。

PolicyObject

`PolicyObject` 对应配置文件中的 `policy` 项。

```
{
  "levels": {
    "0": {
      "handshake": 4,
      "connIdle": 300,
      "uplinkOnly": 2,
      "downlinkOnly": 5,
      "statsUserUplink": false,
      "statsUserDownlink": false,
      "bufferSize": 10240
    }
  },
  "system": {
    "statsInboundUplink": false,
    "statsInboundDownlink": false
  }
}
```

> `level` : map{string: LevelPolicyObject}

一组键值对，每个键是一个字符串形式的数字（JSON 的要求），比如 `"0"` 、`"1"` 等，双引号不能省略，这个数字对应用户等级。每一个值是一个 LevelPolicyObject.

> 每个入站出站代理现在都可以设置用户等级，V2Ray 会根据实际的用户等级应用不同的本地策略。

> `system` : SystemPolicyObject

V2Ray 系统的策略

LevelPolicyObject

```
{
  "handshake": 4,
  "connIdle": 300,
  "uplinkOnly": 2,
  "downlinkOnly": 5,
  "statsUserUplink": false,
  "statsUserDownlink": false,
  "bufferSize": 10240
}
```

> `handshake` : number

连接建立时的握手时间限制。单位为秒。默认值为 `4` 。在入站代理处理一个新连接时，在握手阶段（比如 VMess 读取头部数据，判断目标服务器地址），如果使用的时间超过这个时间，则中断该连接。

> `connIdle` : number

连接空闲的时间限制。单位为秒。默认值为 `300`。在入站出站代理处理一个连接时，如果在 `connIdle` 时间内，没有任何数据被传输（包括上行和下行数据），则中断该连接。

> `uplinkOnly`: number

当连接下行线路关闭后的时间限制。单位为秒。默认值为 `2`。当服务器（如远端网站）关闭下行连接时，出站代理会在等待 `uplinkOnly` 时间后中断连接。

> `downlinkOnly`: number

当连接上行线路关闭后的时间限制。单位为秒。默认值为 `5`。当客户端（如浏览器）关闭上行连接时，入站代理会在等待 `downlinkOnly` 时间后中断连接。

在 HTTP 浏览的场景中，可以将 `uplinkOnly` 和 `downlinkOnly` 设为 `0`，以提高连接关闭的效率。

> `statsUserUplink`: true | false

当值为 `true` 时，开启当前等级的所有用户的上行流量统计。

> `statsUserDownlink`: true | false

当值为 `true` 时，开启当前等级的所有用户的下行流量统计。

> `bufferSize`: number

每个连接的内部缓存大小。单位为 kB。当值为 `0` 时，内部缓存被禁用。

默认值 (V2Ray 4.4+):

- 在 ARM、MIPS、MIPSLE 平台上，默认值为 `0`。
- 在 ARM64、MIPS64、MIPS64LE 平台上，默认值为 `4`。
- 在其它平台上，默认值为 `512`。

默认值 (V2Ray 4.3-):

- 在 ARM、MIPS、MIPSLE、ARM64、MIPS64、MIPS64LE 平台上，默认值为 `16`。
- 在其它平台上，默认值为 `2048`。

`bufferSize` 选项会覆盖环境变量中 `v2ray.ray.buffer.size` 的设定。

SystemPolicyObject

```
{
  "statsInboundUplink": false,
  "statsInboundDownlink": false
}
```

> `statsInboundUplink`: true | false

当值为 `true` 时，开启所有入站代理的上行流量统计。

> `statsInboundDownlink`: true | false

当值为 `true` 时，开启所有入站代理的下行流量统计。

路由功能

V2Ray 内建了一个简单的路由功能，可以将入站数据按需求由不同的出站连接发出，以达到按需代理的目的。这一功能的常见用法是分流国内外流量，V2Ray 可以通过内部机制判断不同地区的流量，然后将它们发送到不同的出站代理。

RoutingObject

`RoutingObject` 对应主配置文件中的 `routing` 项。

```
{
  "domainStrategy": "AsIs",
  "rules": [],
  "balancers": []
}
```

> `domainStrategy`: "AsIs" | "IPIfNonMatch" | "IPOnDemand"

域名解析策略，根据不同的设置使用不同的策略。

- `"AsIs"`：只使用域名进行路由选择。默认值。
- `"IPIfNonMatch"`：当域名没有匹配任何规则时，将域名解析成 IP（A 记录或 AAAA 记录）再次进行匹配；
  - 当一个域名有多个 A 记录时，会尝试匹配所有的 A 记录，直到其中一个与某个规则匹配为止；
  - 解析后的 IP 仅在路由选择时起作用，转发的数据包中依然使用原始域名；
- `"IPOnDemand"`：当匹配时碰到任何基于 IP 的规则，将域名立即解析为 IP 进行匹配；

> `rules`: [RuleObject]

对应一个数组，数组中每个元素是一个规则。对于每一个连接，路由将根据这些规则依次进行判断，当一个规则生效时，即将这个连接转发至它所指定的 `outboundTag` (或 `balancerTag`，V2Ray 4.4+)。当没有匹配到任何规则时，流量默认由主出站协议发出。

> `balancers`: [BalancerObject]

(V2Ray 4.4+)一个数组，数组中每个元素是一个负载均衡器的配置。当一个规则指向一个负载均衡器时，V2Ray 会通过此负载均衡器选出一个出站协议，然后由它转发流量。

RuleObject

```
{
  "type": "field",
  "domain": [
    "baidu.com",
    "qq.com",
    "geosite:cn"
  ],
  "ip": [
    "0.0.0.0/8",
    "10.0.0.0/8",
    "fc00::/7",
    "fe80::/10",
    "geoip:cn"
  ],
  "port": "53,443,1000-2000",
  "network": "tcp",
  "source": [
    "10.0.0.1"
  ],
  "user": [
    "love@v2ray.com"
  ],
```

```
  "inboundTag": [
    "tag-vmess"
  ],
  "protocol":["http", "tls", "bittorrent"],
  "attrs": "attrs[':method'] == 'GET'",
  "outboundTag": "direct",
  "balancerTag": "balancer"
}
```

> 当多个属性同时指定时，这些属性需要同时满足，才可以使当前规则生效。如果多个规则分别使用了 `domain` 或者 `ip`，需要对应添加多条规则。

> `type`: "field"

目前只支持 `"field"` 这一个选项。

> `domain`: [string]

一个数组，数组每一项是一个域名的匹配。有以下几种形式:

- 纯字符串: 当此字符串匹配目标域名中任意部分，该规则生效。比如"sina.com"可以匹配"sina.com"、"sina.com.cn"和"www.sina.com"，但不匹配"sina.cn"。
- 正则表达式: 由 `"regexp:"` 开始，余下部分是一个正则表达式。当此正则表达式匹配目标域名时，该规则生效。例如"regexp:\\.goo.*\\.com$"匹配"www.google.com"、"fonts.googleapis.com"，但不匹配"google.com"。
- 子域名 (推荐): 由 `"domain:"` 开始，余下部分是一个域名。当此域名是目标域名或其子域名时，该规则生效。例如"domain:v2ray.com"匹配"www.v2ray.com"、"v2ray.com"，但不匹配"xv2ray.com"。
- 完整匹配: 由 `"full:"` 开始，余下部分是一个域名。当此域名完整匹配目标域名时，该规则生效。例如"full:v2ray.com"匹配"v2ray.com"但不匹配"www.v2ray.com"。
- 预定义域名列表：由 `"geosite:"` 开头，余下部分是一个名称，如 `geosite:google` 或者 `geosite:cn`。名称及域名列表参考预定义域名列表。
- 从文件中加载域名: 形如 `"ext:file:tag"`，必须以 `ext:` （小写）开头，后面跟文件名和标签，文件存放在资源目录中，文件格式与 `geosite.dat` 相同，标签必须在文件中存在。

> `ip`: [string]

一个数组，数组内每一个元素代表一个 IP 范围。当某一元素匹配目标 IP 时，此规则生效。有以下几种形式:

- IP: 形如 `"127.0.0.1"`。
- CIDR: 形如 `"10.0.0.0/8"`.
- GeoIP: 形如 `"geoip:cn"`，必须以 `geoip:` （小写）开头，后面跟双字符国家代码，支持几乎所有可以上网的国家。
  - 特殊值： `"geoip:private"` (V2Ray 3.5+)，包含所有私有地址，如 `127.0.0.1`。
- 从文件中加载 IP: 形如 `"ext:file:tag"`，必须以 `ext:` （小写）开头，后面跟文件名和标签，文件存放在资源目录中，文件格式与 `geoip.dat` 相同标签必须在文件中存在。

> `"ext:geoip.dat:cn"` 等价于 `"geoip:cn"`

> `port`: number | string

端口范围，有三种形式:

- `"a-b"`: a 和 b 均为正整数，且小于 65536。这个范围是一个前后闭合区间，当目标端口落在此范围内时，此规则生效。
- `a`: a 为正整数，且小于 65536。当目标端口为 a 时，此规则生效。

- (V2Ray 4.18+) 以上两种形式的混合，以逗号","分隔。形如：`"53,443,1000-2000"`。

> `network`: "tcp" | "udp" | "tcp,udp"

可选的值有"tcp"、"udp"或"tcp,udp"，当连接方式是指定的方式时，此规则生效。

> `source`: [string]

一个数组，数组内每一个元素是一个 IP 或 CIDR。当某一元素匹配来源 IP 时，此规则生效。

> `user`: [string]

一个数组，数组内每一个元素是一个邮箱地址。当某一元素匹配来源用户时，此规则生效。当前 Shadowsocks 和 VMess 支持此规则。

> `inboundTag`: [string]

一个数组，数组内每一个元素是一个标识。当某一元素匹配入站协议的标识时，此规则生效。

> `protocol`: ["http" | "tls" | "bittorrent"]

一个数组，数组内每一个元素表示一种协议。当某一个协议匹配当前连接的流量时，此规则生效。必须开启入站代理中的 `sniffing` 选项。

> `attrs`: string

(V2Ray 4.18+) 一段脚本，用于检测流量的属性值。当此脚本返回真值时，此规则生效。

脚本语言为 Starlark，它的语法是 Python 的子集。脚本接受一个全局变量 `attrs`，其中包含了流量相关的属性。

目前只有 http 入站代理会设置这一属性。

示例：

- 检测 HTTP GET: `"attrs[':method'] == 'GET'"`
- 检测 HTTP Path: `"attrs[':path'].startswith('/test')"`
- 检测 Content Type: `"attrs['accept'].index('text/html') >= 0"`

> `outboundTag`: string

对应一个额外出站连接配置的标识。

> `balancerTag`: string

对应一个负载均衡器的标识。`balancerTag` 和 `outboundTag` 须二选一。当同时指定时，`outboundTag` 生效。

BalancerObject

负载均衡器配置。当一个负载均衡器生效时，它会从指定的出站协议中，按配置选出一个最合适的出站协议，进行流量转发。

```
{
  "tag": "balancer",
  "selector": []
}
```

> `tag`: string

此负载均衡器的标识，用于匹配 `RuleObject` 中的 `balancerTag`。

> `selector`: [string]

一个字符串数组，其中每一个字符串将用于和出站协议标识的前缀匹配。在以下几个出站协议标识中：`[ "a", "ab", "c", "ba" ]`，`"selector": ["a"]` 将匹配到 `[ "a", "ab" ]`。

如果匹配到多个出站协议，负载均衡器目前会从中随机选出一个作为最终的出站协议。

预定义域名列表

此列表由 domain-list-community 项目维护，预置于每一个 V2Ray 的安装包中，文件名为 `geosite.dat`。

这个文件包含了一些常见的域名，可用于路由和 DNS 筛选。常用的域名有：

- `category-ads`：包含了常见的广告域名。
- `category-ads-all`：包含了常见的广告域名，以及广告提供商的域名。
- `cn`：相当于 `geolocation-cn` 和 `tld-cn` 的合集。
- `google`：包含了 Google 旗下的所有域名。
- `facebook`：包含了 Facebook 旗下的所有域名。
- `geolocation-cn`：包含了常见的国内站点的域名。
- `geolocation-!cn`：包含了常见的非国内站点的域名。
- `speedtest`：包含了所有 Speedtest 所用的域名。
- `tld-cn`：包含了所有 .cn 和 .中国 结尾的域名。

DNS 服务器

V2Ray 内置了一个 DNS 服务器，可以将 DNS 查询根据路由设置转发到不同的远程服务器中。由此 DNS 服务器所发出的 DNS 查询请求，会自动根据路由配置进行转发，无需额外配置。

> 由于 DNS 协议的复杂性，V2Ray 只支持最基本的 IP 查询（A 和 AAAA 记录）。推荐使用本机 DNS 配合一个额外的 DNS 服务器来做 DNS 查询，如 CoreDNS，以使用完整的 DNS 功能。

DnsObject

`DnsObject` 对应配置文件中的 `dns` 项。

```
{
  "hosts": {
    "baidu.com": "127.0.0.1"
  },
  "servers": [
    {
      "address": "1.2.3.4",
      "port": 5353,
      "domains": [
        "domain:v2ray.com"
      ]
    },
    "8.8.8.8",
    "8.8.4.4",
    "localhost"
  ],
  "clientIp": "1.2.3.4",
  "tag": "dns_inbound"
}
```

> `hosts` : map{string: address}

静态 IP 列表，其值为一系列的"域名":"地址"。其中地址可以是 IP 或者域名。在解析域名时，如果域名匹配这个列表中的某一项，当该项的地址为 IP 时，则解析结果为该项的 IP，而不会使用下述的 servers 进行解析；当该项的地址为域名时，会使用此域名进行 IP 解析，而不使用原始域名。

域名的格式有以下几种形式:

- 纯字符串: 当此字符串匹配目标域名中任意部分，该规则生效。当此域名完整匹配目标域名时，该规则生效。例如"v2ray.com"匹配"v2ray.com"但不匹配"www.v2ray.com"。
- 正则表达式: 由 `"regexp:"` 开始，余下部分是一个正则表达式。当此正则表达式匹配目标域名时，该规则生效。例如"regexp:\\.goo.*\\.com$"匹配"www.google.com"、"fonts.googleapis.com"，但不匹配"google.com"。
- 子域名 (推荐): 由 `"domain:"` 开始，余下部分是一个域名。当此域名是目标域名或其子域名时，该规则生效。例如"domain:v2ray.com"匹配"www.v2ray.com"、"v2ray.com"，但不匹配"xv2ray.com"。
- 子串: 当此字符串匹配目标域名中任意部分，该规则生效。比如"keyword:sina.com"可以匹配"sina.com"、"sina.com.cn"和"www.sina.com"，但不匹配"sina.cn"。
- 预定义域名列表：由 `"geosite:"` 开头，余下部分是一个名称，如 `geosite:google` 或者 `geosite:cn`。名称及域名列表参考预定义域名列表。

> `servers` : [string | ServerObject | "localhost"]

一个 DNS 服务器列表，支持的类型有三种：IP 地址（字符串形式），ServerObject，或者 `"localhost"`。

当它的值是一个 IP 地址时，如 `"8.8.8.8"`，V2Ray 会使用此地址的 53 端口进行 DNS 查询。

当值为 `"localhost"` 时，表示使用本机预设的 DNS 配置。

> 当使用 localhost 时，本机的 DNS 请求不受 V2Ray 控制，需要额外的配置才可以使 DNS 请求由 V2Ray 转发。

> `clientIp` : string

当前系统的 IP 地址，用于 DNS 查询时，通知服务器客户端的所在位置。不能是私有地址。

> `tag` : string

(V2Ray 4.13+) 由此 DNS 发出的查询流量，除 `localhost` 外，都会带有此标识，可在路由使用 `inboundTag` 进行匹配。

ServerObject

```
{
  "address": "1.2.3.4",
  "port": 5353,
  "domains": [
    "domain:v2ray.com"
  ]
}
```

> `address` : address

DNS 服务器地址，如 `"8.8.8.8"` 。目前只支持 UDP 协议的 DNS 服务器。

> `port` : number

DNS 服务器端口，如 `53` 。

> `domains` : [string]

一个域名列表，此列表包含的域名，将优先使用此服务器进行查询。域名格式和路由配置中相同。

若要使 DNS 服务生效，需要配置路由功能中的 `domainStrategy` 。

当某个 DNS 服务器指定的域名列表匹配了当前要查询的域名，V2Ray 会优先使用这个 DNS 服务器进行查询，否则按从上往下的顺序进行查询。

Mux 多路复用

Mux 功能是在一条 TCP 连接上分发多个 TCP 连接的数据。实现细节详见Mux.Cool。Mux 是为了减少 TCP 的握手延迟而设计，而非提高连接的吞吐量。使用 Mux 看视频、下载或者测速通常都有反效果。Mux 只需要在客户端启用，服务器端自动适配。

MuxObject

`MuxObject` 对应 `OutboundObject` 中的 `mux` 项。

```
{
  "enabled": false,
  "concurrency": 8
}
```

> `enabled` : true | false

是否启用 Mux

> `concurrency` : number

最大并发连接数。最小值 `1`，最大值 `1024`，默认值 `8`。这个数值表示了一个 TCP 连接上最多承载的 Mux 连接数量。当客户端发出了 8 个 TCP 请求，而 `concurrency=8` 时，V2Ray 只会发出一条实际的 TCP 连接，客户端的 8 个请求全部由这个 TCP 连接传输。

远程控制

V2Ray 中可以开放一些 API 以便远程调用。这些 API 都基于 gRPC。

当远程控制开启时，V2Ray 会自建一个出站代理，以 `tag` 配置的值为标识。用户必须手动将所有的 gRPC 入站连接通过路由指向这一出站代理。

ApiObject

`ApiObject` 对应配置文件中的 `api` 项。

```
{
  "tag": "api",
  "services": [
    "HandlerService",
    "LoggerService",
    "StatsService"
  ]
}
```

> `tag` : string

出站代理标识

> `services` : [string]

开启的 API 列表，可选的值见API 列表。

支持的 API 列表

HandlerService

一些对于入站出站代理进行修改的 API，可用的功能如下：

- 添加一个新的入站代理；
- 添加一个新的出站代理；
- 删除一个现有的入站代理；
- 删除一个现有的出站代理；
- 在一个入站代理中添加一个用户（仅支持 VMess）；
- 在一个入站代理中删除一个用户（仅支持 VMess）；

LoggerService

支持对内置 Logger 的重启，可配合 logrotate 进行一些对日志文件的操作。

StatsService

内置的数据统计服务，详见统计信息。

统计信息

V2Ray 提供了一些关于其运行状况的统计信息。

StatsObject

`StatsObject` 对应配置文件中的 `stats` 项。

```
{
}
```

目前统计信息没有任何参数，只要 `StatsObject` 项存在，内部的统计即会开启。同时你还需要在 Policy 中开启对应的项，才可以统计对应的数据。

目前已有的统计信息如下：

用户数据

> `user>>>[email]>>>traffic>>>uplink`

特定用户的上行流量，单位字节。

> `user>>>[email]>>>traffic>>>downlink`

特定用户的下行流量，单位字节。

> 如果对应用户没有指定 Email，则不会开启统计。

全局数据

> `inbound>>>[tag]>>>traffic>>>uplink`

特定入站代理的上行流量，单位字节。

> `inbound>>>[tag]>>>traffic>>>downlink`

特定入站代理的下行流量，单位字节。

反向代理

反向代理是一个 V2Ray 的附加功能，可以把服务器端的流量向客户端转发，即逆向流量转发。

> 反向代理功能在 V2Ray 4.0+ 可用。目前处于测试阶段，可能会有一些问题。

反向代理的大致工作原理如下:

- 假设在主机 A 中有一个网页服务器，这台主机没有公网 IP，无法在公网上直接访问。另有一台主机 B，它可以由公网访问。现在我们需要把 B 作为入口，把流量从 B 转发到 A。
- 在主机 A 中配置一个 V2Ray，称为 `bridge`，在 B 中也配置一个 V2Ray，称为 `portal`。
- `bridge` 会向 `portal` 主动建立连接，此连接的目标地址可以自行设定。`portal` 会收到两种连接，一是由 `bridge` 发来的连接，二是公网用户发来的连接。`portal` 会自动将两类连接合并。于是 `bridge` 就可以收到公网流量了。
- `bridge` 在收到公网流量之后，会将其原封不动地发给主机 A 中的网页服务器。当然，这一步需要路由的协作。
- `bridge` 会根据流量的大小进行动态的负载均衡。

> 反向代理默认已开启 Mux，请不要在其用到的出站代理上再次开启 Mux。

ReverseObject

`ReverseObject` 对应配置文件中的 `reverse` 项。

```
{
  "bridges": [{
    "tag": "bridge",
    "domain": "test.v2ray.com"
  }],
  "portals": [{
    "tag": "portal",
    "domain": "test.v2ray.com"
  }]
}
```

> `bridges`: [BridgeObject]

一个数组，每一项表示一个 `bridge`。每个 `bridge` 的配置是一个 BridgeObject。

> `portals`: [PortalObject]

一个数组，每一项表示一个 `portal`。每个 `portal` 的配置是一个 PortalObject。

BridgeObject

```
{
  "tag": "bridge",
  "domain": "test.v2ray.com"
}
```

> `tag`: string

一个标识，所有由 `bridge` 发出的连接，都会带有这个标识。可以在路由中使用 `inboundTag` 进行识别。

> `domain` : string

一个域名。`bridge` 向 `portal` 建立的连接，都会使用这个域名进行发送。这个域名只作为 `bridge` 和 `portal` 的通信用途，不必真实存在。

PortalObject

> `tag` : string

`portal` 的标识。在路由中使用 `outboundTag` 将流量转发到这个 `portal` 。

> `domain` : string

一个域名。当 `portal` 接收到流量时，如果流量的目标域名是此域名，则 `portal` 认为当前连接上 `bridge` 发来的通信连接。而其它流量则会被当成需要转发的流量。`portal` 所做的工作就是把这两类连接进行识别并拼接。

和其它配置一样，一个 V2Ray 既可以作为 `bridge`，也可以作为 `portal`，也可以同时两者，以适用于不同的场景需要。

完整配置

`bridge` 通常需要两个出站代理，一个用于连接 `portal`，另一个用于发送实际的流量。也就是说，你需要用路由区分两种流量。

反向代理配置:

```
{
  "bridges": [{
    "tag": "bridge",
    "domain": "test.v2ray.com"
  }]
}
```

出站代理:

```
{
  "tag": "out",
  "protocol": "freedom",
  "settings": {
    "redirect": "127.0.0.1:80" // 将所有流量转发到网页服务器
  }
},
{
  "protocol": "vmess",
  "settings": {
    "vnext": [{
      "address": "portal的IP地址",
      "port": 1024,
      "users": [{"id": "27848739-7e62-4138-9fd3-098a63964b6b"}]
    }]
  },
  "tag": "interconn"
}
```

路由配置:

```
"routing": {
  "rules": [{
    "type": "field",
    "inboundTag": ["bridge"],
    "domain": ["full:test.v2ray.com"],
```

```
    "outboundTag": "interconn"
  },{
    "type": "field",
    "inboundTag": ["bridge"],
    "outboundTag": "out"
  }]
}
```

`portal` 通常需要两个入站代理，一个用于接收 `bridge` 的连接，另一个用于接收实际的流量。同时你也需要用路由区分两种流量。

反向代理配置:

```
{
  "portals": [{
    "tag": "portal",
    "domain": "test.v2ray.com"  // 必须和 bridge 的配置一样
  }]
}
```

入站代理:

```
{
  "tag": "external",
  "port": 80,  // 开放 80 端口，用于接收外部的 HTTP 访问
  "protocol": "dokodemo-door",
  "settings": {
    "address": "127.0.0.1",
    "port": 80,
    "network": "tcp"
  }
},
{
  "port": 1024, // 用于接收 bridge 的连接
  "tag": "interconn",
  "protocol": "vmess",
  "settings": {
    "clients": [{"id": "27848739-7e62-4138-9fd3-098a63964b6b"}]
  }
}
```

路由配置:

```
"routing": {
  "rules": [{
    "type": "field",
    "inboundTag": ["external"],
    "outboundTag": "portal"
  },{
    "type": "field",
    "inboundTag": ["interconn"],
    "outboundTag": "portal"
  }]
}
```

在运行过程中，建议先启用 `bridge`，再启用 `portal`。

底层传输配置

底层传输方式（transport）是当前 V2Ray 节点和其它节点对接的方式。底层传输方式提供了稳定的数据传输通道。通常来说，一个网络连接的两端需要有对称的传输方式。比如一端用了 WebSocket，那么另一个端也必须使用 WebSocket，否则无法建立连接。

底层传输（transport）配置分为两部分，一是全局设置(TransportObject)，二是分协议配置(StreamSettingsObject)。分协议配置可以指定每个单独的入站出站协议用怎样的方式传输。通常来说客户端和服务器对应的出站入站协议需要使用同样的传输方式。当分协议传输配置指定了一种传输方式，但没有填写其设置时，此传输方式会使用全局配置中的设置。

TransportObject

`TransportObject` 对应配置文件的 `transport` 项。

```
{
  "tcpSettings": {},
  "kcpSettings": {},
  "wsSettings": {},
  "httpSettings": {},
  "dsSettings": {},
  "quicSettings": {}
}
```

> `tcpSettings` : TcpObject

针对 TCP 连接的配置。

> `kcpSettings` : KcpObject

针对 mKCP 连接的配置。

> `wsSettings` : WebSocketObject

针对 WebSocket 连接的配置。

> `httpSettings` : HttpObject

针对 HTTP/2 连接的配置。

> `dsSettings` : DomainSocketObject

针于Domain Socket 连接的配置。

> `quicSettings` : QUICObject

(V2Ray 4.7+) 针于QUIC 连接的配置。

StreamSettingsObject

`TransportObject` 对应出站入站协议中的 `streamSettings` 项。每一个入站、出站连接都可以分别配置不同的传输配置，都可以设置 `streamSettings` 来进行一些传输的配置。

```
{
  "network": "tcp",
  "security": "none",
  "tlsSettings": {},
  "tcpSettings": {},
  "kcpSettings": {},
```

```
  "wsSettings": {},
  "httpSettings": {},
  "dsSettings": {},
  "quicSettings": {},
  "sockopt": {
    "mark": 0,
    "tcpFastOpen": false,
    "tproxy": "off"
  }
}
```

> `network`: "tcp" | "kcp" | "ws" | "http" | "domainsocket" | "quic"

数据流所使用的网络类型，默认值为 `"tcp"`

> `security`: "none" | "tls"

是否启入传输层加密，支持的选项有 `"none"` 表示不加密（默认值），`"tls"` 表示使用 TLS。

> `tlsSettings`: TLSObject

TLS 配置。TLS 由 Golang 提供，支持 TLS 1.2，不支持 DTLS。

> `tcpSettings`: TcpObject

当前连接的 TCP 配置，仅当此连接使用 TCP 时有效。配置内容与上面的全局配置相同。

> `kcpSettings`: KcpObject

当前连接的 mKCP 配置，仅当此连接使用 mKCP 时有效。配置内容与上面的全局配置相同。

> `wsSettings`: WebSocketObject

当前连接的 WebSocket 配置，仅当此连接使用 WebSocket 时有效。配置内容与上面的全局配置相同。

> `httpSettings`: HttpObject

当前连接的 HTTP/2 配置，仅当此连接使用 HTTP/2 时有效。配置内容与上面的全局配置相同。

> `dsSettings`: DomainSocketObject

当前连接的 Domain socket 配置，仅当此连接使用 Domain socket 时有效。配置内容与上面的全局配置相同。

> `quicSettings`: QUICObject

(V2Ray 4.7+) 当前连接的 QUIC 配置，仅当此连接使用 QUIC 时有效。配置内容与上面的全局配置相同。

> `sockopt`: SockoptObject

连接选项

TLSObject

```
{
  "serverName": "v2ray.com",
  "allowInsecure": false,
  "alpn": ["http/1.1"],
  "certificates": [],
  "disableSystemRoot": false
}
```

> `serverName`: string

指定服务器端证书的域名，在连接由 IP 建立时有用。当目标连接由域名指定时，比如在 Socks 入站时接收到了域名，或者由 Sniffing 功能探测出了域名，这个域名会自动用于 `serverName`，无须手动配置。

> `alpn` : [string]

一个字符串数组，指定了 TLS 握手时指定的 ALPN 数值。默认值为 `["http/1.1"]`。

> `allowInsecure` : true | false

是否允许不安全连接（用于客户端）。当值为 `true` 时，V2Ray 不会检查远端主机所提供的 TLS 证书的有效性。

> `allowInsecureCiphers` : true | false

是否允许不安全的加密方式。默认情况下 TLS 只使用 TLS 1.3 推荐的加密算法套件，开启这一选项会增加一些与 TLS 1.2 兼容的加密套件。

> `disableSystemRoot` : true | false

(V2Ray 4.18+) 是否禁用操作系统自带的 CA 证书。默认值为 `false`。当值为 `true` 时，V2Ray 只会使用 `certificates` 中指定的证书进行 TLS 握手。

> `certificates` : [CertificateObject]

证书列表，其中每一项表示一个证书

CertificateObject

```
{
  "usage": "encipherment",

  "certificateFile": "/path/to/certificate.crt",
  "keyFile": "/path/to/key.key",

  "certificate": [
    "-----BEGIN CERTIFICATE-----",
    "MIICwDCCAaigAwIBAgIRAO16JMdESAuHidFYJAR/7kAwDQYJKoZIhvcNAQELBQAw",
    "ADAeFw0xODA0MTAxMzU1MTdaFw0xODA0MTAxNTU1MTdaMAAwggEiMA0GCSqGSIb3",
    "DQEBAQUAA4IBDwAwggEKAoIBAQCs2PX0fFSCjOemmdm9UbOvcLctF94Ox4BpSfJ+",
    "3lJHwZbvnOFuo56WhQJWrclKoImp/c9veL1J4Bbtam3sW3APkZVEK9UxRQ57HQuw",
    "OzhV0FD20/0YELou85TwnkTw5l9GVCXT02NG+pGlYsFrxesUHpojdl8tIcn113M5",
    "pypgDPVmPeeORRf7nseMC6GhvXYM4txJPyenohwegl8DZ6OE5FkSVR5wFQtAhbON",
    "OAkIVVmw002K2J6pitPuJGOka9PxcCVWhko/W+JCGapcC7O74palwBUuXE1iH+Jp",
    "noPjGp4qE2ognW3WH/sgQ+rvo20eXb9Um1steaYY8xlxgBsXAgMBAAGjNTAzMA4G",
    "A1UdDwEB/wQEAwIFoDATBgNVHSUEDDAKBggrBgEFBQcDATAMBgNVHRMBAf8EAjAA",
    "MA0GCSqGSIb3DQEBCwUAA4IBAQBUd9sGKYemzwPnxtw/vzkV8Q32NILEMlPVqeJU",
    "7UxVgIODBV6A1b3tOUoktuhmgSSaQxjhYbFAVTD+LUglMUCxNbj56luBRlLLQWo+",
    "9BUhC/ow393tLmqKcB59qNcwbZER6XT5POYwcaKM75QVqhCJVHJNb1zSEE7Co7iO",
    "6wIan3lFyjBfYlBEz5vyRWQNIwKfdh5cK1yAu13xGENwmtlSTHiwbjBLXfk+0A/8",
    "r/2s+sCYUkGZHhj8xY7bJ1zg0FRalP5LrqY+r6BckT1QPDIQKYy615j1LpOtwZe/",
    "d4q7MD/dkzRDsch7t2cIjM/PYeMuzh87admSyL6hdtK0Nm/Q",
    "-----END CERTIFICATE-----"
  ],
  "key": [
    "-----BEGIN RSA PRIVATE KEY-----",
    "MIIEowIBAAKCAQEArNj19HxUgoznppnZvVGzr3C3LRfeDseAaUnyft5SR8GW75zh",
    "bqOeloUCVq3JSqCJqf3Pb3i9SeAW7Wpt7FtwD5GVRCvVMUUOex0LsDs4VdBQ9tP9",
    "GBC6LvOU8J5E8OZfRlQl09NjRvqRpWLBa8XrFB6aI3ZfLSHJ9ddzOacqYAz1Zj3n",
    "jkUX+57HjAuhob12DOLcST8np6IcHoJfA2ejhORZElUecBULQIWzjTgJCFVZsNNN",
    "itieqYrT7iRjpGvT8XAlVoZKP1viQhmqXAuzu+KWpcAVLlxNYh/iaZ6D4xqeKhNq",
    "IJ1t1h/7IEPq76NtHl2/VJtbLXmmGPMZcYAbFwIDAQABAoIBAFCgG4phfGIxK9Uw",
    "qrp+o9xQLYGhQnmOYb27OpwnRCYojSlT+mvLcqwvevnHsr9WxyA+PkZ3AYS2PLue",
    "C4xW0pzQgdn8wENtPOX8lHkuBocw1rNsCwDwvIguIuliSjI8o3CAy+xVDFgNhWap",
    "/CMzfQYziB7GlnrM6hH838iiy0dlv4I/HKk+3/YlSYQEvnFokTf7HxbDDmznkJTM",
    "aPKZ5qbnV+4AcQfcLYJ8QE0ViJ8dVZ7RLwIf7+SG0b0bqloti4+oQXqGtiESUwEW",
    "/Wzi7oyCbFJoPsFWp1P5+wD7jAGpAd9lPIwPahdr1wl6VwIx9W0XYjoZn71AEaw4",
    "bK4xUXECgYEA3g2o9WqyrhYSax3pGEdvV2qN0VQhw7Xe+jyy98CELOO2DNbB9QNJ",
    "8cSSU/PjkxQlgbOJc8DEprdMldN5xI/srlsbQWCj72wXxXnVnh991bI2clwt7oYi",
    "pcGZwzCrJyFL+QaZmYzLxkxYl1tCiiuqLm+EkjxCWKTX/kKEFb6rtnMCgYEAx0WR",
    "L8Uue3lXxhXRdBS5QRTBNklkSxtU+2yyXRpvFa7Qam+GghJs5RKfJ9lTvjfM/PxG",
    "3vhuBliWQOKQbm1ZGLbgGBM505EOP7DikUmH/kzKxIeRo4l64mioKdDwK/4CZtS7",
    "az0Lq3eS6bq11qL4mEdE6Gn/Y+sqB83GHZYju80CgYABFm4KbbBcW+1RKv9WSBtK",
    "gVIagV/89moWLa/uuLmtApyEqZSfn5mAHqdc0+f8c2/Pl9KHh50u99zfKv8AsHfH",
    "TtjuVAvZg10GcZdTQ/I41ruficYL0gpfZ3haVWWxNl+J47di4iapXPxeGWtVA+u8",
    "eH1cvgDRMFWCgE7nUFzE8wKBgGndUomfZtdgGrp4ouLZk6W4ogD2MpsYNSixkXyW",
    "64cIbV7uSvZVVZbJMtaXxb6bpIKOgBQ6xTEH5SMpenPAEgJoPVts816rhHdfwK5Q",
    "8zetklegckYAZtFbqmM0xjOI6bu5rqwFLWr1xo33jF0wDYPQ8RHMJkruB1FIB8V2",
```

```
    "GxvNAoGBAM4g2z8NTPMqX+8IBGkGgqmcYuRQxd3cs7LOSEjF9hPy1it2ZFe/yUKq",
    "ePa2E8osffK5LBkFzhyQb0WrGC9ijM9E6rv10gyuNjlwXdFJcdqVamxwPUBtxRJR",
    "cYTY2HRkJXDdtT0Bkc3josE6UUDvwMpO0CfAETQPto1tjNEDhQhT",
    "-----END RSA PRIVATE KEY-----"
  ]
}
```

> `usage`: "encipherment" | "verify" | "issue"

证书用途，默认值为 `"encipherment"`

- `"encipherment"`：证书用于 TLS 认证和加密。
- `"verify"`：证书用于验证远端 TLS 的证书。当使用此项时，当前证书必须为 CA 证书。
- `"issue"`：证书用于签发其它证书。当使用此项时，当前证书必须为 CA 证书。

> 在 Windows 平台上可以将自签名的 CA 证书安装到系统中，即可验证远端 TLS 的证书。

> 当有新的客户端请求时，假设所指定的 `serverName` 为 `"v2ray.com"`，V2Ray 会先从证书列表中寻找可用于 `"v2ray.com"` 的证书，如果没有找到，则使用任一 `usage` 为 `"issue"` 的证书签发一个适用于 `"v2ray.com"` 的证书，有效期为一小时。并将新的证书加入证书列表，以供后续使用。

> `certificateFile`: string

证书文件路径，如使用 OpenSSL 生成，后缀名为 .crt。

> 使用 `v2ctl cert -ca` 可以生成自签名的 CA 证书。

> `certificate`: [string]

一个字符串数组，表示证书内容，格式如样例所示。`certificate` 和 `certificateFile` 二者选一。

> `keyFile`: string

密钥文件路径，如使用 OpenSSL 生成，后缀名为 .key。目前暂不支持需要密码的 key 文件。

> `key`: [string]

一个字符串数组，表示密钥内容，格式如样例如示。`key` 和 `keyFile` 二者选一。

当 `certificateFile` 和 `certificate` 同时指定时，V2Ray 优先使用 `certificateFile`。`keyFile` 和 `key` 也一样。

> 当 `usage` 为 `"verify"` 时，`keyFile` 和 `key` 可均为空。

SockoptObject

```
{
  "mark": 0,
  "tcpFastOpen": false,
  "tproxy": "off"
}
```

> `mark`: number

一个整数。当其值非零时，在出站连接上标记 SO_MARK。

- 仅适用于 Linux 系统。
- 需要 CAP_NET_ADMIN 权限。

> `tcpFastOpen`: true | false

是否启用 TCP Fast Open。当其值为 `true` 时，强制开启TFO；当其它为 `false` 时，强制关闭TFO；当此项不存在时，使用系统默认设置。可用于入站出站连接。

- 仅在以下版本（或更新版本）的操作系统中可用:
  - Windows 10 (1604)
  - Mac OS 10.11 / iOS 9
  - Linux 3.16: 系统已默认开启，无需要配置。

> `tproxy`: "redirect" | "tproxy" | "off"

是否开启透明代理 (仅适用于 Linux)。

- `"redirect"`: 使用 Redirect 模式的透明代理。仅支持 TCP/IPv4 和 UDP 连接。
- `"tproxy"`: 使用 TProxy 模式的透明代理。支持 TCP 和 UDP 连接。
- `"off"`: 关闭透明代理。

透明代理需要 Root 或 CAP_NET_ADMIN 权限。

当 Dokodemo-door 中指定了 `followRedirect`，且 `sockopt.tproxy` 为空时，`sockopt.tproxy` 的值会被设为 `"redirect"`。

TCP 传输方式

TcpObject

```
{
  "header": {
    "type": "none"
  }
}
```

> `header` : NoneHeaderObject | HttpHeaderobject

数据包头部伪装设置，默认值为 `NoneHeaderObject` 。

NoneHeaderObject

不进行伪装

```
{
  "type": "none"
}
```

> `type` : "none"

指定不进行伪装

HttpHeaderObject

HTTP 伪装配置必须在对应的入站出站连接上同时配置，且内容必须一致。

```
{
  "type": "http",
  "request": {},
  "response": {}
}
```

> `type` : "http"

指定进行 HTTP 伪装

> `request` : HTTPRequestObject

HTTP 请求

> `response` : HTTPResponseObject

HTTP 响应

HTTPRequestObject

```
{
  "version": "1.1",
  "method": "GET",
  "path": ["/"],
  "headers": {
    "Host": ["www.baidu.com", "www.bing.com"],
    "User-Agent": [
      "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.143 Safari/537.36",
```

```
      "Mozilla/5.0 (iPhone; CPU iPhone OS 10_0_2 like Mac OS X) AppleWebKit/601.1 (KHTML, like Gecko) CriOS/53.0.2785.10
9 Mobile/14A456 Safari/601.1.46"
    ],
    "Accept-Encoding": ["gzip, deflate"],
    "Connection": ["keep-alive"],
    "Pragma": "no-cache"
  }
}
```

> `version` : string

HTTP 版本，默认值为 `"1.1"` 。

> `method` : string

HTTP 方法，默认值为 `"GET"` 。

> `path` : [string]

路径，一个字符串数组。默认值为 `["/"]` 。当有多个值时，每次请求随机选择一个值。

> `headers` : map{ string, [string]}

HTTP 头，一个键值对，每个键表示一个 HTTP 头的名称，对应的值是一个数组。每次请求会附上所有的键，并随机选择一个对应的值。默认值见上方示例。

HTTPResponseObject

```
{
  "version": "1.1",
  "status": "200",
  "reason": "OK",
  "headers": {
    "Content-Type": ["application/octet-stream", "video/mpeg"],
    "Transfer-Encoding": ["chunked"],
    "Connection": ["keep-alive"],
    "Pragma": "no-cache"
  }
}
```

> `version` : string

HTTP 版本，默认值为 `"1.1"` 。

> `status` : string

HTTP 状态，默认值为 `"200"` 。

> `reason` : string

HTTP 状态说明，默认值为 `"OK"` 。

> `headers` : map{string, [string]}

HTTP 头，一个键值对，每个键表示一个 HTTP 头的名称，对应的值是一个数组。每次请求会附上所有的键，并随机选择一个对应的值。默认值见上方示例。

mKCP 传输方式

mKCP 使用 UDP 来模拟 TCP 连接，请确定主机上的防火墙配置正确。mKCP 牺牲带宽来降低延迟。传输同样的内容，mKCP 一般比 TCP 消耗更多的流量。

KcpObject

```
{
  "mtu": 1350,
  "tti": 20,
  "uplinkCapacity": 5,
  "downlinkCapacity": 20,
  "congestion": false,
  "readBufferSize": 1,
  "writeBufferSize": 1,
  "header": {
    "type": "none"
  }
}
```

> `mtu` : number

最大传输单元（maximum transmission unit），请选择一个介于 `576` - `1460` 之间的值。默认值为 `1350`。

> `tti` : number

传输时间间隔（transmission time interval），单位毫秒（ms），mKCP 将以这个时间频率发送数据。请选译一个介于 `10` - `100` 之间的值。默认值为 `50`。

> `uplinkCapacity` : number

上行链路容量，即主机发出数据所用的最大带宽，单位 MB/s，默认值 `5`。注意是 Byte 而非 bit。可以设置为 `0`，表示一个非常小的带宽。

> `downlinkCapacity` : number

下行链路容量，即主机接收数据所用的最大带宽，单位 MB/s，默认值 `20`。注意是 Byte 而非 bit。可以设置为 `0`，表示一个非常小的带宽。

> `uplinkCapacity` 和 `downlinkCapacity` 决定了 mKCP 的传输速度。以客户端发送数据为例，客户端的 `uplinkCapacity` 指定了发送数据的速度，而服务器端的 `downlinkCapacity` 指定了接收数据的速度。两者的值以较小的一个为准。推荐把 `downlinkCapacity` 设置为一个较大的值，比如 100，而 `uplinkCapacity` 设为实际的网络速度。当速度不够时，可以逐渐增加 `uplinkCapacity` 的值，直到带宽的两倍左右。

> `congestion` : true | false

是否启用拥塞控制，默认值为 `false`。开启拥塞控制之后，V2Ray 会自动监测网络质量，当丢包严重时，会自动降低吞吐量；当网络畅通时，也会适当增加吞吐量。

> `readBufferSize` : number

单个连接的读取缓冲区大小，单位是 MB。默认值为 `2`。

> `writeBufferSize` : number

单个连接的写入缓冲区大小，单位是 MB。默认值为 `2`。

> `readBufferSize` 和 `writeBufferSize` 指定了单个连接所使用的内存大小。在需要高速传输时，指定较大的 `readBufferSize` 和 `writeBufferSize` 会在一定程度上提高速度，但也会使用更多的内存。在网速不超过 20MB/s 时，默认值 1MB 可以满足需求；超过之后，可以适当增加 `readBufferSize` 和 `writeBufferSize` 的值，然后手动平衡速度和内存的关系。

> `header`: HeaderObject

数据包头部伪装设置

HeaderObject

```
{
  "type": "none"
}
```

> `type`: string

伪装类型，可选的值有：

- `"none"`：默认值，不进行伪装，发送的数据是没有特征的数据包。
- `"srtp"`：伪装成 SRTP 数据包，会被识别为视频通话数据（如 FaceTime）。
- `"utp"`：伪装成 uTP 数据包，会被识别为 BT 下载数据。
- `"wechat-video"`：伪装成微信视频通话的数据包。
- `"dtls"`：伪装成 DTLS 1.2 数据包。
- `"wireguard"`：伪装成 WireGuard 数据包。(并不是真正的 WireGuard 协议)

鸣谢

- @skywind3000 发明并实现了 KCP 协议；
- @xtaci 将 KCP 由 C 语言实现翻译成 Go；
- @xiaokangwang 测试 KCP 与 V2Ray 的整合并提交了最初的 PR。

对 **KCP** 协议的改进

更小的协议头

原生 KCP 协议使用了 24 字节的固定头部，而 mKCP 修改为数据包 18 字节，确认（ACK）包 16 字节。更小的头部有助于躲避特征检查，并加快传输速度。

另外，原生 KCP 的单个确认包只能确认一个数据包已收到，也就是说当 KCP 需要确认 100 个数据已收到时，它会发出 24 * 100 = 2400 字节的数据。其中包含了大量重复的头部数据，造成带宽的浪费。mKCP 会对多个确认包进行压缩，100 个确认包只需要 16 + 2 + 100 * 4 = 418 字节，相当于原生的六分之一。

确认包重传

原生 KCP 协议的确认（ACK）包只发送一次，如果确认包丢失，则一定会导致数据重传，造成不必要的带宽浪费。而 mKCP 会以一定的频率重发确认包，直到发送方确认为止。单个确认包的大小为 22 字节，相比起数据包的 1000 字节以上，重传确认包的代价要小得多。

连接状态控制

mKCP 可以有效地开启和关闭连接。当远程主机主动关闭连接时，连接会在两秒钟之内释放；当远程主机断线时，连接会在最多 30 秒内释放。

原生 KCP 不支持这个场景。

WebSocket 传输方式

使用标准的 WebSocket 来传输数据。WebSocket 连接可以被其它 HTTP 服务器（如 NGINX）分流。

> Websocket 会识别 HTTP 请求的 X-Forwarded-For 头来用做流量的源地址。

示例配置

```
{
  "path": "/",
  "headers": {
    "Host": "v2ray.com"
  }
}
```

> `path` string

WebSocket 所使用的 HTTP 协议路径，默认值为 `"/"`。

> `headers` : map{string: string}

自定义 HTTP 头，一个键值对，每个键表示一个 HTTP 头的名称，对应的值是字符串。默认值为空。

HTTP/2 传输方式

V2Ray 3.17 中加入了基于 HTTP/2 的传输方式。它完整按照 HTTP/2 标准实现，可以通过其它的 HTTP 服务器（如 Nginx）进行中转。

由 HTTP/2 的建议，客户端和服务器必须同时开启 TLS 才可以正常使用这个传输方式。

HttpObject

`HttpObject` 对应传输配置中的 `httpSettings` 项。

```
{
  "host": ["v2ray.com"],
  "path": "/random/path"
}
```

> `host`: [string]

一个字符串数组，每一个元素是一个域名。客户端会随机从列表中选出一个域名进行通信，服务器会验证域名是否在列表中。

> `path` string

HTTP 路径，由 `/` 开头。客户端和服务器必须一致。可选参数，默认值为 `"/"`。

DomainSocket 传输方式

Domain Socket 使用标准的 Unix domain socket 来传输数据。它的优势是使用了操作系统内建的传输通道，而不会占用网络缓存。相比起本地环回网络（local loopback）来说，Domain socket 速度略快一些。

目前仅可用于支持 Unix domain socket 的平台，如 macOS 和 Linux。在 Windows 上不可用。

> 如果指定了 domain socket 作为传输方式，在入站出站代理中配置的端口和 IP 地址将会失效，所有的传输由 domain socket 取代。

DomainSocketObject

`DomainSocketObject` 对应传输配置中的 `dsSettings` 项。

```
{
  "path": "/path/to/ds/file"
}
```

> `path` : string

一个合法的文件路径。在运行 V2Ray 之前，这个文件必须不存在。

QUIC 传输方式

QUIC 全称 Quick UDP Internet Connection，是由 Google 提出的使用 UDP 进行多路并发传输的协议。其主要优势是:

1. 减少了握手的延迟（1-RTT 或 0-RTT）
2. 多路复用，并且没有 TCP 的阻塞问题
3. 连接迁移，（主要是在客户端）当由 Wifi 转移到 4G 时，连接不会被断开。

QUIC 目前处于实验期，使用了正在标准化过程中的 IETF 实现，不能保证与最终版本的兼容性。

版本历史

V2Ray 4.7:

- 开始支持 QUIC。
- 默认设定:
  - 12 字节的 Connection ID
  - 30 秒没有数据通过时自动断开连接 (可能会影响一些长连接的使用)

QuicObject

QUIC 的配置对应传输配置中的 `quicSettings` 项。对接的两端的配置必须完全一致，否则连接失败。QUIC 强制要求开启 TLS，在传输配置中没有开启 TLS 时，V2Ray 会自行签发一个证书进行 TLS 通讯。在使用 QUIC 传输时，可以关闭 VMess 的加密。

```
{
  "security": "none",
  "key": "",
  "header": {
    "type": "none"
  }
}
```

> `security`: "none" | "aes-128-gcm" | "chacha20-poly1305"

加密方式。默认值为不加密。

此加密是对 QUIC 数据包的加密，加密后数据包无法被探测。

> `key`: string

加密时所用的密钥。可以是任意字符串。当 `security` 不为 `"none"` 时有效。

> `header`: HeaderObject

数据包头部伪装设置

HeaderObject

```
{
  "type": "none"
}
```

> `type`: string

伪装类型，可选的值有：

- `"none"`：默认值，不进行伪装，发送的数据是没有特征的数据包。
- `"srtp"`：伪装成 SRTP 数据包，会被识别为视频通话数据（如 FaceTime）。
- `"utp"`：伪装成 uTP 数据包，会被识别为 BT 下载数据。
- `"wechat-video"`：伪装成微信视频通话的数据包。
- `"dtls"`：伪装成 DTLS 1.2 数据包。
- `"wireguard"`：伪装成 WireGuard 数据包。(并不是真正的 WireGuard 协议)

> 当加密和伪装都不启用时，数据包即为原始的 QUIC 数据包，可以与其它的 QUIC 工具对接。为了避免被探测，建议加密或伪装至少开启一项。

环境变量

V2Ray 提供以下环境变量以供修改 V2Ray 的一些底层配置。

每个连接的缓存大小

- 名称: `v2ray.ray.buffer.size` 或 `V2RAY_RAY_BUFFER_SIZE`
- 单位: MBytes
- 默认值: 在 x86、amd64、arm64、s390x 上为 2，其它平台上禁用该缓存。
- 特殊值: 0 表示缓存无上限

已过时，请使用本地策略中的 **bufferSize**

对于一个代理连接，当上下游网络速度有差距时，V2Ray 会缓存一部分数据，以减小对网络传输的影响。这个配置设置了缓存的大小，越大的缓存会占用更多的内存，也会使网络性能越好。

资源文件路径

- 名称: `v2ray.location.asset` 或 `V2RAY_LOCATION_ASSET`
- 默认值: 和 v2ray 文件同路径

这个环境变量指定了一个文件夹位置，这个文件夹应当包含 geoip.dat 和 geosite.dat 文件。

配置文件位置

- 名称: `v2ray.location.config` 或 `V2RAY_LOCATION_CONFIG`
- 默认值: 和 v2ray 文件同路径

这个环境变量指定了一个文件夹位置，这个文件夹应当包含 config.json 文件。

分散读取

- 名称: `v2ray.buf.readv` 或 `V2RAY_BUF_READV`
- 默认值: `auto`

V2Ray 3.37 开始使用 Scatter/Gather IO，这一特性可以在大流量（超过 100 MByte/s）的时候依然使用较低的内存。可选的值有 `auto`、`enable` 和 `disable`。

- `enable`: 强制开启分散读取特性。
- `disable`: 强制关闭分散读取特性
- `auto`: 仅在 Windows、MacOS、Linux 并且 CPU 平台为 x86、AMD64、s390x 时，开启此特性。

在流量没有达到 100 MByte/s 时，开启与否在内存使用上没有明显的差异。

神一样的工具们

图形客户端

> V2RayW 

V2RayW 是一个基于 V2Ray 内核的 Windows 客户端。用户可以通过界面生成配置文件，并且可以手动更新 V2Ray 内核。下载：GitHub

> V2RayN 

V2RayN 是一个基于 V2Ray 内核的 Windows 客户端。下载：GitHub

> V2RayS 

下载：GitHub

> Clash for Windows 

下载：GitHub

> V2RayX 

V2RayX 是一个基于 V2Ray 内核的 Mac OS X 客户端。用户可以通过界面生成配置文件，并且可以手动更新 V2Ray 内核。V2RayX 还可以配置系统代理。下载：Github

> V2RayU 

下载：GitHub

> V2RayC 

下载：GitHub

> ClashX 

下载：GitHub

> Kitsunebi 

Kitsunebi 是一个基于 V2Ray 核心的 iOS 应用。它可以创建基于 VMess 或者 Shadowsocks 的 VPN 连接。Kitsunebi 支持导入和导出与 V2Ray 兼容的 JSON 配置。

由于使用 V2Ray 核心，Kitsunebi 几乎支持 V2Ray 的所有功能，比如 Mux 和 mKCP。

下载：iTunes

> i2Ray 

i2Ray 是另一款基于 V2Ray 核心的iOS应用。界面简洁易用，适合新手用户使用。同时兼容Shadowrocket和Quantumult格式的规则导入。

下载：iTunes

> Shadowrocket 

Shadowrocket 是一个通用的 iOS VPN 应用，它支持众多协议，如 Shadowsocks、VMess、SSR 等。

下载：iTunes

> Pepi（原名ShadowRay）

Pepi 是一个兼容 V2Ray 的 iOS 应用，它可以创建基于 VMess 的 VPN 连接，并与 V2Ray 服务器通信。

下载：iTunes

> Quantumult

下载：iTunes

> BifrostV

BifrostV 是一个基于 V2Ray 内核的 Android 应用，它支持 VMess、Shadowsocks、Socks 协议。

下载：Play Store | APK Pure

> V2RayNG

V2RayNG 是一个基于 V2Ray 内核的 Android 应用，它可以创建基于 VMess 的 VPN 连接。

下载：Play Store | GitHub

在线工具/资源

> VeekXT V2Ray配置生成

支持 4.x 版本的配置文件生成器 veekxt.com

> V2Ray 配置生成器

静态 V2Ray 配置文件生成页面 GitHub

> UUID Generator

VMess User ID 生成工具 uuidgenerator.net

> vTemplate 项目仓库

一个 V2Ray 配置文件模板收集仓库 GitHub

一些推广

以下服务均由第三方提供，和 Project V 没有利益关系，但它们可能会帮助你更好地使用 Project V。

自建代理

> Let's Encrypt

Let's Encrypt是一个于2015年三季度推出的数字证书认证机构，旨在以自动化流程消除手动创建和安装证书的复杂流程，并推广使万维网服务器的加密连接无所不在，为安全网站提供免费的SSL/TLS证书。

> Vultr

Vultr 是一家提供日本、美国、欧洲等多个国家和地区机房的 VPS 主机商，价格低至 2.5 美元/月。Vultr 根据VPS使用小时来计费，使用多长时间就算多长时间，计费对应的款，并且支持支付宝（Alipay）付费。

> BlueHost

BlueHost作为美国的老牌主机商，在国内也是非常受欢迎的美国主机之一。以稳定性强。速度快著称，在站长中的口碑和流行度也是非常之高。BlueHost主机商为了开拓国内市场，自2014年就推出了中文站，而且也极大的丰富了主机产品，可以满足大多数站长的建站需求。

> ClouDNS

ClouDNS成立于2010年，公司位于保加利亚，提供免费和收费的 DNS 托管服务。

> Bandwagon

老牌 VPS 销售商，价格低廉，性价比高。

VPN

> BabyDriver

支持 V2Ray 的 VPN 服务。优惠码：bcb518

> 喵帕斯

V2Ray 小范围内测中。

> 蓝岸

基于 V2Ray 的网络加速服务。优惠码：v2ray

> V2rayPro

基于 V2Ray 的网络加速服务。专属优惠码：v2ray.com

> V2Net

提供專屬客戶端的V2Ray服務。9折促銷代碼: v2ray.com

> NicoNode

支持 V2Ray 的网络加速改善服务。专属促销代码：V2RAYNOW

数字货币

> LocalBitcoins

线下交易比特币

> CoinCola

CoinCola 是香港场外交易平台，支持简体中文界面，买卖方几乎都是国人。网页和APP均操作流畅。订单使用支付宝、网银、微信支付，像淘宝购物一样买卖比特币。现支持BTC、ETH、BCH、USDT等货币。

> 币安

数字货币交易所

> Coinex

数字货币交易所

> CoinPayment

在线数字货币钱包

> PrimeDice

用比特币玩骰子游戏

> OneHash

用比特币竞猜体育比赛，包括世界杯

> Bitsler

使用比特币玩赌场游戏

开发人员手册

Project V

Project V 由以下几部分组成:

- V2Ray 核心: v2ray.com/core
- V2Ray 手册: v2ray.com
- 周边项目: 详见客户端

V2Ray 项目组

V2Ray 项目组最初是为了 V2Ray 核心的开发而建立，现在已推广到整个 Project V 生态环境。我们欢迎你加入组织参与开发。当然你也选择可以使用自己的 V2Ray 之外的环境。

加入 V2Ray 的朋友们可以使用项目组提供的如下设施:

- 私有讨论组: 只对 V2Ray 项目组开放的讨论组。
- 私有构建环境: 完全封闭的编译环境，可以安全地使用私有内容，如 GPG 私钥。

加入方式

只要你的 Github 帐号已有一些合理的代码，或已向 V2Ray 项目提交过 PR，都可以申请加入。请向 `love@v2ray.com` 发送邮件，注明你的 Github 帐号，稍后你会收到邀请。

权限及义务

- 项目组的所有成员都自动获得 push 和创建 repo 的权限。
- 除了几个核心的 branch 外，所有成员可以自由地提交代码。
- 项目组成员有义务汇报任何可能威胁到项目安全的行为。

周边项目

即使不加入项目组，Project V 也欢迎你开发兼容 Project V 的软件。兼容 Project V 的软件须遵循以下协议:

第三方开发者的权利和义务

1. 开发者拥有所开发软件的全部版权。
2. 开发者拥有所开发软件的所有盈利，同时也须独立承担开发过程中所有的开销和风险。
3. 开发者可以在软件中使用 “V2Ray” 和 “Project V” 一词、Project V 图标和其它 Project V 相关的用语。
4. 开发者可以自由地使用 V2Ray 的核心代码。
5. 开发者有义务向公众宣传 Project V 项目。
6. 开发者有义务向 Project V 官方汇报使用过程中的问题。
7. 开发者有义务在其软件或介绍中提到 Project V 项目和 Project V 官网链接。

Project V 官方的权利和义务

1. Project V 官方有义务向开发者提供技术支持。

2. Project V 官方有义务向开发者提供资金支持。在开发者接受的情况下，资金将以不定期捐赠的方式进行。
3. Project V 官方保留向任何侵权行为追责的权利。

开发计划

版本号

V2Ray Core 的版本号形如 X.Y.Z，其中 X 表示 Milestone，Y 表示 Release，如 2.3 表示第二个 Milestone 的第三个 Release；Z 表示测试版本。

周期

V2Ray Core 每周发布一个 Release。从 2.0 开始，每个 Milestone 持续一年。

进度管理

所有新功能的讨论和计划都放在 v2ray/Planning。

开发指引

基本

版本控制

Git

分支（**Branch**）

本项目只使用一个分支，即 master。所有更改全部提交进 master，并确保 master 在任一时刻都是可编译可使用的。

发布（**Release**）

尽量使用自动化工具发布，比如 v2ray-core 使用 Travis-ci 作为自动编译和发布工具。

引用其它项目

- Golang
  - 产品代码只能使用 golang 的标准库，即名称不包含任何网址的包；
  - 测试代码可以使用 golang.org/x/... ；
  - 如需引用其它项目请事先创建 Issue 讨论；
- 其它
  - 只要不违反双方的协议（本项目为 MIT），且对项目有帮助的工具，都可以使用。

开发流程

写代码之前

发现任何问题，或对项目有任何想法，请立即创建 Issue 讨论之，以减少重复劳动和消耗在代码上的时间。

修改代码

- Golang
  - 请参考 Effective Go；
  - 每一次 commit 之前请运行： gofmt -w v2ray.com/core/
  - 每一次 commit 之前请确保测试通过： go test v2ray.com/core/...
  - 提交 PR 之前请确保新增代码有超过 70% 的代码覆盖率（code coverage）。
- 其它
  - 请注意代码的可读性

Pull Request

- 提交 PR 之前请先运行 git pull 以确保 merge 可顺利进行；
- 一个 PR 只做一件事，如有对多个 bug 的修复，请对每一个 bug 提交一个 PR；
- 由于 Golang 的特殊需求（Package path），Go 项目的 PR 流程和其它项目有所不同：
  1. 先 Fork 本项目，创建你自己的 github.com/your/v2ray-core；
  2. 在你的 Go workspace 中运行： `go get -u v2ray.com/core/...` ；

3. 在 go get 创建的 v2ray-core 目录中运行：`git remote add fork https://github.com/you/cooltool.git`；
4. 然后你可以在 v2ray-core 中修改代码，由于这是一个 v2ray 的 clone，import path 不受影响；
5. 修改完成之后，运行：`git push fork`；
6. 然后去你的 fork（就是 v2ray.com/core）中发一个 PR 即可；
7. 以上内容修改自这篇文章。

对代码的修改

功能性问题

请提交至少一个测试用例（test case）来验证对现有功能的改动。

性能相关

请提交必要的测试数据来证明现有代码的性能缺陷，或是新增代码的性能提升。

新功能

- 如果新增功能对已有功能不影响，请提供可以开启/关闭的开关（如 flag），并使新功能保持默认关闭的状态；
- 大型新功能（比如增加一个新的协议）开发之前，请先提交一个 issue，讨论完毕之后再进行开发。

其它

视具体情况而定。

V2Ray 编码规范

以下内容适用于 V2Ray 中的 Golang 代码。

代码结构

```
v2ray-core
├── app        // 应用模块
│   ├── router // 路由
├── common     // 公用代码
├── proxy      // 通讯协议，参见[协议列表](../chapter_02/02_protocols.md)
│   ├── blackhole
│   ├── dokodemo-door
│   ├── freedom
│   ├── socks
│   ├── vmess
├── transport  // 传输模块
```

编码规范

基本和 Golang 的官方推荐做法基本一致，有一些例外。写在这里以方便大家熟悉 Golang。

命名

- 文件和目录名尽量使用单个英文单词，比如 hello.go；
  - 如果实在没办法，则目录使用连接线／文件名使用下划线连接两个（或多个单词），比如 hello-world/hello_again.go；
  - 测试代码使用 _test.go 结尾；
- 类型使用 Pascal 命名法，比如 ConnectionHandler；

    - 对缩写不强制小写，即 HTML 不必写成 Html；
- 公开成员变量也使用 Pascal 命名法；
- 私有成员变量使用小驼峰式命名法，如 privateAttribute；
- 为了方便重构，方法建议全部使用 Pascal 命名法；
    - 尽管 Golang 中的以大小写区分公开和私有方法，但在实际操作中并不方便。
    - 完全私有的类型放入 internal。

内容组织

- 一个文件包含一个主要类型，及其相关的私有函数等；
- 测试相关的文件，如 Mock 等工具类，放入 testing 子目录；

核心设计

本文描述了 V2Ray 内核（v2ray-core）的设计思路。

目标

- V2Ray 内核提供了一个平台，支持必要的网络代理功能，在其之上可以进二次开发，以提供更好的用户体验；
- 以跨平台为首要原则，以减少二次开发的成本；

架构

内核分为三层：应用层、代理层和传输层。每一层内包含数个模块，模块间互相独立，同类型的模块可以无缝替换。

应用层

应用层包含一些代理层中常用的功能，这些功能被抽象出来，以便在不同的代理模块中复用。应用层的模块应为纯软件实现，不与硬件或平台相关的技术有关。

重要模块列表：

- Dispatcher: 用于把入站代理所接收到的数据，传送给出站代理；
- Router: 内置路由，详见路由配置；
- DNS: 内置的 DNS 缓存；
- Proxy Manager: 入站代理的管理器；

代理层

代理层分为两部分：入站代理（Inbound Proxy）和出站代理（Outbound Proxy）。两部分相互独立，入站代理不依赖于某个特定的出站代理，反之亦然。所有已实现的协议列表一览。

入站代理

- 实现 proxy.Inbound 接口；

出站代理

- 实现 proxy.Outbound 接口；

传输层

传输层提供一些网络数据传输相关的工具模块。

配置开发环境

V2Ray 使用 Golang 作为主要编程语言、Bazel作为构建工具。推荐使用 Mac OS 或 Linux 进行开发，少量的脚本可能无法在 Windows 上正常运行。

前序工作

- 安装 Golang: golang.org/doc/install
- 安装 Bazel: docs.bazel.build/install

拉取 **V2Ray** 源代码

```
go get -u v2ray.com/core/...
```

自动构建

如果只需要构建某个特定平台的安装包，如 Linux / AMD64:

```
cd $GOPATH/src/v2ray.com/core
bazel build --action_env=GOPATH=$GOPATH --action_env=PATH=$PATH //release:v2ray_linux_amd64_package
#Output: bazel-bin/release/v2ray-linux-64.zip
```

构建所有安装包:

```
cd $GOPATH/src/v2ray.com/core
bazel build --action_env=GOPATH=$GOPATH --action_env=PATH=$PATH //release:all
```

安装构建完成的安装包

```
$GOPATH/src/v2ray.com/core/release/install-release.sh --local <path/to/zip/file>
```

自动化从源代码构建

某些场景可能需要从源代码构建，而不能直接下载安装包，比如制作一个安装源的时候。以下提供一个简单的自动构建方法:

1. 安装 Golang 和 Bazel，并设置 GOPATH。
2. 下载完整的源代码: `curl -L -O https://github.com/v2ray/v2ray-core/releases/latest/src_all.zip` 。这个压缩包从 3.46.4 开始提供，包含了编译 V2Ray 所需的所有代码。
3. 解压: `unzip -d $GOPATH/src/ src_all.zip`
4. 构建:

```
cd $GOPATH/src/v2ray.com/core
bazel build --action_env=GOPATH=$GOPATH --action_env=PATH=$PATH //release:v2ray_linux_amd64_package
```

1. 然后可以解压安装包并重新打包: `unzip bazel-bin/release/v2ray-linux-64.zip`

开发工具

第三方 **SDK**

- C#: v2ray-dotnet-sdk

自动化工具

V2Ray 使用下列自动化工具进行编译和发布。

- Bazel: 用于编译和打包。
- Azure DevOps: 用于部分项目的自动化发布。
- Google Cloud: 用于部分项目的自动化发布。
- CloudFlare: 用于支持官网和域名解析。

VMess 协议

VMess 是 V2Ray 原创的加密通讯协议。

版本

当前版本号为 1。

依赖

底层协议

VMess 是一个基于 TCP 的协议，所有数据使用 TCP 传输。

用户 ID

ID 等价于 UUID，是一个 16 字节长的随机数，它的作用相当于一个令牌（Token）。一个 ID 形如：de305d54-75b4-431b-adb2-eb6b9e546014，几乎完全随机，可以使用任何的 UUID 生成器来生成，比如这个。

用户 ID 可在配置文件中指定。

函数

- MD5: MD5 函数
  - 输入参数为任意长度的 byte 数组
  - 输出为一个 16 byte 的数组
- HMAC: HMAC 函数
  - 输入参数为：
    - H：散列函数
    - K：密钥，任意长度的 byte 数组
    - M：消息，任意长度的 byte 数组
- Shake: SHA3-Shake128 函数
  - 输入参数为任意长度的字符串
  - 输出为任意长度的字符串

通讯过程

VMess 是一个无状态协议，即客户端和服务器之间不需要握手即可直接传输数据，每一次数据传输对之前和之后的其它数据传输没有影响。VMess 的客户端发起一次请求，服务器判断该请求是否来自一个合法的客户端。如验证通过，则转发该请求，并把获得的响应发回给客户端。VMess 使用非对称格式，即客户端发出的请求和服务器端的响应使用了不同的格式。

客户端请求

16 字节	**X** 字节	余下部分
认证信息	指令部分	数据部分

认证信息

认证信息是一个 16 字节的哈希（hash）值，它的计算方式如下：

- H = MD5
- K = 用户 ID (16 字节)
- M = UTC 时间，精确到秒，取值为当前时间的前后 30 秒随机值(8 字节, Big Endian)
- Hash = HMAC(H, K, M)

指令部分

指令部分经过 AES-128-CFB 加密：

- Key：MD5(用户 ID + []byte('c48619fe-8f02-49e0-b9e9-edf763e17e21'))
- IV：MD5(X + X + X + X)，X = []byte(认证信息生成的时间) (8 字节, Big Endian)

1 字节	**16** 字节	**16** 字节	**1** 字节	**1** 字节	**4** 位	**4** 位	**1** 字节	**1** 字节	**2** 字节	**1** 字节	**N** 字节	**P** 字节	**4** 字节
版本号 Ver	数据加密 IV	数据加密 Key	响应认证 V	选项 Opt	余量 P	加密方式 Sec	保留	指令 Cmd	端口 Port	地址类型 T	地址 A	随机值	校验 F

选项 Opt 细节：（当某一位为 1 时，表示该选项启用）

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
X	X	X	X	X	M	R	S

其中：

- 版本号 Ver：始终为 1；
- 数据加密 IV：随机值；
- 数据加密 Key：随机值；
- 响应认证 V：随机值；
- 选项 Opt：
  - S (0x01)：标准格式的数据流（建议开启）；
  - R (0x02)：客户端期待重用 TCP 连接（V2Ray 2.23+ 弃用）；
    - 只有当 S 开启时，这一项才有效；
  - M (0x04)：开启元数据混淆（建议开启）；
    - 只有当 S 开启时，这一项才有效；
    - 当其项开启时，客户端和服务器端需要分别构造两个 Shake 实例，分别为 RequestMask = Shake(请求数据 IV), ResponseMask = Shake(响应数据 IV)。
  - X：保留
- 余量 P：在校验值之前加入 P 字节的随机值；
- 加密方式：指定数据部分的加密方式，可选的值有：
  - 0x00：AES-128-CFB；
  - 0x01：不加密；
  - 0x02：AES-128-GCM；
  - 0x03：ChaCha20-Poly1305；
- 指令 Cmd：
  - 0x01：TCP 数据；
  - 0x02：UDP 数据；

- 端口 Port：Big Endian 格式的整型端口号；
- 地址类型 T：
  - 0x01：IPv4
  - 0x02：域名
  - 0x03：IPv6
- 地址 A：
  - 当 T = 0x01 时，A 为 4 字节 IPv4 地址；
  - 当 T = 0x02 时，A 为 1 字节长度（L） + L 字节域名；
  - 当 T = 0x03 时，A 为 16 字节 IPv6 地址；
- 校验 F：指令部分除 F 外所有内容的 FNV1a hash；

数据部分

数据部分有两种格式，默认为基本格式。

基本格式 （弃用）

此格式仅作为向后兼容所用，在之后的版本中可能被删除。

所有数据均认为是请求的实际内容。这些内容将被发往指令部分所指定的地址。当 Cmd = 0x01 时，这些数据将以 TCP 的形式发送；当 Cmd = 0x02 时，这些数据将以 UDP 形式发送。

此格式支持“不加密”和“AES-128-CFB”两种加密方式，加密的 Key 和 IV 由指令部分指定。

标准格式

当 Opt(S) 开启时，数据部分使用此格式。实际的请求数据被分割为若干个小块，每个小块的格式如下。服务器校验完所有的小块之后，再按基本格式的方式进行转发。

2 字节	**L** 字节
长度 L	数据包

其中：

- 长度 L：Big Endian 格式的整型，最大值为 2^14；
  - 当 Opt(M) 开启时，L 的值 = 真实值 xor Mask。Mask = (RequestMask.NextByte() << 8) + RequestMask.NextByte()；
- 数据包：由指定的加密方式加密过的数据包；

在传输结束之前，数据包中必须有实际数据，即除了长度和认证数据之外的数据。当传输结束时，客户端必须发送一个空的数据包，即 L = 0（不加密） 或认证数据长度（有加密），来表示传输结束。

按加密方式不同，数据包的格式如下：

- 不加密：
  - L 字节：实际数据；
- AES-128-CFB：整个数据部分使用 AES-128-CFB 加密
  - 4 字节：实际数据的 FNV1a hash；
  - L - 4 字节：实际数据；
- AES-128-GCM：Key 为指令部分的 Key，IV = count (2 字节) + IV (10 字节)。count 从 0 开始递增，每个数据包加 1；IV 为 指令部分 IV 的第 3 至第 12 字节。
  - L - 16 字节：实际数据；
  - 16 字节：GCM 认证信息
- ChaCha20-Poly1305：Key = MD5(指令部分 Key) + MD5(MD5(指令部分 Key))，IV = count (2 字节) + IV (10 字节)。

count 从 0 开始递增，每个数据包加 1；IV 为 指令部分 IV 的第 3 至第 12 字节。

- L - 16 字节：实际数据；
- 16 字节：Poly1305 认证信息

服务器应答

应答头部数据使用 AES-128-CFB 加密，IV 为 MD5(数据加密 IV)，Key 为 MD5(数据加密 Key)。实际应答数据视加密设置不同而不同。

1 字节	**1** 字节	**1** 字节	**1** 字节	**M** 字节	余下部分
响应认证 V	选项Opt	指令 Cmd	指令长度 M	指令内容	实际应答数据

其中：

- 响应认证 V：必须和客户端请求中的响应认证 V 一致；
- 选项 Opt：
  - 0x01：服务器端准备重用 TCP 连接（V2Ray 2.23+ 弃用）；
- 指令 Cmd：
  - 0x01：动态端口指令
- 实际应答数据：
  - 如果请求中的 Opt(S) 开启，则使用标准格式，否则使用基本格式。
  - 格式均和请求数据相同。
    - 当 Opt(M) 开启时，长度 L 的值 = 真实值 xor Mask。Mask = (ResponseMask.NextByte() << 8) + ResponseMask.NextByte()；

动态端口指令

1 字节	**2** 字节	**16** 字节	**2** 字节	**1** 字节	**1** 字节
保留	端口 Port	用户 ID	AlterID	用户等级	有效时间 T

其中：

- 端口 Port：Big Endian 格式的整型端口号；
- 有效时间 T：分钟数；

客户端在收到动态端口指令时，服务器已开放新的端口用于通信，这时客户端可以将数据发往新的端口。在 T 分钟之后，这个端口将失效，客户端必须重新使用主端口进行通信。

注释

- 为确保向前兼容性，所有保留字段的值必须为 0。

mKCP 协议

mKCP 是流式传输协议，由 KCP 协议修改而来，可以按顺序传输任意的数据流。

版本

mKCP 没有版本号，不保证版本之间兼容性。

依赖

底层协议

mKCP 是一个基于 UDP 的协议，所有通讯使用 UDP 传输。

函数

- fnv: FNV-1a 哈希函数
  - 输入参数为任意长度的字符串；
  - 输入出一个 32 位无符号整数；

通讯过程

1. mKCP 将数据流拆成若干个数据包进行发送。一个数据流有一个唯一标识，用以区分不同的数据流。数据流中的每一个数据包都携带了同样的标识。
2. mKCP 没有握手过程，当收到一个数据包时，根据其携带的数据流的标识来判断是否为新的通话，或是正在进行中的通话。
3. 每一个数据包中包含若干个片段（Segment），片段分为三类：数据（Data）、确认（ACK）、心跳（Ping）。每个片段需要单独处理。

数据格式

数据包

4 字节	**2** 字节	**L** 字节
认证信息 A	数据长度 L	片段部分

其中：

- 认证信息 A = fnv(片段部分)，big endian；
- 片段部分可能包含多个片段；

数据片段

2 字节	**1** 字节	**1** 字节	**4** 字节	**4** 字节	**4** 字节	**2** 字节	**Len** 字节
标识 Conv	指令 Cmd	选项 Opt	时间戳 Ts	序列号 Sn	未确认序列号 Una	长度 Len	数据

其中：

- 标识 Conv: mKCP 数据流的标识
- 指令 Cmd: 常量 0x01
- 选项 Opt: 可选的值有：
  - 0x00: 空选项
  - 0x01: 对方已发出所有数据
- 时间戳 Ts: 当前片段从远端发送出来时的时间，big endian
- 序列号 Sn: 该数据片段时数据流中的位置，起始片段的序列号为 0，之后每个新片段按顺序加 1
- 未确认序列号 Una: 远端主机正在发送的，且尚未收到确认的最小的 Sn

确认片段

2 字节	**1** 字节	**1** 字节	**4** 字节	**4** 字节	**4** 字节	**2** 字节	**Len * 4** 字节
标识 Conv	指令 Cmd	选项 Opt	窗口 Wnd	下一接收序列号 Sn	时间戳 Ts	长度 Len	已收到的序列号

其中：

- 标识 Conv: mKCP 数据流的标识
- 指令 Cmd: 常量 0x00
- 选项 Opt: 同上
- 窗口 Wnd: 远端主机可以接收的最大序列号
- 下一接收序列号 Sn: 远端主机未收到的数据片段中的最小序列号
- 时间戳 Ts: 远端主机最新收到的数据片段的时间戳，可用于计算延迟
- 已收到的序列号: 每个 4 字节，表示此序列号的数据已经确认收到

注释：

- 远程主机期待收到序列号 [Sn, Wnd) 范围内的数据

心跳片段

2 字节	**1** 字节	**1** 字节	**4** 字节	**4** 字节	**4** 字节
标识 Conv	指令 Cmd	选项 Opt	未确认序列号 Una	下一接收序列号 Sn	延迟 Rto

其中：

- 标识 Conv: mKCP 数据流的标识
- 指令 Cmd: 可选的值有
  - 0x02: 远端主机强行终止会话
  - 0x03: 正常心跳
- 选项 Opt: 同上
- 未确认序列号 Una: 同数据片段的 Una
- 下一接收序列号 Sn: 同确认片段的 Sn
- 延迟 Rto: 远端主机自己计算出的延迟

Mux.Cool 协议

Mux.Cool 协议是一个多路复用传输协议，用于在一条已建立的数据流中传输多个各自独立的数据流。

版本

当前版本是 1 Beta。

依赖

底层协议

Mux.Cool 必须运行在一个已建立的可靠数据流之上。

通讯过程

一个 Mux.Cool 连接中可传输若干个子连接，每个子连接有一个独立的 ID 和状态。传输过程由帧（Frame）组成，每一帧用于传输一个特定的子连接的数据。

客户端行为

当有连接需求时并且没有现有可用的连接时，客户端向服务器发起一个新连接，以下称为“主连接”。

1. 一个主连接可用于发送若干个子连接。客户端可自主决定主连接可承载的子连接数量。
2. 对于一个新的子连接，客户端必须发送状态 `New` 以通知服务器建立子连接，然后使用状态 `Keep` 来传送数据。
3. 当子连接结束时，客户端发送 `End` 状态来通知服务器关闭子连接。
4. 客户端可自行决定何时关闭主连接，但必须确定服务器也同时保持连接。
5. 客户端可使用 KeepAlive 状态来避免服务器关闭主连接。

服务器端行为

当服务器端接收到新的子连接时，服务器应当按正常的连接来处理。

1. 当收到状态 `End` 时，服务器端可以关闭对目标地址的上行连接。
2. 在服务器的响应中，必须使用与请求相同的 ID 来传输子连接的数据。
3. 服务器不能使用 `New` 状态。
4. 服务器可使用 KeepAlive 状态来避免客户端关闭主连接。

传输格式

Mux.Cool 使用对称传输格式，即客户端和服务器发送和接收相同格式的数据。

帧格式

2 字节	**L** 字节	**X** 字节
元数据长度 L	元数据	额外数据

元数据

元数据有若干种类型，由状态 S 来区分。所有类型的元数据都包含 ID 和 Opt 两项，其含义为：

- ID: 子连接的唯一标识
- Opt:
  - D(0x01): 有额外数据

当选项 Opt(D) 开启时，额外数据格式如下：

2 字节	**L** 字节
长度 L	数据

新建子连接 **(New)**

2 字节	**1** 字节	**1** 字节	**1** 字节	**2** 字节	**1** 字节	**X** 字节
ID	0x01	选项 Opt	网络类型 N	端口	地址类型 T	地址 A

其中：

- 网络类型 N：
  - 0x01：TCP，表示当前子连接的流量应当以 TCP 的方式发送至目标。
  - 0x02：UDP，表示当前子连接的流量应当以 UDP 的方式发送至目标。
- 地址类型 T:
  - 0x01：IPv4
  - 0x02：域名
  - 0x03：IPv6
- 地址 A：
  - 当 T = 0x01 时，A 为 4 字节 IPv4 地址；
  - 当 T = 0x02 时，A 为 1 字节长度（L） + L 字节域名；
  - 当 T = 0x03 时，A 为 16 字节 IPv6 地址；

在新建子连接时，若 Opt(D) 开启，则这一帧所带的数据需要被发往目标主机。

保持子连接 **(Keep)**

2 字节	**1** 字节	**1** 字节
ID	0x02	选项 Opt

在保持子连接时，若 Opt(D) 开启，则这一帧所带的数据需要被发往目标主机。

关闭子连接 **(End)**

2 字节	**1** 字节	**1** 字节
ID	0x03	选项 Opt

在保持子连接时，若 Opt(D) 开启，则这一帧所带的数据需要被发往目标主机。

保持连接 **(KeepAlive)**

2 字节	**1** 字节	**1** 字节
ID	0x04	选项 Opt

在保持连接时:

- 若 Opt(D) 开启，则这一帧所带的数据必须被丢弃。
- ID 可为随机值。

应用

Mux.Cool 协议与底层协议无关，理论上可以使用任何可靠的流式连接来传输 Mux.Cool 的协议数据。

在目标导向的协议如 Shadowsocks 和 VMess 协议中，连接建立时必须包含一个指定的地址。为了保持兼容性，Mux.Cool 协议指定地址为“v1.mux.cool”。即当主连接的目标地址与之匹配时，则进行 Mux.Cool 方式的转发，否则按传统方式进行转发。

修订历史

2017.04.12 重构页面，加入 KeepAlive 状态 2017.04.03 初版