

HTTP 协议探究，基于 Node.js 实战编写 HTTP Server

吴阳



课程目的

- 了解 HTTP Server 的工作方式
- 熟悉 HTTP/1.1 协议结构
- 熟悉 HTTP/1.1 协议中常见 Header 和 Status Code 的含义
- 熟悉 RESTful 约定
- 亲自实现一个基于 RESTful 约定的 HTTP Server
- 为自己的 Server 实现以下三个特性：
CORS (跨域资源共享)
Cookie / Session
浏览器缓存控制

如果你满足以下条件...

- 参与项目开发工作有一段时间
- 有一定的 ES5+ 语法基础
- 有 XHR API 或 Fetch API 使用经验
- 对 HTTP 原理不是很熟悉

课程安排

DAY 1

- OSI 参考模型和 TCP/IP 五层模型
- 学习 HTTP/1.1 协议
- 了解 RESTful 约定
- 设计一个 HTTP Server (静态服务)
- 完成 HTTP Server 主体框架代码

DAY 2

- 编写模块处理符合 RESTful 约定的请求
(GET POST PUT DELETE)
- 修改 GET 模块实现部分内容获取 (断点下载)

DAY 3

- 编写模块实现 HTTP 鉴权(Session 和 Cookie)
- 编写模块支持浏览器跨域访问
- 编写模块支持浏览器缓存控制

我们需要准备...

- Node.js (演示使用 14.17.0 版本)
- Postman
- Chrome
- Visual Studio Code 或任何你喜欢的编辑器
- 任何可以运行上述软件的操作系统

DAY 1

目 录

- OSI 参考模型和 TCP/IP 五层模型
- HTTP/1.1 协议
- 请求和响应
- RESTful 约定
- Node.js 的 net 模块和流
- 设计一个 HTTP Server
- 完成 HTTP Server 主体框架代码

预备知识

OSI 参考模型和 TCP 五层模型

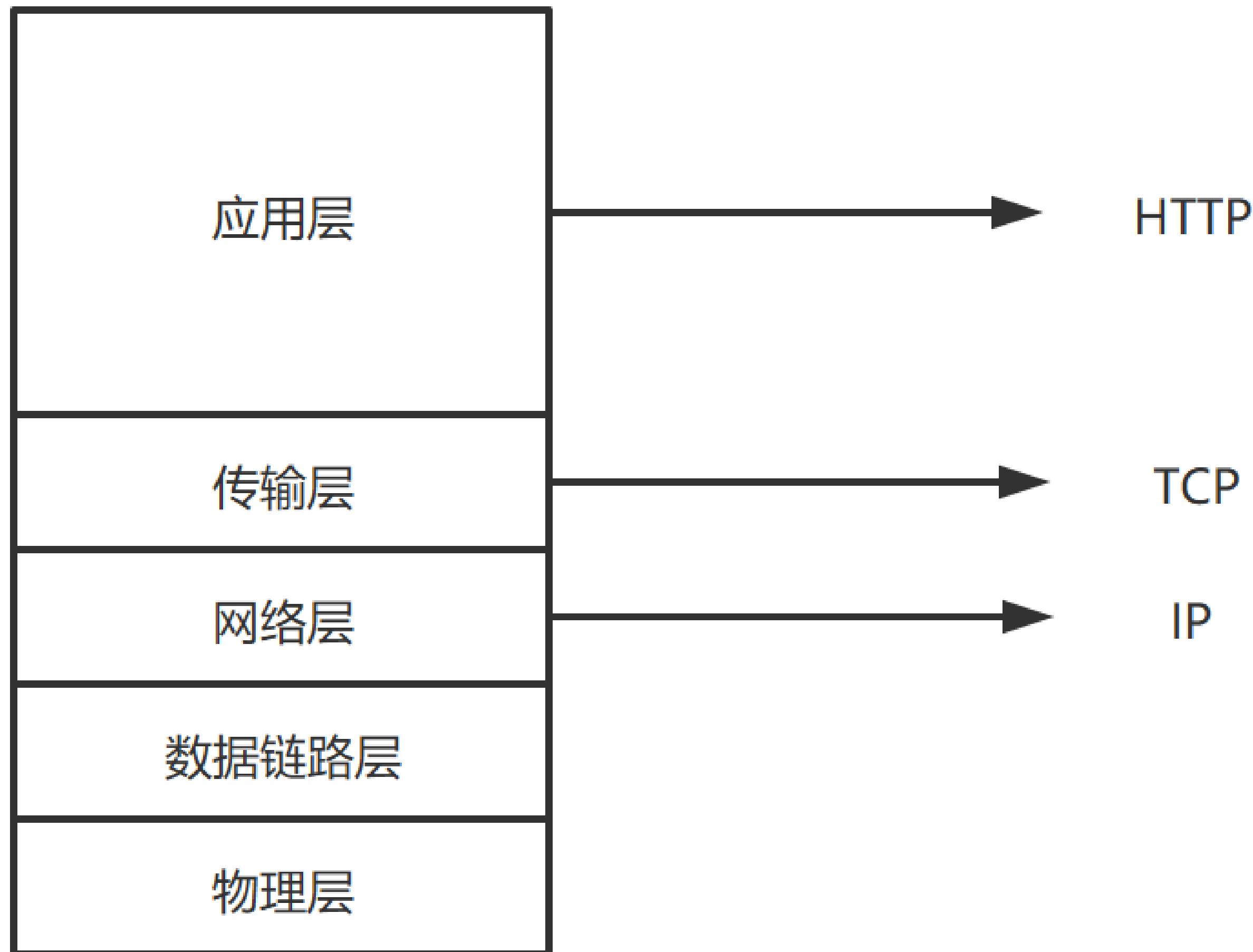


OSI 参考模型



TCP/IP 五层模型

HTTP/1.1 协议



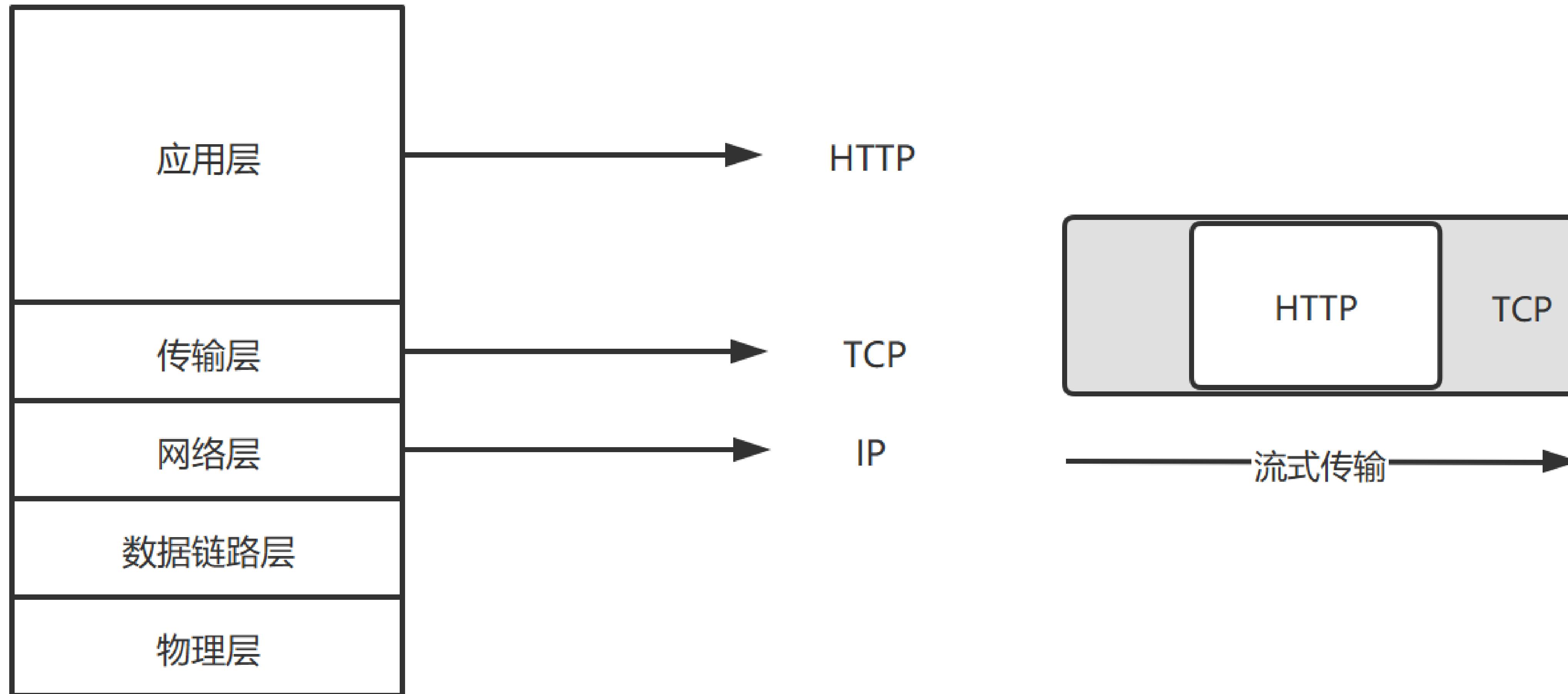
HTTP/1.1 协议特点

- 工作在 TCP 协议之上
- 约定使用文本传输
- 半双工通信
- Request 和 Response 交替成对出现 (1.1)
- 简单、可扩展、无状态

TCP 协议特点

- 工作在 IP 协议之上
- 约定使用字节流传输
- 可以全双工通信
- 建立连接需要三次握手
- 断开连接需要四次握手

HTTP/1.1 协议



HTTP/1.1 协议

参考资料：

<https://datatracker.ietf.org/doc/html/rfc2068> (百度搜索: RFC2068)

<https://developer.mozilla.org/en-US/docs/Web/HTTP> (百度搜索: MDN HTTP)

HTTP/1.1 协议 (RFC 2068)

(4.1)

```
HTTP-message = Request | Response ; HTTP/1.1 messages
```

(4.1)

```
generic-message = start-line  
                  *message-header  
                  CRLF  
                  [ message-body ]  
  
start-line      = Request-Line | Status-Line
```

(5)

```
Request          = Request-Line  
                  *( general-header  
                  | request-header  
                  | entity-header )  
                  CRLF  
                  [ message-body ]
```

(4.2)

```
message-header = field-name ":" [ field-value ] CRLF  
  
field-name     = token  
field-value    = *( field-content | LWS )  
  
field-content  = <the OCTETs making up the field-value  
                  and consisting of either *TEXT or combinations  
                  of token, tspecials, and quoted-string>
```

(6)

```
Response         = Status-Line  
                  *( general-header  
                  | response-header  
                  | entity-header )  
                  CRLF  
                  [ message-body ]
```

(5.1)

```
Request-Line    = Method SP Request-URI SP HTTP-Version CRLF
```

(6.1)

```
Status-Line     = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

HTTP/1.1 协议

Request Headers View parsed

```
Request-Line = Method SP Request-URI SP HTTP-Version CRLF
GET /doc/html/rfc2068 HTTP/1.1
Host: datatracker.ietf.org
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
sec-ch-ua: " Not;A Brand";v="99", "Google Chrome";v="91", "Chromium"
sec-ch-ua-mobile: ?0
Upgrade-Insecure-Requests: 1
```

Request = Request-Line ;
* (general-header ;
| request-header ;
| entity-header) ;
CRLF
[message-body] ;

Response Headers View parsed

```
Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
HTTP/1.1 200 OK
Date: Fri, 18 Jun 2021 13:29:21 GMT
Server: Apache
Strict-Transport-Security: max-age=3600; includeSubDomains
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
X-Frame-Options: ALLOW-FROM TFTF.ORG *.TFTF.ORG MFFTFCHO.COM *
```

Response = Status-Line ;
* (general-header ;
| response-header ;
| entity-header) ;
CRLF
[message-body] ;

本教程会涉及到 HTTP/1.1 的以下内容

Method:

GET POST PUT DELETE OPTIONS

Status Code:

200 201 206 304 400 401 403 404 500

Headers:

Content-Type Content-Length Range Cookie Authorization Cache-Control
ETag

If-None-Match Content-Range WWW-Authenticate Last-Modified

Access-Control-Allow-Origin Access-Control-Allow-Method Access-Control-Allow-Credential

RESTful 约定

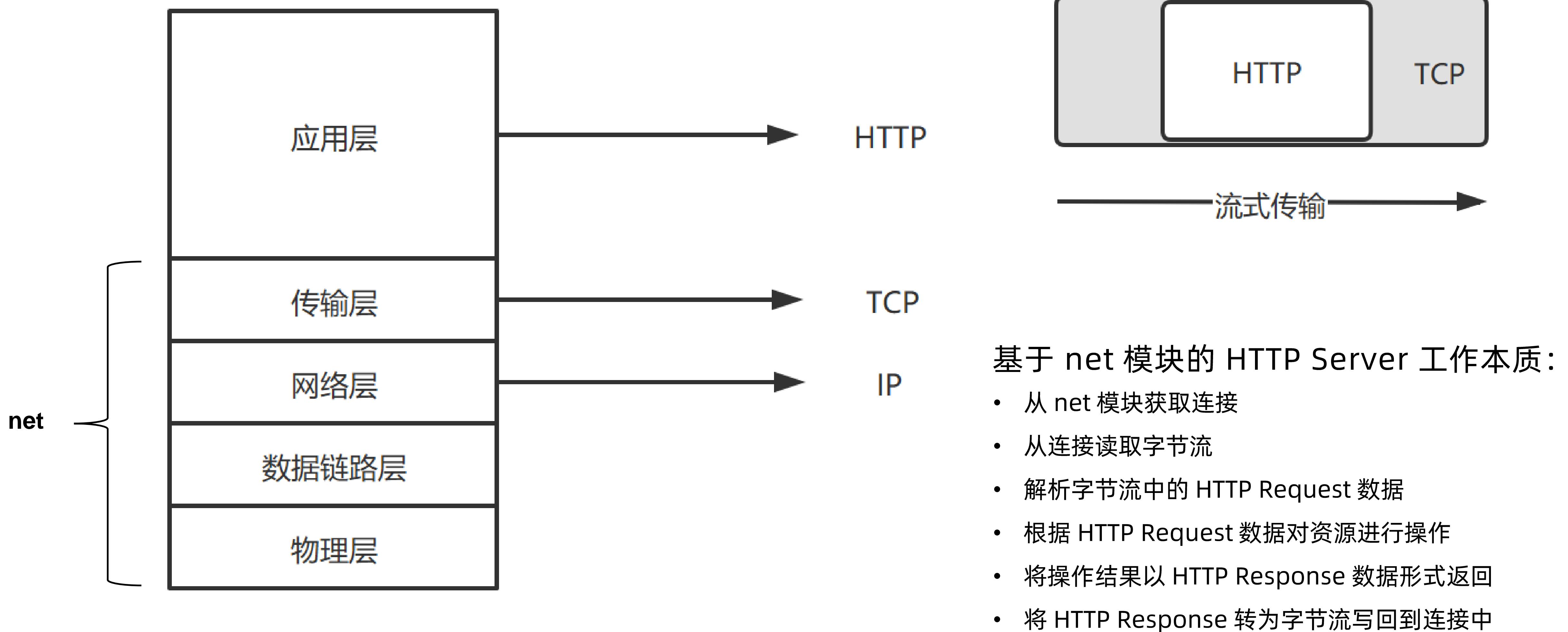
主要原则

- 所有的操作都是对资源操作
- Method 只能用于描述对资源的操作方式
- CURD 操作对应为 POST、PUT、GET 和 DELETE
- URI 只能用于描述资源的位置
- 使用 Status Code 来描述对资源的处理结果

举例

POST /users/username	创建 /users/username
PUT /users/username	更新 /users/username
GET /users/username	获取 /users/username
DELETE /users/username	删除 /users/username

Node.js 的 net 模块

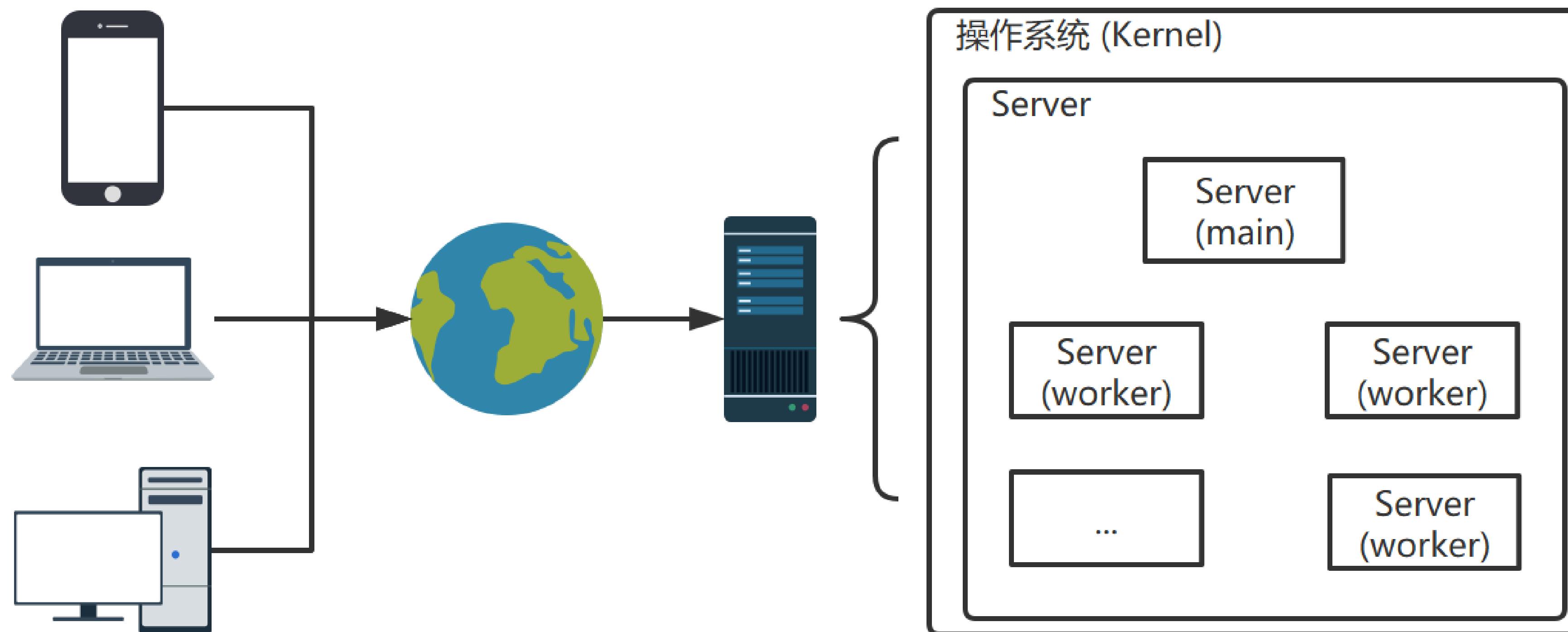


设计 HTTP Server

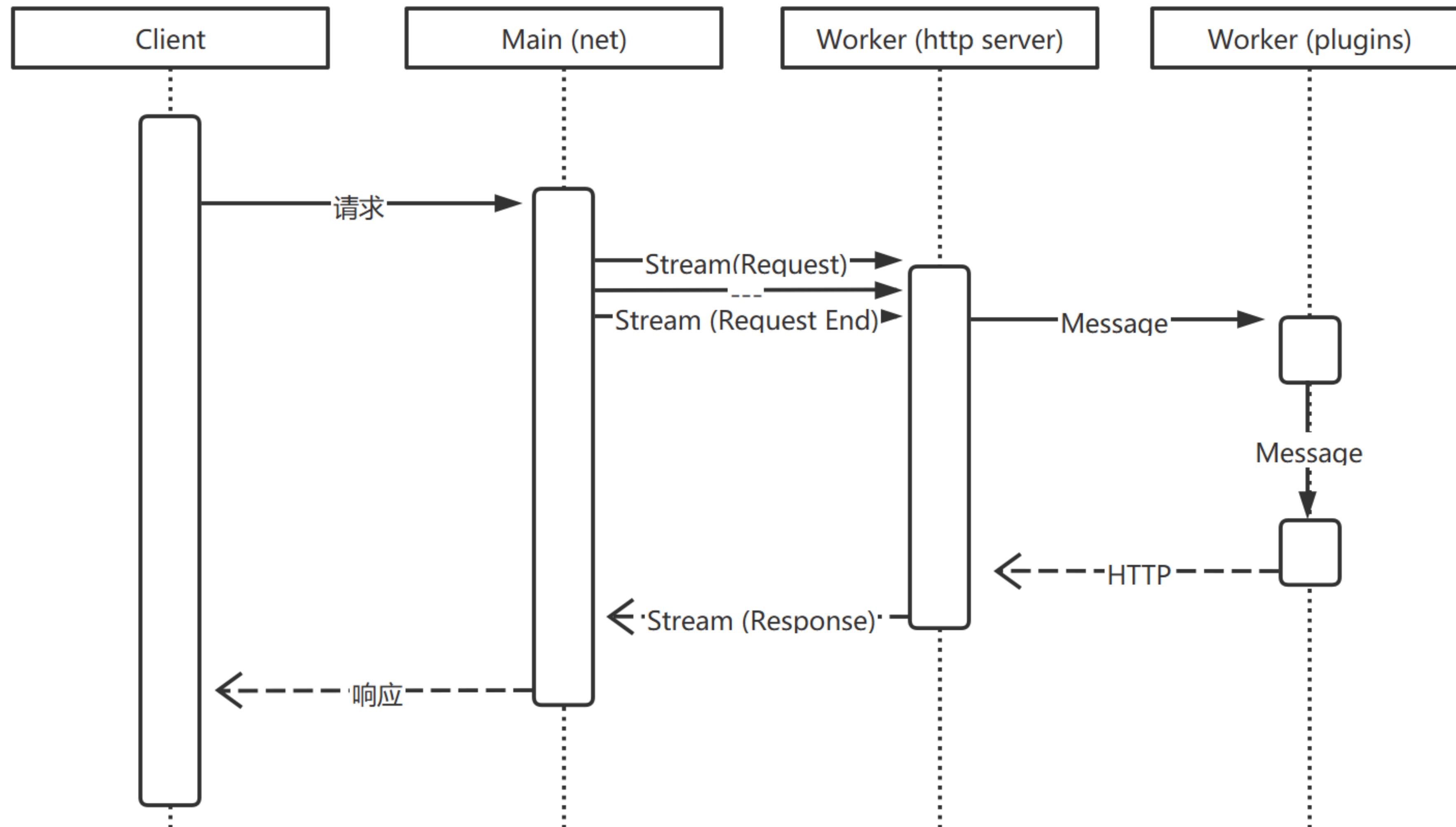
设计一个 HTTP Server (功能要求)

- 资源下载 (全量, 任意范围)
- 资源上传 (新建, 覆盖)
- 资源删除
- 可以自由访问目录里的资源
- 有权限认证
- 资源可以被浏览器缓存
- 可以被浏览器跨域访问

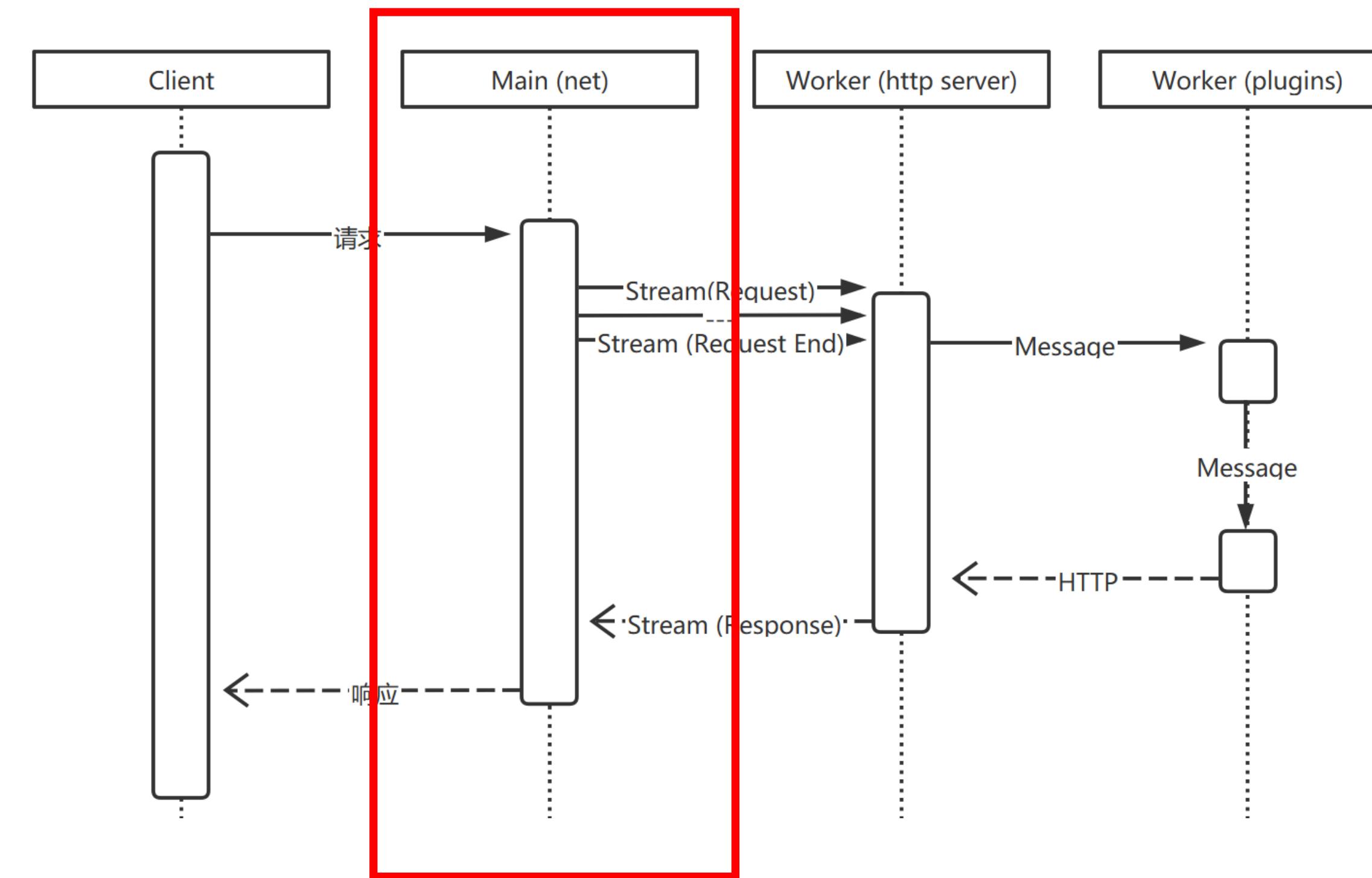
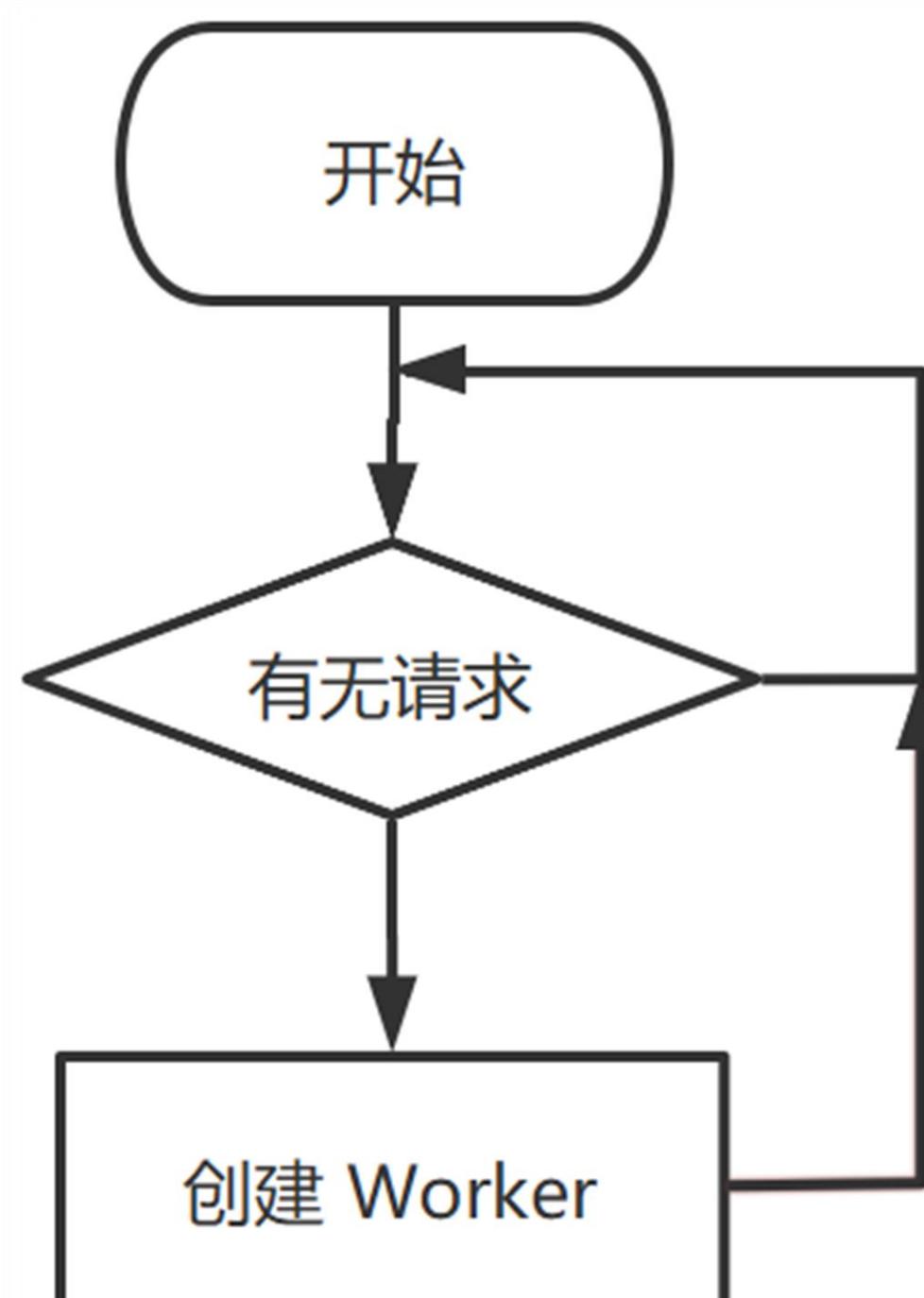
设计一个 HTTP Server (工作原理)



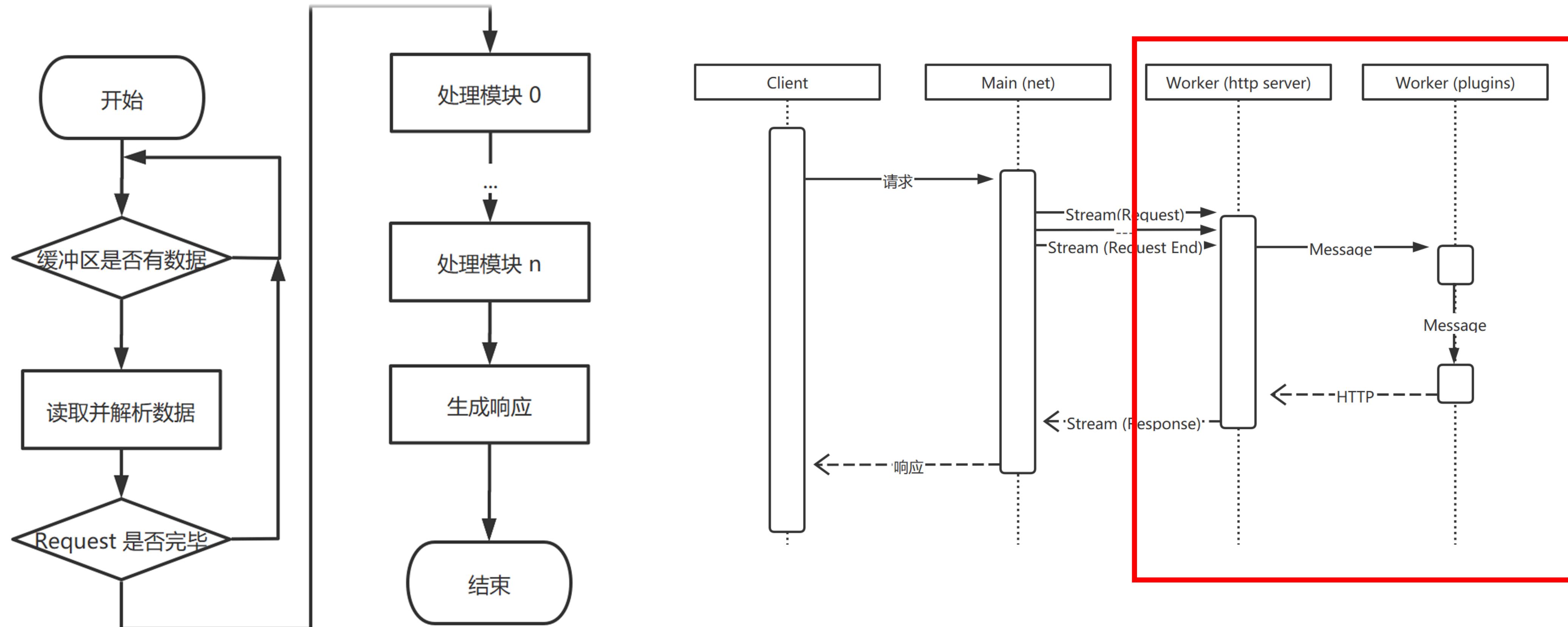
设计一个 HTTP Server (总体设计)



设计一个 HTTP Server (Main)

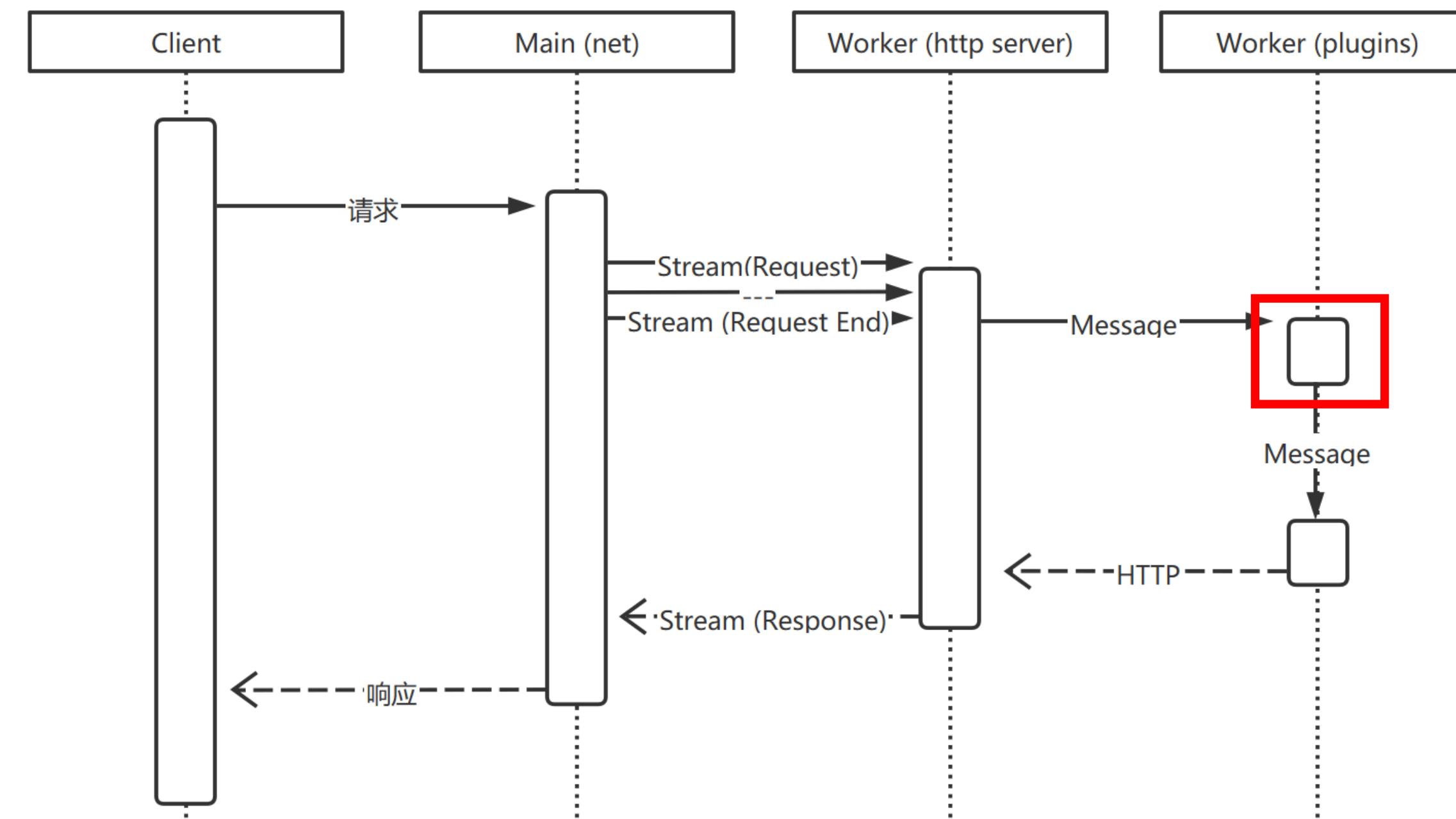
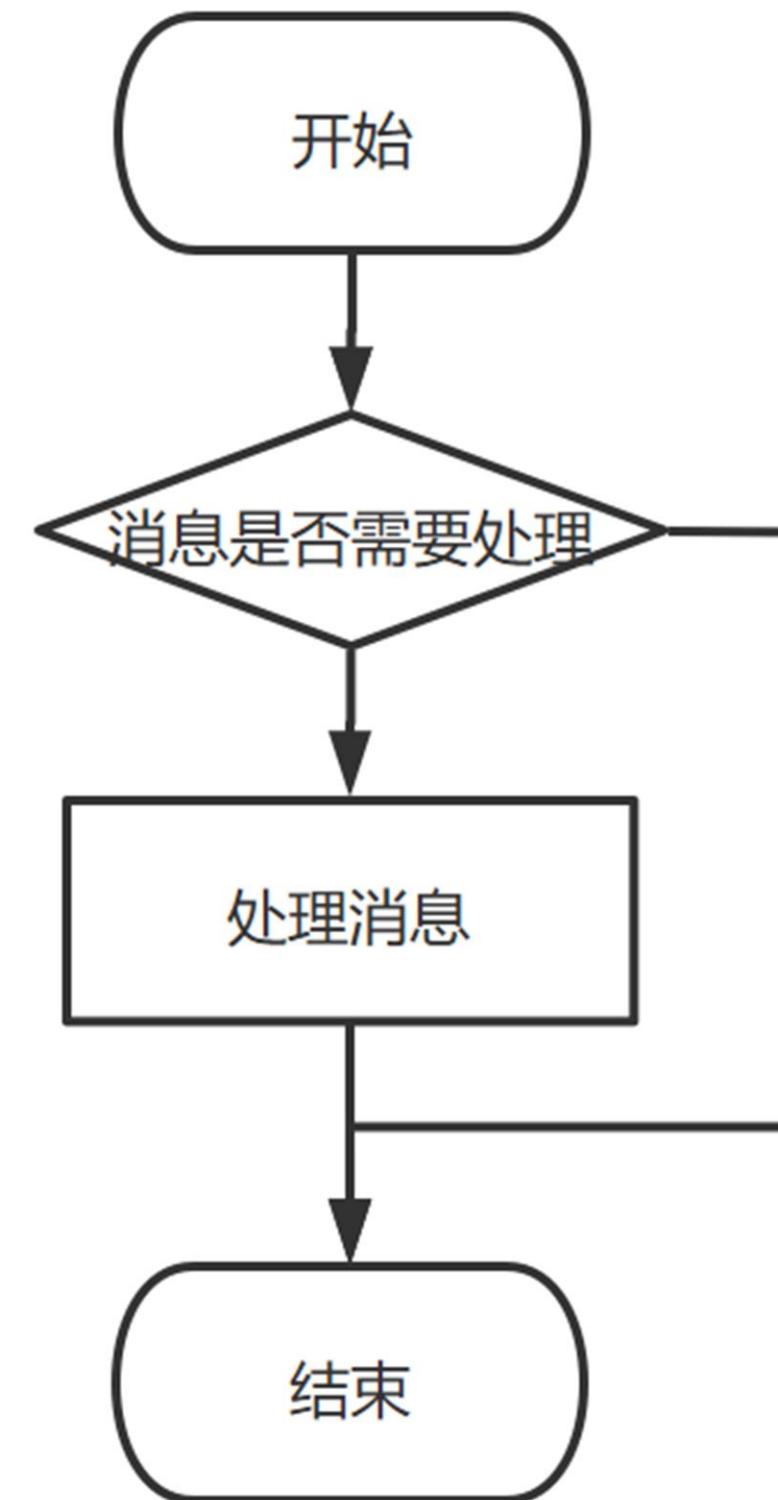


设计一个 HTTP Server (Worker)



管道-过滤器 架构模式

设计一个 HTTP Server (plugin)



实现 HTTP Server

Status Code: 500

HTTP/1.1 500 Internal Server Error

服务器遇到了不知道如何处理的情况。

在还没有完成任何一个处理模块时，我们使用此状态作为默认状态。

来表示我们的 Server 不知道如何处理这个请求

Header: Content-Type

用于描述 HTTP Message 中 Message Body 的类型。

见于 Request 和 Response 之中

其内容一般是 MIME 字符串，用于描述处理 Message Body 的方式

例如 text/html 就用于描述 Message Body 中传输的内容是 html 代码

在 Response 中使用 Content-Type 可以告知客户端如何处理 Response Body

Header: Content-Length

用于描述 HTTP Message 中 Message Body 的长度。

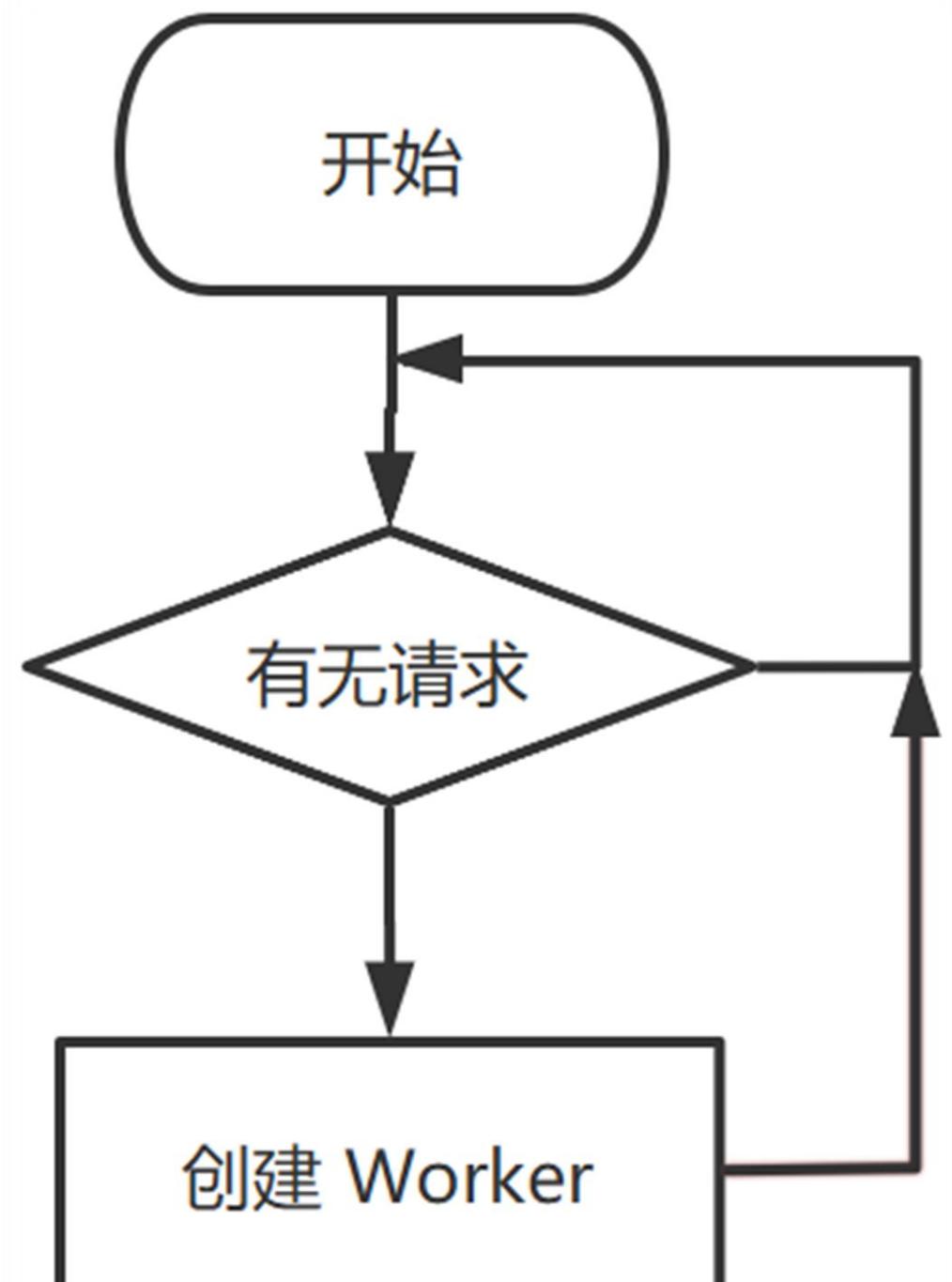
见于 Request 和 Response 之中

单位为 8-bit Byte (Octet)

在 Response 中使用 Content-Length 可以告知客户端
如何处理 Response Body 到哪里结束

在 Request 中使用 Content-Length 可以告知服务端请求在哪里结束

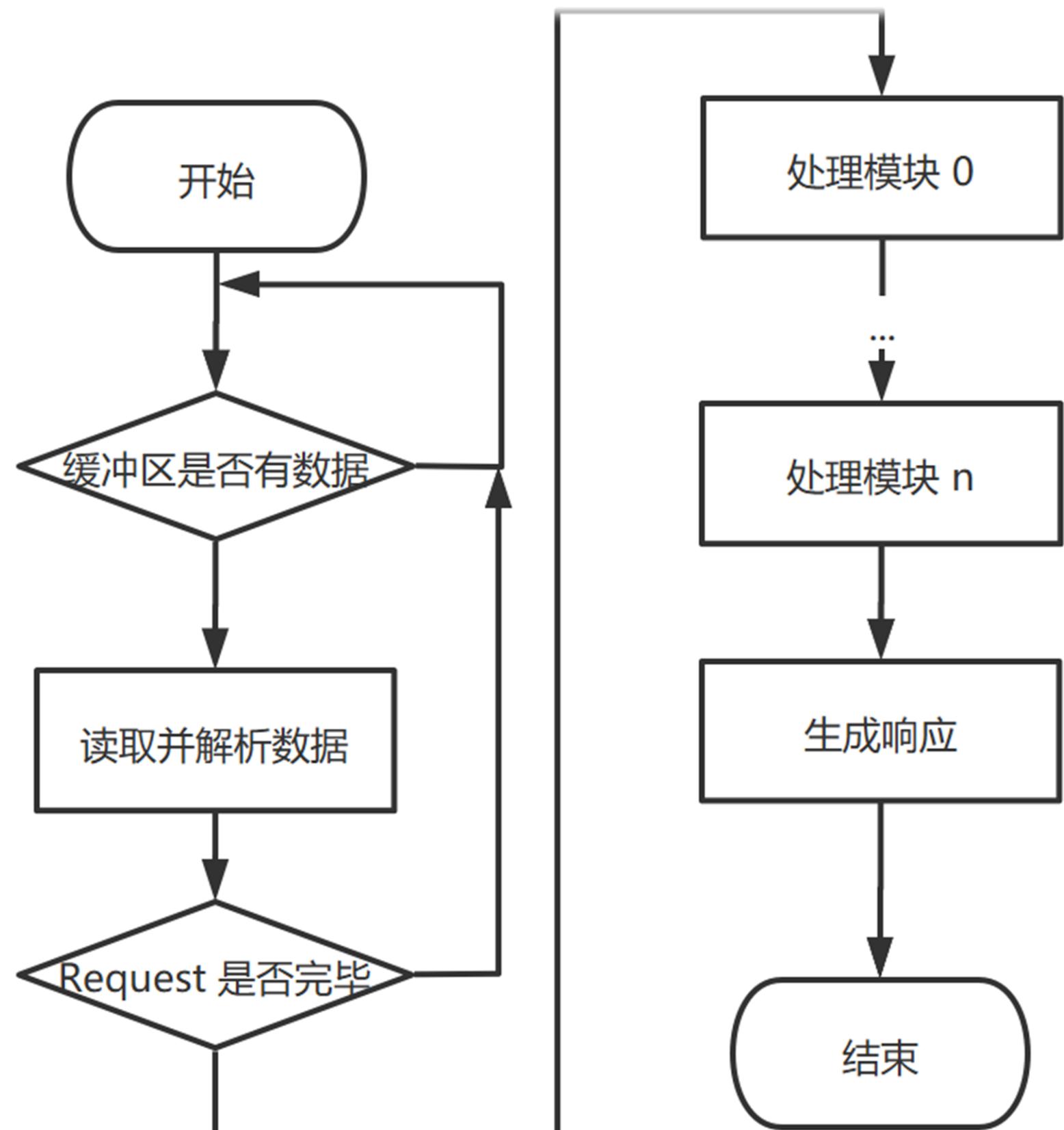
实现 HTTP Server (Main)



新知识点

- `net.createServer`
- 环境参数

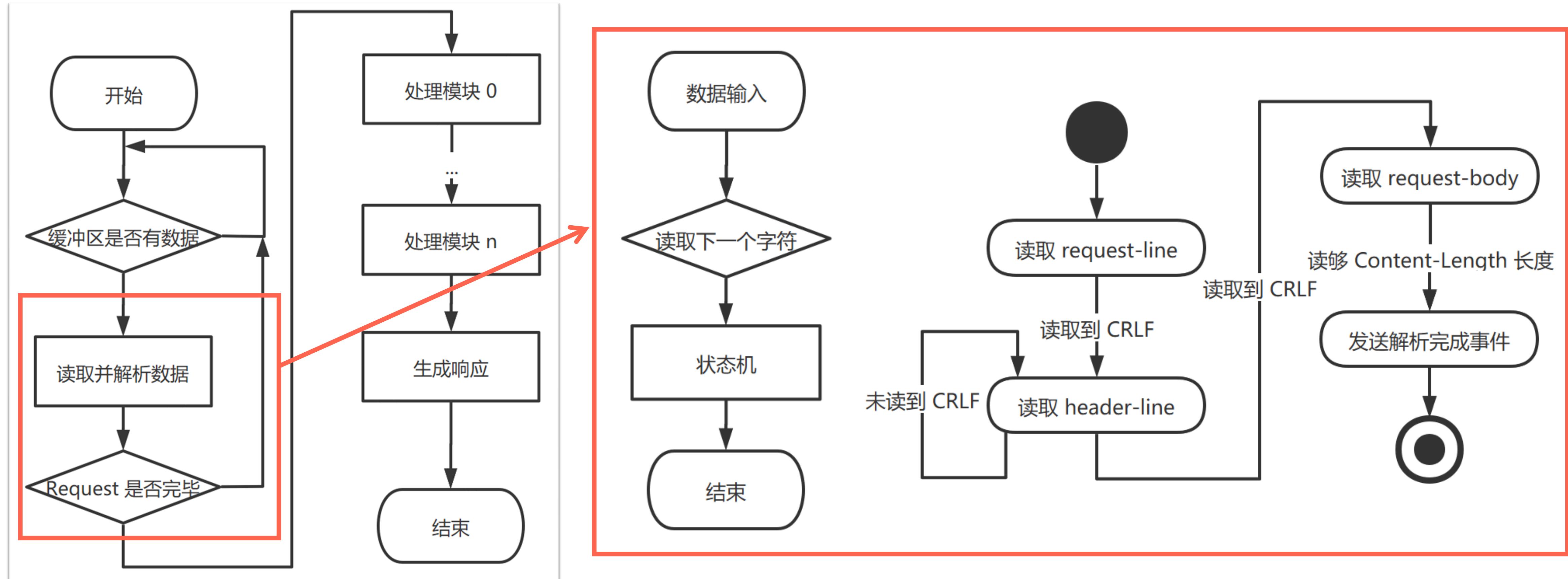
实现 HTTP Server (Worker)



新知识点

- 有限状态机
- Request 解析
- Events. Emit
- Response 生成
- Status Code: 500
- Content-Length
- Content-Type
- MIME

实现 HTTP Server (Request 解析)



作业和练习

1. 根据本次课程中的时序图和流程图完成 HTTP Server 主体部分代码
2. 启动完成的 HTTP Server，并使用浏览器访问

THANKS

