



深入理解BFE



章淼

百度智能云
架构师

什么是BFE?

- 百度统一的七层流量转发平台
 - HTTP, HTTPS, HTTP/2, QUIC
- 2012年开始建设
- 每日转发请求约1万亿，日峰值超过1KW QPS
- 2019年，核心转发引擎对外开源
 - BFE => Beyond Front End
 - <https://github.com/bfenetworks/bfe>
- 2020年6月，成为CNCF Sandbox Project
- 2021年5月，《深入理解BFE》对外发布
 - <https://github.com/baidu/bfe-book>
 - 2021年Q3将由电子工业出版社正式出版



BFE开源项目公众号



GopherChina 2021

目 录

BFE涉及的相关技术原理

01

BFE的设计思想

02

BFE的实现机制

03

为什么需要BFE?

- 没有统一七层接入的问题

- 功能重复开发
- 运维成本高
- 流量统一控制能力低

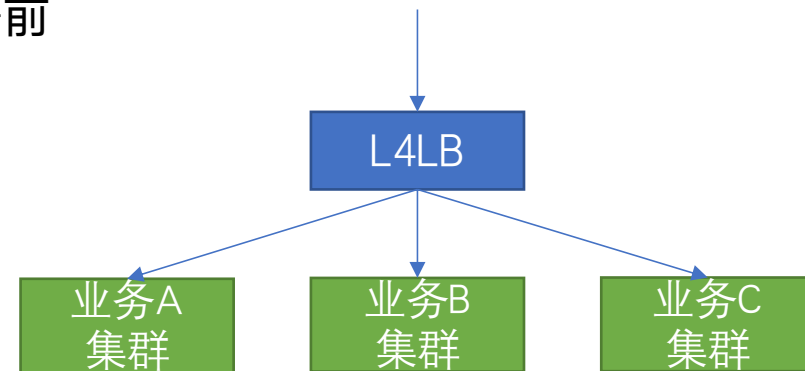
- 引入BFE后

- 功能统一开发
- 运维统一管理
- 流量控制能力增强

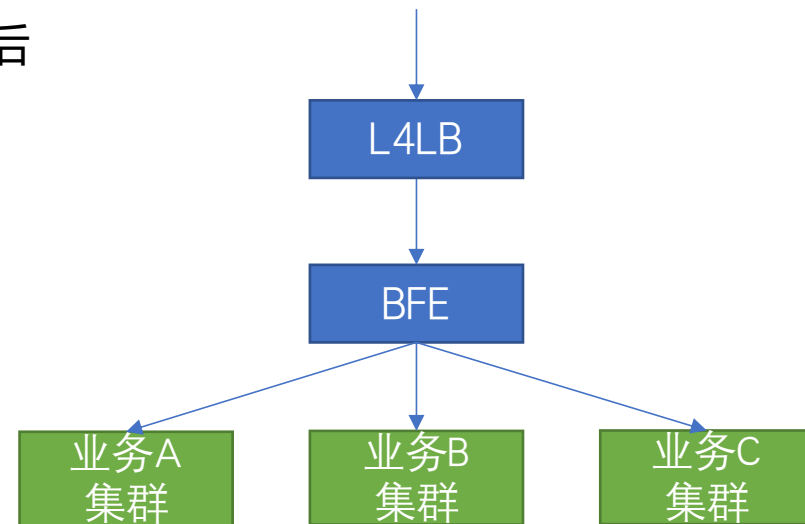
- BFE平台的主要功能

- 接入和转发，流量调度，安全防攻击，数据分析

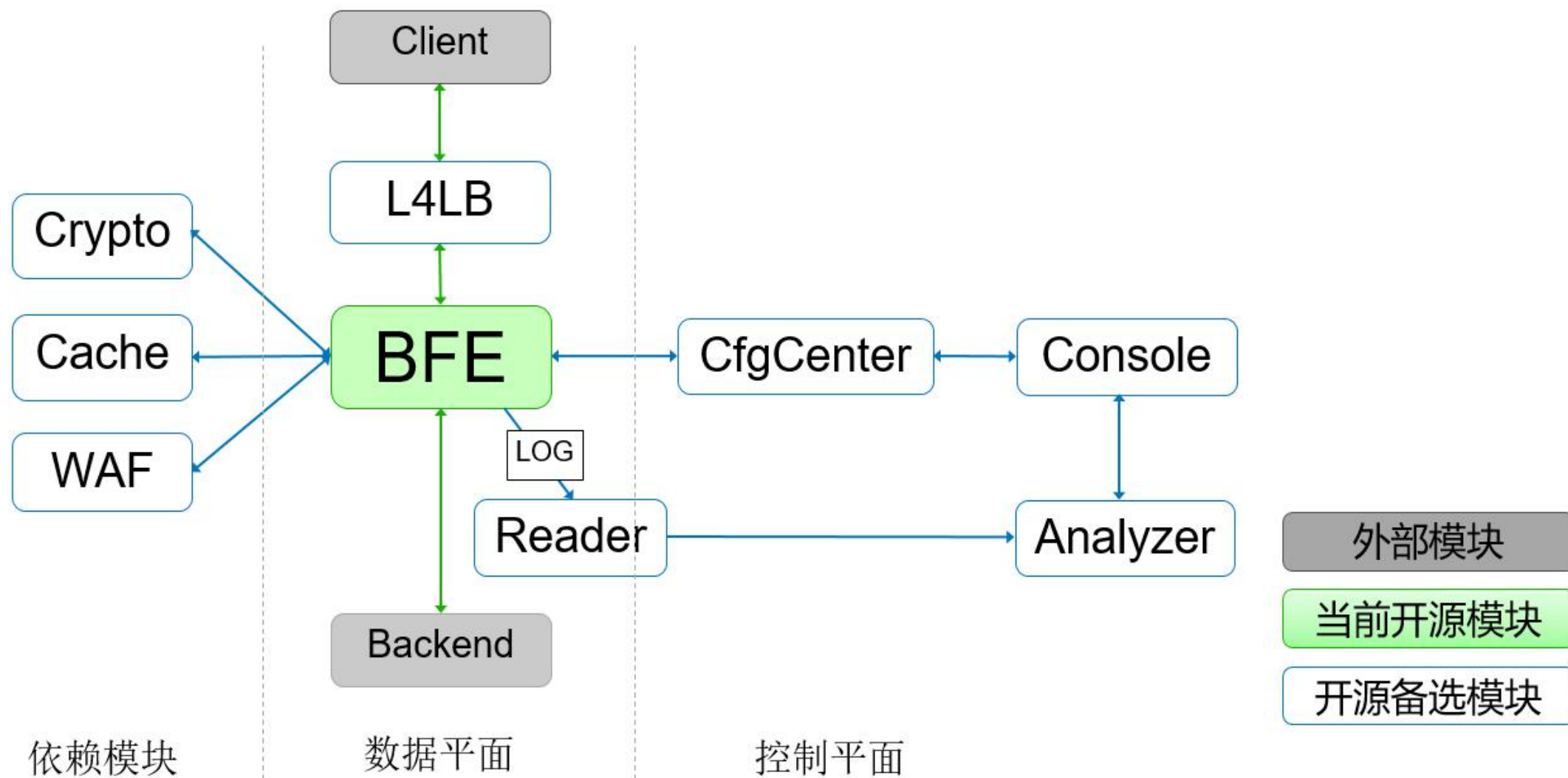
BFE部署前



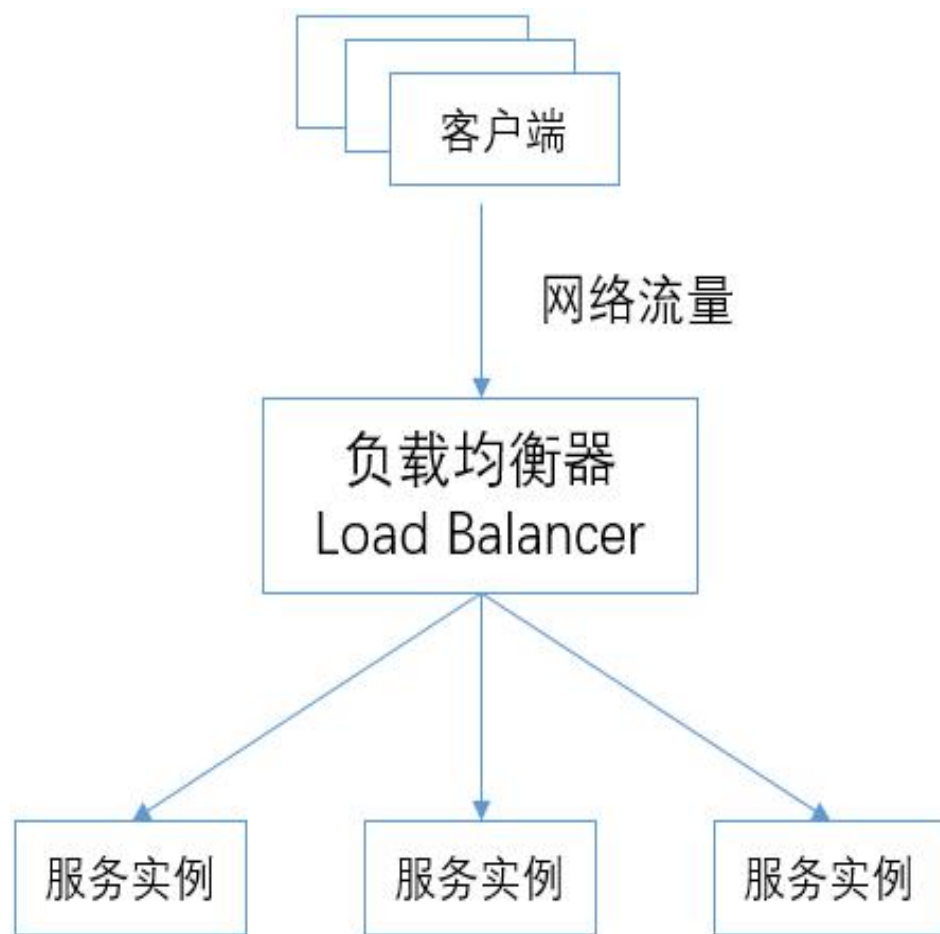
BFE部署后



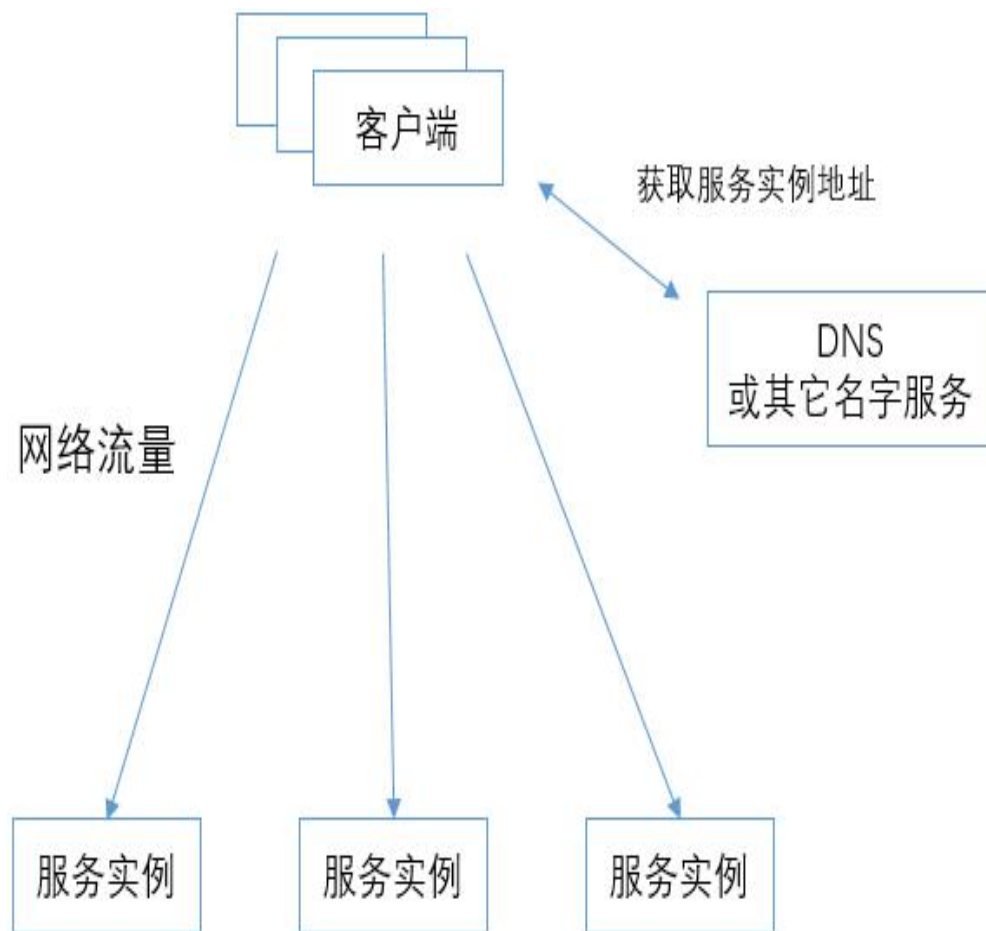
BFE平台架构



负载均衡器 vs 名字服务



基于负载均衡器



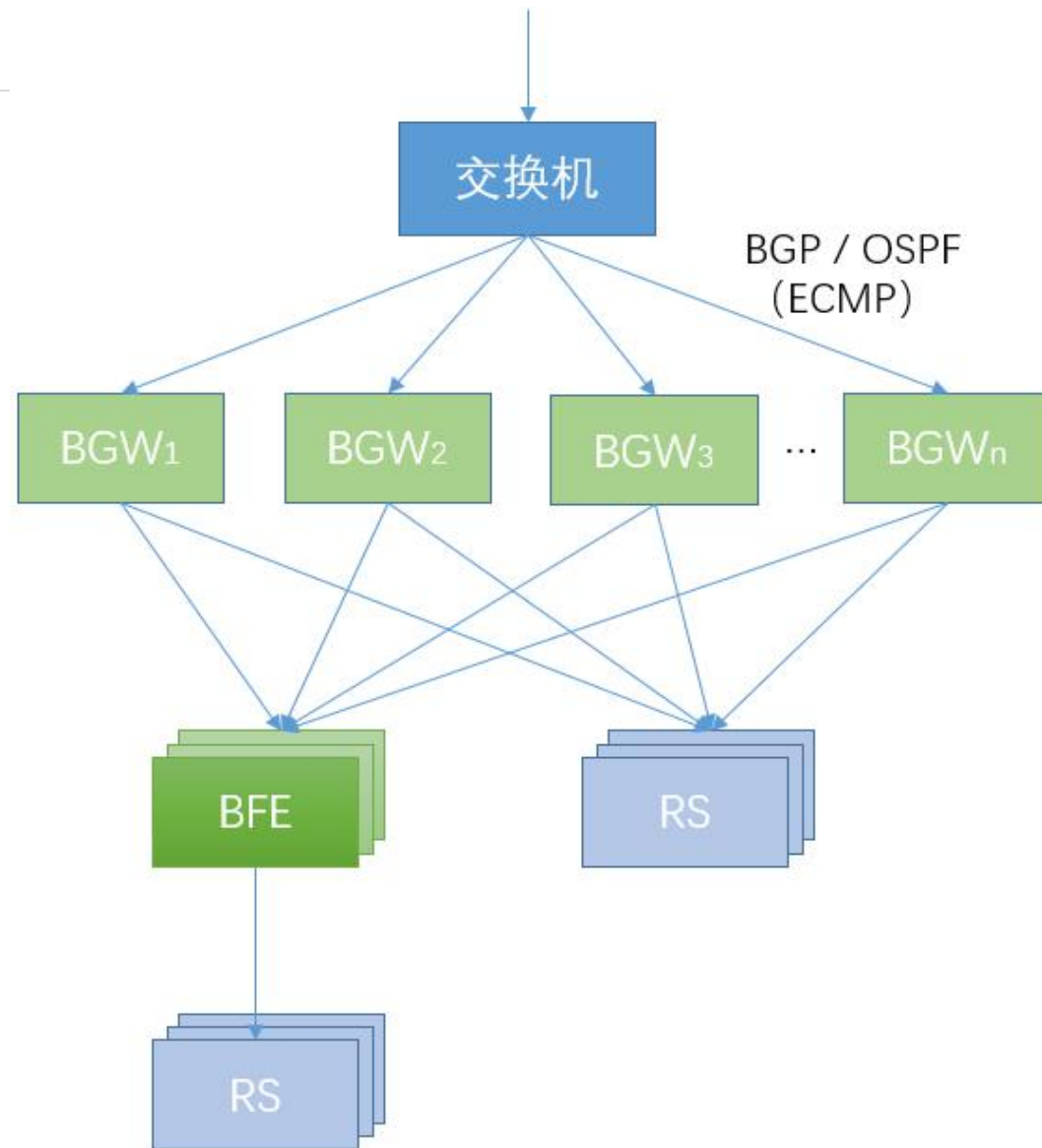
基于名字服务

方案对比

方案	对流量的控制力	资源消耗	对客户端的要求	适用场景
基于负载均衡器	强。可以达到单个连接 / 请求的粒度。	高。负载均衡器引入了额外的资源消耗。	低。客户端基本不需要实现策略。	总体流量规模不大（从负载均衡器资源消耗的角度）；应用场景对流量控制要求高。
基于名字服务 + 客户端策略	弱。客户端直接访问服务，没有可靠的卡控点，无法实现精细的流量控制测量。	低。不需要额外的资源投入。	高。客户端需要支持比较复杂的策略，且涉及升级的问题。	总体流量规模较大；应用场景对流量控制要求低；无法使用负载均衡器的场景。

负载均衡器

- 负载均衡的趋势
 - 硬件 => 软件
 - 四层和七层负载均衡器分离
- 四层负载均衡
 - LVS, DPVS, ...
- 七层负载均衡
 - HAProxy, Nginx, Envoy, Traefik, BFE, ...



BFE为什么基于Go语言

- 研发效率
 - 远高于C语言
- 稳定性
 - 内存方面错误降低
 - 可以捕捉异常
- 安全性
 - 缓存区溢出风险降低
- 代码可维护性
 - 可读性好
 - 易于编写高并发逻辑
- 网络协议栈支持

BFE的短板

- 没有在内存拷贝上做极致优化
 - 使用Go系统协议栈
- 无法利用CPU亲和性（CPU Affinity）
 - 无法控制底层线程

七层负载均衡的生态选择

Nginx / OpenResty 生态

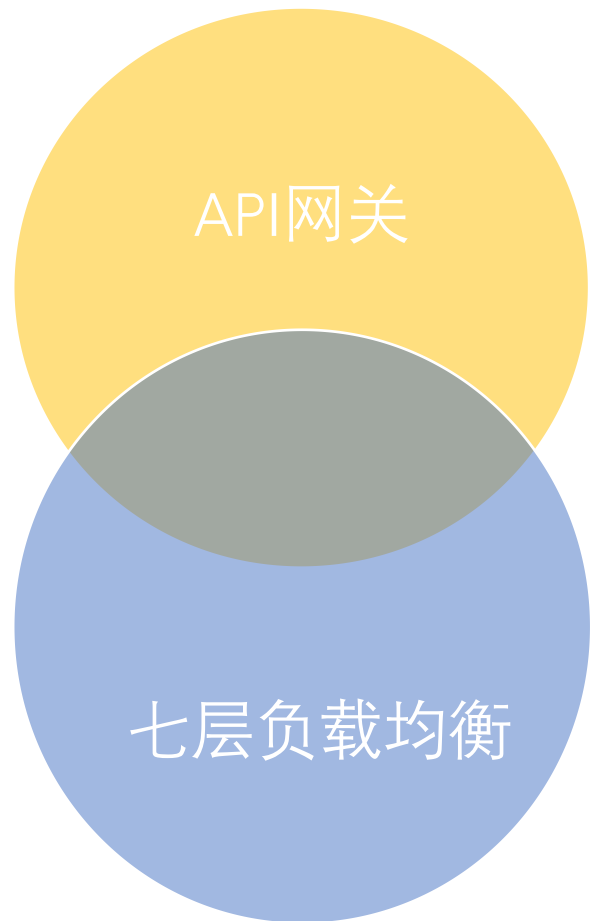
- 利用Nginx积累的大量功能
- 利用Lua的快速开发能力
- 代表：Nginx, APISIX

Envoy 生态

- 最早用于 Service Mesh
- 也可用于网关
- 代表：Envoy

Go 生态

- 基于Go语言的生态积累
- 更好的稳定性和安全性
- 易于开发扩展功能
- 代表：BFE, Traefik

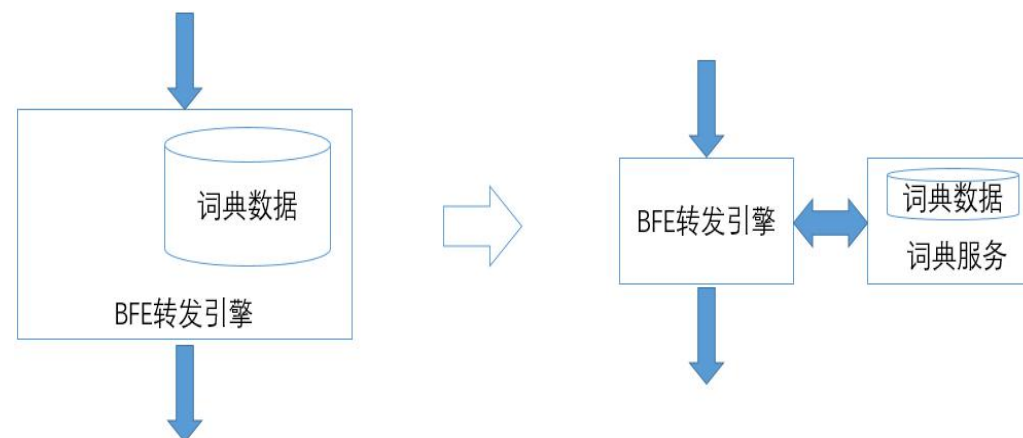


BFE主要设计思想

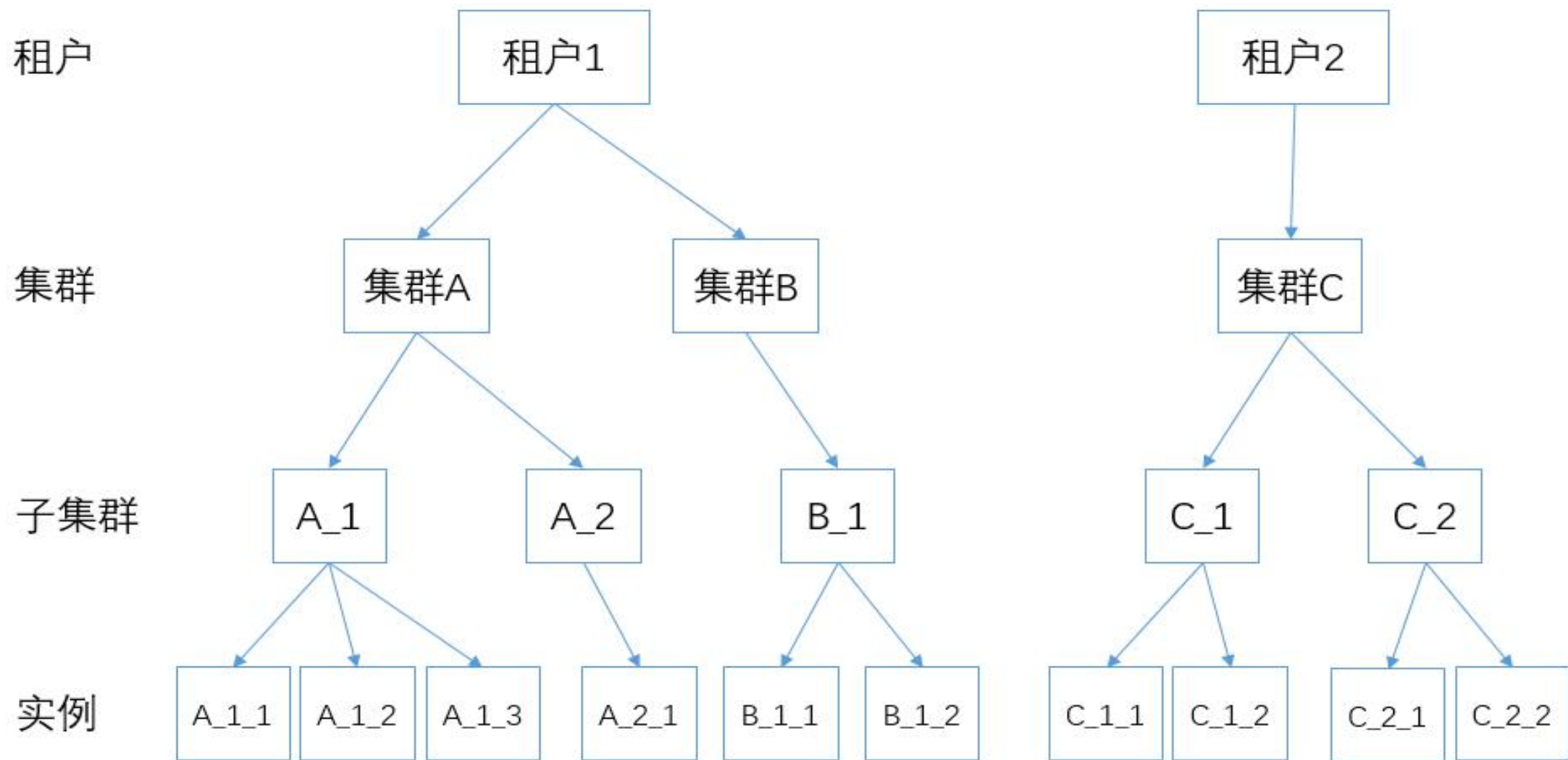
- 转发模型优化
 - 支持多租户
 - 引入条件表达式，减少正则表达式使用
- 降低动态配置加载的难度
 - 区分“常规配置”和“动态配置”
- 增强服务状态监控能力
 - 向外展现大量内部的执行状态
- 将大存储功能转移到外部
 - 加快启动速度

正则表达式 方案的问题

- **配置难以维护**：正则表达式存在严重的可读性问题
- **性能存在隐患**：有可能因编写不当引起严重的性能退化



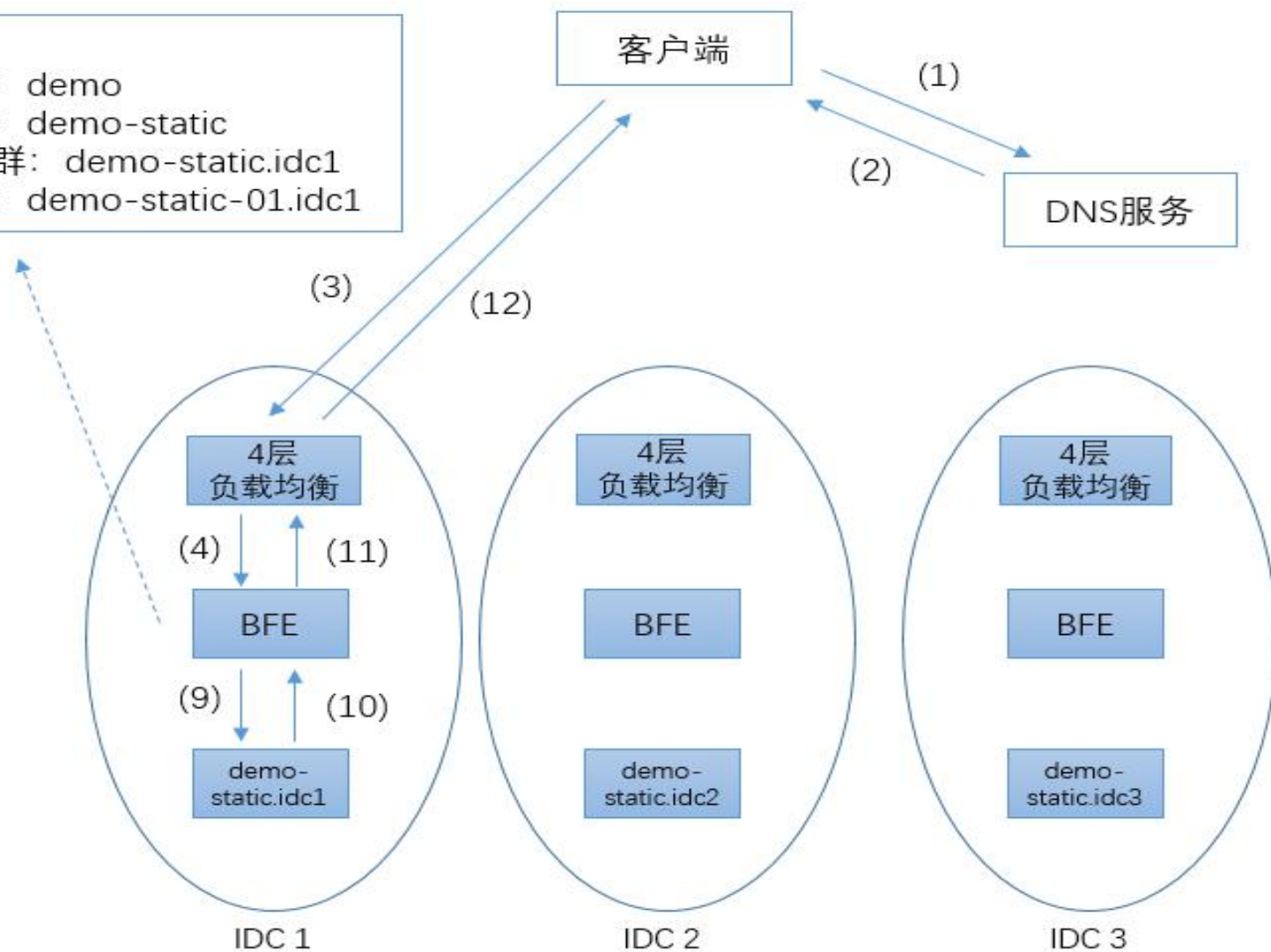
BFE转发的主要概念



BFE的转发过程

BFE的处理:

- (5) 确定租户: demo
- (6) 确定集群: demo-static
- (7) 确定子集群: demo-static.idc1
- (8) 确定实例: demo-static-01.idc1



BFE的路由转发

基础转发表



高级转发表



默认集群

匹配条件	目标集群
www.a.com/a/*	Demo-A
www.a.com/a/b	Demo-B
*.a.com/	Demo-C
www.c.com	ADVANCED_MODE

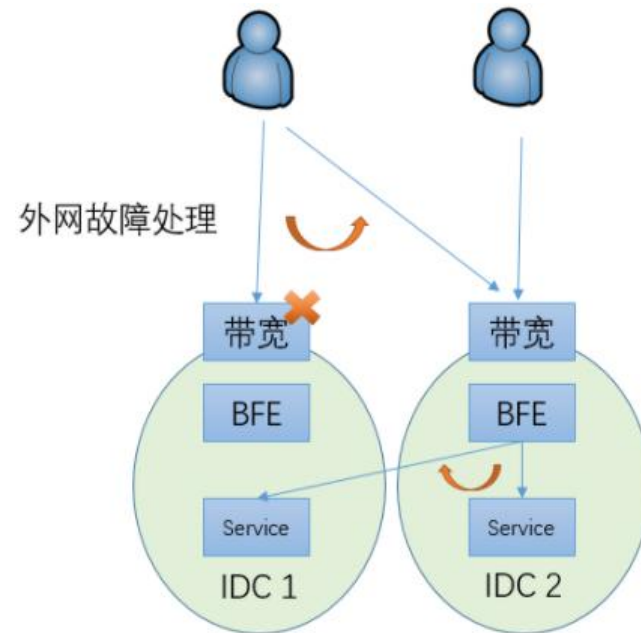
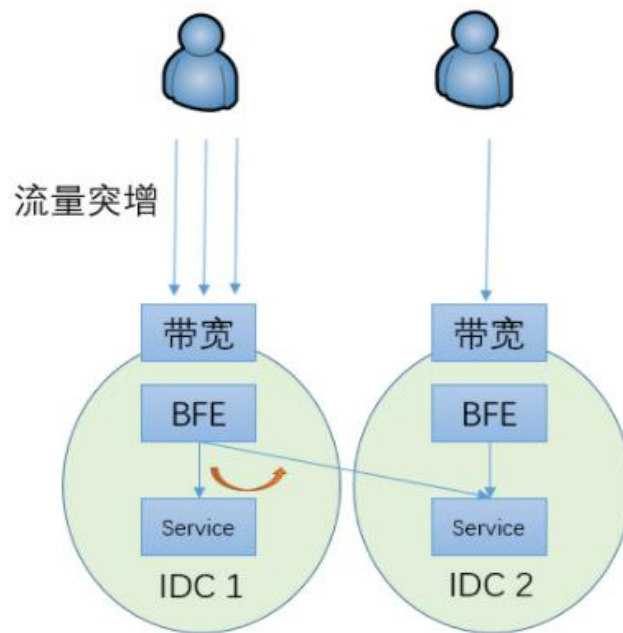
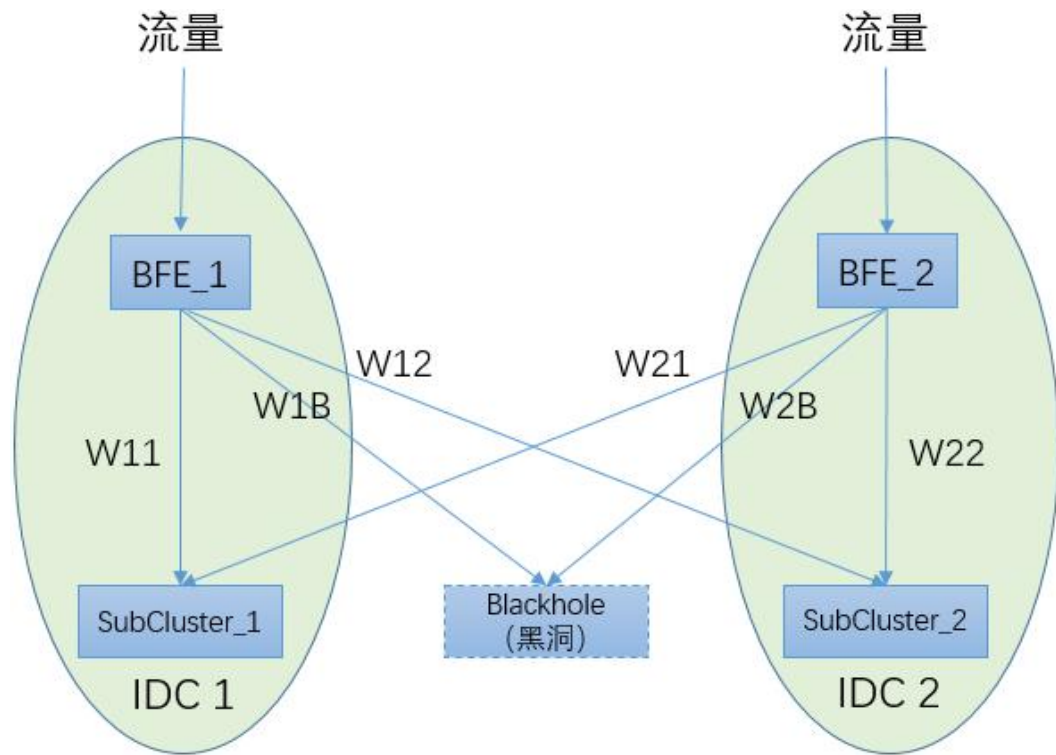
匹配条件	目标集群
req_host_in("www.c.com") && req_cookie_value_prefix_in("deviceid", "x", false)	Demo-D1
req_host_in("www.c.com")	Demo-D

Demo-E

内网流量调度

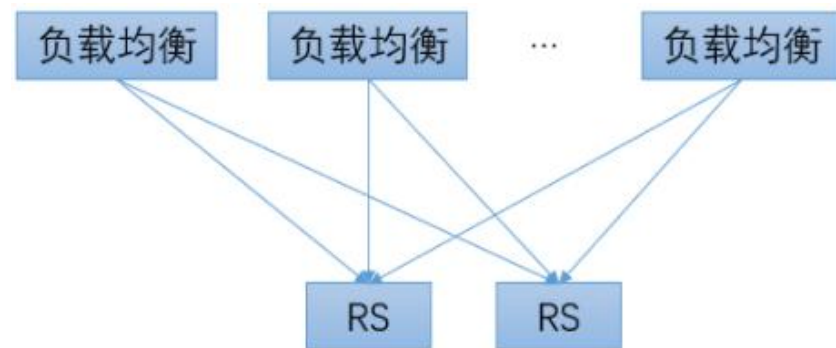
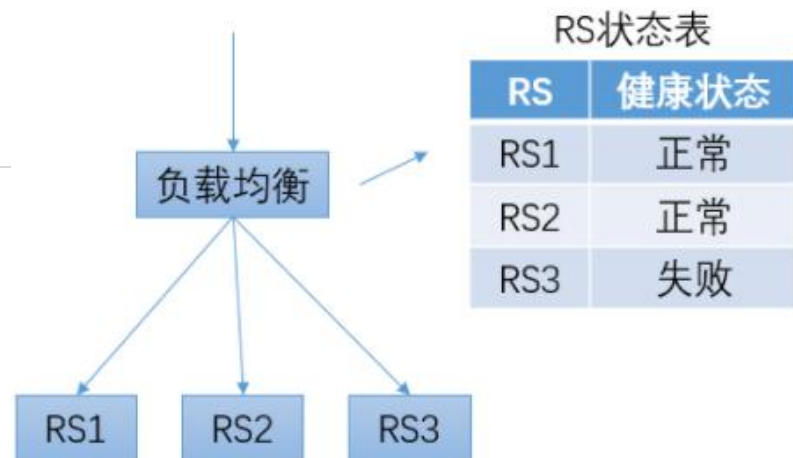
• 使用场景

- 多数据中心 / 多容器云集群
- 内部服务故障
- 内部服务压力不均



健康检查

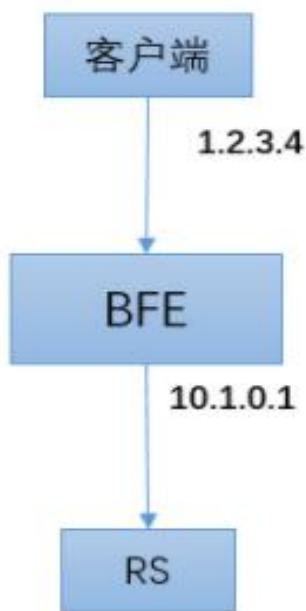
- 主动健康检查
 - 负载均衡系统**持续**向RS发送探测请求
 - 问题：在响应速度和发送压力间存在权衡
 - 在分布式场景下问题更加明显
- 被动健康检查
 - 利用正常业务请求来发现失败
 - 失败后启动主动健康检查
 - 问题：业务请求频度低时无法及时发现失败
- 主动和被动的结合
 - 汇总两种检查的结果
 - 可降低主动检查的频度(如30–60s)



RS	被动检查	主动检查	汇总结果
RS1	正常	失败	失败
RS2	正常	正常	正常
RS3	失败	正常	失败

信息透传

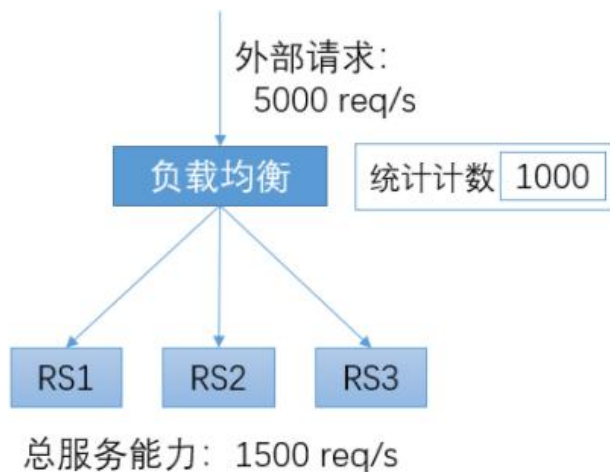
- 客户端IP地址透传
 - X-Forwarded-For: 容易被伪造
- mod_header
 - 默认增加: X-Real-Ip, X-Real-Port
 - 可以透传更多的信息



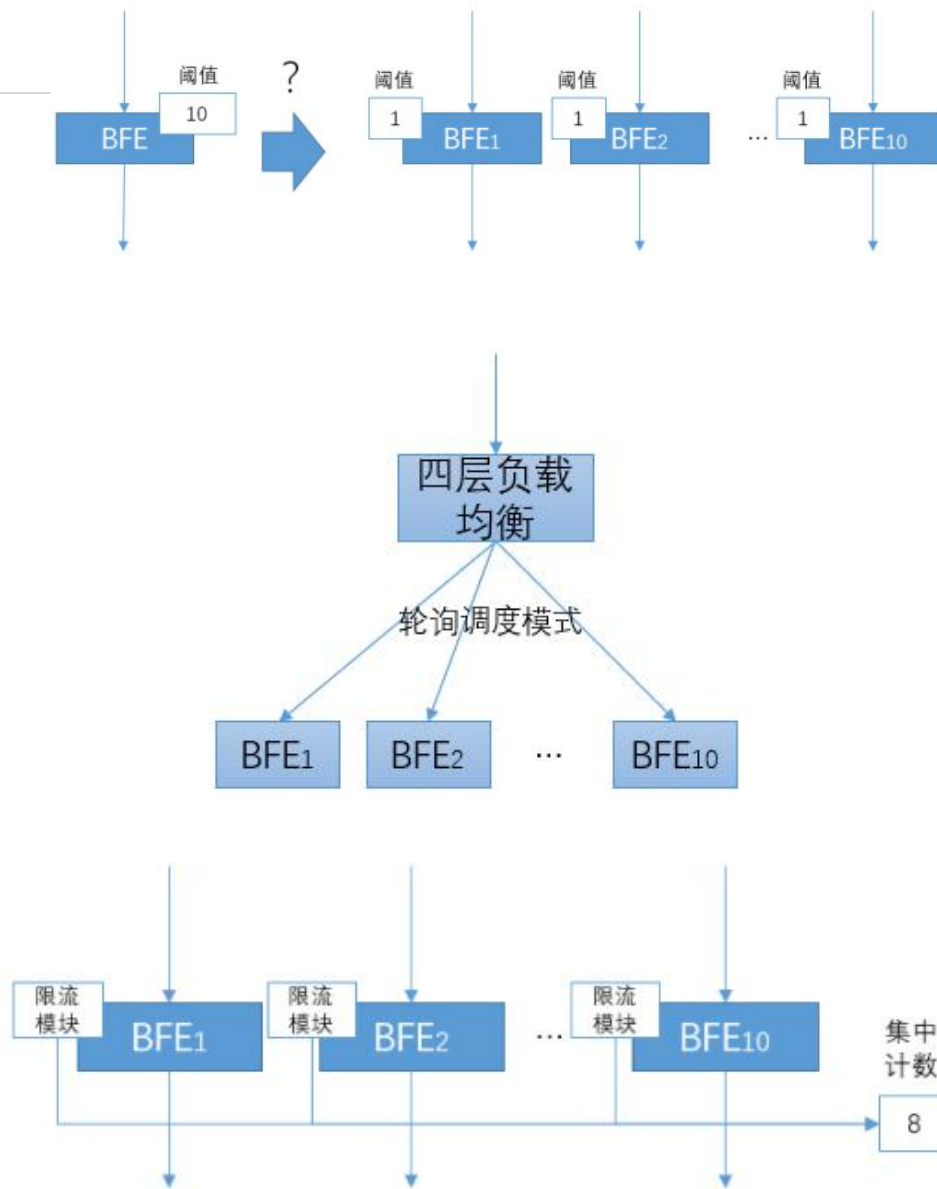
```
{
  "Version": "20190101000000",
  "Config": {
    "example_product": [
      {
        "cond": "req_path_prefix_in(\"/header\",
false)",
        "actions": [
          {
            "cmd": "REQ_HEADER_SET",
            "params": [
              "X-Bfe-Log-Id",
              "%bfe_log_id"
            ]
          },
          {
            "cmd": "RSP_HEADER_SET",
            "params": [
              "X-Proxied-By",
              "bfe"
            ]
          }
        ],
        "last": true
      }
    ]
  }
}
```

限流机制

- 问题：
 - 分布式场景下是否可以均分阈值？
- 要精确限流，需使用集中计数

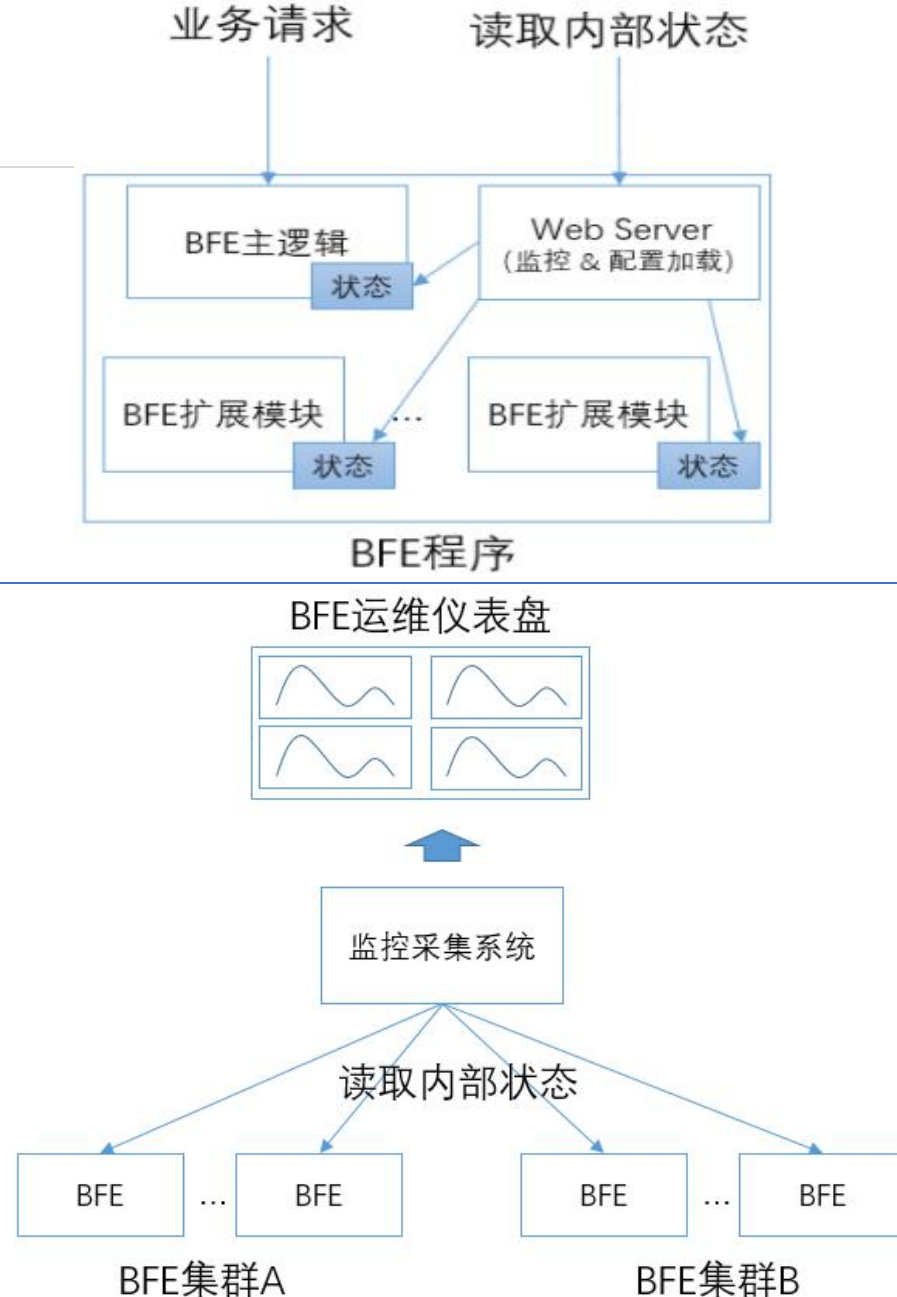


统计特征	统计周期	阈值	动作
www.demo.com	1秒	1200	关闭连接
api.baidu.com/api	1秒	200	返回静态结果



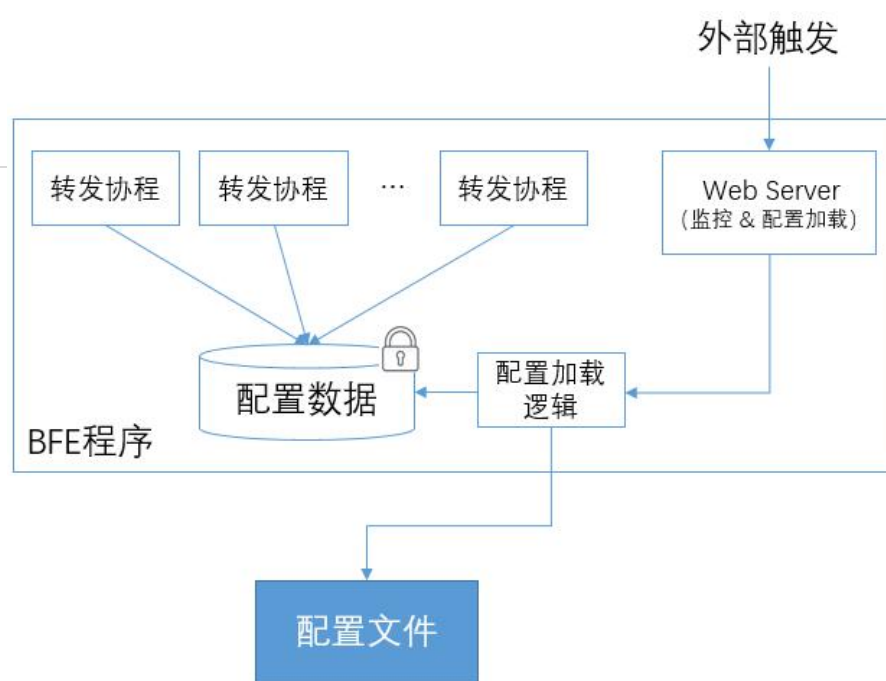
监控机制

- 基于日志监控的问题
 - 被监控系统的资源消耗较高
 - 监控系统的资源消耗较高
 - 很多状态信息并不适合打印输出
- BFE的内部状态输出
 - 通过内嵌web server向外暴露
 - 状态信息的汇聚和读取成本低
- 可以将状态和日志配合使用
- Web Monitor框架
 - <https://github.com/baidu/go-lib>
 - 支持状态、差值、延迟统计等



配置管理

- BFE配置的分类
 - 常规配置: .conf
 - 动态配置: .data
- 配置动态加载
 - 外部触发, 细粒度加载



写配置

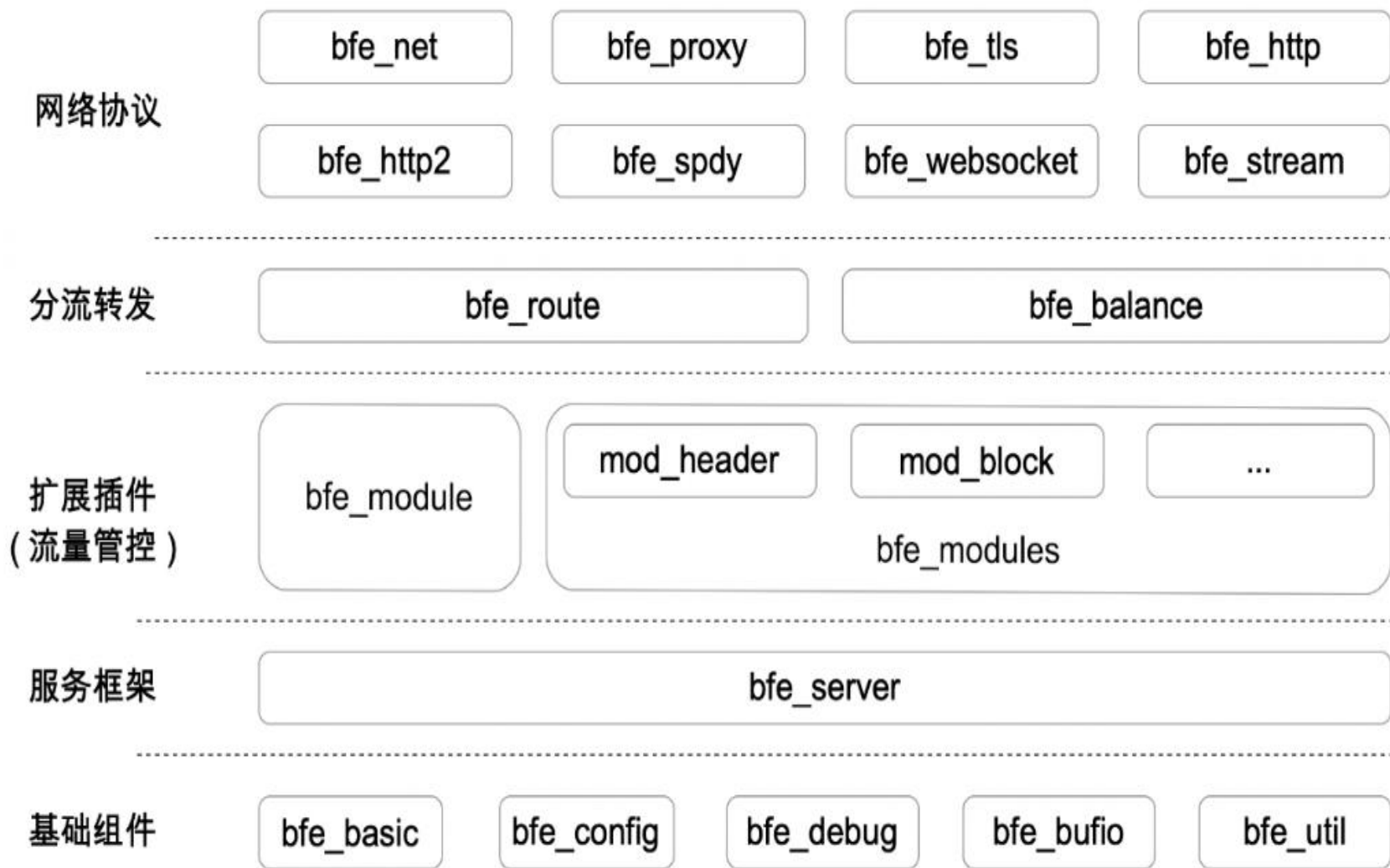
```
func (t *ProductRuleTable) Update(conf
productRuleConf) {
    t.lock.Lock()
    t.version = conf.Version
    t.productRules = conf.Config
    t.lock.Unlock()
}
```

读配置

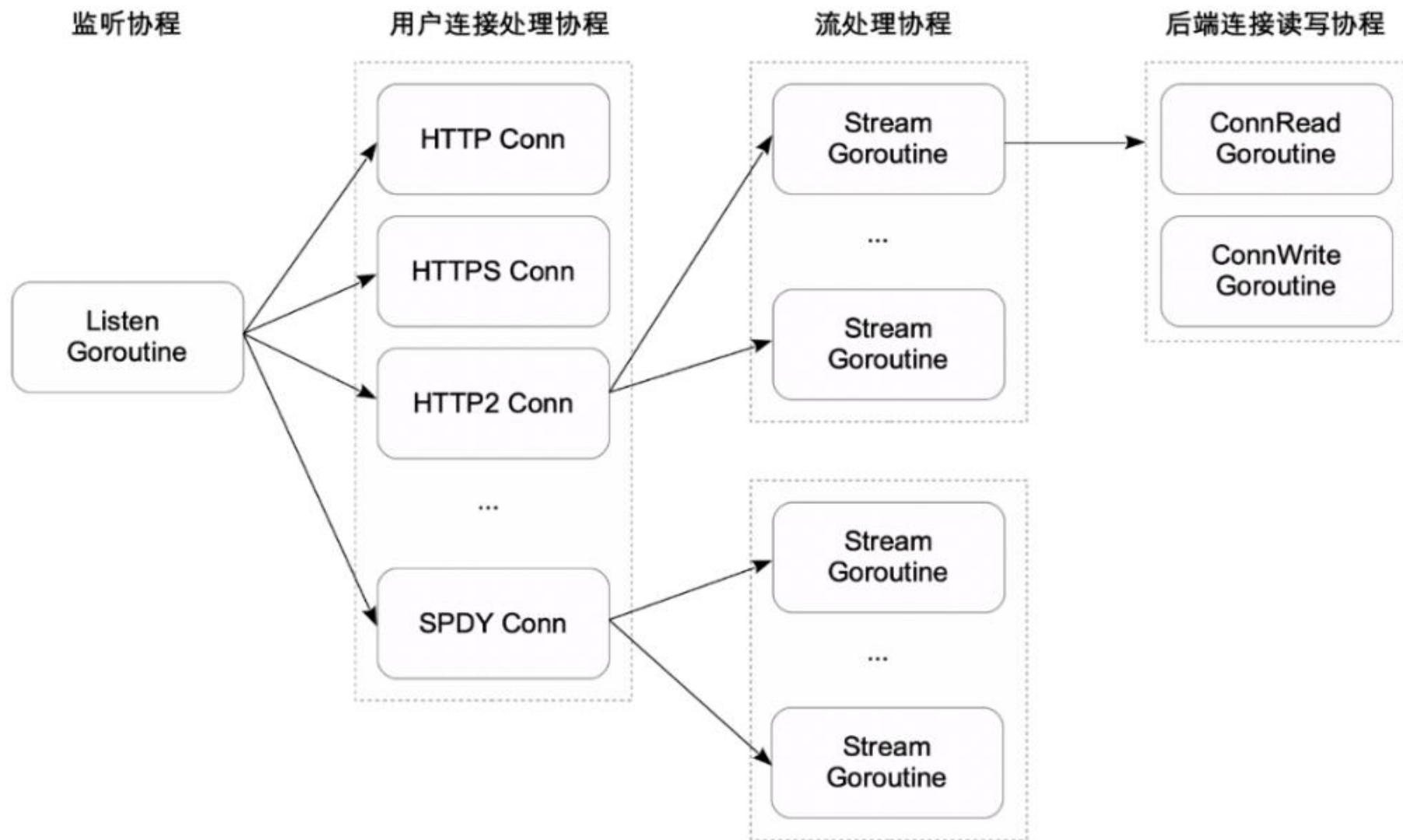
```
func (t *ProductRuleTable) Search(product string)
(*blockRuleList, bool) {
    t.lock.RLock()
    productRules := t.productRules
    t.lock.RUnlock()

    rules, ok := productRules[product]
    return rules, ok
}
```

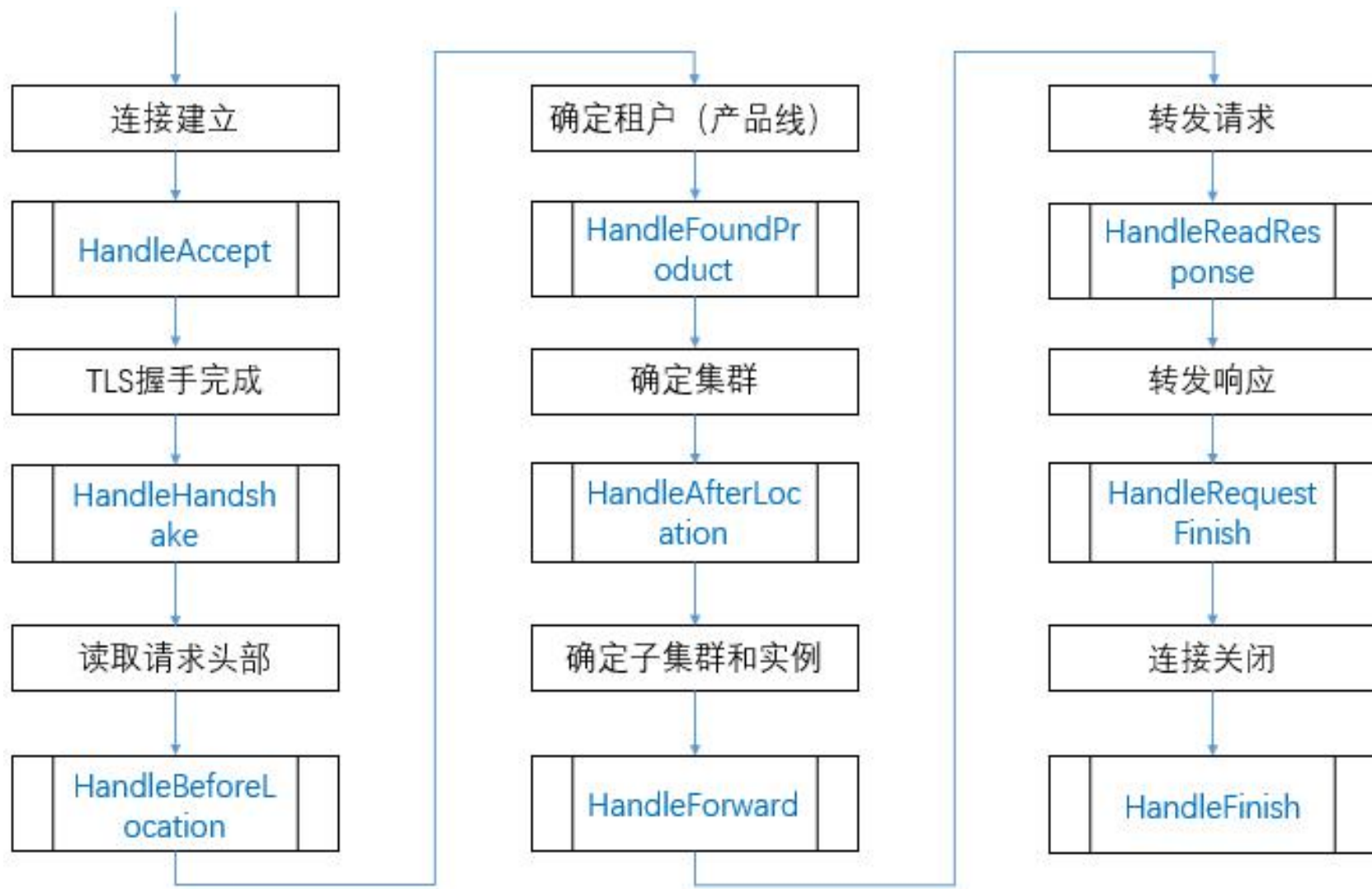
BFE的代码组织



BFE的协程使用



BFE的回调点设置



BFE扩展模块的编写

- 配置加载

- 静态: `mod_block.conf`
- 动态: `block_rules.data`,
`ip_blocklist.data`

- 回调函数编写和注册

- 定义状态变量

```
type ModuleBlock struct {  
    ...  
    state    ModuleBlockState // module state  
    metrics metrics.Metrics  
}  
  
func NewModuleBlock() *ModuleBlock {  
    m := new(ModuleBlock)  
    m.metrics.Init(&m.state, ModBlock, 0)  
    ...  
}
```

```
// register web handler for reload  
err = whs.RegisterHandler(web_monitor.WebHandleReload,  
m.name, m.loadConfData)  
if err != nil {  
    ...  
}
```

```
func (m *ModuleBlock) productBlockHandler(request  
*bfe_basic.Request) (int, *bfe_http.Response) {  
    ...  
}  
  
func (m *ModuleBlock) Init(cbs *bfe_module.BfeCallbacks, whs  
*web_monitor.WebHandlers, cr string) error {  
    // register handler  
    err = cbs.AddFilter(bfe_module.HandleFoundProduct,  
m.productBlockHandler)  
    ...  
}
```

未来计划

- BFE管理控制台对外开源
- BFE ingress controller对外开源

欢迎广大Gopher使用或参与共建！



BFE开源项目公众号

谢谢