ALIBABA CLOUD

阿里云

CDN

边缘脚本

文档版本：20201109

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.简介

1.1. EdgeScript概述

EdgeScript即边缘脚本，用于支持CDN的可编程配置化。通过EdgeScript，您可以快速支持CDN的定制化业务需求，解决定制化需求发布周期长、客户业务变更不敏捷等问题。通过本文您可以了解EdgeScript的概念、应用场景和在CDN控制台上的配置方法。

什么是EdgeScript?

EdgeScript是为CDN设计的专用脚本，具有强大领域操控能力的同时，仍然保持简单易学的语法。通过EdgeScript，您可以快速构建基于阿里云CDN的个性化业务体系，全网秒级生效，敏捷的业务迭代会持续地为您赢得交付收益。

应用场景

EdgeScript适用场景如下：

- 定制化鉴权逻辑
- 定制化请求头&响应头控制
- 定制化改写&重定向
- 定制化A/B Testing
- 定制化缓存控制
- 定制化限速
- 其它定制化业务

边缘脚本控制台使用流程

通过CDN控制台配置EdgeScript

1.2. 原理介绍

本文为您介绍EdgeScript的规则模型、执行位置与优先级、命中与中断执行，以及规则字段说明。

规则模型

EdgeScript的规则模型如下：

- EdgeScript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。
- EdgeScript的规则模型中的每条规则的地位都是平等的，可以各自选择规则的执行位置（即介入请求处理的位置）和优先级。
- EdgeScript的规则模型具有域名粒度。

执行位置与优先级

EdgeScript的执行位置与优先级如下：

- 执行位置

  EdgeScript规则的执行位置为请求处理开始和请求处理结尾。

  - 请求处理开始：常用应用场景为一次鉴权、拦截、限速等。
  - 请求处理结束：常用应用场景为缓存设置、回源鉴权、A/B Testing等。

- 优先级

  处于同一执行位置的多条规则，可以通过选择优先级，决定其先后执行顺序。

命中与中断执行

EdgeScript的命中与中断执行如下：

- EdgeScript规则的命中定义
  - 当规则结尾是以 `return true` 返回时：规则模型认为该规则命中。
  - 当规则结尾是以 `return false` 返回时：规则模型认为该规则未命中。
- EdgeScript规则的中断执行

  对于命中规则，可以选择终止本阶段剩余规则的执行。

规则字段说明

一条EdgeScript规则，可配置字段如下表所示。

字段名称	是否必选	字段含义	字段说明
enable	Y	是否生效。	取值范围： • on • off
pos	Y	执行位置。	取值范围： • head • foot
pri	Y	执行优先级。	取值范围：0~999。0表示优先级最高，999表示优先级最低。不同执行位置的优先级各自独立。
rule	Y	规则内容。	-
brk	N	本规则命中情况下，是否终止本阶段剩余规则的执行。	取值范围： • on • off
testip	N	测试客户IP。	默认为空。如果配置了该参数，则只有客户端IP地址可触发本条规则执行。
option	N	扩展项。	当前支持扩展。`_es_dbg=signature` 用于增加调试响应头。

1.3. 快速入门

本文通过样例为您介绍EdgeScript规则的编写、保存、测试和发布的基本操作方法。

EdgeScript的基本操作如下：

- 将编写的EdgeScript规则存储为本地文件

  例如：m3u8.es规则拦截所有.m3u8请求

```
$cat m3u8.es
if eq(substr($uri, -5, -1), '.m3u8') {
  add_rsp_header('X-DEBUG-DENY-REASON', 'block m3u8')
  exit(400)
}
```

- 发布规则至模拟环境

```
$./es.py action=push_test_env domain=<your domain> rule='{"pos":"head","pri":"0","rule_path":"./m3u8.es","enable":"on"}'

Response Code:
=============
200 OK

Response Info:
==============
{
  "RequestId": "FB98CC67-8FBA-44CF-A98A-BCE3B19FE510"
}
```

- 查看模拟环境的规则列表

```
$./es.py action=query_test_env domain=<your domain>
Response Code:
=============
200 OK

Response Info:
==============
{
  "DomainConfigs": [
    {
      "Status": "success",  # Status为success时，表示规则已成功设置
      "ConfigId": 17432558,
      "FunctionArgs": [
        {
          "ArgName": "enable",
          "ArgValue": "on"
        },
        {
          "ArgName": "pri",
          "ArgValue": "0"
        },
        {
          "ArgName": "pos",
          "ArgValue": "head"
        },
        {
          "ArgName": "rule",
          "ArgValue": "if eq(substr($uri, -5, -1), '.m3u8') {\n    add_rsp_header('X-DEBUG-DENY-REASON', 'bl
ock m3u8')\n    exit(400)\n}\n"
        }
      ],
      "FunctionName": "dsl_ex"
    }
  ],
  "RequestId": "4DDBF3DB-BCAC-4074-AC1E-B6C1F1C6CBFB"
}
```

- 测试规则

```
$curl -x 模拟环境IP:80 -o /dev/null -v 'http://www.archnote.net/test.m3u8'
< HTTP/1.1 400 Bad Request
< Server: Tengine
< Date: Thu, 18 Jul 2019 09:40:41 GMT
< Content-Type: text/html
< Content-Length: 265
< Connection: close
< X-DEBUG-DENY-REASON: block m3u8
< Via: cache1.cn1191-1[,0]
< Timing-Allow-Origin: *
< EagleId: 2a7b771b15634428415537484e
```

- 发布规则至生产环境

```
$./es.py action=push_product_env domain=<your domain>
Response Code:
=============
200 OK

Response Info:
==============
{
  "RequestId": "F4B378F8-6AAE-457A-A70C-E856ED8341D8"
}
```

- 查看生产环境的规则列表

```
$./es.py action=query_product_env domain=<your domain>
Response Code:
=============
200 OK

Response Info:
==============
{
  "DomainConfigs": {
    "DomainConfig": [
      {
        "Status": "success",
        "ConfigId": 17432558,
        "FunctionArgs": {
          "FunctionArg": [
            {
              "ArgName": "enable",
              "ArgValue": "on"
            },
            {
              "ArgName": "pri",
              "ArgValue": "0"
            },
            {
              "ArgName": "pos",
              "ArgValue": "head"
            },
            {
              "ArgName": "rule",
              "ArgValue": "if eq(substr($uri, -5, -1), '.m3u8') {\n    add_rsp_header('X-DEBUG-DENY-REASON',
'block m3u8')\n    exit(400)\n}\n"
            }
          ]
        },
        "FunctionName": "dsl_ex"
      }
    ]
  },
  "RequestId": "36D57C1D-C820-43DA-8E70-DADC4B8BD4DD"
}
```

1.4. 通过CDN控制台配置EdgeScript

边缘脚本用于支持CDN的可编程配置化。通过边缘脚本，您可以快速实现CDN的定制化业务需求。在CDN控制台上，您也可以根据边缘脚本定义的代码规则，创建边缘脚本规则，并发布到生产环境，实现对CDN产品的定制化管理。本文为您介绍了在CDN控制台上配置边缘脚本操作方法。

背景信息

EdgeScript是为CDN设计的专用脚本，具有强大领域操控能力的同时，仍然保持简单易学的语法。通过EdgeScript，您可以快速构建基于阿里云CDN的个性化业务体系，全网秒级生效，敏捷的业务迭代会持续地为您赢得交付收益。

在CDN控制台上配置边缘脚本规则的流程，如下图所示。

操作步骤

1. 登录CDN控制台。
2. 在左侧导航栏，单击**域名管理**
3. 在**域名管理**页面，单击域名右侧的**管理**。
4. 单击**边缘脚本**。
5. 添加规则到模拟环境。
   i. 在**模拟环境**页面，单击**添加规则**。

      > **说明** 目前单个域名仅支持添加一条边缘脚本规则。如果您需要添加多条规则，请提交工单申请。

      

   ii. 根据界面提示，配置规则。

      

      边缘脚本参数配置规则，详情请参见规则字段说明。

      您可以按照使用场景编写规则代码，详情请参见EdgeScript场景示例。

   iii. 单击**发布到模拟环境**完成添加。
6. 在模拟环境中，测试规则。

   模拟环境测试IP地址（该IP地址请以实际页面显示为准），如下图红框所示。

   

   在客户端路径*C:\Windows\System32\drivers\etc*中找到文件**hosts**。将模拟环境测试IP地址添加到**hosts**文件中。
7. 测试完成后，单击**发布所有规则到生产环境**，将模拟环境规则发布至生产环境。

   > **注意** 模拟环境规则发布到生产环境后，模拟环境的规则自动被清空。

   
8. 如果您需要基于最新发布的规则进行增改，则需要将发布到生产环境的规则同步到模拟环境后再执行操

作。 单击**从生产环境复制规则**，将发布到生产环境的规则同步到模拟环境。

生产环境的规则同步到模拟环境后，即可在模拟环境进行修改或新增规则，如下图所示。

1.5. 通过CLI工具配置EdgeScript

本文为您介绍EdgeScript CLI工具的用途和使用说明。

用途

使用EdgeScript CLI工具，用户可以执行如下操作：

- 将本地编写的EdgeScript规则发布到模拟环境进行测试。
- 将模拟环境的EdgeScript规则发布至全网（生产环境）或回滚（模拟环境）。
- 对模拟环境和生产环境的EdgeScript规则进行查询、修改和删除。

工具下载

您可以单击附件，下载工具。

使用说明

EdgeScript CLI的使用说明如下：

- 配置AK

```
$python ./es.py config --id=AK_ID --secret=AK_SECRET
$cat aliyun.ini
[Credentials]
accesskeyid = 更新后AK
accesskeysecret = 更新后AK Secret
```

- 发布EdgeScript规则至模拟环境或生产环境

```
./es.py action=push_test_env domain=<domain> rule='{"pos":"<head|foot>","pri":"0-999","rule_path":"<the es code path>","enable":"<on|off>"}'
./es.py action=push_product_env domain=<domain> rule='{"pos":"<head|foot>","pri":"0-999","rule_path":"<the es code path>","enable":"<on|off>","configid":"<configid>"}'
```

> **说明**
>
> - 新增规则：不需要指定configid。
> - 修改规则：需要指定configid; 使用查询接口可获取到对应规则的configid。
> - 您可以指定多条rule。

- 查询模拟环境或生产环境下的EdgeScript规则

```
./es.py action=query_test_env domain=<domain>
./es.py action=query_product_env domain=<domain>
```

- 删除模拟环境或生产环境下的EdgeScript规则

```
./es.py action=del_test_env domain=<domain> configid=<configid>
./es.py action=del_product_env domain=<domain> configid=<configid>
```

> **说明** 使用查询接口可获取到对应规则的configid。

- 模拟环境的EdgeScript规则执行正式发布或回滚

```
./es.py action=publish_test_env domain=<domain>
./es.py action=rollback_test_env domain=<domain>
```

实时调试方式

1. 配置实时调试。

   您可以通过控制台的WebIDE图形化操作页面进行_es_dbg配置，也可以使用如下命令进行配置。

   ```
   ./es.py action=push_test_env domain=<domain> rule='{"pos":"<head|foot>","pri":"0-999","rule_path"
   :"<the es code path>","enable":"<on|off>","configid":"<configid>", "option":"_es_dbg=123"}'
   ```

2. 测试请求。

   测试时请求头携带_es_dbg参数，参数值为配置的option中_es_dbg值，关注如下响应头信息。

   Trace信息：`X-DEBUG-ES-TRACE-RULE-{规则ID}`，查看对应规则的执行流，格式为`_行号_函数名(入参):返回值{_执行耗时}`。

2.EdgeScript语法

本文为您介绍EdgeScript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。

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

标识符规则如下：

- 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

数据类型规则如下：

- 字符串

  字面常量：使用单引号括起来，例如：`'hello, EdgeScript'`。

- 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

变量规则如下：

- 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。

运算符

运算符规则如下：

- = ： 赋值运算符
  - 例如： `seckey = 'ASDLFJ234dxvf34sDF'`
  - 例如： `seckeys = ['key1', 'key2']`
- - ： 负号运算符

  例如： `inum = -10`
- 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
    - `sval = concat(sval, 'trail')`
    - `len(arrvar)`

语句

语句规则如下：

- 条件判断语句

```
if condition {
  ...
}

if condition1 {
  if conditon2 {
     ...
  }
}

if condition {
  ...
} else {
  ...
}
```

- 语句解释
  - `condition` 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 `if condition` 之后，且同行。

函数

函数语法和定义说明如下：

- 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

EdgeScript全文不允许出现任何双引号。

3.EdgeScript内置变量表

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

EdgeScript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。	$arg_ 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。	$http_ 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 。
$cookie_{name}	请求cookie头中的name值。	$cookie_ 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。	$scheme
$server_protocol	协议版本。	$server_protocol
$host	原始host。	$host
$uri	原始URI。	-

内置变量名	含义	对应nginx原生变量
$args	`$args` 表示当前HTTP请求的全部请求参数，但不包含 `?` 。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 `?` 。	$args
$request_method	请求方法。	$request_method
$request_uri	*uri*+'?'+args 的内容。	$request_uri
$remote_addr	客户的IP地址。	$remote_addr

说明

- 内置变量名前的 `$` ，仅为强调其内置变量属性，去掉不影响使用。
- 内置变量不允许担当左值，即不允许被赋值。

4.EdgeScript内置函数库

4.1. 概述

本文为您介绍EdgeScript内置函数的分类和包括的函数。

EdgeScript内置函数分类和包括的函数如下表所示。

函数分类	函数
条件判断相关	条件判断相关函数包括：and、or、not、eq、ne。
数字类型相关	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	字符串类型相关函数包括：substr、concat、upper、lower、len、byte、match、re、capture、re、gsub、re、split、split、as、key、tohex、tostring、tochar。
字典类型相关	字典类型相关函数包括：set、get、foreach。
请求处理相关	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。
限速相关	限速相关函数包括：limit_rate_after、limit_rate。
缓存相关	缓存相关函数为：set_cache_ttl。
时间相关	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	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	JSON相关函数包括：json_enc、json_dec。
Misc相关	Misc相关函数包括：base64_enc、base64_dec、url_escape、url_unescape、rand、rand_hit、rand_bytes、crc、tonumber。

4.2. 条件判断相关

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

函数详细解释如下：

- 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。

- 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')
}

说明:
a. 当请求携带mode参数、并且mode参数等于set_header时，设置响应头USER-DEFINED-1
b. 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

函数详细解释如下：

- 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` ，全部参数为假返时 `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')
}

说明:
a. 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]
b. 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

函数详细解释如下：

- 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。

- 返回值
  - true
  - false

- 示例

```
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
```

eq

函数详细解释如下：

- 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`。

- 示例

```
key1 = 'value1'
key2 = 'value2'
if and($arg_k1, $arg_k2, eq(key1, $arg_k1), ne(key2, $arg_k2)) {
  say('match condition')
}

说明:
a. 请求参数k1存在，且请求参数k2存在时，执行后续的比较操作;
b. 请求参数k1 or k2不存在时，短路语义生效，不再执行后续的比较操作;
c. eq: 请求参数k1的值是否等于value1;
d. ne: 请求参数k2的值不等于value2;
e. 当请求参数k1/k2均存在，且k1等于value1，且k2不等于value2时，输出响应体match condition
```

ne

函数详细解释如下：

- 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')
}

说明:
a. 请求参数k1存在，且请求参数k2存在时，执行后续的比较操作;
b. 请求参数k1 or k2不存在时，短路语义生效，不再执行后续的比较操作;
c. eq: 请求参数k1的值是否等于value1;
d. ne: 请求参数k2的值不等于value2;
e. 当请求参数k1/k2均存在，且k1等于value1，且k2不等于value2时，输出响应体match conditionkey1 = 'value
1'
key2 = 'value2'
if and($arg_k1, $arg_k2, eq(key1, $arg_k1), ne(key2, $arg_k2)) {
  say('match condition')
}

说明:
a. 请求参数k1存在，且请求参数k2存在时，执行后续的比较操作;
b. 请求参数k1 or k2不存在时，短路语义生效，不再执行后续的比较操作;
c. eq: 请求参数k1的值是否等于value1;
d. ne: 请求参数k2的值不等于value2;
e. 当请求参数k1/k2均存在，且k1等于value1，且k2不等于value2时，输出响应体match condition
```

4.3. 数字类型相关

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

函数详细解释如下：

- 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。

- 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

函数详细解释如下：

- 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。
- 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=15n1 = 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

函数详细解释如下:

- 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。
- 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=15n1 = 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

函数详细解释如下：

- 语法：`div(n1, n2)`。
- 说明

  除法操作。
- 参数
  - n1：数字类型。
  - n2：数字类型。
- 返回值

  返回 `n1/n2` 的结果。
- 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=15n1 = 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

函数详细解释如下:

- 语法: `mod(n1, n2)` 。
- 说明

  求余操作。

- 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))

输出：
n1=30
n2=-10
n3=200
n4=0.5
n5=15n1 = 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

函数详细解释如下：

- 语法：`gt(n1, n2)` 。
- 说明

  大于比较。
- 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` 。
- 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

函数详细解释如下：

- 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` 。

- 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 <= 10if 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

函数详细解释如下：

- 语法： `lt(n1, n2)` 。
- 说明

小于比较。

- 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` 。
- 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 <= 10if 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

函数详细解释如下：

- 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`。
- 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 <= 10if 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

函数详细解释如下：

- 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

函数详细解释如下：

- 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))))
}

请求：/path1/path2/file?num=9.3 响应：ceil: 10  floor: 9if $arg_num {
  say(concat('ceil: ', ceil(tonumber($arg_num))))
  say(concat('floor: ', floor(tonumber($arg_num))))
}

请求：/path1/path2/file?num=9.3 响应：ceil: 10  floor: 9
```

4.4. 字符串类型相关

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

函数详细解释如下：

- 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]` 。
- 示例

```
//说明：判断文件类型是否为.m3u8的2种方法
if eq(substr($uri, -5, -1), '.m3u8') {
  say(concat($uri, ' is .m3u8'))
}

uri_len = len($uri)
if eq(substr($uri, -5, uri_len), '.m3u8') {
  say(concat($uri, ' is .m3u8'))
}
```

concat

函数详细解释如下：

- 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

```
//说明：判断文件类型是否为.m3u8的2种方法
if eq(substr($uri, -5, -1), '.m3u8') {
  say(concat($uri, ' is .m3u8'))
}

uri_len = len($uri)
if eq(substr($uri, -5, uri_len), '.m3u8') {
  say(concat($uri, ' is .m3u8'))
}//说明：判断文件类型是否为.m3u8的2种方法
if eq(substr($uri, -5, -1), '.m3u8') {
  say(concat($uri, ' is .m3u8'))
}

uri_len = len($uri)
if eq(substr($uri, -5, uri_len), '.m3u8') {
  say(concat($uri, ' is .m3u8'))
}
```

upper

函数详细解释如下：

- 语法：`upper(s)`。
- 说明

  大写操作。
- 参数

  s：目标字符串。
- 返回值

  返回大写 `s`。
- 示例

```
//输出：
//HELLO,DSL
//hello, dsl

mystr = 'Hello, Dsl'
say(upper(mystr))
say(lower(mystr))
```

lower

函数详细解释如下：

- 语法：`lower(s)`。
- 说明

  小写操作。
- 参数

  s：目标字符串。
- 返回值

  返回小写 `s`。
- 示例

```
//输出：
//HELLO,DSL
//hello, dsl

mystr = 'Hello, Dsl'
say(upper(mystr))
say(lower(mystr)) //输出：
//HELLO,DSL
//hello, dsl

mystr = 'Hello, Dsl'
say(upper(mystr))
say(lower(mystr))
```

len

函数详细解释如下：

- 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。
- 示例

```
//说明：判断文件类型是否为.m3u8的2种方法
if eq(substr($uri, -5, -1), '.m3u8') {
  say(concat($uri, ' is .m3u8'))
}

uri_len = len($uri)
if eq(substr($uri, -5, uri_len), '.m3u8') {
  say(concat($uri, ' is .m3u8'))
}
```

byte

函数详细解释如下：

- 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

```
//输出：97
//65

say(byte('a'))
say(byte('A'))
```

match_re

函数详细解释如下：

- 语法：`match_re(s, p [, o])` 。
- 说明

  使用PCRE正则引擎，进行正则匹配。
- 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。
  - i：大小写不敏感。
- 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

函数详细解释如下：

- 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。
- 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。
- 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('X-TENGINE-ERROR', 'auth failed - missing necessary uri set')
  exit(403)
}
digest = md5(concat(sec1, sec3))
if ne(digest, sec2) {
   add_rsp_header('X-TENGINE-ERROR', 'auth failed - invalid digest')
   exit(403)
}
```

gsub_re

函数详细解释如下：

- 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。
- 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。
- 示例

```
//输出:
//X-DEBUG-GSUB-RE: [Hello,H], [Es,E]

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

- 参考

  PCRE正则语法：pcre2syntax man page。

split

函数详细解释如下：

- 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。
- 示例

```
//请求：?from=xx1,xx2,xx3
//响应：[1]=xx1
//[2]=xx1

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)))
  }
}
```

split_as_key

函数详细解释如下：

- 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]* 。
- 示例

```
//请求：?from=xx1,xx2,xx3
//响应：xx2=xx2 u=hi,dsl
//xx1=xx1 u=hi,dsl
//xx3=xx3 u=hi,dsl

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,dsl'])
}
```

tohex

函数详细解释如下：

- 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

```
//说明：增加响应头
//X-DSL-TOHEX: 4ad583af22c2e7d40c1c916b2920299155a46464

digest = sha1('xxxx')
add_rsp_header('X-DSL-TOHEX', tohex(digest))
```

tostring

函数详细解释如下：

- 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

```
//说明：增加响应头
//X-DSL-TOSTRING: 123

s = tostring(123)
add_rsp_header('X-DSL-TOSTRING', s)
```

tochar

函数详细解释如下：

- 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。
- 示例

```
//输出：增加响应头
//X-DSL-TOCHAR: a
//X-DSL-TOCHAR: ab

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
//Content-Disposition: attachment;filename="请求参数filename的值"

if $arg_filename {
  hn = 'Content-Disposition'
  add_rsp_header('Content-Disposition', concat('attachment;filename=', tochar(34), filename, tochar(34)
))
  add_rsp_header(hn, hv)
}
```

4.5. 字典类型相关

本文为您介绍字典类型相关函数的语法、说明、参数、返回值和示例。

set

函数详细解释如下：

- 语法：`set(d, k, v)`。
- 说明

  在字典`d`中设置`k/v`。
- 参数
  - d：目标字典。
  - k：key值，任意类型。
  - v：value值，任意类型。
- 返回值

  返回`true`。
- 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 dsl')

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-DSL-NESTED-DICT', v)
  }
}

输出：增加响应头
X-DSL-NESTED-DICT: inner dsl
```

get

函数详细解释如下：

- 语法：`get(d, k)` 。
- 说明

  获取字典 `d` 中 `k` 对应的 `v` 。
- 参数
  - d：目标字典。
  - k：key值，任意类型。
- 返回值

  成功返回对应值，失败返回 `false` 。
- 示例

```
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
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
```

foreach

函数详细解释如下：

- 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，字典类型。
- 返回值

  返回 `true`。
- 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
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。

```
def echo_each(k, v, u) {
  say(v)

  if match_re(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_140651513.ts\n')
m3u8 = concat(m3u8, '#EXTINF:9,\n')
m3u8 = concat(m3u8, 'http://vapp1.fw.live.cntv.cn/cache/289_/seg0/index140651514_140651514.ts\n')
m3u8 = concat(m3u8, '#EXTINF:10,\n')
m3u8 = concat(m3u8, 'http://vapp1.fw.live.cntv.cn/cache/289_/seg0/index140651514_140651515.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_140651513.ts
#EXTINF:9,
http://vapp1.fw.live.cntv.cn/cache/289_/seg0/index140651514_140651514.ts
```

4.6. 请求处理相关

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

函数详细解释如下：

- 语法：`add_req_header(name, value [, append])`。
- 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_req_header('USER-DEFINED-REQ-1', '1')
add_req_header('USER-DEFINED-REQ-1', 'x', true)
add_req_header('USER-DEFINED-REQ-2', '2')
del_req_header('USER-DEFINED-REQ-2')

说明：添加2个请求头，分别为
USER-DEFINED-REQ-1: 1
USER-DEFINED-REQ-1: x

USER-DEFINED-REQ-2先添加、后删除，故回源请求头中无USER-DEFINED-REQ-2
```

del_req_header

函数详细解释如下：

- 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`。

- 示例

```
add_req_header('USER-DEFINED-REQ-1', '1')
add_req_header('USER-DEFINED-REQ-1', 'x', true)
add_req_header('USER-DEFINED-REQ-2', '2')
del_req_header('USER-DEFINED-REQ-2')

说明：添加2个请求头，分别为
USER-DEFINED-REQ-1: 1
USER-DEFINED-REQ-1: x

USER-DEFINED-REQ-2先添加、后删除，故回源请求头中无USER-DEFINED-REQ-2add_req_header('USER-DEFI
NED-REQ-1', '1')
add_req_header('USER-DEFINED-REQ-1', 'x', true)
add_req_header('USER-DEFINED-REQ-2', '2')
del_req_header('USER-DEFINED-REQ-2')

说明：添加2个请求头，分别为
USER-DEFINED-REQ-1: 1
USER-DEFINED-REQ-1: x

USER-DEFINED-REQ-2先添加、后删除，故回源请求头中无USER-DEFINED-REQ-2
```

add_rsp_header

函数详细解释如下：

- 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` 。

- 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')

说明：添加2个响应头，分别为
USER-DEFINED-RSP-1：1
USER-DEFINED-RSP-1：x

USER-DEFINED-RSP-2先添加、后删除，故响应头中无USER-DEFINED-RSP-2
```

del_rsp_header

函数详细解释如下：

- 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')

说明：添加2个响应头，分别为
USER-DEFINED-RSP-1：1
USER-DEFINED-RSP-1：x

USER-DEFINED-RSP-2先添加、后删除，故响应头中无USER-DEFINED-RSP-2add_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')

说明：添加2个响应头，分别为
USER-DEFINED-RSP-1：1
USER-DEFINED-RSP-1：x

USER-DEFINED-RSP-2先添加、后删除，故响应头中无USER-DEFINED-RSP-2
```

encode_args

函数详细解释如下：

- 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-DSL-ENCODE-ARGS', my_args_str)

to_args = decode_args(my_args_str)
if get(to_args, 'algo') {
  add_rsp_header('X-DSL-DECODE-ARGS-ALGO', get(to_args, 'algo'))
}
if get(to_args, 'signature') {
  add_rsp_header('X-DSL-DECODE-ARGS-SIGN', get(to_args, 'signature'))
}

输出：添加3个响应头，分别为
X-DSL-ENCODE-ARGS: signature=da9dc4b7-87ae-4330-aaaf-e5454e2c2af1&algo=private%20sign1
X-DSL-DECODE-ARGS-ALGO: private sign1
X-DSL-DECODE-ARGS-SIGN: da9dc4b7-87ae-4330-aaaf-e5454e2c2af1
```

decode_args

函数详细解释如下：

- 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。
- 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-DSL-ENCODE-ARGS', my_args_str)

to_args = decode_args(my_args_str)
if get(to_args, 'algo') {
  add_rsp_header('X-DSL-DECODE-ARGS-ALGO', get(to_args, 'algo'))
}
if get(to_args, 'signature') {
  add_rsp_header('X-DSL-DECODE-ARGS-SIGN', get(to_args, 'signature'))
}

输出：添加3个响应头，分别为
X-DSL-ENCODE-ARGS: signature=da9dc4b7-87ae-4330-aaaf-e5454e2c2af1&algo=private%20sign1
X-DSL-DECODE-ARGS-ALGO: private sign1
X-DSL-DECODE-ARGS-SIGN: da9dc4b7-87ae-4330-aaaf-e5454e2c2af1my_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-DSL-ENCODE-ARGS', my_args_str)

to_args = decode_args(my_args_str)
if get(to_args, 'algo') {
  add_rsp_header('X-DSL-DECODE-ARGS-ALGO', get(to_args, 'algo'))
}
if get(to_args, 'signature') {
  add_rsp_header('X-DSL-DECODE-ARGS-SIGN', get(to_args, 'signature'))
}

输出：添加3个响应头，分别为
X-DSL-ENCODE-ARGS: signature=da9dc4b7-87ae-4330-aaaf-e5454e2c2af1&algo=private%20sign1
X-DSL-DECODE-ARGS-ALGO: private sign1
X-DSL-DECODE-ARGS-SIGN: da9dc4b7-87ae-4330-aaaf-e5454e2c2af1
```

rewrite

函数详细解释如下：

- 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。
- 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

函数详细解释如下：

- 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

函数详细解释如下：

- 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
byebyebyebyesay('hello')
print('byebye')
print('byebye')

输出:
hello
byebyebyebye
```

exit

函数详细解释如下：

- 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。
- 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('X-TENGINE-ERROR', 'auth failed - missing necessary uri set')
  exit(403)
}
digest = md5(concat(sec1, sec3))
if ne(digest, sec2) {
   add_rsp_header('X-TENGINE-ERROR', 'auth failed - invalid digest')
   exit(403)
}
```

4.7. 限速相关

本文为您介绍限速相关函数的语法、说明、参数、返回值和示例。

limit_rate_after

函数详细解释如下：

- 语法：`limit_rate_after(n, unit)`。
- 说明

  设置不限速大小。
- 参数
  - n：整型。如果不配置`unit`，则默认单位为Byte。
  - （可选）unit：字符类型。如果配置`unit`，则可用值为`K`、`M`或`G`，不区分大小写。
- 返回值

  设置成功返回`true`，设置失败返回`false`。
- 示例

```
limit_rate_after(10, 'k')
limit_rate(1, 'm')
//说明：设置前10KB/s不限速，之后限速为1MB/s
```

limit_rate

函数详细解释如下：

- 语法：`limit_rate(n, unit)`。
- 说明

设置限速值。限速值默认下限为4KB/s，当您设置的限速值低于4KB/s时，默认生效4KB/s。

- 参数
  - n：整型。如果不配置 `unit` ，则默认单位为Byte。
  - （可选）unit：字符类型。如果配置 `unit` ，则可用值为 `K` 、 `M` 或 `G` ，不区分大小写。
- 返回值

  成功设置返回 `true` ，失败设置返回 `false` 。
- 示例

```
limit_rate_after(10, 'k')
limit_rate(1, 'm')
//说明：设置前10KB/s不限速，之后限速为1MB/s
```

4.8. 缓存相关

本文为您介绍缓存相关函数的语法、说明、参数、返回值和示例。

set_cache_ttl

函数详细解释如下：

- 语法： `set_cache_ttl(type, ttl)` 。
- 说明

  设置资源缓存时长。
- 参数
  - type

    缓存类型，可用值 `'path|code'` ，字符类型。
  - ttl
    - 当type=path时，表明针对某类资源设置缓存时间，则 `ttl` 为缓存时长，要求数字类型。
    - 当type=code时，表明针对响应码设置缓存时间，则为 `ttl` 各响应码的缓存时长，要求字符类型。
    - 缓存时长默认单位：秒。
- 返回值

  设置成功返回 `true` ，设置失败返回 `false` 。
- 示例

```
if match_re($uri, '^/image') {
  set_cache_ttl('code', '301=10,302=5')
}
说明：/image开头的uri，针对响应码设置缓存时长，301缓存10s，302缓存5s

if eq(substr($uri, -4, -1), '.mp4') {
  set_cache_ttl('path', 5)
}
if match_re($uri, '^/201801/mp4/') {
  set_cache_ttl('path', 50)
}
if match_re($uri, '^/201802/flv/') {
  set_cache_ttl('path', 10)
}
说明：针对文件名、uri路径，设置各种不同的缓存时长
```

4.9. 时间相关

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

函数详细解释如下：

- 语法：`today()` 。
- 说明

  返回当前时间字符串，格式：yyyy-mm-dd。
- 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:2019-05-23
```

time

函数详细解释如下：

- 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

函数详细解释如下：

- 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。

- 示例

```
say(concat('now:', now()))

输出:
now:1559109666.644
```

localtime

函数详细解释如下：

- 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:2019-05-29 14:02:41
```

utctime

函数详细解释如下:

- 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:2019-05-29 06:02:41
```

cookie_time

函数详细解释如下:

- 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-May-19 06:02:41 GMT
```

http_time

函数详细解释如下：

- 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 May 2019 06:02:41 GMT
```

parse_http_time

函数详细解释如下：

- 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` 。
- 示例

```
say(concat('parse_http_time:', parse_http_time(http_time(time()))))

输出：
parse_http_time:1559109761
```

unixtime

函数详细解释如下：

- 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
```

4.10. 密码算法相关

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

函数详细解释如下：

- 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。
- 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')) {
```

```
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_enc

函数详细解释如下：

- 语法：`aes_enc(o, s)`。
- 说明

  AES加密。
- 参数
  - s：明文或密文。
  - o：表示 `aes_new` 返回的AES对象。
- 返回值

  返回对 `s` 加密后的密文。
- 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

函数详细解释如下：

- 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)
}
```

sha1

函数详细解释如下：

- 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。
- 示例

```
digest = sha1('hello sha')
say(concat('sha1:', tohex(digest)))

输出：
sha1:853789bc783a6b573858b6cc9f913afe82962956
```

sha2

函数详细解释如下：

- 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。

- 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

函数详细解释如下：

- 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。

- 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): 59d7c07996ff675b45bd5fd40a6122bb5f40f597357a9b4a9e29da6f5c7cb806798c016fe09cb4
6457b6df9717d26d0af19896f72eaf4296be03e3681fea59ad
hmac_sha1(): 5555633cef48c3413b68f9330e99357df1cc3d93
```

hmac_sha1

函数详细解释如下：

- 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。
- 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): 59d7c07996ff675b45bd5fd40a6122bb5f40f597357a9b4a9e29da6f5c7cb806798c016fe09cb4
6457b6df9717d26d0af19896f72eaf4296be03e3681fea59ad
hmac_sha1(): 5555633cef48c3413b68f9330e99357df1cc3d93
```

md5

函数详细解释如下：

- 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

函数详细解释如下：

- 语法： `md5_bin(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摘要的二进制形式。

- 示例

```
say(concat('md5_bin: ', tohex(md5_bin('hello md5'))))

输出:
md5_bin: 741fc6b1878e208346359af502dd11c5
```

4.11. JSON相关

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

函数详细解释如下：

- 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

函数详细解释如下：

- 语法：`json_dec(s)`。

- 说明

  JSON解码。

- 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
```

4.12. Misc相关

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

函数详细解释如下：

- 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

函数详细解释如下：

- 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

函数详细解释如下：

- 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

函数详细解释如下：

- 语法：`url_unescape(s)`。
- 说明：

  URL解码。

- 参数：

  s：待解码的字符串。

- 返回值：

  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
```

rand

函数详细解释如下：

- 语法： `rand(n1, n2)` 。
- 说明

  生成随机数，随机数范围：n1 <= 返回值 <= n2。
- 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

函数详细解释如下：

- 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

函数详细解释如下：

- 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

函数详细解释如下：

- 语法：`tonumber(s [, base])` 。
- 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
```

5.EdgeScript场景示例

本文为您介绍EdgeScript的定制化鉴权逻辑、定制化请求头和响应头控制、定制化改写和重定向、定制化M3U8改写、定制化缓存控制、定制化限速的场景示例。

定制化鉴权逻辑

自定义鉴权算法场景示例如下：

- 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EdgeScript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_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('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)
  }

}
```

定制化请求头和响应头控制

文件自动重命名场景示例如下：

- 需求

  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EdgeScript规则

```
//filename增加双引号，34为双引号的ascii，可经tochar转回字符串。
//示例：add_rsp_header('Content-Disposition', concat('attachment;filename=', tochar(34), filename, tochar(34)))
//输出：Content-Disposition: attachment;filename="monitor.apk"

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

定制化改写和重定向

定制化改写和重定向场景示例如下：

- 精确URI改写
  - 需求

    将用户请求 `/hello` 在CDN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EdgeScript规则

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

- 文件后缀改写
  - 需求

    将用户请求 `/1.txt` ，通过302重定向到 `/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EdgeScript规则

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

- 文件后缀小写化
  - 需求

    将URI改成小写。

- 对应的EdgeScript规则

```
pcs = capture_re($uri, '^(.+%.)([^.]+)')
section = get(pcs, 1)
postfix = get(pcs, 2)

if and(section, postfix) {
  rewrite(concat(section, lower(postfix)), 'break')
}
```

- 添加URI前缀
  - 需求

    将用户请求 `^/nn_live/(.*)` ，通过302重定向到 `/3rd/nn_live/$1` 。
  - 对应的EdgeScript规则

```
pcs = capture_re($uri, '^/nn_live/(.*)')
sec = get(pcs, 1)

if sec {
   dst = concat('/3rd/nn_live/', sec)
   rewrite(dst, 'break')
}
```

- 302重定向
  - 需求

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

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

- 302重定向HTTPS
  - 需求

    将如下URI（对根目录匹配， `^/$` ）跳转到 `https://rtmp.cdnpe.com/index.html` ，跳转后的URI可按需填写。
    - `http://rtmp.cdnpe.com`
    - `https://rtmp.cdnpe.com`

- 对应的EdgeScript规则

```
if eq($uri, '/') {
  rewrite('https://rtmp.cdnpe.com/index.html', 'redirect')
}
```

定制化缓存控制

自定义缓存时长的场景样例如下：

- 需求

  根据各类条件，自定义资源缓存时长。

- 对应的EdgeScript规则

```
//说明：/image开头的uri，针对响应码设置缓存时长，301缓存10s，302缓存5s
if match_re($uri, '^/image') {
  set_cache_ttl('code', '301=10,302=5')
}

if eq(substr($uri, -4, -1), '.mp4') {
  set_cache_ttl('path', 5)
}
if match_re($uri, '^/201801/mp4/') {
  set_cache_ttl('path', 50)
}
if match_re($uri, '^/201802/flv/') {
  set_cache_ttl('path', 10)
}
```

定制化限速

自定义限速值的场景样例如下：

- 需求

  如果有参数 `sp` 和 `unit`，则实施限速。 `sp` 参数指明限速数值， `unit` 为参数单位，KB或MB。

- 对应的EdgeScript规则

```
if and($arg_sp, $arg_unit) {
  sp = tonumber($arg_sp)
  if not(sp) {
    add_rsp_header('X-LIMIT-DEBUG', 'invalid sp')
    return false
  }

  if and(ne($arg_unit, 'k'), ne($arg_unit, 'm')) {
    add_rsp_header('X-LIMIT-DEBUG', 'invalid unit')
    return false
  }

  add_rsp_header('X-LIMIT-DEBUG', concat('set on: ', sp, $arg_unit))
  limit_rate(sp, $arg_unit)
  return true
}
```