ALIBABA CLOUD

Alibaba Cloud

API Gateway

Plugins

Document Version: 20211123

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.Overview

Plug-ins are only available for API operations on shared and dedicated instances in virtual private clouds (VPCs). You cannot use plug-ins for API operations on shared instances on the classic network.

> **Note**
>
> The latest version of API Gateway that was released in 2019 provides plug-ins to support existing features such as throttling, IP address-based access control, backend signature, and JSON Web Token (JWT) authorization. Plug-ins of the JWT Authorization type function the same as the OpenId Connect feature. In addition, plug-ins also support the following new features: cross-origin resource sharing (CORS), caching, routing, parametric access control, error mapping, and circuit breakers. API Gateway will provide more plug-ins in the future.

1. Usage notes

- Only one plug-in of each type can be bound to an API operation.
- You can bind a plug-in only to an API operation that is in the same region as the plug-in. Each user can create a maximum of 1,000 plug-ins in each region.
- Plug-in configurations and configurations of API operations are managed separately. Configurations of a plug-in take effect only after you bind the plug-in to a published API operation.
- Before you bind a plug-in to an API operation, you must publish the API operation.
- A plug-in can be bound, unbound, and updated with immediate effect. You do not need to re-publish the relevant API operation. For an API operation that requires high security, we recommend that you publish the API operation to the test environment and test plug-ins on the API operation first.
- After an API operation is unpublished, the plug-in that is bound to the API operation is still bound. When the API operation is re-published, the plug-in still takes effect.
- If a plug-in is bound to a published API operation or an API operation that is unpublished but not deleted, you cannot delete the plug-in.

2. Plug-ins supported by API Gateway

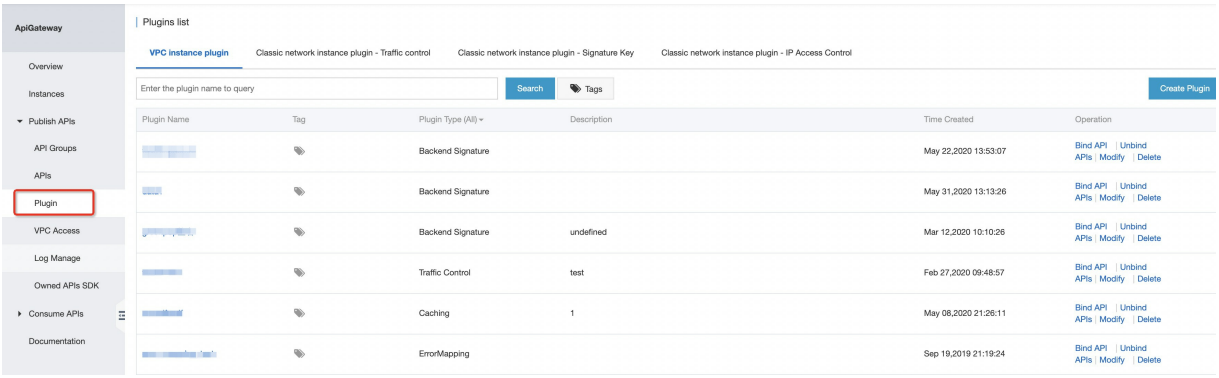
API Gateway supports the following types of plug-ins. You can click each plug-in type to view information about the plug-in type.

- Throttling
- IP address-based access control
- Plug-ins of the Parametric Access Control type
- Backend signature
- JWT authentication
- CORS
- Plug-ins of the Caching type
- routing plug-in
- Plug-ins of the Error Mapping type

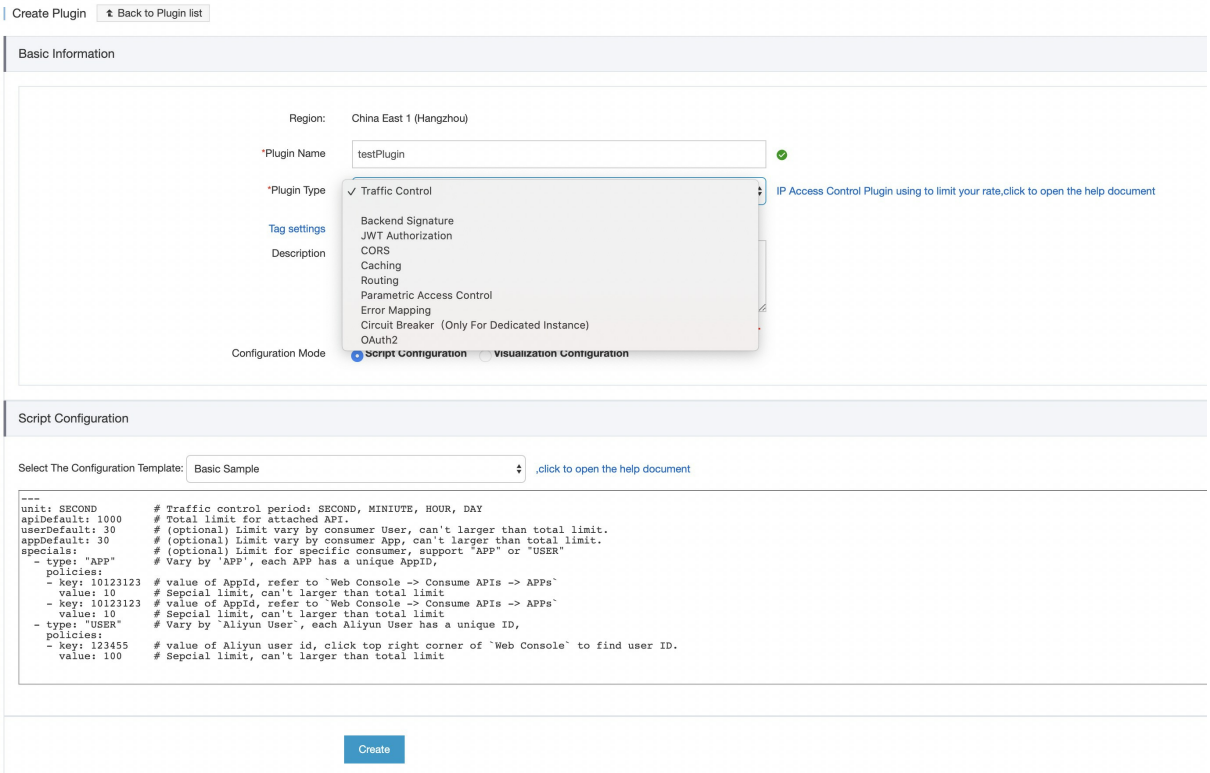
- Plug-ins of the Circuit Breaker type (only available for API operations on dedicated instances)

3. Quick start

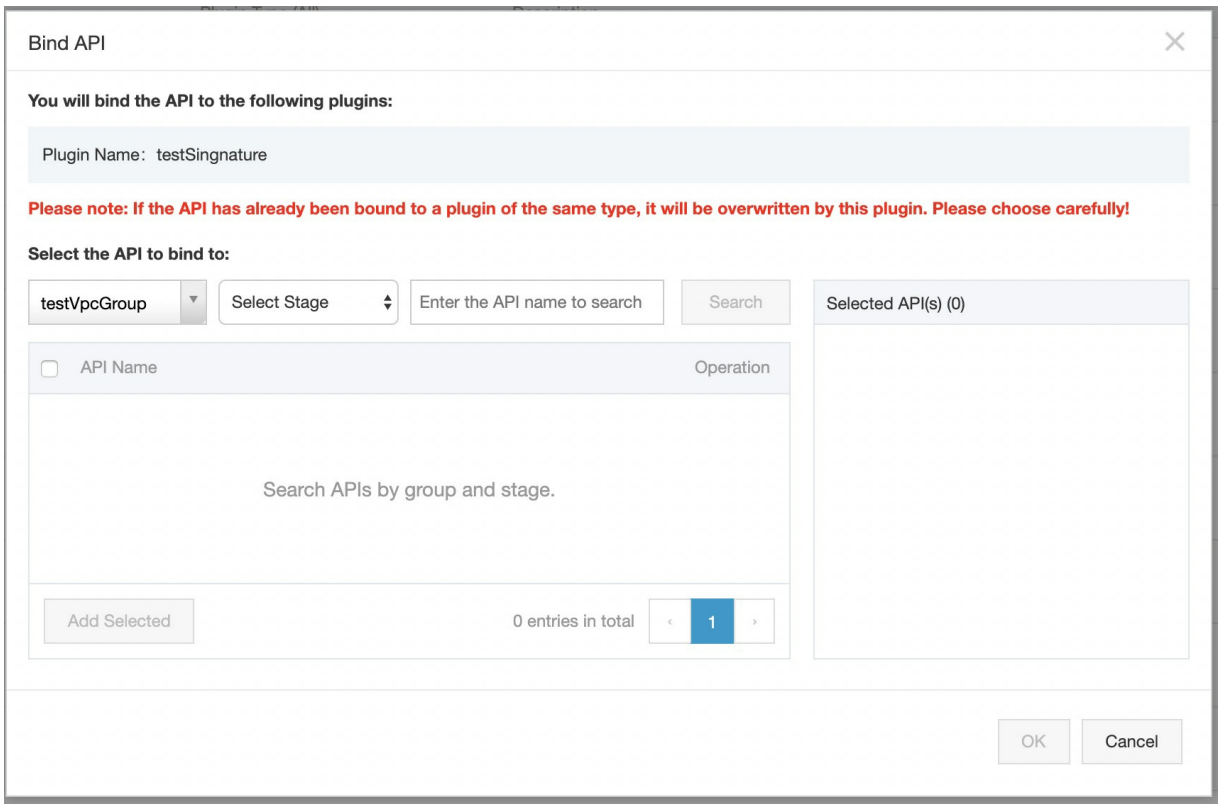
- Log on to the API Gateway console. In the left-side navigation pane, choose Publish APIs > Plugin.



- On the Plugin list page, click Create Plugin. On the Create Plugin page, set the parameters as required and click Create.



- After you create the plug-in, the plug-in appears on the Plugin list page. Find the plug-in and click Bind API in the Operation column.



- The plug-in takes effect immediately after you bind it to an API operation.

4. API reference

You can use the following API operations to manage plug-ins in API Gateway:

- Create a plug-in: Create an Plugin
- Modify a plug-in: Modify an Plugin
- Delete a plug-in: Delete the Plugin
- Query plug-ins: Describe an Plugin
- Bind a plug-in to an API operation: Attach Plugin
- Unbind a plug-in from an API operation: Detach Plugin
- Query the API operations that are bound to a plug-in: API for binding under query Plug-ins
- Query the plug-ins that are bound to an API operation: Describe Plugins by API

5. Limits

- For each plug-in, you can configure a maximum of 16,380 bytes of metadata.
- Each user can create a maximum of 1,000 plug-ins in each region.

2.Backend signature

A backend signature (formerly known as a signature key) is a key-secret pair that you create and issue to API Gateway. It works in a way similar to an account and password pair. After a backend signature plug-in is bound with an API, API Gateway uses the key-secret pair to sign requests designated for your backend service. Your backend service also performs symmetric encryption on the received requests. API Gateway passes authentication if the signature on API Gateway is consistent with that on the backend service.

1. What is a backend signature?

A backend signature (formerly known as a signature key) is a key-secret pair that you create and issue to API Gateway. It works in a way similar to an account and password pair. When API Gateway sends a request to your backend service, API Gateway uses the backend signature to calculate a signature string and pass it to your backend service. Your backend service obtains the signature string and authenticates API Gateway by using symmetric calculation. If your backend service accesses API Gateway over a VPC, you do not need to use backend signatures because the transmission channel is secure.

The original signature key feature has been integrated into the plug-in system. The original signature key interface and console are still in use. The original signature key feature and the backend signature plug-in are of the same plug-in type and are subject to the binding limits of that type.

When you create or modify keys in the original signature key interface or console, the data modifications are synchronized to the plug-in system. However, the modifications you make in the plug-in system cannot be synchronized to the original signature key interface or console.

2. Plug-in binding

After you bind a key to an API, API Gateway contains signature information in all the requests sent to your backend service. The backend service performs symmetric calculation to parse the signature information and authenticate API Gateway.

If you want to replace the key bound to an API, modify the key and secret in the backend signature plug-in that is bound to the API. The new key takes effect immediately after it is bound to the API.

3. Plug-in configurations

You can configure backend signature plug-ins in JSON or YAML format because these two formats use the same schema. You can use the yaml to json tool to convert the configuration format of a backend signature plug-in. The following example describes a plug-in configuration template in YAML format:

```
---
type: APIGW_BACKEND
key: SampleKey
secret: SampleSecret
```

4. Storage of the backend signatures of API Gateway

Save the signature information calculated by API Gateway into the X-Ca-Proxy-Signature header of a request.

5. Backend signature rules

For more information about the JAVA demo for signature calculation, visit https://github.com/aliyun/api-gateway-demo-sign-backend-java.

Follow these steps to perform signature calculation:

1. API Gateway extracts key data from the HTTP requests that are sent to your backend service and combines the data into a signature string. The generated signature string is in the following format:

```
HTTPMethod
Content-MD5
Headers
PathAndParameters
```

The preceding four fields constitute an entire signature string and are separated with \n. If the Headers field is left empty, \n is not required. If the other fields are left empty, \n must be retained. The name of the signature string is case-sensitive. The following content describes the data extraction rules for each field:

- HTTPMethod: the HTTP method used to send a request, such as POST. The field value is in uppercase.
- Content-MD5: the value of the Content-MD5 header in a request. This field can be left empty. The value is calculated only when a request contains a body of a non-Form type. The following example is used to calculate the value of the Content-MD5 header in Java format:

```
String content-MD5 = Base64.encodeBase64(MD5(bodyStream.getbytes("UTF-8")));
```

- Headers: specifies the keys and values of all headers involved in signature calculation. You can read the keys named X-Ca-Proxy-Signature-Headers from the headers of requests. The keys are separated with commas (,). The keys involved in signature calculation are sorted in alphabetical order, converted into lowercase letters, and then combined based on the following rules:

```
String headers = HeaderKey1.toLowerCase() + ":" + HeaderValue1 +"\n"+
 HeaderKey2.toLowerCase() + ":" + HeaderValue2 +"\n"+
 ... +
HeaderKeyN.toLowerCase() + ":" + HeaderValueN + "\n"
```

- PathAndParameters

This field contains all the path, query, and form parameters. To form a signature string with query and form parameters, add a question mark (?), sort the keys of the query and form parameters in alphabetical order, and combine the keys based on the following rules. If the signature is not formed with query or form parameters, set PathAndParameters to Path.

```
String PathAndParameters =
 Path +
 "?" +
 Key1 + "=" + Value1
+ "&" + Key2 + "=" + Value2 +
 ...
"&" + KeyN + "=" + ValueN
```

Note: A query or form parameter may have multiple values. Only the first value is used for signature calculation. The equal sign (=) in the signature calculation process must be retained for the parameters that have been transferred. For example, "path?a=&b" must be written as "path?a=&b=" in the request signature process.

2. Calculate a signature.

```
Mac hmacSha256 = Mac.getInstance("HmacSHA256");
byte[] keyBytes = secret.getBytes("UTF-8"); //The secret value is the secret of the signature key bound to th
e API.
hmacSha256.init(new SecretKeySpec(keyBytes, 0, keyBytes.length, "HmacSHA256"));
String sign = new String(Base64.encodeBase64(Sha256.doFinal(stringToSign.getBytes("UTF-8")),"UTF-8"));
```

Decode stringToSign by using UTF-8 to obtain a Byte array. Then use an encryption algorithm to encrypt the Byte array, and Base64 encode the encrypted Byte array to form a final signature.

6. Debug mode

To access and debug a backend signature efficiently, you can enable the Debug mode. The debugging procedure is to add X-Ca-Request-Mode = debug to the header of a request that is sent to API Gateway.

The backend service reads X-Ca-Proxy-Signature-String-To-Sign from the header of a request. Line breaks are not allowed in the value of an HTTP request header and are replaced with vertical bars (|).

Note: X-Ca-Proxy-Signature-String-To-Sign is not involved in backend signature calculation.

7. Timestamp verification

If your backend service needs to verify the timestamp of a request, set CaRequestHandleTime to the Greenwich Mean Time (GMT) when API Gateway receives the request.

3.Use parameters and conditional expressions

1. Overview

In `an access control plug-in` , `a throttling plug-in` , `a backend routing plug-in` , or `an error code mapping plug-in` , you can obtain parameters from `requests` , `responses` , and `system context` . Then, you can use **conditional expressions** to perform judgment on these parameters. This topic describes how to define **parameters** and write **conditional expressions**.

2. Define parameters

2.1. Definition method

Before you use a conditional expression, you must explicitly define all the parameters that are required in this conditional expression in the `parameters` field. Example:

```
---
parameters:
 method: "Method"
 appId: "System:CaAppId"
 action: "Query:action"
 userId: "Token:UserId"
```

The parameters specified in the `parameters` field are `key` - `value` pairs of the STRING type.

- `key` indicates the name of a variable to be used in a conditional expression. The name must be unique and must conform to the following regular expression: `[a-zA-Z_][a-zA-Z0-9]+` .
- `value` indicates the location of a parameter. It is specified in the `{location}` or `{location}:{name}` format.
- ○ `location` indicates the location of a parameter. For more information, see the following table.
  - ○ `name` indicates the name of a parameter. The name is used to locate the parameter at a specific location. For example, `Query:q1` indicates the first value of the parameter that is named q1 in the query string.

2.1. Parameter locations

Before you use a conditional expression, you must define the parameters that are required in this conditional expression. The following table describes parameters at specific locations that the plug-ins of API Gateway can use.

Location	Scope	Description
Method	Requests	The HTTP request method, in uppercase. Examples: `GET` and `POST` .
Path	Requests	The full path of the HTTP request. Example: `/path/to/query`
StatusCode	Responses	The HTTP response code in a backend response. Examples: `200` and `400`
ErrorCode	Responses	The error code of a system error response in API Gateway. For more information, see Error codes.
Header	Requests or responses	Use `Header:{Name}` to obtain the first value of the header whose name is `{Name}` .
Query	Requests	Use `Query:{Name}` to obtain the first value of the parameter whose name is `{Name}` in the query string.
Form	Requests	Use `Form:{Name}` to obtain the first value of the parameter whose name is `{Name}` in the request form.

Location	Scope	Description
Host	Requests	Use `Host:{Name}` to obtain the template parameters of the matched wildcard domain names.
Parameter	Requests	Use `Parameter:{Name}` to obtain the first value of the custom API parameter whose name is `Name`.
BodyJsonField	Responses	Use `BodyJson:{JPath}` to obtain the value of the parameters in the JSON-formatted body of an API request or a backend response. `JPath` is a JSONPath expression.
System	Requests or responses	Use `System:{Name}` to obtain the value of the system parameter whose name is `{Name}`.
Token	Requests or responses	If `JSON Web Token (JWT)` is used with `OAuth2` for authentication, you can use `Token:{Name}` to obtain the value of the parameter whose name is `{Name}` in a token.

Usage notes of parameter locations

- If you use an access control plug-in, a throttling plug-in, or a backend routing plug-in at the request phase, you can use only the parameters at the following locations: `Method`, `Path`, `Header`, `Query`, `Form`, `Parameter`, `System`, and `Token`.
- You can also use the error code mapping plug-in at the response phase. This plug-in supports only the parameters at the following locations: `StatusCode`, `ErrorCode`, `Header`, `BodyJsonField`, `System`, and `Token`.
- Parameters at the `Method`, `Path`, `StatusCode`, and `ErrorCode` locations are defined in the {location} format.
- If you use parameters at the `Header` location in a plug-in at the request phase, headers that are in the requests from the client are read. If you use these parameters at the response phase, headers from backend responses are read.
- Parameters at the `Parameter` location are available only for plug-ins at the request phase. A `frontend parameter`, instead of a `backend parameter`, is used to search for the parameter with the same name in the API definition. If no parameter with the same name exists, a `null` value is returned.
- Parameters at the `Host` location are available only when you obtain the parameters of wildcard domain names. For more information, see Bind a domain name to an API group. To obtain the system parameters of full domain names, use `System:CaDomain`.
- A complete request path is returned from `Path`. If you require a parameter at the `Path` location, use the corresponding parameter at the `Parameter` location.
- Parameters at the `BodyJsonField` location are available only for the error code mapping plug-in. Obtain the parameters in the JSON-formatted body of a backend response in `JSONPath` mode. For more information, see 2.4. Usage notes of JSONPath.
- If a plug-in of the JWT Authorization type is used for authentication, use `Token:{CliamName}` to obtain the value of the parameter specified by `{CliamName}` in a token. For more information, see JWT authentication.

2.3. Usage notes of JSONPath

You can use JSONPath expressions to obtain parameters at the `BodyJsonField` location only for the error code mapping plug-in. The parameters are in the `JSON-formatted` body returned in a backend response. For more information about JSONPath expressions, see JSONPath.

- Example: Use `code:"BodyJsonField:$.result_code"` to obtain the value of the `result_code` parameter in the following body. The parsing result of the body is `code`: `ok`.

```
{"result_code": "ok", "message": ...}
```

2.4. System parameters

Parameter	Description	Valid value or sample value
CaClientIp	The IP address of the client from which the request is sent.	Sample value: `37.78.3.3` or `fe80::1849:59fd:993c:fcff`
CaDomain	The full domain name that is specified after the Host header in a request.	Sample value: `api.foo.com`
CaAppId	The ID of the application that sends the request.	Sample value: `49382332`
CaAppKey*	The key of the application that sends the request.	Sample value: `12983883923`
CaRequestId	The unique request ID that API Gateway generates.	Sample value: `CCE4DEE6-26EF-46CB-B5EB-327A9FE20ED1`
CaApiName	The API name	Sample value: `TestAPI`
CaHttpSchema	The protocol that is used to send the request.	Valid values: `http`, `https`, and `ws`.
CaClientUa	The UserAgent header of the client.	Pass the value that is uploaded by the client.

Parameter	Description	Valid value or sample value
CaCloudMarketInstanceId	The ID of an instance on Alibaba Cloud Marketplace.	You can obtain the ID on Alibaba Cloud Marketplace.
CaMarketExpriencePlan	Specifies whether to activate an experience plan on Alibaba Cloud Marketplace.	If you want to activate an experience plan, set the parameter to `true`. If you do not want to activate an experience plan, set the parameter to `false`.

3. Conditional expressions

You can use conditional expressions for judgment in scenarios such as plug-ins.

3.1. Syntax

- Conditional expressions are similar to SQL statements. Example: `$A > 100 and '$B = 'B'`.
- An expression is in the following format: `{Parameter} {Operator} {Parameter}`. In the preceding example, you can specify a `variable` or a `constant` for `$A > 100`.
- A `variable` starts with `$` and references a parameter defined in the context. For example, `q1:"Query:q1"` is defined in `parameters`. You can use the variable `$q1` in your expression. The value of this variable is the value of the `q1` query parameter in the request.
- A `constant` can be a `string`, a `number`, or a `boolean value`. For example, the constant can be `"Hello"`, `'foo'`, `100`, `-1`, `0.1`, or `true`. For more information, see 3.2. Value types and judgment rules.
- The following `operators` are supported:
- ○ `=` and `==`: equal to.
  - ○ `<>` and `!=`: not equal to.
  - ○ `>`, `>=`, `<`, and `<=`: comparison.
  - ○ `like` and `!like`: check whether a specific string matches a specified pattern. The percent sign `%` is used as a wildcard in the judgment. Example: `$Query like 'Prefix%'`.

- `in_cidr` and `!in_cidr` : check whether an IP address is in a CIDR block. Example:

  `$ClientIp in_cidr '47.89.0.0/24'`

- You can use `null` to check whether a parameter is empty. Example: `$A == null` or `$A != null`
- You can use the operators `and` , `or` , and `xor` to combine different expressions in a right-to-left order by default.
- You can use parentheses `( )` to specify the priority of conditional expressions.
- You can use `!( )` to perform the logical negation operation on the enclosed expression. For example, the result of `!(1=1)` is `false` .
- The following built-in functions are used for judgment in special scenarios:
- - `Random()` : generates a parameter whose value is a floating-point number from `0 to 1` . This parameter is used in scenarios where random input is required, such as blue-green release.
  - `Timestamp()` : returns a `UNIX timestamp` . The UNIX timestamp is the number of milliseconds that have elapsed since 00:00:00 Thursday, 1 January 1970.
  - `TimeOfDay()` : returns the number of milliseconds from the current time to 00:00 of the current day in `GMT` .

3.2. Value types and judgment rules

- The following value types are supported in expressions:
  - `STRING` : The value is a string enclosed in single quotation marks (')or double quotation marks ("). Examples: `"Hello` and `'Hello'` .
  - `NUMBER` : The value is an integer or a floating-point number. Examples: `1001` , `-1` , `0.1` , and `-100.0` .
  - `BOOLEAN` : The value is a boolean value. Examples: `true` and `false` .
- For the operator types `equal to` , `not equal to` , and `comparison` , the following judgment rules apply:
  - `STRING` : Perform judgment based on the string order. Examples:

- ■ `'123' > '1000'` : The result is true.
- ■ `'A123' > 'A120'` : The result is true.
- ■ `'' < 'a'` : The result is true.
- `NUMBER` : Perform judgment based on numeric values. Examples:
- ■ `123 > 1000` : The result is false.
- ■ `100.0 == 100` : The result is true.
- `BOOLEAN` : For Boolean values, `true` is greater than `false` . Examples:
- ■ `true == true` : The result is true.
- ■ `false == false` : The result is true.
- ■ `true > false` : The result is true.

- For the operator types `equal to` , `not equal to` , and `comparison` , if the value types before and after an operator are different, the following judgment rules apply:
  - `STRING` `NUMBER` : For example, the value before an operator is of the STRING type and that after the operator is of the `NUMBER` type. If the value type before the operator can be changed to NUMBER, use numerical values for judgment. Otherwise, use the string order for judgment. Examples:
  - ■ `'100' == 100.0` : The result is true.
  - ■ `'-100' > 0` : The result is false.
  - `STRING` `BOOLEAN` : For example, the value before an operator is of the STRING type and that after the operator is of the BOOLEAN type. If the value type before the operator can be changed to `BOOLEAN` and the value is not case-sensitive, use `BOOLEAN` values, including `true` and `false` , for judgment. Otherwise, except for the judgment result of `!=` , all the other judgment results are `false` . Examples:

- `'True' == true`: The result is true.
- `'False' == false`: The result is true.
- `'bad' == false`: The result is false.
- `'bad' != false`: The result is true. If the value before the operator is not `true` or `false`, only the result for `!=` is `true`.
- `'bad' != true`: The result is true.
- `'0' > false`: The result is false.
- `'0' <= false`: The result is false.

- `NUMBER BOOLEAN`: If the value before an operator is of the NUMBER type and that after the operator is of the BOOLEAN type, the result is `false`.

- The `null` value is used to check whether a parameter is empty. For the operator types `equal to`, `not equal to`, and `comparison`, the following judgment rules apply:
  - If the `$A` variable is empty, the result of `$A == null` is true, and the result of `$A !=null` is false.
  - If the empty string `''` is not equal to `null`, the result of `'' == null` is `false`, and the result of `'' == ''` is `true`.
  - For the `comparison` operator type, if the value on either side of the operator is `null`, the result is `false`.
- `like` and `!like` operators are used to match the prefix, suffix, and inclusion of a string. The following judgment rules apply:
  - In an expression, the value after the operator must be a constant of the `STRING` type. Example: `$Path like '/users/%'`.
  - The `'%'` wildcard character in the value after the operator is used to match the prefix, suffix, or inclusion of a string. Examples:
    - Prefix matching: `$Path like '/users/%'` and `$Path !like '/admin/%'`
    - Suffix matching: `$q1 like '%search'` and `$q1 !like '%.do'`,
    - Inclusion relation matching: `$ErrorCode like '%400%'` and `$ErrorCode !like '%200%'`,

- If the value type before an operator is not `NUMBER` or `BOOLEAN`, change the type to `STRING` and perform the judgment.
- If the value before an operator is `null`, the result is `false`.

- `in_cidr` and `!in_cidr` operators are used to check whether an IP address is in a CIDR block. The following judgment rules apply:
  - The value after an operator must be a constant of the `STRING` type and must be an IPv4 or IPv6 CIDR block. Examples:
  - `$ClientIP in_cidr '10.0.0.0/8'`
  - `$ClientIP !in_cidr '0:0:0:0:0:FFFF::/96'`
  - If the value type before an operator is `STRING`, the value is considered as an IPv4 CIDR block for judgment.
  - If the value type before an operator is `NUMBER` or `BOOLEAN` or the value is empty, the result is `false`.
  - The `System:CaClientIp` parameter specifies the IP address of the client. This parameter can be used for judgment.

4. Use cases

- Check whether the probability is less than 5%:

```
Random() < 0.05
```

- Check whether the requested API operation is published to the test environment.

```
parameters:
 stage: "System:CaStage"
```

```
$CaStage = 'TEST'
```

- Check whether the custom parameter UserName is set to Admin and the source IP address is in `47.47.74.0/24`.

```
parameters:
 UserName: "Token:UserName"
 ClientIp: "System:CaClientIp"
```

```
$UserName = 'Admin' and $CaClientIp in_cidr '47.47.74.0/24'
```

- If the result of the following expression is true, the CaAppId parameter is set to 1001, 1098, or 2011, and the protocol that is used by the API request is HTTPS.

```
parameters:
 CaAppId: "System:CaAppId"
 HttpSchema: "System:CaHttpSchema"
```

```
$CaHttpScheme = 'HTTPS' and ($CaAppId = 1001 or $CaAppId = 1098 or $CaAppId = 2011)`
```

- If the result of the following expression is true, the `JSON-formatted` body contains the `result_code` parameter whose value is not `ok` when StatusCode in a response is 200.

```
parameters:
 StatusCode: "StatusCode"
 ResultCode: "BodyJsonField:$.result_code"
```

```
$StatusCode = 200 and ($ResultCode <> null and $ResultCode <> 'ok')
```

5. Limits

- A maximum of **16** parameters can be specified in a plug-in.
- A single conditional expression can contain a maximum of **512** characters.
- The size of a request or response body specified by BodyJsonField cannot exceed **16 KB**. Otherwise, the settings will not take effect.

4.Throttling

1. Overview

- A throttling plug-in is used to control traffic from the perspectives of `APIs`, `apps`, `users who own the apps`, and `custom parameters`. You can control the traffic of apps based on their AppKeys.
- The `throttling plug-in` supports two configuration templates:
  - **Configuration template for parameter-based throttling**: This template is used to configure `custom parameter`-based throttling.
  - **Configuration template for basic throttling**: This template is compatible with the `throttling` feature in the API Gateway console.
- `Throttling` is integrated into the plug-in system. The original `throttling` interface and console are still in use. `Throttling policies` and `throttling plug-ins` belong to the same plug-in type. If you bind a `throttling plug-in` to an API, the `throttling policies` that are bound to this API become invalid.
- When you create or modify throttling policies in the original throttling interface or console, modifications are synchronized to the plug-in system. However, the modifications you made in the plug-in system cannot be synchronized to the throttling interface or console.

2. Basic throttling configuration

2.1. Throttling thresholds

Basic throttling supports the following throttling thresholds:

- **API-level throttling threshold**: the maximum number of times that an API bound with a throttling policy can be called within a specific unit of time. This unit of time can be second, minute, hour, or day. For example, you can set this threshold to 5,000 times per minute.
- **App-level throttling threshold**: the maximum number of times that each app can call an API bound with a throttling policy within a specific unit of time. For example, you can set this threshold to 50,000 times per hour.
- **User-level throttling threshold**: the maximum number of times that each Alibaba Cloud account can call an API bound with a throttling policy within a specific unit of time. An Alibaba Cloud account may have multiple apps. This throttling threshold limits the total number of API calls that all the apps under this account can initiate. For example, you can set this threshold to 500,000 times per day.

You can specify all the preceding thresholds in one throttling policy. The user-level throttling threshold cannot be greater than the API-level throttling threshold. The app-level throttling threshold cannot be greater than the user-level throttling threshold.

You can also add special apps or users to a throttling policy. Only the API-level throttling threshold in the throttling policy applies to these special apps or users. However, you still need to set a special throttling threshold for each of them. This special throttling threshold cannot be greater than the API-level throttling threshold.

2.2. Plug-in configuration

You can configure a throttling plug-in in JSON or YAML format. The two formats use the same schema. You can use the `YAML to JSON` converter to convert the configuration format of a throttling plug-in. The following code describes a plug-in configuration template in YAML format:

```
---
unit: SECOND       # The unit of time. Valid values: SECOND, MINUTE, HOUR, and DAY.
apiDefault: 1000    # The default API-level throttling threshold.
userDefault: 30     # Optional. The default user-level throttling threshold. If you set this threshold to 0, user-level throttling is not performed. The user-level throttling threshold cannot be greater than the API-level throttling threshold.
appDefault: 30      # Optional. The default app-level throttling threshold. If you set this threshold to 0, app-level throttling is not performed. The app-level throttling threshold cannot be greater than the user-level throttling threshold.
specials:         # Optional. The special throttling settings. You can set throttling thresholds for special apps or users in a throttling policy.
 - type: "APP"     # The special throttling type. The value APP indicates that throttling is performed for special apps based on their AppKeys.
   policies:
   - key: 10123123  # The app ID. You can obtain the ID of an app from the app details page. To go to this page, click Consume APIs and then APPs in the left-side navigation pane of the API Gateway console and click the name of the app.
     value: 10     # The special throttling threshold for the app. This threshold cannot be greater than the user-level throttling threshold in the throttling policy.
   - key: 10123123  # The app ID.
     value: 10     # The special throttling threshold for the app. This threshold cannot be greater than the user-level throttling threshold in the throttling policy.
 - type: "USER"     # The special throttling type. The value USER indicates that throttling is performed for special Alibaba Cloud accounts.
   policies:
   - key: 123455    # The ID of an Alibaba Cloud account. You can move the pointer over the profile picture in the upper-right corner of the Alibaba Cloud Management Console to obtain the ID.
     value: 100     # The special throttling threshold for the Alibaba Cloud account. This threshold cannot be greater than the API-level throttling threshold in the throttling policy.
```

3. Parameter-based throttling configuration

Parameter-based throttling allows you to control traffic based on the request parameters of users and conditional executions. The following parameter-based throttling configurations are supported:

- Throttling at the second, minute, hour, or day level is supported.
- You can use request and system parameters to set conditions and implement different throttling thresholds.
- You can configure throttling by using a single parameter or a combination of multiple parameters.
- You can set the throttling scope to API or PLUGIN.

3.1. Quick start

Use a scenario where the following throttling rules apply as an example: If the AppKey of the app with the ID of `10001` is used as the signature information of a user, the throttling threshold for each client IP address is `100 requests per second`. Otherwise, the throttling threshold is `10 requests per second`.

For this scenario, a throttling plug-in in `YAML` format is configured.

```
---
scope: "PLUGIN"
#
# Throttling depends on two system parameters.
# 1. The app ID signed by the user is obtained from the CaAppId system parameter.
# 2. The client IP address of the user is obtained from the CaClientIp system parameter.
parameters:
 AppId: "System: CaAppId"
 ClientIP: "System: CaClientIp"
rules:
 # The first throttling policy. This policy applies to the app whose ID is 10001. The throttling threshold for ea
ch client IP address is 100 requests per second.
 - name: "Vip"
   condition: "$AppId = 10001"
   byParameters: "ClientIP"
   value: 100
   period: SECOND
 # The second throttling policy named PerClientIP. The throttling threshold for each client IP address is 10 re
quests per second.
 - name: "PerClientIP"
   byParameters: "ClientIP"
   value: 10
   period: SECOND
```

3.2. Plug-in configuration

The `YAML` or equivalent `JSON` format is used to configure the metadata of the throttling plug-in.

```
---
scope: "PLUGIN"           # The scope of the throttling plug-in. Valid values: PLUGIN and API.
defaultLimit: 100         # The default throttling threshold.
defaultPeriod: SECOND     # The default unit of time for throttling.
defaultErrorMessage: "Throttled by 100/SECOND"
parameters:               # The parameters that can be used for throttling.
 clientIp: "System:CaClientIp"
 userId: "Token:userId"
rules:
 - name: "ByClientIp"
   byParameters: "clientIp"
   condition: "$clientIp ! in_cidr '61.7.8.8/24'"
   limit: 10
   period: MINUTE
   errorMessage: "Throttled by 10/MINUTE from ${clientIp}"
 - name: "A maximum of 10 requests per minute are allowed for each user, except the administrator."
   byParameters: "clientIp"
   condition: "$userId ! like 'admin%'"
   limit: 10
   period: MINUTE
 - name: "A maximum of 10 requests per minute are allowed for each client IP address."
   byParameters: "clientIp"
   condition: "$clientIp in_cidr '67.0.0.0/8'"
   limit: 10
   period: MINUTE
 - name: "A maximum of 15 requests per minute are allowed for each user."
   condition: "$userId ! like 'admin%'"
   limit: 15
   period: MINUTE
   byParameters: "clientIp"
```

Fields:

- `scope`: required. The scope of the throttling plug-in. Valid values: `API` and `PLUGIN`. If one throttling plug-in is bound to multiple APIs, the setting of the `scope` field affects the scope of throttling policies bound to the APIs. For example, a throttling policy is `10 times per second`.
  - If the value of this field is `API`, the throttling threshold for each API is `10 times per second`.
  - If the value of this field is `PLUGIN`, the total throttling threshold for all the APIs is `10 times per second`.
- `parameters`: required. The parameters that are used for throttling. For more information, see Use parameters and conditional expressions
- `rules`: optional. The throttling policies. If `defaultLimit` and `defaultPeriod` are not specified, the rules field must be specified. Each throttling policy contains the following fields:

- `name`: required. The name of the throttling policy. The value of this field must comply with the `[A-Za-z0-9_-]+` regular expression. The name of each throttling policy must be unique in a throttling plug-in.
- `byParameters`: required. The parameter that is used for throttling. If multiple parameters are specified, separate them with commas (`,`). If the value of this field is `clientIp`, throttling is performed based on the value of the `clientIp` parameter. If the value of this field is `userId,action`, throttling is performed based on the combined values of the two parameters.
- `condition`: optional. If you specify this field, the related throttling policy is used only when the condition indicated by this field is met.
- `limit`: required. The throttling threshold. The value must be a positive integer. If you set this field to `-1`, throttling is not performed if the specified condition is met.
- `period`: required. The unit of time for throttling. Valid values: `SECOND`, `MINUTE`, `HOUR`, and `DAY`.
- `errorMessage`: required. The error message. You can use a template to customize error messages. For parameters specified in the `parameters` field, you can define error messages for them by using the `${Name}` format.

- `defaultLimit`: optional. The default throttling threshold. The value must be a positive integer.
- `defaultPeriod`: optional. The unit of time for throttling. Valid values: `SECOND`, `MINUTE`, `HOUR`, and `DAY`.
- `defaultErrorMessage`: optional. The custom error message. If this field is specified, the returned `X-Ca-Error-Message` header contains the custom error message.

3.3. Parameters supported by a throttling plug-in

The following table describes the parameters that are supported by a throttling plug-in.

Parameter	Scope	Description
Method	Requests	The HTTP request method, in uppercase, such as `GET` and `POST`.

Parameter	Scope	Description
Path	Requests	The complete HTTP request path, such as `/path/to/query`.
Header	Requests	Use `Header:{Name}` to obtain the first value of the HTTP header that is specified by `{Name}`.
Query	Requests	Use `Query:{Name}` to obtain the first value of the query string that is specified by `{Name}`.
Form	Requests	Use `Form:{Name}` to obtain the first value of the request form that is specified by `{Name}`.
Host	Requests	Use `Host:{Name}` to obtain the template parameters of the matched wildcard domain names.
Parameter	Requests	Use `Parameter:{Name}` to obtain the first value of the custom API parameter that is specified by `{Name}`.
System	Requests	Use `System:{Name}` to obtain the value of the system parameter that is specified by `{Name}`.

Parameter	Scope	Description
Token	Requests	If `JWT` or `OAuth 2.0` is used for authorization, use `Token:{Name}` to obtain the value of the token that is specified by `{Name}`.

3.4. Throttling rules

API Gateway performs parameter-based throttling by using the following rules:

- Plug-ins use the `parameters` field to obtain parameters from the context of a request.
- If all specified `conditions` are met or no `conditions` are specified in a throttling policy, the policy is executed.
- If multiple matched policies have the same `byParameters` setting, only the policy that comes first in the throttling plug-in takes effect.

4. Configuration examples

4.1. Basic throttling configuration

You can perform basic throttling for APIs, apps, or users. Throttling for apps is based on their AppKeys.

```
---
unit: SECOND        # The default unit of time for throttling. Valid values: SECOND, MINUTE, HOUR, and DAY.
apiDefault: 50      # The default API-level throttling threshold.
appDefault: 20      # Optional. The default app-level throttling threshold. This threshold cannot be greater t
han the user-level throttling threshold.
userDefault: 30     # Optional. The default user-level throttling threshold. This threshold cannot be greater t
han the API-level throttling threshold.
specials:           # Optional. The special throttling settings. You can set throttling thresholds for special apps a
nd users in a throttling policy.
 - type: "APP"      # The special throttling type. The value APP indicates that throttling is performed for speci
al apps based on their AppKeys.
   policies:
    - key: 10001    # The app ID. You can obtain the ID of an app from the app details page. To go to this page, c
lick Consume APIs and then APPs in the left-side navigation pane of the API Gateway console and click the na
me of the app.
      value: 3      # The special throttling threshold for the app. This threshold cannot be greater than the user-
level throttling threshold.
    - key: 10003
      value: 40
 - type: "USER"     # The special throttling type. The value USER indicates that throttling is performed for spe
cial Alibaba Cloud accounts.
   policies:
    - key: 102      # The ID of an Alibaba Cloud account. You can move the pointer over the profile picture in the
upper-right corner of the Alibaba Cloud Management Console to obtain the ID.
      value: 10     # The special throttling threshold for the Alibaba Cloud account. This threshold cannot be gr
eater than the API-level throttling threshold.
    - key: 233
      value: 35
```

4.2. Throttling based on the ClientIP parameter

In this example, the following throttling policies are used:

- A maximum of `100` API calls per minute are allowed for each client IP address.
- If a client IP address falls in the `58.66.66.0/24` CIDR block, throttling is not performed for this client IP address.
- If a client IP address falls in the `63.0.0.22/24` or `73.0.2.0/24` CIDR block, a maximum of `five` API calls per day are allowed for this client IP address.

```
---
scope: API          # The throttling scope. Valid values: API and PLUGIN.
parameters:         # The parameters that are used for throttling. Throttling is performed only based on the ClientIP parameter. You can obtain the values of this parameter from the CaClientIp system parameter.
  ClientIp: "System:CaClientIp"
rules:
  - name: whitelist    # The whitelist policy. Throttling is not performed for client IP addresses that meet the condition specified in this policy.
    condition: "$ClientIp in_cidr '58.66.66.0/24'"
    limit: -1        # The value -1 indicates that throttling is not performed.
  - name: banList     # The special throttling policy. If a client IP address meets the condition specified in this policy, a maximum of five API calls per day are allowed for the client IP address.
    condition: "$ClientIp in_cidr '63.0.0.22' or $ClientIp in_cidr '73.0.2.0/24'"
    byParameters: "ClientIp"
    limit: 5
    period: DAY
  - name: 100perIp    # The default throttling policy. A maximum of 100 API calls per minute are allowed for each client IP address.
    byParameters: "ClientIp"
    limit: 100
    period: MINUTE    # The unit of time. Valid values: SECOND, MINUTE, HOUR, and DAY.
```

5. Error codes

T429ID	429	Throttled by INNER DOMAIN Flow Control, ${Domain} is a test domain, only 1000 requests per day	The error message returned because the number of requests initiated has exceeded the upper limit allowed for a default second-level domain name. To increase the quota, use your own domain name.
T429IN	429	Throttled by INSTANCE Flow Control	The error message returned because throttling is performed for the current instance.
T429GR	429	Throttled by GROUP Flow Control	The error message returned because throttling is performed for the current group.

T429PA	429	Throttled by API Flow Control	The error message returned because the default API-level throttling policy defined in the throttling plug-in is used.
T429PR	429	Throttled by PLUGIN Flow Control	The error message returned because the special throttling policy defined in the throttling plug-in is used.
T429UP	429	Throttled by Usage Plan Flow Control	The error message returned because throttling is performed for the usage plan.

6. Limits

- A maximum of **16** parameters can be specified in a throttling plug-in.
- A single expression can contain a maximum of **512** characters.
- The metadata of a throttling plug-in can contain a maximum of **16,380** characters.
- A maximum of **16** `throttling policies` can be defined in a throttling plug-in.
- A maximum of three parameters can be specified in the `byParameters` field of each `throttling policy`.
- If day-level throttling is specified for client IP addresses and the system stores excessive throttling records, some records are released. For a shared instance, a throttling plug-in can control traffic based on a maximum of **1,000** parameters. For a dedicated instance, a throttling plug-in can control traffic based on a maximum of **100,000** parameters.

5.Plug-ins of the IP Access Control type

API Gateway provides plug-ins of the IP Access Control type to enhance the security of API operations. These plug-ins are used to specify IP addresses or Classless Inter-Domain Routing (CIDR) blocks from which API requests can be sent. You can add an IP address to the whitelist or blacklist of an API operation to allow or reject API requests from that IP address.

Instructions

When you configure a plug-in of the IP Access Control type, you can use two control modes:

- **Allow**: In the Allow mode, you can configure a whitelist to allow certain API requests. The following types of whitelists are supported:
  - You can configure a whitelist that includes only IP addresses. In this case, only API requests from the IP addresses in the whitelist are allowed.
  - You can configure a whitelist that specifies applications and their IP addresses. In this case, each application can send API requests only from its IP addresses in the whitelist.
- **Refuse**: In the Refuse mode, you can configure an IP address blacklist. API Gateway rejects all API requests from the IP addresses in the blacklist.

Configurations

You can configure a plug-in of the IP Access Control type in the JSON or YAML format. The two formats have the same schema and can be converted to each other by using a conversion tool. The following code snippet is a YAML template for configuring a plug-in of the IP Access Control type:

```
---
type: ALLOW   # The control mode of the plug-in. You can set the control mode to ALLOW or REFUSE.
items:
- blocks:      # A CIDR block from which API requests are allowed.
  - 61.3.9.0/24  # Specify a CIDR block.
  appId: 219810  # Optional. If you specify an application for a CIDR block, the CIDR block applies only to this application.
- blocks:      # An IP address from which API requests are allowed.
  - 79.11.12.2   # Specify an IP address.
- blocks:      # The CIDR block of a virtual private cloud (VPC) from which API requests are allowed.
  - 100.64.0.0/10   # Specify the CIDR block of a VPC. This item applies only to dedicated instances. When an API request is sent from a dedicated instance in a VPC, the source IP address of the API request is in the specified CIDR block.
```

Pay special attention to the last item in the code snippet. API Gateway allows you to send an API request from a dedicated instance in a VPC. In this example, the source IP address of the API request is in the CIDR block 100.64.0.0/10. If a dedicated instance is accessible within the VPC and also from certain public IP addresses, you can specify 100.64.0.0/10 as the source IP address of API requests that are sent from the CIDR of the VPC. You must specify the certain public IP addresses in another item if required.

6.JWT authentication

RFC 7519-compliant JSON Web Token (JWT) is a simple method used by API Gateway to authenticate requests. API Gateway hosts the public JSON Web Keys (JWKs) of users and uses these JWKs to sign and authenticate JWTs in requests. Then, API Gateway forwards claims to backend services as backend parameters. This simplifies the development of backend applications.

You can use the JWT authentication plug-in to implement the original OpenID Connect feature configured on APIs. We recommend that you use the JWT authentication plug-in, which has the following advantages over the OpenID Connect feature:

- You do not need to configure an additional authorization API. JWTs can be generated and distributed in multiple ways. API Gateway is only responsible for JWT authentication by using public JWKs.
- JWKs without kid specified are supported.
- Multiple JWKs can be configured.
- You can read token information from the header of a request or a query parameter.
- If you want to transmit a JWT in an Authorization header, such as Authorization bearer {token}, you can set parameter to Authorization and parameterLocation to header, so the token information is correctly read.
- The jti claim-based anti-replay check is supported if you set preventJtiReplay to true.
- Requests that do not include tokens can be forwarded to backend services without verification if you set bypassEmptyToken to true.
- The verification on the exp setting for tokens can be skipped if you set ignoreExpirationCheck to true.

If you configure a JWT authentication plug-in and bind it to an API for which the OpenID Connect feature is configured, the JWT authentication plug-in takes effect in place of the OpenID Connect feature.

1. Obtain a JWK

RFC 7517-compliant JWK is used to sign and authenticate JWTs. If you want to configure a `JWT authentication plug-in`, you need to generate an available `JWK` manually or by using an available online `JWK generator` such as mkjwk.org.The following example shows an available `JWK`. In the JWK example, the private key is used to sign the token, and the public key is configured in the `JWT authentication plug-in` to authenticate the signature.

```
{
 "kty": "RSA",
 "e": "AQAB",
 "kid": "O9fpdhrViq2zaaaBEWZITz",
 "use": "sig",
 "alg": "RS256",
 "n": "qSVxcknOm0uCq5vGsOmaorPDzHUubBmZZ4UXj-9do7w9X1uKFXAnqfto4TepSNuYU2bA_-tzSLAGBsR-BqvT6w9SjxakeiyQpVmexxnDw5WZwpWenUAcYrfSPEoNU-0hAQwFYgqZwJQMN8ptxkd0170PFauwACOx4Hfr-9FPGy8NCoIO4MfLXzJ3mJ7xqgIZp3NIOGXz-GIAbCf13ii7kSStpYqN3L_zzpvXUAos1FJ9IPXRV84tIZpFVh2lmRh0h8ImK-vI42dwlD_hOIzayL1Xno2R0T-d5AwTSdnep7g-Fwu8-sj4cCRWq3bd61Zs2QOJ8iustH0vSRMYdP5oYQ"
}
```

*The preceding JWK is in JSON format. If you want to configure a JWT authentication plug-in in YAML format, you must use a JWK in YAML format.**

- For a `JWT authentication plug-in`, you only need to configure a `public key`. Keep your `private key` safe. The following table lists the signature algorithms supported by the JWT authentication plug-in.

Signature algorithm	Supported `alg` setting
RSASSA-PKCS1-V1_5 with SHA-2	RS256, RS384, RS512
Elliptic Curve (ECDSA) with SHA-2	ES256, ES384, ES512
HMAC using SHA-2	HS256, HS384, HS512

When you configure a key of the HS256, HS384, or HS512 type, the key value is base64url encoded. If the signature is invalid, check whether your key is in the same format as the key used to generate the token.

2. Plug-in configurations

You can configure a JWT authentication plug-in in JSON or YAML format because these two formats use the same schema. You can use the `yaml to json` tool to convert the plug-in configuration format. The following example describes a plug-in configuration template in YAML format:

```
---
parameter: X-Token        # The parameter from which the JWT is read. It corresponds to an API parameter.
parameterLocation: header   # The location from which the JWT is read. Valid values: query and header. This parameter is optional if Request Mode for the bound API is set to Request Parameter Mapping. It is required i f Request Mode for the bound API is set to Request Parameter Passthrough.
preventJtiReplay: false     # Controls whether to enable the anti-replay check for jti. Default value: false.
bypassEmptyToken: false     # Controls whether to forward requests that do not include tokens to backend services without verification.
ignoreExpirationCheck: false # Controls whether to ignore the verification of the exp setting.
claimParameters:          # The conversion of claims to parameters. API Gateway maps JWT claims to backend parameters.
- claimName: aud          # The name of the JWT claim, which can be public or private.
  parameterName: X-Aud     # The name of the backend parameter, to which the JWT claim is mapped.
  location: header         # The location of the backend parameter, to which the JWT claim is mapped. Valid val ues: query, header, path, and formData.
- claimName: userId        # The name of the JWT claim, which can be public or private.
  parameterName: userId     # The name of the backend parameter, to which the JWT claim is mapped.
  location: query          # The location of the backend parameter, to which the JWT claim is mapped. Valid valu es: query, header, path, and formData.
#
# Public key in the JWK
jwk:
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5vGsOmaorPDzHUubBmZZ4UXj-9do7w9X1uKFXAnqfto4TepSNuYU2bA_-tzSLAGBsR-Bqv T6w9SjxakeiyQpVmexxnDw5WZwpWenUAcYrfSPEoNU-0hAQwFYgqZwJQMN8ptxkd0170PFauwACOx4Hfr-9F PGy8NCoIO4MfLXzJ3mJ7xqgIZp3NIOGXz-GIAbCf13ii7kSStpYqN3L_zzpvXUAos1FJ9IPXRV84tIZpFVh2lmRh0h8 ImK-vI42dwlD_hOIzayL1Xno2R0T-d5AwTSdnep7g-Fwu8-sj4cCRWq3bd61Zs2QOJ8iustH0vSRMYdP5oYQ
#
# You can configure multiple JWKs and use them together with the jwk field.
# If multiple JWKs are configured, kid is required. If the JWT does not include kid, the consistency check on ki d fails.
jwks:
- kid: O9fpdhrViq2zaaaBEWZITz      # If only one JWK is configured, kid is optional. If the JWT includes kid, AP I Gateway checks the consistency of kid.
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5v....
- kid: 10fpdhrViq2zaaaBEWZITz      # If only one JWK is configured, kid is optional. If the JWT includes kid, API Gateway checks the consistency of kid.
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5v...
```

- JWT authentication plug-ins retrieve JWTs based on the parameter and parameterLocation settings. For example, if parameter is set to X-Token and parameterLocation is set to header, the JWT is read from the X-Token header.
- If the parameter configured in an API has the same name as the parameter specified by parameter, do not specify parameterLocation. Otherwise, an error is reported when the API is called.
- If you want to transmit a token in an Authorization header, such as Authorization bearer {token}, you can set parameter to Authorization and parameterLocation to header, so the token information is correctly read.
- If preventJtiReplay is set to true, the JWT authentication plug-in uses jti in claims to perform an anti-replay check.
- If bypassEmptyToken is set to true and a token is not included in a request, API Gateway skips the check and directly forwards the request to a backend service.
- If ignoreExpirationCheck is set to true, API Gateway skips the verification of the exp setting. Otherwise, API Gateway checks whether a token expires.
- If API Gateway is required to forward claims in tokens to backend services, you can set tokenParameters to configure the following parameters to be forwarded:
  - claimName: the name of the claim in a token, which can be kid.
  - parameterName: the name of the parameter forwarded to a backend service.
  - location: the location of the parameter forwarded to a backend service. Valid values: header, query, path, and formData.
    - If this parameter is set to path, the backend path must contain a parameter with the same name, such as /path/{userId}.
    - If this parameter is set to formData, the body of a received request in a backend service must be of the Form type.
- You can configure a single key in the jwk field. You can also configure multiple keys in the jwks field.
  - You can configure only one key with kid not specified.
  - You can configure multiple keys with kid specified. kid must be unique.

3. Verification rules

- A JWT authentication plug-in obtains tokens based on `parameter` and `parameterToken` settings. If API Gateway is required to forward requests to backend services even when tokens are not included in the requests, set `bypassEmptyToken` to true.
- If you want to configure multiple keys, abide by the following principles:
  - Preferentially select a key whose ID is the same as the value of `kid` in a token for signature and authentication.
  - You can configure only one key with `kid` not specified. If there is no key whose ID is the same as the value of `kid` in a token, use the key with `kid` not specified for signature and authentication.
  - If all the configured keys have specified `kid` settings, and the token in a request does not contain `kid` or no keys match `kid`, an `A403JK` error is reported.
- If a token contains `iat`, `nbf`, and `exp`, the JWT authentication plug-in verifies the validity of the time format.
- By default, API Gateway verifies the `exp` setting. If you want to skip the verification, set `ignoreExpirationCheck` to true.
- `tokenParameters` is configured to extract the required parameters from the `claims` of a token. These parameters are forwarded to backend services.

4. Configuration examples

4.1 Configure a single JWK

```
---
parameter: X-Token       # The parameter from which the JWT is read. It corresponds to an API parameter.
parameterLocation: header  # The location from which the JWT is read. Valid values: query and header. This
parameter is optional if Request Mode for the bound API is set to Request Parameter Mapping. It is required i
f Request Mode for the bound API is set to Request Parameter Passthrough.
claimParameters:        # The conversion of claims to parameters. API Gateway maps JWT claims to backend p
arameters.
- claimName: aud        # The name of the JWT claim, which can be public or private.
  parameterName: X-Aud    # The name of the backend parameter, to which the JWT claim is mapped.
  location: header       # The location of the backend parameter, to which the JWT claim is mapped. Valid valu
es: query, header, path, and formData.
- claimName: userId      # The name of the JWT claim, which can be public or private.
  parameterName: userId   # The name of the backend parameter, to which the JWT claim is mapped.
  location: query        # The location of the backend parameter, to which the JWT claim is mapped. Valid value
s: query, header, path, and formData.
preventJtiReplay: false   # Controls whether to enable the anti-replay check for jti. Default value: false.
#
# Public key in the JWK
jwk:
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5vGsOmaorPDzHUubBmZZ4UXj-9do7w9X1uKFXAnqfto4TepSNuYU2bA_-tzSLAGBsR-Bqv
T6w9SjxakeiyQpVmexxnDw5WZwpWenUAcYrfSPEoNU-0hAQwFYgqZwJQMN8ptxkd0170PFauwACOx4Hfr-9F
PGy8NCoIO4MfLXzJ3mJ7xqgIZp3NIOGXz-GIAbCf13ii7kSStpYqN3L_zzpvXUAos1FJ9IPXRV84tIZpFVh2lmRh0h8
ImK-vI42dwlD_hOIzayL1Xno2R0T-d5AwTSdnep7g-Fwu8-sj4cCRWq3bd61Zs2QOJ8iustH0vSRMYdP5oYQ
```

4.2 Configure multiple JWKs

```
---
parameter: Authorization  # The parameter from which the token is obtained.
parameterLocation: header  # The location from which the token is obtained.
claimParameters:        # The conversion of claims to parameters. API Gateway maps JWT claims to backend p
arameters.
- claimName: aud        # The name of the JWT claim, which can be public or private.
  parameterName: X-Aud    # The name of the backend parameter, to which the JWT claim is mapped.
  location: header       # The location of the backend parameter, to which the JWT claim is mapped. Valid valu
es: query, header, path, and formData.
- claimName: userId      # The name of the JWT claim, which can be public or private.
  parameterName: userId   # The name of the backend parameter, to which the JWT claim is mapped.
  location: query        # The location of the backend parameter, to which the JWT claim is mapped. Valid value
s: query, header, path, and formData.
preventJtiReplay: true    # Controls whether to enable the anti-replay check for jti. Default value: false.
jwks:
- kid: O9fpdhrViq2zaaaBEWZITz       # kid must be set to different values for multiple different JWKs.
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5v....
- kid: 10fpdhrViq2zaaaBEWZITz       # kid must be set to different values for multiple different JWKs.
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5v....
```

5. Error codes

Status	Code	Message	Description
400	I400JR	JWT required	No JWT-related parameters are found.
403	S403JI	Claim `jti` is required when `preventJtiReplay:true`	If preventJtiReplay is set to true in a `JWT authentication plug-in`, no valid `jti` claim is included in the request.

Status	Code	Message	Description
403	S403JU	Claim `jti` in JWT is used	If preventJtiReplay is set to true in a **JWT authentication plug-in**, the `jti` claim that is included in the request is used.
403	A403JT	Invalid JWT: ${Reason}	The `JWT` that is read from the request is invalid.
400	I400JD	JWT Deserialize Failed: `${Token}`	The `JWT` that is read from the request fails to be parsed.
403	A403JK	No matching JWK, `kid:${kid}` not found	There is no `JWK` that matches `kid` configured in the `JWT` included in the request.
403	A403JE	JWT is expired at `${Date}`	The `JWT` that is read from the request expires.
400	I400JP	Invalid JWT plugin config: ${JWT}	The **JWT authentication** plug-in is incorrectly configured.

If an HTTP response message includes an unexpected response code specified by `ErrorCode` in the `X-Ca-Error-Code` header, such as `A403JT` or `I400JD`, you can visit the jwt.io website to check the token validity and format.

6. Limits

- The metadata of a JWT authentication plug-in contains a maximum of **16,380** characters.
- You can configure a maximum of **16** parameters to be forwarded. Both the `claimName` and `parameterName` parameters cannot exceed **32** characters in length. Only the following regular expression is supported: [A-Za-z0-9-_].
- `alg` can be set to RS256, RS384, RS512, ES256, ES384, ES512, HS256, HS384, or HS512 for JWKs.

7.Plug-ins of the CORS type

This topic describes plug-ins of the CORS type. For information about cross-origin resource sharing (CORS), see CORS.

Configurations

You can configure a plug-in of the CORS type in the JSON or YAML format. The two formats have the same schema and can be converted to each other by using a conversion tool. The following code snippet is a YAML template for configuring a plug-in of the CORS type:

```
---allowOrigins: api.foo.com,api2.foo.com   # The origins from which API requests are allowed. Default value: *.
allowMethods: GET,POST,PUT          # The HTTP methods that can be used to send API requests. Separate multiple methods with commas (,).
allowHeaders: X-Ca-RequestId        # The header fields that can be used in API requests. Separate multiple header fields with commas (,).
exposeHeaders: X-RC1,X-RC2          # The header fields that can be exposed to the XMLHttpRequest object. Separate multiple header fields with commas (,).
allowCredentials: true              # Specifies whether to enable cookies.
maxAge: 172800
```

8.Plug-ins of the Caching type

You can bind a plug-in of the Caching type to an API operation to cache responses from the backend service of the API operation in API Gateway. This effectively reduces load on the backend and shortens response time.

1. Usage notes

- Plug-ins of the Caching type can only cache responses to API requests that use the GET method.
- Plug-ins of the Caching type cannot cache responses to API requests that use the default second-level domain name of the corresponding API group. The default second-level domain name of an API group can only be used for testing API calls and can be used for a maximum of 1,000 times per day.
- When you configure a plug-in of the Caching type, you can use the following parameters to sort responses in a cache:
  - **varyByApp**: specifies whether to sort cached responses based on applications from which API requests are sent.
  - **varyByParameters**: specifies whether to sort cached responses based on values of request parameters in API requests. The plug-in uses the same request parameters of API operations that are bound to the plug-in to sort responses to API requests.
  - **varyByHeaders**: specifies whether to sort cached responses based on request header fields in API requests, for example, sorts responses based on the `Accept` or `Accept-Language` header field.
- API Gateway provides each user with 5 MB of cache space in each region. Caches are cleared upon expiration. After a cache reaches its space limit, no more responses will be stored in the cache.
- If the `Cache-Control` header field is specified in a response from the backend of an API operation, the response is stored in a cache based on the specified cache policy. If the `Cache-Control` header field is not specified in a response, the response is stored in a cache based on the default cache policy and for a period of time that is specified by the `duration` parameter of the plug-in of the Caching type that is bound to the API operation.
- A response can be stored in a cache for a maximum of 48 hours, that is, 172,800 seconds. If you set the duration parameter of a plug-in of the Caching type to a value greater than 172800, the cache retention period is still 48 hours.
- By default, API Gateway ignores the `Cache-Control` header field in client requests. You can specify the `clientCacheControl` parameter for a plug-in of the Caching type to decide whether to ignore or handle the `Cache-Control` header field in client requests. You can set the clientCacheControl parameter to the following modes:
  - `off` : ignores the `Cache-Control` header field in all client requests.
  - `all` : handles the `Cache-Control` header field in all client requests.
  - `app` : handles the `Cache-Control` header field in requests from applications that are specified by the `apps` parameter of the plug-in of the Caching type.
- By default, API Gateway caches only the `Content-Type` , `Content-Encoding` , and `Content-Language` header fields in responses. If you need to cache more header fields, add the header fields in the `cacheableHeaders` parameter of the plug-in of the Caching type.

2. Configurations

You can configure a plug-in of the Caching type in the JSON or YAML format. The two formats have the same schema and can be converted to each other by using a conversion tool. The following code snippet is a YAML template for configuring a plug-in of the Caching type:

```
---
varyByApp: false  # Specifies whether to sort cached responses based on the IDs of applications from which API requests are sent. Default value: false.
varyByParameters:  # Specifies whether to sort cached responses based on values of request parameters in API requests.
- userId       # The name of a request parameter that is used to sort cached responses. If a request parameter is mapped to a different name at the backend, use the name after mapping.
varyByHeaders:    # Specifies whether to sort cached responses based on request header fields in API requests.
- Accept       # A request header field that is used to sort cached responses. In this example, the `Accept` header field is used.
clientCacheControl: # Specifies whether to allow the `Cache-Control` header field in client requests to affect the cache policy. Default value: `off`.
 mode: "app" # The mode of the clientCacheControl parameter. Valid values: off: ignores the Cache-Control header field in all client requests. all: handles the Cache-Control header field in all client requests. apps: handles the Cache-Control header field in requests from applications that are specified by the `apps` parameter.
 apps:       # The applications from which the Cache-Control header field in API requests is allowed to affect the cache policy. Before you set this parameter, you must set the clientCacheControl parameter to `apps`.
 - 1992323     # The AppId of an application. Do not confuse the AppId and the AppKey of an application.
 - 1239922     # The AppId of an application. Do not confuse the AppId and the AppKey of an application.
cacheableHeaders:  # The response header fields that can be cached. By default, only the `Content-Type`, `Content-Length`, and `Content-Language` header fields can be cached.
- X-Customer-Token  # The name of a response header field that can be cached.
duration: 3600    # The default cache retention period. Unit: seconds.
```

3. Working mechanism

- If an API request hits the cache of an API operation, the `X-Ca-Caching: true` header field is included in the response to the API request.

4. Limits

- For each plug-in of the Caching type, you can configure a maximum of 16,380 bytes of metadata.
- A response body that exceeds 128 KB in size cannot be cached.
- For API operations on shared instances, each user has a maximum of 30 MB of total cache space in each region. For information about limits on API operations on dedicated instances, see the specifications of dedicated instances.

9.Plug-ins of the Routing type

1. Overview

A plug-in of the Routing type is used to route API requests to different backends by changing the backend type, address, path, and response parameters of an API operation based on request and system parameters in API requests. These plug-ins can be used for multi-tenant routing and blue-green release. They can also be used to distinguish between environments, that is, route all requests for API operations that are published to the same environment to the same server.

2. Configurations

2.1. Template

You can configure a plug-in of the Routing type in the JSON or YAML format. The two formats have the same schema and can be converted to each other by using a conversion tool. The following code snippet is a YAML template for configuring a plug-in of the Routing type:

```
---
routes:
# Route requests from the same caller to an independent backend service address. In this example, requests from the caller whose AppId is 123456 are routed to an address that belongs to the Classless Inter-Domain Routing (CIDR) block of a virtual private cloud (VPC) and is named slbAddressForVip.
- name: Vip
  condition: "$CaAppId = 123456"
  backend:
    type: "HTTP-VPC"
    vpcAccessName: "slbAccessForVip"
# If an API request is sent from an out-dated client, return a message to remind the caller that the client version is no longer supported. You can specify the ClientVersion parameter as needed when you create an API operation.
- name: MockForOldClient
  condition: "$ClientVersion < '2.0.5'"
  backend:
    type: "MOCK"
    statusCode: 400
    mockBody: "This version is not supported!!!"
# In scenarios where blue-green release is involved, route a specified percentage of requests to a backend that applies blue-green release. In this example, the specified percentage is 5%.
- name: BlueGreenPercent05
  condition: "Random() < 0.05"
  backend:
    type: "HTTP"
    address: "https://beta-version.api.foo.com"
  constant-parameters:
  - name: x-route-blue-green
    location: header
    value: "route-blue-green"
```

The template shows a `routes` object that contains multiple `route` objects. Each `route` object is used to specify a routing rule. Each routing rule consists of the following parts:

- **name**: the name of the routing rule. The name must be unique within each plug-in, and can contain

letters and digits. If an API request hits the rule, an HTTP header field `X-Ca-Routing-Name` that contains the name of the rule is added to the request before the request is routed to the backend.

- **condition**: the conditional expression of the routing rule. For information about how to write conditional expressions, see section 2.2. If an API request meets the condition, the request hits the routing rule. A plug-in of the Routing type checks the routing rules based on the order in which they are configured. An API request is routed to the backend in the first routing rule that the request hits. After that, the plug-in does not check the remaining routing rules. If you configure multiple routing rules, make sure that they are configured in the order that meets your service expectations.
- **backend**: the information about the backend. Configurations of the backend must be consistent with the OpenAPI specification files that are imported to API Gateway. For more information, see []. The backend configurations in a plug-in of the Routing type override the backend configurations in an API operation that is bound to the plug-in. If the backend configurations are incomplete after the overriding, the following message is returned to the client: `X-Ca-Error-Code: I504RB`. If you receive this message, check whether the backend configurations are complete. For more information about how to configure a backend, see section 2.3.
- **constant-parameters**: the constant parameters that you can customize in the routing rule. The constant parameters are attached to an API request before the request is routed to the backend. These parameters are used in the business logic of the backend. The location parameter can be set to `header` or `query`.

2.2. Conditional expressions

2.2.1 Syntax

- The syntax of conditional expressions in plug-ins of the Routing type are similar to that of SQL statements. The basic format is `$A = 'A' and '$B = 'B'`.
- Each parameter starts with `$`. You can reference the request parameters that are defined in an API operation to which a plug-in is bound. The request mode of the API operation can be set to either mapping or passthrough. If you defined a request parameter named query1 when you configured the API operation, you can use `$query1` to reference the parameter in conditional expressions.
- The following constant types are supported:
  - `STRING`: the string data type. Single quotation marks (' ') or double quotation marks (" ") can be used to enclose a string, for example, "Hello".
  - `INTEGER`: the integer data type, such as 1001 and -1.
  - `NUMBER`: the floating-point data type, such as 0.1 and 100.0.
  - `BOOLEAN`: the Boolean data type. Valid values: true and false.
- You can use `and` and `or` to connect different expressions.
- You can use parentheses `( )` to specify the priority of conditional expressions.
- The built-in function `Random()` can be used to generate a floating-point number in the range of `[0,1)`.
- You can use `$CaAppId` to reference system parameters in an API request. You do not need to define system parameters in an API operation. However, if you have defined a parameter in the API operation with the same name as a system parameter, the value of the system parameter is overwritten by that of the parameter that is defined in the API operation. The following system parameters can be referenced in a plug-in of the Routing type:
  - CaStage: the environment to which the requested API operation is published. Valid values: `RELEASE`, `PRE`, and `TEST`.

- CaDomain: the domain name of the API group to which the requested API operation belongs.
- CaRequestHandleTime: the time in UTC at which the current request is received.
- CaAppId: the value of the AppId parameter in the current request.
- CaAppKey: the value of the AppKey parameter in the current request.
- CaClientIp: the IP address of the client from which the current request is sent.
- CaApiName: the name of the requested API operation.
- CaHttpScheme: the protocol used by the current request. Valid values: `HTTP`, `HTTPS`, and `WS`.
- CaClientUa: the user agent string in the current request.

- If you use a non-existent parameter in a conditional expression, such as `$UnknonwParameter = 1`, the result of the expression is false.

2.2.2 Examples

- The following expression indicates that a probability must be less than 5%:

  ```
  Random() < 0.05
  ```

- The following expression indicates that the requested API operation must be published to the test environment:

  ```
  $CaStage = 'TEST'
  ```

- The following expression indicates that the custom parameter UserName must be specified as Admin and the IP address of the client must be `47.47.74.77`:

  ```
  $UserName = 'Admin' and $CaClientIp = '47.47.74.77'
  ```

- The following expression indicates that the AppId parameter must be specified as 1001, 1098, or 2011, and the protocol that is used by the API request must be HTTPS:

  ```
  $CaHttpScheme = 'HTTPS' and ($CaAppId = 1001 or $CaAppId = 1098 or $CaAppId = 2011)
  ```

2.3. Backend configuration and overriding rules

Configurations of a backend must be consistent with the OpenAPI specification files that are imported to API Gateway. For more information, see Import Swagger files to create APIs. The following examples show the supported backend types and configuration samples. The backend configurations in a plug-in of the Routing type override the backend configurations in an API operation that is bound to the plug-in. If you do not need to change the backend type, specify only the parameters whose values you want to change.

- HTTP

```
---
backend:
 type: HTTP
 address: "http://10.10.100.2:8000"
 path: "/users/{userId}"
 method: GET
 timeout: 7000
```

- HTTP-VPC

```
---
backend:
 type: HTTP-VPC
 vpcAccessName: vpcAccess1
 path: "/users/{userId}"
 method: GET
 timeout: 10000
```

- Function Compute

```
---
backend:
 type: FC
 fcRegion: cn-shanghai
 serviceName: fcService
 functionName: fcFunction
 arn: "acs:ram::111111111:role/aliyunapigatewayaccessingfcrole"
```

- MOCK

```
---
backend:
 type: MOCK
 mockResult: "mock resul sample"
 mockStatusCode: 200
 mockHeaders:
  - name: server
    value: mock
  - name: proxy
    value: GW
```

2.4. Limits

- For each plug-in of the Routing type, you can configure a maximum of 16,384 bytes of metadata. If this limit is exceeded, the `InvalidPluginData.TooLarge` error is returned.
- A maximum of 16 routing rules can be configured in each plug-in of the Routing type. If this limit is exceeded, the `InvalidPluginData.TooManyRoutes` error is returned.
- Each conditional expression can contain a maximum of 512 characters. If this limit is exceeded, the `InvalidPluginData.ConditionTooLong` error is returned.
- Configuration updates of a plug-in of the Routing type are synchronized in real time to all API operations that are bound to the plug-in. The minimum interval between two updates is 45 seconds. If you try to update a plug-in in less than 45 seconds after the last update, the `InvalidPluginData.UpdateTooBusy` error is returned.

3. Typical scenarios

3.1. Configure multi-tenant routing: Route API requests to different backend addresses based on the value of the AppId parameter

Assume that users whose AppId is 10098 or 10099 are your VIP customers. You want to route API requests from these two users to an independent server cluster.

```
---
routes:
# If the AppId of the caller is 10098 or 10099, route the API request to an independent address.
# In this example, the address belongs to the CIDR block of a VPC and is named slbAddressForVip.
- name: Vip
 condition: "$CaAppId = 10098 or $CaAppId = 10099"
 backend:
  type: "HTTP-VPC"
  vpcAccessName: "slbAccessForVip"
```

3.2. Configure routing based on environments: Route all requests for API operations that are published to the same environment to the same server

Assume that you want all requests for API operations that are published to the test environment to be directed to your test server on the Internet.

```
---
routes:
# Route all requests for API operations that are published to the test environment to the test server on the I
nternet.
- name: Vip
 condition: "$CaStage = 'TEST'"
 backend:
  type: "HTTP"
  address: "https://test-env.foo.com"
```

3.3. Configure routing for performing blue-green release

You want to direct 5% of API requests to a group of test servers when you perform blue-green release.

```
---
routes:
# In scenarios where blue-green release is involved, route a specified percentage of requests to a backend th
at applies blue-green release. In this example, the specified percentage is 5%.
- name: BlueGreenPercent05
 condition: "Random() < 0.05"
 backend:
  type: "HTTP"
  address: "https://beta-version.api.foo.com"
```

10.Plug-ins of the Parametric Access Control type

1. Overview

In a plug-in of the Parametric Access Control type, you can define conditions based on the request parameters or context of an API operation to which the plug-in is bound. This allows you to decide whether to deliver an API request to the backend of an API operation. For information about how to define parameters and write conditional expressions, see Use parameters and conditional expressions.

2. Configurations

Assume that a plug-in of the Parametric Access Control type is bound to an API operation. The request path of the API operation is `/{userId}/...`. The JSON Web Token (JWT) authorization feature is configured for the API operation and two claim parameters, userId and userType, are defined. In this case, you can define the following conditions:

- If the userType parameter is set to admin, all request paths are allowed.
- If the userType parameter is set to user, only requests with the `/{userId}/...` path are allowed.

```
---
#
# In this example, the request path of the API operation is `/{userId}/...`.
# The JWT authorization feature is configured for the API operation and two claim parameters, userId and userType, are defined.
# You can define the following conditions:
# - If the userType parameter is set to admin, all request paths are allowed.
# - If the userType parameter is set to user, only requests with the `/{userId}/...` path are allowed.
parameters:
 userId: "Token:userId"
 userType: "Token:userType"
 pathUserId: "path:userId"
#
# The following rules are defined based on the preceding parameters. For each API request, the plug-in of the Parametric Access Control type checks the rules in sequence. If the condition in a rule is met, the result is `true` and the action that is specified by the `ifTrue` parameter is performed. If the condition in a rule is not met, the result is `false` and the action that is specified by the `ifFalse` parameter is performed.
# The action `ALLOW` indicates that the request is routed to the backend. The action `DENY` returns an error code to the client. After the `ALLOW` or `DENY` action is performed, the plug-in does not check the remaining rules.
# If neither the `ALLOW` action nor the `DENY` action is performed, the plug-in continues to check the next rule.
rules:
 - name: admin
   condition: "$userType = 'admin'"
   ifTrue: "ALLOW"
 - name: user
   condition: "$userId = $pathUserId"
   ifFalse: "DENY"
   statusCode: 403
   errorMessage: "Path not match ${userId} vs /${pathUserId}"
   responseHeaders:
    Content-Type: application/xml
   responseBody:
    <Reason>Path not match ${userId} vs /${pathUserId}</Reason>
```

3. Error codes

Error code	HTTP status code	Error message	Description
A403AC	403	Access Control Forbidden by ${RuleName}	The error message returned because the request is rejected by the plug-in of the Parametric Access Control type that is bound to the API operation.

4. Limits

- A maximum of 16 parameters can be defined in each plug-in of the Parametric Access Control type.
- Each conditional expression can contain a maximum of 512 characters.
- For each plug-in of the Parametric Access Control type, you can configure a maximum of 16,380 bytes of metadata.
- A maximum of 16 rules can be configured in each plug-in of the Parametric Access Control type.

11.Plug-ins of the Circuit Breaker type

1. Overview

API Gateway provides a circuit breaker for each API to protect the API in the event of abnormal backend performance. By default, if timeout occurs 1,000 times at the backend of an API within 30 seconds, the circuit breaker trips. The circuit breaker stays open for 90 seconds, during which the following error is returned for all API requests: Status=503,X-Ca-Error-Code=D503CB. After 90 seconds, the circuit breaker allows a limited number of concurrent API requests to pass through. If these requests are successful, the circuit breaker closes and API requests can be handled as expected again.

You can also bind a plug-in of the Circuit Breaker type to an API to customize the configurations of its circuit breaker. Take note that plug-ins of the Circuit Breaker type take effect only for APIs on dedicated instances. You can customize the following configurations of a circuit breaker:

- The condition under which the circuit breaker trips. You can specify that the circuit breaker trips after the number of occurrences of timeout, the number of occurrences of a long response time, or the number of occurrences of a specified error, at the backend reaches a threshold within a specified period of time.
- The time window during which the number of occurrences of timeout, the number of occurrences of a long response time, or the number of occurrences of a specified error, at the backend is checked by the circuit breaker to determine whether to trip.
- The period of time during which the circuit breaker stays open after it trips.
- The backend to which API requests are directed when the circuit breaker is open.

2. Configurations

Plug-ins of the Circuit Breaker type take effect only for APIs on dedicated instances. If you bind a plug-in of the Circuit Breaker type to an API on a shared instance, the circuit breaker still uses the default configurations.

2.1 Specify that the circuit breaker trips after the number of occurrences of timeout at the backend reaches a threshold

When you configure a plug-in of the Circuit Breaker type, you can specify that the circuit breaker trips after the number of occurrences of timeout at the backend reaches a threshold within a specified period of time. If the backend timeout threshold specified for an API is 10 seconds and no response is received from the backend within 10 seconds, one occurrence of timeout is counted.

```
timeoutThreshold: 15      # The threshold of the number of occurrences of timeout at the backend.
windowInSeconds: 30       # The time window during which the number of occurrences of timeout at the bac
kend is checked by the circuit breaker to determine whether to trip.
openTimeoutSeconds: 15    # The period of time during which the circuit breaker stays open after it trips.
downgradeBackend:         # The backend to which API requests are directed when the circuit breaker is open
.
 type: mock
 statusCode: 418
```

In the preceding code snippet, you can specify the following parameters:

- `timeoutThreshold`: the threshold of the number of occurrences of timeout at the backend. If this threshold is reached, the circuit breaker trips. The maximum value of this parameter is 5000. We recommend that you specify an appropriate value. If the value is too small, the circuit breaker trips after timeout occurs only several times.
- `windowsInSeconds`: the time window during which the number of occurrences of timeout at the backend is checked by the circuit breaker to determine whether to trip. Valid values: 10 to 90. Unit: seconds.
- `openTimeoutSeconds`: the period of time during which the circuit breaker stays open after it trips. Valid values: 15 to 300. Unit: seconds.
- `downgradeBackend`: optional. The backend to which API requests are directed when the circuit breaker is open.

2.2 Specify that the circuit breaker trips after the number of occurrences of a long response time at the backend reaches a threshold

When you configure a plug-in of the Circuit Breaker type, you can specify that the circuit breaker trips after the number of occurrences of a long response time at the backend reaches a threshold within a specified period of time. The backend response time is the duration between when API Gateway sends a request to the backend and when API Gateway receives a response from the backend.

```
---
errorThreshold: 10       # The threshold of the number of occurrences of a long response time.
windowInSeconds: 60         # The time window during which the number of occurrences of a long response time is checked by the circuit breaker to determine whether to trip.
openTimeoutSeconds: 120       # The period of time during which the circuit breaker stays open after it trips.
errorCondition: "$LatencyMilliSeconds > 500"    # The conditional expression that is used to determine whether the backend response time is counted as a long response time. In this example, if the backend response time exceeds 500 ms, it is considered as a long response time.
downgradeBackend:           # The backend to which API requests are directed when the circuit breaker is open.
 type: mock
 statusCode: 403
```

In the preceding code snippet, you can specify the following parameters:

- `errorThreshold`: the threshold of the number of occurrences of a long response time.
- `windowsInSeconds`: the time window during which the number of occurrences of a long response time is checked by the circuit breaker to determine whether to trip. Valid values: 10 to 90. Unit: seconds.
- `openTimeoutSeconds`: the period of time during which the circuit breaker stays open after it trips. Valid values: 15 to 300. Unit: seconds.

- `errorCondition`: the conditional expression that is used to determine whether the backend response time is counted as a long response time. You can use the `$LatencyMilliSeconds` and `$LatencySeconds` variables. The unit of `$LatencyMilliSeconds` is milliseconds. The unit of `$LatencySeconds` is seconds.
- `downgradeBackend`: optional. The backend to which API requests are directed when the circuit breaker is open.

2.3 Specify that the circuit breaker trips after the number of occurrences of a specified error at the backend reaches a threshold

When you configure a plug-in of the Circuit Breaker type, you can specify that the circuit breaker trips after the number of occurrences of a specified error at the backend reaches a threshold within a specified period of time.

```
errorCondition: "$StatusCode == 503"  # The conditional expression that specifies the error whose number of occurrences is checked by the circuit breaker to determine whether to trip.
errorThreshold: 1000              # The threshold of the number of occurrences of the specified error.
windowInSeconds: 30               # The time window during which the number of occurrences of the specified error at the backend is checked by the circuit breaker to determine whether to trip.
openTimeoutSeconds: 15            # The period of time during which the circuit breaker stays open after it trips.
downgradeBackend:                 # The backend to which API requests are directed when the circuit breaker is open.
 type: "HTTP"
 address: "http://api.foo.com"
 path: "/system-busy.json"
 method: GET
```

- `errorCondition`: the conditional expression that specifies the error whose number of occurrences is checked by the circuit breaker to determine whether to trip. You can use the `$StatusCode` and `$LatencySeconds` variables.
  - If you specify a conditional expression as `$StatusCode = 503 or $StatusCode = 504`, the circuit breaker checks the total number of occurrences of HTTP status codes 503 and 504.
  - If you specify a conditional expression as $LatancySeconds > 30, the circuit breaker checks the total number of occurrences of timeout that is greater than 30 seconds.
- `errorThreshold`: the threshold of the number of occurrences of the specified error.
- `windowsInSeconds`: the time window during which the number of occurrences of the specified error at the backend is checked by the circuit breaker to determine whether to trip. Valid values: 10 to 90. Unit: seconds.
- `openTimeoutSeconds`: the period of time during which the circuit breaker stays open after it trips. Valid values: 15 to 300. Unit: seconds.

- `downgradeBackend`: optional. The backend to which API requests are directed when the circuit breaker is open.

3. Specify the backend to which API requests are directed when the circuit breaker is open

You can set the `downgradeBackend` parameter to specify a backend to which API requests are directed when the circuit breaker is open. The configurations of the backend must be consistent with the API specification files that are imported to API Gateway. For more information, see Import Swagger files to create APIs. You can configure the following types of backends by using the samples:

- HTTP

```
---
backend:
 type: HTTP
 address: "http://10.10.100.2:8000"
 path: "/users/{userId}"
 method: GET
 timeout: 7000
```

- HTTP-VPC

```
---
backend:
 type: HTTP-VPC
 vpcAccessName: vpcAccess1
 path: "/users/{userId}"
 method: GET
 timeout: 10000
```

- Function Compute

```
---
backend:
 type: FC
 fcRegion: cn-shanghai
 serviceName: fcService
 functionName: fcFunction
 arn: "acs:ram::111111111:role/aliyunapigatewayaccessingfcrole"
```

- MOCK

```
---
backend:
 type: MOCK
 mockResult: "mock resul sample"
 mockStatusCode: 200
 mockHeaders:
  - name: Content-Type
    value: text-plain
  - name: Content-Language
    value: zhCN
```

4. Error codes

Error code	HTTP status code	Error message	Description
D503BB	503	Backend circuit breaker busy	The error message returned because the API is protected by its circuit breaker.
D503CB	503	Backend circuit breaker open, ${Reason}	The error message returned because the circuit breaker of the API is open. Test API calls after you check the backend performance of the API.

5. Limits

- Plug-ins of the Circuit Breaker type take effect only for APIs on dedicated instances.
- Each conditional expression can contain a maximum of 512 characters.
- For each plug-in of the Circuit Breaker type, you can configure a maximum of 50 KB of metadata.

12.Plug-ins of the Error Mapping type

1. Overview

A plug-in of the Error Mapping type is used to map backend error responses to new error responses based on mapping rules that are defined as required by clients.

2. Quick start

The following example shows an error response that is returned by the backend of an API operation. The HTTP status code is 200, but the response body contains an error message in a JSON string.

```
HTTP 200 OK
Content-Type:application/json

{"req_msg_id":"d02afa56394f4588832bed46614e1772","result_code":"ROLE_NOT_EXISTS"}
```

- Assume that clients want to receive an HTTP status code other than 200 but do not want to realize this by modifying backend configurations. For example, clients expect the following error response:

```
HTTP 404
X-Ca-Error-Message: Role Not Exists, ResultId=d02afa56394f4588832bed46614e1772
```

In this case, you can configure a plug-in of the Error Mapping type by using the following sample and bind the plug-in to the relevant API operation:

```
---
# The parameters that are involved in mapping.
parameters:
 statusCode: "StatusCode"
 resultCode: "BodyJsonField:$.result_code"
 resultId: "BodyJsonField:$.req_msg_id"
# The mapping condition under which an error response needs to be mapped.
errorCondition: "$statusCode = 200 and $resultCode <> 'OK'"
# The parameter in an error response that is used to specify the error code and hit mapping rules.
errorCode: "resultCode"
# The mapping rules.
mappings:
 - code: "ROLE_NOT_EXISTS"
   statusCode: 404
   errorMessage: "Role Not Exists, RequestId=${resultId}"
 - code: "INVALID_PARAMETER"
   statusCode: 400
   errorMessage: "Invalid Parameter, RequestId=${resultId}"
# Optional. The default mapping rule.
defaultMapping:
 statusCode: 500
 errorMessage: "Unknown Error, ${resultCode}, RequestId=${resultId}"
```

In this example, the HTTP status code and the `result_code` parameter in an error response are used to define the mapping condition. If the HTTP status code of an error response is 200 and the value of the `result_code` parameter is not `OK`, the mapping starts. The `result_code` parameter is used to define the mapping rules. One rule is that if the value of the result_code parameter is `ROLE_NOT_EXISTS`, the original HTTP status code is mapped to 404. Another rule is that if the value of the result_code parameter is `INVALID_PARAMETER`, the original HTTP status code is mapped to 400. A default mapping rule is also defined. If the value of the result_code parameter is neither of the preceding values, the original HTTP status code is mapped to 500.

3. Plug-in configurations and mapping rules

3.1. Plug-in configurations

You can configure a plug-in of the Error Mapping type in the JSON or YAML format. The following parameters can be specified:

- `parameters`: required. The parameters that are involved in mapping. These parameters are specified as key-value pairs in the `map` format. For information about how to define parameters and write conditional expressions, see Use parameters and conditional expressions.
- `errorCondition`: required. The mapping condition under which an error response needs to be mapped. If the result of the conditional expression is `true`, the mapping starts.
- `errorCode`: optional. The parameter in an error response that is used to specify the error code and hit mapping rules. The error code that is specified by the value of the `errorCode` parameter is compared with the value of the `code` parameter in the mapping rules.
- `mappings`: required. The mapping rules. API Gateway reconstructs error responses based on these mapping rules. A mapping rule may contain the following parameters:
  - `code`: optional. A specified value that is used to identify an error response whose error code that is specified by the value of the `errorCode` parameter is the same. The value of this parameter must be unique among all mapping rules. If the error code of an error response is the same as the value of the `code` parameter in the current mapping rule, the error response is mapped based on the current rule.
  - `condition`: optional. The condition under which an error response needs to be mapped based on the current mapping rule. If the result of the conditional expression is `true`, the error response is mapped based on the current rule.
  - `statusCode`: required. The HTTP status code that replaces the original HTTP status code of an error response if the error response needs to be mapped based on the current mapping rule.
  - `errorMessage`: optional. The error message that is returned to the client after mapping. The value of this parameter is obtained from the parameters in the original backend error response and is also stored in the `errorMessage` parameter in error logs. In the error response after mapping, this parameter is displayed as the value of the `X-Ca-Error-Message` header field.
  - `responseHeaders`: optional. The response header fields that are included in the error response after mapping if the current mapping rule is hit. This parameter is specified as key-value pairs in the `map` format.
  - `responseBody`: optional. The response body that overwrites the original response body of an error response if the error response needs to be mapped based on the current mapping rule.

- `defaultMapping`: optional. The default mapping rule. If an error response does not hit rules that are defined in the `mappings` parameter, the error response is mapped based on the default mapping rule.
  - `statusCode`: required. The HTTP status code that replaces the original HTTP status code of an error response if the error response needs to be mapped based on the default mapping rule.
  - `errorMessage`: optional. The error message that is returned to the client after mapping. The value of this parameter is obtained from the parameters in the original backend error response and is also stored in the `errorMessage` parameter in error logs. In the error response after mapping, this parameter is displayed as the value of the `X-Ca-Error-Message` header field.
  - `responseHeaders`: optional. The response header fields that are included in the error response after mapping if the default mapping rule is hit. This parameter is specified as key-value pairs in the `map` format.
  - `responseBody`: optional. The response body that overwrites the original response body of an error response if the error response needs to be mapped based on the default mapping rule.

Take note of the following items when you configure a plug-in of the Error Mapping type:

- The parameters that are used to write conditional expressions in `mappingCondition` and `mappings[].condition` must be defined in the `parameters` parameter. Otherwise, the plug-in does not work and reports an error. For information about how to define parameters and write conditional expressions, see Use parameters and conditional expressions.
- The value of the `errorCode` parameter must be the name of a parameter that is defined in the `parameters` parameter.
- When you configure a mapping rule, you must specify at least one of the `code` and `condition` parameters. When you specify the `code` parameter, the value of this parameter must be unique among all mapping rules. When you specify the `condition` parameter, you must write conditional expressions in the order that meets your requirements. This is because the order of conditions determines their priorities.
- You can replace the `errorMessage` and `responseBody` parameters with the `"${Code}: ${Message}"` format. The values of the Code and Message parameters in this format must be obtained from the parameters that are defined in the `parameters` parameter.
- You can also replace the `responseHeaders` parameter with the `${Message}` format.
- If you do not specify the `responseBody` parameter, the response body of the error response that is returned to the client after mapping is the same as that of the original error response.
- You can use the `responseHeaders` parameter to specify header fields and their values to replace corresponding header fields in a backend error response. If you specify the value of a header field as `''`, this header field will be deleted after mapping. If you do not specify this parameter, the header fields of the error response that is returned to the client after mapping are the same as those of the original error response.
- If you do not specify the `defaultMapping` parameter and an error response does not hit mapping rules, the original backend error response is returned to the client.

3.2. Parameters involved in mapping

As shown in the following code snippet, you must specify parameters that are involved in mapping as key-value pairs in the `parameters` parameter. Each key is the name of a parameter. Each value is specified in the `Location:Name` format. This format indicates that the value of the parameter is obtained from a response or the context of the system.

```
---
# The parameters that are involved in mapping.
parameters:
 statusCode: "StatusCode"
 resultCode: "BodyJsonField:$.result_code"
 resultId: "BodyJsonField:$.req_msg_id"
```

You can specify the following locations when you use the `Location:Name` format to obtain parameter values. For more information about the locations, see Use parameters and conditional expressions.

Location	Source	Description
StatusCode	Responses	The HTTP status code in a backend error response, such as `200` or `400`.
ErrorCode	Responses	The error code of a system error response in API Gateway.
ErrorMessage	Responses	The error message of a system error response in API Gateway.
Header	Responses	Use `Header:{Name}` to obtain the value of the HTTP header field that is specified by `{Name}`. If multiple HTTP header fields have the same name, the value of the first header field with the name is obtained.
BodyJsonField	Responses	Use `BodyJsonField:{JPath}` to obtain the JSON string in the body of an API request or a backend response. `{JPath}` is a JSONPath expression.
System	Responses	Use `System:{Name}` to obtain the value of the system parameter that is specified by `{Name}`.

Location	Source	Description
Token	Responses	If JSON Web Token (JWT) is used with OAuth2 for authentication, you can use `Token:{Name}` to obtain the value of the parameter that is specified by `{Name}`, in a token.

- The `ErrorCode` and `ErrorMessage` locations are used to obtain error codes and error messages in system error responses in API Gateway. For more information, see Error code table.
- The `BodyJsonField` location can be used to obtain the JSON string in the body of a backend response. However, if the size of the response body exceeds 16,380 bytes, the obtained string is `null`.

3.3. Working mechanism

The following actions describe how a plug-in of the Error Mapping type works:

- i. Step 1: Based on the list of parameters that are defined in the `parameters` parameter, obtain the values of the parameters from a backend error response and the context of the system.
- i. Step 2: Use the parameters and obtained values to execute the conditional expression that is written in the `errorCondition` parameter. If the result is `true`, go to the next step. If the result is `false`, the process ends.
- i. Step 3: If the `errorCode` parameter is specified, obtain the error code that is specified by the value of the `errorCode` parameter. Then, check if the error code is equal to the value of the `code` parameter in mapping rules.
- i. Step 4: If no mapping rule is hit in step 3, execute in sequence the conditional expressions that are written in the `condition` parameter in mapping rules.
- i. Step 5: If a mapping rule is hit in step 3 or step 4, the original error response is mapped based on the mapping rule. Otherwise, the original error response is mapped based on the default mapping rule.

3.4. Mapping of system error responses and error logs

- In API Gateway, system errors may occur in processes such as checking, verification, throttling, and plug-in operation. For more information, see Error code table. You can use `ErrorCode` as a location to obtain information in a system error response. For example, clients support only HTTP status code 200 and want to map HTTP status code 429 that is returned by API Gateway to HTTP status code 200.
- For a system error response, the values that are obtained from locations such as `StatusCode`, `Header`, and `BodyJsonField` are all `null`. When you define the mapping condition for a plug-in of the Error Mapping type, note that for a backend error response, the value that is obtained from the `ErrorCode` location is `0`.
- The error code of a system error response is returned as the value of the `X-Ca-Error-Code` header field and is also stored in the `errorCode` parameter in error logs. This value cannot be overwritten by using a plug-in of the Error Mapping type.

- The `statusCode` parameter in error logs records the value of the HTTP status code that is sent from API Gateway to the client. This value can be overwritten by using a plug-in of the Error Mapping type.

4. Configuration examples

4.1. Use error codes in error responses for error mapping

Mapping

```
---
# The parameters that are involved in mapping.
parameters:
 statusCode: "StatusCode"
 resultCode: "BodyJsonField:$.result_code"
 resultId: "BodyJsonField:$.req_msg_id"
# The mapping condition under which an error response needs to be mapped.
errorCondition: "$statusCode = 200 and $resultCode <> 'OK'"
# The parameter in an error response that is used to specify the error code and hit mapping rules.
errorCode: "resultCode"
# The mapping rules.
mappings:
 - code: "ROLE_NOT_EXISTS"
   statusCode: 404
   errorMessage: "Role Not Exists, RequestId=${resultId}"
 - code: "INVALID_PARAMETER"
   statusCode: 400
   errorMessage: "Invalid Parameter, RequestId=${resultId}"
# Optional. The default mapping rule.
defaultMapping:
 statusCode: 500
 errorMessage: "Unknown Error, ${resultCode}, RequestId=${resultId}"
```

5. Limits

- A maximum of 16 parameters can be defined in each plug-in of the Error Mapping type.
- Each conditional expression can contain a maximum of 512 characters.
- If you use the `BodyJsonField` location to obtain the JSON string in the body of an error response, the response body can be a maximum of 16,380 bytes in size. If the size of the response body exceeds this limit, the obtained string is null.
- For each plug-in of the Error Mapping type, you can configure a maximum of 16,380 bytes of metadata.
- For each plug-in of the Error Mapping type, you can configure a maximum of 20 mapping rules by using the `condition` parameter.