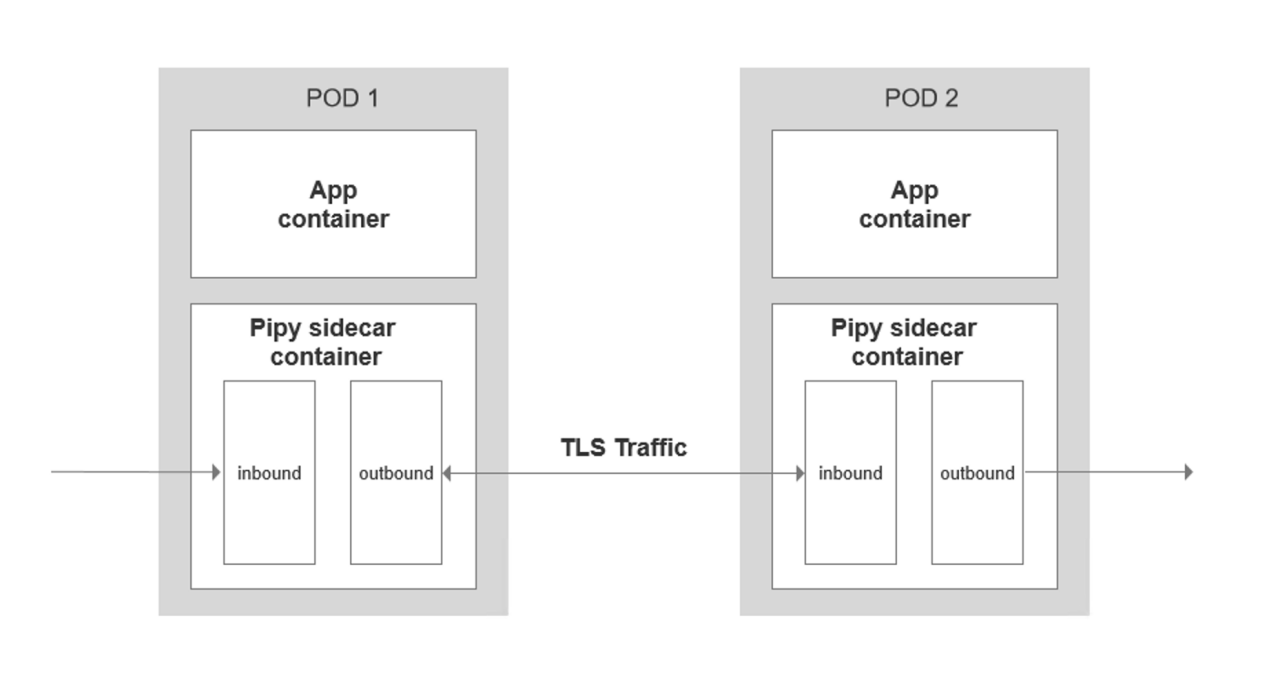
# Pipy sidecar 模块设计与插件开发说明

### 概述

Pipy 是一款可编程流量引擎，使用 pipy js 可以灵活、简洁的实现各种需求。

Pipy 在服务网格中，是以sidecar的方式来实现对 Pod 网络的入站流量、出站流量进行管理。

示意图如下：



Pipy 对流量处理，大体有如下几个方面：

* 对流量进行加密、解密处理
* 对流量进行压缩、解压处理
* 对各种应用协议进行 Codec, 实现网络Data 和 应用 Message 的相互转换
* 对流量进行网络转发、实现连接的多路复用等

### 模块设计

Pipy js 提供了基于命名空间（namespace）的变量导入/导出 （import/export），使得可以将一个比较大的功能拆分成多个独立的子模块，模块之间使用变量进行交互。

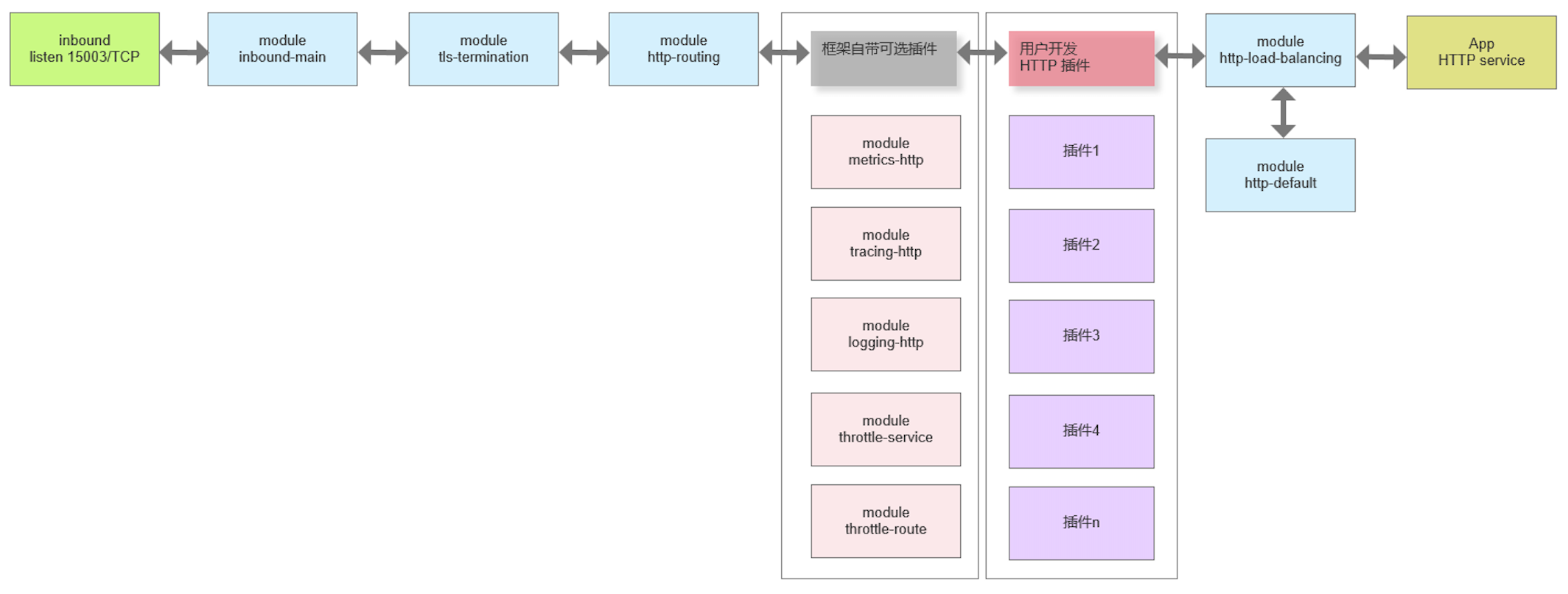
Inbound、Outbound 分别支持 HTTP 应用（L7）、TCP应用（L4）两种场景。

1. inbound 的模块划分

* Inbound HTTP 应用（L7）包含如下几个核心模块：

|  |  |
| --- | --- |
| **模块名称** | **模块作用** |
| inbound-main | 入站流量主模块，执行初始化操作 |
| inbound-tls-termination | 将 TLS加密流量解码成明文流量 |
| inbound-http-routing | 将 tcp 流量解码成 HTTP 消息  并执行路由策略 |
| inbound-http-load-balancing | 对请求执行负载均衡策略，  并访问本地服务 |
| inbound-http-default | 异常时返回默认数据 |

完整的模块结构图如下：

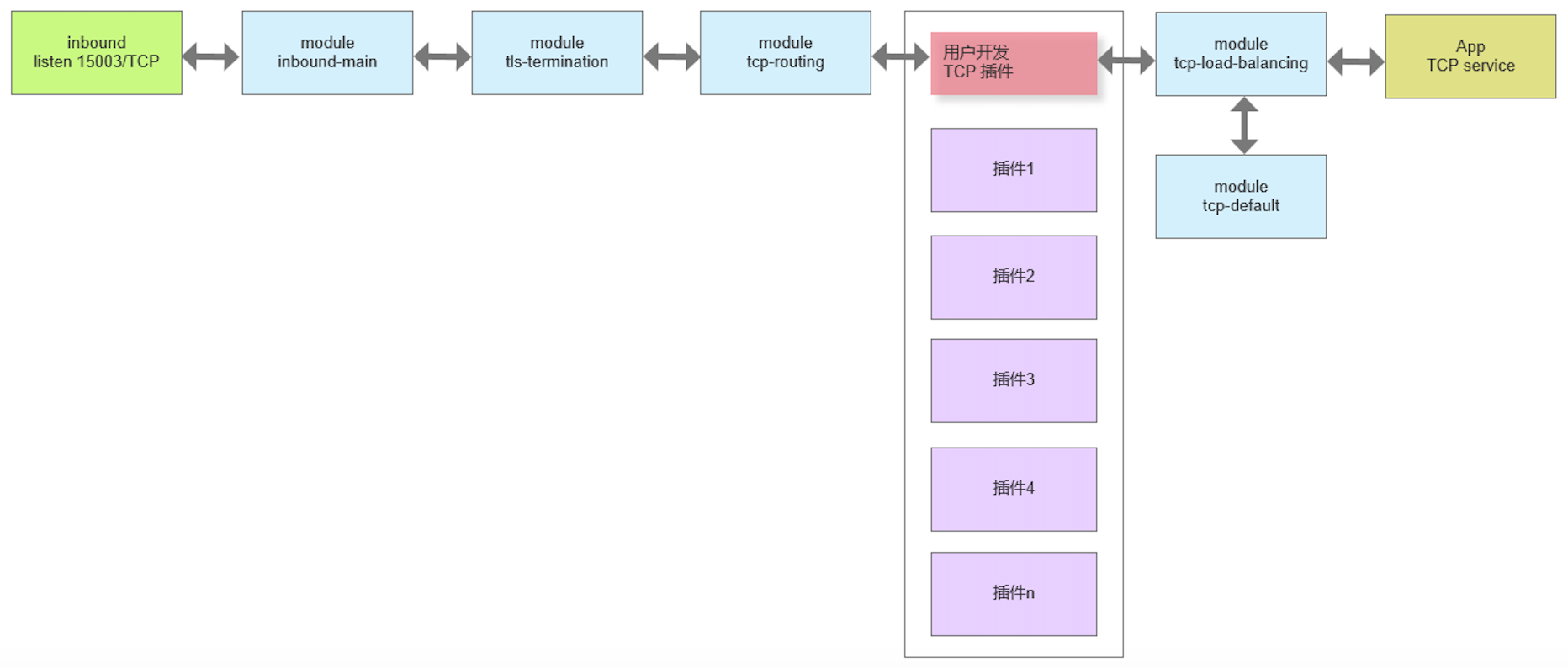


其中：浅蓝色模块是必须的基础模块、橙色为框架提供的可选模块、紫色为用户开发模块。

* Inbound TCP 应用（L4）包含如下几个核心模块：

|  |  |
| --- | --- |
| **模块名称** | **模块作用** |
| inbound-main | 入站流量主模块，执行初始化操作 |
| inbound-tls-termination | 将 TLS加密流量解码成明文流量 |
| inbound-tcp-routing | 对 tcp 请求并执行路由策略 |
| inbound-tcp-load-balancing | 对请求执行负载均衡策略，  并访问本地服务 |
| inbound-tcp-default | 异常时返回默认数据 |

完整的模块结构图如下：



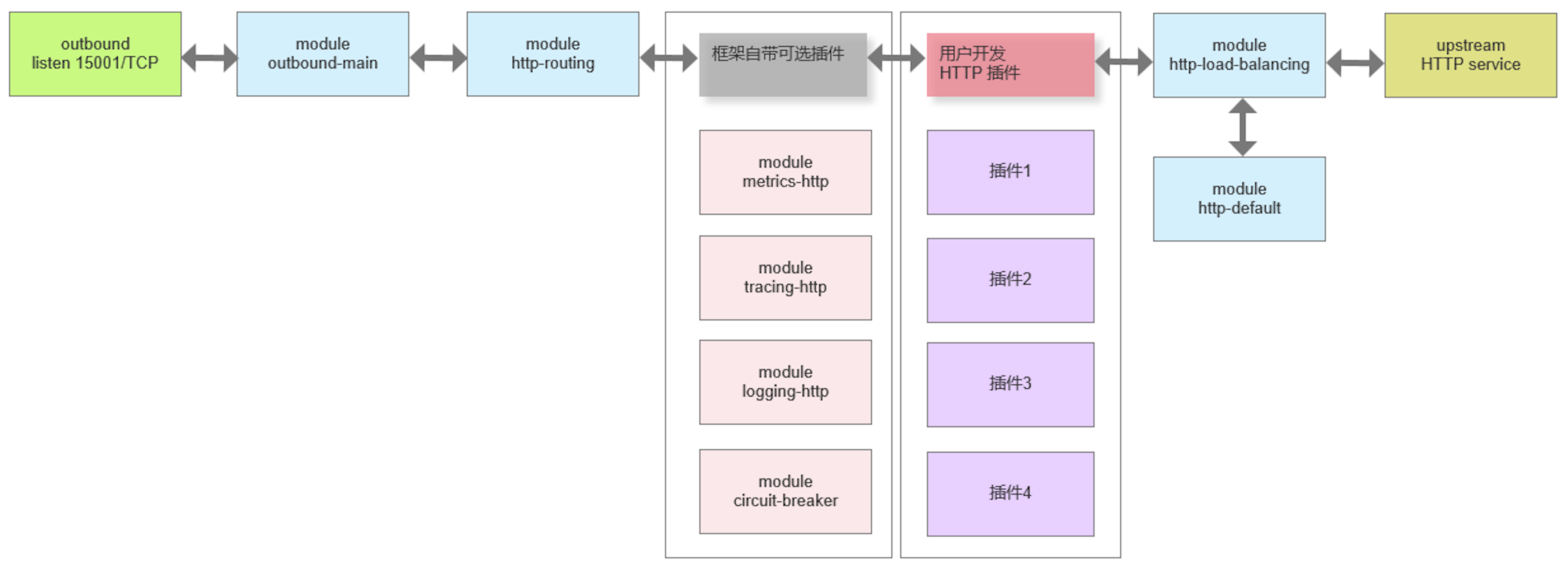
其中：浅蓝色模块是必须的基础模块、紫色为用户开发模块。

1. outbound 的模块划分

* Outbound HTTP 应用（L7）包含如下几个核心模块：

|  |  |
| --- | --- |
| **模块名称** | **模块作用** |
| outbound-main | 出站流量主模块，执行初始化操作 |
| outbound-http-routing | 将 tcp 流量解码成 HTTP 消息  并执行路由策略 |
| outbound-http-load-balancing | 对请求执行负载均衡策略，  并访问上游服务 |
| outbound-http-default | 异常时返回默认数据 |

完整的模块结构图如下：

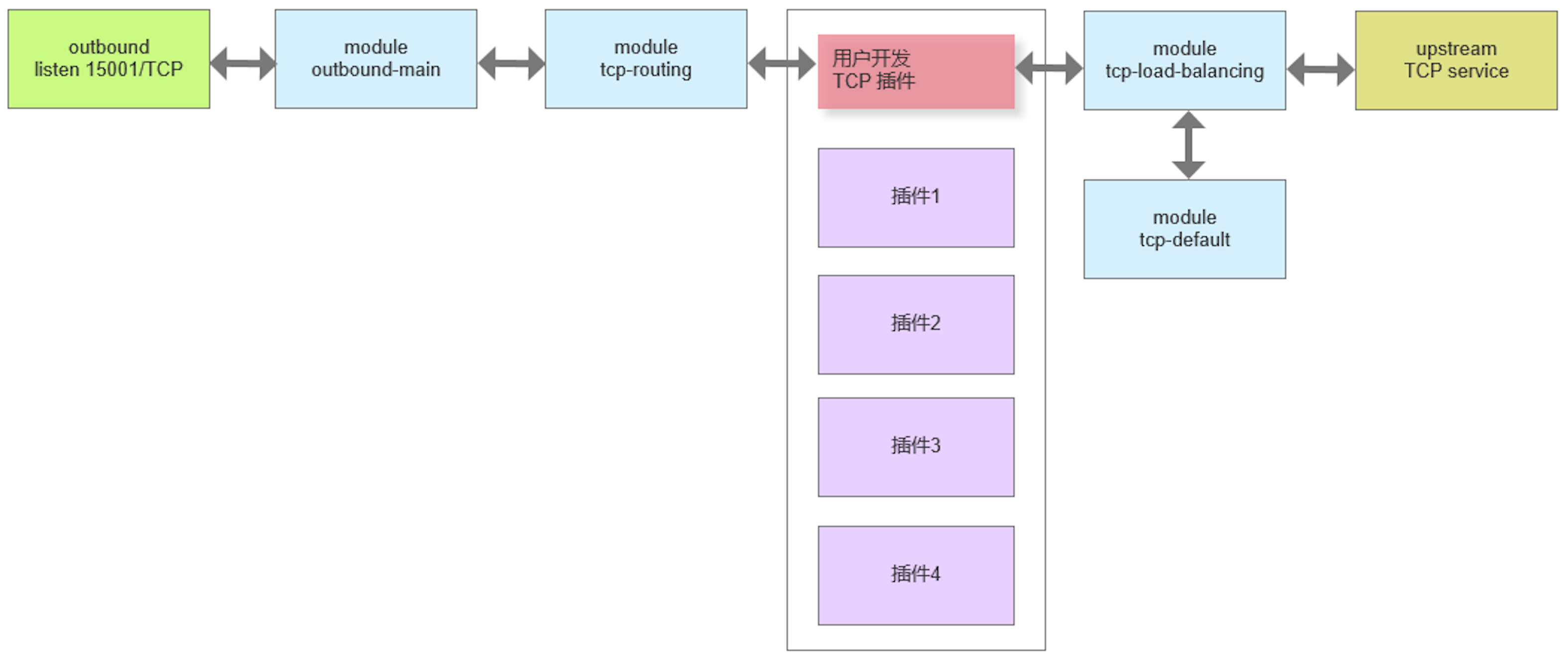


其中：浅蓝色模块是必须的基础模块、橙色为框架提供的可选模块、紫色为用户开发模块。

* Outbound TCP 应用（L4）包含如下几个核心模块：

|  |  |
| --- | --- |
| **模块名称** | **模块作用** |
| outbound-main | 出站流量主模块，执行初始化操作 |
| outbound-tcp-routing | 对 tcp 请求并执行路由策略 |
| outbound-tcp-load-balancing | 对请求执行负载均衡策略，  并访问上游服务 |
| outbound-tcp-default | 异常时返回默认数据 |

完整的模块结构图如下：



其中：浅蓝色模块是必须的基础模块、紫色为用户开发模块。

### 插件开发

1. 确定开发场景

* 确定插件是用于Pod的入站流量还是出站流量

入站流量，比如：校验用户访问令牌，就是 inbound 应用。

出站流量，比如：增加 tracing 字段，就是 outbound 应用。

* 确定插件是 HTTP 应用、TCP应用场景

目前涉及的应用基本都是 HTTP应用。

1. 基于事件模型编写插件业务逻辑

以HTTP应用为例，只需要在：

* onStart
* handleMessageStart
* replaceMessage

等方法里面，对 HTTP message 进行编程。

1. 如果插件不生成完整的response/应答，需要在插件中调用.chain() , 将请求转发到上游。

**插件开发演示**

1. 为请求增加访问令牌，插件代码：



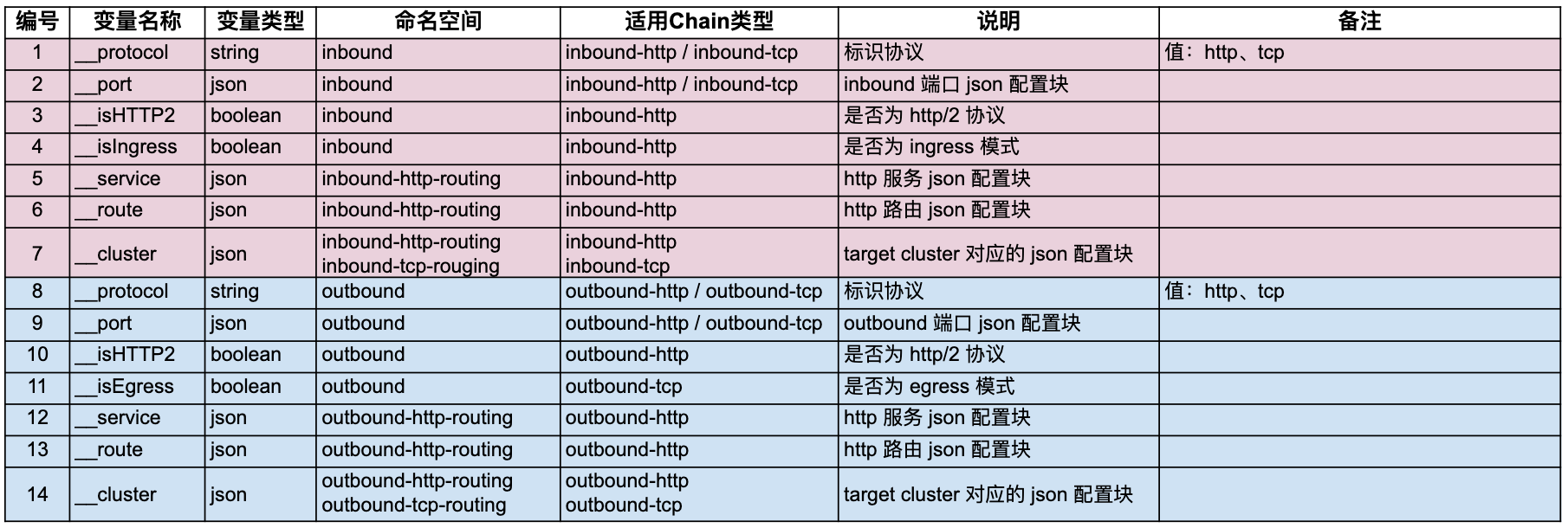
1. 验证请求中的令牌，插件代码：



### 插件中可使用的框架变量

插件需要 import 框架变量，来获取插件的配置/数据。

目前，框架提供的变量情况如下：



### 插件测试 demo

<https://github.com/cybwan/osm-edge-start-demo/blob/main/demo/plugin/README.zh.md>