ALIBABA CLOUD

阿里云

负载均衡

应用型负载均衡ALB

文档版本：20220324

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

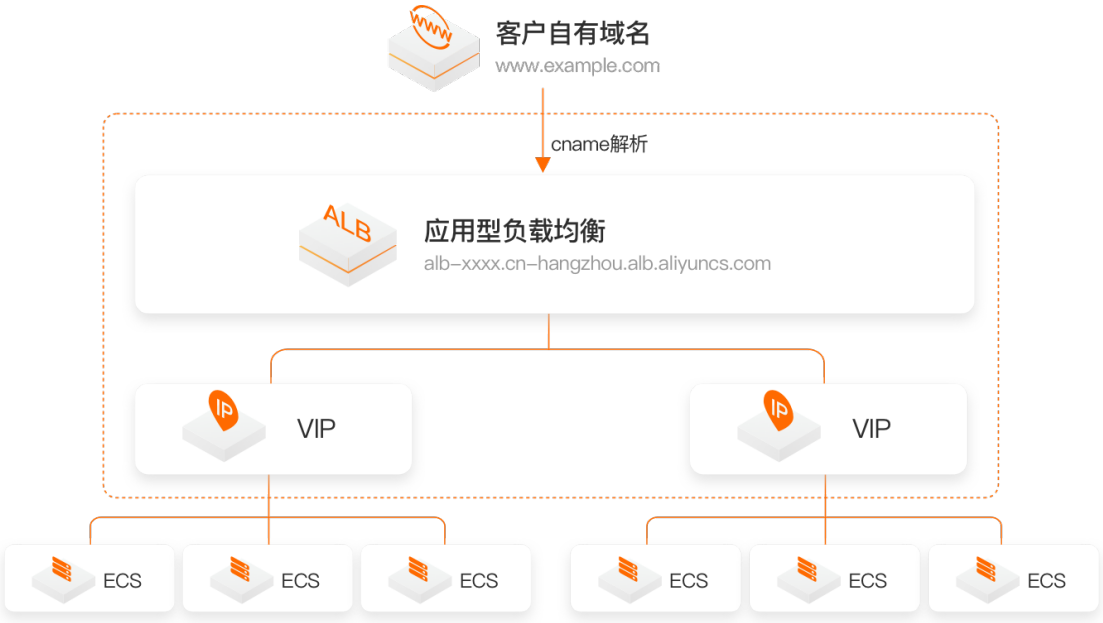
格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.ALB产品简介

1.1. 什么是应用型负载均衡ALB

什么是应用型负载均衡ALB
应用型负载均衡ALB
（Application Load Balancer）是阿里云推出的专门面向HTTP、HTTPS和QUIC等应用层负载场景的负载均衡服务，具备超强弹性及大规模应用层流量处理能力。
ALB
具备处理复杂业务路由的能力，与云原生相关服务深度集成，是阿里云官方提供的云原生Ingress网关。



为什么选择

应用型负载均衡ALB

应用型负载均衡ALB
，提供强大的应用层处理能力和丰富的高级路由功能，聚焦HTTP、HTTPS和QUIC应用层协议，是阿里云官方云原生Ingress网关。

应用层高弹性
ALB面向应用层，提供域名与VIP，多级分发承载海量请求。ALB支持通过流量分发扩展应用系统的服务能力，消除单点故障提升应用系统的可用性。ALB允许自定义可用区组合和在可用区间弹性伸缩，避免单可用区资源瓶颈。

先进的协议支持
ALB支持HTTP、HTTPS和QUIC协议，具备超大规模的流量处理能力。在实时音视频、互动直播和游戏等移动互联网应用中，访问速度更快，传输链路更安全可靠。ALB支持gRPC框架，可实现海量微服务间的高效API通信。

基于内容的高级路由
ALB支持基于路径、HTTP标头、查询字符串、HTTP请求方法、Cookie和SourceIp等多种条件来识别特定业务流量，

并将其转发至不同的后端服务器。同时ALB还支持重定向、重写以及自定义HTTPS标头等高级操作。

安全可靠

ALB自带DDoS防护，可一键集成Web应用防火墙。同时ALB支持全链路HTTPS加密，支持TLS 1.3等高效安全的加密协议，面向加密敏感型业务，满足Zero-Trust新一代安全技术架构需求；支持预制和自定义安全策略。

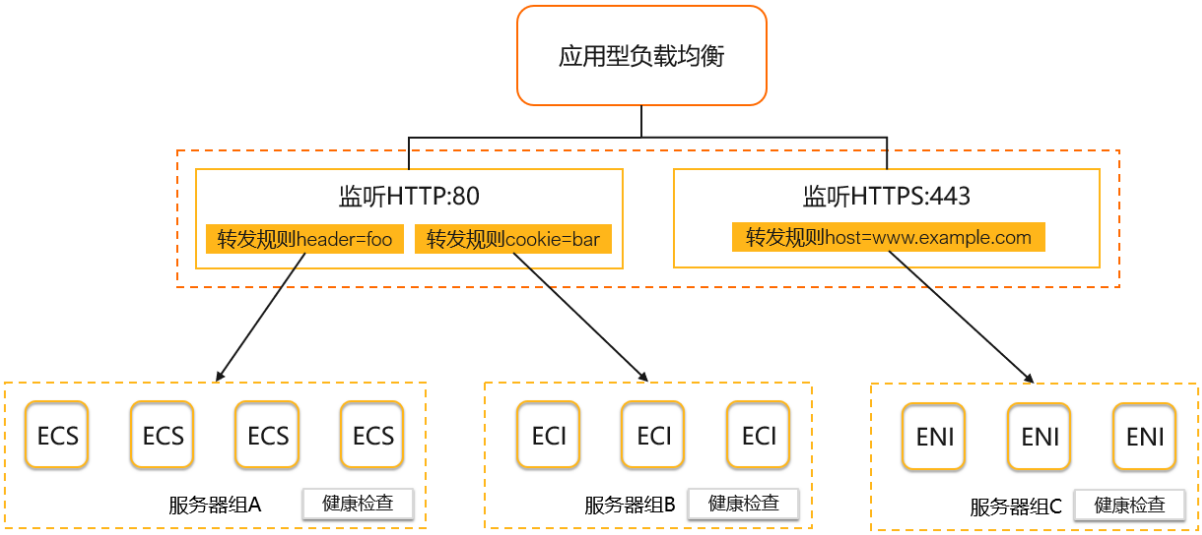
面向云原生

随着云原生逐步成熟，互联网、金融、企业等行业新建业务时都会选择云原生部署，或对现有业务进行云原生化改造。ALB与容器服务Kubernetes版、SAE、函数计算和开源K8s等深度集成，是阿里云的官方云原生Ingress网关。

弹性灵活的计费

ALB通过弹性公网IP（Elastic IP Address，简称EIP）和共享带宽提供公网能力，实现公网灵活计费；同时采用了更先进的、更适合弹性业务峰值的性能容量单位LCU（Loadbalancer Capacity Unit）的计价方案。

ALB
组成

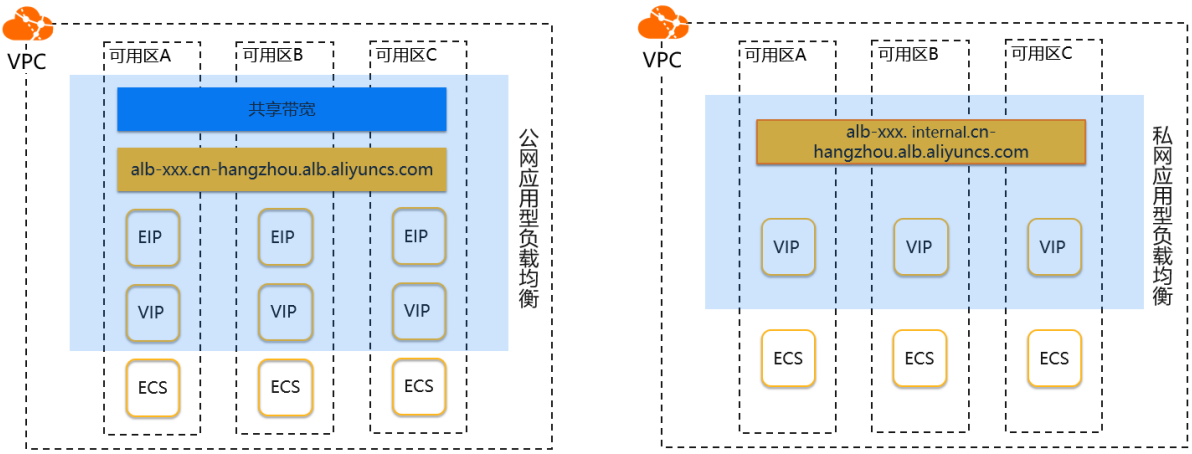


概念	说明
实例	面向七层，提供了超强七层负载均衡能力，通过将流量分发到不同的后端服务器来扩展应用系统的服务吞吐能力。单实例可处理高达100万QPS。

概念	说明
监听	监听是 ALB 最小业务单元，监听上需要配置协议与端口以告知 ALB 需要处理什么流量，例如HTTP协议，80端口。每个 ALB 至少有一个监听，才能开始流量处理与分发。每个 ALB 最多可以配置50个监听，用于处理不同的业务流量。
转发规则	转发规则用于确定 ALB 实例如何将请求路由到一个或多个后端服务器组中的后端服务器。 ALB 具备强大的高级路由能力，在传统的路由规则基础上，还可以基于HTTP标头、Cookie和HTTP请求方法等多种规则进行转发，实现基于业务的灵活调度。
服务器组	服务器组是一个逻辑组，包含多个后端服务器用于处理 ALB 分发的业务请求。 ALB 中服务器组独立于 ALB 存在，可以将同一服务器组挂载在不同 ALB 内。服务器组最大可以包含1000个后端服务器。 ALB 服务器组支持云ECS、ECI、ENI等多种类型的后端服务器。
健康检查	ALB 通过健康检查来判断后端服务器的业务可用性。 ALB 探测服务器组中不健康的服务器，并避免将流量分发给不健康的服务器。 ALB 支持丰富灵活的健康检查配置，如协议、端口、以及各种健康检查阈值。同时 ALB 提供健康检查模板，可将健康检查模板快速地应用到不同的服务器组。

ALB
类型
阿里云提供公网和私网两种类型的

ALB
。您可以根据业务场景选择配置对外公开或对内私有的
ALB
，系统会根据您的选择来决定是否使用共享带宽和弹性公网IP。



上图中半透明框中所有元素分别实现了一个面向公网（私网）的

ALB

。

概念	说明
共享带宽	共享带宽提供地域级的带宽共享和复用能力，以及按带宽计费和按增强型95计费等多种计费模式，可有效节省公网带宽使用成本。公网 ALB 中将使用共享带宽来提供增强型95计费和按带宽计费能力。
域名	一个在公网（私网）上可解析的域名解析至对应的VIP。您也可以将所拥有的可读性强的域名通过CNAME方式解析到 ALB 的域名上来使用，具体操作，请参见设置CNAME域名解析。
EIP	您仅在创建公网 ALB 时需要使用EIP，在创建私网 ALB 时无需配置。 ALB 对公网服务的IP地址，一个公网 ALB 可以有多个EIP。为了实现高可用性，一个公网 ALB 至少应包含两个分布在不同可用区的EIP。
VIP（Virtual IP address）	ALB 实施流量分发的实体。每个VIP都是专有网络VPC（Virtual Private Cloud）中的一个私网IP地址。

开通应用型负载均衡ALB

单击下方按钮可立即前往

ALB

产品购买页面。

创建应用型负载均衡ALB

相关文档

- 版本功能对比和使用限制
- ALB快速入门
- ALB计费项概述

1.2. 版本功能对比和使用限制

ALB基础版和标准版功能对比和使用限制
应用型负载均衡ALB
包含基础版和标准版，两个版本支持的功能和限制规则存在差异。本文介绍
ALB
的基础版和标准版的功能对比和限制规则。

功能差异

ALB
的基础版和标准版的功能差异如下：

功能	基础版	标准版
监听协议		
QUIC协议	支持	支持
HTTP/2协议	支持	支持
WEBSOCKET协议	支持	支持
转发规则		
基于域名或路径匹配	支持	支持
基于HTTP Header匹配	支持	支持
基于查询字符串匹配	不支持	支持
基于Cookie匹配	不支持	支持
基于HTTP请求方法匹配	不支持	支持
基于源IP匹配	不支持	支持
基于响应中的状态码匹配	不支持	支持
基于响应中的Header匹配	不支持	支持
重定向	支持	支持
重写或返回固定响应	不支持	支持
写入Header或删除Header	不支持	支持

功能	基础版	标准版
流量镜像	不支持	支持（白名单开放，请提交工单申请）
QPS限速	不支持	支持（白名单开放，请提交工单申请）
可编程脚本AScript	不支持	支持（白名单开放，请提交工单申请）
安全		
访问控制黑白名单	支持	支持
TLS加密算法套件选择	支持	支持
SNI多证书支持	支持	支持
RSA&ECC双证书	支持	支持
ECC证书	支持	支持
全链路HTTPS	不支持	支持
自定义TLS算法套件	不支持	支持
TLS 1.3协议支持	支持	支持
监控统计		
访问日志	支持	支持

使用限制

ALB	基础版	标准版
实例		
一个ALB实例可添加的扩展证书数（不计入默认证书）	10	25
一个ALB实例可添加的转发规则数（不计入默认规则）	40	100
一个ALB实例可添加的监听数	50	50

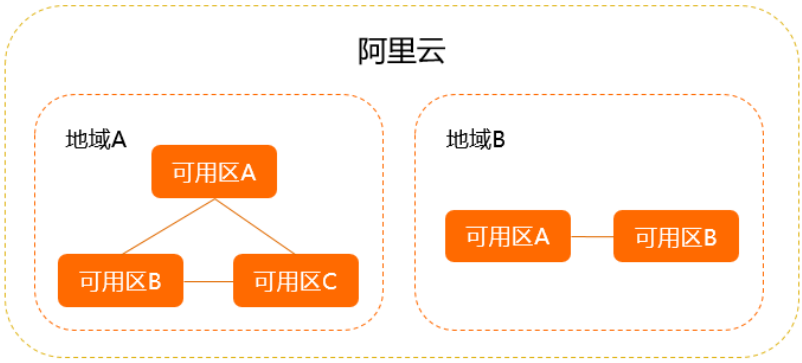
ALB	基础版	标准版
一个 ALB 实例可添加的后端服务器数	200	1000
监听		
一个监听可关联的访问控制数	3	3
一个监听可关联的访问控制条目数	300	1000
地域		
一个地域可支持的自定义安全策略数	50	
一个地域可支持的健康检查模板数	50	
一个地域可支持的 ALB 实例数	60	
一个地域可支持的访问控制数	1000	
一个地域可支持的服务器组数	3000	
服务器组		
一个服务器组可被关联到的监听或转发规则数	50	
同一个后端服务器（IP）可被添加到 ALB 后端服务器组的次数	200	
一个服务器组可添加的后端服务器数（IP和端口）	1000	
转发规则		
一条转发规则可添加的动作数	3	5
一条转发规则可添加的匹配评估数	5	10
一条转发规则可添加的通配符数	5	10
访问控制和安全策略		
一个访问控制可关联的监听数	50	
一个访问控制可添加的条目数	1000	
一个自定义安全策略可以关联的监听数	10	

1.3. 支持的地域与可用区

本文介绍
应用型负载均衡ALB
支持的地域（Region）与可用区（Availability Zone，简称AZ）。

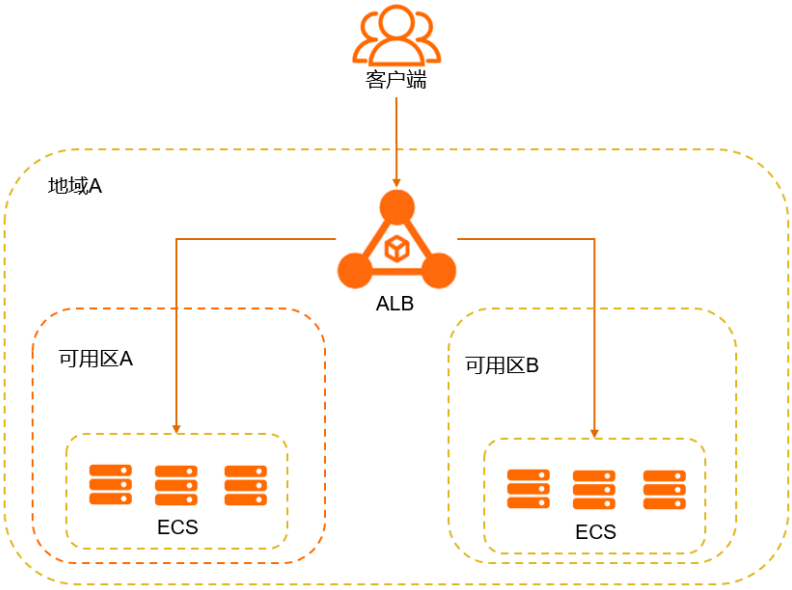
地域和可用区概述

地域是指物理数据中心，资源创建成功后不能更换地域。可用区是指在同一地域内，电力和网络互相独立的技术设施区域，即一个可用区出现基础设施故障不影响另外一个可用区。每个地域完全独立，不同地域的可用区完全隔离，但同一个地域内的可用区之间使用低时延链路相连。



为了向广大用户提供更加稳定可靠的负载均衡服务，

ALB
可将流量跨可用区分发，将业务负载到您选择的可用区，不同可用区间建立实时的业务容灾，某个可用区故障不影响其它可用区的流量转发。



ALB
支持的地域与可用区

地域	可用区
华东1（杭州）	可用区H、可用区I、可用区G、可用区J、可用区K

地域	可用区
华东2（上海）	可用区E、可用区F、可用区G、可用区L
华南1（深圳）	可用区D、可用区E
西南1（成都）	可用区A、可用区B
华北1（青岛）	可用区B、可用区C
华北2（北京）	可用区H、可用区G、可用区K
华北3（张家口）	可用区A、可用区B、可用区C
华北6（乌兰察布）	可用区A、可用区B
中国（香港）	可用区B、可用区C
华南3（广州）	可用区A、可用区B
新加坡	可用区B、可用区C
马来西亚（吉隆坡）	可用区A、可用区B
澳大利亚（悉尼）	可用区A、可用区B
印度（孟买）	可用区A、可用区B
日本（东京）	可用区A、可用区B
印度尼西亚（雅加达）	可用区A、可用区B
德国（法兰克福）	可用区A、可用区B
美国（弗吉尼亚）	可用区A、可用区B
美国（硅谷）	可用区A、可用区B

ALB
金融云支持的地域与可用区

地域	可用区
华东1（杭州）	可用区J、可用区K
华东2（上海）	可用区K、可用区Z
华南1（深圳）	可用区D、可用区E

相关文档

- DescribeRegions：查询可用地域。
- DescribeZones：查询一个地域下的可用区列表。

2.ALB快速入门

阿里云
应用型负载均衡ALB
支持HTTP、HTTPS和QUIC协议，专门面向网络应用层，提供强大的业务处理能力。本文介绍如何快速创建一个
ALB
实例，并将来自客户端的访问请求转发到后端服务器。

操作流程

1. 准备工作

   搭建负载均衡服务前，您需要根据业务需求规划
   ALB
   实例的地域，创建专有网络VPC（Virtual Private Cloud）和后端服务器ECS实例等。

2. 步骤一：创建实例

   使用负载均衡服务时，您需要先创建一个
   ALB
   实例，每一个
   ALB
   实例代表着一个负载均衡服务实体。

3. 步骤二：创建后端服务器组

   您需要创建服务器组并添加后端服务器来接收
   ALB
   转发的客户端请求。

4. 步骤三：配置监听

   您需要为实例配置监听，检查连接请求，然后根据调度算法定义的转发策略将客户端请求分发至后端服务器。

5. （可选）步骤四：设置域名解析

   ALB
   支持将您拥有的常用域名通过CNAME方式解析到
   ALB
   实例的公网服务域名上，使您可以更加方便地访问各种网络资源。

准备工作

您需要根据业务需求规划

ALB
实例的地域，创建VPC和后端服务器ECS实例等。

- 规划
  ALB
  实例的地域。确保ECS实例的地域和
  ALB
  实例的地域相同，并且ECS实例与
  ALB
  实例属于同一个VPC。此外，建议您将ECS实例部署在不同的可用区内，提高业务的可用性。

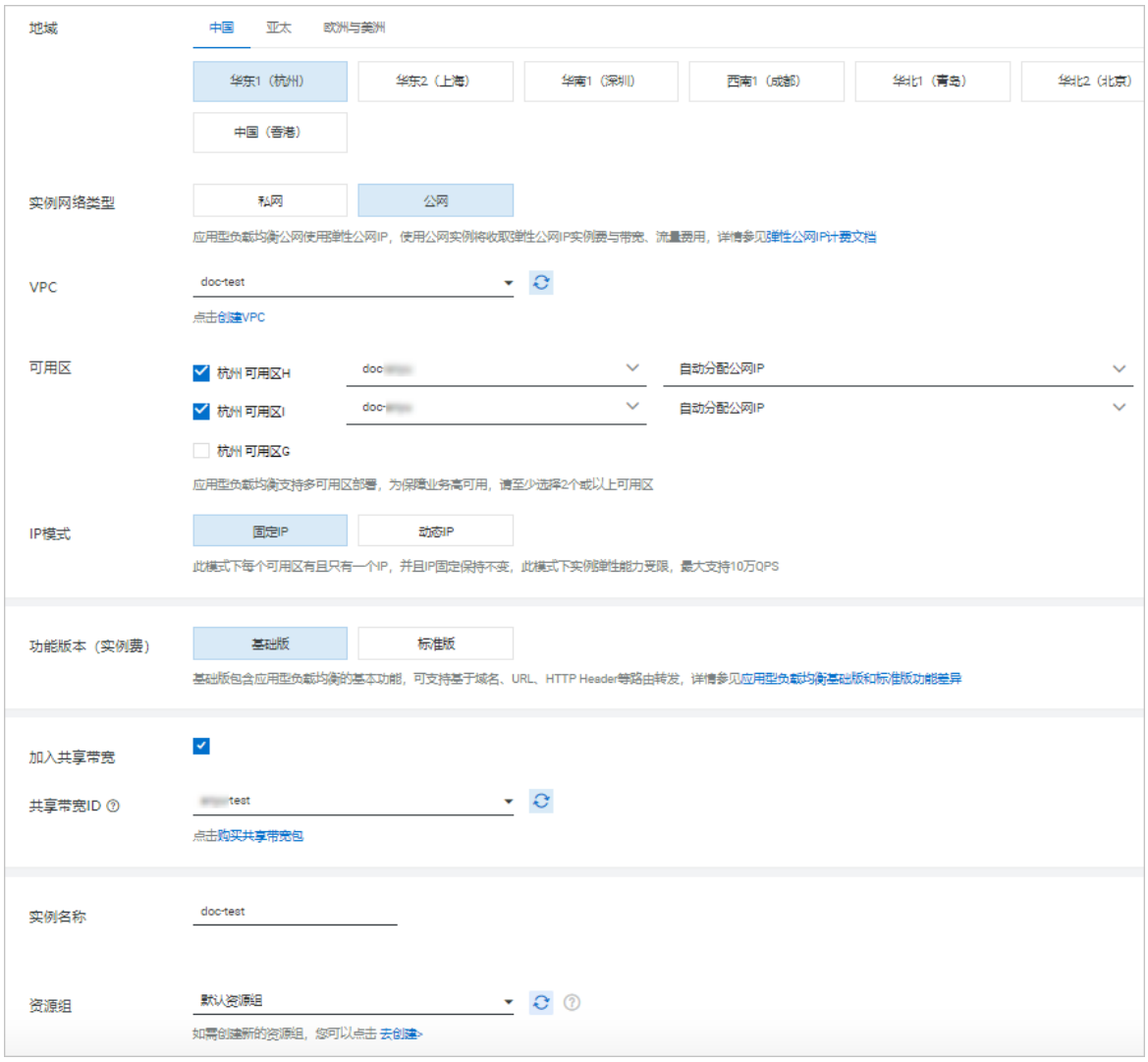
- 创建专有网络
- 使用向导创建实例

步骤一：创建

ALB

实例

1. 登录应用型负载均衡ALB控制台。
2. 在**实例**页面，单击**创建应用型负载均衡**。
3. 在**应用型负载均衡（按量付费）**购买页面，根据需要配置实例。



配置	说明
地域	选择实例所属的地域。

配置	说明
实例网络类型	选择实例网络类型，系统会根据您的选择分配私网或公网服务地址。 ○ **私网**：提供私网IP，只能通过阿里云内部网络访问该负载均衡服务，无法从互联网访问。 ○ **公网**：提供公网IP，可以通过互联网访问负载均衡服务。公网ALB通过弹性公网IP（Elastic IP Address，简称EIP）提供公网能力，选择公网将会收取弹性公网IP实例费、带宽与流量费用。 ■ 选择**自动分配公网IP**：系统将帮您自动创建后付费的按流量计费的EIP，并绑定到ALB。 ■ 选择已有的EIP：您可以指定已经创建过的EIP并绑定至新购的ALB实例上。 **注意** 具有以下特征的EIP不支持绑定至ALB实例。 ■ 包年包月类型的EIP。 ■ 选择按固定带宽计费的按量付费EIP。 ■ 已加入其它共享带宽的EIP。
VPC	选择实例所属的VPC。
可用区	i. 选择至少2个或以上的可用区。 ii. 分别在所选可用区内选择vSwitch，如果可用区下无vSwitch，请根据控制台提示创建vSwitch。 iii. （可选）分别在所选可用区内选择EIP，如果可用区下无EIP，可保持默认选项**自动分配公网IP**，ALB将为您自动购买并绑定一个EIP。
IP模式	选择实例的IP地址模式。 ○ **固定IP**：每个可用区有且只有一个IP，并且IP地址保持固定不变。此模式下实例弹性能力有限，最大支持10万QPS。 ○ **动态IP**：每个可用区至少有一个IP，随着业务请求的增加，会自动扩展IP数量。此模式下实例具备良好的弹性能力。
功能版本（实例费）	选择实例的功能版本。 ○ **基础版**：包含应用型负载均衡的基本功能，可支持基于域名、URL、HTTP Header等路由转发。 ○ **标准版**：在基础版的功能基础上，还包含自定义TLS安全策略，重定向、重写等高级路由功能。 关于基础版和标准版功能差异的更多信息，请参见版本功能对比和使用限制。

配置	说明
加入共享带宽	选择是否要加入共享带宽。如选中**加入共享带宽**，则需选择共享带宽包，如果没有共享带宽包可选择，可单击**购买共享带宽包**并完成购买，然后返回ALB 购买页面单击 图标，即可选择共享带宽包。
公网计费方式	公网计费方式默认选项为**按流量计费**：带宽峰值不作为业务承诺指标，仅作为参考值和带宽上限峰值。当出现资源争抢时，带宽峰值可能会受到限制。 **说明** 该参数仅在**实例网络类型**为**公网**，并且未选择**加入共享带宽**时有效。
实例名称	自定义实例名称。
资源组	选择云资源所属的资源组。
关联角色创建须知	首次创建应用型负载均衡需要一个服务关联角色，允许应用型负载均衡访问您的弹性网卡，安全组，弹性公网IP，共享带宽包等产品服务。更多信息，请参见ALB服务关联角色。

4. 单击**立即购买**，然后根据控制台提示完成实例开通。
5. 返回**实例**页面，选择对应的地域即可看到新建的实例。

步骤二：创建后端服务器组

1. 在左侧导航栏，选择**应用型负载均衡ALB > 服务器组**。
2. 在**服务器组**页面，单击**创建服务器组**。
3. 在**创建服务器组**对话框配置服务器组相关的参数，然后单击**创建**。

参数	描述
服务器组类型	选择一种服务器组类型： ○ **服务器类型**：按照ECS实例添加后端服务器。 ○ **IP类型**：按照IP地址添加后端服务器。 ○ **函数计算类型**：按照函数形式添加后端服务器。 本文选择**服务器类型**。
服务器组名称	输入服务器组名称。
VPC	从VPC下拉列表中选择ECS实例所属的VPC。

参数	描述
选择后端协议	选择一种后端协议，本文选择**HTTP**。
选择调度算法	选择一种调度算法，本文选择**加权轮询**。
选择资源组	在下拉列表选择所属的资源组。
开启会话保持	开启会话保持功能后，负载均衡会把来自同一客户端的访问请求分发到同一台后端服务器上进行处理。本文选择关闭会话保持。
配置健康检查	本文选择开启健康检查，并保持默认设置。

4. 在**服务器组创建成功**弹窗单击**添加后端服务器**。
5. 在**后端服务器**页签单击**添加后端服务器**。
6. 在**添加后端服务器**面板，选择已创建的ECS实例，然后单击**下一步**。
7. 为已添加的服务器设置端口和权重，然后单击**确定**。
8. 返回**服务器组**页面，查看完成配置的服务器组。

步骤三：配置监听

1. 在左侧导航栏，选择**应用型负载均衡ALB > 实例**。
2. 在**实例**页面，在实例**操作**列单击**创建监听**。
3. 在**配置监听**配置向导页面，完成以下配置，然后单击**下一步**。
   - **选择负载均衡协议**：选择监听的协议类型。本文选择**HTTP**。
   - **监听端口**：用来接收请求并向后端服务器进行请求转发的监听端口，端口范围为1~65535，本文填写*80*。
   - **监听名称**：输入自定义监听名称。
   - **高级配置**：本文保持默认，可单击**修改**进行设置。更多参数说明，请参见添加HTTP监听。
   - **WAF安全防护**：可为监听开启WAF安全防护，选中后可在**开通WAF防护能力**对话框确认该能力并单击**确认开启**。
4. 在**选择服务器组**配置向导页面，选择已创建的后端服务器组，用于处理
   ALB
   实例接收到的访问请求。然后单击**下一步**。
5. 在**配置审核**配置向导页面，确认监听配置信息，然后单击**提交**。
6. 在实例的**监听**页签，查看已配置的监听信息。

您可以定义
ALB
实例的监听转发规则，设置

ALB
实例如何将请求转发到后端服务器组中的后端服务器。具体操作，请参见管理监听转发规则。

（可选）步骤四：设置域名解析

ALB
支持将您拥有的常用域名通过CNAME方式解析到
ALB
实例的公网服务域名上，使您可以更加方便地访问各种网络资源。更多信息，请参见设置CNAME域名解析。

1. 在左侧导航栏，选择**应用型负载均衡ALB > 实例**。
2. 在**实例**页面，复制已创建的
   ALB
   实例的DNS名称。
3. 完成以下步骤来添加CNAME解析记录。
   i. 登录域名解析控制台。
   ii. 在**域名解析**页面单击**添加域名**。
   iii. 在**添加域名**对话框中输入您的主机域名，然后单击**确定**。

   > **注意** 您的主机域名需已完成TXT记录验证。

   iv. 在目标域名的**操作**列单击**解析设置**。
   v. 在**解析设置**页面单击**添加记录**。
   vi. 在**添加记录**面板配置以下信息完成CNAME解析配置，然后单击**确认**。

配置	说明
记录类型	在下拉列表中选择**CNAME**。
主机记录	您的域名的前缀。
解析线路	选择默认。
记录值	输入加速域名对应的CNAME地址，即您复制的 ALB 实例的DNS名称。
TTL	全称Time To Live，表示DNS记录在DNS服务器上的缓存时间，本文使用默认值。

   > **说明**
   > - 新增CNAME记录实时生效，修改CNAME记录取决于本地DNS缓存的解析记录的TTL到期时间，一般默认为10分钟。
   > - 添加时如遇添加冲突，请换一个解析域名。更多信息，请参见解析记录互斥规则。

4. 验证CNAME配置是否生效。

   在浏览器中输入自定义的域名，如果能正常访问应用服务则说明CNAME解析配置已生效。关于CNAME解析的验证操作，请参见解析生效测试方法。

释放

ALB
实例
释放

ALB
实例后，您无需为
ALB
实例付费，但绑定的后端服务器仍会照常计费。

> **说明** 您只有关闭**删除保护**开关，才能释放实例，否则系统会报错。

1. 在左侧导航栏，选择**应用型负载均衡ALB > 实例**。
2. 在目标实例**操作**列选择 ⋮ **> 释放**。
3. 在**释放实例**对话框，单击**确定**。

相关文档

介绍类文档：

- 什么是应用型负载均衡ALB
- 版本功能对比和使用限制
- 支持的地域与可用区

API文档：

- CreateLoadBalancer：调用CreateLoadBalancer接口创建负载均衡实例。
- CreateServerGroup：调用CreateServerGroup接口创建服务器组。
- CreateListener：调用CreateListener接口创建HTTP、HTTPS或QUIC监听。
- DeleteLoadBalancer：调用DeleteLoadBalancer接口删除负载均衡实例。

3.ALB产品计费

3.1. ALB计费项概述

本文介绍
应用型负载均衡ALB
的计费组成和相关定价。

ALB
的计费组成

ALB
目前支持按量付费。
ALB
的费用由三部分组成：实例费、性能容量单位LCU（Loadbalancer Capacity Unit）费和公网网络费。

实例网络类型	实例费	性能容量单位LCU费	公网网络费
公网	包含	包含	包含
私网	包含	包含	不涉及

实例费

ALB
的实例费目前按小时收取，不足1小时按1小时算。
ALB
实例费的计费周期为从创建到释放的时间段。

> **说明** 下表中的价格仅供参考，实际价格请以购买页为准。

功能版本	实例单价（元/小时）
基础版	0.049
标准版	0.147

性能容量单位LCU费

性能容量单位LCU是用来衡量

ALB
处理流量时的性能指标。

性能容量单位LCU衡量指标简介

指标名称	描述	单个LCU对应的性能	LCU换算值
新建连接数	每秒处理的新建连接数量。	25个/秒	新建连接数÷25
并发连接数	每分钟内活跃连接的数量。	3000个/分	并发连接数÷3000

指标名称	描述	单个LCU对应的性能	LCU换算值
处理数据量	ALB 处理的HTTP(S) 请求和响应的数据处理量，单位为GB。	1 GB/时	处理数据量÷1
规则评估数	指 ALB 处理的规则总数与每秒请求数（QPS）的乘积。 • 当处理的规则数量均未超过免费额度时：规则评估数=QPS • 当处理的规则数量超过免费额度时：规则评估数=QPS×(超出部分转发规则数+超出部分AScript可编程脚本行数+超出部分扩展证书个数) 注意 • 影响规格评估数的指标包含转发规则数，AScript可编程脚本行数和扩展证书个数，这三个指标的免费额度均为25。上述三个指标只有超过免费额度的部分会被收费，未超过免费额度的指标无需计算。 • 2022年03月22日起，转发规则数的免费额度由10个提升至25个。 • ALB 可编程脚本AScript配置转发规则功能目前白名单开放，如需体验请提交工单，更多信息，请参见可编程脚本AScript概述。	1000/个	规则评估数÷1000

在1小时内消耗的LCU数量根据上述4个指标进行换算，按照LCU使用量最高的指标来进行付费。

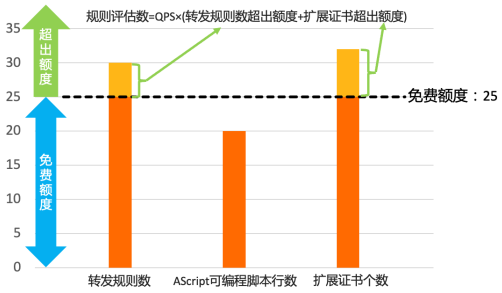
性能容量单位LCU费=max{**新建连接数LCU值，并发连接数LCU值，处理数据量LCU值，规则评估数LCU值**}×**LCU单价**

说明 下表中的价格仅供参考，实际价格请以购买页为准。

计费项	LCU单价（元/个）
性能容量单位LCU	0.049

LCU费用计算示例

指标名称	运行示例	计算过程	对应的LCU使用量
新建连接数（秒）	平均每秒有100个	本示例中平均每秒有100个新建连接，而每个LCU每秒提供25个新建连接，则可换算的LCU数为： `100÷25=4`	4

指标名称	运行示例	计算过程	对应的LCU使用量
并发连接数（分钟）	每个连接持续3分钟，每个连接每秒发送4个请求。	本示例中每秒有100个新建连接，则： **每分钟的新建连接数**=100×60=6000（个） 每个新建连接持续3分钟，则每分钟的并发连接数是当前分钟内的新建连接数与前两分钟内的新建连接数之和： **每分钟的并发连接数** =6000+6000+6000=18000（个） 每个LCU每分钟提供3000个并发连接，则可换算的LCU数为： 18000÷3000=6	6
处理数据量（小时）	每秒1000 KB	每秒处理1000 KB数据量，则每小时处理的数据量为： 1000 KB×60×60=3,600,000 KB=3.6 GB 每个LCU每小时提供1 GB的处理数据量，则可换算的LCU数为： 3.6÷1=3.6	3.6
规则评估数（秒）	• 30条转发规则 • 20行AScript可编程脚本 • 32个扩展证书	平均每秒有100个新建连接，每个连接平均每秒发送4个请求，则每秒接收请求量为： 100×4=400（个）  本示例中，您配置的转发规则数和扩展证书数超过了免费配额，则每秒产生的收费规则评估为： [(30-25)+(32-25)]×400=4800（个） 每个LCU每秒提供1000个规则评估数，则可换算的LCU数为： 4800÷1000=4.8	4.8

指标名称	运行示例	计算过程	对应的LCU使用量

在本示例中，LCU消耗最大的指标是并发连接数（6个LCU），因此LCU费用按照并发连接数计算。

每小时的LCU费用=0.049元/个×6个=0.294元
按上述消耗情况估算，平均月度LCU费约为：0.294元×24小时×30天=211.68元

您可以使用ALB LCU估算器来预估LCU的消耗：

使用ALB LCU估算器

公网网络费

私网

ALB
不收取公网网络费用，只有当您购买公网
ALB
才会收取公网网络费用。公网
ALB
通过弹性公网IP（EIP）提供公网能力，选择公网
ALB
将会收取EIP实例费、带宽与流量费用，更多信息，请参见按量付费。

相关文档

- 什么是应用型负载均衡ALB
- 版本功能对比和使用限制

- 管理监听转发规则
- 可编程脚本AScript概述
- 管理证书

3.2. 应用型负载均衡ALB资源包

3.2.1. ALB资源包简介

ALB
资源包是
应用型负载均衡ALB
推出的预付费套餐包，购买生效后可按抵扣比例自动抵扣
ALB
产生的LCU费和实例费。与按量付费相比，
ALB
资源包享有更高的折扣优惠，并可根据业务需要灵活设置
ALB
资源包的生效时间，适合有固定预算开支的企业。

产品特点

- 即开即用，无需额外配置
  购买时可以选择生效时间，生效后自动抵扣，无需您手动配置。
- 多种规格，按需选择
  支持1000～100000 LCU（Loadbalancer Capacity Unit）多种规格，可叠加购买，满足中小企业及大企业的抵扣需求。
- 节约成本
  ALB
  资源包相比后付费价格更优惠，且规格越大单价越低，
  ALB
  搭配
  ALB
  资源包可以帮助您节约网络成本。

ALB资源包区域划分

资源包区域	支持抵扣的地域
中国内地区域	华北1（青岛）、华北2（北京）、华北3（张家口）、华北6（乌兰察布）、华东1（杭州）、华东2（上海）、华南1（深圳）、西南1（成都）。
中国香港及海外区域	中国（香港）、日本（东京）、新加坡（新加坡）、澳大利亚（悉尼）、马来西亚（吉隆坡）、印度尼西亚（雅加达）、美国（弗吉尼亚）、美国（硅谷）、德国（法兰克福）、英国（伦敦）。

资源包规格及定价

> **说明** 下表中的价格仅供参考，实际价格请以购买页为准。

名称	规格	定价（元）
微型资源包	1,000 LCU	49
小型资源包	10,000 LCU	485
中型资源包	50,000 LCU	2377
大型资源包	100,000 LCU	4655

ALB资源包购买和抵扣规则

规则	描述
ALB 资源包购买规则	• ALB 资源包购买后可以选择支付后立即开通或指定时间开通。 • 支持叠加购买，不设置上限。多个 ALB 资源包按资源包到期时间进行抵扣，优先抵扣先到期的 ALB 资源包。 **说明** 如果实际使用的LCU量超出了已购买的 ALB 资源包规格，超出部分将自动转为按量付费。
支持购买的账号	暂时只支持中国站账号购买。
可抵扣的计费项	实例费和LCU费。 **注意** ALB 资源包不支持抵扣 ALB 的公网费用。
抵扣比例	• 实例费：基础版实例费按照每小时1 LCU消耗抵扣；标准版实例费按照每小时3 LCU消耗抵扣。 • LCU费：按照实际消耗量 `1:1` 抵扣LCU。

ALB
资源包常见问题

- ALB
  资源包何时生效？

- 在购买
  ALB
  资源包时选择**支付后立即开通**：表示
  ALB
  资源包支付后即时生效。
- 在购买
  ALB
  资源包时选择**指定时间开通**：
  ALB
  资源包将在您设置的指定时间生效。

- ALB
  资源包是先购买先抵扣吗？
  ALB
  按照
  ALB
  资源包生效后的到期时间进行先后抵扣，即优先抵扣先到期的
  ALB
  资源包。
- ALB
  资源包支持导出抵扣明细吗？
  ALB
  资源包支持查看具体的抵扣明细，包括抵扣实例ID、抵扣量、抵扣时间、抵扣的计费项等信息。具体操作，请参见查询资源包消费明细。
- ALB
  资源包支持退款吗？
  ALB
  资源包不支持退款，到期后未用完的
  ALB
  资源包将自动清零，不支持转移到其它资源包。
- ALB
  资源包单价比单独使用
  ALB
  按量付费更优惠吗？
  除微型
  ALB
  资源包跟按量付费价格相同外，其余规格均比ALB按量付费更优惠，且购买规格越大越优惠。
- ALB
  资源包支持叠加购买吗？
  可以叠加购买，同类型资源包按照到期的先后顺序进行抵扣。
- ALB
  资源包购买完成后需要设置吗？
  ALB
  资源包作为资源型产品，购买后免配置。您仅需要在购买时设置
  ALB
  资源包的生效时间。
- 如何查看
  ALB
  资源包的使用情况？

阿里云用户中心提供资源包概览和使用明细，可以根据**生效时间**或**资源包实例ID**搜索名下的资源包余量以及抵扣明细，更多信息，请进入资源包页面查看。

3.2.2. 购买ALB资源包

ALB
资源包是
ALB
针对实例费和LCU费推出的固定套餐，与按量付费相比，资源包享有更高的折扣优惠。本文介绍了如何购买
ALB
资源包和查询资源包的消费明细。

购买资源包

1. 登录ALB资源包购买页。
2. 在**ALB资源包**页面根据您的实际情况选择区域、资源包规格、购买数量、生效时间和资源包有效期等参数，然后单击**立即购买**。

3. 根据购买流程提示完成支付。

查询资源包消费明细

1. 登录应用型负载均衡ALB控制台。
2. 在左侧导航栏选择**应用型负载均衡 ALB > ALB资源包**。
3. 在**ALB资源包**页面单击**查看已有资源包**。
4. 在**资源实例管理**页面，您可以查看资源包的汇总信息和使用明细。
   - 在**实例汇总**页签，您可以查看资源包的归属账号、区域属性、资源包ID、规格、剩余量、生效时间和失效时间等信息。

- 在资源包列表区域右上角单击**设置额度预警**，可以设置资源包的额度预警阈值并订阅通知消息。

- 在资源包列表区域右上角单击**定制列**，可以自定义资源包的显示信息。
- 在资源包列表区域右上角单击**导出**，可以导出并下载资源报表总览表格。

- 单击**使用明细**页签，可以查看资源包的具体消费明细。
  - 通过选择**抵扣时间**，输入**资源实例ID**和**被抵扣实例ID**可筛选资源抵扣明细。
  - 在使用明细列表区域右上角单击**定制列**，可以自定义消费明细的显示信息。
  - 在资源包列表区域右上角单击**导出**，可以导出并下载资源抵扣明细表格。

相关文档

- ALB计费项概述
- ALB资源包简介

4.ALB用户指南

4.1. 日志与监控

4.1.1. 访问日志

应用型负载均衡ALB
联合日志服务（SLS）推出访问日志功能，您可以通过访问日志分析用户行为、了解用户的地域分布、排查问题等。

背景信息

负载均衡作为访问入口，承载着海量的访问请求，负载均衡支持将访问日志投递到日志服务，结合日志服务强大的大数据计算能力，您可以通过访问日志分析用户行为、了解用户的地域分布、进行问题排查等。负载均衡日志访问功能具有以下优势：

- 简单：将开发、运维人员从日志处理的繁琐耗时中解放出来，将更多的精力集中到业务开发和技术探索上。
- 海量：负载均衡的访问日志数据规模通常很大，处理访问日志需要考虑性能和成本问题。日志服务可以一秒钟分析一亿条日志，相较于自建开源方案有明显成本优势和性能优势。
- 实时：DevOps、监控、报警等场景要求日志数据的实时性。日志服务强大的大数据计算能力，可秒级分析处理实时产生的日志。
- 弹性：按负载均衡实例级别开通或关闭访问日志功能，Logstore容量可动态伸缩满足业务增长需求。

费用说明

ALB
将日志投递到日志服务后，日志服务根据存储空间、读取流量、请求数量、数据加工、数据投递等进行计费，更多信息，请参见日志服务计费。

前提条件

使用访问日志功能前，请确保您已开通日志服务。具体操作，请参见开通日志服务。

创建访问日志

1. 登录应用型负载均衡ALB控制台。
2. 在顶部菜单栏，选择实例的所属地域。
3. 在**实例**页面，单击目标实例ID。
4. 单击**实例详情**页签，在**实例属性**区域，单击**访问日志**后面的**创建**。
5. 在**访问日志**面板，配置**项目Project**和**日志库Logstore**，然后单击**确定**。
   - **项目Project**：日志服务中的资源管理单元，用于资源隔离和控制。
   - **日志库Logstore**：日志服务中日志数据的采集、存储和查询单元。

   配置完成后，日志服务默认为该Logstore设置索引，如果该Logstore已经设置了索引，原有的索引配置将被覆盖。
6. 单击**访问日志**后面的日志库链接打开日志服务控制台。

   日志服务采集到访问日志后，您可以执行查询分析、下载、投递、加工日志、创建告警等操作。具体操作，请参见云产品日志通用操作。

日志字段说明

字段	说明
app_lb_id	负载均衡实例ID。
__topic__	日志主题，固定为alb_layer7_access_log。
body_bytes_sent	发送给客户端的HTTP Body的字节数。
client_ip	请求客户端IP地址。
host	域名或IP地址。优先从请求参数中获取host，如果获取不到则从host header取值，如果还是获取不到则以处理请求的后端服务器IP地址作为host。
http_host	请求报文host header的内容。
http_referer	负载均衡收到的请求报文中HTTP的referer header的内容。
http_user_agent	负载均衡收到的请求报文中HTTP的user-agent header的内容。
http_x_forwarded_for	负载均衡收到的请求报文中x-forwarded-for的内容。
http_x_real_ip	客户端的真实IP地址。
read_request_time	负载均衡读取请求的时间，单位：毫秒。
request_length	请求报文的长度，包括startline、HTTP头报文和HTTP body。
request_method	请求报文的方法。
request_time	负载均衡收到第一个请求报文的时间到返回应答之间的时间间隔，单位：秒。
request_uri	负载均衡收到的请求报文的URI。
scheme	请求的schema：HTTP或HTTPS。
server_protocol	负载均衡收到的HTTP协议的版本，例如HTTP/1.0或HTTP/1.1。
slb_vport	负载均衡的监听端口。
ssl_cipher	建立SSL连接使用的密码，例如ECDHE-RSA-AES128-GCM-SHA256等。
ssl_protocol	建立SSL连接使用的协议，例如TLSv1.2。
status	负载均衡应答报文的状态。
tcpinfo_rtt	客户端TCP连接时间，单位：微秒。
time	日志记录时间。
upstream_addr	后端服务器的IP地址和端口。

字段	说明
upstream_response_time	从负载均衡向后端服务器建立连接开始到接受完数据然后关闭连接为止的时间，单位：秒。
upstream_status	负载均衡收到的后端服务器的响应状态码。
vip_addr	虚拟IP地址。
write_response_time	负载均衡写的响应时间，单位：毫秒。

4.2. 可编程脚本AScript

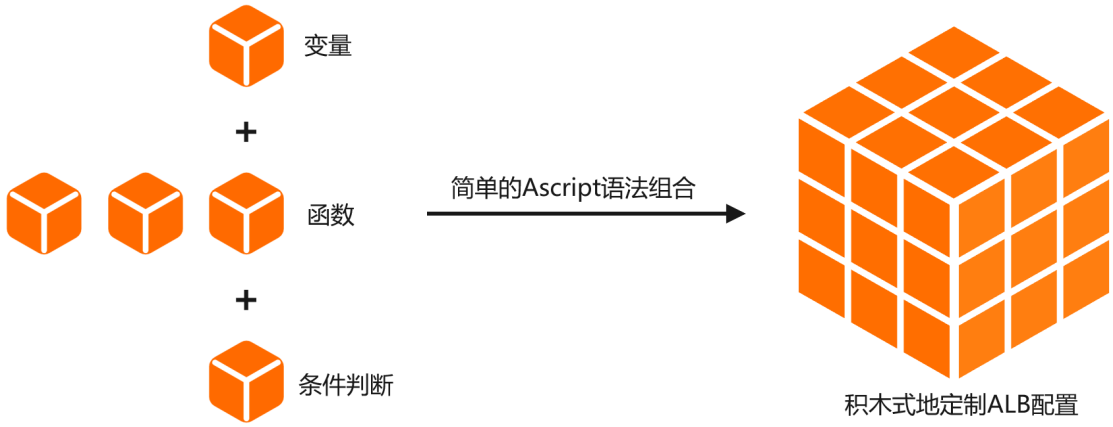
4.2.1. 可编程脚本AScript概述

AScript是应用型负载均衡ALB（Application Load Balancer）面向标准版实例推出的可编程脚本，其中A代表Aglie（敏捷）、Application（面向应用）和Aliyun（阿里云自研）。AScript有强大的自定义能力，当ALB
控制台上的标准配置无法满足您的业务需求时，可通过简单的可编程脚本AScript来实现功能的二次开发。

AScript简介

AScript通过简单易学的语法和庞大的函数库，能够积木式地组合出个性化的

ALB
配置。



AScript内置了可以识别的变量、简单的判断语句，同时提供了可直接调用的封装好的函数。通过简单的变量判断并调用现成的函数，即可满足您对转发规则的各类定制需求，有效地解决配置需求无法实现、业务变更不敏捷的问题。

关于AScript的语法的更多信息，请参见AScript语法。

限制说明

- ALB
  支持使用可编程脚本AScript来配置转发规则，该功能目前白名单开放，如需体验请提交工单或联系您的客户经理申请。
- 公网
  ALB

和私网
ALB
都支持使用可编程脚本来配置转发规则。

- ALB
  包含基础版和标准版，仅标准版
  ALB
  实例支持使用可编程脚本AScript来配置转发规则。

应用场景

场景	描述
防盗链需求	应用于自定义鉴权算法、User-Agent黑名单和Referer白名单等场景需求。基于请求参数、Cookie或其他复杂算法等各类鉴权需求，帮您快速实现鉴权，从而完成对资源的保护。
黑白名单管控	通过设置客户端IP的黑白名单，来完成权限管控。
请求头和响应头控制	可以使用AScript脚本对请求参数和请求头等变量进行灵活修改。
改写和重定向	通过改写URI、文件后缀、添加URI前缀、302重定向等操作，实现您的改写和重定向目标。 **说明** 多应用在多语言版本的网站之上，例如中文网站可能会302重定向到1个位置，英文网站或者德文网站可能会302重定向到不同的位置。

功能计费

目前仅标准版

ALB
实例支持使用可编程脚本AScript来配置转发规则。系统根据
ALB
实例中生效中的AScript代码行数和规则数来决定是否计费。

- 一个实例中生效的AScript代码行数小于20行时，不影响性能容量单位LCU（Loadbalancer Capacity Unit）中的规则评估数。

  评估规则数=QPS×(**评估规则数**-**免费规则数**10)

- 一个实例中生效的AScript代码行数大于20行时，通过影响每个LCU中规则评估数来计费：

  评估规则数=QPS×(**评估规则数**-**免费规则数**10)+QPS×(**生效的**AScript**代码总行数**-**免费的**AScript**代码行数**20)

规则评估数计算示例

一个

ALB
实例每秒的新建连接数有100个，每个连接持续3分钟，每个连接每秒发送4个请求，同时您创建了20条转发规则和30行AScript代码，那么您的规则评估数对应的LCU的计算过程如下：

1. 平均每秒有100个新建连接，每个连接平均每秒发送4个请求，则每秒接收请求量QPS为：

   100×4=400（个）

2. 您配置了20条转发规则和30行AScript代码，则每秒产生的收费规则评估数为：

```
(20-10)×400+(30-20)×400=8000（个）
```

3. 每个LCU每秒提供1000个规则评估数，则可换算的LCU数为：

```
8000÷1000=8
```

关于规则评估数如何影响LCU费，更多信息，请参见性能容量单位LCU费。

4.2.2. AScript原理介绍

本文为您介绍AScript的运行原理、规则模型和生效位置。

运行原理

您配置的AScript规则与应用型负载均衡ALB（Application Load Balancer）控制台上的标准配置一样，都是对

ALB
请求进行处理。AScript的执行位置如图所示，当客户端请求到达
ALB
监听后，
ALB
监听会根据您在控制台上配置的转发规则对请求进行处理。以
ALB
控制台上的标准配置为参照物，AScript可选择在规则处理前或规则处理后生效。

规则模型

AScript的规则模型如下：

- AScript的规则模型的核心出发点是将不同业务功能隔离至不同规则，以及控制规则的执行流。
- AScript的规则模型中的每条规则可以各自选择规则的执行位置。
- AScript的规则模型是以监听维度来进行设计的。

规则执行位置

AScript规则的执行位置包含请求方向规则执行前、请求方向规则执行后和响应方向规则执行前。

- 请求方向规则执行前：常用文件自动重命名、文件后缀小写化、添加URI前缀和文件后缀名改写等场景。
- 请求方向规则执行后：常用文件自动重命名、文件后缀小写化、添加URI前缀和文件后缀名改写等场景。
- 响应方向规则执行前：常用文件自动重命名等场景。

规则执行情况

规则执行情况字段详细说明：

- 规则ID：标识每条规则的执行情况。

- 规则ID代表的执行情况：

规则ID	执行情况
-1	默认值。
1	未执行。
2	执行命中。 当规则含有 `if condition {}` ，且 `condition` 为真。
3	执行未命中。 当规则含有 `if condition {}` ，且 `condition` 为假；或规则不包含 `if condition {}` 。
4	执行异常。常见的异常情况如下所示： ○ **400**： `not found inline func %s` ○ **401**： `not found argument %d in %s` ○ **402**： `mismatch string type of arg %d in %s` ○ **403**： `mismatch number type of arg %d in %s` ○ **404**： `mismatch table type of arg %d in %s` ○ **405**： `mismatch boolean type of arg %d in %s` ○ **406**： `mismatch function type of arg %d in %s` ○ **407**： `exceed the exec cputime limit %d-%d us` ○ **408**： `exceed the api call limit %s %d-%d times` ○ **409**： `exceed the max loops limit %d in m3u8_rewrite` ○ **410**： `exceed the max loops limit %d in foreach` ○ **499**： `unknwon reason`

- 执行耗时：
  - 单位：微秒us。
  - 默认值：-1。
  - 前端呈现的耗时区间分布：
    - 第1档：0~100us
    - 第2档：100~500us
    - 第3档：500~1000us
    - 第4档：1000~5000us
    - 第5档：5000~20000us
    - 第6档：20000~50000us
    - 第7档：>50000us
- AScript规则的中断执行：
  默认值：-1。

4.2.3. 添加和管理AScript可编程脚本规则

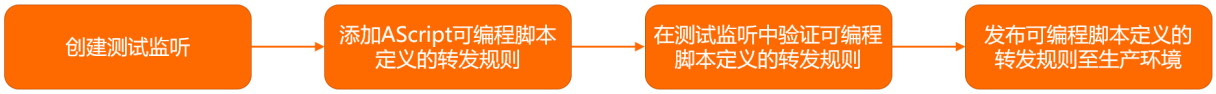
在应用型负载均衡ALB（Application Load Balancer）控制台上，您可以使用AScript可编程脚本创建转发规则，实现对
ALB
转发规则的定制化管理。本文介绍在
ALB
控制台上，如何添加AScript可编程脚本规则。

前提条件

您已创建了实验测试的标准版

ALB
实例。具体操作，请参见创建应用型负载均衡。

操作流程



创建测试监听

在实验测试ALB实例中创建一个HTTP、HTTPS或QUIC监听，本文以创建一个HTTP监听为例。

1. 登录应用型负载均衡ALB控制台。
2. 在顶部菜单栏，选择
   ALB
   的所属地域。
3. 选择以下一种方法，打开监听配置向导。
   - 在**实例**页面，在目标实例**操作**列单击**创建监听**。
   - 在**实例**页面，单击目标实例ID。在**监听**页签，单击**创建监听**。
4. 在**配置监听**配置向导，完成以下配置，然后单击**下一步**。

监听配置	说明
选择负载均衡协议	选择监听的协议类型。本文以选择**HTTP**为例。
监听端口	输入用来接收请求并向后端服务器进行请求转发的监听端口，端口范围为1~65535。 通常HTTP协议使用80端口，HTTPS协议使用443端口。 **说明** 在同一个负载均衡实例内，监听端口不可重复。 本文以输入**80**为例。
监听名称	输入监听名称。
高级配置	单击**修改**展开高级配置。本文以**高级配置**保持默认值为例。

5. 在**选择服务器组**配置向导，选择服务器组为**服务器类型**，并选择服务器组，然后单击**下一步**。
6. 在**配置审核**配置向导，确认配置信息，单击**提交**。

添加AScript可编程脚本定义的转发规则

1. 登录应用型负载均衡ALB控制台。

2. 在顶部菜单栏选择测试实例所在的地域。
3. 在实例列表页面单击目标实例ID。
4. 在**监听**页签单击创建的测试监听ID。
5. 在监听详情页面单击**转发规则**页签，然后在**转发规则**页面配置可编程脚本规则。
   i. 根据需求单击**请求方向转发规则**或**响应方向转发规则**。
   ii. 根据需求单击**在规则执行前添加可编程脚本**或**在规则执行后添加可编程脚本**。

   > 说明 选择**响应方向转发规则**时，只允许**在规则执行前添加可编程脚本**。

6. 在**添加可编程脚本规则**页面中添加规则，然后单击**确定**。

参数	是否必填	描述
规则名称	是	规则名称。长度限制为1~127个字符，必须以中文或字母开头，允许包含中英文字母、数字、短划线（-）、半角句点（.）和下划线（_）。
规则代码	是	规则代码。 ○ 您可以直接输入规则代码，也可以单击**使用代码模板**，选择对应的场景代码模板。 ○ 您可以按照使用场景编写规则代码。更多详情，请参见AScript场景示例。
执行位置	是	规则执行位置，不可更改。
启用状态	是	规则启用状态。
高级配置	否	当请求携带 `_es_dbg` 参数、且值为此处配置的密钥时，开启相应的调试响应头，以输出规则执行记录。

在测试监听中验证可编程脚本定义的转发规则

启动测试监听将实际的业务流量切换部分至测试监听在创建的测试监听中，测试规则。

1. 登录应用型负载均衡ALB控制台。
2. 在顶部菜单栏选择测试实例所在的地域。
3. 在实例列表页面单击目标实例ID。
4. 在**监听**页签单击创建的测试监听ID。
5. 在**监听详情**页签右上角单击**启动**，单击**转发规则**页签启用转发规则。

   启动监听后，根据配置的可编程脚本的转发规则来验证流量转发，验证通过后即可将可编程脚本配置的转发规则发布至生产环境。

发布可编程脚本定义的转发规则至生产环境

1. 登录应用型负载均衡ALB控制台。
2. 在顶部菜单栏选择生产实例所在的地域。
3. 在生产实例列表页面单击目标实例ID。
4. 在**监听**页签单击目标生产监听ID。
5. 在监听详情页面单击**转发规则**页签，然后在**转发规则**页面创建经过验证的可编程脚本定义的转发规则，然后启用对应规则使之在生产环境生效。

可编程脚本规则生命周期管理

您可以在转发规则页面启用、禁用、编辑和删除目标可编程脚本：

- 打开**已禁用**开关，则可启用该可编程脚本。
- 关闭**已启动**开关，则可禁用该可编程脚本。
- 单击**编辑**，可修改**规则名称**、**规则代码**、**启用状态**和**高级配置**等信息。
- 单击**删除**，然后在**删除可编程脚本**对话框单击**确定**即可删除该可编程脚本。

相关文档

- 可编程脚本AScript概述

- 添加HTTP监听
- 添加HTTPS监听
- 添加QUIC监听

4.2.4. AScript语法

本文为您介绍AScript语法中注释、标识符、数据类型、变量、运算符、语句和函数的使用规则。
AScript语法使用规则，请参见下表：

语法	规则
注释	以#开头的当前行后续内容，均为注释。例如：`# this is annotation`。
标识符规则	• 由字母、数字、下划线组成，数字不能开头，区分大小写。 • 变量名（内置 、自定义）和函数名（内置 、自定义）均遵守标识符规则。
数据类型	• 字符串 字面常量：使用单引号括起来，例如：`'hello, AScript'`。 • 数字 字面常量：十进制数字，例如：10、-99、1.1。 • 布尔值 字面常量：true、false。 • 字典 字面常量如下： ○ []：空 ○ `['key1', 'key2', 100]`： ■ `1 -> 'key1'` ■ `2 -> 'key2'` ■ `3 -> 'key3'` ○ `['key1' = 'value1', 'key2' = 1000]` ■ `'key1' -> 'value1'` ■ `'key2' -> 1000`
变量	• 定义 赋值即定义。 • 使用 ○ 内置和自定义变量，均由变量名进行引用 。 ■ 引用内置变量：`host`。 ■ 引用自定义变量：`seckey`。 ○ 为强调变量的内置属性，可通过`$`进行引用。 引用内置变量：`$host`。 ○ 自定义变量的名称不能与内置变量同名。 内置变量，请参见EdgeScript内置变量表。

<table>
<tr><th>语法</th><th>规则</th></tr>
<tr><td>运算符</td><td>

= ：赋值运算符

例如：<code>seckey = 'ASDLFJ234dxvf34sDF'</code>
例如：<code>seckeys = ['key1', 'key2']</code>

- ：负号运算符
例如：<code>inum = -10</code>
对各数据类型的操作，不再另行支持运算符，均由内置函数支持，请参见条件判断相关。

各数据类型内置函数支持

字符串类型内置处理函数。
数字类型内置处理函数。
字典类型内置处理函数。

示例

<code>sval = concat(sval, 'trail')</code>
<code>len(arrvar)</code>

</td></tr>
<tr><td>语句</td><td>

条件判断语句
<pre>
if condition {
   ...
}
if condition1 {
   if conditon2 {
        ...
   }
}
if condition {
   ...
} else {
   ...
}
</pre>
语句解释

<code>condition</code> 可由如下语法元素组成：

字面值
变量
函数调用

body部分

允许空body。
允许多语句：一行一条语句。

支持多层嵌套
CodingStyle
语法强制要求左大扩号跟随在 <code>if condition</code> 之后，且同行。

</td></tr>
</table>

语法	规则
函数	• 定义语法 `def 函数名（参数列表） {` `...` `}` • 定义说明 ○ 形参列表 ■ 允许无参。 ■ 允许多参：由逗号分隔。 ○ 函数体部分 ■ 允许空body。 ■ 允许多语句：一行一条语句。 ■ 返回值：支持return语句。 ○ CodingStyle 语法强制要求左大扩号跟随在 `def` 函数名（参数列表）之后，且同行。 • 函数调用 无论内置、自定义函数，均通过 *函数名*`()` 进行调用。
其他	AScript全文不允许出现任何双引号。

4.2.5. AScript内置变量表

本文为您介绍AScript脚本中所有内置变量的含义和对应Nginx原生变量。

内置变量表

AScript内置变量如下表所示。

内置变量名	含义	对应Nginx原生变量
$arg_{name}	*Query String* 中的参数 `name` 值。`Query String` 表示HTTP请求中的请求参数。	ngx.var.arg_{name} **说明** `{name}` 中出现的连接号（-），需要使用下划线（_）替代，例如：`X-USER-ID` 对应为 `$arg_x_user_id` 。
$http_{name}	请求头中的 `name` 值。	ngx.var.http_{name} **说明** `{name}` 中出现的连接号（-），需要使用下划线（_）替代，例如：`X-USER-ID` 对应为 `$http_x_user_id` 。

内置变量名	含义	对应Nginx原生变量
$cookie_{name}	请求cookie头中的 `name` 值。	ngx.var.cookie_{name} **说明** `{name}` 中出现的连接号（-），需要使用下划线（_）替代，例如：`X-USER-ID` 对应为 `$cookie_x_user_id` 。
$scheme	协议类型。	ngx.var.scheme
$server_protocol	协议版本。	ngx.var.server_protocol
$host	原始host。	ngx.var.host
$uri	原始URI。	ngx.var.raw_uri
$args	`$args` 表示当前HTTP请求的全部请求参数，但不包含问号（`?`）。例如：`http://www.a.com/1k.file?k1=v1&k2=v2` 。 • `$arg_k1` 可以获得对应的 `v1` 值。 • `$args` 可以获得整个请求参数字符串，即 `k1=v1&k2=v2` ，不包括问号（`?`）。	ngx.var.args
$request_method	请求方法。	ngx.var.request_method
$request_uri	*uri*+'?'+args 的内容。	ngx.var.request_uri
$remote_addr	客户的IP地址。	ngx.var.remote_addr

说明

- 内置变量名前的美元符号（`$`）仅为强调其内置变量属性，删除后不影响使用。
- 内置变量不允许担当左值，即内置变量不允许被赋值。
- 每条AScript规则中最多支持使用200个全局变量，局部变量不限。如果全局变量超过200个请自定义函数，并在函数中以局部变量的形式使用全局变量。

4.2.6. AScript内置函数库

4.2.6.1. AScript函数概述

本文为您介绍AScript内置函数的分类和各个分类包含的函数。

函数分类	函数
条件判断相关函数	条件判断相关函数包括：and、or、not、eq、ne、null。
数字类型相关函数	数字类型相关函数包括：add、sub、mul、div、mod、gt、ge、lt、le、floor、ceil。
字符串类型相关函数	字符串类型相关函数包括：substr、concat、format、upper、lower、len、byte、match_re、capture_re、gsub_re、split、split_as_key、tohex、tobin、tostring、tochar。
字典类型相关函数	字典类型相关函数包括：set、get、foreach、del。
请求处理相关函数	请求处理相关函数包括：add_req_header、del_req_header、add_rsp_header、del_rsp_header、encode_args、decode_args、rewrite、say、print、exit。
时间相关函数	时间相关函数包括：today、time、now、localtime、utctime、cookie_time、http_time、parse_http_time、unixtime。
密码算法相关函数	密码算法相关函数包括：aes_new、aes_enc、aes_dec、sha1、sha2、hmac、hmac_sha1、md5、md5_bin。
JSON相关函数	JSON相关函数包括：json_enc、json_dec。
Misc相关函数	Misc相关函数包括：base64_enc、base64_dec、url_escape、url_unescape、randomseed、rand、rand_hit、crc、tonumber、base64_enc_safe、base64_dec_safe。
数组类型相关函数	数组类型相关函数包括：arr_concat、arr_insert、arr_remove、arr_sort、arr_len。
请求判断相关函数	请求判断相关函数包括：server_addr、server_port、client_addr、client_port、req_uri、req_uri_basename、req_uri_ext、req_uri_seg、req_uri_arg、req_uri_query_string、req_scheme、req_method、req_host、req_user_agent、req_referer、req_cookie、req_first_x_forwarded、req_header、req_id。

4.2.6.2. 条件判断相关函数

本文为您介绍条件判断相关函数的语法、说明、参数、返回值和示例。

and | or | not | eq | ne | null

and

项目	描述
语法	`and(arg, ...)` 。
说明	• 逻辑与运算符。 • 支持短路语义，即某个参数为假时，后续参数不再进行求值。
参数	一个或多个参数，参数类型不限。

项目	描述
返回值	全部参数为真时返回 `true` ，任一参数为假时返回 `false` 。
示例	`if and($arg_mode, eq($arg_mode, 'set_header')) { add_rsp_header('USER-DEFINED-1','path1') }` • 当请求携带 `mode` 参数且 `mode` 参数等于 `set_header` 时，设置响应头 `USER-DEFINED-1` 。 • 当请求不携带 `mode` 参数，短路语义生效，不再执行后续的 `eq` 比较；由于 `and()` 为假，不会设置响应头 `USER-DEFINED-1` 。

or

项目	描述
语法	`or(arg, ...)` 。
说明	• 逻辑或运算符。 • 支持短路语义，即某个参数为真时，后续参数不再进行求值。
参数	一个或多个参数，参数类型不限。
返回值	任一参数为真时返回 `true` ，全部参数为假时返回 `false` 。
示例	`if and($http_from, or(eq($http_from, 'wap'), eq($http_from, 'comos'))) { rewrite(concat('http://tech.com.cn/zt_d/we2015/', $http_from), 'enhance_redirect') }` • 当请求头 `from` 存在，且其值为 `[wap|comos]` 时，302重写向至 `http://tech.com.cn/zt_d/we2015/[wap|comos]` 。 • 当请求头 `from` 存在，且其值为 `wap` 时，短路语义生效，不再执行后续 `eq comos` 比较，同时 `or()` 返回 `true` 。

not

项目	描述
语法	`not(arg)` 。
说明	逻辑运算符取反。参数 `undef` 和 `false` 为假，其余为真。
参数	仅接受1个参数，参数类型不限。

项目	描述
返回值	• true • false
示例	`if not($arg_key) { exit(403) } if not($cookie_user) { exit(403, 'not cookie user') } if not(0) { exit(403) } if not(false) { exit(403) }` • 如果请求未携带参数 `key` 时，403拒绝请求。 • 当请求未携带 `cookie user` 时，403拒绝请求，响应body为 `'not cookie user'` 。 • `not(0)` 的结果为 `false` 。 • `not(false)` 的结果为 `true` 。

eq

项目	描述
语法	`eq(arg1, arg2)` 。
说明	比较2个参数是否相等。
参数	• arg1：任意类型。 • arg2：应与 `arg1` 类型相同。
返回值	参数相等返回 `true` ，否则返回 `false` 。

项目	描述
示例	``` key1 = 'value1' key2 = 'value2' if and($arg_k1, $arg_k2, eq(key1, $arg_k1), ne(key2, $arg_k2)) { say('match condition') } ``` - 请求参数 `k1` 和 `k2` 都存在时，执行后续的比较操作。 - 请求参数 `k1` 或 `k2` 不存在时，短路语义生效，不再执行后续的比较操作。 - `eq` ：请求参数 `k1` 的值是否等于 `value1` 。 - `ne` ：请求参数 `k2` 的值不等于 `value2` 。 - 当请求参数 `k1` 和 `k2` 均存在，且 `k1` 等于 `value1` ， `k2` 不等于 `value2` 时，输出响应体 `match condition` 。

ne

项目	描述
语法	`ne(arg1, arg2)` 。
说明	比较2个参数是否不等。
参数	- arg1：任意类型。 - arg2：应与 `arg1` 类型相同。
返回值	参数不等返回 `true` ，否则返回 `false` 。
示例	``` key1 = 'value1' key2 = 'value2' if and($arg_k1, $arg_k2, eq(key1, $arg_k1), ne(key2, $arg_k2)) { say('match condition') }key1 = 'value1' key2 = 'value2' if and($arg_k1, $arg_k2, eq(key1, $arg_k1), ne(key2, $arg_k2)) { say('match condition') } ``` - 请求参数 `k1` 和 `k2` 都存在时，执行后续的比较操作。 - 请求参数 `k1` 或 `k2` 不存在时，短路语义生效，不再执行后续的比较操作。 - `eq` ：请求参数 `k1` 的值是否等于 `value1` 。 - `ne` ：请求参数 `k2` 的值不等于 `value2` 。 - 当请求参数 `k1` 和 `k2` 均存在，且 `k1` 等于 `value1` ， `k2` 不等于 `value2` 时，输出响应体 `match condition` 。 -

null

项目	描述
语法	`null(v)` 。
说明	判断AScript数据类型是否为空。
参数	v：需要传入的参数，类型为数组、字典和字符串，其他类型均返回false。
返回值	返回值为bool类型。 • v是数组和字典，如果为空，返回true。 • v是字符串，如果值为空串，返回true。 • 其他情况均返回false。
示例	`d = []` `say(null(d))` `set(d, 1, 'v1')` `say(null(d))` `say(tostring(null('x')))` `say(tostring(null('')))` 输出： `true` `false` `false` `true`

4.2.6.3. 数字类型相关函数

本文为您介绍数字类型相关函数的语法、说明、参数、返回值和示例。

add | sub | mul | div | mod | gt | ge | lt | le | floor | ceil

add

项目	描述
语法	`add(n1, n2)` 。
说明	加法操作。
参数	• n1：数字类型。 • n2：数字类型。
返回值	返回 `n1+n2` 的结果。

项目	描述
示例	``` n1 = add(10, 20) n2 = sub(10, 20) n3 = mul(10, 20) n4 = div(10, 20) n5 = mod(35, 20) say(concat('n1=', n1)) say(concat('n2=', n2)) say(concat('n3=', n3)) say(concat('n4=', n4)) say(concat('n5=', n5)) ``` 输出： ``` n1=30 n2=-10 n3=200 n4=0.5 n5=15 ```

sub

项目	描述
语法	`sub(n1, n2)` 。
说明	减法操作。
参数	• n1：数字类型。 • n2：数字类型。
返回值	返回 `n1-n2` 的结果。

项目	描述
示例	``` n1 = add(10, 20) n2 = sub(10, 20) n3 = mul(10, 20) n4 = div(10, 20) n5 = mod(35, 20) say(concat('n1=', n1)) say(concat('n2=', n2)) say(concat('n3=', n3)) say(concat('n4=', n4)) say(concat('n5=', n5))n1 = add(10, 20) n2 = sub(10, 20) n3 = mul(10, 20) n4 = div(10, 20) n5 = mod(35, 20) say(concat('n1=', n1)) say(concat('n2=', n2)) say(concat('n3=', n3)) say(concat('n4=', n4)) say(concat('n5=', n5)) ``` 输出： ``` n1=30 n2=-10 n3=200 n4=0.5 n5=15 ```

mul

项目	描述
语法	`mul(n1, n2)` 。
说明	乘法操作。
参数	• n1：数字类型。 • n2：数字类型。
返回值	返回 `n1×n2` 的结果。

项目	描述
示例	``` n1 = add(10, 20) n2 = sub(10, 20) n3 = mul(10, 20) n4 = div(10, 20) n5 = mod(35, 20) say(concat('n1=', n1)) say(concat('n2=', n2)) say(concat('n3=', n3)) say(concat('n4=', n4)) say(concat('n5=', n5))n1 = add(10, 20) n2 = sub(10, 20) n3 = mul(10, 20) n4 = div(10, 20) n5 = mod(35, 20) say(concat('n1=', n1)) say(concat('n2=', n2)) say(concat('n3=', n3)) say(concat('n4=', n4)) say(concat('n5=', n5)) ``` 输出： ``` n1=30 n2=-10 n3=200 n4=0.5 n5=15 ```

div

项目	描述
语法	`div(n1, n2)` 。
说明	除法操作。

项目	描述
参数	• n1：数字类型。 • n2：数字类型。
返回值	返回 `n1÷n2` 的结果。
示例	`n1 = add(10, 20)` `n2 = sub(10, 20)` `n3 = mul(10, 20)` `n4 = div(10, 20)` `n5 = mod(35, 20)` `say(concat('n1=', n1))` `say(concat('n2=', n2))` `say(concat('n3=', n3))` `say(concat('n4=', n4))` `say(concat('n5=', n5))n1 = add(10, 20)` `n2 = sub(10, 20)` `n3 = mul(10, 20)` `n4 = div(10, 20)` `n5 = mod(35, 20)` `say(concat('n1=', n1))` `say(concat('n2=', n2))` `say(concat('n3=', n3))` `say(concat('n4=', n4))` `say(concat('n5=', n5))` 输出： `n1=30` `n2=-10` `n3=200` `n4=0.5` `n5=15`

mod

项目	描述
语法	`mod(n1, n2)` 。
说明	求余操作。

项目	描述
参数	• n1：数字类型。 • n2：数字类型。
返回值	返回 `n1%n2` 的结果（余数）。
示例	``` n1 = add(10, 20) n2 = sub(10, 20) n3 = mul(10, 20) n4 = div(10, 20) n5 = mod(35, 20) say(concat('n1=', n1)) say(concat('n2=', n2)) say(concat('n3=', n3)) say(concat('n4=', n4)) say(concat('n5=', n5))n1 = add(10, 20) n2 = sub(10, 20) n3 = mul(10, 20) n4 = div(10, 20) n5 = mod(35, 20) say(concat('n1=', n1)) say(concat('n2=', n2)) say(concat('n3=', n3)) say(concat('n4=', n4)) say(concat('n5=', n5)) ``` 输出： ``` n1=30 n2=-10 n3=200 n4=0.5 n5=15 ```

gt

项目	描述
语法	`gt(n1, n2)` 。
说明	大于比较。
参数	• n1：数字类型。 • n2：数字类型。

项目	描述
返回值	若 `n1>n2` ，返回 `true` ，否则返回 `false` 。
示例	``` if and($arg_num, gt(tonumber($arg_num), 10)) { say('num > 10') } if and($arg_num, ge(tonumber($arg_num), 10)) { say('num >= 10') } if and($arg_num, lt(tonumber($arg_num), 10)) { say('num < 10') } if and($arg_num, le(tonumber($arg_num), 10)) { say('num <= 10') } ``` • 请求： `/path1/path2/file?num=10` ，响应： `num <= 10 num >= 10` 。 • 请求： `/path1/path2/file?num=11` ，响应： `num > 10 num >= 10` 。 • 请求： `/path1/path2/file?num=9` ，响应： `num < 10 num <= 10` 。

ge

项目	描述
语法	`ge(n1, n2)` 。
说明	大于等于比较。
参数	• n1：数字类型。 • n2：数字类型。
返回值	若 `n1>=n2` ，返回 `true` ，否则返回 `false` 。

项目	描述
示例	``` if and($arg_num, gt(tonumber($arg_num), 10)) { say('num > 10') } if and($arg_num, ge(tonumber($arg_num), 10)) { say('num >= 10') } if and($arg_num, lt(tonumber($arg_num), 10)) { say('num < 10') } if and($arg_num, le(tonumber($arg_num), 10)) { say('num <= 10') }if and($arg_num, gt(tonumber($arg_num), 10)) { say('num > 10') } if and($arg_num, ge(tonumber($arg_num), 10)) { say('num >= 10') } if and($arg_num, lt(tonumber($arg_num), 10)) { say('num < 10') } if and($arg_num, le(tonumber($arg_num), 10)) { say('num <= 10') } ``` • 请求：`/path1/path2/file?num=10`，响应：`num <= 10 num >= 10`。 • 请求：`/path1/path2/file?num=11`，响应：`num > 10 num >= 10`。 • 请求：`/path1/path2/file?num=9`，响应：`num < 10 num <= 10`。 •

lt

项目	描述
语法	`lt(n1, n2)`。
说明	小于比较。
参数	• n1：数字类型。 • n2：数字类型。

项目	描述
返回值	若 `n1<n2` ，返回 `true` ，否则返回 `false` 。
示例	`if and($arg_num, gt(tonumber($arg_num), 10)) {` `  say('num > 10')` `}` `if and($arg_num, ge(tonumber($arg_num), 10)) {` `  say('num >= 10')` `}` `if and($arg_num, lt(tonumber($arg_num), 10)) {` `  say('num < 10')` `}` `if and($arg_num, le(tonumber($arg_num), 10)) {` `  say('num <= 10')` `}if and($arg_num, gt(tonumber($arg_num), 10)) {` `  say('num > 10')` `}` `if and($arg_num, ge(tonumber($arg_num), 10)) {` `  say('num >= 10')` `}` `if and($arg_num, lt(tonumber($arg_num), 10)) {` `  say('num < 10')` `}` `if and($arg_num, le(tonumber($arg_num), 10)) {` `  say('num <= 10')` `}` • 请求： `/path1/path2/file?num=10` ，响应： `num <= 10 num >= 10` 。 • 请求： `/path1/path2/file?num=11` ，响应： `num > 10 num >= 10` 。 • 请求： `/path1/path2/file?num=9` ，响应： `num < 10 num <= 10` 。 •

le

项目	描述
语法	`le(n1, n2)` 。
说明	小于等于比较。
参数	• n1：数字类型。 • n2：数字类型。
返回值	若 `n1<=n2` ，返回 `true` ，否则返回 `false` 。

项目	描述
示例	`if and($arg_num, gt(tonumber($arg_num), 10)) {` `  say('num > 10')` `}` `if and($arg_num, ge(tonumber($arg_num), 10)) {` `  say('num >= 10')` `}` `if and($arg_num, lt(tonumber($arg_num), 10)) {` `  say('num < 10')` `}` `if and($arg_num, le(tonumber($arg_num), 10)) {` `  say('num <= 10')` `}if and($arg_num, gt(tonumber($arg_num), 10)) {` `  say('num > 10')` `}` `if and($arg_num, ge(tonumber($arg_num), 10)) {` `  say('num >= 10')` `}` `if and($arg_num, lt(tonumber($arg_num), 10)) {` `  say('num < 10')` `}` `if and($arg_num, le(tonumber($arg_num), 10)) {` `  say('num <= 10')` `}` • 请求：`/path1/path2/file?num=10`，响应：`num <= 10 num >= 10`。 • 请求：`/path1/path2/file?num=11`，响应：`num > 10 num >= 10`。 • 请求：`/path1/path2/file?num=9`，响应：`num < 10 num <= 10`。 •

floor

项目	描述
语法	`floor(n)`。
说明	向下取整。
参数	n：数字类型。
返回值	返回`n`的向下取整。
示例	`if $arg_num {` `  say(concat('ceil: ', ceil(tonumber($arg_num))))` `  say(concat('floor: ', floor(tonumber($arg_num))))` `}` 请求：`/path1/path2/file?num=9.3`，响应：`ceil: 10 floor: 9`。

ceil

项目	描述
语法	`ceil(n)` 。
说明	向上取整。
参数	n：数字类型。
返回值	返回 `n` 的向上取整。
示例	`if $arg_num {` `  say(concat('ceil: ', ceil(tonumber($arg_num))))` `  say(concat('floor: ', floor(tonumber($arg_num))))` `}if $arg_num {` `  say(concat('ceil: ', ceil(tonumber($arg_num))))` `  say(concat('floor: ', floor(tonumber($arg_num))))` `}` 请求：`/path1/path2/file?num=9.3`，响应：`ceil: 10 floor: 9` 。

4.2.6.4. 字符串类型相关函数

本文为您介绍字符串类型相关函数的语法、说明、参数、返回值和示例。

substr | concat | format | upper | lower | len | byte | match_re | capture_re | gsub_re | split | split_as_key | tohex | tobin | tostring | tochar | reverse | find | trim

substr

项目	描述
语法	`substr(s, i, j)` 。
说明	字符串截取操作。
参数	• s：目标字符串。 • i：整型，截取起始下标。从1开始，-1表示字符串最尾字符。 • j：整型，截取终止下标。从1开始，-1表示字符串最尾字符。
返回值	返回 `s` 的子字符串 `s[i, j]` 。

项目	描述
示例	判断文件类型是否为.m3u8的两种方法： • 方法一： `if eq(substr($uri, -5, -1), '.m3u8') {` `  say(concat($uri, ' is .m3u8'))` `}` • 方法二： `uri_len = len($uri)` `if eq(substr($uri, -5, uri_len), '.m3u8') {` `  say(concat($uri, ' is .m3u8'))` `}`

concat

项目	描述
语法	`concat(s1, ...)` 。
说明	字符串连接操作。
参数	一个或多个参数，参数类型允许为数字字符串。
返回值	将多个参数连接为一个字符串，并返回该字符串。
示例	判断文件类型是否为.m3u8的两种方法： • 方法一： `if eq(substr($uri, -5, -1), '.m3u8') {` `  say(concat($uri, ' is .m3u8'))` `}` • 方法二： `uri_len = len($uri)` `if eq(substr($uri, -5, uri_len), '.m3u8') {` `  say(concat($uri, ' is .m3u8'))` `}` •

format

项目	描述
语法	`format(fmt, ···)` 。

项目	描述
说明	返回不定数量参数的格式化版本，格式化字符串为第一个参数（必须是一个字符串）。格式化字符串遵循 `ISO C` 函数 `sprintf` 的规则。 `fmt` 规则的格式为：`%[指定参数][标识符][宽度][精度]指示符` 。 `%%` 印出百分比符号，不转换。 `%c` 整数转成对应的ASCII字元。 `%d` 整数转成十进位。 `%f` 倍精确度数字转成浮点数。 `%o` 整数转成八进位。 `%s` 整数转成字符串。 `%x` 整数转成小写十六进位。 `%X` 整数转成大写十六进位。
参数	• fmt：String类型，格式化字符串。 • 可变参数：任意类型。
返回值	String类型。
示例	`say(concat('format:', format('%%%s$%.2s$%s$%c$%d$%2.2f$%.2o$%x$%X', 'format', 3.1415926, true, 95, 3.1415926, 3.1415926, 3.1415926, 10, 10)))` 输出： `format:%format$3.$true$_$3$3.14$03$a$A`

upper

项目	描述
语法	`upper(s)` 。
说明	将字符串中所有的小写字母转换成大写字母。
参数	s：目标字符串。
返回值	返回大写 `s` 。

项目	描述
示例	``` mystr = 'Hello, AScript' say(upper(mystr)) say(lower(mystr)) ``` 输出： ``` HELLO, AScript hello, ascript ```

lower

项目	描述
语法	`lower(s)` 。
说明	将字符串中所有的大写字母转换成小写字母。
参数	s：目标字符串。
返回值	返回小写 `s` 。
示例	``` mystr = 'Hello, AScript' say(upper(mystr)) say(lower(mystr))mystr = 'Hello, AScript' say(upper(mystr)) say(lower(mystr)) ``` 输出： ``` HELLO, AScript hello, ascriptHELLO, AScript hello, ascript ```

len

项目	描述
语法	`len(s)` 。
说明	获取字符串的长度。
参数	s：目标字符串。
返回值	返回 `s` 的长度，整型。

项目	描述
示例	判断文件类型是否为.m3u8的两种方法： • 方法一： `if eq(substr($uri, -5, -1), '.m3u8') {` `  say(concat($uri, ' is .m3u8'))` `}` • 方法二： `uri_len = len($uri)` `if eq(substr($uri, -5, uri_len), '.m3u8') {` `  say(concat($uri, ' is .m3u8'))` `}` •

byte

项目	描述
语法	`byte(c)` 。
说明	获取字符的ASCII码。
参数	c：目标字符，必须为单个字符。
返回值	返回对应的ASCII码，数字类型。
示例	`say(byte('a'))` `say(byte('A'))` 输出： `97` `65`

match_re

项目	描述
语法	`match_re(s, p [, o])` 。
说明	使用PCRE正则引擎，进行正则匹配，判断字符串是否匹配对应的正则表达式。更多信息，请参见PCRE正则语法。

项目	描述
参数	• s：目标字符串，字符类型。 • p：正则表达式，字符类型。 • o：正则引擎参数，字符类型，可选填。
返回值	匹配成功返回 `true` ，否则返回 `false` 。
示例	`url = concat('http://', $host, $uri)` `m1 = match_re(url, 'http://.*\.dslex\.com/.*')` `m2 = match_re(url,` `'^http://.*\.alibaba\.com\.cn/.*\.d\\.html(\?.*)?$')` `m3 = match_re(url, '^http://.*.test.dslex.com/.*\.d\.html(\?.*)?` `$')` `m4 = match_re(url, '^http://.*\.alibaba\.com\.cn/zt_d/')` `m5 = match_re(url, '^http://tech.alibaba.com.cn/zt_d/we2015/?$')` `m6 = match_re($args, 'from=wap1$')` `m7 = match_re($args, 'from=comos1$')` `if and(m1, or(m2, m3), not(m4), not(m5), or(not(m6), not(m7))) {` `add_rsp_header('USER-DEFINED-1', 'hit1')` `add_rsp_header('USER-DEFINED-2', 'hit2')` `}`

capture_re

项目	描述
语法	`capture_re(s, p [,init])` 。
说明	正则捕获，并返回捕获结果。使用PCRE正则引擎，更多信息，请参见PCRE正则语法。
参数	• s：目标字符串，字符类型。 • p：正则表达式，字符类型。 • init：指定匹配开始位置，下标从1开始，整型。
返回值	匹配成功的若干子串通过字典类型返回，匹配失败返回空字典。

项目	描述
示例	`pcs = capture_re($request_uri,'^/([^/]+)/([^/]+)([^?]+)\?(.*)')` `sec1 = get(pcs, 1)` `sec2 = get(pcs, 2)` `sec3 = get(pcs, 3)` `if or(not(sec1), not(sec2), not(sec3)) {` `add_rsp_header_imm('X-TENGINE-ERROR', 'auth failed - missing necessary uri set')` `exit(403)` `}` `digest = md5(concat(sec1, sec3))` `if ne(digest, sec2) {` `add_rsp_header_imm('X-TENGINE-ERROR', 'auth failed - invalid digest')` `exit(403)` `}`

gsub_re

项目	描述
语法	`gsub_re(subject, regex, replace [,option])` 。
说明	正则替换，并返回替换后的副本。使用PCRE正则引擎，详细信息，请参见PCRE正则语法。
参数	• subject：目标字符串，字符类型。 • regex：正则表达式，字符类型。 • replace：替换字符，字符类型。`replace` 部分可以引用匹配部分，即： ○ $0：表示 `regex` 整体匹配的部分。 ○ $N：表示 `regex` 第N个 `()` 匹配的部分。 • option：正则引擎参数，字符类型，可选。
返回值	`subject` 中所有的符合参数 `regex` 的子串都将被参数 `replace` 所指定的字符串所替换，并返回替换后的副本。

项目	描述
示例	`subject = 'Hello, Es'` `regex = '([a-zA-Z])[a-z]+'` `replace = '[$0,$1]'` `add_rsp_header('X-DEBUG-GSUB-RE', gsub_re(subject, regex, replace))` 输出： `X-DEBUG-GSUB-RE: [Hello,H], [Es,E]`

split

项目	描述
语法	`split(s [,sep])` 。
说明	分隔字符串，并返回分隔结果。
参数	• s：目标字符串，字符类型。 • sep：字符类型。
返回值	分隔元素包含在字典类型中返回，由数字下标作 `key` ，起始下标为1，例如：[1]=xx，[2]=yy；若 `sep` 为空，则默认以任意空白字符分隔。默认空白字符包含：空格、Tab。

项目	描述
示例	`if $arg_from {` `  t = split($arg_from, ',')` `  if get(t, 1) {` `  say(concat('[1]=', get(t, 1)))` `  }` `  if get(t, 2) {` `  say(concat('[2]=', get(t, 1)))` `  }` `}` 请求： `?from=xx1,xx2,xx3` 响应： `[1]=xx1` `[2]=xx1`

split_as_key

项目	描述
语法	`split_as_key(s [,sep])` 。
说明	分隔字符串，并返回分隔结果。
参数	• s：目标字符串，字符类型。 • sep：分隔符，字符类型。
返回值	同 `split()` ，区别在于 `key` ： *[分割元素]* -> *[分割元素]* 。

项目	描述
示例	`def echo_each(k, v, u) { s = concat(k, '=', v, ' u=', get(u, 1)) say(s) } if $arg_from { t = split_as_key($arg_from, ',') foreach(t, echo_each, ['hi,ascript']) }` 请求： `?from=xx1,xx2,xx3` 响应： `xx2=xx2 u=hi,ascript` `xx1=xx1 u=hi,ascript` `xx3=xx3 u=hi,ascript`

tohex

项目	描述
语法	`tohex(s)` 。
说明	十六进制转换。
参数	s：字符串。
返回值	返回 `s` 的十六进制可读形式。
示例	`digest = sha1('xxxx')` `add_rsp_header('X-AScript-TOHEX', tohex(digest))` 输出： `X-AScript-TOHEX:4ad583af22c2e7d40c1c916b2920299155a46464`

tobin

项目	描述
语法	`tobin(str)` 。
说明	16进制转ASCII字符串。

项目	描述
参数	str: 双字节16进制字符串，不区分大小写。
返回值	String类型。
示例	`say(concat('tobin:', tobin('2F2F')))` 输出： `tobin://`

tostring

项目	描述
语法	`tostring(a)` 。
说明	字符串类型转换。
参数	a：任意类型。
返回值	返回参数 `a` 转换后的字符串。
示例	`s = tostring(123)` `add_rsp_header('X-DSL-TOSTRING', s)` 输出： `X-DSL-TOSTRING: 123`

tochar

项目	描述
语法	`tochar(n1, n2, ...)` 。
说明	• 接受1个或多个整型参数，返回对应整型参数的内部数值表示的字符串，例如：48对应于字符“0”。 • 返回字符串的长度为参数个数。
参数	nX：整型参数。
返回值	返回转换后的字符串。

项目	描述
示例	`add_rsp_header('X-DSL-TOCHAR', tochar(97))` `add_rsp_header('X-DSL-TOCHAR', tochar(97, 98), true)` `//输出：增加响应头` `//X-DSL-TOCHAR: a` `//X-DSL-TOCHAR: ab` `if $arg_filename {` `hn = 'Content-Disposition'` `add_rsp_header('Content-Disposition',` `concat('attachment;filename=', tochar(34), filename, tochar(34)))` `add_rsp_header(hn, hv)` `}` 输出：增加响应头。 `Content-Disposition: attachment;filename="请求参数filename的值"`

reverse

项目	描述
语法	`reverse(str)` 。
说明	字符串反转。
参数	str：待反转的字符串。
返回值	返回字符类型，返回反转后的字符串。
示例	`say(reverse('hello'))` 输出： `#olleh`

find

项目	描述
语法	`string.find (s, substr, pos)` 。
说明	在目标字符串中搜索指定的字符串。

项目	描述
参数	• s：待查找的字符串。 • substr：需要查找的子串。 • pos（可选参数）：该参数为索引，类型为数值，指定查找的起始位置。可以为负数，默认起始位置为1。
返回值	• 成功：返回数组类型。 ◦ 索引1，返回查找的起始位置。 ◦ 索引2，返回查找的截止位置。 • 失败：返回空数组。
示例	`str = 'hello dsl'` `add_rsp_header('string-find()-start', tostring(get(find(str, 'dsl'), 1)))` `str = 'hello dsl 12'` `add_rsp_header('string-find()-end', tostring(get(find(str, 'dsl'), 2)))` `str = 'hello dsl'` `add_rsp_header('string-find()-tail-start', tostring(get(find(str, 'dsl', -6), 1)))` `str = 'hello dsl 12'` `add_rsp_header('string-find()-tail-end', tostring(get(find(str, 'dsl', -6), 2)))` 输出： `string-find()-start:7` `string-find()-end:9` `string-find()-tail-start:7` `string-find()-tail-end:9`

trim

项目	描述
语法	`trim(s, [, loc])` 。
说明	剔除s两端或指定端的全部空白字符，并返回剔除后的字符串。
参数	• s：目标字符串。 • loc（可选参数）：默认为both，可用值如下： ◦ both：剔除两端。 ◦ left：仅剔除左端。 ◦ right：仅剔除右端。

项目	描述
返回值	返回剔除后的字符串。
示例	``` say(concat('trim():', trim(' abcd '))) say(concat('trim(left):', trim(' abcd ', 'left'))) say(concat('trim(right):', trim(' abcd ', 'right'))) ``` 输出： ``` trim():abcd trim(left):abcd trim(right): abcd ```

4.2.6.5. 字典类型相关函数

本文为您介绍字典类型相关函数的语法、参数、示例和返回值。

set | get | foreach | del

set

项目	描述
语法	`set(d, k, v)`
说明	在字典d中设置k和v。
参数	• d：目标字典。 • k：key值，可以为任意类型。 • v：value值，可以为任意类型。
返回值	永远返回 `true` 。

项目	描述
示例	• 示例1 `outer_keys=['e66fd4aa-f281-472f-b919-fc7e7474de25', '66fee78d-1887-42ec-9119-a9b50b7fbca2']` `say(concat('keys[1]=', get(outer_keys, 1)))` `say(concat('keys[2]=', get(outer_keys, 2)))` `inner_keys=[]` `set(inner_keys, 'dev', '243390eb-00b7-4551-a6b8-021bb34d1674')` `set(inner_keys, 'zeus', '4747d33b-12b0-45e6-ac10-a8e191d6adaa')` `def echo_each(k, v, u) {` `  s = concat('keys[', k, ']=', v)` `  say(s)` `}` `foreach(inner_keys, echo_each, [])` 输出： `keys[1]=e66fd4aa-f281-472f-b919-fc7e7474de25` `keys[2]=66fee78d-1887-42ec-9119-a9b50b7fbca2` `keys[dev]=243390eb-00b7-4551-a6b8-021bb34d1674` `keys[zeus]=4747d33b-12b0-45e6-ac10-a8e191d6adaa` • 示例2 `d_inner = []` `set(d_inner, 'name', 'inner ascript')` `d_outer = []` `set(d_outer, 'dictA', d_inner)` `v = get(d_outer, 'dictA')` `if v {` `  v = get(v, 'name')` `  if v {` `  add_rsp_header('X-AScript-NESTED-DICT', v)` `  }` `}` 输出： `X-AScript-NESTED-DICT: inner ascript`

get

项目	描述
语法	`get(d, k)`
说明	获取字典d中k对应的v。

项目	描述
参数	• d：目标字典。 • k：key值，可以为任意类型。
返回值	成功返回对应值，失败返回 `false` 。
示例	• 示例1 `outer_keys=['e66fd4aa-f281-472f-b919-fc7e7474de25', '66fee78d-1887-42ec-9119-a9b50b7fbca2']` `say(concat('keys[1]=', get(outer_keys, 1)))` `say(concat('keys[2]=', get(outer_keys, 2)))` `inner_keys=[]` `set(inner_keys, 'dev', '243390eb-00b7-4551-a6b8-021bb34d1674')` `set(inner_keys, 'zeus', '4747d33b-12b0-45e6-ac10-a8e191d6adaa')` `def echo_each(k, v, u) {` `s = concat('keys[', k, ']=', v)` `say(s)` `}` `foreach(inner_keys, echo_each, [])` 输出： `keys[1]=e66fd4aa-f281-472f-b919-fc7e7474de25` `keys[2]=66fee78d-1887-42ec-9119-a9b50b7fbca2` `keys[dev]=243390eb-00b7-4551-a6b8-021bb34d1674` `keys[zeus]=4747d33b-12b0-45e6-ac10-a8e191d6adaa` • 示例2 `d_inner = []` `set(d_inner, 'name', 'inner ascript')` `d_outer = []` `set(d_outer, 'dictA', d_inner)` `v = get(d_outer, 'dictA')` `if v {` `v = get(v, 'name')` `if v {` `add_rsp_header('X-AScript-NESTED-DICT', v)` `}` `}` 输出： `X-AScript-NESTED-DICT: inner ascript` •

foreach

项目	描述
语法	`foreach(d, f, user_data)`
说明	• 遍历字典d中的元素，依次回调函数f。 • f原型要求为 `f(key, value, user_data)` 。 • 当 `f()` 返回false时， `foreach()` 循环终止。
参数	• d：目标字典。 • f：回调函数。 • user_data：用于传递用户数据，为字典类型。
返回值	永远返回 `true` 。
	• 示例1 `outer_keys=['e66fd4aa-f281-472f-b919-fc7e7474de25', '66fee78d-1887-42ec-9119-a9b50b7fbca2']` `say(concat('keys[1]=', get(outer_keys, 1)))` `say(concat('keys[2]=', get(outer_keys, 2)))` `inner_keys=[]` `set(inner_keys, 'dev', '243390eb-00b7-4551-a6b8-021bb34d1674')` `set(inner_keys, 'zeus', '4747d33b-12b0-45e6-ac10-a8e191d6adaa')` `def echo_each(k, v, u) {` `  s = concat('keys[', k, ']=', v)` `  say(s)` `}` `foreach(inner_keys, echo_each, [])` 输出： `keys[1]=e66fd4aa-f281-472f-b919-fc7e7474de25` `keys[2]=66fee78d-1887-42ec-9119-a9b50b7fbca2` `keys[dev]=243390eb-00b7-4551-a6b8-021bb34d1674` `keys[zeus]=4747d33b-12b0-45e6-ac10-a8e191d6adaa` • 示例2：输出 `m3u8` 的前2个分片，演示终止foreach循环。

项目	描述
示例	见下方

```
def echo_each(k, v, u) {
    say(v)
    if match(v, '.*ts') {
        ts_cnt = get(u, 'ts_cnt')
        ts_cnt = add(ts_cnt, 1)
        set(u, 'ts_cnt', ts_cnt)
        if ge(ts_cnt, 2) {
            return false
        }
    }
}
m3u8 = ''
m3u8 = concat(m3u8, '#EXTM3U8', '\n')
m3u8 = concat(m3u8, '#EXT-X-MEDIA-SEQUENCE:140651513\n')
m3u8 = concat(m3u8, '#EXT-X-TARGETDURATION:10\n')
m3u8 = concat(m3u8, '#EXTINF:8,\n')
m3u8 = concat(m3u8,
'http://vapp1.fw.live.cntv.cn/cache/289_/seg0/index140651514_1406
51513.ts\n')
m3u8 = concat(m3u8, '#EXTINF:9,\n')
m3u8 = concat(m3u8,
'http://vapp1.fw.live.cntv.cn/cache/289_/seg0/index140651514_1406
51514.ts\n')
m3u8 = concat(m3u8, '#EXTINF:10,\n')
m3u8 = concat(m3u8,
'http://vapp1.fw.live.cntv.cn/cache/289_/seg0/index140651514_1406
51515.ts\n')
lines = split(m3u8, '\n')
u = []
set(u, 'ts_cnt', 0)
foreach(lines, echo_each, u)
```

输出：

```
#EXTM3U8
#EXT-X-MEDIA-SEQUENCE:140651513
#EXT-X-TARGETDURATION:10
#EXTINF:8,
http://vapp1.fw.live.cntv.cn/cache/289_/seg0/index140651514_14065
1513.ts
#EXTINF:9,
http://vapp1.fw.live.cntv.cn/cache/289_/seg0/index140651514_14065
1514.ts
```

项目	描述

del

项目	描述
语法	del(d, k)
说明	删除字典d中的k/v对。
参数	• d：目标字典。 • k：key值，可以为任意类型。
示例	`var_a = []` `set(var_a, 'note_a', 'note a info')` `if get(var_a, 'note_a') {` `add_rsp_header('X-RESPOND-OUTPUT', 'found var_a key', true)` `} else {` `add_rsp_header('X-RESPOND-OUTPUT', 'del var_a key', true)` `}` `del(var_a, 'note_a')` `if get(var_a, 'note_a') {` `add_rsp_header('X-RESPOND-OUTPUT', 'found var_a key', true)` `} else {` `add_rsp_header('X-RESPOND-OUTPUT', 'del var_a key', true)` `}`
返回值	始终返回 `true` 。本示例的返回值如下： `X-RESPOND-OUTPUT: found var_a key` `X-RESPOND-OUTPUT: del var_a key`

4.2.6.6. 请求处理相关函数

本文为您介绍请求处理相关函数的语法、说明、参数、返回值和示例。

add_req_header | del_req_header | add_rsp_header | del_rsp_header | encode_args | decode_args | rewrite | say | print | exit

add_req_header

项目	描述
语法	`add_req_header(name, value [, append])` 。

项目	描述
说明	添加请求头，即回源请求头。
参数	• name：待添加的请求头 `name` ，字符类型。 • value：待添加的请求头 `value` ，字符类型。 • append：若请求头已添加， `append` 决定是否追加 `value` ，默认覆盖（即默认不追加），布尔类型。
返回值	默认返回 `true` ，无效请求头返回 `false` 。
示例	`add_rsp_header('USER-DEFINED-RSP-1', '1')` `add_rsp_header('USER-DEFINED-RSP-1', 'x', true)` `add_rsp_header('USER-DEFINED-RSP-2', '2')` `del_rsp_header('USER-DEFINED-RSP-2')` 输出2个响应头： `USER-DEFINED-RSP-1: 1` `USER-DEFINED-RSP-1: x` **说明** `USER-DEFINED-RSP-2` 先添加、后删除，故响应头中无 `USER-DEFINED-RSP-2` 。

del_req_header

项目	描述
语法	`del_req_header(name)` 。
说明	删除请求头，即回源请求头。
参数	name：待删除的请求头 `name` ，字符类型。
返回值	默认返回 `true` ，无效请求头返回 `false` 。

项目	描述
示例	`add_rsp_header('USER-DEFINED-RSP-1', '1')` `add_rsp_header('USER-DEFINED-RSP-1', 'x', true)` `add_rsp_header('USER-DEFINED-RSP-2', '2')` `del_rsp_header('USER-DEFINED-RSP-2')` 输出2个响应头： `USER-DEFINED-RSP-1: 1` `USER-DEFINED-RSP-1: x` **说明** `USER-DEFINED-RSP-2` 先添加、后删除，故响应头中无 `USER-DEFINED-RSP-2` 。

add_rsp_header

项目	描述
语法	`add_rsp_header(name, value [, append])` 。
说明	添加响应头。
参数	• name：待添加的响应头 `name` ，字符类型。 • value：待添加的响应头 `value` ，字符类型。 `value` 可包含如下标记，用于在响应阶段执行动态替换： ○ ${x}：将替换为 `ngx.var.x` 的值。 ○ @{y}：将替换为 *响应头*`y` 的值。 • append：若响应头已添加， `append` 决定是否追加 `value` ，默认覆盖，布尔类型。 。
返回值	默认返回 `true` ，无效响应头返回 `false` 。

项目	描述
示例	``` add_rsp_header('USER-DEFINED-RSP-1', '1') add_rsp_header('USER-DEFINED-RSP-1', 'x', true) add_rsp_header('USER-DEFINED-RSP-2', '2') del_rsp_header('USER-DEFINED-RSP-2') ``` 输出2个响应头： ``` USER-DEFINED-RSP-1: 1 USER-DEFINED-RSP-1: x ``` **说明** `USER-DEFINED-RSP-2` 先添加、后删除，故响应头中无 `USER-DEFINED-RSP-2` 。

del_rsp_header

项目	描述
语法	`del_rsp_header(name)` 。
说明	删除响应头。
参数	name：待删除的响应头 `name` ，字符类型。
返回值	默认返回 `true` ，无效响应头返回 `false` 。
示例	``` add_rsp_header('USER-DEFINED-RSP-1', '1') add_rsp_header('USER-DEFINED-RSP-1', 'x', true) add_rsp_header('USER-DEFINED-RSP-2', '2') del_rsp_header('USER-DEFINED-RSP-2') ``` 输出2个响应头： ``` USER-DEFINED-RSP-1: 1 USER-DEFINED-RSP-1: x ``` **说明** `USER-DEFINED-RSP-2` 先添加、后删除，故响应头中无 `USER-DEFINED-RSP-2` 。

encode_args

项目	描述
语法	`encode_args(d)` 。
说明	将字典 `d` 中的 `k/v` ，转换为URI编码的 `k1=v1&k2=v2` 格式的字符串。
参数	d：字典类型。
返回值	返回URI编码格式的字符串。
示例	`my_args = []` `set(my_args, 'signature', 'da9dc4b7-87ae-4330-aaaf-e5454e2c2af1')` `set(my_args, 'algo', 'private sign1')` `my_args_str = encode_args(my_args)` `add_rsp_header('X-AScript-ENCODE-ARGS', my_args_str)` `to_args = decode_args(my_args_str)` `if get(to_args, 'algo') {` `add_rsp_header('X-AScript-DECODE-ARGS-ALGO', get(to_args, 'algo'))` `}` `if get(to_args, 'signature') {` `add_rsp_header('X-AScript-DECODE-ARGS-SIGN', get(to_args, 'signature'))` `}` 输出3个响应头： `X-AScript-ENCODE-ARGS: signature=da9dc4b7-87ae-4330-aaaf-e5454e2c2af1&algo=private%20sign1` `X-AScript-DECODE-ARGS-ALGO: private sign1` `X-AScript-DECODE-ARGS-SIGN: da9dc4b7-87ae-4330-aaaf-e5454e2c2af1`

decode_args

项目	描述
语法	`decode_args(s)` 。
说明	将URI编码的 `k1=v1&k2=v2` 格式的字符串，转换为字典类型。
参数	s：目标字符串。
返回值	返回转换后的字典对象。

项目	描述
示例	``` my_args = [] set(my_args, 'signature', 'da9dc4b7-87ae-4330-aaaf-e5454e2c2af1') set(my_args, 'algo', 'private sign1') my_args_str = encode_args(my_args) add_rsp_header('X-AScript-ENCODE-ARGS', my_args_str) to_args = decode_args(my_args_str) if get(to_args, 'algo') { add_rsp_header('X-AScript-DECODE-ARGS-ALGO', get(to_args, 'algo')) } if get(to_args, 'signature') { add_rsp_header('X-AScript-DECODE-ARGS-SIGN', get(to_args, 'signature')) } ``` 输出3个响应头： ``` X-AScript-ENCODE-ARGS: signature=da9dc4b7-87ae-4330-aaaf-e5454e2c2af1&algo=private%20sign1 X-AScript-DECODE-ARGS-ALGO: private sign1 X-AScript-DECODE-ARGS-SIGN: da9dc4b7-87ae-4330-aaaf-e5454e2c2af1 ```

rewrite

项目	描述
语法	`rewrite(url, flag, code)` 。
说明	改写操作或重定向操作。
参数	• url：目标URL，字符类型。 ◦ 当flag=redirect或flag=break时，仅改写URI，则参数URL表示改写后的目标URI。 ◦ 当flag=enhance_redirect或flag=enhance_break时，修改整个URI和参数，参数URL表示改写后的目标URI和参数。 • flag：重写模式，字符类型。 ◦ redirect ：仅修改URI，不修改参数；默认执行302跳转至URL；如果指定参数 `code` ，则响应码可自定义为：301、302（默认）、303、307或308。 ◦ break ：仅修改URI，不修改参数，将URI修改为URL。 ◦ enhance_redirect：与 `redirect` 类似，但是修改整个URI和参数。 ◦ enhance_break：与 `break` 类似，但是修改整个URI和参数。 • code：为响应码，数字类型 。 当flag=redirect或flag=enhance_redirect时，可自定义响应码。

项目	描述
返回值	• 对于改写操作，默认返回 `true` 。 • 对于重定向操作，默认不返回。
示例	`if and($arg_mode, eq($arg_mode, 'rewrite:enhance_break')) {` `  rewrite('/a/b/c.txt?k=v', 'enhance_break')` `}` **说明：回源和缓存的uri+参数，修改为/a/b/c.txt?k=v** `if and($arg_mode, eq($arg_mode, 'rewrite:enhance_redirect')) {` `  rewrite('/a/b/c.txt?k=v', 'enhance_redirect')` `}` `if and($arg_mode, eq($arg_mode, 'rewrite:enhance_redirect_301')) {` `  rewrite('/a/b/c.txt?k=v', 'enhance_redirect', 301)` `}` **说明：302或301跳转至/a/b/c.txt?k=v** `if and($arg_mode, eq($arg_mode, 'rewrite:break')) {` `  rewrite('/a/b/c.txt', 'break')` `}` **说明：回源和缓存的uri，修改为/a/b/c.txt，保持原参数不变** `if and($arg_mode, eq($arg_mode, 'rewrite:redirect')) {` `  rewrite('/a/b/c.txt', 'redirect')` `}` `if and($arg_mode, eq($arg_mode, 'rewrite:redirect_301')) {` `  rewrite('/a/b/c.txt', 'redirect', 301)` `}` **说明：302或301跳转至/a/b/c.txt，保持原参数不变**

say

项目	描述
语法	`say(arg)` 。
说明	输出响应体，并在行尾追加换行符。
参数	arg：任意类型。
返回值	无。
示例	`say('hello')` `print('byebye')` `print('byebye')` 输出： `hello` `byebyebyebye`

print

项目	描述
语法	`print(arg)` 。
说明	输出响应体与 `say()` 相同，但不会在行尾追加换行符。
参数	arg：任意类型。
返回值	无。
示例	`say('hello')` `print('byebye')` `print('byebye')` 输出： `hello` `byebyebyebye`

exit

项目	描述
语法	`exit(code [, body])` 。
说明	以状态码 `code` 结束当前请求。若有 `body` ，则为响应体。
参数	• code：响应状态码。 • body：响应体。
返回值	无。

项目	描述
示例	• 示例1 `if not($arg_key) {` `exit(403)` `}` **说明：如果请求未携带参数key时，403拒绝请求。** `if not($cookie_user) {` `exit(403, 'not cookie user')` `}` **说明：当请求未携带cookie user时，403拒绝请求，响应body为'not cookie user'** `if not(0) {` `exit(403)` `}` **说明：not(0)的结果为false** `if not(false) {` `exit(403)` `}` **说明：not(false)的结果为true** • 示例2 `pcs = capture_re($request_uri,'^/([^/]+)/([^/]+)([^?]+)\?(.*)')` `sec1 = get(pcs, 1)` `sec2 = get(pcs, 2)` `sec3 = get(pcs, 3)` `if or(not(sec1), not(sec2), not(sec3)) {` `add_rsp_header_imm('X-TENGINE-ERROR', 'auth failed - missing necessary uri set')` `exit(403)` `}` `digest = md5(concat(sec1, sec3))` `if ne(digest, sec2) {` `add_rsp_header_imm('X-TENGINE-ERROR', 'auth failed - invalid digest')` `exit(403)` `}`

4.2.6.7. 时间相关函数

本文为您介绍时间相关函数的语法、说明、参数、返回值和示例。

today | time | now | localtime | utctime | cookie_time | http_time | parse_http_time | unixtime

today

项目	描述
语法	`today()` 。
说明	返回当前时间字符串，格式：yyyy-mm-dd。

项目	描述
参数	无。
返回值	返回当前时间字符串，格式：yyyy-mm-dd。
示例	`say(concat('today:', today()))` 输出： `today:2021-12-29`

time

项目	描述
语法	`time()` 。
说明	返回当前的UNIX时间戳（不包含毫秒的小数部分），单位：秒。
参数	无。
返回值	返回当前的UNIX时间戳。
示例	`say(concat('time:', time()))` 输出： `time:1559109666`

now

项目	描述
语法	`now()` 。
说明	返回当前时间字符串（包含毫秒的小数部分），单位：秒。
参数	无。
返回值	返回当前的UNIX时间戳。

项目	描述
示例	`say(concat('now:', now()))` 输出： `now:1559109666.644`

localtime

项目	描述
语法	`localtime()` 。
说明	返回当前时间字符串，格式：yyyy-mm-dd hh:mm:ss。
参数	无。
返回值	返回当前时间字符串，格式：yyyy-mm-dd hh:mm:ss。
示例	`say(concat('localtime:', localtime()))` 输出： `localtime:2021-12-29 14:02:41`

utctime

项目	描述
语法	`utctime()` 。
说明	返回当前时间字符串（UTC时间），格式：yyyy-mm-dd hh:mm:ss。
参数	无。
返回值	返回当前时间字符串，格式：yyyy-mm-dd hh:mm:ss。
示例	`say(concat('utctime:', utctime()))` 输出： `utctime:2021-12-29 06:02:41`

cookie_time

项目	描述
语法	`cookie_time(sec)` 。
说明	生成cookie格式的时间字符串。
参数	sec：UNIX时间戳。例如：调用 `time()` 获取。
返回值	基于 `sec` 表示的UNIX时间戳，返回cookie格式的时间字符串。
示例	`say(concat('cookie_time:', cookie_time(time())))` 输出： `cookie_time:Wed, 29-Dec-21 06:02:41 GMT`

http_time

项目	描述
语法	`http_time(sec)` 。
说明	生成HTTP header格式的时间字符串。例如：Last-Modified。
参数	sec：UNIX时间戳。例如：调用 `time()` 获取。
返回值	基于 `sec` 表示的UNIX时间戳，返回HTTP header格式的时间字符串，用于HTTP头的时间。
示例	`say(concat('http_time:', http_time(time())))` 输出 `http_time:Wed, 29 Dec 2021 06:02:41 GMT`

parse_http_time

项目	描述
语法	`parse_http_time(str)` 。
说明	解析HTTP header格式的时间字符串，并返回对应的UNIX时间戳。
参数	str：待转换的HTTP header格式的时间字符串。格式： `Thu, 22-Dec-10 10:20:35 GMT` 。调用 `http_time()` 获取。
返回值	成功返回UNIX时间戳，失败返回 `false` 。

项目	描述
示例	`say(concat('parse_http_time:', parse_http_time(http_time(time()))))` 输出 `parse_http_time:1559109761`

unixtime

项目	描述
语法	`unixtime(year, month, day, hour, min, sec)` 。
说明	根据年、月、日、时、分、秒，生成UNIX时间戳并返回。
参数	• year：指定年。 • month：指定月。 • day：指定日。 • hour：指定小时。 • min：指定分钟。 • sec：指定秒。
返回值	返回转换后的UNIX时间戳。
示例	`t = UNIXtime(1970, 1, 1, 8, 0, 0)` `say(concat('UNIXtime()=', t))` 输出 `UNIXtime()=0`

4.2.6.8. 密码算法相关函数

本文为您介绍密码算法相关函数的语法、说明、参数、返回值和示例。

aes_new | aes_enc | aes_dec | sha1 | sha2 | hmac | hmac_sha1 | md5 | md5_bin

aes_new

项目	描述
语法	`aes_new(config)` 。
说明	创建AES对象，用于后续的 `aes_enc()` 加密和 `aes_dec()` 解密。

项目	描述
参数	`config` 参数为字典类型，包含如下参数： • （必选）key：密钥，字符串类型。 • （可选）salt：盐值，字符串类型。 • （必选）cipher_len：密码器长度。取值范围：128、192和256。 • （必选）cipher_mode：密码器模式。取值范围：ecb、cbc、ctr、cfb和ofb。 • （可选）iv：初始向量，字符串类型。
返回值	成功返回AES对象（字典类型），失败返回 `false` 。
示例	

```
aes_conf = []
plaintext = ''
if and($http_mode, eq($http_mode, 'ecb-128')) {
    set(aes_conf, 'key', 'ab8bfd9f-a1af-4ba2-bbb0-1ee520e3d8bc')
    set(aes_conf, 'salt', '1234567890')
    set(aes_conf, 'cipher_len', 128)
    set(aes_conf, 'cipher_mode', 'ecb')
    plaintext = 'hello aes ecb-128'
}
if and($http_mode, eq($http_mode, 'cbc-256')) {
    set(aes_conf, 'key', '146ebcc8-392b-4b3a-a720-e7356f62')
    set(aes_conf, 'cipher_len', 256)
    set(aes_conf, 'cipher_mode', 'cbc')
    set(aes_conf, 'iv', '0123456789abcdef')
    plaintext = 'hello aes cbc-256'
}
if and($http_mode, eq($http_mode, 'ofb-256')) {
    set(aes_conf, 'key', '146ebcc8-392b-4b3a-a720-e7356f62')
    set(aes_conf, 'cipher_len', 256)
    set(aes_conf, 'cipher_mode', 'ofb')
    set(aes_conf, 'iv', tochar(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0))
    plaintext = 'hello aes ofb-256'
}
aes_obj = aes_new(aes_conf)
if not(aes_obj) {
    say(concat('aes obj failed'))
    exit(400)
}
ciphertext = aes_enc(aes_obj, plaintext)
plaintext_reverse = aes_dec(aes_obj, ciphertext)
say(concat('plain: ', plaintext))
say(concat('cipher: ', tohex(ciphertext)))
say(concat('plain_reverse: ', plaintext_reverse))
if ne(plaintext, plaintext_reverse) {
    say('plaintext ~= plaintext_reverse')
    exit(400)
}
```

项目	描述

aes_enc

项目	描述
语法	`aes_enc(o, s)` 。
说明	AES加密。
参数	• s：明文。 • o：表示 `aes_new` 返回的AES对象。
返回值	返回对 `s` 加密后的密文。

项目	描述
示例	

```
aes_conf = []
plaintext = ''
if and($http_mode, eq($http_mode, 'ecb-128')) {
    set(aes_conf, 'key', 'ab8bfd9f-a1af-4ba2-bbb0-1ee520e3d8bc')
    set(aes_conf, 'salt', '1234567890')
    set(aes_conf, 'cipher_len', 128)
    set(aes_conf, 'cipher_mode', 'ecb')
    plaintext = 'hello aes ecb-128'
}
if and($http_mode, eq($http_mode, 'cbc-256')) {
    set(aes_conf, 'key', '146ebcc8-392b-4b3a-a720-e7356f62')
    set(aes_conf, 'cipher_len', 256)
    set(aes_conf, 'cipher_mode', 'cbc')
    set(aes_conf, 'iv', '0123456789abcdef')
    plaintext = 'hello aes cbc-256'
}
if and($http_mode, eq($http_mode, 'ofb-256')) {
    set(aes_conf, 'key', '146ebcc8-392b-4b3a-a720-e7356f62')
    set(aes_conf, 'cipher_len', 256)
    set(aes_conf, 'cipher_mode', 'ofb')
    set(aes_conf, 'iv', tochar(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0))
    plaintext = 'hello aes ofb-256'
}
aes_obj = aes_new(aes_conf)
if not(aes_obj) {
    say(concat('aes obj failed'))
    exit(400)
}
ciphertext = aes_enc(aes_obj, plaintext)
plaintext_reverse = aes_dec(aes_obj, ciphertext)
say(concat('plain: ', plaintext))
say(concat('cipher: ', tohex(ciphertext)))
say(concat('plain_reverse: ', plaintext_reverse))
if ne(plaintext, plaintext_reverse) {
    say('plaintext ~= plaintext_reverse')
    exit(400)
}
```

aes_dec

项目	描述
语法	`aes_dec(o, s)` 。
说明	AES解密。
参数	• s：密文。 • o：表示 `aes_new` 返回的AES对象。
返回值	返回对 `s` 解密后的明文。
示例	

```
aes_conf = []
plaintext = ''
if and($http_mode, eq($http_mode, 'ecb-128')) {
    set(aes_conf, 'key', 'ab8bfd9f-a1af-4ba2-bbb0-1ee520e3d8bc')
    set(aes_conf, 'salt', '1234567890')
    set(aes_conf, 'cipher_len', 128)
    set(aes_conf, 'cipher_mode', 'ecb')
    plaintext = 'hello aes ecb-128'
}
if and($http_mode, eq($http_mode, 'cbc-256')) {
    set(aes_conf, 'key', '146ebcc8-392b-4b3a-a720-e7356f62')
    set(aes_conf, 'cipher_len', 256)
    set(aes_conf, 'cipher_mode', 'cbc')
    set(aes_conf, 'iv', '0123456789abcdef')
    plaintext = 'hello aes cbc-256'
}
if and($http_mode, eq($http_mode, 'ofb-256')) {
    set(aes_conf, 'key', '146ebcc8-392b-4b3a-a720-e7356f62')
    set(aes_conf, 'cipher_len', 256)
    set(aes_conf, 'cipher_mode', 'ofb')
    set(aes_conf, 'iv', tochar(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0))
    plaintext = 'hello aes ofb-256'
}
aes_obj = aes_new(aes_conf)
if not(aes_obj) {
    say(concat('aes obj failed'))
    exit(400)
}
ciphertext = aes_enc(aes_obj, plaintext)
plaintext_reverse = aes_dec(aes_obj, ciphertext)
say(concat('plain: ', plaintext))
say(concat('cipher: ', tohex(ciphertext)))
say(concat('plain_reverse: ', plaintext_reverse))
if ne(plaintext, plaintext_reverse) {
    say('plaintext ~= plaintext_reverse')
    exit(400)
}
```

项目	描述

sha1

项目	描述
语法	`sha1(s)` 。
说明	计算SHA1摘要。
参数	s：待计算摘要的字符串。
返回值	返回SHA1摘要的二进制形式。

项目	描述
示例	`digest = sha1('hello sha')` `say(concat('sha1:', tohex(digest)))` 输出： `sha1:853789bc783a6b573858b6cc9f913afe82962956`

sha2

项目	描述
语法	`sha2(s, l)` 。
说明	计算SHA2摘要。
参数	• s：待计算摘要的字符串。 • l：指明SHA2长度。取值范围：224、256、384和512。
返回值	使用SHA2摘要的二进制形式。

项目	描述
示例	`digest = sha2('hello sha2', 224)` `say(concat('sha2-224:', tohex(digest)))` `digest = sha2('hello sha2', 256)` `say(concat('sha2-256:', tohex(digest)))` `digest = sha2('hello sha2', 384)` `say(concat('sha2-384:', tohex(digest)))` `digest = sha2('hello sha2', 512)` `say(concat('sha2-512:', tohex(digest)))` 输出： `sha2-224:b24b7effcf53ce815ee7eb73c7382613aba1c334e2a1622655362927` `sha2-256:af0425cee23c236b326ed1f008c9c7c143a611859a11e87d66d0a4c3217c7792` `sha2-384:bebbdde9efabd4b9cf90856cf30e0b024dd13177d9367d2dcf8d7a04e059f92260f16b21e261358c2271be32086ef35b` `sha2-512:a1d1aef051c198c0d26bc03500c177a315fa248cea815e04fbb9a75e5be5061617daab311c5e3d0b215dbfd4e83e73f23081242b0143dcdfce5cd92ec51394f7`

hmac

项目	描述
语法	`hmac(k, s, v)` 。
说明	计算HMAC类算法摘要。
参数	• k：算法密钥。 • s：待计算摘要的字符串。 • v：算法版本。取值范围： md5、sha256和sha512。
返回值	使用对应算法HMAC摘要的二进制形式。

项目	描述
示例	``` k = '146ebcc8-392b-4b3a-a720-e7356f62f87b' v = 'hello mac' say(concat('hmac(md5): ', tohex(hmac(k, v, 'md5')))) say(concat('hmac(sha1): ', tohex(hmac(k, v, 'sha1')))) say(concat('hmac(sha256): ', tohex(hmac(k, v, 'sha256')))) say(concat('hmac(sha512): ', tohex(hmac(k, v, 'sha512')))) say(concat('hmac_sha1(): ', tohex(hmac_sha1(k, v)))) ``` 输出： ``` hmac(md5): 358cbfca8ad663b547c83748de2ea778 hmac(sha1): 5555633cef48c3413b68f9330e99357df1cc3d93 hmac(sha256): 7a494543cad3b92ce1e7c4bbc86a8f5212b53e4d661f7830f455847540a85771 hmac(sha512): 59d7c07996ff675b45bd5fd40a6122bb5f40f597357a9b4a9e29da6f5c7cb80679 8c016fe09cb46457b6df9717d26d0af19896f72eaf4296be03e3681fea59ad hmac_sha1(): 5555633cef48c3413b68f9330e99357df1cc3d93 ```

hmac_sha1

项目	描述
语法	`hmac_sha1(k, s)` 。
说明	计算HMAC-SHA-1摘要。
参数	• s：待计算摘要的字符串。 • k：HMAC-SHA-1密钥。
返回值	返回HMAC-SHA-1摘要的二进制形式。

项目	描述
示例	k = '146ebcc8-392b-4b3a-a720-e7356f62f87b' v = 'hello mac' say(concat('hmac(md5): ', tohex(hmac(k, v, 'md5')))) say(concat('hmac(sha1): ', tohex(hmac(k, v, 'sha1')))) say(concat('hmac(sha256): ', tohex(hmac(k, v, 'sha256')))) say(concat('hmac(sha512): ', tohex(hmac(k, v, 'sha512')))) say(concat('hmac_sha1(): ', tohex(hmac_sha1(k, v)))) 输出： hmac(md5): 358cbfca8ad663b547c83748de2ea778 hmac(sha1): 5555633cef48c3413b68f9330e99357df1cc3d93 hmac(sha256): 7a494543cad3b92ce1e7c4bbc86a8f5212b53e4d661f7830f455847540a85771 hmac(sha512): 59d7c07996ff675b45bd5fd40a6122bb5f40f597357a9b4a9e29da6f5c7cb80679 8c016fe09cb46457b6df9717d26d0af19896f72eaf4296be03e3681fea59ad hmac_sha1(): 5555633cef48c3413b68f9330e99357df1cc3d93

md5

项目	描述
语法	`md5(s)` 。
说明	计算MD5摘要。
参数	s：待计算摘要的字符串。
返回值	返回MD5摘要的十六进制形式。
示例	say(concat('md5: ', md5('hello md5'))) 输出： md5:741fc6b1878e208346359af502dd11c5

md5_bin

项目	描述
语法	`md5_bin(s)` 。
说明	计算MD5摘要。
参数	s：待计算摘要的字符串。

项目	描述
返回值	返回MD5摘要的二进制形式。
示例	`say(concat('md5_bin: ', tohex(md5_bin('hello md5'))))` 输出： `md5_bin: 741fc6b1878e208346359af502dd11c5`

4.2.6.9. JSON相关函数

本文为您介绍JSON相关函数的语法、说明、参数、返回值和示例。

json_enc | json_dec

json_enc

项目	描述
语法	`json_enc(d)` 。
说明	JSON编码。
参数	d：待编码的字典对象。
返回值	成功返回编码后的字符串，失败返回 `false` 。
示例	`var_a = []` `var_b = ['v1', 'v2']` `set(var_a, 'k1', 'v1')` `set(var_a, 'k2', var_b)` `var_c = '{"k1":"v1","k2":["v1","v2"]}'` `say(concat('json_enc=', json_enc(var_a)))` `say(concat('json_dec=', get(json_dec(var_c), 'k1')))` 输出： `json_enc={"k1":"v1","k2":["v1","v2"]}` `json_dec=v1`

json_dec

项目	描述
语法	`json_dec(s)` 。
说明	JSON解码。

项目	描述
参数	s：待解码的JSON格式字符串。
返回值	成功返回解码后的字典，失败返回 `false` 。
示例	`var_a = []` `var_b = ['v1', 'v2']` `set(var_a, 'k1', 'v1')` `set(var_a, 'k2', var_b)` `var_c = '{"k1":"v1","k2":["v1","v2"]}'` `say(concat('json_enc=', json_enc(var_a)))` `say(concat('json_dec=', get(json_dec(var_c), 'k1')))` 输出： `json_enc={"k1":"v1","k2":["v1","v2"]}` `json_dec=v1`

4.2.6.10. Misc相关函数

本文为您介绍Misc相关函数的语法、说明、参数、返回值和示例。

base64_enc | base64_dec | url_escape | url_unescape | randomseed | rand | rand_hit | crc | tonumber | base64_enc_safe | base64_dec_safe

base64_enc

项目	描述
语法	`base64_enc(s [, no_padding])` 。
说明	base64编码。
参数	• s：待编码的字符串。 • no_padding： `true` 表示无填充，默认 `false` 。
返回值	base64编码后的字符串。
示例	`if $http_data {` `  decdata = base64_dec($http_data)` `  say(concat('base64_decdata=', decdata))` `  say(concat('base64_encdata=', base64_enc('hello, dsl')))` `}` **请求**header: "data: aGVsbG8sIGRzbA==" **响应：** base64_decdata=hello, dsl base64_encdata=aGVsbG8sIGRzbA==

base64_dec

项目	描述
语法	`base64_dec(s)` 。
说明	base64解码。
参数	s：待解码的字符串。
返回值	base64解码后的字符串。
示例	`if $http_data {` `  decdata = base64_dec($http_data)` `  say(concat('base64_decdata=', decdata))` `  say(concat('base64_encdata=', base64_enc('hello, dsl')))` `}` **请求**header: "data: aGVsbG8sIGRzbA==" **响应：**base64_decdata=hello, dsl base64_encdata=aGVsbG8sIGRzbA==

url_escape

项目	描述
语法	`url_escape(s)` 。
说明	URL编码。
参数	s：待编码的字符串。
返回值	URL编码后的字符串。
示例	`raw = '/abc/123/ dd/file.m3u8'` `esdata = url_escape(raw)` `dsdata = url_unescape(esdata)` `if eq(raw, dsdata) {` `  say(concat('raw=', raw))` `  say(concat('dsdata=', dsdata))` `}` 输出： `raw=/abc/123/ dd/file.m3u8` `esdata=%2Fabc%2F123%2F%20dd%2Ffile.m3u8` `dsdata=/abc/123/ dd/file.m3u8`

url_unescape

项目	描述
语法	`url_unescape(s)` 。
说明	URL解码。
参数	s：待解码的字符串。
返回值	返回URL解码后的字符串。
示例	`raw = '/abc/123/ dd/file.m3u8'` `esdata = url_escape(raw)` `dsdata = url_unescape(esdata)` `if eq(raw, dsdata) {` `say(concat('raw=', raw))` `say(concat('dsdata=', dsdata))` `}` 输出： `raw=/abc/123/ dd/file.m3u8` `esdata=%2Fabc%2F123%2F%20dd%2Ffile.m3u8` `dsdata=/abc/123/ dd/file.m3u8`

randomseed

项目	描述
语法	`randomseed()` 。
说明	指定生成随机数种子。
参数	无。
返回值	无。
示例	`randomseed()` `r = rand(1,100)`

rand

项目	描述
语法	`rand(n1, n2)` 。
说明	生成随机数，随机数范围：n1≤返回值≤n2。

项目	描述
参数	• n1：随机数下限。 • n2：随机数上限。
返回值	返回生成的随机数。
示例	`r = rand(1,100)`

rand_hit

项目	描述
语法	`rand_hit(ratio)` 。
说明	按指定概率返回真假。
参数	ratio：为真概率，有效值范围为[0~100]。
返回值	按ratio概率返回 `true` 。例如：当ratio为100时，返回 `true` ，当ratio为0时，返回 `false` 。
示例	`rand_hit(80)`

crc

项目	描述
语法	`crc(s)` 。
说明	计算crc摘要。
参数	s：待计算摘要的字符串。
返回值	返回 `s` 的crc摘要。
示例	`crc('hello edgescript')`

tonumber

项目	描述
语法	`tonumber(s [, base])` 。

项目	描述
说明	类型转换，将字符串类型转换为数字类型。
参数	• s：待转换的字符串。 • base：可指定目标转换进制，可用值：10和16，默认10进制。
示例	`n = tonumber('100')` `say(concat('tonumber()=', n))` **输出：** `tonumber()=100`

base64_enc_safe

项目	描述
语法	`base64_enc_safe(str)` 。
说明	对输入的字符串进行Base64安全编码。安全编码后输出时，需要将“+”替换成“-”、“/”替换成“_”，同时去掉编码后的“=”。
参数	str：待加密的字符串。
返回值	返回字符串类型
示例	`add_rsp_header('X-RESPOND-OUTPUT', concat('base64_enc_safe=', base64_enc_safe('hello, dsl')), true)` 输出响应头： `X-RESPOND-OUTPUT: base64_enc_safe=aGVsbG8sIGRzbA`

base64_dec_safe

项目	描述
语法	`base64_dec_safe(str)` 。
说明	对输入的字符串进行Base64安全解码。安全解码后输出时，需要将“-”替换成“+”、“_”替换成“/”，末尾用“=”按照4的余数补齐。
参数	str：Base64加密后的内容。
返回值	返回字符串类型。

项目	描述
示例	`add_rsp_header('X-RESPOND-OUTPUT', concat('base64_dec_safe=', base64_dec_safe(base64_enc_safe('hello, dsl'))), true)` 输出响应头： `X-RESPOND-OUTPUT:base64_dec_safe=hello, dsl`

4.2.6.11. 数组类型相关函数

本文为您介绍数组类型相关函数的语法、说明、参数、示例和返回值。

arr_concat | arr_insert | arr_remove | arr_sort | arr_len

arr_concat

项目	描述
语法	arr_concat(tbl, sep)
说明	使用arr_concat将字段表中的字符串按照指定的字符进行连接。
参数	• tbl：数组变量。 • sep（可选参数）：拼接符，默认为空串拼接。
示例	`d = ['t1','t2','t3']` `say(arr_concat(d, '&'))`
返回值	返回拼接后的字符串，返回值为字符类型。本示例的返回值为 `t1&t2&t3` 。

arr_insert

项目	描述
语法	arr_insert(list, value, [pos])
说明	使用arr_insert向数组中插入值。
参数	• list：数组类型。 • value：任意类型。 • pos：number类型，不能为0。list索引默认从1开始，在list的pos处插入元素value，并将元素pos+1后移至尾部元素；如果未指定pos，默认从末尾插入。

项目	描述
示例	``` tbl_1 = [] arr_insert(tbl_1, '1') arr_insert(tbl_1, '3') arr_insert(tbl_1, '5') arr_insert(tbl_1, '2') arr_insert(tbl_1, '6', 1) str = arr_concat(tbl_1, '') say(concat('arr_insert:', str)) ```
返回值	始终返回 `true` 。本示例的返回值为 `arr_insert:61352` 。

arr_remove

项目	描述
语法	arr_remove(list, [pos])
说明	使用arr_remove移除list中指定位置的元素并返回被移除元素的值，无pos时返回尾部位置的元素对应的值。
参数	• list：数组类型。 • pos：number类型。
示例	``` tbl_1 = [] arr_insert(tbl_1, '1') arr_insert(tbl_1, '3') arr_insert(tbl_1, '5') arr_insert(tbl_1, '2') say(concat('arr_remove:', arr_remove(tbl_1, 2))) ```
返回值	返回值为被移除位置的数据。本示例的返回值为 `arr_remove:3` 。

arr_sort

项目	描述
语法	arr_sort(list, [comp])
说明	使用arr_sort对list元素从索引的头部到索引的尾部按指定次序排序。 • 如果提供了comp，comp必须是一个可以接收两个列表内元素为参数的函数。当第一个元素需要排在第二个元素之前时，返回true。 • 如果未提供comp，则按字符的ANSII码从小到大排序。这种排序算法不稳定，即当两个元素次序相等时，这两个元素在排序后的相对位置可能会改变。

项目	描述
参数	• list：数组类型。 • comp：自定义排序算法函数，该参数的类型为函数。
示例	`tbl_1 = []` `arr_insert(tbl_1, '1')` `arr_insert(tbl_1, '3')` `arr_insert(tbl_1, '5')` `arr_insert(tbl_1, '2')` `say(concat('remove:', arr_remove(tbl_1, 2)))` `str = arr_concat(tbl_1, '')` `say(concat('insert:', str))` `arr_sort(tbl_1)` `str = arr_concat(tbl_1, '')` `say(concat('sort:', str))` `def my_comp(a, b){` `  a = tonumber(a)` `  b = tonumber(b)` `  if gt(a, b) {` `  return true` `  }` `  return false` `}` `arr_sort(tbl_1, my_comp)` `str = arr_concat(tbl_1, '')` `say(concat('sort_comp:', str))`
返回值	始终返回 `true` 。本示例的返回值如下： `remove:3` `insert:152` `sort:125` `sort_comp:521`

arr_len

项目	描述
语法	arr_len(arr)
说明	使用arr_len计算数组的元素个数。
参数	arr：数组变量。

项目	描述
示例	`d = []` `set(d, 1, 'v1')` `say(arr_len(d))`
返回值	返回值为数字类型，本示例的返回值为 `1` 。

4.2.6.12. 请求判断相关函数

本文为您介绍请求判断相关函数的语法、说明、参数、示例和返回值。

server_addr | server_port | client_addr | client_port | req_uri | req_uri_basename | req_uri_ext | req_uri_seg | req_uri_arg | req_uri_query_string | req_scheme | req_method | req_host | req_user_agent | req_referer | req_cookie | req_first_x_forwarded | req_header | req_id

server_addr

项目	描述
语法	server_addr()
说明	使用server_addr返回接收了当前请求的服务器地址。
参数	无
示例	`s_addr = server_addr()` `say(concat('s_addr:', s_addr))`
返回值	返回服务器地址，返回值为字符串类型。

server_port

项目	描述
语法	server_port()
说明	使用server_port返回接收了当前请求的服务器端口。
参数	无
示例	`s_addr = server_port()` `say(concat('s_port:', s_port))`
返回值	返回服务器端口，返回值为数字类型。

client_addr

> **注意** 风险提示：因运营商网络NAT策略导致客户端地址被修改，会影响该接口真实的返回值，请谨慎使用。

项目	描述
语法	client_addr()
说明	使用client_addr返回客户端IP地址。
参数	无
示例	`c_addr = client_addr()` `c_port = client_port()` `say(concat('c_addr:', c_addr))` `say(concat('c_port:', tostring(c_port)))`
返回值	返回客户端IP地址，返回值为字符串类型。

client_port

项目	描述
语法	client_port()
说明	使用client_port返回客户端的端口。
参数	无
示例	`c_addr = client_addr()` `c_port = client_port()` `say(concat('c_addr:', c_addr))` `say(concat('c_port:', tostring(c_port)))`
返回值	返回客户端的端口，返回值为数字类型。

req_uri

项目	描述
语法	req_uri([pattern])
说明	• 如果无pattern参数，则返回请求URI，不包含参数部分。 • 如果有pattern参数，则针对请求URI进行匹配判断。

项目	描述
参数	pattern：使用该参数进行匹配，支持以下两种模式。 • 简单匹配：相等判断，默认为简单匹配。 • 正则匹配：`re:` 前导的正则表达式。
示例	``` # req_uri say(concat('req_uri: ', req_uri())) if req_uri('/path1/path2') { say('req_uri: plain match') } if req_uri('re:/path[0-9]/path[0-9]') { say('req_uri: regex match') } ```
返回值	• 无pattern参数，返回请求URI。返回值为字符串类型。 • 有pattern参数，返回 `true` ，表示匹配成功；返回 `false` ，表示匹配失败。 本示例的返回值如下： **请求：**`/path1/path2?mode=ip` **响应：** ``` req_uri: /path1/path2 req_uri: plain match req_uri: regex match ```

req_uri_basename

项目	描述
语法	req_uri_basename([pattern])
说明	• 如果无pattern参数，则返回请求URI中的文件名部分。 • 如果有pattern参数，则针对请求URI中的文件名部分进行匹配判断。 文件名部分示例如下： • 示例1：对于/document_detail/30360.html，文件名部分为30360。 • 示例2：对于/M604/guopei_mp4/ZYJY2017BJGL0101/2-1_g.mp4，文件名部分为2-1_g。 • 示例3：对于/tarball/foo.tar.bz2，文件名部分为foo。
参数	pattern：使用该参数进行匹配，支持以下两种模式。 • 简单匹配：相等判断，默认为简单匹配。 • 正则匹配：`re:` 前导的正则表达式。

项目	描述
示例	`# req_uri_basename` `basename = req_uri_basename()` `say(concat('req_uri_basename: ', basename, ' ', len(basename)))` `if req_uri_basename('foo') {` `  say('req_uri_basename: plain match')` `}` `if req_uri_basename('re:^f.*') {` `  say('req_uri_basename: regex match')` `}`
返回值	• 无pattern参数，返回请求URI中的文件名部分。返回值为字符串类型。 • 有pattern参数，返回 `true` ，表示匹配成功；返回 `false` ，表示匹配失败。 本示例的返回值如下： **请求：** `/path1/path2/foo.tar.bz2` **响应：** `req_uri_basename: foo 3` `req_uri_basename: plain match` `req_uri_basename: regex match`

req_uri_ext

项目	描述
语法	req_uri_ext([pattern])
说明	• 如果无pattern参数，则返回请求URI中的扩展名部分。 • 如果有pattern参数，则针对请求URI中的扩展名部分进行匹配判断。 扩展名部分示例如下： • 示例1：对于/document_detail/30360.html，扩展名部分为.html。 • 示例2：对于/M604/guopei_mp4/ZYJY2017BJGL0101/2-1_g.mp4，扩展名部分为.mp4。 • 示例3：对于/tarball/foo.tar.bz2，扩展名部分为.tar.bz2。
参数	pattern：使用该参数进行匹配，支持以下两种模式。 • 简单匹配：相等判断，默认为简单匹配。 • 正则匹配： `re:` 前导的正则表达式。

项目	描述
示例	`# req_uri_ext` `ext = req_uri_ext()` `say(concat('req_uri_ext: ', ext, ' ', len(ext)))` `if req_uri_ext('.tar.bz2') {` `  say('req_uri_ext: plain match')` `}` `if req_uri_ext('re:\.tar\.bz[0-2]') {` `  say('req_uri_ext: regex match')` `}`
返回值	• 无pattern参数，返回请求URI中的扩展名部分。返回值为字符串类型。 • 有pattern参数，返回 `true` ，表示匹配成功；返回 `false` ，表示匹配失败。 本示例的返回值如下： **请求：** `/path1/path2/foo.tar.bz2` **响应：** `req_uri_ext: .tar.bz2 8` `req_uri_ext: plain match` `req_uri_ext: regex match`

req_uri_seg

项目	描述
语法	req_uri_seg([idx])
说明	• 使用正斜线（/）对uri进行分隔并返回所有段落。 ◦ 如果无idx参数，返回所有段落。 ◦ 如果有idx参数，返回指定索引后的所有段落，包含对应的索引。 • 段落索引：段落索引从头部递增，依次从左向右。 • 段落上限：段落上限为128个字符，超出上限的段落会被丢弃。
参数	idx（可选参数）：允许指定起始索引。

项目	描述
示例	<pre># req_uri_seg def echo_each(k, v, u) { say(concat(get(u, 'msg'), ' : segs[', k, ']=', v)) } # fetch all segments segs = req_uri_seg() foreach(segs, echo_each, ['msg'='req_uri_seg()']) # fetch segments from idx 3 segs = req_uri_seg(3) if get(segs, 3) { say(concat('req_uri_seg(3): segs[3]=', get(segs, 3))) } if get(segs, 4) { say(concat('req_uri_seg(3): segs[4]=', get(segs, 4))) } if get(segs, 5) { say(concat('req_uri_seg(3): segs[5]=', get(segs, 5))) }</pre>
返回值	返回值为字典类型，包含相应段落。 **说明** 从返回字典中获取指定索引段落时，必须判断是否为空。 本示例的返回值如下： **请求：**/path1/path2/path3/path4?mode=req2 **响应：** <pre>req_uri_seg() : segs[1]=path1 req_uri_seg() : segs[2]=path2 req_uri_seg() : segs[3]=path3 req_uri_seg() : segs[4]=path4 req_uri_seg(3): segs[3]=path3 req_uri_seg(3): segs[4]=path4</pre>

req_uri_arg

项目	描述
语法	req_uri_arg(name, [pattern])
说明	使用req_uri_arg默认返回指定参数的值，如果有pattern参数，则对指定参数的值进行匹配判断。

项目	描述
参数	• name：参数名称。 • pattern：使用该参数进行匹配，支持以下两种模式。 ○ 简单匹配：相等判断，默认为简单匹配。 ○ 正则匹配：`re:` 前导的正则表达式。
示例	`# req_uri_arg` `uid = req_uri_arg('uid')` `if uid {` `  say(concat('found uid ', uid))` `} else {` `  say('not found uid')` `}` `uid_chk = req_uri_arg('uid', '058334')` `if uid_chk {` `  say('check uid ok. plain mode')` `} else {` `  say('check uid fail. plain mode')` `}` `uid_chk = req_uri_arg('uid', 're:[0-9]+')` `if uid_chk {` `  say('check uid ok. regex mode')` `} else {` `  say('check uid fail. regex mode')` `}`

项目	描述
返回值	• 如果无pattern参数 ◦ 参数存在：返回name指定参数的字符串值。 ◦ 参数不存在：返回false。 • 如果有pattern参数 ◦ 参数存在：进行匹配，返回 `true` ，表示匹配成功；返回 `false` ，表示匹配失败。 ◦ 参数不存在：返回 `false` 。 本示例的返回值如下： **请求：** /path1/path2/path3/path4?mode=req4&uid **响应：** not found uid check uid fail. plain mode check uid fail. regex mode **请求：** /path1/path2/path3/path4?mode=req4&uid= **响应：** found uid check uid fail. plain mode check uid fail. regex mode **请求：** /path1/path2/path3/path4?mode=req4&uid=12345 **响应：** found uid 12345 check uid fail. plain mode check uid ok. regex mode

req_uri_query_string

项目	描述
语法	req_uri_query_string([pattern])
说明	• 如果无pattern参数，则返回请求中的参数部分，不包含问号（？）。 • 如果有pattern参数，则针对请求中的参数部分进行匹配判断。
参数	pattern：使用该参数进行匹配，支持以下两种模式。 • 简单匹配：相等判断，默认为简单匹配。 • 正则匹配： `re:` 前导的正则表达式。

项目	描述
示例	`# req_uri_query_string` `say(concat('req_uri_query_string: ', req_uri_query_string()))` `if req_uri_query_string('mode=') {` `say('check uri query string ok. plain mode')` `} else {` `say('check uri query string fail. plain mode')` `}` `if req_uri_query_string('re:mode=[0-9a-z]+') {` `say('check uri query string ok. regex mode')` `} else {` `say('check uri query string fail. regex mode')` `}`
返回值	• 无pattern参数，返回请求中的参数部分。返回值为字符串类型。 • 有pattern参数，返回 `true` ，表示匹配成功；返回 `false` ，表示匹配失败。 本示例的返回值如下： **请求：** `/path1/path2/path3/path4?mode=req5&token=34Deasd#243` **响应：** `req_uri_query_string: mode=req5&token=34Deasd` `check uri query string fail. plain mode` `check uri query string ok. regex mode`

req_scheme

项目	描述
语法	req_scheme([pattern])
说明	• 如果无pattern参数，则返回请求scheme。 • 如果有pattern参数，则针对请求scheme进行匹配判断。
参数	pattern：使用该参数进行匹配，支持以下两种模式。 • 简单匹配：相等判断，默认为简单匹配。 • 正则匹配： `re:` 前导的正则表达式。

项目	描述
示例	`# req_scheme` `say(concat('req_scheme: ', req_scheme()))` `if req_scheme('https') {` `say('check scheme ok. plain mode')` `} else {` `say('check scheme fail. plain mode')` `}` `if req_scheme('re:https?') {` `say('check scheme ok. regex mode')` `} else {` `say('check scheme fail. regex mode')` `}`
返回值	• 无pattern参数，返回请求scheme。返回值为字符串类型。 • 有pattern参数，返回 `true` ，表示匹配成功；返回 `false` ，表示匹配失败。 本示例的返回值如下： **请求：** `http://xx..` `req_scheme: http` `check scheme fail. plain mode` `check scheme ok. regex mode`

req_method

项目	描述
语法	req_method([pattern])
说明	• 如果无pattern参数，则返回请求method。 • 如果有pattern参数，则针对请求method进行匹配判断。
参数	pattern：使用该参数进行匹配，支持以下两种模式。 • 简单匹配：相等判断，默认为简单匹配。 • 正则匹配： `re:` 前导的正则表达式。

项目	描述
示例	`# req_method` `say(concat('req_method: ', req_method()))` `if req_method('GET') {` `say('check method ok. plain mode')` `} else {` `say('check method fail. plain mode')` `}` `if req_method('re:(GET|POST)') {` `say('check method ok. regex mode')` `} else {` `say('check method fail. regex mode')` `}`
返回值	• 无pattern参数，返回请求method。返回值为字符串类型。 • 有pattern参数，返回 `true` ，表示匹配成功；返回 `false` ，表示匹配失败。 本示例的返回值如下： **请求：** `POST /xxxx/xxx` **响应：** `req_method: POST` `check method fail. plain mode` `check method ok. regex mode`

req_host

项目	描述
语法	req_host([pattern])
说明	• 如果无pattern参数，则返回请求头Host的值。 • 如果有pattern参数，则针对请求头Host的值进行匹配判断。
参数	pattern：使用该参数进行匹配，支持以下两种模式。 • 简单匹配：相等判断，默认为简单匹配。 • 正则匹配： `re:` 前导的正则表达式。

项目	描述
示例	<pre>``` # req_host say(concat('req_host: ', req_host())) if req_host('image.developer.aliyundoc.com') { say('check host ok. plain mode') } else { say('check host fail. plain mode') } if req_host('re:.+\.y\.z\.com') { say('check host ok. regex mode') } else { say('check host fail. regex mode') } ```</pre>
返回值	• 无pattern参数，返回请求头Host的值。返回值为字符串类型。 • 有pattern参数，返回 `true` ，表示匹配成功；返回 `false` ，表示匹配失败。 本示例的返回值如下： **请求：** `Host: image.developer.aliyundoc.com` **响应：** `req_host: image.developer.aliyundoc.com` `check host fail. plain mode` `check host ok. regex mode`

req_user_agent

项目	描述
语法	req_user_agent([pattern])
说明	• 如果无pattern参数，则返回请求头User-Agent的值。 • 如果有pattern参数，则针对请求头User-Agent的值进行匹配判断。
参数	pattern：使用该参数进行匹配，支持以下两种模式。 • 简单匹配：相等判断，默认为简单匹配。 • 正则匹配： `re:` 前导的正则表达式。

项目	描述
示例	`# req_user_agent` `say(concat('req_user_agent: ', req_user_agent()))` `if req_user_agent('Mozilla') {` `  say('check user_agent ok. plain mode')` `} else {` `  say('check user_agent fail. plain mode')` `}` `if req_user_agent('re:^Mozilla') {` `  say('check user_agent ok. regex mode')` `} else {` `  say('check user_agent fail. regex mode')` `}`
返回值	• 无pattern参数，返回请求头User-Agent的值。返回值为字符串类型。 • 有pattern参数，返回 `true` ，表示匹配成功；返回 `false` ，表示匹配失败。 本示例的返回值如下： **请求：** `User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)` **响应：** `req_user_agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)` `check user_agent fail. plain mode` `check user_agent ok. regex mode`

req_referer

项目	描述
语法	req_referer([pattern])
说明	• 如果无pattern参数，则返回请求头Referer的值。 • 如果有pattern参数，则针对请求头Referer的值进行匹配判断。
参数	pattern：使用该参数进行匹配，支持以下两种模式。 • 简单匹配：相等判断，默认为简单匹配。 • 正则匹配： `re:` 前导的正则表达式。

项目	描述
示例	<pre>``` # req_referer say(concat('req_referer: ', req_referer())) if req_referer('https://example.aliyundoc.com/******00003') { say('check referer ok. plain mode') } else { say('check referer fail. plain mode') } if req_referer('re:https://foo\.bar\.cn/*+[0-9]+') { say('check referer ok. regex mode') } else { say('check referer fail. regex mode') } ```</pre>
返回值	• 无pattern参数，返回请求头Referer的值。返回值为字符串类型。 • 有pattern参数，返回 `true` ，表示匹配成功；返回 `false` ，表示匹配失败。 本示例的返回值如下： **请求：** `Referer: https://example.aliyundoc.com/******00003` **响应：** `req_referer: https://example.aliyundoc.com/******00003` `check referer ok. plain mode` `check referer ok. regex mode`

req_cookie

项目	描述
语法	req_cookie(name, [pattern])
说明	使用req_cookie默认返回指定cookie的值，如果有pattern参数，则对指定cookie的值进行匹配判断。
参数	• name：cookie名称。 • pattern：使用该参数进行匹配，支持以下两种模式。 ○ 简单匹配：相等判断，默认为简单匹配。 ○ 正则匹配： `re:` 前导的正则表达式。

项目	描述
示例	`# req_cookie` `uid = req_cookie('uid')` `if uid {` `  say(concat('found cookie uid ', uid))` `} else {` `  say('not found cookie uid')` `}` `uid_chk = req_cookie('uid', '058334')` `if uid_chk {` `  say('check cookie uid ok. plain mode')` `} else {` `  say('check cookie uid fail. plain mode')` `}` `uid_chk = req_cookie('uid', 're:^[0-9]+')` `if uid_chk {` `  say('check cookie uid ok. regex mode')` `} else {` `  say('check cookie uid fail. regex mode')` `}`

项目	描述
返回值	• 如果无pattern参数 ◦ 参数存在：返回name指定cookie的字符串值。 ◦ 参数不存在：返回 `false` 。 • 如果有pattern参数 ◦ 参数存在：进行匹配，返回 `true` ，表示匹配成功；返回 `false` ，表示匹配失败。 ◦ 参数不存在：返回 `false` 。 本示例的返回值如下： **请求：** `Cookie: uid=123456; token=value2` **响应：** `found cookie uid 123456` `check cookie uid fail. plain mode` `check cookie uid ok. regex mode`

req_first_x_forwarded

项目	描述
语法	req_first_x_forwarded
说明	• 如果无pattern参数，则返回请求头X-Forwarded-For中的第一个地址。 • 如果有pattern参数，则针对请求头X-Forwarded-For中的第一个地址进行匹配判断。
参数	pattern：使用该参数进行匹配，支持以下两种模式。 • 简单匹配：相等判断，默认为简单匹配。 • 正则匹配： `re:` 前导的正则表达式。
示例	``` # req_first_x_forwarded say(concat('req_first_x_forwarded: ', req_first_x_forwarded())) if req_first_x_forwarded('1.1.1.1') { say('check first_x_forwarded ok. plain mode') } else { say('check first_x_forwarded fail. plain mode') } if req_first_x_forwarded('re:1.1.1.[0-9]') { say('check first_x_forwarded ok. regex mode') } else { say('check first_x_forwarded fail. regex mode') } ```

项目	描述
返回值	• 无pattern参数，返回请求头X-Forwarded-For中的第一个地址。返回值为字符串类型。 • 有pattern参数，返回 `true` ，表示匹配成功；返回 `false` ，表示匹配失败。 本示例的返回值如下： **请求：** `X-Forwarded-For: 1.1.1.1, 10.10.10.10, 172.16.0.1` **响应：** `req_first_x_forwarded: 1.1.1.1` `check first_x_forwarded ok. plain mode` `check first_x_forwarded ok. regex mode`

req_header

项目	描述
语法	req_header(name, [pattern])
说明	使用req_header默认返回指定请求头的值，如果有pattern参数，则对指定请求头的值进行匹配判断。
参数	• name：请求头名称。 请求头名称中出现的短划线（-），需要使用下划线（_）替代。例如：X-USER-ID对应为x_user_id。 • pattern：使用该参数进行匹配，支持以下两种模式。 ◦ 简单匹配：相等判断，默认为简单匹配。 ◦ 正则匹配： `re:` 前导的正则表达式。
示例	``` # req_header uid = req_header('x_uid') if uid { say(concat('found header x-uid ', uid)) } else { say('not found header x-uid') } uid_chk = req_header('x_uid', 'es developer') if uid_chk { say('check header x-uid ok. plain mode') } else { say('check header x-uid fail. plain mode') } uid_chk = req_header('x_uid', 're:es [a-z]+') if uid_chk { say('check header x-uid ok. regex mode') } else { say('check header x-uid fail. regex mode') } ```

项目	描述
返回值	• 如果无pattern参数 ○ 参数存在：返回name指定请求头的字符串值。 ○ 参数不存在：返回false。 • 如果有pattern参数 ○ 参数存在：进行匹配，返回 `true` ，表示匹配成功；返回 `false` ，表示匹配失败。 ○ 参数不存在：返回 `false` 。 本示例的返回值如下： **请求：**`X-UID: es developer` **响应：** `found header x-uid es developer` `check header x-uid ok. plain mode` `check header x-uid ok. regex mode`

req_id

项目	描述
语法	req_id()
说明	使用req_id获取每个请求唯一的标识（EagleeyeID），用于标识对应的请求。
参数	无
示例	`# req_id` `say(concat('req_id: ', req_id()))`
返回值	返回字符串类型的请求ID。本示例的返回值为 `req_id: 6451c43d15815890089411000e` 。

4.2.7. AScript场景示例

本文为您介绍AScript的防盗链需求、黑白名单、请求头和响应头、改写和重定向和远程鉴权的场景示例。

防盗链需求 | 黑白名单 | 定制化请求头和响应头控制 | 定制化改写和重定向 | 远程鉴权

防盗链需求

自定义鉴权算法场景示例：

- 需求：
  - 请求URL格式： `/path/digest/?.ts?key=&t=` 。

- 针对 `.ts` 类请求，自定义防盗链需求如下：
  - 规则1：参数 `t` 或参数 `key` 不存在，响应403，增加响应头 `X-AUTH-MSG` 标识鉴权失败原因。
  - 规则2：参数 `t` 表示绝对过期时间，若参数 `t` 小于当前时间，则响应403，增加响应头 `X-AUTH-MSG` 标识鉴权失败原因。
  - 规则3：`md5` 与 `digest` 匹配。若 `md5` 与 `digest` 不匹配，响应403。
    md5取值格式为：`私钥 + path + 文件名.后缀` 。

- 对应的AScript规则：

```
if eq(substr($uri, -3, -1), '.ts') {
   if or(not($arg_t), not($arg_key)) {
       add_rsp_header('X-AUTH-MSG', 'auth failed - missing necessary arg')
       exit(403)
   }
   t = tonumber($arg_t)
   if not(t) {
       add_rsp_header('X-AUTH-MSG', 'auth failed - invalid time')
       exit(403)
   }
   if gt(now(), t) {
       add_rsp_header('X-AUTH-MSG', 'auth failed - expired url')
       exit(403)
   }
    pcs = capture($request_uri,'^/([^/]+)/([^/]+)/([^?]+)%?(.*)')
    sec1 = get(pcs, 1)
    sec2 = get(pcs, 2)
    sec3 = get(pcs, 3)
    if or(not(sec1), not(sec2), not(sec3)) {
        add_rsp_header('X-AUTH-MSG', 'auth failed - malformed url')
        exit(403)
    }
    key = 'b98d643a-9170-4937-8524-6c33514bbc23'
    signstr = concat(key, sec1, sec3)
    digest = md5(signstr)
    if ne(digest, sec2) {
        add_rsp_header('X-AUTH-DEBUG', concat('signstr: ', signstr))
        add_rsp_header('X-AUTH-MSG', 'auth failed - invalid digest')
        exit(403)
    }
}
```

User-Agent黑名单场景示例：

- 需求：当请求 `User-Agent` 以 `ijkplayer` 开头、或者以 `Ysten` 开头，则响应403。
- 对应的AScript规则：

```
if and($http_user_agent, match($http_user_agent, '^[ijkplayer|Ysten].*$')) {
    add_rsp_header('X-BLOCKLIST-DEBUG', 'deny')
    exit(403)
}
```

Referer白名单场景示例：

- 需求：当 `referer` 不是 `http[s]://***alibaba.com***` 时，响应403。
- 对应的AScript规则：

```
if and($http_referer, match($http_referer, '^(http|https)://(.)+\.alibaba\.com.*$')) {
    return true
}
add_rsp_header('X-WHITELIST-DEBUG', 'missing')
exit(403)
```

自定义鉴权算法

User-Agent黑名单

Referer白名单

黑白名单

IP黑名单场景示例：

- 需求：当请求客户端IP为 `127.0.0.1` 、或者为 `10.0.0.1` 时，响应403。
- 对应的AScript规则：

```
if match($remote_addr, '[127.0.0.1|10.0.0.1]') {
    add_rsp_header('X-IPBLOCK-DEBUG', 'hit')
    exit(403)
}
```

IP黑名单

区域限制

定制化请求头和响应头控制

文件自动重命名场景示例：

- 需求：当有参数 `filename` 时，自动重命名为 `filename` ，无参数时，使用默认命名。
- 对应的AScript规则：

```
if $arg_filename {
    hn = 'Content-Disposition'
    hv = concat('attachment;filename=', $arg_filename)
    add_rsp_header(hn, hv)
}
```

- 示例：

```
add_rsp_header('Content-Disposition', concat('attachment;filename=', tochar(34), filename
, tochar(34)))
```

> 说明
> - 通过在HTTP应答中添加响应头 `Content-Disposition: attachment` 来实现消息体的强制下载，并且有参数 `filename` 时，自动重命名为 `filename` ，无参数时，使用默认命名。
> - `filename` 增加双引号，双引号的ASCII编码为34，可经tochar转回字符串。

- 输出：

```
Content-Disposition: attachment;filename="monitor.apk"
```

覆盖源站响应头场景示例：

- 需求：覆盖源站 `Content-Type` 响应头。
- 对应的AScript规则：

```
add_rsp_header('Content-Type', 'audio/mpeg')
```

文件自动重命名

覆盖源站响应头

定制化改写和重定向

精确URI改写场景示例：

- 需求：将用户请求 `/hello` 内部改写成 `/index.html` ，回源和缓存的URI都将变成 `/index.html` ，参数保持原样。
- 对应的AScript规则：

```
if match($uri, '^/hello$') {
    rewrite('/index.html', 'break')
}
```

文件后缀改写场景示例：

- 需求：将用户请求 `/1.txt` 内部改写成 `/1.<url参数type>` 。例如： `/1.txt?type=mp4` 将会被改成 `/1.mp4?type=mp4` 回源并缓存。
- 对应的AScript规则：

```
if and(match($uri, '^/1.txt$'), $arg_type) {
     rewrite(concat('/1.', $arg_type), 'break')
}
```

文件后缀小写化场景示例：

- 需求：将URI改成小写。
- 对应的AScript规则：

```
pcs = capture($uri, '^(.+%.)([^.]+)')
section = get(pcs, 1)
postfix = get(pcs, 2)
if and(section, postfix) {
    rewrite(concat(section, lower(postfix)), 'break')
}
```

添加URI前缀场景示例：

- 需求：将用户请求 `^/nn_live/(.*)` 内部改写为 `/3rd/nn_live/$1` 。
- 对应的AScript规则：

```
pcs = capture($uri, '^/nn_live/(.*)')
sec = get(pcs, 1)
if sec {
     dst = concat('/3rd/nn_live/', sec)
     rewrite(dst, 'break')
}
```

302重定向场景示例：

- 需求：将根目录 `/` ，302重定向到 `/app/movie/pages/index/index.html` 页面。
- 对应的AScript规则：

```
if eq($uri, '/') {
    rewrite('/app/movie/pages/index/index.html', 'redirect')
}
```

302重定向HTTPS场景示例：

- 需求：
  将以下URI（对根目录匹配， `^/$` ）
  - `http://rtmp.cdnpe.com`
  - `https://rtmp.cdnpe.com`

  跳转到 `https://aliyun.com` ，跳转后的URI可按需填写。
- 对应的AScript规则：

```
if eq($uri, '/') {
    rewrite('https://aliyun.com', 'redirect')
}
```

精确URI改写

文件后缀改写

文件后缀小写化

添加URI前缀

302重定向

302重定向HTTPS

远程鉴权

自定义远程鉴权场景示例：

- 需求：根据请求发送的数据格式，获得响应结果，决定当前请求是否放过。
- 对应的AScript规则：

```
auth_addr = 'http://xx.cn/outer/cdn/checkplay?host='
```

```
auth_addr = 'http://xx.cn/outer/cdn/checkplay?host='
remote_addr_t = ''
if $remote_addr {
    remote_addr_t = $remote_addr
}
sp = ''
if $arg_sp {
    sp = $arg_sp
}
token = ''
if $arg_token {
    token = $arg_token
}
auth_key = ''
if $arg_auth_key {
    auth_key = $arg_auth_key
}
t_cookie = ''
if $http_cookie {
    t_cookie = $http_cookie
}
referer = ''
if $http_referer {
    referer = $http_referer
}
range = ''
if $http_range {
    range = $http_range
}
auth_addr = concat(auth_addr, $host, '&ip=', remote_addr_t, '&sp=', sp, '&token=', token,
'&auth_key=', auth_key, '&cookie=', t_cookie, '&referer=', referer, '&range=', range)
req_info = []
set(req_info, 'addr', auth_addr)
set(req_info, 'retry', 1)
set(req_info, 'timeout', 1000)
set(req_info, 'method', 'POST')
req_header = []
set(req_header, 'Connection', 'close')
set(req_info, 'send_headers', req_header)
rs = http_request(req_info)
if rs {
    code = get(rs, 'code')
    if eq(code, '403') {
        exit(403)
    }
    headers = get(rs, 'headers')
    if headers {
        x_limit_rate_after = get(headers, 'x-limit-rate-after')
        if and(x_limit_rate_after, match_re(x_limit_rate_after, '\d+(k|m|g)$')) {
            num = substr(x_limit_rate_after, 1, -2)
            unit = substr(x_limit_rate_after, -1, -1)
            limit_rate_after(tonumber(num), unit)
        }
        x_speed = get(headers, 'x-speed')
        if and(x_speed, match_re(x_speed, '\d+(k|m|g)$')) {
```

```
            num = substr(x_speed, 1, -2)
            unit = substr(x_speed, -1, -1)
            limit_rate(tonumber(num), unit)
        }
    }
}
```

自定义远程鉴权

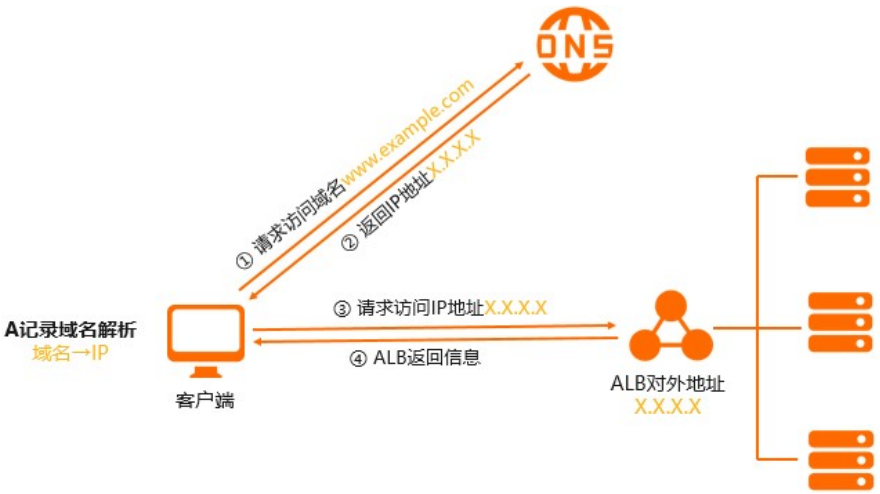
4.3. 设置CNAME域名解析

CNAME域名解析即别名记录，当您需要将域名指向另一个域名，再由另一个域名提供IP地址时，就需要添加CNAME记录。阿里云
应用型负载均衡ALB
支持将您拥有的常用域名通过CNAME方式解析到
ALB
实例的公网服务域名上，使您可以更加方便地访问各种网络资源。

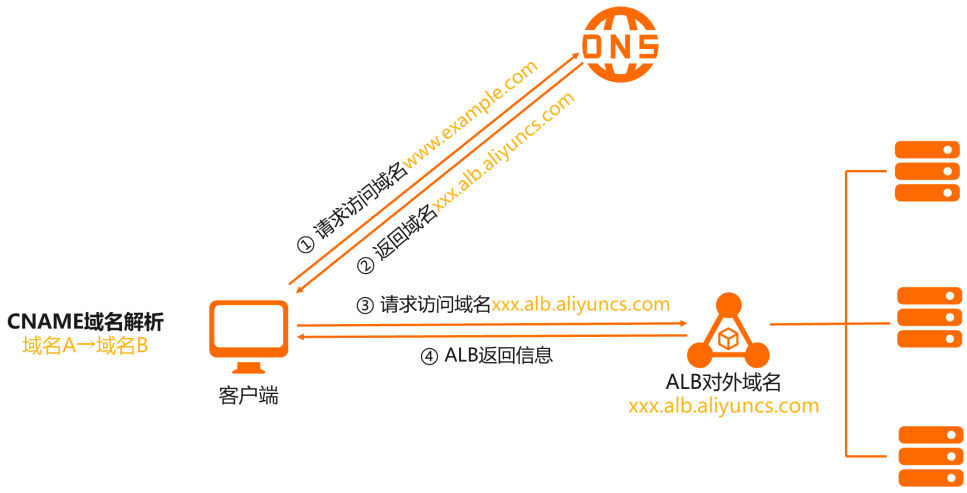
ALB
相关的域名解析方式简介
最常见的两种域名解析方式为A记录域名解析和CNAME域名解析。

ALB
对外提供域名，只支持CNAME域名解析。

- **A记录域名解析**
  A记录域名解析又称IP指向，您可以设置子域名并指向到自己的目标主机IP上，从而实现通过域名找到指定IP。
  应用型负载均衡ALB
  默认对外提供公网IP访问，如需通过域名访问主机，可以配置A记录域名解析，具体实现方案如下图所示：



- **CNAME域名解析**
  CNAME域名解析又称别名解析，您可以设置子域名并指向到其他域名，从而实现将一个域名指向另一个域名。
  应用型负载均衡ALB
  默认对外提供域名访问，如果通过其他域名访问请配置CNAME域名解析，具体实现方案如下图所示：



前提条件

您已创建了

ALB

实例。具体操作，请参见创建应用型负载均衡。

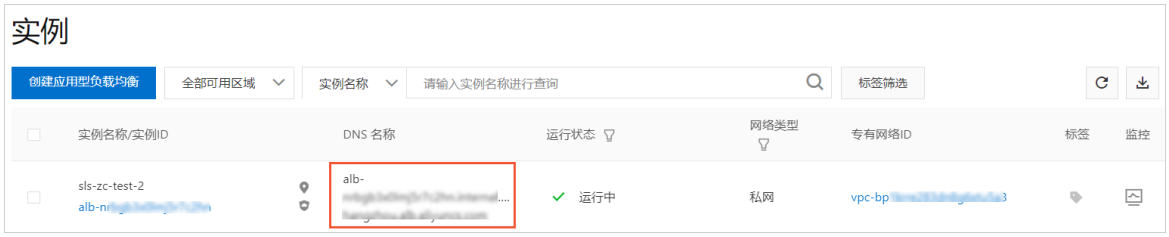
ALB

配置CNAME域名解析

1. 登录应用型负载均衡ALB控制台。
2. 在顶部菜单栏选择地域。
3. 选择要进行域名解析的

   ALB

   实例，复制其对应的DNS名称。



4. 完成以下步骤来添加CNAME解析记录。
   i. 登录域名解析控制台。
   ii. 在**域名解析**页面单击**添加域名**。
   iii. 在**添加域名**对话框中输入您的主机域名，然后单击**确定**。

   > 注意 您的主机域名需已完成TXT记录验证。

   iv. 在目标域名的**操作**列单击**解析设置**。
   v. 在**解析设置**页面单击**添加记录**。

vi. 在**添加记录**面板配置以下信息完成CNAME解析配置，然后单击**确认**。

配置	说明
记录类型	在下拉列表中选择**CNAME**。
主机记录	您的域名的前缀。
解析线路	选择默认。
记录值	输入加速域名对应的CNAME地址，即您复制的ALB实例的DNS名称。
TTL	全称Time To Live，表示DNS记录在DNS服务器上的缓存时间，本文使用默认值。

> **说明**
>
> - 新增CNAME记录实时生效，修改CNAME记录取决于本地DNS缓存的解析记录的TTL到期时间，一般默认为10分钟。
> - 添加时如遇添加冲突，请换一个解析域名。更多信息，请参见解析记录互斥规则。

执行结果

验证CNAME配置是否生效：在命令行中`ping`或`dig`您的自定义域名，如果被转向ALB实例的DNS名称，即表示CNAME配置已生效。

5.ALB开发指南

6.ALB常见问题

本文为您介绍应用型负载均衡ALB（Application Load Balancer）的常见问题。

ALB
是否有具体的实例规格？

ALB
无需选择实例规格，但不同的**IP模式**能力上限不同。

- **固定IP**：每个可用区有且只有一个IP，并且IP地址固定不变。此模式下实例弹性能力有限，最大支持10万业务请求（QPS）。
- **动态IP**：每个可用区至少有一个IP，随着业务请求（QPS）的增加，会自动扩展IP数量。此模式下实例具备良好的弹性能力。

如何修改监听的健康检查配置？

1. 登录应用型负载均衡ALB控制台。
2. 在左侧导航栏，选择**应用型负载均衡ALB > 服务器组**。
3. 在**服务器组**页面，找到目标服务器组，然后单击服务器组ID。
4. 在**详细信息**的健康检查区域，单击**编辑健康检查**。
5. 在**编辑健康检查**对话框，单击**高级配置**右侧的**修改**，根据需求修改健康检查配置。然后单击**保存**。
   更多信息，请参见健康检查。

ALB
的公网流量能否使用共享流量包进行抵扣？

ALB
通过绑定弹性公网IP（EIP）提供公网能力，当您的ALB实例未绑定共享带宽时，EIP产生的公网流量可以使用共享流量包进行抵扣。

ALB
是否支持CA双向认证？
基础版

ALB
实例不支持CA双向认证，标准版
ALB
实例添加HTTPS监听时支持配置CA双向认证。如果基础版
ALB
需要使用CA双向认证功能，请升级实例功能版本。具体操作，请参见实例变配和添加HTTPS监听。
在使用CA双向认证功能时，您需要选择一个CA证书或在CA证书下拉列表中单击购买CA证书购买新证书。更多信息，请参见创建私有CA和启用私有CA。

私网ALB支持绑定弹性公网IP吗？

支持。

如果您需要给私网

ALB
绑定弹性公网IP，可以通过变更实例网络类型的方式，将私网
ALB
变更为公网
ALB
。具体操作，请参考变更ALB实例的网络类型。
私网类型转换为公网类型时会绑定弹性公网IP，产生公网网络费用。更多信息，请参见弹性公网IP计费。

ALB控制台可以上传证书吗？

不可以。

ALB
是直接调用SSL证书服务（Alibaba Cloud SSL Certificates Service）的证书，需要您将证书上传至SSL证书控制台，因此不支持在
ALB
控制台进行证书上传操作。更多信息，请参见SSL上传证书。

ALB是否支持流量镜像功能？

支持。目前流量镜像功能白名单开放，如需体验请提交工单。更多信息，请参见使用ALB流量镜像功能实现仿真压测。

ALB如何提升带宽？

ALB
实例未加入共享带宽时，公网默认采用按流量计费模式，此时实例的最大带宽峰值为400 Mbps。按流量计费的实例不支持调整带宽，如果您需要调整按流量计费实例的带宽峰值，可以将实例加入共享带宽，然后调整共享带宽的带宽峰值。更多信息，请参见实例变配。

7.联系我们

如果您在使用ALB过程中有任何疑问和建议，请打开钉钉搜索群号加入钉钉群与我们联系。
群名：ALB客户交流群

群号：31945843