ALIBABA CLOUD

# Alibaba Cloud

CDN

EdgeScript

Document Version: 20201125

Alibaba Cloud

# Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

# Document conventions

| Style | Description | Example |
| --- | --- | --- |
| Danger | A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results. | **Danger:** Resetting will result in the loss of user configuration data. |
| Warning | A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results. | **Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance. |
| Notice | A caution notice indicates warning information, supplementary instructions, and other content that the user must understand. | **Notice:** If the weight is set to 0, the server no longer receives new requests. |
| Note | A note indicates supplemental instructions, best practices, tips, and other content. | **Note:** You can use Ctrl + A to select all files. |
| > | Closing angle brackets are used to indicate a multi-level menu cascade. | Click **Settings**> **Network**> **Set network type**. |
| **Bold** | Bold formatting is used for buttons , menus, page names, and other UI elements. | Click **OK**. |
| `Courier font` | Courier font is used for commands | Run the `cd /d C:/window` command to enter the Windows system folder. |
| *Italic* | Italic formatting is used for parameters and variables. | `bae log list --instanceid` *Instance_ID* |
| [] or [a\|b] | This format is used for an optional value, where only one item can be selected. | `ipconfig [-all\|-t]` |
| {} or {a\|b} | This format is used for a required value, where only one item can be selected. | `switch {active\|stand}` |

# Table of Contents

1.Introduction 05
1.1. Overview 05
1.2. How EdgeScript works 05
1.3. Quick start 07
1.4. Configure a script in the console 12
1.5. Use the EdgeScript CLI to configure scripts 16
2.EdgeScript syntax 19
3.EdgeScript built-in variables 23
4.EdgeScript built-in functions 25
4.1. Overview 25
4.2. IF functions 25
4.3. Numeric functions 30
4.4. String functions 41
4.5. Dictionary functions 51
4.6. Request processing functions 56
4.7. Throttling functions 66
4.8. Cache functions 67
4.9. Time functions 68
4.10. Cipher algorithm functions 72
4.11. JSON functions 81
4.12. Miscellaneous functions 82
5.EdgeScript scenarios 87

# 1.Introduction

# 1.1. Overview

EdgeScript enables script-based programmable configurations for CDN. You can use EdgeScript to efficiently customize the CDN service according to your business needs. In this way, you can shorten the cycle of releasing custom features, and rapidly optimize your business. This topic introduces the concept of EdgeScript and describes corresponding application scenarios and how to configure EdgeScript rules in the CDN console.

## What is EdgeScript?

EdgeScript is a script tool designed for CDN. This tool supports powerful scripts and simple syntax, and allows you to efficiently customize the CDN service. You can use EdgeScript to easily build a custom business system based on Alibaba Cloud CDN. Your EdgeScript configurations take effect among all CDN nodes within several seconds. In addition, you can continuously benefit from agile and iterative business development.

## Scenarios

EdgeScript can be used to customize the following features:

- Authentication
- Request and response header control
- Configuration rewriting and redirection
- A/B testing
- Cache control
- Throttling
- Other features

## Process of configuring EdgeScript rules in the console

Configure a script in the console

# 1.2. How EdgeScript works

This topic describes the model of scripts in EdgeScript, positions where the scripts are executed, script priorities, execution and termination of scripts, and fields that each script contains.

## Model

Scripts in EdgeScript are executed by using the following model:

- EdgeScript executes scripts to achieve different functions. Functions are triggered when the conditions in scripts are met.
- You can specify the position where a script is executed and the priority of the script in a request processing pipeline.
- Scripts in EdgeScript are managed by domain name.

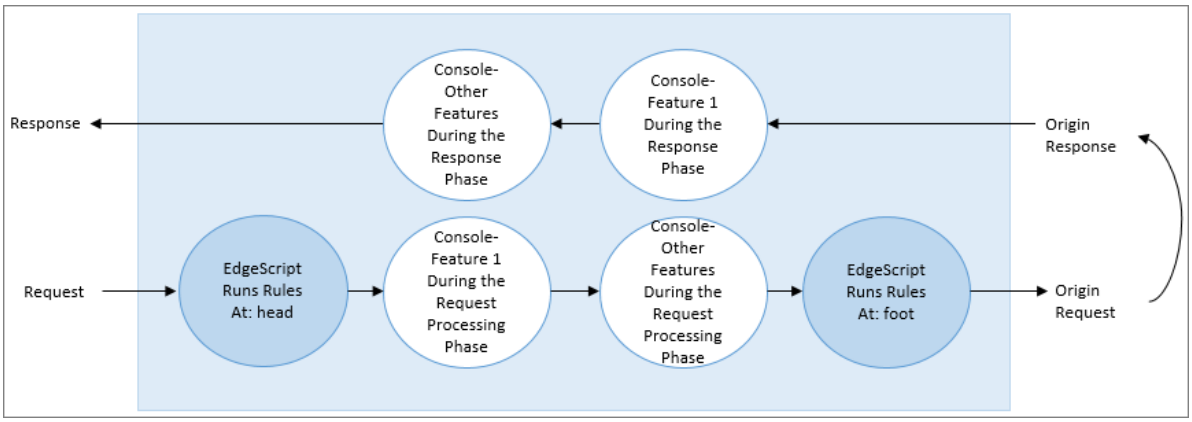
## Positions and priorities

The execution positions and priorities for executing scripts are:

- Positions

  You can execute a script at the start or end of a request processing pipeline.

  - Start: for authentication, blocking, or throttling.
  - End: for cache setting, back-to-origin authentication, and A/B testing.




- Priorities

  If you want to execute more than one script at the start or end of a request processing pipeline, you can assign priorities to these scripts to determine the execution order.

## Execution and termination

The execution and termination of scripts are defined as follows:

- Execution of scripts
  - If a condition ends with `return true` in a script is met, the script is executed.
  - If a condition ends with `return false` in a script is met, the script is skipped.
- Termination of scripts

  For scripts that are executed in the same position, you can choose to skip the subsequent scripts when a script is executed.

## Fields

A script contains the following fields:

| Field | Required | Definition | Description |
| --- | --- | --- | --- |
| enable | Y | Enables or disables the script. | Valid values:<br>• on<br>• off |
| pos | Y | The position where the script is executed. | Valid values:<br>• head<br>• foot |

| Field | Required | Definition | Description |
| --- | --- | --- | --- |
| pri | Y | The priority of the script. | Valid values: from 0 to 999. Value 0 indicates the highest priority and value 999 indicates the lowest priority. You can only prioritize scripts that are executed in the same position. |
| rule | Y | The content of the script. | - |
| brk | N | Indicates whether to skip the subsequent scripts if the current script is executed. | Valid values:<br>• on<br>• off |
| testip | N | The IP address of the client. | By default, this field is empty. If you specify a client IP address, only requests sent from the specified IP address can trigger the execution of the script. |
| option | N | An extension. | EdgeScript supports extensions. You can set this field to `_es_dbg=signature` to perform response header debugging. |

# 1.3. Quick start

This topic provides examples to describe how to use EdgeScript to create, save, test, and publish scripts.

You can use EdgeScript to perform the following operations:

- Save a script to a local file.

  For example, the m3u8.es script is used to block all M3U8 requests. You can save the script to a local file.

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

- Publish a script to the staging environment.

```
$./es.py action=push_test_env domain=<your domain> rule='{"pos":"head","pri":"0","rule_path":"./m3u
8.es","enable":"on"}'
Response Code:
=============
200 OK
Response Info:
==============
{
  "RequestId": "FB98CC67-8FBA-44CF-A98A-BCE3B19FE510"
}
```

- Query scripts in the staging environment.

```
$./es.py action=query_test_env domain=<your domain>
Response Code:
=============
200 OK
Response Info:
==============
{
  "DomainConfigs": [
    {
      "Status": "success",   # If success is returned, the script is enabled.
      "ConfigId": 17432558,
      "FunctionArgs": [
        {
          "ArgName": "enable",
          "ArgValue": "on"
        },
        {
          "ArgName": "pri",
          "ArgValue": "0"
        },
        {
          "ArgName": "pos",
          "ArgValue": "head"
        },
        {
          "ArgName": "rule",
          "ArgValue": "if eq(substr($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"
}
```

- Test scripts.

```
$curl -x Staging environment 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
```

- Publish all scripts from the staging environment to the production environment.

```
$./es.py action=push_product_env domain=<your domain>
Response Code:
=============
200 OK
Response Info:
==============
{
  "RequestId": "F4B378F8-6AAE-457A-A70C-E856ED8341D8"
}
```

- Query scripts in the production environment.

```
$./es.py action=query_product_env domain=<your domain>
Response Code:
=============
200 OK
Response Info:
==============
{
  "DomainConfigs": {
    "DomainConfig": [
      {
        "Status": "success",
        "ConfigId": 17432558,
        "FunctionArgs": {
          "FunctionArg": [
            {
              "ArgName": "enable",
              "ArgValue": "on"
            },
            {
              "ArgName": "pri",
              "ArgValue": "0"
            },
            {
              "ArgName": "pos",
              "ArgValue": "head"
            },
            {
              "ArgName": "rule",
              "ArgValue": "if eq(substr($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"
}
```

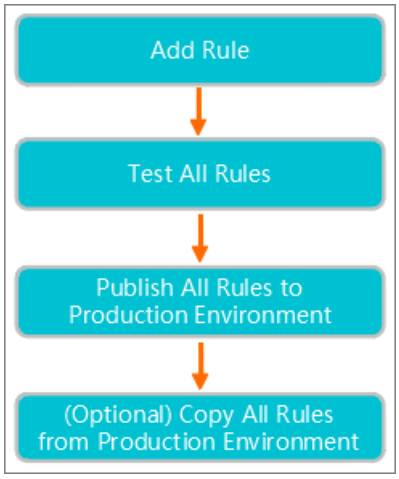
# 1.4. Configure a script in the console

EdgeScript enables script-based programmable configurations for Alibaba Cloud Content Delivery Network (CDN). EdgeScript allows you to efficiently customize your CDN service based on your business requirements. In the Alibaba Cloud CDN console, you can create scripts based on the EdgeScript coding standard, and publish the scripts to the production environment to customize your CDN service. This topic describes how to create a script in the Alibaba Cloud CDN console.

## Context

EdgeScript is a type of script designed for Alibaba Cloud CDN. EdgeScript supports complex scripts with simple syntax, and allows you to efficiently customize your CDN service. EdgeScript enables you to easily build a custom business system based on Alibaba Cloud CDN. Custom scripts can take effect among all CDN nodes within several seconds. Agile and fast service releases and upgrades help you develop services in a more efficient way.

The following figure shows the procedure for configuring a script in the Alibaba Cloud CDN console.

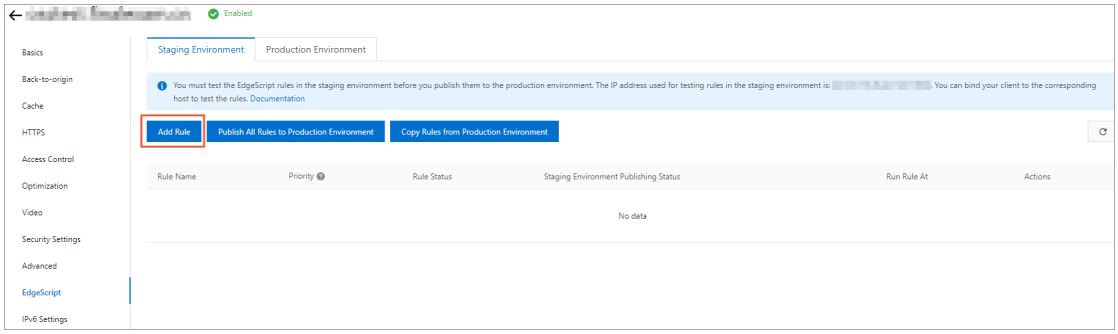



## Procedure

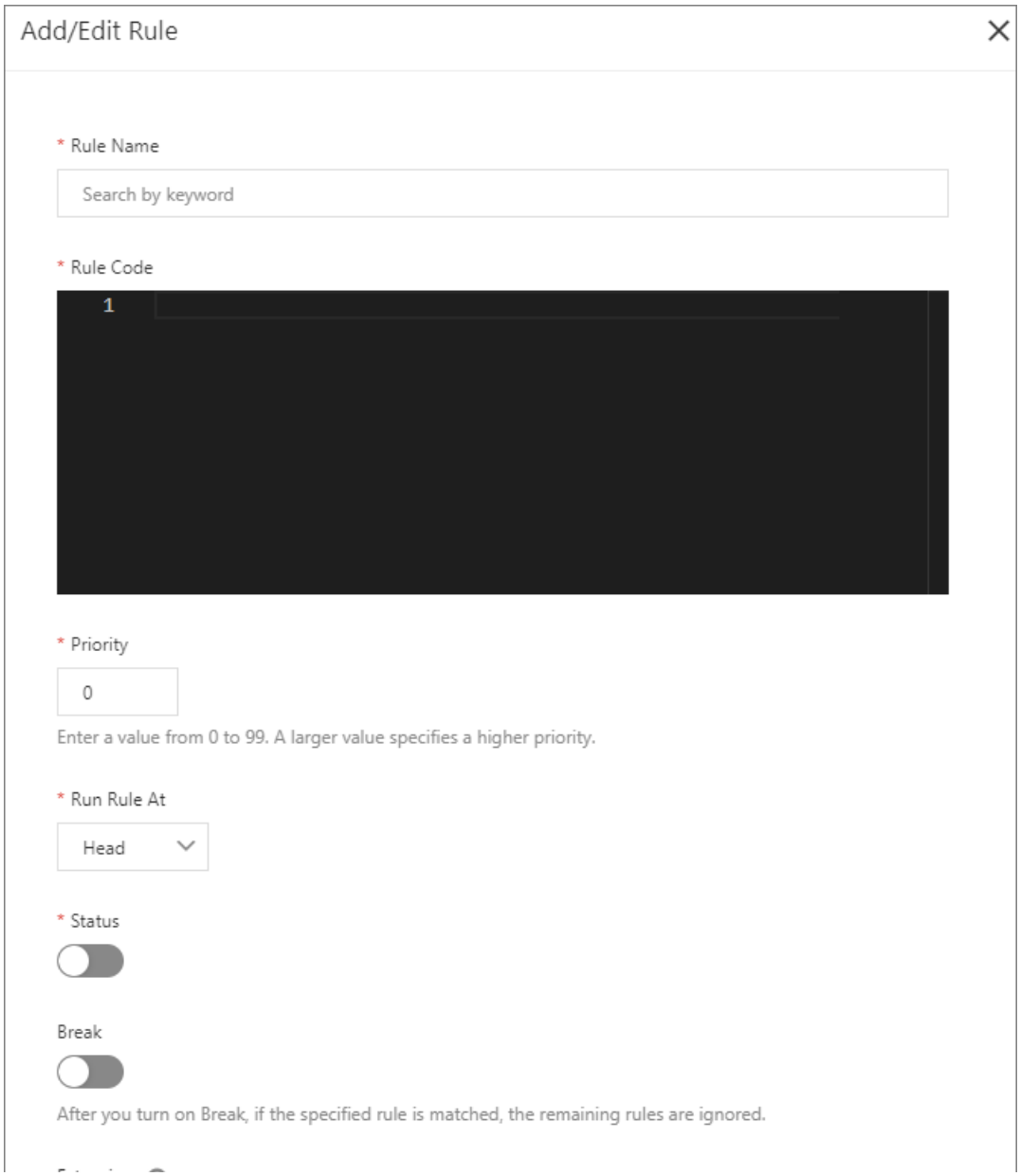
1. Log on to the Alibaba Cloud CDN console.
2. In the left-side navigation pane, click **Domain Names**.
3. On the **Domain Names** page, find the domain name that you want to manage and click **Manage** in the Actions column.
4. Click **EdgeScript**.
5. Create a script in the staging environment.

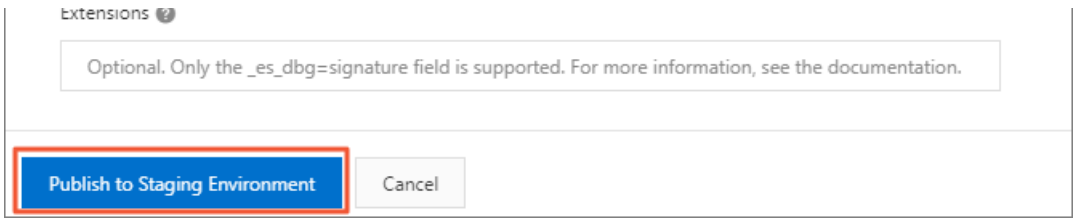
i. On the **Staging Environment** page, click **Add Rule**.

> **Note** You can create only one script for each domain name. To create more scripts for a domain name, submit a ticket.



ii. In the Add/Edit Rule pane, set the parameters.





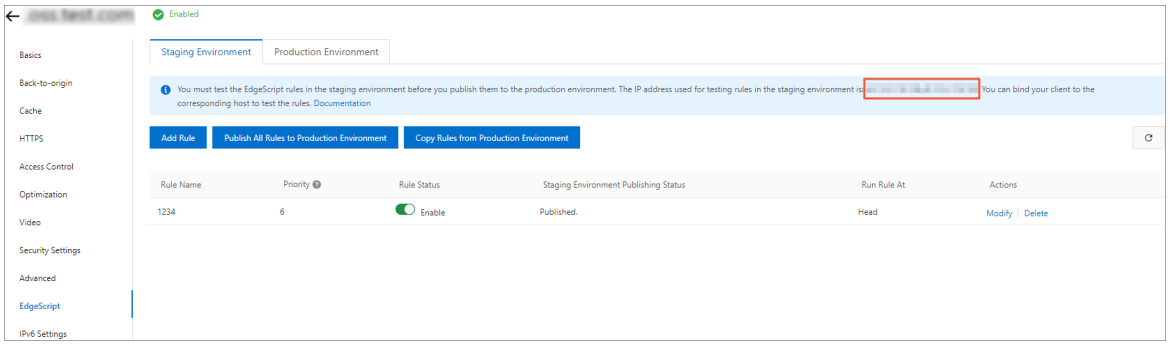

For more information about the EdgeScript parameters, see Fields.

You can customize the script based on the requirements of the scenario. For more information, see EdgeScript scenarios.

iii. Click **Publish to Staging Environment**.

6. Test the script in the staging environment.

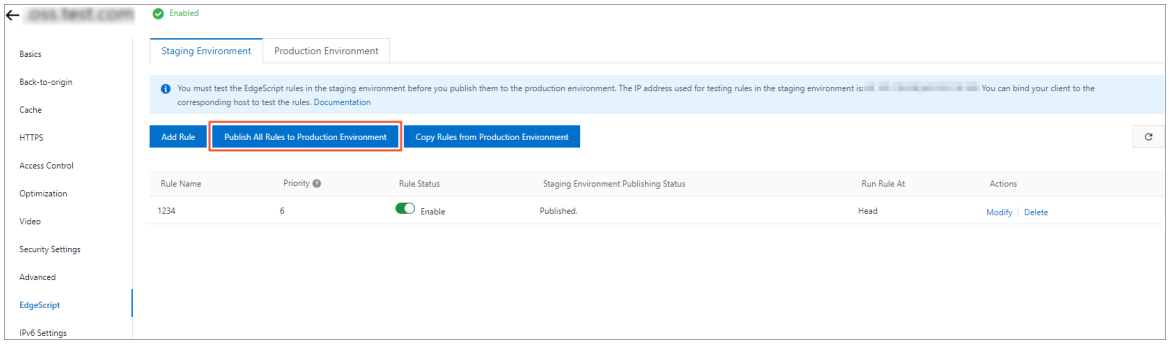
   The following figure shows the IP address of the host for testing scripts in the staging environment. Use the IP address of the test host that is displayed in the console.



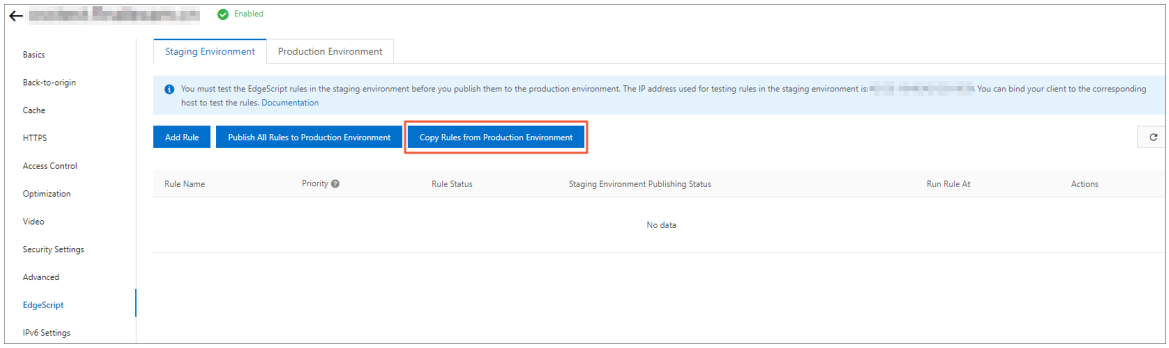

   On your machine, find the **hosts** file under the path *C:\Windows\System32\drivers\etc*. Add the IP address of the test host to the **hosts** file.

7. After the test is completed, click **Publish All Rules to Production Environment** to publish the script to the production environment.

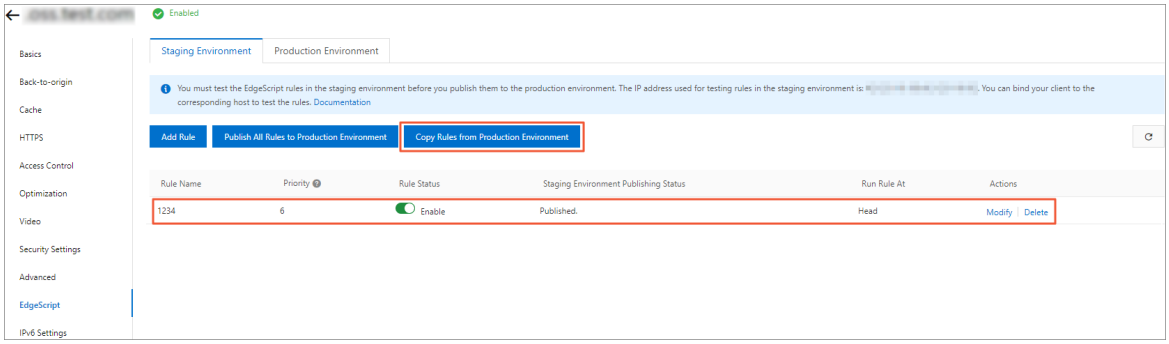
   > **Notice** After you publish a script from the staging environment to the production environment, the script in the staging environment is automatically cleared.




8. To modify scripts that you have already published to the production environment, you must copy the scripts from the production environment to the staging environment, and then modify them. You can click **Copy Rules from Production Environment** to copy scripts from the production environment to the staging environment.




After you copy a script from the production environment to the staging environment, you can modify the copied script in the staging environment.




# 1.5. Use the EdgeScript CLI to configure scripts

This topic describes the functions of the EdgeScript command line interface (CLI) and how to use the EdgeScript CLI.

## Functions

The EdgeScript CLI supports the following functions:

- Publish scripts to the staging environment.
- Publish scripts from the staging environment to all CDN nodes (production environment), and roll back scripts from the production environment to the staging environment.
- Query, modify, and delete scripts in the staging and production environments.

## Download the EdgeScript CLI

You can click attachment to download the EdgeScript CLI.

## Work with EdgeScript CLI

You can use the EdgeScript CLI to perform the following tasks:

- Configure your AccessKey pair.

```
$python ./es.py config --id=AK_ID --secret=AK_SECRET
$cat aliyun.ini
[Credentials]
accesskeyid = AccessKey ID
accesskeysecret = AccessKey secret
```

- Publish a script to the staging or production environment.

```
./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>"}'
```

> Note
> - Create a script: No configuration ID (configid) is required.
> - Modify a script: You must specify the configuration ID (configid). You can perform the query action to query the configuration ID (configid) of a script.
> - You can specify multiple scripts.

- Query scripts in the staging or production environment.

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

- Delete a script from the staging or production environment.

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

> Note You can also call an API operation to query the configuration IDs (configid) of scripts.

- Publish scripts from the staging environment to the production environment or roll back scripts from the production environment to the staging environment.

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

## Request header debugging

1. Perform debugging.

   To perform debugging, you can configure the _es_dbg parameter in the Alibaba Cloud CDN console that supports WebIDE or run the following command:

```
./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. Check debugging results.

   The _es_dbg parameter is contained in the request header. The value of _es_dbg is set to the value of _es_dbg that you set in the option extension of the script. To view the debugging result, check the following information contained in the response header:

   TRACE information: `X-DEBUG-ES-TRACE-RULE-{Script ID}`. Check the control flow of the scrip. The format of the control flow is `_flow number_function name (input parameter): Response{_execution time}`.

# 2.EdgeScript syntax

This topic describes conventions of annotations, identifiers, data types, variables, operators, clauses, and functions in the EdgeScript syntax.

## Annotations

All annotations must start with a number sign (#).

Example: # this is an annotation

## Identifiers

The identifier conventions are:

- Identifiers are case-sensitive. An identifier can contain letters, digits, and underscores (_). It must not start with a digit.
- All names of built-in variables, custom variables, built-in functions, and custom functions must comply with the identifier conventions.

## Data types

The data type conventions are:

- String

  Literal constants: use a pair of apostrophes ('') to quote a literal constant, for example, `'hello, EdgeScript'`.

- Number

  Literal constants: decimal numbers, for example, 10, -99, or 1.1.

- Boolean

  Literal constants: true or false.

- Dictionary

  Literal constants:

  - []: empty string.
  - `['key1', 'key2', 100]`:
    - `1 -> 'key1'`
    - `2 -> 'key2'`
    - `3 -> 'key3'`
  - `['key1' = 'value1', 'key2' = 1000]`
    - `'key1' -> 'value1'`
    - `'key2' -> 1000`

## Variables

The variable conventions are:

- Definition

  A variable is a symbolic name associated with a value that may change.

- Use variables
  - Both built-in and custom variables are referenced by using their names.
    - Reference a built-in variable: `host`.
    - Reference a custom variable: `seckey`.
  - To indicate that the variable is a built-in variable, add a dollar sign ( `$` ) before the variable name.

    Reference a built-in variable: `$host`.
  - Custom variables and built-in variables must not use the same name.

    For more information about built-in variables, see EdgeScript built-in variables.

## Operators

The operator conventions are:

- =: the evaluation operator.
  - Example: `seckey = 'ASDLFJ234dxvf34sDF'`
  - Example: `seckeys = ['key1', 'key2']`
- -: the minus operator.

  Example: `inum = -10`
- Built-in functions are used to process different types of data. No additional operators are provided. For more information about built-in functions, see IF functions.
  - The built-in functions support the following data types:
    - String
    - Numeric
    - Dictionary
  - Examples
    - `sval = concat(sval, 'trail')`
    - `len(arrvar)`

## Clauses

The clause conventions are:

- Condition clause

```
if condition {
  ...
}
if condition1 {
  if conditon2 {
     ...
  }
}
if condition {
  ...
} else {
  ...
}
```

- Clause descriptions
  - A `condition` clause contains the following elements:
    - Literal constant
    - Variable
    - Function call
  - Body
    - The body can be empty.
    - Multiple statements are allowed: only one statement is allowed on each line.
  - Statement wrapping is allowed.
  - CodingStyle

    The opening brace ({) must follow `if condition` on the same line.

## Functions

The syntax and conventions of functions are:

- Syntax

```
def Function name(Parameter list) {
  ...
}
```

- Descriptions
  - Parameter list
    - The parameter list can be empty.
    - Multiple parameters are allowed: separate parameters with commas (,).

- Body
  - The body can be empty.
  - Multiple statements are allowed: only one statement is allowed on each line.
  - Return values: the return clause is supported.
- CodingStyle

  The opening brace ({) must follow `def` Function name(Parameters list) on the same line.

- Function calls

  You must use `Function name()` to call both built-in and custom functions.

## Others

You must not use quotation marks (") in EdgeScript.

# 3.EdgeScript built-in variables

This topic describes EdgeScript built-in variables and the corresponding NGINX variables.

The following table lists the EdgeScript built-in variables.

| Built-in variable | Description | NGINX variable |
| --- | --- | --- |
| $arg_{name} | The value of the **name** parameter in the *Query String*. The **Query String** represents request parameters in an HTTP request. | $arg_<br>**Note** Hyphens (-) in the {name} field must be replaced by underscores (_). For example, X-USER-ID must be changed to $arg_x_user_id. |
| $http_{name} | The value of the name field in the request header. | $http_<br>**Note** Hyphens (-) in the {name} field must be replaced by underscores (_). For example, X-USER-ID must be changed to $http_x_user_id. |
| $cookie_{name} | The value of the name field in the request cookie header. | $cookie_<br>**Note** Hyphens (-) in the {name} field must be replaced by underscores (_). For example, X-USER-ID must be changed to $http_x_user_id. |
| $scheme | The protocol type. | $scheme |
| $server_protocol | The version of the protocol. | $server_protocol |
| $host | The original host. | $host |

| Built-in variable | Description | NGINX variable |
| --- | --- | --- |
| $uri | The original URI. | - |
| $args | `$args` represents all request parameters in an HTTP request, excluding question marks ( `?` ). For example, the URI of the request is `http://www.a.com/1k.file?k1=v1&k2=v2` .<br>• `$arg_k1` returns the value of the k1 parameter: `v1` .<br>• `$args` is used to return the entire Query String: `k1=v1&k2=v2` . Question marks ( `?` ) are excluded. | $args |
| $request_method | The HTTP method. | $request_method |
| $request_uri | *uri*+'?'+args . | $request_uri |
| $remote_addr | The IP address of the client that sends the request. | $remote_addr |

**Note**

- The dollar sign ( `$` ) before a variable is used to indicate that the variable is a built-in variable. You can remove the dollar sign as needed.
- Do not assign values to built-in variables in the same way as parameters.

# 4.EdgeScript built-in functions

## 4.1. Overview

This topic describes the categorization of EdgeScript built-in functions.

The following table lists the categories and the built-in functions of EdgeScript.

| Category | Function |
| --- | --- |
| IF functions | and, or, not, eq, and ne |
| Numeric functions | add, sub, mul, div, mod, gt, ge, lt, le, floor, and ceil |
| String functions | substr, concat, upper, lower, len, byte, match_re, capture_re, gsub_re, split, split_as_key, tohex, tostring, and tochar |
| Dictionary functions | set, get, and foreach |
| Request processing functions | add_req_header, del_req_header, add_rsp_header, del_rsp_header, encode_args, decode_args, rewrite, say, print, and exit |
| Throttling functions | limit_rate_after and limit_rate |
| Cache functions | set_cache_ttl |
| Time functions | today, time, now, localtime, utctime, cookie_time, http_time, parse_http_time, and unixtime |
| Cipher algorithm functions | aes_new, aes_enc, aes_dec, sha1, sha2, hmac, hmac_sha1, md5, and md5_bin |
| JSON functions | json_enc and json_dec |
| Miscellaneous functions | base64_enc, base64_dec, url_escape, url_unescape, rand, rand_hit, rand_bytes, crc, and tonumber |

## 4.2. IF functions

This topic describes the syntax, description, parameters, return values, and examples of IF functions.

### and

This function is described as follows:

- Syntax: `and(arg, ...)`
- Description
  - You can call this function to execute a logical AND operation.

- Short-circuit semantics is supported. When a value evaluates to false, the subsequent values are not evaluated.

- Parameters

  arg: the value to check whether is true. Data type: any type. You can specify one or more values.

- Return values

  This function returns `true` if all values evaluate to true and returns `false` if any value evaluates to false.

- Examples

```
if and($arg_mode, eq($arg_mode, 'set_header')) {
  add_rsp_header('USER-DEFINED-1','path1')
}
Notes
a. If the request includes the mode parameter and the value of the mode parameter is set_header, set the USER-DEFINED-1 response header.
b. If the request does not include the mode parameter, the short-circuit semantics takes effect and the subsequent eq comparison is not executed. The response header USER-DEFINED-1 will not be set because the and () function returns false.
```

## or

This function is described as follows:

- Syntax: `or(arg, ...)`
- Description
  - You can call this function to execute a logical OR operation.
  - Short-circuit semantics is supported. When a value evaluates to true, the subsequent values are not evaluated.

- Parameters

  arg: the value to check whether is true. Data type: any type. You can specify one or more values.

- Return values

  This function returns `true` if any value evaluates to true and returns `false` if all values evaluate to false.

- Examples

```
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')
}
```

Notes

a. If the request includes the from header and its value is wap or comos, the URL is 302 rewritten as http://tech.com.cn/zt_d/we2015/[wap|comos].

b. If the request includes the from header and its value is wap, the short-circuit semantics takes effect and the subsequent eq comos comparison is not executed. At the same time, the or () function returns true.

## not

This function is described as follows:

- Syntax: `not(arg)`
- Description

  You can call this function to execute a logical NOT operation. The values of `undef` and `false` evaluate to false, and other values evaluate to true.
- Parameters

  arg: the value to convert to its opposite value. Data type: any type. You can specify only one value.
- Return values
  - true
  - false
- Examples

```
if not($arg_key) {
  exit(403)
}
```

Note: If a request does not include the key parameter, the request is denied and status code 403 is returned.

```
if not($cookie_user) {
  exit(403, 'not cookie user')
}
```

Note: If a request does not include cookie_user, the request is denied, a response that contains "not cookie user" is returned with status code 403.

```
if not(0) {
  exit(403)
}
```

Note: The not (0) function returns false.

```
if not(false) {
  exit(403)
}
```

Note: The not (false) function returns true.

## eq

This function is described as follows:

- Syntax: `eq(arg1, arg2)`
- Description

  You can call this function to compare whether the two values are equal.
- Parameters
  - arg1: the first value to compare. Data type: any type.
  - arg2: the second value to compare. Data type: same as the `arg1` parameter.
- Return values

  This function returns `true` if the two values are equal and returns `false` if they are not equal.
- Examples

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

Notes

a. If the request includes both the k1 and k2 parameters, the subsequent comparison operations are executed.

b. If the request does not include the k1 or k2 parameter, the short-circuit semantics take effect and the subsequent comparison operations are not executed.

c. eq: whether the value of the k1 parameter is equal to value1.

d. ne: whether the value of the k2 parameter is not equal to value2.

e. The response that contains "match condition" is returned only if the following conditions are met: the request includes both the k1 and k2 parameters, the value of the k1 parameter is equal to value1, and the value of the k2 parameter is not equal to value2.

## ne

This function is described as follows:

- Syntax: `ne(arg1, arg2)`
- Description

  You can all this function to compare whether the two values are not equal.
- Parameters
  - arg1: the first value to compare. Data type: any type.
  - arg2: the second value to compare. Data type: same as the `arg1` parameter.
- Return values

  This function returns `true` if the two values are not equal and returns `false` if they are equal.
- Examples

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

Notes

a. If the request includes both the k1 and k2 parameters, the subsequent comparison operations are executed.

b. If the request does not include the k1 or k2 parameter, the short-circuit semantics take effect and the subsequent comparison operations are not executed.

c. eq: whether the value of the k1 parameter is equal to value1.

d. ne: whether the value of the k2 parameter is not equal to value2.

e. The response that contains "match condition" is returned only if the following conditions are met: the request includes both the k1 and k2 parameters, the value of the k1 parameter is equal to value1, and the value of the k2 parameter is not equal to value2.key1 = 'value1'
key2 = 'value2'
if and($arg_k1, $arg_k2, eq(key1, $arg_k1), ne(key2, $arg_k2)) {
  say('match condition')
}
Notes
a. If the request includes both the k1 and k2 parameters, the subsequent comparison operations are executed.
b. If the request does not include the k1 or k2 parameter, the short-circuit semantics take effect and the subsequent comparison operations are not executed.
c. eq: whether the value of the k1 parameter is equal to value1.
d. ne: whether the value of the k2 parameter is not equal to value2.
e. The response that contains "match condition" is returned only if the following conditions are met: the request includes both the k1 and k2 parameters, the value of the k1 parameter is equal to value1, and the value of the k2 parameter is not equal to value2.

# 4.3. Numeric functions

This topic describes the syntax, description, parameters, and response parameters of numeric functions. This topic also provides examples of these functions.

## add

Details about this function:

- Syntax: `add(n1, n2)`.
- Description

  Calculates the sum of two addends.

- Parameters
  - n1: the addend.
  - n2: the other addend.
- Response parameters

  Returns the sum of `n1 + n2`.
- Examples

```
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))
Output:
n1=30
n2=-10
n3=200
n4=0.5
n5=15
```

## sub

Details about this function:

- Syntax: `sub(n1, n2)`.
- Description

  Calculates the difference between two numbers.
- Parameters
  - n1: the number to be compared.
  - n2: the other number to be compared.
- Response parameters

  Returns the difference of `n1 - n2`.
- Examples

```
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))
Output:
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))
Output:
n1=30
n2=-10
n3=200
n4=0.5
n5=15
```

## mul

Details about this function:

- Syntax: `mul(n1, n2)`.
- Description

  Calculates the product of two numbers.
- Parameters
  - n1: the multiplicand.

- n2: the other multiplicand.

- Response parameters

  Returns the product of `n1 × n2`.

- Examples

```
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))
Output:
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))
Output:
n1=30
n2=-10
n3=200
n4=0.5
n5=15
```

## div

Details about this function:

- Syntax: `div(n1, n2)`.

- Description

  Calculates the quotient of two numbers.

- Parameters
  - n1: the dividend.
  - n2: the divisor.
- Response parameters

  Returns the quotient of `n1/n2`.

- Examples

```
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))
Output:
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))
Output:
n1=30
n2=-10
n3=200
n4=0.5
n5=15
```

## mod

Details about this function:

- Syntax: `mod(n1, n2)`.
- Description

  Calculates the remainder after division of one number by another.
- Parameters
  - n1: the dividend.

- n2: the divisor.

- Response parameters

  Returns the remainder of `n1 % n2`.

- Examples

```
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))
Output:
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))
Output:
n1=30
n2=-10
n3=200
n4=0.5
n5=15
```

## gt

Details about this function:

- Syntax: `gt(n1, n2)`

- Description

  Determines whether a number is greater than another one.

- Parameters
  - n1: the number to be compared.
  - n2: the other number to be compared.

- Response parameters

  If `n1 > n2`, `true` is returned. Otherwise, `false` is returned.

- Examples

```
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')
}
Request: /path1/path2/file? num=10  Response: num <= 10   num >= 10
Request: /path1/path2/file? num=11  Response: num > 10    num >= 10
Request: /path1/path2/file? num=9   Response: num < 10    num <= 10
```

## ge

Details about this function:

- Syntax: `ge(n1, n2)`.
- Description

  Determines whether a number is greater than or equal to another one.

- Parameters
  - n1: the number to be compared.
  - n2: the other number to be compared.

- Response parameters

  If `n1 >= n2`, `true` is returned. Otherwise, `false` is returned.

- Examples

```
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')
}
Request: /path1/path2/file? num=10  Response: num <= 10   num >= 10
Request: /path1/path2/file? num=11  Response: num > 10    num >= 10
Request: /path1/path2/file? num=9   Response: 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')
}
Request: /path1/path2/file? num=10  Response: num <= 10   num >= 10
Request: /path1/path2/file? num=11  Response: num > 10    num >= 10
Request: /path1/path2/file? num=9   Response: num < 10    num <= 10
```

## lt

Details about this function:

- Syntax: `lt(n1, n2)`.
- Description

  Determines whether a number is smaller than another one.
- Parameters
  - n1: the number to be compared.
  - n2: the other number to be compared.

- Response parameters

  If `n1 < n2`, `true` is returned. Otherwise, `false` is returned.

- Examples

```
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')
}
Request: /path1/path2/file? num=10  Response: num <= 10   num >= 10
Request: /path1/path2/file? num=11  Response: num > 10    num >= 10
Request: /path1/path2/file? num=9   Response: 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')
}
Request: /path1/path2/file? num=10  Response: num <= 10   num >= 10
Request: /path1/path2/file? num=11  Response: num > 10    num >= 10
Request: /path1/path2/file? num=9   Response: num < 10    num <= 10
```

## le

Details about this function:

- Syntax: `le(n1, n2)`.
- Description

  Determines whether a number is smaller than or equal to another one.

- Parameters
  - n1: the number to be compared.
  - n2: the other number to be compared.
- Response parameters

  if `n1 <= n2` , `true` is returned. Otherwise, `false` is returned.
- Examples

```
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')
}
Request: /path1/path2/file? num=10  Response: num <= 10   num >= 10
Request: /path1/path2/file? num=11  Response: num > 10    num >= 10
Request: /path1/path2/file? num=9   Response: 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')
}
Request: /path1/path2/file? num=10  Response: num <= 10   num >= 10
Request: /path1/path2/file? num=11  Response: num > 10    num >= 10
Request: /path1/path2/file? num=9   Response: num < 10    num <= 10
```

## floor

Details about this function:

- Syntax: `floor(n)`.
- Description

  Generates the greatest integer smaller than or equal to a number.
- Parameters

  n: the number to be rounded down.
- Response parameters

  Returns the greatest integer smaller than or equal to the number specified by `n`.
- Examples

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

### ceil

Details about this function:

- Syntax: `ceil(n)`.
- Description

  Generates the smallest integer greater than or equal to a number.
- Parameters

  n: the number to be rounded up.
- Response parameters

  Returns the smallest integer greater than or equal to the number specified by `n`.
- Examples

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

# 4.4. String functions

This topic describes the syntax, description, parameters, and response parameters of string functions. This topic also provides examples of these functions.

## substr

Details about this function:

- Syntax: `substr(s, i, j)`.
- Description

  Extracts parts of a string.
- Parameters
  - s: the string that you want to extract.
  - i: the position to start extraction in the source string, counting from 1. A value of -1 specifies the rightmost character of the string.
  - j: the position to end extraction in the source string, counting from 1. A value of -1 specifies the rightmost character of the string.
- Response parameters

  Returns the substring `s[i, j]` of `s`.
- Examples

```
//The two methods that are used to determine whether a file is an M3U8 file
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

Details about this function:

- Syntax: `concat(s1, ...)`.
- Description

  Concatenates strings.
- Parameters

  s1: the strings that you want to concatenate. You can specify one or more strings. Numeric values are supported.
- Response parameters

  Returns the concatenated string.
- Examples

```
//The two methods that are used to determine whether a file is an M3U8 file
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($uri, ' is .m3u8'))
}//The two methods that are used to determine whether a file is an M3U8 file
if eq(substr($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

Details about this function:

- Syntax: `upper(s)`.
- Description

  Converts a string to uppercase letters.
- Parameters

  s: the string that you want to convert.
- Response parameters

  Returns the string specified by the `s` parameter in uppercase letters.
- Examples

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

## lower

Details about this function:

- Syntax: `lower(s)`.
- Description

Converts a string to lowercase letters.

- Parameters

  s: the string that you want to convert.

- Response parameters

  Returns the string specified by the `s` parameter in lowercase letters.

- Examples

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

## len

Details about this function:

- Syntax: `len(s)`.
- Description

  Queries the length of a string.

- Parameters

  s: the string that you want to measure.

- Response parameters

  Returns the length of the string specified by the `s` parameter. Data type: integer.

- Examples

```
//The two methods that are used to determine whether a file is an M3U8 file
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

Details about this function:

- Syntax: `byte(c)`.
- Description

  Queries the ASCII value of a character.
- Parameters

  c: the character whose ASCII value you want to query. You can specify only one character.
- Response parameters

  Returns the ASCII value of the specified character. Data type: numeric.
- Examples

```
//Output: 97
//65
say(byte('a'))
say(byte('A'))
```

## match_re

Details about this function:

- Syntax: `match_re(s, p [, o])`.
- Description

  Uses the PCRE engine for regular expression matching.
- Parameters
  - s: the string that you want to match. Data type: string.
  - p: the regular expression. Data type: string.
  - o: the regular expression engine. Data type: string. This parameter is optional.
  - i: specifies that this function is not case-sensitive.
- Response parameters

  If the string matches the regular expression, `true` is returned. Otherwise, `false` is returned.
- Examples

```
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

Details about this function:

- Syntax: `capture_re(s, p [,init])` .
- Description

  Captures the matches of a string and returns the matching substrings. The PCRE engine is used.
- Parameters
  - s: the string that you want to match. Data type: string.
  - p: the regular expression. Data type: string.
  - init: the position to start matching, counting from 1. Data type: integer.
- Response parameters

  Returns the matching substrings in the dictionary type if the string matches the regular expression. Otherwise, an empty dictionary is returned.
- Examples

```
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

Details about this function:

- Syntax: `gsub_re(subject, regex, replace [,option])`.
- Description

  Replaces all matches of a string and returns the string after the replacement. The PCRE engine is used.
- Parameters
  - subject: the string that you want to match. Data type: string.
  - regex: the regular expression. Data type: string.
  - replace: the string for replacement. Data type: string.

    You can specify the `replace` parameter by using the matching substrings.
    - $0: specifies all substrings that match `regex`.
    - $N: specifies the substring that matches the Nth parenthesized subexpression `()` of `regex`.
  - option: the regular expression engine. Data type: string. This parameter is optional.
- Response parameters

  Replaces all substrings that match the specified `regex` parameter in the specified `subject` parameter with the specified `replace` parameter and returns the string after the replacement.
- Examples

```
//Output:
//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))
```

- References

  For more information about the PCRE syntax, see pcre2syntax man page.

## split

Details about this function:

- Syntax: `split(s [,sep])` .
- Description

  Splits a string into an array of substrings and returns the array.
- Parameters
  - s: the string that you want to split. Data type: string.
  - sep: the separator that is used to split the string. Data type: string.
- Response parameters

  Returns an array of key-value pairs in the dictionary type. The value of the `key` parameter is a number that starts from 1, for example, [1]=xx and [2]=y. If `sep` is left empty, the string is split by whitespace characters. Whitespace characters include space characters and tab characters (\t).
- Examples

```
//Request: ? from=xx1,xx2,xx3
//Response: [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

Details about this function:

- Syntax: `split_as_key(s [,sep])`.
- Description

  Splits a string into an array of substrings and returns the array.
- Parameters
  - s: the string that you want to split. Data type: string.
  - sep: the separator that is used to split the string. Data type: string.
- Response parameters

  Returns response parameters in the same way as the `split()` function. However, the `key` parameter is named after each split element: *element 1* -> *element 2*.
- Examples

```
//Request: ? from=xx1,xx2,xx3
//Response: 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

Details about this function:

- Syntax: `tohex(s)`.
- Description

  Converts a string to a hexadecimal string.
- Parameters

  s: the string that you want to convert.
- Response parameters

  Returns the hexadecimal string that is converted from the string specified by the `s` parameter.
- Examples

```
//A response header is added
//X-DSL-TOHEX: 4ad583af22c2e7d40c1c916b2920299155a46464
digest = sha1('xxxx')
add_rsp_header('X-DSL-TOHEX', tohex(digest))
```

## tostring

Details about this function:

- Syntax: `tostring(a)` .
- Description

  Converts data of any type to a string.
- Parameters

  a: the data that you want to convert. Data type: any type.
- Response parameters

  Returns the string that is converted from the value specified by the `a` parameter.
- Examples

```
//A response header is added
//X-DSL-TOSTRING: 123
s = tostring(123)
add_rsp_header('X-DSL-TOSTRING', s)
```

## tochar

Details about this function:

- Syntax: `tochar(n1, n2, ...)` .
- Description
  - Converts one or more internal integers (ASCII values) to a string. For example, 48 corresponds to the character "0".
  - The length of the returned string is based on the number of specified parameters.
- Parameters

  nX: the integers that you want to convert. You can specify multiple integers.
- Response parameters

  Returns a string converted from integers.
- Examples

```
//Output: A response header is added
//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)
//Output: A response header is added
//Content-Disposition: attachment;filename="The value of the filename parameter"
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. Dictionary functions

This topic describes the syntax, description, parameters, and response parameters of dictionary functions. This topic also provides examples of these functions.

## set

Details about this function:

- Syntax: `set(d, k, v)` .
- Description

  Sets the `k/v` pairs in a dictionary specified by the `d` parameter.
- Parameters
  - d: the name of the dictionary.
  - k: the key. Data type: any type.
  - v: the value. Data type: any type.
- Response parameters

  Returns `true` .
- Examples

- Example 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, [])
Output:
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
```

- Example 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)
  }
}
Output: A response header is added
X-DSL-NESTED-DICT: inner dsl
```

## get

Details about this function:

- Syntax: `get(d, k)`.
- Description

  Retrieves the value ( `v` ) corresponding to a key ( `k` ) from a dictionary ( `d` ).
- Parameters

- d: the name of the dictionary.
- k: the key. Data type: any type.

- Response parameters

  If the function succeeds, the value that corresponds to the specified key is returned. Otherwise, `false` is returned.

- Examples

```
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, [])
Output:
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, [])
Output:
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

Details about this function:

- Syntax: `foreach(d, f, user_data)`.
- Description
  - Traverses elements in a dictionary ( `d` ) and executes a callback function ( `f` ) for each element.
  - Specify the `f` parameter in the syntax of `f(key, value, user_data)`.
  - When the `f()` function returns `false`, the `foreach()` loop ends.
- Parameters
  - d: the name of the dictionary.
  - f: the callback function.
  - user_data: the user data that you want to transmit. Data type: dictionary.
- Response parameters

  Returns `true`.
- Examples
  - Example 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, [])
Output:
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, [])
Output:
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
```

- Example 2

  Outputs the first two `M3U8` slices and ends the foreach loop.

```
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)
Output:
#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. Request processing functions

This topic describes the syntax, description, parameters, and response parameters of request processing functions. This topic also provides examples of these functions.

## add_req_header

Details about this function:

- Syntax: `add_req_header(name, value [, append])` .
- Description

  Adds an request header to back-to-origin requests.
- Parameters
  - `name` : the name of the request header that you want to add. Data type: string.
  - `value` : the value of the request header that you want to add. Data type: string.
  - append: specifies whether to `append` a request header with the specified `value` if a request header with the same name already exists. Valid values: true and false. Default value: false. If you set this parameter to false, the specified value will overwrite the value of the existing request header.
- Response parameters

  Returns `true` by default and returns `false` if you have specified an invalid request header.
- Examples

```
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')
The following request headers are added:
USER-DEFINED-REQ-1: 1
USER-DEFINED-REQ-1: x
The USER-DEFINED-REQ-2 header is added first and then deleted. Therefore, the USER-DEFINED-REQ-2 req
uest header is not included in the back-to-origin request.
```

## del_req_header

Details about this function:

- Syntax: `del_req_header(name)` .
- Description

  Deletes a request header from back-to-origin requests.
- Parameters

  `name` : the name of the request header that you want to delete. Data type: string.
- Response parameters

  Returns `true` by default and returns `false` if you have specified an invalid request header.
- Examples

```
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')
The following request headers are added:
USER-DEFINED-REQ-1: 1
USER-DEFINED-REQ-1: x
The USER-DEFINED-REQ-2 header is added first and then deleted. Therefore, the USER-DEFINED-REQ-2 req
uest header is not included in the back-to-origin request.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')
The following request headers are added:
USER-DEFINED-REQ-1: 1
USER-DEFINED-REQ-1: x
The USER-DEFINED-REQ-2 header is added first and then deleted. Therefore, the USER-DEFINED-REQ-2 req
uest header is not included in the back-to-origin request.
```

## add_rsp_header

Details about this function:

- Syntax: `add_rsp_header(name, value [, append])`.
- Description

  Adds a response header.
- Parameters
  - `name`: the name of the response header that you want to add. Data type: string.
  - `value`: the value of the response header that you want to add. Data type: string.

    You can specify one of the following expressions for the `value` parameter to enable the value to be dynamically replaced in the response phase.
    - ${x}: replaced with the value of `ngx.var.x`.
    - @{y}: replaced with the value of *response header*`y`.
  - `append`: specifies whether to append a response header with the specified `value` if a response header with the same name already exists. Valid values: true and false. Default value: false. If you set this parameter to false, the specified value will overwrite the value of the existing response header.
- Response parameters

  Returns `true` by default and returns `false` if you have specified an invalid response header.
- Examples

```
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')
The following response headers are added:
USER-DEFINED-RSP-1: 1
USER-DEFINED-RSP-1: x
The USER-DEFINED-RSP-2 header is added first and then deleted. Therefore, the USER-DEFINED-RSP-2 header is not included in the response.
```

## del_rsp_header

Details about this function:

- Syntax: `del_rsp_header(name)`.
- Description

  Deletes a response header.
- Parameters

  `name`: the name of the response header that you want to delete. Data type: string.
- Response parameters

  Returns `true` by default and returns `false` if you have specified an invalid response header.
- Examples

```
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')
The following response headers are added:
USER-DEFINED-RSP-1: 1
USER-DEFINED-RSP-1: x
The USER-DEFINED-RSP-2 header is added first and then deleted. Therefore, the USER-DEFINED-RSP-2 header is not included in the response.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')
The following response headers are added:
USER-DEFINED-RSP-1: 1
USER-DEFINED-RSP-1: x
The USER-DEFINED-RSP-2 header is added first and then deleted. Therefore, the USER-DEFINED-RSP-2 header is not included in the response.
```

## encode_args

Details about this function:

- Syntax: `encode_args(d)`.
- Description

  Converts the `k/v` pairs in the dictionary specified by `d` to a URI-encoded string in the format of k1=v1&k2=v2.
- Parameters

  d: the dictionary that you want to convert.
- Response parameters

  Returns a URI-encoded string.
- Examples

```
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'))
}
Output: The following response headers are added.
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

Details about this function:

- Syntax: `decode_args(s)`.
- Description

  Converts a URI-encoded string in the format of k1=v1&k2=v2 to a string of the dictionary type.
- Parameters

  s: the string that you want to convert.
- Response parameters

  Returns a dictionary object converted from the specified string.

- Examples

```
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'))
}
Output: The following response headers are added.
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'))
}
Output: The following response headers are added.
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

Details about this function:

- Syntax: `rewrite(url, flag, code)` .
- Description

  Performs a rewrite or redirect operation.
- Parameters

- url: the URI in the rewrite rule. Data type: string.
  - If you set the flag parameter to redirect or break, only the URI is rewritten. This parameter specifies the URI after the rewrite operation.
  - If you set the flag parameter to enhance_redirect or enhance_break, the URI and parameters are rewritten. This parameter specifies the URI and parameters after the rewrite operation.
- flag: the rewrite mode. Data type: string.
  - redirect: only rewrites the URI. Parameters are not rewritten. By default, a 302 redirect is performed. If you specify this mode, the `code` parameter is configurable. Valid values for the code parameter are 301, 302 (default), 303, 307, and 308.
  - break: only rewrites the URI to a URL. Parameters are not rewritten.
  - enhance_redirect: similar to `redirect`. However, both the URI and parameters are rewritten.
  - enhance_break: similar to `break`. However, both the URI and parameters are rewritten.
- code: the HTTP status code. Data type: numeric.

  This parameter is available for customization only when you set the flag parameter to redirect or enhance_redirect.

- Response parameters
  - Returns `true` by default for a rewrite operation.
  - Does not return a value by default for a rewrite operation.
- Examples

```
if and($arg_mode, eq($arg_mode, 'rewrite:enhance_break')) {
  rewrite('/a/b/c.txt? k=v', 'enhance_break')
}
```

The URI and parameters of the back-to-origin request are rewritten to /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)
}
```

A 302 or 301 redirect to /a/b/c.txt? is performed. k=v

```
if and($arg_mode, eq($arg_mode, 'rewrite:break')) {
  rewrite('/a/b/c.txt', 'break')
}
```

The URI of the back-to-origin request is rewritten to /a/b/c.txt and the original parameters in the request remain unchanged.

```
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)
}
```

A 302 or 301 redirect to /a/b/c.txt is performed and the original parameters remain unchanged.

## say

Details about this function:

- Syntax: `say(arg)`.
- Description

  Outputs a response body and appends a newline character at the end of the output.
- Parameters

  arg: the content of the response body. Data type: any type.
- Response parameters

  None.
- Examples

```
say('hello')
print('byebye')
print('byebye')
Output:
hello
byebyebyebye
```

## print

Details about this function:

- Syntax: `print(arg)`
- Description

  Outputs a response body. This function is different from the `say()` function. This function does not append a newline at the end of the output.
- Parameters

  arg: the content of the response body. Data type: any type.
- Response parameters

  None.
- Examples

```
say('hello')
print('byebye')
print('byebye')
Output:
hello
byebyebyebyesay('hello')
print('byebye')
print('byebye')
Output:
hello
byebyebyebye
```

## exit

Details about this function:

- Syntax: `exit(code [, body])`.
- Description

  Ends the current request with the specified `code`. If you also set the `body` parameter, a response that includes the specified response body is returned.
- Parameters
  - code: the status code to return.

- body: the response body.

- Response parameters

  None.

- Examples
  - Example 1

```
if not($arg_key) {
  exit(403)
}
If a request does not include the key parameter, the request is denied and status code 403 is returned.
if not($cookie_user) {
  exit(403, 'not cookie user')
}
If a request does not include cookie_user, the request is denied and a response that contains "not cookie user" is returned with the status code 403.
if not(0) {
  exit(403)
}
The not (0) function returns false.
if not(false) {
  exit(403)
}
The not (false) function returns true.
```

  - Example 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. Throttling functions

This topic describes the syntax, description, parameters, and response parameters of throttling functions. This topic also provides examples of these functions.

## limit_rate_after

Details about this function:

- Syntax: `limit_rate_after(n, unit)`.
- Description

  Sets the upper limit of the data transfer rate.
- Parameters
  - n: the upper limit. Data type: integer. If you do not specify the `unit` parameter, the unit is byte/s.
  - unit: the unit to measure the data transfer rate. Data type: char. This parameter is optional. Valid values of `unit`: `K`, `M`, and `G`. These values are not case-sensitive.
- Response parameters

  If the function succeeds, `true` is returned. Otherwise, `false` is returned.
- Examples

```
limit_rate_after(10, 'k')
limit_rate(1, 'm')
//If the data transfer rate is lower than 10 Kbit/s, it is not throttled. If the data transfer rate is higher than 10 Kbit/s, it is throttled to 1 Mbit/s.
```

## limit_rate

Details about this function:

- Syntax: `limit_rate(n, unit)`.
- Description

  Sets the value to which you want to throttle the data transfer rate. The lowest data transfer rate that you throttle to is 4 Kbit/s. If you specify a data transfer rate that is lower than 4 Kbit/s, the data transfer rate is throttled to 4 Kbit/s.
- Parameters
  - n: the lowest data transfer rate. Data type: integer. If you do not specify the `unit` parameter, the unit is byte/s.
  - unit: the unit to measure the data transfer rate. Data type: char. This parameter is optional. Valid values of `unit`: `K`, `M`, and `G`. These values are not case-sensitive.
- Response parameters

  If the function succeeds, `true` is returned. Otherwise, `false` is returned.
- Examples

```
limit_rate_after(10, 'k')
limit_rate(1, 'm')
//If the data transfer rate is lower than 10 Kbit/s, it is not throttled. If the data transfer rate is higher than 10 Kbit/s, it is throttled to 1 Mbit/s.
```

# 4.8. Cache functions

This topic describes the syntax, description, parameters, return values, and examples of cache functions.

## set_cache_ttl

This function is described as follows:

- Syntax: `set_cache_ttl(type, ttl)`
- Description

  You can call this function to set the cache duration for specific resources.
- Parameters
  - type

    The cache type. Valid values: `path and code`.
  - ttl
    - If the type parameter is set to path, the `ttl` parameter specifies the cache duration for specific resources. The value must be an integer.
    - If the type parameter is set to code, the `ttl` parameter specifies the cache duration for specific response codes. The value must be a string.
    - Default unit for the cache duration: seconds.
- Return values

  This function returns `true` upon a success and returns `false` upon a failure.
- Examples

```
if match_re($uri, '^/image') {
  set_cache_ttl('code', '301=10,302=5')
}
```

Note: Set the cache durations for status codes responded to requests to URIs that start with "/image", 10 seconds for status code 301 and 5 seconds for status code 302.

```
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)
}
```

Note: Set different cache durations for file names or URI paths.

# 4.9. Time functions

This topic describes the syntax, description, parameters, and response parameters of time functions. This topic also provides examples of these functions.

## today

Details about this function:

- Syntax: `today()`.
- Description

  Queries the current date in the format of yyyy-mm-dd.
- Parameters

  None.
- Response parameters

  Returns the current date in the format of yyyy-mm-dd.
- Examples

```
say(concat('today:', today()))
Output:
today:2019-05-23
```

## time

Details about this function:

- Syntax: `time()`.
- Description

Queries the current UNIX timestamp, excluding the fractional part of milliseconds. Unit: seconds.

- Parameters

  None.

- Response parameters

  Returns the current UNIX timestamp.

- Examples

```
say(concat('time:', time()))
Output:
time:1559109666
```

## now

Details about this function:

- Syntax: `now()`.
- Description

  Queries the current UNIX timestamp, including the fractional part of milliseconds. Unit: seconds.

- Parameters

  None.

- Response parameters

  Returns the current UNIX timestamp.

- Examples

```
say(concat('now:', now()))
Output:
now:1559109666.644
```

## localtime

Details about this function:

- Syntax: `localtime()`.
- Description

  Queries the current UTC time in the format of yyyy-mm-dd hh:mm:ss.

- Parameters

  None.

- Response parameters

  Queries the current UTC time in the format of yyyy-mm-dd hh:mm:ss.

- Examples

```
say(concat('localtime:', localtime()))
Output:
localtime:2019-05-29 14:02:41
```

## utctime

Details about this function:

- Syntax: `utctime()`.
- Description

  Queries the current UTC time in the format of yyyy-mm-dd hh:mm:ss.
- Parameters

  None.
- Response parameters

  Queries the current UTC time in the format of yyyy-mm-dd hh:mm:ss.
- Examples

```
say(concat('utctime:', utctime()))
Output:
utctime:2019-05-29 06:02:41
```

## cookie_time

Details about this function:

- Syntax: `cookie_time(sec)`.
- Description

  Generates a time string in the cookie time format from a UNIX timestamp.
- Parameters

  sec: the UNIX timestamp. To obtain the UNIX timestamp, you can call the `time()` function.
- Response parameters

  Returns a time string in the cookie time format based on the specified `sec` parameter.
- Examples

```
say(concat('cookie_time:', cookie_time(time())))
Output:
cookie_time:Wed, 29-May-19 06:02:41 GMT
```

## http_time

Details about this function:

- Syntax: `http_time(sec)`.
- Description

Generates a time string in the HTTP header time format from a UNIX timestamp. For example, this function can generate the time displayed for the Last-Modified field in the HTTP header.

- Parameters

  sec: the UNIX timestamp. To obtain the UNIX timestamp, you can call the `time()` function.

- Response parameters

  Returns a time string in the HTTP header time format based on the specified `sec` parameter.

- Examples

```
say(concat('http_time:', http_time(time())))
Output:
http_time:Wed, 29 May 2019 06:02:41 GMT
```

## parse_http_time

Details about this function:

- Syntax: `parse_http_time(str)`.
- Description

  Parses a time string in the HTTP header time format and returns the corresponding UNIX timestamp.

- Field

  str: the time string in the HTTP header format that you want to parse. Example: `Thu, 22-Dec-10 10:20:35 GMT`. To obtain the time string, you can call the `http_time()` function.

- Response parameters

  If the function succeeds, a UNIX timestamp is returned. Otherwise, `false` is returned.

- Examples

```
say(concat('parse_http_time:', parse_http_time(http_time(time()))))
Output:
parse_http_time:1559109761
```

## unixtime

Details about this function:

- Syntax: `unixtime(year, month, day, hour, min, sec)`.
- Description

  Generates and returns a UNIX timestamp based on the provided values of the year, month, day, hour, min, and sec parameters.

- Parameters
  - year: specifies the year.
  - month: specifies the month.
  - day: specifies the day.

- hour: specifies the hour.
- min: specifies the minute.
- sec: specifies the second.

- Valid values:

  Returns the UNIX timestamp.

- Examples

```
t = unixtime(1970, 1, 1, 8, 0, 0)
say(concat('unixtime()=', t))
Output: unixtime()=0
```

# 4.10. Cipher algorithm functions

This topic describes the syntax, description, parameters, and response parameters of cipher algorithm functions. This topic also provides examples of these functions.

## aes_new

Details about this function:

- Syntax: `aes_new(config)`.
- Description

  Creates an AES object, which is used for the subsequent encryption and decryption operations. To perform an encryption, call the `aes_enc()` function. To perform a decryption, call the `aes_dec()` function.

- Parameters

  The `config` parameter is of the dictionary type and includes the following arguments:

  - key: the key, which is of the string type. This parameter is required.
  - salt: the salt value, which is of the string type. This parameter is optional.
  - cipher_len: the length of the key. This parameter is required. Valid values: 128, 192, and 256.
  - cipher_mode: the mode used for encryption or decryption. This parameter is required. Valid values: ecb, cbc, ctr, cfb, and ofb.
  - iv: the initial vector, which is of the string type. This parameter is optional.

- Response parameters

  If the function succeeds, an AES object (dictionary type) is returned. Otherwise, `false` is returned.

- Examples

```
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_enc

Details about this function:

- Syntax: `aes_enc(o, s)` .
- Description

  Encrypts data by using the AES encryption algorithm.
- Parameters
  - s: the plaintext to be encrypted.
  - o: the AES object returned by the `aes_new` function.
- Response parameters

  Returns the ciphertext after the text specified by the `s` parameter is encrypted.
- Examples

```
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))
```

```
  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

Details about this function:

- Syntax: `aes_enc(o, s)` .
- Description

  Decrypts data by using the AES encryption algorithm.
- Parameters
  - s: the ciphertext to be decrypted.
  - o: the AES object returned by the `aes_new` function.
- Response parameters

  Returns the plaintext after the text specified by the `s` parameter is decrypted.
- Examples

```
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

Details about this function:

- Syntax: `sha1(s)`.
- Description

  Calculates an SHA-1 value.
- Parameters

  s: the string for which you want to calculate an SHA-1 value.
- Response parameters

  Returns an SHA-1 value in binary format.
- Examples

```
digest = sha1('hello sha')
say(concat('sha1:', tohex(digest)))
Output:
sha1:853789bc783a6b573858b6cc9f913afe82962956
```

## sha2

Details about this function:

- Syntax: `sha2(s, l)`.
- Description

  Calculates an SHA-2 value.
- Parameters
  - s: the string for which you want to calculate an SHA-1 value.
  - l: the length of the SHA-2 value. Valid values: 224, 256, 384, and 512.
- Response parameters

  Returns an SHA-2 value in the binary format.
- Examples

```
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)))
Output:
sha2-224:b24b7effcf53ce815ee7eb73c7382613aba1c334e2a1622655362927
sha2-256:af0425cee23c236b326ed1f008c9c7c143a611859a11e87d66d0a4c3217c7792
sha2-384:bebbdde9efabd4b9cf90856cf30e0b024dd13177d9367d2dcf8d7a04e059f92260f16b21e261358c2271be32086ef35b
sha2-512:a1d1aef051c198c0d26bc03500c177a315fa248cea815e04fbb9a75e5be5061617daab311c5e3d0b215dbfd4e83e73f23081242b0143dcdfce5cd92ec51394f7
```

## hmac

Details about this function:

- Syntax: `hmac(k, s, v)`.
- Description

  Calculates an HMAC value.
- Parameters
  - k: the key of the algorithm.
  - s: the string for which you want to calculate an HMAC value.
  - v: the version of the algorithm. Valid values: md5, sha256, and sha512.
- Response parameters

  Returns an HMAC value in binary format by using the corresponding algorithm.
- Examples

```
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))))
Output:
hmac(md5): 358cbfca8ad663b547c83748de2ea778
hmac(sha1): 5555633cef48c3413b68f9330e99357df1cc3d93
hmac(sha256): 7a494543cad3b92ce1e7c4bbc86a8f5212b53e4d661f7830f455847540a85771
hmac(sha512): 59d7c07996ff675b45bd5fd40a6122bb5f40f597357a9b4a9e29da6f5c7cb806798c016fe09cb4
6457b6df9717d26d0af19896f72eaf4296be03e3681fea59ad
hmac_sha1(): 5555633cef48c3413b68f9330e99357df1cc3d93
```

## hmac_sha1

Details about this function:

- Syntax: `hmac_sha1(k, s)`.
- Description

  Calculates an HMAC-SHA-1 value.
- Parameters
  - s: the string for which you want to calculate an HMAC-SHA-1 value.
  - k: the HMAC-SHA-1 key.
- Response parameters

  Returns an HMAC-SHA-1 value in binary format.
- Examples

```
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))))
Output:
hmac(md5): 358cbfca8ad663b547c83748de2ea778
hmac(sha1): 5555633cef48c3413b68f9330e99357df1cc3d93
hmac(sha256): 7a494543cad3b92ce1e7c4bbc86a8f5212b53e4d661f7830f455847540a85771
hmac(sha512): 59d7c07996ff675b45bd5fd40a6122bb5f40f597357a9b4a9e29da6f5c7cb806798c016fe09cb4
6457b6df9717d26d0af19896f72eaf4296be03e3681fea59ad
hmac_sha1(): 5555633cef48c3413b68f9330e99357df1cc3d93
```

## md5

Details about this function:

- Syntax: `md5(s)`.
- Description

  Calculates an MD5 value.
- Parameters

  s: the string for which you want to calculate an MD5 value.
- Response parameters

  Returns an MD5 value in hexadecimal format.
- Examples

```
say(concat('md5: ', md5('hello md5')))
Output:
md5: 741fc6b1878e208346359af502dd11c5
```

## md5_bin

Details about this function:

- Syntax: `md5_bin(s)`.
- Description

  Calculates an MD5 digest.
- Parameters

  s: the string for which you want to calculate an MD5 value.
- Response parameters

Returns an MD5 digest in binary format.

- Examples

```
say(concat('md5_bin: ', tohex(md5_bin('hello md5'))))
Output:
md5_bin: 741fc6b1878e208346359af502dd11c5
```

# 4.11. JSON functions

This topic describes the syntax, description, parameters, return values, and examples of JSON functions.

## json_enc

This function is described as follows:

- Syntax: `json_enc(d)`
- Description

  You can call this function to encode a dictionary object into a JSON string.
- Parameters

  d: the dictionary object to be encoded.
- Return values

  This function returns a JSON-encoded string upon a success and returns `false` upon a failure.
- Examples

```
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')))
Output:
json_enc={"k1":"v1","k2":["v1","v2"]}
json_dec=v1
```

## json_dec

This function is described as follows:

- Syntax: `json_dec(s)`
- Description

  You can use this function to decode a JSON string.
- Parameters

  s: the JSON string to be decoded.

- Return values

  This function returns the decoded string upon a success and returns `false` upon a failure.

- Examples

```
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')))
Output:
json_enc={"k1":"v1","k2":["v1","v2"]}
json_dec=v1
```

# 4.12. Miscellaneous functions

This topic describes the syntax, description, parameters, and response parameters of miscellaneous functions. This topic also provides examples of these functions.

## base64_enc

Details about this function:

- Syntax: `base64_enc(s [, no_padding])` .
- Description

  Encodes a string in Base64.

- Parameters
  - s: the string that you want to encode.
  - no_padding: specifies whether to pad the string. A value of `true` indicates that the string is not padded. A value of `false` indicates that the string is padded. Default value: false.
- Response parameters

  Returns a Base64-encoded string.

- Examples

```
if $http_data {
  decdata = base64_dec($http_data)
  say(concat('base64_decdata=', decdata))
  say(concat('base64_encdata=', base64_enc('hello, dsl')))
}
Request header: "data: aGVsbG8sIGRzbA=="
Response: base64_decdata=hello, dsl
base64_encdata=aGVsbG8sIGRzbA==
```

## base64_dec

Details about this function:

- Syntax: `base64_dec(s)` .
- Description

  Decodes a Base64-encoded string.
- Parameters

  s: the string that you want to decode.
- Response parameters

  Returns a decoded raw string.
- Examples

```
if $http_data {
  decdata = base64_dec($http_data)
  say(concat('base64_decdata=', decdata))
  say(concat('base64_encdata=', base64_enc('hello, dsl')))
}
Request header: "data: aGVsbG8sIGRzbA=="
Response: base64_decdata=hello, dsl
base64_encdata=aGVsbG8sIGRzbA==
```

## url_escape

Details about this function:

- Syntax: `url_escape(s)` .
- Description

  Encodes a string in URL.
- Parameters

  s: the string that you want to encode.
- Response parameters

  Returns a URL-encoded string.

- Examples

```
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))
}
Output: raw=/abc/123/ dd/file.m3u8
esdata=%2Fabc%2F123%2F%20dd%2Ffile.m3u8
dsdata=/abc/123/ dd/file.m3u8
```

## url_unescape

Details about this function:

- Syntax: `url_unescape(s)`.
- Description

  Decodes a URL-encoded string.
- Parameters

  s: the string that you want to decode.
- Response parameters

  Returns a decoded raw string.
- Examples

```
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))
}
Output: raw=/abc/123/ dd/file.m3u8
esdata=%2Fabc%2F123%2F%20dd%2Ffile.m3u8
dsdata=/abc/123/ dd/file.m3u8
```

## rand

Details about this function:

- Syntax: `rand(n1, n2)`.
- Description

Generates and returns a random number. Value values: n1 <= returned number<= n2. The n1 parameter specifies the smallest number. The n2 parameter specifies the greatest number.

- Parameters
  - n1: the smallest number.
  - n2: the greatest number.
- Response parameters

  Returns a random number.
- Examples

```
r = rand(1,100)
```

## rand_hit

Details about this function:

- Syntax: `rand_hit(ratio)`.
- Description

  Retrieves a value of true or false based on the specified probability.
- Parameters

  ratio: the probability. Valid values: 0 to 100.
- Response parameters

  Returns `true` or false based on the specified probability. If you set ratio to 100, `true` is returned. If you set ratio to 0, `false` is returned.
- Examples

```
rand_hit(80)
```

## crc

Details about this function:

- Syntax: `crc(s)`.
- Description

  Calculates a Cyclic Redundancy Check (CRC) value.
- Parameters

  s: the string for which you want to calculate a CRC digest.
- Response parameters

  Returns the CRC value of the string specified by the `s` parameter.
- Examples

```
crc('hello edgescript')
```

## tonumber

Details about this function:

- Syntax: `tonumber(s [, base])` .
- Description

  Converts a string to numeric values.
- Parameters
  - s: the string that you want to convert.
  - base: the positional notation that you want to use to convert the string. Valid values: 10 and 16. Default value: 10.
- Examples

```
n = tonumber('100')
say(concat('tonumber()=', n))
Output: tonumber()=100
```

# 5.EdgeScript scenarios

This topic describes the application scenarios of EdgeScript, including authentication logic customization, request header and response header customization, rewrite and redirect customization, M3U8 rewrite customization, cache control customization, and throttling customization.

## Customize the authentication logic

The following example shows custom authentication algorithms:

- Requirements
  - Request URL format: `/path/digest/?.ts? key=&t=`
  - For `.ts` requests, the requirements for customizing hotlinking protection are:
    - Rule 1: If the request does not contain the `t` or `key` parameter, the CDN node returns the 403 status code and adds the `X-AUTH-MSG` response header to indicate the failure cause.
    - Rule 2: The `t` parameter specifies the expiration time. If the specified `t` parameter is earlier than the current time, the CDN node returns the 403 status code and adds the `X-AUTH-MSG` response header to indicate the failure cause.
    - Rule 3: The CDN node compares the `md5` parameter with the `digest` parameter. If `md5` does not match `digest`, the CDN node returns the 403 status code.

      Value format of the md5 parameter: `Private key + Path + File name.extension`.
- Corresponding EdgeScript script:

```
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)
  }
}
```

## Customize request headers and response headers

The following example shows automatic file renaming:

- Requirements

  If the `filename` parameter is set, the file is automatically renamed the value specified by the `filename` parameter. If no file name is specified, the default file name is used.

- Corresponding EdgeScript script:

```
// The value for the filename parameter is enclosed in a pair of double quotation marks (""). The string "34
" is the ASCII string for double quotation marks. It can be converted back to the quotation mark string ("")
through the tochar function.
// Example: add_rsp_header('Content-Disposition', concat('attachment;filename=', tochar(34), filename, t
ochar(34)))
// Output: Content-Disposition: attachment;filename="monitor.apk"
if $arg_filename {
  hn = 'Content-Disposition'
  hv = concat('attachment;filename=', $arg_filename)
  add_rsp_header(hn, hv)
}
```

## Customize rewrites or redirects

The following examples show how to customize rewrites and redirects:

- Rewrite a URI.
  - Requirements

    Configure Alibaba Cloud Content Delivery Network (CDN) to rewrite `/hello` as `/index.html`. As a result, the URI of the back-to-origin request is changed to `/index.html` and the parameters remain unchanged.
  - Corresponding EdgeScript script:

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

- Rewrite a file extension.
  - Requirements

    Configure Alibaba Cloud CDN to rewrite `/1.txt` as `/1<url parameter type>`. For example, `/1.txt?type=mp4` will be rewritten as `/1.mp4?type=mp4` in back-to-origin requests and cached on CDN nodes.
  - Corresponding EdgeScript script:

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

- Convert a file extension to lowercase letters.
  - Requirements

    Convert the URI string to lowercase letters.

- Corresponding EdgeScript script:

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

- Add a URI prefix.
  - Requirements

    Configure Alibaba Cloud CDN to rewrite `^/nn_live/(. *)` as `/3rd/nn_live/$1`.
  - Corresponding EdgeScript script:

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

- Set a 302 redirect.
  - Requirements

    Perform a 302 redirect to `/app/movie/pages/index/index.html` for the `/` root directory.
  - Corresponding EdgeScript script:

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

- Set a 302 redirect to HTTPS URIs
  - Requirements

    Redirect the following URIs that match the `^/$` root directory to `https://rtmp.cdnpe.com/index.html`. You can specify the destination URI as needed.
    - `http://rtmp.cdnpe.com`
    - `https://rtmp.cdnpe.com`
  - Corresponding EdgeScript script:

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

## Customize cache control

The following example shows how to customize the cache duration:

- Requirements

  Customize the resource cache duration based on various conditions.

- Corresponding EdgeScript script:

```
// Note: Set the cache duration for status codes responded to requests whose URIs start with "/image". Se
t 10 seconds for the 301 status code and 5 seconds for the 302 status code.
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)
}
```

## Customize throttling policies

The following example shows how to customize a throttling policy:

- Requirements

  If the `sp` and `unit` parameters are set, throttling is implemented. The `sp` parameter specifies the throttling threshold. The `unit` parameter specifies the unit. The unit can be KB or MB.

- Corresponding EdgeScript script:

```
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
}
```