



ZHDBA.COM
中华数据库行业协会

沪江ApiGateway实践

关于我

- 夏志培
- 沪江网运维架构师/基础运维团队负责人,10+年运维经验。
- 负责沪江Redis/Codis,DNS,ApiGateway,RabbitMQ,ELK,Ceph等基础组件运维和架构工作。
- 对系统底层以及开源的软件有浓厚的兴趣。
- 微信:summer_xia_027



目录

CONTENTS

01

背景和目的

02

ApiGateway架构

03

ApiGateway的实现

04

ApiGateway部署/性能

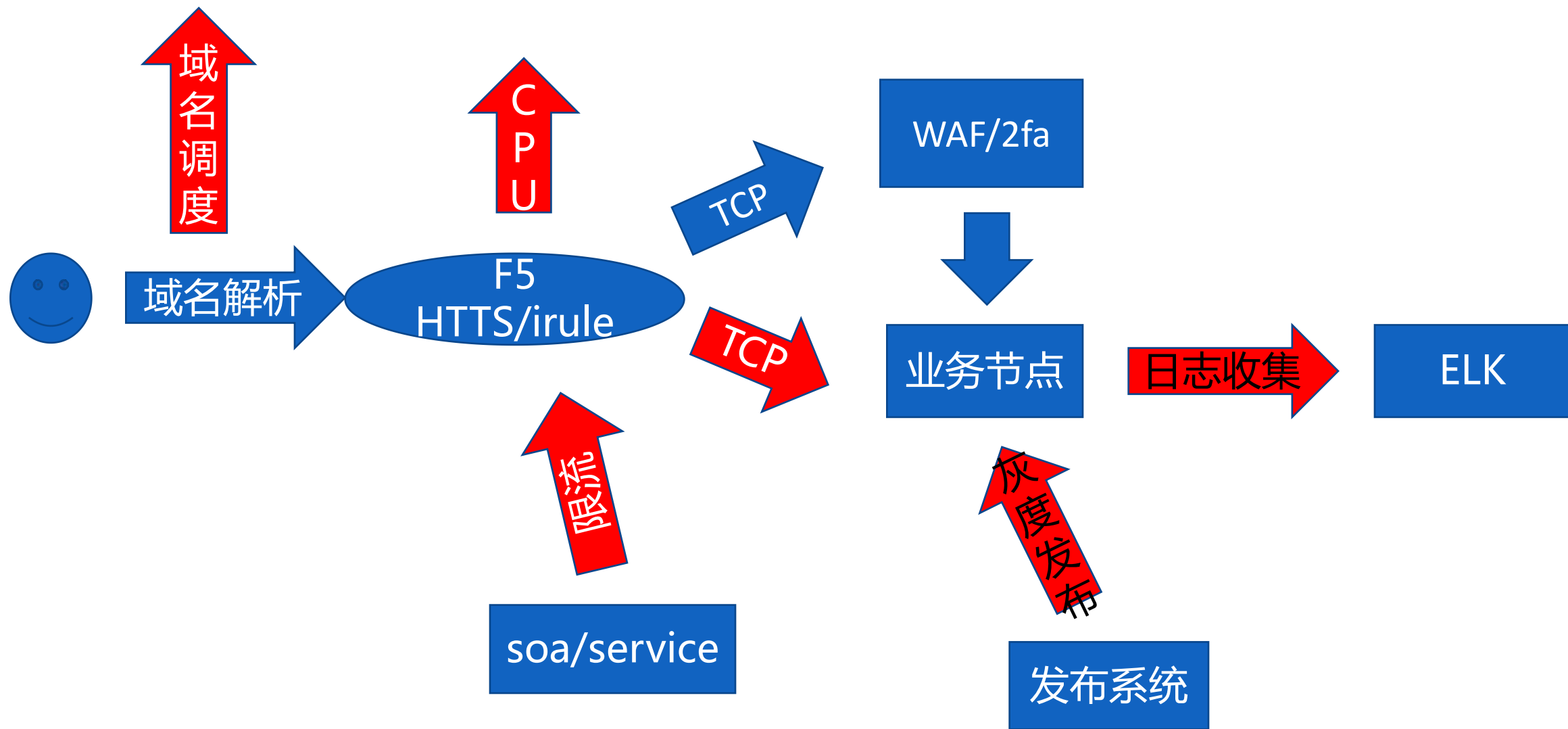
05

QA

01

背景和目标

项目背景



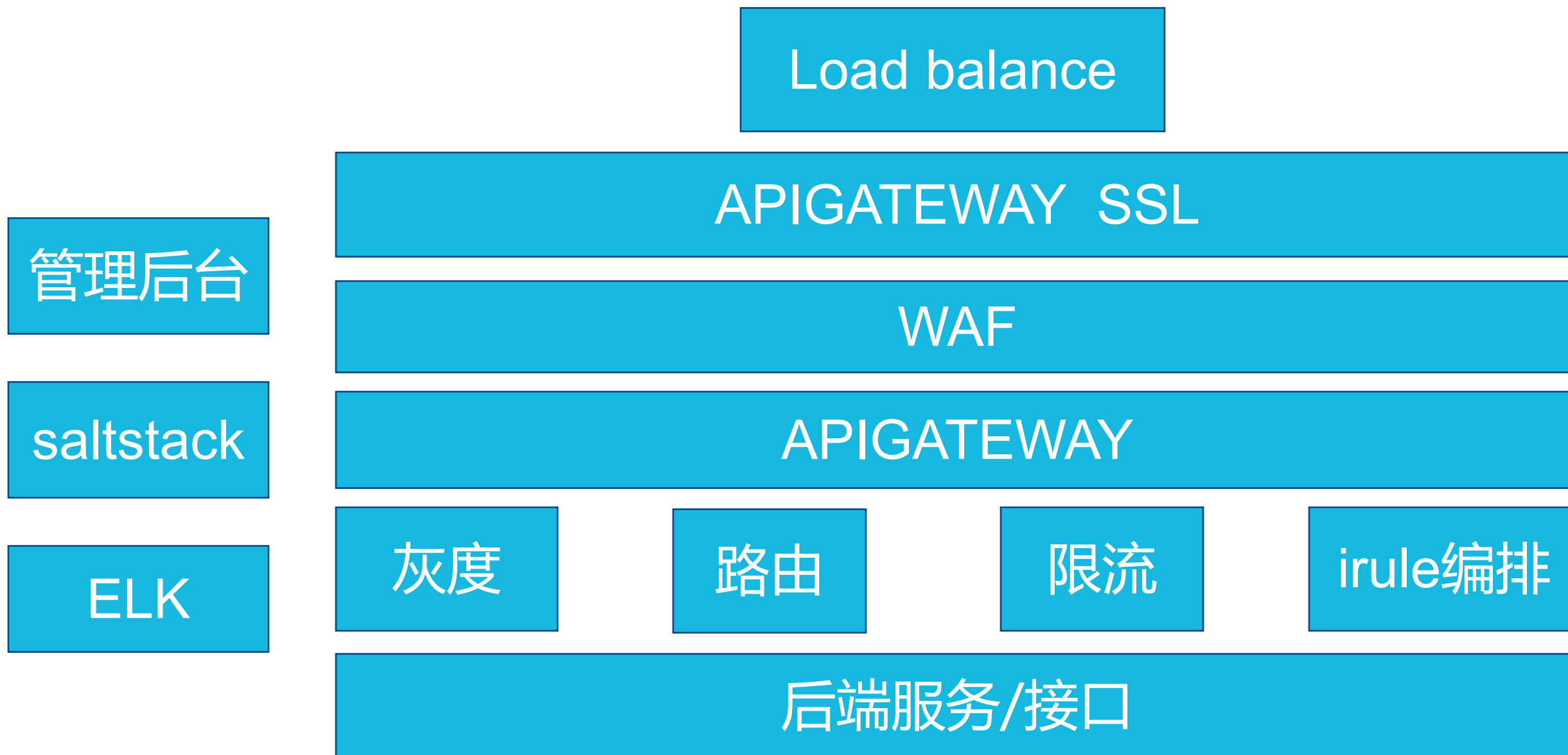
目标

1. 流量调度
2. 协议适配/SSL卸载/路由/足够的扩展
 - 限流
 - 灰度发布/黑白名单
5. 统一日志收集
6. 贴近业务的health_check
7. 轻松运维

02

ApiGateway架构

ApiGateway架构



03

ApiGateway的实现

ApiGateway实现:流量调度-调度域名

A机房SSL的调度域名

group1.idcA.CNC.ssl.cname.com

A机房ApiGateway的调度域名

group1.idcA.CNC.gateway.cname.c
om

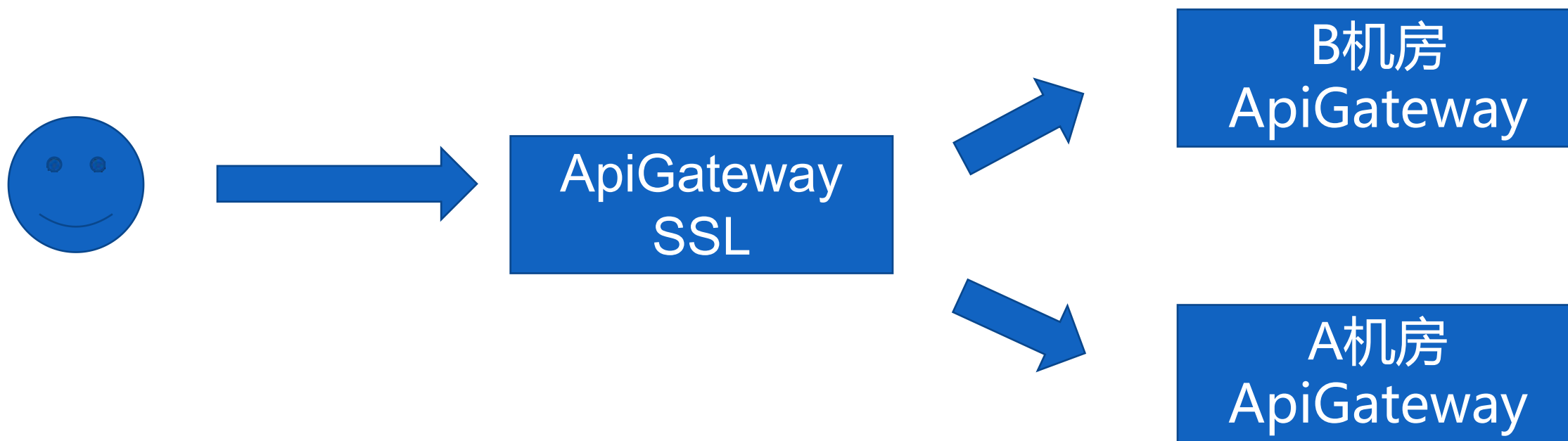
B机房SSL的调度域名

group1.idcB.CNC.ssl.cname.com

B机房ApiGateway的调度域名

group1.idcB.CNC.gateway.cname.c
om

ApiGateway实现:流量调度-流量转发



ApiGateway实现:流量调度-流量转发



The screenshot displays the '节点组选择' (Node Group Selection) interface in the API Gateway console. It is divided into two main sections: '安全节点组' (Security Node Groups) and 'apigateway节点组' (API Gateway Node Groups). Each section contains two columns: '未选择的安全组' (Unselected Security Groups) and '已选择的安全组' (Selected Security Groups), with a double-headed arrow between them indicating the selection process.

安全节点组 (Security Node Groups):

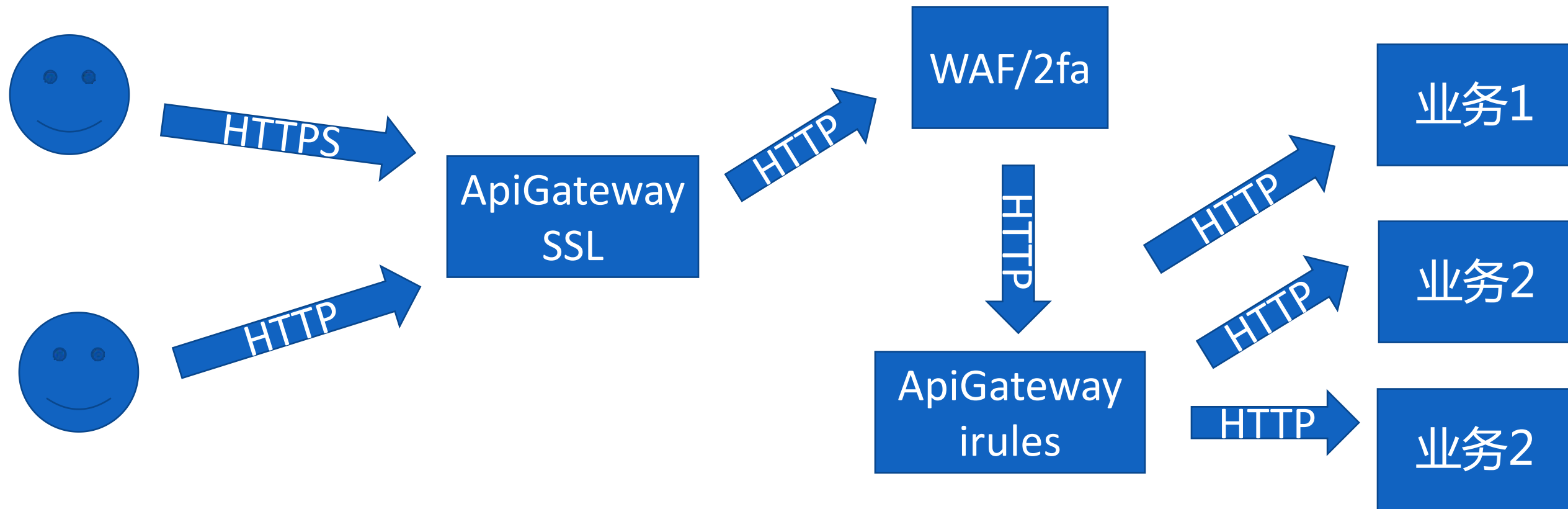
- 未选择的安全组 (Unselected Security Groups): waf_group2(bx)
- 已选择的安全组 (Selected Security Groups): 2fa_group1(b7), waf_group1(b7)

apigateway节点组 (API Gateway Node Groups):

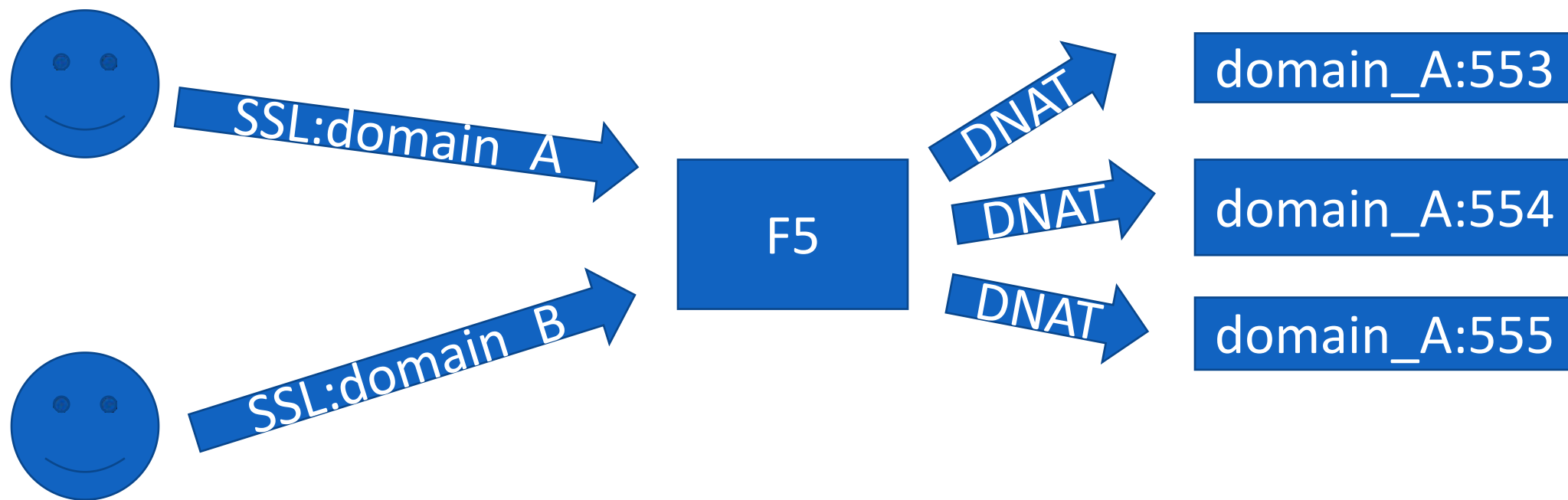
- 未选择的apigateway组 (Unselected API Gateway Groups): BAOXIN_openresty(bx)
- 已选择的apigateway组 (Selected API Gateway Groups): openresty_b7_group1(b7)

通过专线将流量引到其他IDC

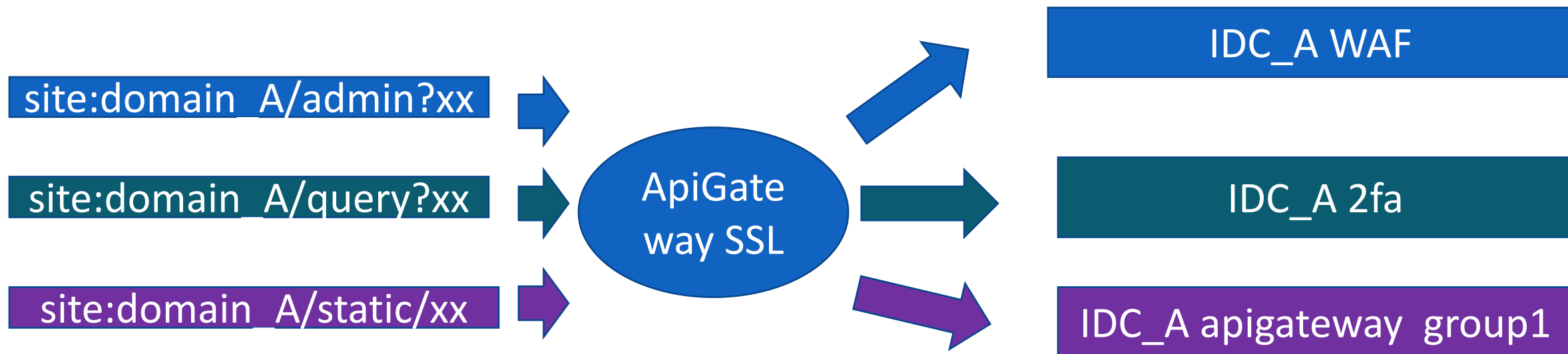
ApiGateway实现:协议适配



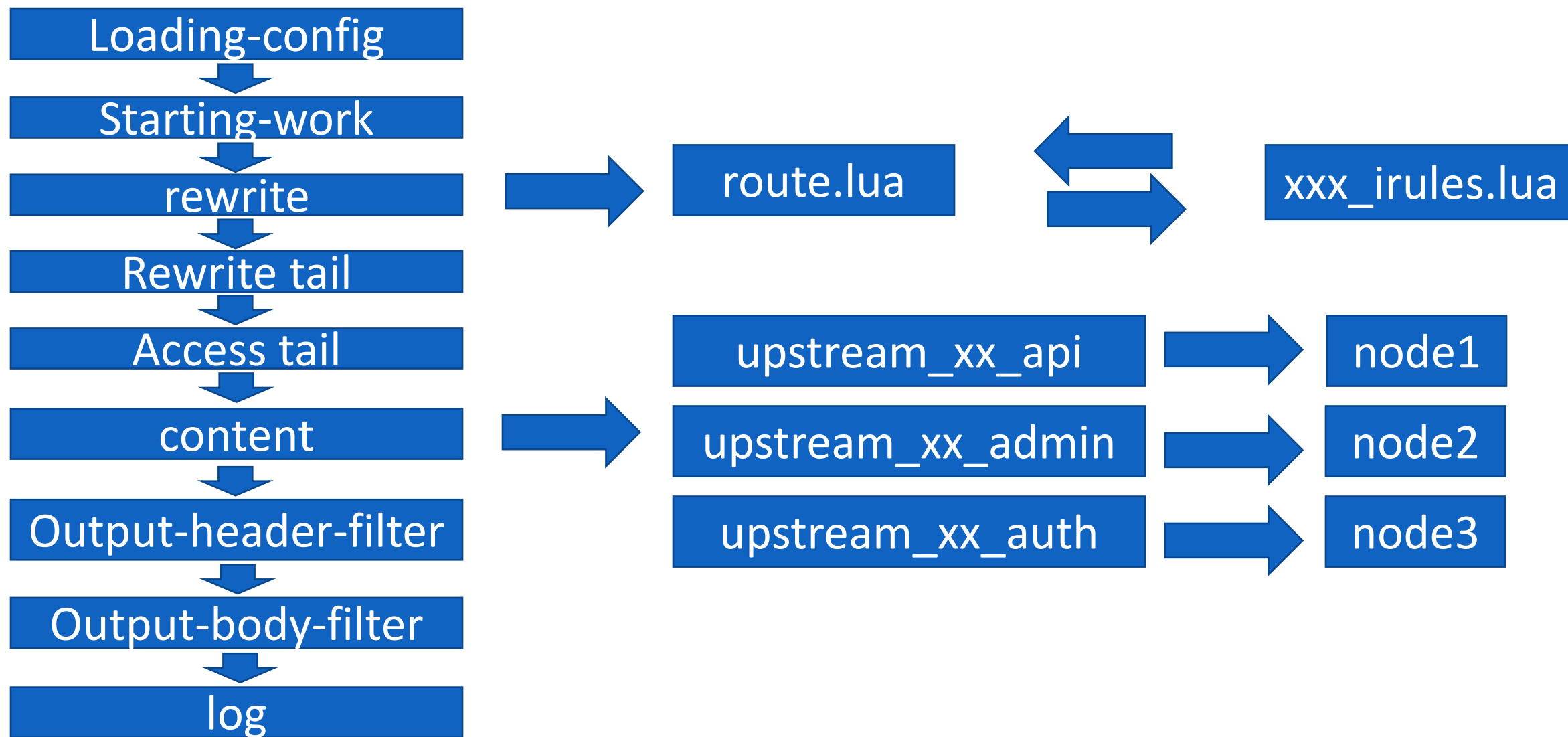
ApiGateway实现:协议适配(SSL-offload)



ApiGateway实现:路由(SSL 路由)



ApiGateway实现:路由



ApiGateway实现:路由-irules实例

```
local modulename = "apigatewayiruleslogin_hujiang.com"
local systemConfig = require( 'apigateway.utils.config' )
local cJSON = require( 'cjson.safe' )
local ERRORINFO = require( 'apigateway.error.errcode' ).info
local iputils = require( 'apigateway.utils.iputils' )
local utils = require( 'apigateway.utils.utils' )
local cacheProcessPrefix = systemConfig.prefix.cacheProcessPrefix
local iproute = systemConfig.routeType.iproute
local _M = {}
local mt = { __index = _M }
_M._VERSION = "0.0.1"
_M.new = function(self)
    return setmetatable(self, mt)
end
_M.process = function(self)
    local http_host = ngx.var.http_host
    local cookie = ngx.req.get_headers()[ "Cookie" ]
    ngx.log(ngx.INFO, cookie)
    local url = ngx.var.request_uri
    if http_host == "login.hj.com" and string.find(url, "/?auth?") then
        return "login_hj_com_auth?"
    elseif http_host == "login.hj.com" and string.find(url, "/?auth") then
        ngx.req.set_header("Host", "hujiang.com")
        return "login_hj_com"
    else
        ngx.req.set_uri("/")
        return "login_hj_com"
    end
end
return _M
```

ApiGateway实现:路由-irules配置实例

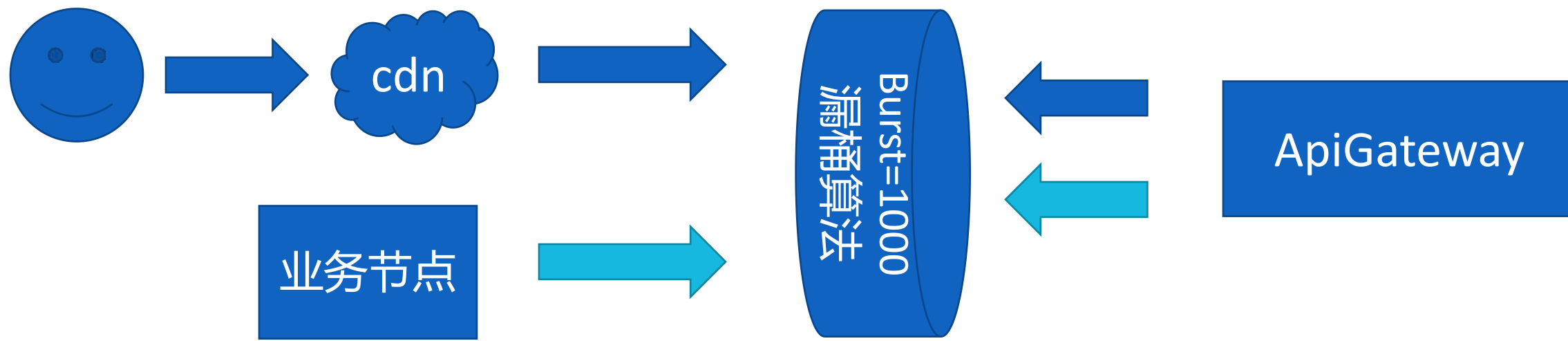


```
1 local modulename = "apigatewayiruleslms_[redacted]"
2 local systemConfig = require('apigateway.utils.config')
3 local cJSON = require('cjson.safe')
4 local ERRORINFO = require('apigateway.error.errcode').info
5 local iputils = require('apigateway.utils.iputils')
6 local utils = require('apigateway.utils.utils')
7 local cacheProcessPrefix = systemConfig.prefix.cacheProcessPrefix
8 local iproute = systemConfig.routeType.iproute
9 local _M = {}
10 local mt = { __index = _M }
11 _M._VERSION = "0.0.1"
12 _M.new = function(self)
13     return setmetatable(self, mt)
14 end
15 _M.process = function(self)
16     local http_host = ngx.var.http_host
17     local url = ngx.var.request_uri
18     if http_host == "lms.[redacted].com" and (string.find(url, "/catalog")) then
19         return "lms_[redacted].com_catalog"
20     elseif http_host == "lms.[redacted].com" and (string.find(url, "/log")) then
21         return "lms_[redacted].com_log"
22     elseif http_host == "lms.[redacted].com" and (string.find(url, "/content")) then
23         return "lms_[redacted].com_content"
24     else
25         return "lms_[redacted].com_content"
26     end
27 end
28 return _M
```

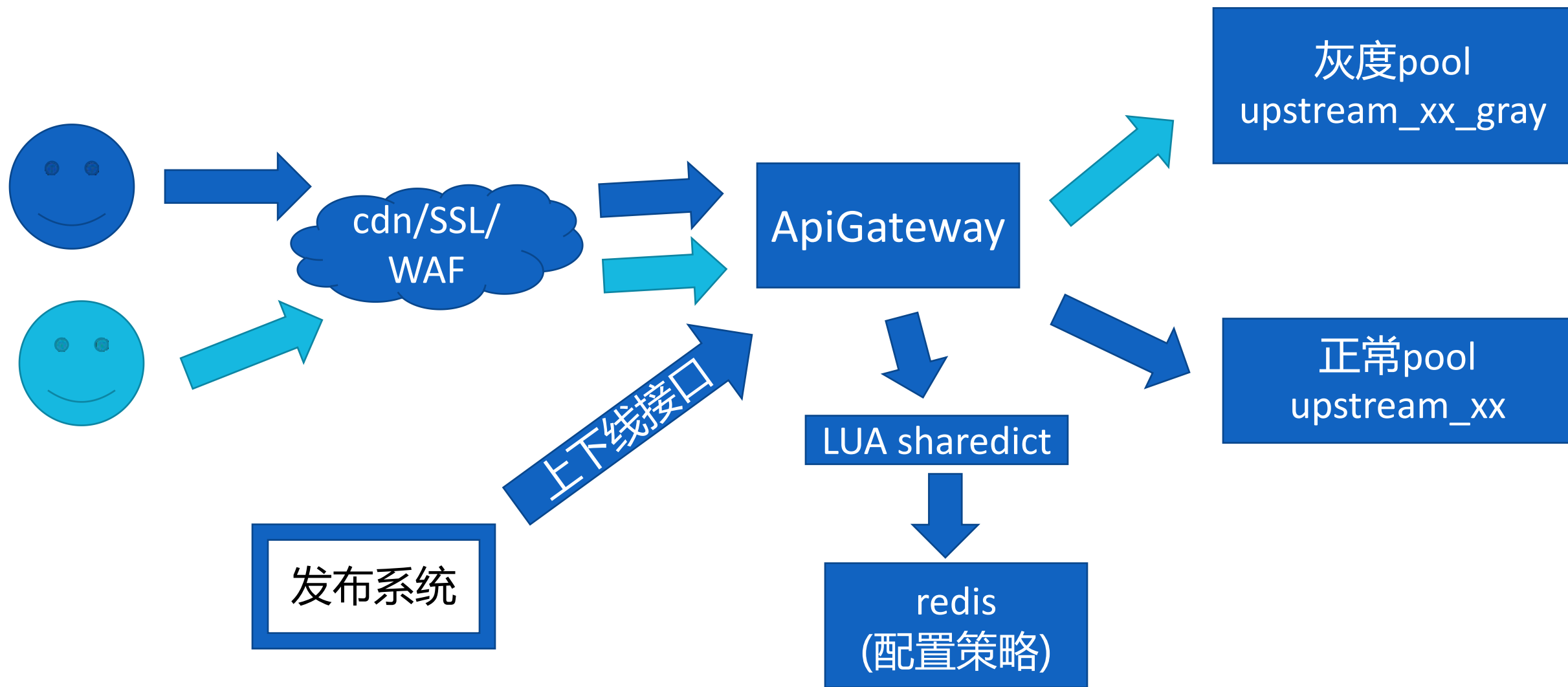
ApiGateway实现:限流(客户端/服务器端)

```
limit_req_zone $x_forwarded_for zone=domainx_C:50m rate=100r/s;  
limit_req_zone $remote_addr zone=domainx_S:50m rate=300r/s;
```

```
limit_req zone=domainx_C_client burst=1000;  
limit_req zone=domainx_S burst=1000;
```



ApiGateway实现:灰度发布



ApiGateway实现:灰度发布

路由策略：

```
{“api:gateway:strategy:login.hj.com”:” iproute,vesion_range”}
```

策略处理-灰度处理

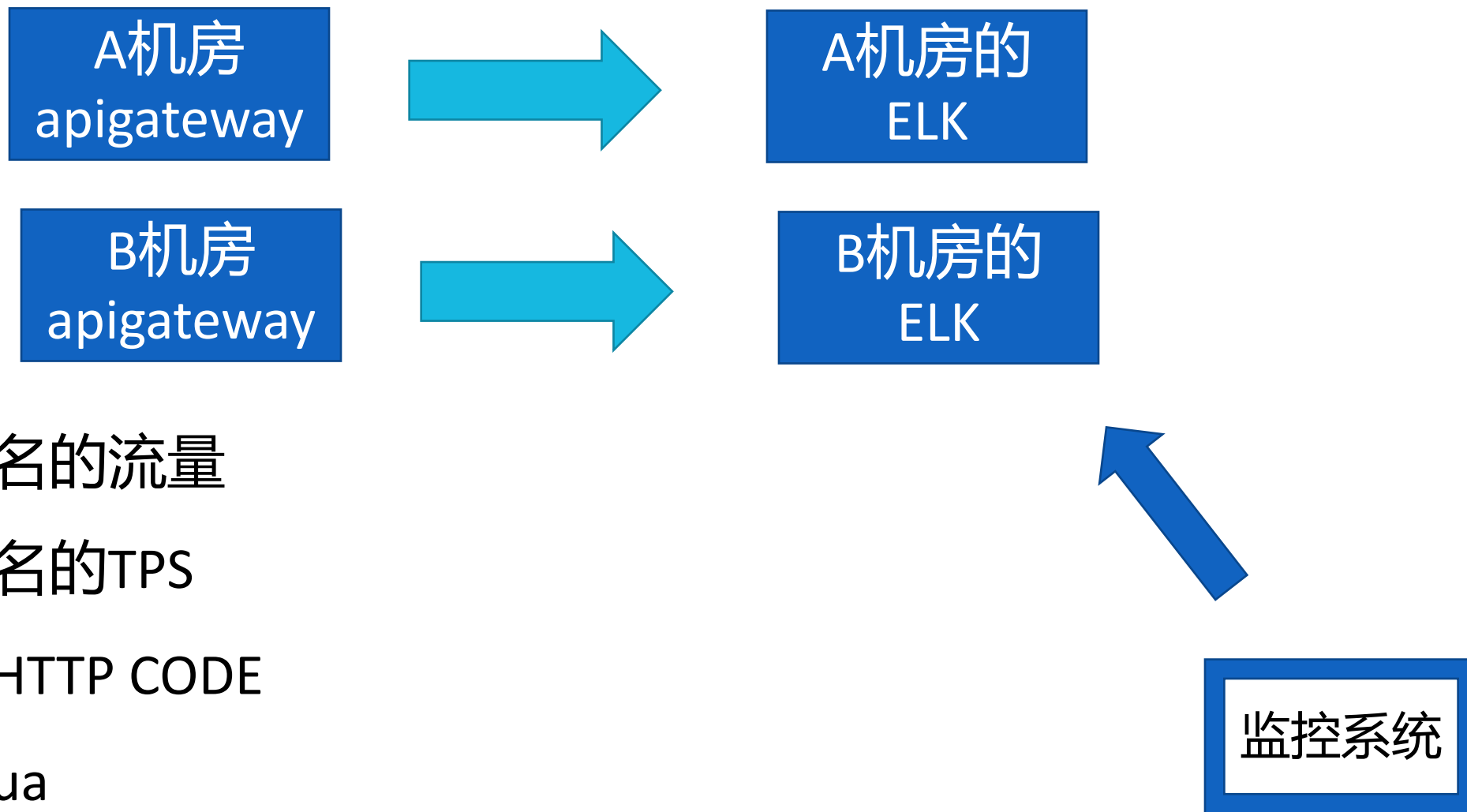
Local cache格式

```
{“api:gateway:process:iproute:login.hj.com:192.168.164.128”:”gray_upstream”}
```

Redis格式

```
{“api:gateway:process:iproute:login.hj.com”:[{“filterIP”:”192.168.164.128”,“upstream”:“gray1”},{“filterIP”:”192.168.1.0/24”,“upstream”: “gray2”}]}
```

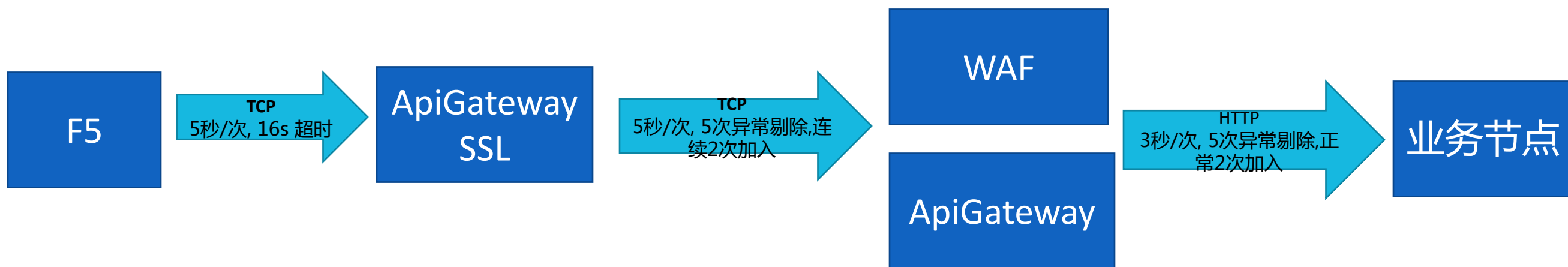
ApiGateway实现:日志收集/监控



- 1.单域名的流量
- 2.单域名的TPS
- 3.请求HTTP CODE
- 4.设备ua
- 5.响应时间

ApiGateway实现: health_check

nginx_upstream_check_module-0.3.0



ApiGateway实现: health_check

upstream

+添加upstream

upstream名称

Ims_..._catalog

删除此upstream

```
check interval=3000 rise=2 fall=4 timeout=3000 type=http;
check_http_send "GET /catalog/do_not_delete/health_check/ HTTP/1.0\r\nUser-Agent:...\r\nHOST:Ims....\r\nmonitor-hj:7BCDB077276BEC8A9D2CE39548B5D57E\r\n\r\n";
check_http_expect_alive http_2xx http_3xx;
keepalive 300;
```

upstream名称

Ims_..._catalog_gray

删除此upstream

```
check interval=3000 rise=2 fall=4 timeout=3000 type=http;
check_http_send "GET /catalog/do_not_delete/health_check/ HTTP/1.0\r\nUser-Agent:...\r\nHOST:Ims....\r\nmonitor-hj:7BCDB077276BEC8A9D2CE39548B5D57E\r\n\r\n";
check_http_expect_alive http_2xx http_3xx;
keepalive 300;
```


04

ApiGateway部署和性能

ApiGateway/SSL的部署以及性能参数

安装:`yum install apigateway redis -y`

服务器的内核参数:

<code>net.core.somaxconn = 655360</code>	系统设定的backlog 值，若listen 时backlog大于此值，则不会生效
<code>net.core.netdev_max_backlog = 6553600</code>	网卡设备的请求队列长度（硬件backlog）
<code>net.ipv4.tcp_max_tw_buckets = 50000</code> <code>net.ipv4.tcp_tw_timeout = 5</code> <code>net.ipv4.tcp_tw_reuse = 1</code>	端口回收，以及限制time_wait状态的tcp连接
<code>net.ipv4.ip_local_port_range = 1025 65535</code>	可用端口范围
<code>net.unix.max_dgram_qlen = 655360</code>	unix domain socket的数据包队列

ApiGateway的部署以及性能参数

Nginx的部署参数:

worker_processes`	8	nginx进程数，建议按照cpu数目来指定，一般为它的倍数
worker_cpu_affinity	00000001 00000010 00000100 00001000 00010000 00100000 01000000 10000000	为每个进程分配 cpu，上例中将 8 个进程分配到 8 个 cpu，当然可以写多个，或者将一个进程分配到多个 cpu
worker_rlimit_nofile	102400	这个指令是指当一个 nginx 进程打开的最多文件描述符数目
events: worker_connections accept_mutex multi_accept use epoll;	65535 Off On	每个进程允许的最多连接数，理论上每台nginx服务器的最大连接数为worker_processes*worker_connections
前端开启keepalive keepalive_timeout keepalive_requests	75 100	表示一条长连接可以保持的时间为75s 表示一条长连接可以处理的请求数100个
后端开启keepalive upstream keepalive location proxy_http_version	1000 1.1	其中keepalive 参数表示proxy与upstream间每个worker维持的长连接数，而location中需要将外部请求的Connection头部清空，并设置请求的http版本为1.1版。这些配置才能使能proxy 和upstream 间的长连接。

ApiGateway的部署和性能

ApiGateway SSL的压力测试数据(不跨网络):

请求次数	请求内容大小	TPS	完成时间	服务器最高负载	带宽(未跨网络)
43561497	4k	481628	90s	32	1.95GB/s
40028046	8k	442440	90s	32	3.48GB/s
32425127	16k	358258	90s	32	5.55GB/s
26482727	32k	293251	90s	32	7.98GB/s

ApiGateway SSL的压力测试数据:(跨网络)

请求次数	请求内容大小	TPS	完成时间	服务器最高负载	带宽
2487381	4k	27553	90s	<0.5	114.3MB/s
1312779	8k	14531	90s	<0.5	117.02MB/s
669161	16k	7406	90s	<0.5	117.54MB/s
337922	32k	3740	90s	<0.5	118.12MB/s

ApiGateway的部署和性能

ApiGateway 的压力测试数据(不跨网络):

请求次数	请求内容大小	TPS	完成时间	服务器最高负载	带宽(未跨网络)
1kw	8k	525974	86s	0.59, 0.19, 0.09	0
1kw	16k	99125.71	184s	0.68, 0.28, 0.13	0
1kw	32k	11700.03	1303s	0.22, 0.10, 0.08	0
1kw	64k	10206.13	7513s	0.18, 0.05, 0.01	0

ApiGateway 的压力测试数据:(跨网络)

请求次数	请求内容大小	TPS	完成时间	服务器最高负载	带宽
1kw	8k	14720.81	73s	0.59, 0.19, 0.09	114.3MB/s
1kw	16k	7823.42	104s	0.68, 0.28, 0.13	112.7MB/s
1kw	32k	3416.15	1710s	0.22, 0.10, 0.08	116.5MB/s
1kw	64k	1715.43	8583s	0.18, 0.05, 0.01	118.8MB/s

ApiGateway实施后的效果

- 1.SSL的吞吐能力增加1个数量级+
- 2.irules处理能力增加一倍+，并且横向扩展
- 3.更容易流量调度

ApiGateway实施过程中的TIP

1. 证书链/SNI
2. Nginx 加载配置文件
3. Lua的变量



ZHDBA.COM
中华数据库行业协会

QA