目錄

The OpenAPI Specification | 开放API规范

中文翻译进行中，欢迎大家协助翻译

https://github.com/fishead/OpenAPI-Specification

翻译进度

- [x] Definitions
  - [x] OpenAPI Document
  - [x] Path Templating
  - [x] Media Types
  - [x] HTTP Status Codes
- [ ] Specification
  - [x] Versions
  - [x] Format
  - [ ] Document Structure
  - [ ] Data Types
  - [ ] Rich Text Formatting
  - [ ] Relative References In URLs
  - [ ] Schema
    - [x] OpenAPI Object
    - [x] Info Object
    - [x] Contact Object
    - [x] License Object
    - [x] Server Object
    - [x] Server Variable Object
    - [x] Components Object
    - [x] Paths Object
    - [x] Path Item Object
    - [x] Operation Object
    - [x] External Documentation Object
    - [x] Parameter Object
    - [x] Request Body Object
    - [x] Media Type Object
    - [x] Encoding Object
    - [x] Responses Object
    - [ ] Response Object

The OpenAPI Specification is a community driven, open specification within the Open API Initiative, a Linux Foundation Collaborative Project.

The OpenAPI Specification (OAS) defines a standard, programming language-agnostic interface description for REST APIs, which allows both humans and computers to discover and understand the capabilities of a service without requiring access to source code, additional documentation, or inspection of network traffic. When properly defined via OpenAPI, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. Similar to what interface descriptions have done for lower-level programming, the OpenAPI Specification removes guesswork in calling a service.

Use cases for machine-readable API definition documents include, but are not limited to, interactive documentation; code generation for documentation, clients, and servers; and automation of test cases. OpenAPI documents describe an API's services and are represented in either YAML or JSON formats. These documents may either be produced and served statically or be generated dynamically from an application.

The OpenAPI Specification does not require rewriting existing APIs. It does not require binding any software to a service—the service being described may not even be owned by the creator of its description. It does, however, require the capabilities of the service be described in the structure of the OpenAPI Specification. Not all services can be described by OpenAPI—this specification is not intended to cover every possible style of REST APIs. The OpenAPI Specification does not mandate a specific development process such as design-first or code-first. It does facilitate either technique by establishing clear interactions with a REST API.

This GitHub project is the starting point for OpenAPI. Here you will find the information you need about the OpenAPI Specification, simple examples of what it looks like, and some general information regarding the project.

Current Version - 3.0

The current version of the OpenAPI specification is OpenAPI Specification 3.0.

Future Versions

3.0.1 - The next PATCH version. Patch-level fixes (typos, clarifications, etc.) should be submitted against this branch.

Previous Versions

This repository also contains the OpenAPI Specification 2.0, which is identical to the Swagger 2.0 specification before it was renamed to “OpenAPI Specification”, as well as the Swagger 1.2 and Swagger 2.0 specifications.

Each folder in this repository, such as examples and schemas, should contain folders pertaining to the current and previous versions of the specification.

See It in Action

If you just want to see it work, check out the list of current examples.

Tools and Libraries

Looking to see how you can create your own OpenAPI definition, present it, or otherwise use it? Check out the growing list of 3.0 Implementations.

Participation

The current process for development of the OpenAPI Specification is described in Development Guidelines. Development of the next version of the OpenAPI Specification is guided by the Technical Developer Community (TDC). This group of committers bring their API expertise, incorporate feedback from the community, and expand the group of committers as appropriate. All development activity on the future specification will be performed as features and merged into this branch. Upon release of the future specification, this branch will be merged to master.

The TDC holds weekly web conferences to review open pull requests and discuss open issues related to the evolving OpenAPI Specification. Participation in weekly calls and scheduled working sessions is open to the community. You can view the TDC calendar online, and import it to your calendar using the iCal link.

The Open API Initiative encourages participation from individuals and companies alike. If you want to participate in the evolution of the OpenAPI Specification, consider taking the following actions:

- Review the current specification. The human-readable markdown file *is the source of truth* for the specification.
- Review the development process so you understand how the spec is evolving.
- Check the issues and pull requests to see if someone has already documented your idea or feedback on the specification. You can follow an existing conversation by adding a comment to the existing issue or PR.
- Create an issue to describe a new concern. If possible, propose a solution.

Not all feedback can be accommodated and there may be solid arguments for or against a change being appropriate for the specification.

License

See: License (Apache-2.0)

开放API规范

版本 3.0.0

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 RFC2119 RFC8174 when, and only when, they appear in all capitals, as shown here.

介绍

The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic.

An OpenAPI definition can then be used by documentation generation tools to display the API, code generation tools to generate servers and clients in various programming languages, testing tools, and many other use cases.

目录

术语定义

开放API文档

一（或多）份用来定义或描述一个API的文档。

路径模板

路径模板指用大括号标记来标记一段URL作为可替换的路径参数。

媒体类型

媒体类型定义分散于多处。 媒体类型定义应当符合RFC6838。

以下是一些媒体类型定义的示例：

```
text/plain; charset=utf-8
application/json
application/vnd.github+json
application/vnd.github.v3+json
application/vnd.github.v3.raw+json
application/vnd.github.v3.text+json
application/vnd.github.v3.html+json
application/vnd.github.v3.full+json
application/vnd.github.v3.diff
application/vnd.github.v3.patch
```

HTTP状态码

HTTP状态码被用来表示一次请求的被执行状态。 RFC7231定义了有效的状态码，可以在IANA Status Code Registry找到已经被注册的状态码的列表。

规范

版本

开放API规范使用符合语义化版本 2.0.0(semver)规范的版本号。

语义化版本的 `主版本号` 、 `次版本号` 部分（比如 `3.0` ）应当被用来标记开放API规范的特性变动。通常 `.修订号` 版本被用来表示本文档文档的错误修正而不是特性变动。支持开放API规范3.0的工具应该兼容所有3.0.*的版本，工具不应当关注修订版本号，比如 `3.0.0` 和 `3.0.1` 对它来说应该没有任何区别。

此后开放API规范的相同主版本号下更高次要版本的发布不应当对面向低于此次要版本号开发的工具的造成干扰。因此 `3.1.0` 版本的规范应当可以在面向 `3.0.0` 版本规范开发的工具内使用。

任何兼容开放API规范 3.*.* 的文档应当包含一个 `openapi` 字段用来表明它使用的规范的语义化版本。

Format | 格式

一份遵从开放API规范的文档是一个自包含的JSON对象，可以使用JSON或YAML格式编写。

比如一个字段有一组值，用JSON格式表示为：

```
{
   "field": [ 1, 2, 3 ]
}
```

规范内的所有字段名都是小写。

本纲要提供了两种类型的字段：固定字段和模式字段，固定字段表示字段有固定的命名，模式字段表示命名需要符合一定的模式。

模式字段必须包含它的对象内的名字必须是唯一的。

为了保留在 YAML 和 JSON 格式之间转换的能力，推荐使用1.2版本的YAML格式，而且还需要符合以下限制：

- Tags 必须被限制在JSON Schema ruleset允许的范围内。
- Keys 必须是YAML Failsafe schema ruleset规范定义的纯字符串。

注意： 虽然API文档是使用 YAML 或 JSON 格式书写的，但是API的请求体和响应体或者其他内容可以是任何格式。

文档结构

An OpenAPI document MAY be made up of a single document or be divided into multiple, connected parts at the discretion of the user. In the latter case, `$ref` fields MUST be used in the specification to reference those parts as follows from the JSON Schema definitions.

推荐将根开放API文档命名为 `openapi.json` 或 `openapi.yaml` 。

数据类型

Primitive data types in the OAS are based on the types supported by the JSON Schema Specification Wright Draft 00. Note that `integer` as a type is also supported and is defined as a JSON number without a fraction or exponent part. `null` is not supported as a type (see `nullable` for an alternative solution). Models are defined using the Schema Object, which is an extended subset of JSON Schema Specification Wright Draft 00.

Primitives have an optional modifier property: `format` . OAS uses several known formats to define in fine detail the data type being used. However, to support documentation needs, the `format` property is an open `string` -valued property, and can have any value. Formats such as `"email"` , `"uuid"` , and so on, MAY be used even though undefined by this

specification. Types that are not accompanied by a `format` property follow the type definition in the JSON Schema. Tools that do not recognize a specific `format` MAY default back to the `type` alone, as if the `format` is not specified.

The formats defined by the OAS are:

Common Name	`type`	`format`	Comments
integer	`integer`	`int32`	signed 32 bits
long	`integer`	`int64`	signed 64 bits
float	`number`	`float`	
double	`number`	`double`	
string	`string`		
byte	`string`	`byte`	base64 encoded characters
binary	`string`	`binary`	any sequence of octets
boolean	`boolean`		
date	`string`	`date`	As defined by `full-date` - RFC3339
dateTime	`string`	`date-time`	As defined by `date-time` - RFC3339
password	`string`	`password`	A hint to UIs to obscure input.

富文本格式

Throughout the specification `description` fields are noted as supporting CommonMark markdown formatting. Where OpenAPI tooling renders rich text it MUST support, at a minimum, markdown syntax as described by CommonMark 0.27. Tooling MAY choose to ignore some CommonMark features to address security concerns.

URL的相对引用

Unless specified otherwise, all properties that are URLs MAY be relative references as defined by RFC3986. Relative references are resolved using the URLs defined in the `Server Object` as a Base URI.

Relative references used in `$ref` are processed as per JSON Reference, using the URL of the current document as the base URI. See also the Reference Object.

纲要

在接下来的叙述中，如果一个字段没有被明确的标记为 必选 或者被描述为 必须 或 应当，那么可以认为它是一个 可选 字段

OpenAPI 对象

这是OpenAPI document的根文档对象。

固定字段

字段名	类型	描述
openapi	`string`	必选. 这个字符串必须是开放API规范版本号提到的符合语义化版本号规范的版本号。 `openapi` 字段应该被工具或者客户端用来解释开放API文档。这个值和API `info.version` 字符串没有关联。
info	Info 对象	必选。此字段提供API相关的元数据。相关工具可能需要这个字段。
servers	[Server 对象]	这是一个Server对象的数组， 提供到服务器的连接信息。如果没有提供 `servers` 属性或者是一个空数组，那么默认为是url值为 `/` 的 Server 对象 。
paths	Paths 对象	必选。对所提供的API有效的路径和操作。
components	Components 对象	一个包含多种纲要的元素。
security	[Security Requirement 对象]	声明API使用的安全机制。The list of values includes alternative security requirement objects that can be used. 认证一个请求时仅允许使用一种安全机制。单独的操作可以覆盖这里的定义。
tags	[Tag 对象]	提供更多元数据的一系列标签，标签的顺序可以被转换工具用来决定API的顺序。不是所有被Operation 对象用到的标签都必须被声明。没有被声明的标签可能被工具按自己的逻辑任意整理，每个标签名都应该是唯一的。
externalDocs	External Documentation 对象	附加的文档。这个对象可能会被规范扩展扩展。

Info 对象

这个对象提供API的元数据。如果客户端需要时可能会用到这些元数据，而且可能会被呈现在编辑工具或者文档生成工具中。

固定字段

字段名	类型	描述
title	`string`	必选. 应用的名称。
description	`string`	对应用的简短描述。 CommonMark syntax 可以被用来表示富文本呈现。
termsOfService	`string`	指向服务条款的URL地址，必须是URL地址格式。
contact	Contact Object	所开放的API的联系人信息。
license	License Object	所开放的API的证书信息。
version	`string`	必选. API文档的版本信息（注意：这个版本和开放API规范版本没有任何关系）。

Info 对象示例:

```
{
  "title": "Sample Pet Store App",
  "description": "This is a sample server for a pet store.",
  "termsOfService": "http://example.com/terms/",
  "contact": {
    "name": "API Support",
    "url": "http://www.example.com/support",
    "email": "support@example.com"
  },
  "license": {
    "name": "Apache 2.0",
    "url": "http://www.apache.org/licenses/LICENSE-2.0.html"
  },
  "version": "1.0.1"
}
```

```
title: Sample Pet Store App
description: This is a sample server for a pet store.
termsOfService: http://example.com/terms/
contact:
  name: API Support
  url: http://www.example.com/support
  email: support@example.com
license:
  name: Apache 2.0
  url: http://www.apache.org/licenses/LICENSE-2.0.html
version: 1.0.1
```

Contact 对象

所公开的API的联系人信息

固定字段

字段名	类型	描述
name	`string`	人或组织的名称。
url	`string`	指向联系人信息的URL地址，必须是URL地址格式。
email	`string`	人或组织的email地址，必须是email地址格式。

这个对象可能会被规范扩展扩展。

Contact 对象示例：

```
{
  "name": "API Support",
  "url": "http://www.example.com/support",
  "email": "support@example.com"
}
```

```
name: API Support
url: http://www.example.com/support
email: support@example.com
```

License 对象

公开API的证书信息。

固定字段

字段名	类型	描述
name	`string`	必选. API的证书名。
url	`string`	指向API所使用的证书的URL地址，必须是URL地址格式。

这个对象可能会被规范扩展扩展。

License 对象示例：

```
{
  "name": "Apache 2.0",
  "url": "http://www.apache.org/licenses/LICENSE-2.0.html"
}
```

```
name: Apache 2.0
url: http://www.apache.org/licenses/LICENSE-2.0.html
```

Server 对象

表示一个服务器的对象。

固定字段

字段名	类型	描述
url	`string`	必选. 指向目标主机的URL地址。这个URL地址支持服务器变量而且可能是相对路径，表示主机路径是相对于本文档所在的路径。当一个变量被命名为类似 `{` brackets `}` 时需要替换此变量。
description	`string`	一个可选的字符串，用来描述此URL地址。CommonMark syntax可以被用来呈现富文本格式.
variables	Map[`string` , Server Variable Object]	一组变量和值的映射，这些值被用来替换服务器URL地址内的模板参数。

这个对象可能会被规范扩展扩展。

Server 对象示例

单个服务器可以这样描述：

```
{
  "url": "https://development.gigantic-server.com/v1",
  "description": "Development server"
}
```

```
url: https://development.gigantic-server.com/v1
description: Development server
```

以下内容表示的是有多个服务器时应该如何描述，比如OpenAPI 对象的 `servers` ：

```
{
  "servers": [
    {
      "url": "https://development.gigantic-server.com/v1",
      "description": "Development server"
    },
    {
      "url": "https://staging.gigantic-server.com/v1",
      "description": "Staging server"
    },
    {
      "url": "https://api.gigantic-server.com/v1",
      "description": "Production server"
    }
  ]
}
```

```
servers:
- url: https://development.gigantic-server.com/v1
  description: Development server
- url: https://staging.gigantic-server.com/v1
  description: Staging server
- url: https://api.gigantic-server.com/v1
  description: Production server
```

以下内容展示了如何使用变量来配置服务器：

```
{
  "servers": [
    {
      "url": "https://{username}.gigantic-server.com:{port}/{basePath}",
      "description": "The production API server",
      "variables": {
        "username": {
          "default": "demo",
          "description": "this value is assigned by the service provider, in this exam
ple `gigantic-server.com`"
        },
        "port": {
          "enum": [
            "8443",
            "443"
          ],
          "default": "8443"
        },
        "basePath": {
          "default": "v2"
        }
      }
    }
  ]
}
```

```
servers:
- url: https://{username}.gigantic-server.com:{port}/{basePath}
  description: The production API server
  variables:
    username:
      # note! no enum here means it is an open value
      default: demo
      description: this value is assigned by the service provider, in this example `gi
gantic-server.com`
    port:
      enum:
        - '8443'
        - '443'
      default: '8443'
    basePath:
      # open meaning there is the opportunity to use special base paths as assigned by
 the provider, default is `v2`
      default: v2
```

Server Variable 对象

表示可用于服务器URL地址模板变量替换的对象。

固定字段

字段名	类型	描述
enum	[`string`]	一组可枚举字符串值，当可替换选项只能设置为固定的某些值时使用。
default	`string`	必选. 当可替换的值没有被使用者指定时使用的默认值。不像 Schema Object's的 `default` ，这个值必须由使用者提供。
description	`string`	对服务器变量的可选的描述。CommonMark syntax可以被用来呈现富文本格式.

这个对象可能会被规范扩展扩展。

Components 对象

包含开放API规范固定的各种可重用组件。当没有被其他对象引用时，在这里定义定义的组件不会产生任何效果。

固定字段

字段名	类型	描述	
schemas	Map[`string` , Schema Object |	Reference Object]	定义可重用的 Schema 对象 的对象。
responses	Map[`string` , Response Object |	Reference Object]	定义可重用的 Response 对象 的对象。
parameters	Map[`string` , Parameter Object |	Reference Object]	定义可重用的 Parameter 对象 的对象。
examples	Map[`string` , Example Object |	Reference Object]	定义可重用的 Example 对象 的对象。
requestBodies	Map[`string` , Request Body Object |	Reference Object]	定义可重用的 Request Body 对象 的对象。
headers	Map[`string` , Header Object |	Reference Object]	定义可重用的 Header 对象 的对象。
securitySchemes	Map[`string` , Security Scheme Object |	Reference Object]	定义可重用的 Security Scheme 对象 的对象。
links	Map[`string` , Link Object |	Reference Object]	定义可重用的 Link 对象 的对象。
callbacks	Map[`string` , Callback Object |	Reference Object]	定义可重用的 Callback 对象 的对象。

这个对象可能会被 规范扩展 扩展。

上面定义的所有固定字段的值都是对象，对象包含的key的命名必须满足正则表达式： `^[a-zA-Z0-9\.\-_]+$` 。

字段名示例:

```
User
User_1
User_Name
user-name
my.org.User
```

Components 对象示例

```
"components": {
  "schemas": {
    "Category": {
      "type": "object",
      "properties": {
        "id": {
          "type": "integer",
          "format": "int64"
        },
        "name": {
          "type": "string"
        }
      }
    },
    "Tag": {
      "type": "object",
      "properties": {
        "id": {
          "type": "integer",
          "format": "int64"
        },
        "name": {
          "type": "string"
        }
      }
    }
  },
  "parameters": {
    "skipParam": {
      "name": "skip",
      "in": "query",
      "description": "number of items to skip",
      "required": true,
      "schema": {
        "type": "integer",
        "format": "int32"
      }
```

```
    },
    "limitParam": {
      "name": "limit",
      "in": "query",
      "description": "max records to return",
      "required": true,
      "schema" : {
        "type": "integer",
        "format": "int32"
      }
    }
  },
  "responses": {
    "NotFound": {
      "description": "Entity not found."
    },
    "IllegalInput": {
      "description": "Illegal input for operation."
    },
    "GeneralError": {
      "description": "General Error",
      "content": {
        "application/json": {
          "schema": {
            "$ref": "#/components/schemas/GeneralError"
          }
        }
      }
    }
  },
  "securitySchemes": {
    "api_key": {
      "type": "apiKey",
      "name": "api_key",
      "in": "header"
    },
    "petstore_auth": {
      "type": "oauth2",
      "flows": {
        "implicit": {
          "authorizationUrl": "http://example.org/api/oauth/dialog",
          "scopes": {
            "write:pets": "modify pets in your account",
            "read:pets": "read your pets"
          }
        }
      }
    }
  }
}
```

```
components:
  schemas:
    Category:
      type: object
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
    Tag:
      type: object
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
  parameters:
    skipParam:
      name: skip
      in: query
      description: number of items to skip
      required: true
      schema:
        type: integer
        format: int32
    limitParam:
      name: limit
      in: query
      description: max records to return
      required: true
      schema:
        type: integer
        format: int32
  responses:
    NotFound:
      description: Entity not found.
    IllegalInput:
      description: Illegal input for operation.
    GeneralError:
      description: General Error
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/GeneralError'
  securitySchemes:
    api_key:
      type: apiKey
      name: api_key
      in: header
    petstore_auth:
```

```
      type: oauth2
      flows:
        implicit:
          authorizationUrl: http://example.org/api/oauth/dialog
          scopes:
            write:pets: modify pets in your account
            read:pets: read your pets
```

Paths 对象

定义各个的端点和操作的相对路径。这里指定的路径会和 `Server 对象` 内指定的URL地址组成完整的URL地址，路径可以为空，这依赖于 ACL constraints 的设置。

模式字段

字段名模式	类型	描述
/{path}	Path Item 对象	到各个端点的相对路径，路径必须以 `/` 打头，这个路径会被直接连接到 `Server 对象` 的 `url` 字段以组成完整URL地址（不会考虑是否是相对路径）。这里可以使用 Path templating ，当做URL地址匹配时，不带路径参数的路径会被优先匹配。应该避免定义多个具有相同路径层级但是路径参数名不同的路径，因为他们是等价的。当匹配出现歧义时，由使用的工具自行决定使用那个路径。

个对象可能会被规范扩展扩展。

路径模板匹配

假设有以下路径，明确定义的路径 `/pets/mine` 会被优先匹配：

```
  /pets/{petId}
  /pets/mine
```

以下路径被认为是等价的而且是无效的：

```
  /pets/{petId}
  /pets/{name}
```

以下路径会产生歧义：

```
  /{entity}/me
  /books/{id}
```

Paths 对象示例

```
{
  "/pets": {
    "get": {
      "description": "Returns all pets from the system that the user has access to",
      "responses": {
        "200": {
          "description": "A list of pets.",
          "content": {
            "application/json": {
              "schema": {
                "type": "array",
                "items": {
                  "$ref": "#/components/schemas/pet"
                }
              }
            }
          }
        }
      }
    }
  }
}
```

```
/pets:
  get:
    description: Returns all pets from the system that the user has access to
    responses:
      '200':
        description: A list of pets.
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/pet'
```

Path Item 对象

描述对一个路径可执行的有效操作。依赖与 ACL constraints 的设置，一个Path Item可以是一个空对象，文档的读者仍然可以看到这个路径，但是他们将无法了解到对这个路径可用的任何操作和参数。

固定字段

字段名	类型	描述
		指定对此路径的外部定义的引用，引

$ref	`string`	用的格式必须符合 Path Item 对象 的格式，如果引用的外部定义和此对象内的其他定义有冲突，该如何处理冲突尚未被定义。	
summary	`string`	一个可选的简要总结字符串，用来描述此路径内包含的所有操作。	
description	`string`	一个可选的详细说明字符串，用于描述此路径包含的所有操作。CommonMark syntax可以被用来呈现富文本格式.	
get	Operation 对象	定义适用于此路径的 GET 操作。	
put	Operation 对象	定义适用于此路径的 PUT 操作。	
post	Operation 对象	定义适用于此路径的 POST 操作.	
delete	Operation 对象	定义适用于此路径的 DELETE 操作。	
options	Operation 对象	定义适用于此路径的 OPTIONS 操作。	
head	Operation 对象	定义适用于此路径的 HEAD 操作。	
patch	Operation 对象	定义适用于此路径的 PATCH 操作。	
trace	Operation 对象	定义适用于此路径的 TRACE 操作。	
servers	[Server 对象]	一个可用于此路径所有操作的替代根 `server` 的数组定义。	
	[Parameter		一个可用于此路径下所有操作的参数的列表。这些参数可以被具体的操作定义覆盖，但是不能被移除。这个列表禁止包含重复的参数，一个唯一的参数名由 name 和 location

	对象 \		的组合来定义。这个列表可以使用 Reference 格式引用定义在 OpenAPI 对象 components/parameters 内的参数。

这个对象可能会被规范扩展扩展。

Path Item 对象示例

```
{
  "get": {
    "description": "Returns pets based on ID",
    "summary": "Find pets by ID",
    "operationId": "getPetsById",
    "responses": {
      "200": {
        "description": "pet response",
        "content": {
          "*/*": {
            "schema": {
              "type": "array",
              "items": {
                "$ref": "#/components/schemas/Pet"
              }
            }
          }
        }
      },
      "default": {
        "description": "error payload",
        "content": {
          "text/html": {
            "schema": {
              "$ref": "#/components/schemas/ErrorModel"
            }
          }
        }
      }
    }
  },
  "parameters": [
    {
      "name": "id",
      "in": "path",
      "description": "ID of pet to use",
      "required": true,
      "schema": {
        "type": "array",
        "items": {
          "type": "string"
        }
      },
      "style": "simple"
    }
  ]
}
```

```yaml
get:
  description: Returns pets based on ID
  summary: Find pets by ID
  operationId: getPetsById
  responses:
    '200':
      description: pet response
      content:
        '*/*' :
          schema:
            type: array
            items:
              $ref: '#/components/schemas/Pet'
    default:
      description: error payload
      content:
        'text/html':
          schema:
            $ref: '#/components/schemas/ErrorModel'
parameters:
- name: id
  in: path
  description: ID of pet to use
  required: true
  schema:
    type: array
    style: simple
    items:
      type: string
```

Operation Object

描述对路径的某个操作。

固定字段

字段名	类型	描述
tags	[`string`]	用于控制API文档的标签列表，标签可以用于在逻辑上分组对资源的操作或作为其它用途的先决条件。
summary	`string`	对此操作行为的简短描述。
description	`string`	对此操作行为的详细解释。CommonMark

		syntax可以被用来呈现富文本格式.	
externalDocs	External Documentation 对象	附加的外部文档。	
operationId	`string`	用于标识此操作的唯一字符串，这个id在此API内包含的所有操作中必须是唯一的。相关的工具和库可能会使用此operationId来唯一的标识一个操作，因此推荐在命名时符合一般的编程命名习惯。	
parameters	[Parameter 对象 \	Reference 对象]	定义可用于此操作的参数列表，如果一个同名的参数已经存在于 Path Item，那么这里的定义会覆盖它但是不能移除上面的定义。这个列表不允许包含重复的参数，参数的唯一性由 name 和 location 的组合来确定。这个列表可以使用 Reference 对象 来连接定义于 OpenAPI 对象 components/parameters 的参数。
requestBody	Request Body 对象 \	Reference 对象	可用于此操作的请求体。`requestBody` 只能被用于 HTTP 1.1 规范 RFC7231 中明确定义了包含请求体的请求方法，在其他没有明确定义的请求方法中，`requestBody` 的消费者应该应该忽略 `requestBody` 。
responses	Responses 对象	必选. 定义执行此操作后的可能的响应值列表。	
callbacks	Map[`string` , Callback 对象 \	Reference 对象]	一组相对于父操作的可能出现的回调映射，A map of possible out-of band callbacks related to the parent operation. 映射中的每一个键都唯一的映射一个 Callback 对象， that describes a request that may be initiated by the API provider and the expected

			responses. The key value used to identify the callback object is an expression, evaluated at runtime, that identifies a URL to use for the callback operation.
deprecated	`boolean`	声明此操作已经被废弃，使用者应该尽量避免使用此操作，默认的值是 `false` 。	
security	[Security Requirement 对象]	声明那种安全机制可用于此操作。这个列表可以包含多种可用于此操作的安全需求对象，但是在认证一个请求时应该仅使用其中一种。这里的定义会覆盖任何在顶层 `security` 中的安全声明，因此可以声明一个空数组来变相的移除顶层的安全声明。	
servers	[Server 对象]	一个可用于此操作的额外的 `server` 数组，这里的定义会覆盖 Path Item 对象 或 顶层的定义。	

这个对象可能会被规范扩展扩展。

Operation 对象示例

```
{
  "tags": [
    "pet"
  ],
  "summary": "Updates a pet in the store with form data",
  "operationId": "updatePetWithForm",
  "parameters": [
    {
      "name": "petId",
      "in": "path",
      "description": "ID of pet that needs to be updated",
      "required": true,
      "schema": {
        "type": "string"
```

```
      }
    }
  ],
  "requestBody": {
    "content": {
      "application/x-www-form-urlencoded": {
        "schema": {
          "type": "object",
           "properties": {
              "name": {
                "description": "Updated name of the pet",
                "type": "string"
              },
              "status": {
                "description": "Updated status of the pet",
                "type": "string"
            }
           },
        "required": ["status"]
        }
      }
    }
  },
  "responses": {
    "200": {
      "description": "Pet updated.",
      "content": {
        "application/json": {},
        "application/xml": {}
      }
    },
    "405": {
      "description": "Invalid input",
      "content": {
        "application/json": {},
        "application/xml": {}
      }
    }
  },
  "security": [
    {
      "petstore_auth": [
        "write:pets",
        "read:pets"
      ]
    }
  ]
}
```

```
tags:
- pet
summary: Updates a pet in the store with form data
operationId: updatePetWithForm
parameters:
- name: petId
  in: path
  description: ID of pet that needs to be updated
  required: true
  schema:
    type: string
requestBody:
  content:
    'application/x-www-form-urlencoded':
      schema:
       properties:
          name:
            description: Updated name of the pet
            type: string
          status:
            description: Updated status of the pet
            type: string
       required:
         - status
responses:
  '200':
    description: Pet updated.
    content:
      'application/json': {}
      'application/xml': {}
  '405':
    description: Invalid input
    content:
      'application/json': {}
      'application/xml': {}
security:
- petstore_auth:
  - write:pets
  - read:pets
```

External Documentation 对象

允许引用外部资源来扩展文档。

固定字段

字段名	类型	描述
description	`string`	对引用的外部文档的简短描述。CommonMark syntax可以被用来呈现富文本格式.
url	`string`	必选. 外部文档的URL地址，这个值必须是URL地址格式。

这个对象可能会被规范扩展扩展。

External Documentation 对象示例

```
{
  "description": "Find more info here",
  "url": "https://example.com"
}
```

```
description: Find more info here
url: https://example.com
```

Parameter Object

描述一个操作参数。

一个参数的唯一性由 name 和 location 的组合来确定。

参数位置

有4种可能的参数位置值可用于 `in` 字段：

- path - 与 Path Templating 一起使用，当参数的值是URL操作路径的一部分时可以使用，但是不包含主机地址或基础路径。比如在路径 `/items/{itemId}` 中，路径参数是 `itemId` 。
- query - 追加在URL地址之后的参数，比如 `/items?id=###` 中，查询参数是 `id` 。
- header - 请求中使用的自定义请求头，注意在 RFC7230 中规定，请求头的命名是不区分大小写的。
- cookie - 用于传递特定的cookie值。

固定字段

字段名	类型	描述
name	`string`	必选. 参数的名称。参数名是区分大小写。 • 如果 [`in`](#parameterIn) 的值是 `"path"`，那么 `name` 字段的值必须与其关联的 [Paths 对象] (#pathsObject) 内 [path](#pathsPath) 字段的定义相呼应，查看 [Path Templating](#pathTemplating) 了解更多信息。 • 如果 [`in`](#parameterIn) 的值是 `"header"` 而且 `name`字段的值是`"Accept"`, `"Content-Type"`或 `"Authorization"`之一，那么此参数定义应该被忽略。 • 除此之外的情况，`name`表示 [`in`](#parameterIn) 属性的名字.
in	`string`	必选. 参数的位置，可能的值有 "query", "header", "path" 或 "cookie"。
description	`string`	对此参数的简要描述，这里可以包含使用示例。CommonMark syntax可以被用来呈现富文本格式.
required	`boolean`	标明此参数是否是必选参数。如果 参数位置 的值是 `path` ，那么这个参数一定是 必选 的因此这里的值必须是 `true` 。其他的则视情况而定。此字段的默认值是 `false` 。
deprecated	`boolean`	标明一个参数是被弃用的而且应该尽快移除对它的使用。
allowEmptyValue	`boolean`	设置是否允许传递空参数，这只在参数值为 `query` 时有效，默认值是 `false` 。如果同时指定了 `style` 属性且值为 `n/a` （无法被序列化）,那么此字段 `allowEmptyValue` 应该被忽略。

序列化参数的规则有两种。 对于简单的场景， `schema` 和 `style` 可以用于描述参数的结构和语法。

字段名	类型	描述
style	`string`	描述根据参数值类型的不同如何序列化参数。默认值为（基于 `in` 字段的值）： `query` 对应 `form` ； `path` 对应 `simple` ； `header` 对应 `simple` ； `cookie` 对应 `form` 。
explode	`boolean`	当这个值为 `true` 时，参数值类型为 `array` 或 `object` 的参数使用数组内的值或对象的键值对生成带分隔符的参数值。对于其他类型的参数，这个字段没有任何影响。当 `style` 是 `form` 时，这里的

		默认值是 `true` ，对于其他 style 值类型，默认值是 `false` 。	
allowReserved	`boolean`	决定此参数的值是否允许不使用%号编码使用定义于RFC3986内的保留字符 `:/?#[]@!$&'()*+,;=` 。 这个属性仅用于 `in` 的值是 `query` 时，此字段的默认值是 `false` 。	
schema	Schema 对象 \	Reference 对象	定于适用于此参数的类型纲要。
example	Any	不同媒体类型的示例，示例应该符合响应的纲要的编码属性。各个 `example` 之间应该是独立的，而且如果一个引用的 `schema` 也包含一个示例，那么这里定义的示例应该 覆盖 `schema` 包含的示例。为了展现无法被恰当的用 JSON 或 YAML 格式展现的示例时，可以使用经过必要的编码的字符串值。	
examples	Map[`string` , Example 对象 \	Reference 对象]	不同媒体类型的示例。每个示例应该包含一个对应于指定编码格式的格式正确的值，这个 `examples` 映射内包含的对象应该不同于 `example` 内的值。而且如果一个引用的 `schema` 也包含一个示例，那么这里定义的示例应该 覆盖 `schema` 包含的示例。

对于更复杂的场景， `content` 属性可以定义参数的媒体类型和概要。一个参数必须且只能包含 `schema` 和 `content` 属性中的一个。当 `example` 或 `examples` 字段提供了 `schema` 对象时，示例必须遵照参数的序列化策略。

字段名	类型	描述
content	Map[`string` , Media Type Object]	一个定义参数如何呈现的键值对映射。键是媒体类型，值是对应媒体类型的示例数据，此键值对只能包含一组键值对。

样式值

已经定义好了一组 `style` 类型用于支持常见的通用的简单参数序列化。

样式	类型	in	描述	
matrix	`primitive` , `array` , `object`	`path`	Path 样式的参数，参见 RFC6570	
label	`primitive` , `array` , `object`	`path`	Label 样式的参数，参见 RFC6570	
form	`primitive` , `array` , `object`	`query` , `cookie`	Form 样式的参数，参见 RFC6570. 此选项替换定义于OpenAPI 2.0 中 `collectionFormat` 等于 `csv` (当 `explode` 值为 false) 或 `multi` (当 `explode` 值为 true)的情况。	
simple	`array`	`path` , `header`	Simple 样式的参数，参见 RFC6570. 此选项替换定义于OpenAPI 2.0 中 `collectionFormat` 等于 `csv` 的情况。	
spaceDelimited	`array`	`query`	空格分隔的数组值。此选项替换定义于 OpenAPI 2.0 中 `collectionFormat` equal to `ssv` 的情况。	
pipeDelimited	`array`	`query`	管道符`	的数组值。 此选项替 OpenAPI 2.0 中 collectionForm to pipes`的情况
deepObject	`object`	`query`	提供一种简单的方法来表示参数中的嵌套对象值.	

Style 示例

建设一个参数名为 `color` 包含如下之一的值：

```
string -> "blue"
array -> ["blue","black","brown"]
object -> { "R": 100, "G": 200, "B": 150 }
```

下面这个表展示了各个不同类型值之间的例子。

style	explode	empty	string	array
matrix	false	;color	;color=blue	;color=blue,black,brown
matrix	true	;color	;color=blue	;color=blue;color=black;color=br
label	false	.	.blue	.blue.black.brown
label	true	.	.blue	.blue.black.brown
form	false	color=	color=blue	color=blue,black,brown
form	true	color=	color=blue	color=blue&color=black&color=b
simple	false	n/a	blue	blue,black,brown
simple	true	n/a	blue	blue,black,brown
spaceDelimited	false	n/a	n/a	blue%20black%20brown
pipeDelimited	false	n/a	n/a	blue\
deepObject	true	n/a	n/a	n/a

这个对象可能会被规范扩展扩展。

Parameter 对象示例

一个值数组，数组元素为64位整数值的请求头参数：

```
{
  "name": "token",
  "in": "header",
  "description": "token to be passed as a header",
  "required": true,
  "schema": {
    "type": "array",
    "items": {
      "type": "integer",
      "format": "int64"
    }
  },
  "style": "simple"
}
```

```yaml
name: token
in: header
description: token to be passed as a header
required: true
schema:
  type: array
  items:
    type: integer
    format: int64
style: simple
```

一个值类型为字符串的路径参数：

```json
{
  "name": "username",
  "in": "path",
  "description": "username to fetch",
  "required": true,
  "schema": {
    "type": "string"
  }
}
```

```yaml
name: username
in: path
description: username to fetch
required: true
schema:
  type: string
```

一个值类型为字符串的可选查询参数，允许通过通过重复参数来传递多个值：

```json
{
  "name": "id",
  "in": "query",
  "description": "ID of the object to fetch",
  "required": false,
  "schema": {
    "type": "array",
    "items": {
      "type": "string"
    }
  },
  "style": "form",
  "explode": true
}
```

```
name: id
in: query
description: ID of the object to fetch
required: false
schema:
  type: array
  items:
    type: string
style: form
explode: true
```

一个任意格式的查询参数，允许使用指定类型的未定义参数：

```
{
  "in": "query",
  "name": "freeForm",
  "schema": {
    "type": "object",
    "additionalProperties": {
      "type": "integer"
    },
  },
  "style": "form"
}
```

```
in: query
name: freeForm
schema:
  type: object
  additionalProperties:
    type: integer
style: form
```

使用 `content` 定义序列化方法的复杂参数：

```
{
  "in": "query",
  "name": "coordinates",
  "content": {
    "application/json": {
      "schema": {
        "type": "object",
        "required": [
          "lat",
          "long"
        ],
        "properties": {
          "lat": {
            "type": "number"
          },
          "long": {
            "type": "number"
          }
        }
      }
    }
  }
}
```

```
in: query
name: coordinates
content:
  application/json:
    schema:
      type: object
      required:
        - lat
        - long
      properties:
        lat:
          type: number
        long:
          type: number
```

Request Body Object

定义请求体。

固定字段

字段名	类型	描述
description	`string`	对请求体的简要描述，可以包含使用示例，CommonMark syntax可以被用来呈现富文本格式.
content	Map[`string` , Media Type Object]	必选. 请求体的内容。请求体的属性key是一个媒体类型或者媒体类型范围，值是对应媒体类型的示例数据。对于能匹配多个key的请求，定义更明确的请求会更优先被匹配。比如 `text/plain` 会覆盖 `text/*` 的定义。
required	`boolean`	指定请求体是不是应该被包含在请求中，默认值是 `false` 。

这个对象可能会被规范扩展扩展。

Request Body 示例

一个引用了模型定义的请求体。

```
{
  "description": "user to add to the system",
  "content": {
    "application/json": {
      "schema": {
        "$ref": "#/components/schemas/User"
      },
      "examples": {
          "user" : {
            "summary": "User Example",
            "externalValue": "http://foo.bar/examples/user-example.json"
          }
        }
    },
    "application/xml": {
      "schema": {
        "$ref": "#/components/schemas/User"
      },
      "examples": {
          "user" : {
            "summary": "User example in XML",
            "externalValue": "http://foo.bar/examples/user-example.xml"
          }
        }
    },
    "text/plain": {
      "examples": {
        "user" : {
            "summary": "User example in Plain text",
            "externalValue": "http://foo.bar/examples/user-example.txt"
        }
      }
    },
    "*/*": {
      "examples": {
        "user" : {
            "summary": "User example in other format",
            "externalValue": "http://foo.bar/examples/user-example.whatever"
        }
      }
    }
  }
}
```

```
description: user to add to the system
content:
  'application/json':
    schema:
      $ref: '#/components/schemas/User'
    examples:
      user:
        summary: User Example
        externalValue: 'http://foo.bar/examples/user-example.json'
  'application/xml':
    schema:
      $ref: '#/components/schemas/User'
    examples:
      user:
        summary: User Example in XML
        externalValue: 'http://foo.bar/examples/user-example.xml'
  'text/plain':
    examples:
      user:
        summary: User example in text plain format
        externalValue: 'http://foo.bar/examples/user-example.txt'
  '*/*':
    examples:
      user:
        summary: User example in other format
        externalValue: 'http://foo.bar/examples/user-example.whatever'
```

请求体是一个字符串的数组：

```
{
  "description": "user to add to the system",
  "content": {
    "text/plain": {
      "schema": {
        "type": "array",
        "items": {
          "type": "string"
        }
      }
    }
  }
}
```

```
description: user to add to the system
required: true
content:
  text/plain:
    schema:
      type: array
      items:
        type: string
```

Media Type 对象

每种媒体类型对象都有相应的纲要和示例来描述它。

固定字段

字段名	类型	描述	
schema	Schema 对象 \	Reference 对象	定义此媒体类型的纲要。
example	Any	媒体类型的示例。示例对象应该符合此媒体类型的格式， 这里指定的 `example` 对象 object is mutually exclusive of the `examples` object. 而且如果引用的 `schema` 也包含示例，在这里指定的 `example` 值将会覆盖 `schema` 提供的示例。	
examples	Map[`string` , Example 对象 \	Reference 对象]	媒体类型的示例，每个媒体对象的值都应该匹配它对应的媒体类型的格式。 The `examples` object is mutually exclusive of the `example` object. 而且如果引用的 `schema` 也包含示例，在这里指定的 `example` 值将会覆盖 `schema` 提供的示例。
encoding	Map[`string` , Encoding 对象]	属性名与编码信息的映射。每个属性名必须存在于 `schema` 属性的key中，当媒体类型等于 `multipart` 或 `application/x-www-form-urlencoded` 时，编码对象信息仅适用于 `requestBody` 。	

这个对象可能会被规范扩展扩展。

Media Type 示例

```
{
  "application/json": {
    "schema": {
         "$ref": "#/components/schemas/Pet"
    },
    "examples": {
      "cat" : {
        "summary": "An example of a cat",
        "value":
          {
            "name": "Fluffy",
            "petType": "Cat",
            "color": "White",
            "gender": "male",
            "breed": "Persian"
          }
      },
      "dog": {
        "summary": "An example of a dog with a cat's name",
        "value" :  {
          "name": "Puma",
          "petType": "Dog",
          "color": "Black",
          "gender": "Female",
          "breed": "Mixed"
        },
      "frog": {
          "$ref": "#/components/examples/frog-example"
        }
      }
    }
  }
}
```

```
application/json:
  schema:
    $ref: "#/components/schemas/Pet"
  examples:
    cat:
      summary: An example of a cat
      value:
        name: Fluffy
        petType: Cat
        color: White
        gender: male
        breed: Persian
    dog:
      summary: An example of a dog with a cat's name
      value:
        name: Puma
        petType: Dog
        color: Black
        gender: Female
        breed: Mixed
    frog:
      $ref: "#/components/examples/frog-example"
```

对文件上传的考虑

相对于2.0的规范，`file` 内容的上传与下载在开放API规范与其他类型一样使用相同的语法来描述。 特别的是:

```
# content transferred with base64 encoding
schema:
  type: string
  format: base64
```

```
# content transferred in binary (octet-stream):
schema:
  type: string
  format: binary
```

这些示例同时适用于文件上传和下载。

一个使用 `POST` 操作提交文件的 `requestBody` 看起来像下面这样：

```
requestBody:
  content:
    application/octet-stream:
      # any media type is accepted, functionally equivalent to `*/*`
      schema:
        # a binary file of any type
        type: string
        format: binary
```

此外，可以指定明确的媒体类型：

```
# multiple, specific media types may be specified:
requestBody:
  content:
      # a binary file of type png or jpeg
    'image/jpeg':
      schema:
        type: string
        format: binary
    'image/png':
      schema:
        type: string
        format: binary
```

为了同时上传多个文件，必须指定 `multipart` 媒体类型：

```
requestBody:
  content:
    multipart/form-data:
      schema:
        properties:
          # The property name 'file' will be used for all files.
          file:
            type: array
            items:
              type: string
              format: binary
```

x-www-form-urlencoded 请求体的支持

可以使用下面定义的格式来提交form url编码RFC1866的内容：

```
requestBody:
  content:
    application/x-www-form-urlencoded:
      schema:
        type: object
        properties:
          id:
            type: string
            format: uuid
          address:
            # complex types are stringified to support RFC 1866
            type: object
            properties: {}
```

在这个示例中，在内容被传送到服务器之前，`requestBody` 中的内容必须使用RFC1866中定义的方式字符串化。此外 `address` 字段的复杂对象将会被字符串化。

当使用 `application/x-www-form-urlencoded` 格式传送复杂对象时，默认的序列化策略在 `Encoding Object` 的 `style` 属性中定义为 `form` .

对 **`multipart`** 内容的特别思考

使用 `multipart/form-data` 作为 `Content-Type` 来传送请求体是很常见的做法。相对于2.0版本的规范，当定义 `multipart` 内容的输入参数时必须指定 `schema` 属性。这不但支持复杂的结构而且支持多文件上传机制。

当使用 `multipart` 类型是，可以使用boundaries来分隔传送的内容，因此 `multipart` 定义了以下默认的 `Content-Type` ：

- 如果属性是一个原始值或者是一个原始值的数组，那么默认的Content-Type是 `text/plain`
- 如果属性是复杂对象或者复杂对象的数组，那么默认的Content-Type是 `application/json`
- 如果属性是 `type: string` 与 `format: binary` 或 `format: base64` (也就是文件对象)的组合，那么默认的Content-Type是 `application/octet-stream`

示例:

```
requestBody:
  content:
    multipart/form-data:
      schema:
        type: object
        properties:
          id:
            type: string
            format: uuid
          address:
            # default Content-Type for objects is `application/json`
            type: object
            properties: {}
          profileImage:
            # default Content-Type for string/binary is `application/octet-stream`
            type: string
            format: binary
          children:
            # default Content-Type for arrays is based on the `inner` type (text/plain
 here)
            type: array
            items:
              type: string
          addresses:
            # default Content-Type for arrays is based on the `inner` type (object sho
wn, so `application/json` in this example)
            type: array
            items:
              type: '#/components/schemas/Address'
```

这里介绍一下用来控制序列化 `multipart` 请求体的 `encoding` 属性，这个属性只适用于 `multipart` 和 `application/x-www-form-urlencoded` 类型的请求体。

Encoding 对象

一个编码定义仅适用于一个纲要属性。

固定字段

字段名	类型	描述
		对具体属性的 Content-Type的编码。默认值取决于属性的类型：`application/octet-stream` 编码适用于 `binary` 格式的 `string` ；`text/plain` 适用于其他原始值；`application/json` 适

		用于 `object` ；对于 `array` 值类型的默认值取决于数组内元素的类型，默认值可以是明确的媒体类型(比如 `application/json`), 或者通配符类型的媒体类型(比如 `image/*`), 又或者是用分号分隔的两种媒体类型。	
headers	Map[`string` , Header 对象 \	Reference 对象]	提供附加信息的请求头键值对映射。比如 `Content-Disposition` 、 `Content-Type` 各自描述了不同的信息而且在这里将会被忽略，如果请求体的媒体类型不是 `multipart` ，这个属性将会被忽略。
style	`string`	描述一个属性根据它的类型将会被如何序列化。查看 Parameter 对象 的 `style` 属性可以得到更多详细信息。这个属性的行为与 `query` 参数相同，包括默认值的定义。如果请求体的媒体类型不是 `application/x-www-form-urlencoded` ，这个属性将会被忽略。	
explode	`boolean`	当这个值为true时，类型为 `array` 或 `object` 的属性值会为数组的每个元素或对象的每个键值对分开生成参数。这个属性对其他数据类型没有影响。当 `style` 为 `form` 时，这个属性的默认值是 `true` ，对于其他的 `style` 类型，这个属性的默认值是 `false` 。这个属性会被忽略如果请求体的媒体类型不是 `application/x-www-form-urlencoded` 。	
allowReserved	`boolean`	决定此参数的值是否允许不使用%号编码使用定义于RFC3986内的保留字符 `:/?#[]@!$&'()*+,;=` 。这个属性仅用于 `in` 的值是 `query` 时，此字段的默	

		是 `query` 时，此字段的默认值是 `false` 。 这个属性会被忽略如果请求体的媒体类型不是 `application/x-www-form-urlencoded` 。

这个对象可能会被规范扩展扩展。

Encoding 对象示例

```yaml
requestBody:
  content:
    multipart/mixed:
      schema:
        type: object
        properties:
          id:
            # default is text/plain
            type: string
            format: uuid
          address:
            # default is application/json
            type: object
            properties: {}
          historyMetadata:
            # need to declare XML format!
            description: metadata in XML format
            type: object
            properties: {}
          profileImage:
            # default is application/octet-stream, need to declare an image type only!
            type: string
            format: binary
      encoding:
        historyMetadata:
          # require XML Content-Type in utf-8 encoding
          contentType: application/xml; charset=utf-8
        profileImage:
          # only accept png/jpeg
          contentType: image/png, image/jpeg
          headers:
            X-Rate-Limit-Limit:
              description: The number of allowed requests in the current period
              schema:
                type: integer
```

Responses 对象

描述一个操作可能发生的响应的响应码与响应包含的响应体的对象。

一份API文档不必包含所有可能响应码，因为有些状态码无法提前预知。尽管如此，一份文档还是应当包含所有成功的响应和任何已知的错误响应。

`default` 字段可以用来标记一个响应适用于其他未被规范明确定义的HTTP响应码的默认响应。

一个 `Responses 对象` 必须至少包含一个响应码，而且是成功的响应。

固定字段

字段名	类型	描述	
default	Response 对象 | Reference 对象	用于描述未被明确声明的HTTP响应码的响应的文档。使用这个字段来覆盖未声明的响应。一个 Reference 对象 可以链接定义于 OpenAPI 对象 components/responses 区域的响应对象。	

模式字段

字段名模式	类型	描述	
HTTP Status Code	Response 对象 | Reference 对象	任何 HTTP status code 都可以被用作属性名，但是每一个状态码只能使用一次，用于描述此状态码的响应。一个 Reference 对象 可以链接定义于 OpenAPI 对象 components/responses 区域的响应对象。这个字段名必须包含在双引号中 (例如 "200") 以兼容 JSON 和 YAML。这个字段可以包含大写的通配字符 `X` 来定义响应码的范围。例如，`2XX` 代表所有位于 `[200-299]` 范围内的响应码。只允许使用以下范围定义：`1XX`，`2XX`，`3XX`，`4XX`，和 `5XX`。如果同时包含范围定义与明确定义的响应，那么明确定义的响应有更高的优先级。	

这个对象可能会被规范扩展扩展。

Responses 对象示例

一个代表成功操作的 200 响应和一个代表其他操作状态的默认响应（暗示是一个错误）：

```
{
  "200": {
    "description": "a pet to be returned",
    "content": {
      "application/json": {
        "schema": {
          "$ref": "#/components/schemas/Pet"
        }
      }
    }
  },
  "default": {
    "description": "Unexpected error",
    "content": {
      "application/json": {
        "schema": {
          "$ref": "#/components/schemas/ErrorModel"
        }
      }
    }
  }
}
```

```
'200':
  description: a pet to be returned
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/Pet'
default:
  description: Unexpected error
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/ErrorModel'
```

Response Object

Describes a single response from an API Operation, including design-time, static `links` to operations based on the response.

固定字段

字段名	类型	描述	
description	`string`	必选. A short description of the response. CommonMark syntax可以被用来呈现富文本格式.	
headers	Map[`string` , Header Object \	Reference Object]	Maps a header name to its definition. RFC7230 states header names are case insensitive. If a response header is defined with the name `"Content-Type"` , it SHALL be ignored.
content	Map[`string` , Media Type Object]	A map containing descriptions of potential response payloads. The key is a media type or media type range and the value describes it. For responses that match multiple keys, only the most specific key is applicable. e.g. text/plain overrides text/*	
links	Map[`string` , Link Object \	Reference Object]	A map of operations links that can be followed from the response. The key of the map is a short name for the link, following the naming constraints of the names for Component Objects.

这个对象可能会被规范扩展扩展。

Response 对象示例s

Response of an array of a complex type:

```
{
  "description": "A complex object array response",
  "content": {
    "application/json": {
      "schema": {
        "type": "array",
        "items": {
          "$ref": "#/components/schemas/VeryComplexType"
        }
      }
    }
  }
}
```

```
description: A complex object array response
content:
  application/json:
    schema:
      type: array
      items:
        $ref: '#/components/schemas/VeryComplexType'
```

Response with a string type:

```
{
  "description": "A simple string response",
  "content": {
    "text/plain": {
      "schema": {
        "type": "string"
      }
    }
  }

}
```

```
description: A simple string response
representations:
  text/plain:
    schema:
      type: string
```

Plain text response with headers:

```
{
  "description": "A simple string response",
  "content": {
    "text/plain": {
      "schema": {
        "type": "string"
      }
    }
  },
  "headers": {
    "X-Rate-Limit-Limit": {
      "description": "The number of allowed requests in the current period",
      "schema": {
        "type": "integer"
      }
    },
    "X-Rate-Limit-Remaining": {
      "description": "The number of remaining requests in the current period",
      "schema": {
        "type": "integer"
      }
    },
    "X-Rate-Limit-Reset": {
      "description": "The number of seconds left in the current period",
      "schema": {
        "type": "integer"
      }
    }
  }
}
```

```
description: A simple string response
content:
  text/plain:
    schema:
      type: string
    example: 'whoa!'
headers:
  X-Rate-Limit-Limit:
    description: The number of allowed requests in the current period
    schema:
      type: integer
  X-Rate-Limit-Remaining:
    description: The number of remaining requests in the current period
    schema:
      type: integer
  X-Rate-Limit-Reset:
    description: The number of seconds left in the current period
    schema:
      type: integer
```

Response with no return value:

```
{
  "description": "object created"
}
```

```
description: object created
```

Callback Object

A map of possible out-of band callbacks related to the parent operation. Each value in the map is a Path Item Object that describes a set of requests that may be initiated by the API provider and the expected responses. The key value used to identify the callback object is an expression, evaluated at runtime, that identifies a URL to use for the callback operation.

模式字段

字段名模式	类型	描述
{expression}	Path Item Object	A Path Item Object used to define a callback request and expected responses. A complete example is available.

这个对象可能会被规范扩展扩展。

Key Expression

The key that identifies the Path Item Object is a runtime expression that can be evaluated in the context of a runtime HTTP request/response to identify the URL to be used for the callback request. A simple example might be `$request.body#/url`. However, using a runtime expression the complete HTTP message can be accessed. This includes accessing any part of a body that a JSON Pointer RFC6901 can reference.

For example, given the following HTTP request:

```
POST /subscribe/myevent?queryUrl=http://clientdomain.com/stillrunning HTTP/1.1
Host: example.org
Content-Type: application/json
Content-Length: 187

{
  "failedUrl" : "http://clientdomain.com/failed",
  "successUrls" : [
    "http://clientdomain.com/fast",
    "http://clientdomain.com/medium",
    "http://clientdomain.com/slow"
  ]
}

201 Created
Location: http://example.org/subscription/1
```

The following examples show how the various expressions evaluate, assuming the callback operation has a path parameter named `eventType` and a query parameter named `queryUrl`.

Expression	Value
$url	http://example.org/subscribe/myevent?queryUrl=http://clientdomain.com/stillrunning
$method	POST
$request.path.eventType	myevent
$request.query.queryUrl	http://clientdomain.com/stillrunning
$request.header.content-Type	application/json
$request.body#/failedUrl	http://clientdomain.com/stillrunning
$request.body#/successUrls/2	http://clientdomain.com/medium
$response.header.Location	http://example.org/subscription/1

Callback 对象示例

The following example shows a callback to the URL specified by the `id` and `email` property in the request body.

```
myWebhook:
  'http://notificationServer.com?transactionId={$request.body#/id}&email={$request.bod
y#/email}':
    post:
      requestBody:
        description: Callback payload
        content:
          'application/json':
            schema:
              $ref: '#/components/schemas/SomePayload'
      responses:
        '200':
          description: webhook successfully processed and no retries will be performed
```

Example Object

固定字段

字段名	类型	描述
summary	`string`	Short description for the example.
description	`string`	Long description for the example. CommonMark syntax可以被用来呈现富文本格式.
value	Any	Embedded literal example. The `value` field and `externalValue` field are mutually exclusive. To represent examples of media types that cannot naturally represented in JSON or YAML, use a string value to contain the example, escaping where necessary.
externalValue	`string`	A URL that points to the literal example. This provides the capability to reference examples that cannot easily be included in JSON or YAML documents. The `value` field and `externalValue` field are mutually exclusive.

这个对象可能会被规范扩展扩展。

In all cases, the example value is expected to be compatible with the type schema of its associated value. Tooling implementations MAY choose to validate compatibility automatically, and reject the example value(s) if incompatible.

Example 对象示例

```
# in a model
schemas:
  properties:
    name:
      type: string
      examples:
```

```
        name:
          $ref: http://example.org/petapi-examples/openapi.json#/components/examples/name-example

# in a request body:
  requestBody:
    content:
      'application/json':
        schema:
          $ref: '#/components/schemas/Address'
        examples:
          foo:
            summary: A foo example
            value: {"foo": "bar"}
          bar:
            summary: A bar example
            value: {"bar": "baz"}
      'application/xml':
        examples:
          xmlExample:
            summary: This is an example in XML
            externalValue: 'http://example.org/examples/address-example.xml'
      'text/plain':
        examples:
          textExample:
            summary: This is a text example
            externalValue: 'http://foo.bar/examples/address-example.txt'


# in a parameter
  parameters:
    - name: 'zipCode'
      in: 'query'
      schema:
        type: 'string'
        format: 'zip-code'
        examples:
          zip-example:
            $ref: '#/components/examples/zip-example'

# in a response
  responses:
    '200':
      description: your car appointment has been booked
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/SuccessResponse'
          examples:
            confirmation-success:
              $ref: '#/components/examples/confirmation-success'
```

Link Object

The `Link object` represents a possible design-time link for a response. The presence of a link does not guarantee the caller's ability to successfully invoke it, rather it provides a known relationship and traversal mechanism between responses and other operations.

Unlike *dynamic* links (i.e. links provided **in** the response payload), the OAS linking mechanism does not require link information in the runtime response.

For computing links, and providing instructions to execute them, a runtime expression is used for accessing values in an operation and using them as parameters while invoking the linked operation.

固定字段

字段名	Type	描述	
operationRef	`string`	A relative or absolute reference to an OAS operation. This field is mutually exclusive of the `operationId` field, and MUST point to an Operation Object. Relative `operationRef` values MAY be used to locate an existing Operation Object in the OpenAPI definition.	
operationId	`string`	The name of an *existing*, resolvable OAS operation, as defined with a unique `operationId` . This field is mutually exclusive of the `operationRef` field.	
			A map representing parameters to pass to an operation as

parameters	Map[`string` , Any \	{expression}]	specified with `operationId` or identified via `operationRef` . The key is the parameter name to be used, whereas the value can be a constant or an expression to be evaluated and passed to the linked operation. The parameter name can be qualified using the parameter location `[{in}.]{name}` for operations that use the same parameter name in different locations (e.g. path.id).
requestBody	Any \	{expression}	A literal value or {expression} to use as a request body when calling the target operation.
description	`string`	A description of the link. CommonMark syntax可以被用来呈现富文本格式.	
server	Server Object	A server object to be used by the target operation.	

这个对象可能会被规范扩展扩展。

A linked operation MUST be identified using either an `operationRef` or `operationId` . In the case of an `operationId` , it MUST be unique and resolved in the scope of the OAS document. Because of the potential for name clashes, the `operationRef` syntax is preferred for specifications with external references.

Examples

Computing a link from a request operation where the `$request.path.id` is used to pass a request parameter to the linked operation.

```yaml
paths:
  /users/{id}:
    parameters:
    - name: id
      in: path
      required: true
      description: the user identifier, as userId
      schema:
        type: string
    get:
      responses:
        '200':
          description: the user being returned
          content:
            application/json:
              schema:
                type: object
                properties:
                  uuid: # the unique user id
                    type: string
                    format: uuid
        links:
          address:
            # the target link operationId
            operationId: getUserAddress
            parameters:
              # get the `id` field from the request path parameter named `id`
              userId: $request.path.id
  # the path item of the linked operation
  /users/{userid}/address:
    parameters:
    - name: userid
      in: path
      required: true
      description: the user identifier, as userId
      schema:
        type: string
      # linked operation
      get:
        operationId: getUserAddress
        responses:
          '200':
            description: the user's address
```

When a runtime expression fails to evaluate, no parameter value is passed to the target operation.

Values from the response body can be used to drive a linked operation.

```
links:
  address:
    operationId: getUserAddressByUUID
    parameters:
      # get the `id` field from the request path parameter named `id`
      userUuid: $response.body#/uuid
```

Clients follow all links at their discretion. Neither permissions, nor the capability to make a successful call to that link, is guaranteed solely by the existence of a relationship.

OperationRef Examples

As references to `operationId` MAY NOT be possible (the `operationId` is an optional value), references MAY also be made through a relative `operationRef` :

```
links:
  UserRepositories:
    # returns array of '#/components/schemas/repository'
    operationRef: '#/paths/~12.0~1repositories~1{username}/get'
    parameters:
      username: $response.body#/username
```

or an absolute `operationRef` :

```
links:
  UserRepositories:
    # returns array of '#/components/schemas/repository'
    operationRef: 'https://na2.gigantic-server.com/#/paths/~12.0~1repositories~1{usern
ame}/get'
    parameters:
      username: $response.body#/username
```

Note that in the use of `operationRef` , the *escaped forward-slash* is necessary when using JSON references.

Runtime Expressions

Runtime expressions allow defining values based on information that will only be available within the HTTP message in an actual API call. This mechanism is used by Link Objects and Callback Objects.

The runtime expression is defined by the following ABNF syntax

```
      expression = ( "$url" | "$method" | "$statusCode" | "$request." source | "$respo
nse." source )
      source = ( header-reference | query-reference | path-reference | body-reference
)
      header-reference = "header." token
      query-reference = "query." name
      path-reference = "path." name
      body-reference = "body" ["#" fragment]
      fragment = a JSON Pointer [RFC 6901](https://tools.ietf.org/html/rfc6901)
      name = *( char )
      char = as per RFC [7159](https://tools.ietf.org/html/rfc7159#section-7)
      token = as per RFC [7230](https://tools.ietf.org/html/rfc7230#section-3.2.6)
```

The `name` identifier is case-sensitive, whereas `token` is not.

The table below provides examples of runtime expressions and examples of their use in a value:

Examples

Source Location	example expression	notes
HTTP Method	`$method`	The allowable values for the `$method` will be those for the HTTP operation.
Requested media type	`$request.header.accept`	
Request parameter	`$request.path.id`	Request parameters MUST be declared in the `parameters` section of the parent operation or they cannot be evaluated. This includes request headers.
Request body property	`$request.body#/user/uuid`	In operations which accept payloads, references may be made to portions of the `requestBody` or the entire body.
Request URL	`$url`	
Response value	`$response.body#/status`	In operations which return payloads, references may be made to portions of the response body or the entire body.
Response header	`$response.header.Server`	Single header values only are available

Runtime expressions preserve the type of the referenced value. Expressions can be embedded into string values by surrounding the expression with `{}` curly braces.

Header Object

The Header Object follows the structure of the Parameter Object with the following changes:

1. `name` MUST NOT be specified, it is given in the corresponding `headers` map.
2. `in` MUST NOT be specified, it is implicitly in `header`.
3. All traits that are affected by the location MUST be applicable to a location of `header` (for example, `style`).

Header 对象示例

A simple header of type `integer`:

```
{
  "description": "The number of allowed requests in the current period",
  "schema": {
    "type": "integer"
  }
}
```

```
description: The number of allowed requests in the current period
schema:
  type: integer
```

Tag Object

Adds metadata to a single tag that is used by the Operation Object. It is not mandatory to have a Tag Object per tag defined in the Operation Object instances.

固定字段

字段名	类型	描述
name	`string`	必选. The name of the tag.
description	`string`	A short description for the tag. CommonMark syntax可以被用来呈现富文本格式.
externalDocs	External Documentation Object	Additional external documentation for this tag.

这个对象可能会被规范扩展扩展。

Tag 对象示例

```
{
    "name": "pet",
    "description": "Pets operations"
}
```

```
name: pet
description: Pets operations
```

Examples Object

In an `example`, a JSON Reference MAY be used, with the explicit restriction that examples having a JSON format with object named `$ref` are not allowed. Therefore, that `example`, structurally, can be either a string primitive or an object, similar to `additionalProperties`.

In all cases, the payload is expected to be compatible with the type schema for the associated value. Tooling implementations MAY choose to validate compatibility automatically, and reject the example value(s) if they are incompatible.

```yaml
# in a model
schemas:
  properties:
    name:
      type: string
      example:
        $ref: http://foo.bar#/examples/name-example

# in a request body, note the plural `examples`
  requestBody:
    content:
      'application/json':
        schema:
          $ref: '#/components/schemas/Address'
        examples:
          foo:
            value: {"foo": "bar"}
          bar:
            value: {"bar": "baz"}
      'application/xml':
        examples:
          xml:
            externalValue: 'http://foo.bar/examples/address-example.xml'
      'text/plain':
        examples:
          text:
            externalValue: 'http://foo.bar/examples/address-example.txt'

# in a parameter
  parameters:
    - name: 'zipCode'
      in: 'query'
      schema:
        type: 'string'
        format: 'zip-code'
        example:
          $ref: 'http://foo.bar#/examples/zip-example'

# in a response, note the singular `example`:
  responses:
    '200':
      description: your car appointment has been booked
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/SuccessResponse'
          example:
            $ref: http://foo.bar#/examples/address-example.json
```

Reference Object

A simple object to allow referencing other components in the specification, internally and externally.

The Reference Object is defined by JSON Reference and follows the same structure, behavior and rules.

For this specification, reference resolution is accomplished as defined by the JSON Reference specification and not by the JSON Schema specification.

固定字段

字段名	类型	描述
$ref	`string`	必选. The reference string.

This object cannot be extended with additional properties and any properties added SHALL be ignored.

Reference 对象示例

```
{
    "$ref": "#/components/schemas/Pet"
}
```

```
$ref: '#/components/schemas/Pet'
```

Relative Schema Document Example

```
{
  "$ref": "Pet.json"
}
```

```
$ref: Pet.yaml
```

Relative Documents With Embedded Schema Example

```
{
  "$ref": "definitions.json#/Pet"
}
```

```
$ref: definitions.yaml#/Pet
```

Schema Object

The Schema Object allows the definition of input and output data types. These types can be objects, but also primitives and arrays. This object is an extended subset of the JSON Schema Specification Wright Draft 00.

For more information about the properties, see JSON Schema Core and JSON Schema Validation. Unless stated otherwise, the property definitions follow the JSON Schema.

Properties

The following properties are taken directly from the JSON Schema definition and follow the same specifications:

- title
- multipleOf
- maximum
- exclusiveMaximum
- minimum
- exclusiveMinimum
- maxLength
- minLength
- pattern (This string SHOULD be a valid regular expression, according to the ECMA 262 regular expression dialect)
- maxItems
- minItems
- uniqueItems
- maxProperties
- minProperties
- required
- enum

The following properties are taken from the JSON Schema definition but their definitions were adjusted to the OpenAPI Specification.

- type - Value MUST be a string. Multiple types via an array are not supported.
- allOf - Inline or referenced schema MUST be of a Schema Object and not a standard JSON Schema.
- oneOf - Inline or referenced schema MUST be of a Schema Object and not a standard JSON Schema.
- anyOf - Inline or referenced schema MUST be of a Schema Object and not a standard JSON Schema.
- not - Inline or referenced schema MUST be of a Schema Object and not a standard

JSON Schema.

- items - Value MUST be an object and not an array. Inline or referenced schema MUST be of a Schema Object and not a standard JSON Schema. `items` MUST be present if the `type` is `array`.
- properties - Property definitions MUST be a Schema Object and not a standard JSON Schema (inline or referenced).
- additionalProperties - Value can be boolean or object. Inline or referenced schema MUST be of a Schema Object and not a standard JSON Schema.
- description - CommonMark syntax可以被用来呈现富文本格式.
- format - See Data Type Formats for further details. While relying on JSON Schema's defined formats, the OAS offers a few additional predefined formats.
- default - The default value represents what would be assumed by the consumer of the input as the value of the schema if one is not provided. Unlike JSON Schema, the value MUST conform to the defined type for the Schema Object defined at the same level. For example, if `type` is `string`, then `default` can be `"foo"` but cannot be `1`.

Alternatively, any time a Schema Object can be used, a Reference Object can be used in its place. This allows referencing definitions instead of defining them inline.

Additional properties defined by the JSON Schema specification that are not mentioned here are strictly unsupported.

Other than the JSON Schema subset fields, the following fields MAY be used for further schema documentation:

固定字段

字段名	类型	描述
nullable	`boolean`	Allows sending a `null` value for the defined schema. Default value is `false`.
discriminator	Discriminator Object	Adds support for polymorphism. The discriminator is an object name that is used to differentiate between other schemas which may satisfy the payload description. See Composition and Inheritance for more details.
readOnly	`boolean`	Relevant only for Schema `"properties"` definitions. Declares the property as "read only". This means that it MAY be sent as part of a response but SHOULD NOT be sent as part of the request. If the property is marked as `readOnly` being `true` and is in the `required` list, the `required` will take effect on the response only. A property MUST NOT be marked as both `readOnly` and `writeOnly` being `true`. Default value is `false`.
writeOnly	`boolean`	Relevant only for Schema `"properties"` definitions. Declares the property as "write only". Therefore, it MAY be sent as part of a request but SHOULD NOT be sent as part of the response. If the property is marked as `writeOnly` being `true` and is in the `required` list, the `required` will take effect on the request only. A property MUST NOT be marked as both `readOnly` and `writeOnly` being `true`. Default value is `false`.
xml	XML Object	This MAY be used only on properties schemas. It has no effect on root schemas. Adds additional metadata to describe the XML representation of this property.
externalDocs	External Documentation Object	Additional external documentation for this schema.
example	Any	A free-form property to include an example of an instance for this schema. To represent examples that cannot be naturally represented in JSON or YAML, a string value can be used to contain the example with escaping where necessary.
deprecated	`boolean`	Specifies that a schema is deprecated and SHOULD be transitioned out of usage. Default value is `false`.

这个对象可能会被规范扩展扩展。

Composition and Inheritance (Polymorphism)

The OpenAPI Specification allows combining and extending model definitions using the `allOf` property of JSON Schema, in effect offering model composition. `allOf` takes an array of object definitions that are validated *independently* but together compose a single object.

While composition offers model extensibility, it does not imply a hierarchy between the models. To support polymorphism, the OpenAPI Specification adds the `discriminator` field. When used, the `discriminator` will be the name of the property that decides which schema definition validates the structure of the model. As such, the `discriminator` field MUST be a required field. There are are two ways to define the value of a discriminator for an inheriting instance.

- Use the schema name.
- Override the schema name by overriding the property with a new value. If a new value exists, this takes precedence over the schema name. As such, inline schema definitions, which do not have a given id, *cannot* be used in polymorphism.

XML Modeling

The xml property allows extra definitions when translating the JSON definition to XML. The XML Object contains additional information about the available options.

Schema 对象示例**s**

Primitive Sample

```
{
  "type": "string",
  "format": "email"
}
```

```
type: string
format: email
```

Simple Model

```
{
  "type": "object",
  "required": [
    "name"
  ],
  "properties": {
    "name": {
      "type": "string"
    },
    "address": {
      "$ref": "#/components/schemas/Address"
    },
    "age": {
      "type": "integer",
      "format": "int32",
      "minimum": 0
    }
  }
}
```

```
type: object
required:
- name
properties:
  name:
    type: string
  address:
    $ref: '#/components/schemas/Address'
  age:
    type: integer
    format: int32
    minimum: 0
```

Model with Map/Dictionary Properties

For a simple string to string mapping:

```
{
  "type": "object",
  "additionalProperties": {
    "type": "string"
  }
}
```

```
type: object
additionalProperties:
  type: string
```

For a string to model mapping:

```
{
  "type": "object",
  "additionalProperties": {
    "$ref": "#/components/schemas/ComplexModel"
  }
}
```

```
type: object
additionalProperties:
  $ref: '#/components/schemas/ComplexModel'
```

Model with Example

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "name": {
      "type": "string"
    }
  },
  "required": [
    "name"
  ],
  "example": {
    "name": "Puma",
    "id": 1
  }
}
```

```
type: object
properties:
  id:
    type: integer
    format: int64
  name:
    type: string
required:
- name
example:
  name: Puma
  id: 1
```

Models with Composition

```
{
  "components": {
    "schemas": {
      "ErrorModel": {
        "type": "object",
        "required": [
          "message",
          "code"
        ],
        "properties": {
          "message": {
            "type": "string"
          },
          "code": {
            "type": "integer",
            "minimum": 100,
            "maximum": 600
          }
        }
      },
      "ExtendedErrorModel": {
        "allOf": [
          {
            "$ref": "#/components/schemas/ErrorModel"
          },
          {
            "type": "object",
            "required": [
              "rootCause"
            ],
            "properties": {
              "rootCause": {
                "type": "string"
              }
            }
          }
        ]
      }
    }
  }
}
```

```yaml
components:
  schemas:
    ErrorModel:
      type: object
      required:
      - message
      - code
      properties:
        message:
          type: string
        code:
          type: integer
          minimum: 100
          maximum: 600
    ExtendedErrorModel:
      allOf:
      - $ref: '#/components/schemas/ErrorModel'
      - type: object
        required:
        - rootCause
        properties:
          rootCause:
            type: string
```

Models with Polymorphism Support

```json
{
  "components": {
    "schemas": {
      "Pet": {
        "type": "object",
        "discriminator": {
          "propertyName": "petType"
        },
        "properties": {
          "name": {
            "type": "string"
          },
          "petType": {
            "type": "string"
          }
        },
        "required": [
          "name",
          "petType"
        ]
      },
      "Cat": {
        "description": "A representation of a cat. Note that `Cat` will be used as the
 discriminator value.",
        "allOf": [
```

```
          {
            "$ref": "#/components/schemas/Pet"
          },
          {
            "type": "object",
            "properties": {
              "huntingSkill": {
                "type": "string",
                "description": "The measured skill for hunting",
                "default": "lazy",
                "enum": [
                  "clueless",
                  "lazy",
                  "adventurous",
                  "aggressive"
                ]
              }
            },
            "required": [
              "huntingSkill"
            ]
          }
        ]
      },
      "Dog": {
        "description": "A representation of a dog. Note that `Dog` will be used as the
discriminator value.",
        "allOf": [
          {
            "$ref": "#/components/schemas/Pet"
          },
          {
            "type": "object",
            "properties": {
              "packSize": {
                "type": "integer",
                "format": "int32",
                "description": "the size of the pack the dog is from",
                "default": 0,
                "minimum": 0
              }
            },
            "required": [
              "packSize"
            ]
          }
        ]
      }
    }
  }
}
```

```yaml
components:
  schemas:
    Pet:
      type: object
      discriminator:
        propertyName: petType
      properties:
        name:
          type: string
        petType:
          type: string
      required:
      - name
      - petType
    Cat:  ## "Cat" will be used as the discriminator value
      description: A representation of a cat
      allOf:
      - $ref: '#/components/schemas/Pet'
      - type: object
        properties:
          huntingSkill:
            type: string
            description: The measured skill for hunting
            enum:
            - clueless
            - lazy
            - adventurous
            - aggressive
        required:
        - huntingSkill
    Dog:  ## "Dog" will be used as the discriminator value
      description: A representation of a dog
      allOf:
      - $ref: '#/components/schemas/Pet'
      - type: object
        properties:
          packSize:
            type: integer
            format: int32
            description: the size of the pack the dog is from
            default: 0
            minimum: 0
        required:
        - packSize
```

Discriminator Object

When request bodies or response payloads may be one of a number of different schemas, a `discriminator` object can be used to aid in serialization, deserialization, and validation. The discriminator is a specific object in a schema which is used to inform the consumer of the

specification of an alternative schema based on the value associated with it.

When using the discriminator, *inline* schemas will not be considered.

固定字段

字段名	类型	描述
propertyName	`string`	必选. The name of the property in the payload that will hold the discriminator value.
mapping	Map[`string` , `string`]	An object to hold mappings between payload values and schema names or references.

The discriminator attribute is legal only when using one of the composite keywords `oneOf` , `anyOf` , `allOf` .

In OAS 3.0, a response payload MAY be described to be exactly one of any number of types:

```
MyResponseType:
  oneOf:
  - $ref: '#/components/schemas/Cat'
  - $ref: '#/components/schemas/Dog'
  - $ref: '#/components/schemas/Lizard'
```

which means the payload *MUST*, by validation, match exactly one of the schemas described by `Cat` , `Dog` , or `Lizard` . In this case, a discriminator MAY act as a "hint" to shortcut validation and selection of the matching schema which may be a costly operation, depending on the complexity of the schema. We can then describe exactly which field tells us which schema to use:

```
MyResponseType:
  oneOf:
  - $ref: '#/components/schemas/Cat'
  - $ref: '#/components/schemas/Dog'
  - $ref: '#/components/schemas/Lizard'
  discriminator:
    propertyName: pet_type
```

The expectation now is that a property with name `pet_type` *MUST* be present in the response payload, and the value will correspond to the name of a schema defined in the OAS document. Thus the response payload:

```
{
  "id": 12345,
  "pet_type": "Cat"
}
```

Will indicate that the `Cat` schema be used in conjunction with this payload.

In scenarios where the value of the discriminator field does not match the schema name or implicit mapping is not possible, an optional `mapping` definition MAY be used:

```
MyResponseType:
  oneOf:
  - $ref: '#/components/schemas/Cat'
  - $ref: '#/components/schemas/Dog'
  - $ref: '#/components/schemas/Lizard'
  - $ref: 'https://gigantic-server.com/schemas/Monster/schema.json'
  discriminator:
    propertyName: pet_type
    mapping:
      dog: '#/components/schemas/Dog'
      monster: 'https://gigantic-server.com/schemas/Monster/schema.json'
```

Here the discriminator *value* of `dog` will map to the schema `#/components/schemas/Dog`, rather than the default (implicit) value of `Dog`. If the discriminator *value* does not match an implicit or explicit mapping, no schema can be determined and validation SHOULD fail. Mapping keys MUST be string values, but tooling MAY convert response values to strings for comparison.

When used in conjunction with the `anyOf` construct, the use of the discriminator can avoid ambiguity where multiple schemas may satisfy a single payload.

In both the `oneOf` and `anyOf` use cases, all possible schemas MUST be listed explicitly. To avoid redundancy, the discriminator MAY be added to a parent schema definition, and all schemas comprising the parent schema in an `allOf` construct may be used as an alternate schema.

For example:

```
components:
  schemas:
    Pet:
      type: object
      required:
      - pet_type
      properties:
        pet_type:
          type: string
      discriminator:
        propertyName: pet_type
        mapping:
          cachorro: Dog
    Cat:
      allOf:
      - $ref: '#/components/schemas/Pet'
      - type: object
        # all other properties specific to a `Cat`
        properties:
          name:
            type: string
    Dog:
      allOf:
      - $ref: '#/components/schemas/Pet'
      - type: object
        # all other properties specific to a `Dog`
        properties:
          bark:
            type: string
    Lizard:
      allOf:
      - $ref: '#/components/schemas/Pet'
      - type: object
        # all other properties specific to a `Lizard`
        properties:
          lovesRocks:
            type: boolean
```

a payload like this:

```
{
  "pet_type": "Cat",
  "name": "misty"
}
```

will indicate that the `Cat` schema be used. Likewise this schema:

```
{
  "pet_type": "cachorro",
  "bark": "soft"
}
```

will map to `Dog` because of the definition in the `mappings` element.

XML Object

A metadata object that allows for more fine-tuned XML model definitions.

When using arrays, XML element names are *not* inferred (for singular/plural forms) and the `name` property SHOULD be used to add that information. See examples for expected behavior.

固定字段

字段名	类型	描述
name	`string`	Replaces the name of the element/attribute used for the described schema property. When defined within `items`, it will affect the name of the individual XML elements within the list. When defined alongside `type` being `array` (outside the `items`), it will affect the wrapping element and only if `wrapped` is `true`. If `wrapped` is `false`, it will be ignored.
namespace	`string`	The URI of the namespace definition. Value MUST be in the form of an absolute URI.
prefix	`string`	The prefix to be used for the name.
attribute	`boolean`	Declares whether the property definition translates to an attribute instead of an element. Default value is `false`.
wrapped	`boolean`	MAY be used only for an array definition. Signifies whether the array is wrapped (for example, `<books><book/><book/></books>`) or unwrapped (`<book/><book/>`). Default value is `false`. The definition takes effect only when defined alongside `type` being `array` (outside the `items`).

这个对象可能会被规范扩展扩展。

XML 对象示例**s**

The examples of the XML object definitions are included inside a property definition of a Schema Object with a sample of the XML representation of it.

No XML Element

Basic string property:

```
{
    "animals": {
        "type": "string"
    }
}
```

```
animals:
  type: string
```

```
<animals>...</animals>
```

Basic string array property (`wrapped` is `false` by default):

```
{
    "animals": {
        "type": "array",
        "items": {
            "type": "string"
        }
    }
}
```

```
animals:
  type: array
  items:
    type: string
```

```
<animals>...</animals>
<animals>...</animals>
<animals>...</animals>
```

XML Name Replacement

```
{
  "animals": {
    "type": "string",
    "xml": {
      "name": "animal"
    }
  }
}
```

```yaml
animals:
  type: string
  xml:
    name: animal
```

```xml
<animal>...</animal>
```

XML Attribute, Prefix and Namespace

In this example, a full model definition is shown.

```json
{
  "Person": {
    "type": "object",
    "properties": {
      "id": {
        "type": "integer",
        "format": "int32",
        "xml": {
          "attribute": true
        }
      },
      "name": {
        "type": "string",
        "xml": {
          "namespace": "http://example.com/schema/sample",
          "prefix": "sample"
        }
      }
    }
  }
}
```

```yaml
Person:
  type: object
  properties:
    id:
      type: integer
      format: int32
      xml:
        attribute: true
    name:
      type: string
      xml:
        namespace: http://example.com/schema/sample
        prefix: sample
```

```xml
<Person id="123">
    <sample:name xmlns:sample="http://example.com/schema/sample">example</sample:name>
</Person>
```

XML Arrays

Changing the element names:

```json
{
  "animals": {
    "type": "array",
    "items": {
      "type": "string",
      "xml": {
        "name": "animal"
      }
    }
  }
}
```

```yaml
animals:
  type: array
  items:
    type: string
    xml:
      name: animal
```

```xml
<animal>value</animal>
<animal>value</animal>
```

The external `name` property has no effect on the XML:

```json
{
  "animals": {
    "type": "array",
    "items": {
      "type": "string",
      "xml": {
        "name": "animal"
      }
    },
    "xml": {
      "name": "aliens"
    }
  }
}
```

```yaml
animals:
  type: array
  items:
    type: string
    xml:
      name: animal
  xml:
    name: aliens
```

```xml
<animal>value</animal>
<animal>value</animal>
```

Even when the array is wrapped, if a name is not explicitly defined, the same name will be used both internally and externally:

```json
{
  "animals": {
    "type": "array",
    "items": {
      "type": "string"
    },
    "xml": {
      "wrapped": true
    }
  }
}
```

```yaml
animals:
  type: array
  items:
    type: string
  xml:
    wrapped: true
```

```xml
<animals>
  <animals>value</animals>
  <animals>value</animals>
</animals>
```

To overcome the naming problem in the example above, the following definition can be used:

```
{
  "animals": {
    "type": "array",
    "items": {
      "type": "string",
      "xml": {
        "name": "animal"
      }
    },
    "xml": {
      "wrapped": true
    }
  }
}
```

```
animals:
  type: array
  items:
    type: string
    xml:
      name: animal
  xml:
    wrapped: true
```

```
<animals>
  <animal>value</animal>
  <animal>value</animal>
</animals>
```

Affecting both internal and external names:

```
{
  "animals": {
    "type": "array",
    "items": {
      "type": "string",
      "xml": {
        "name": "animal"
      }
    },
    "xml": {
      "name": "aliens",
      "wrapped": true
    }
  }
}
```

```yaml
animals:
  type: array
  items:
    type: string
    xml:
      name: animal
  xml:
    name: aliens
    wrapped: true
```

```xml
<aliens>
  <animal>value</animal>
  <animal>value</animal>
</aliens>
```

If we change the external element but not the internal ones:

```json
{
  "animals": {
    "type": "array",
    "items": {
      "type": "string"
    },
    "xml": {
      "name": "aliens",
      "wrapped": true
    }
  }
}
```

```yaml
animals:
  type: array
  items:
    type: string
  xml:
    name: aliens
    wrapped: true
```

```xml
<aliens>
  <aliens>value</aliens>
  <aliens>value</aliens>
</aliens>
```

Security Scheme Object

Defines a security scheme that can be used by the operations. Supported schemes are HTTP authentication, an API key (either as a header or as a query parameter), OAuth2's common flows (implicit, password, application and access code) as defined in RFC6749, and OpenID Connect Discovery.

固定字段

字段名	类型	**Applies To**	描述
type	`string`	Any	必选. The type of the security scheme. Valid values are `"apiKey"`, `"http"`, `"oauth2"`, `"openIdConnect"`.
description	`string`	Any	A short description for security scheme. CommonMark syntax可以被用来呈现富文本格式.
name	`string`	`apiKey`	必选. The name of the header, query or cookie parameter to be used.
in	`string`	`apiKey`	必选. The location of the API key. Valid values are `"query"`, `"header"` or `"cookie"`.
scheme	`string`	`http`	必选. The name of the HTTP Authorization scheme to be used in the Authorization header as defined in RFC7235.
bearerFormat	`string`	`http` (`"bearer"`)	A hint to the client to identify how the bearer token is formatted. Bearer tokens are usually generated by an authorization server, so this information is primarily for documentation purposes.
flows	OAuth Flows Object	`oauth2`	必选. An object containing configuration information for the flow types supported.
openIdConnectUrl	`string`	`openIdConnect`	必选. OpenId Connect URL to discover OAuth2 configuration values. This MUST be in the form of a URL.

这个对象可能会被规范扩展扩展。

Security Scheme 对象示例

Basic Authentication Sample

```
{
  "type": "http",
  "scheme": "basic"
}
```

```
type: http
scheme: basic
```

API Key Sample

```
{
  "type": "apiKey",
  "name": "api_key",
  "in": "header"
}
```

```
type: apiKey
name: api_key
in: header
```

JWT Bearer Sample

```
{
  "type": "http",
  "scheme": "bearer",
  "bearerFormat": "JWT",
}
```

```
type: http
scheme: bearer
bearerFormat: JWT
```

Implicit OAuth2 Sample

```
{
  "type": "oauth2",
  "flows": {
    "implicit": {
      "authorizationUrl": "https://example.com/api/oauth/dialog",
      "scopes": {
        "write:pets": "modify pets in your account",
        "read:pets": "read your pets"
      }
    }
  }
}
```

```
type: oauth2
flows:
  implicit:
    authorizationUrl: https://example.com/api/oauth/dialog
    scopes:
      write:pets: modify pets in your account
      read:pets: read your pets
```

OAuth Flows Object

Allows configuration of the supported OAuth Flows.

固定字段

字段名	类型	描述
implicit	OAuth Flow Object	Configuration for the OAuth Implicit flow
password	OAuth Flow Object	Configuration for the OAuth Resource Owner Password flow
clientCredentials	OAuth Flow Object	Configuration for the OAuth Client Credentials flow. Previously called `application` in OpenAPI 2.0.
authorizationCode	OAuth Flow Object	Configuration for the OAuth Authorization Code flow. Previously called `accessCode` in OpenAPI 2.0.

这个对象可能会被规范扩展扩展。

OAuth Flow Object

Configuration details for a supported OAuth Flow

固定字段

字段名	类型	**Applies To**	描述
authorizationUrl	`string`	`oauth2` (`"implicit"` , `"authorizationCode"`)	必选. The authorization URL to be used for this flow. This MUST be in the form of a URL.
tokenUrl	`string`	`oauth2` (`"password"` , `"clientCredentials"` , `"authorizationCode"`)	必选. The token URL to be used for this flow. This MUST be in the form of a URL.
refreshUrl	`string`	`oauth2`	The URL to be used for obtaining refresh tokens. This MUST be in the form of a URL.
scopes	Map[`string` , `string`]	`oauth2`	必选. The available scopes for the OAuth2 security scheme. A map between the scope name and a short description for it.

这个对象可能会被规范扩展扩展。

OAuth Flow 对象示例**s**

```
{
  "type": "oauth2",
  "flows": {
    "implicit": {
      "authorizationUrl": "https://example.com/api/oauth/dialog",
      "scopes": {
        "write:pets": "modify pets in your account",
        "read:pets": "read your pets"
      }
    },
    "authorizationCode": {
      "authorizationUrl": "https://example.com/api/oauth/dialog",
      "tokenUrl": "https://example.com/api/oauth/token",
      "scopes": {
        "write:pets": "modify pets in your account",
        "read:pets": "read your pets"
      }
    }
  }
}
```

```
type: oauth2
flows:
  implicit:
    authorizationUrl: https://example.com/api/oauth/dialog
    scopes:
      write:pets: modify pets in your account
      read:pets: read your pets
  authorizationCode:
    authorizationUrl: https://example.com/api/oauth/dialog
    tokenUrl: https://example.com/api/oauth/token
    scopes:
      write:pets: modify pets in your account
      read:pets: read your pets
```

Security Requirement Object

Lists the required security schemes to execute this operation. The name used for each property MUST correspond to a security scheme declared in the Security Schemes under the Components Object.

Security Requirement Objects that contain multiple schemes require that all schemes MUST be satisfied for a request to be authorized. This enables support for scenarios where multiple query parameters or HTTP headers are required to convey security information.

When a list of Security Requirement Objects is defined on the Open API object or Operation Object, only one of Security Requirement Objects in the list needs to be satisfied to authorize the request.

模式字段

字段名 模式	类型	描述
{name}	[`string`]	Each name MUST correspond to a security scheme which is declared in the Security Schemes under the Components Object. If the security scheme is of type `"oauth2"` or `"openIdConnect"`, then the value is a list of scope names required for the execution. For other security scheme types, the array MUST be empty.

Security Requirement 对象示例

Non-OAuth2 Security Requirement

```
{
  "api_key": []
}
```

```
api_key: []
```

OAuth2 Security Requirement

```
{
  "petstore_auth": [
    "write:pets",
    "read:pets"
  ]
}
```

```
petstore_auth:
- write:pets
- read:pets
```

规范扩展

While the OpenAPI Specification tries to accommodate most use cases, additional data can be added to extend the specification at certain points.

The extensions properties are implemented as patterned fields that are always prefixed by `"x-"`.

字段名模式	类型	描述
^x-	Any	Allows extensions to the OpenAPI Schema. The field name MUST begin with `x-`, for example, `x-internal-id`. The value can be `null`, a primitive, an array or an object. Can have any valid JSON format value.

The extensions may or may not be supported by the available tooling, but those may be extended as well to add requested support (if tools are internal or open-sourced).

Security Filtering

Some objects in the OpenAPI Specification MAY be declared and remain empty, or be completely removed, even though they are inherently the core of the API documentation.

The reasoning is to allow an additional layer of access control over the documentation. While not part of the specification itself, certain libraries MAY choose to allow access to parts of the documentation based on some form of authentication/authorization.

Two examples of this:

1. The Paths Object MAY be empty. It may be counterintuitive, but this may tell the viewer that they got to the right place, but can't access any documentation. They'd still have access to the Info Object which may contain additional information regarding authentication.
2. The Path Item Object MAY be empty. In this case, the viewer will be aware that the path exists, but will not be able to see any of its operations or parameters. This is different than hiding the path itself from the Paths Object, so the user will not be aware of its existence. This allows the documentation provider to finely control what the viewer can see.

Appendix A: Revision History

Version	Date	Notes
3.0.0	2017-07-26	Release of the OpenAPI Specification 3.0.0
3.0.0-rc2	2017-06-16	rc2 of the 3.0 specification
3.0.0-rc1	2017-04-27	rc1 of the 3.0 specification
3.0.0-rc0	2017-02-28	Implementer's Draft of the 3.0 specification
2.0	2015-12-31	Donation of Swagger 2.0 to the Open API Initiative
2.0	2014-09-08	Release of Swagger 2.0
1.2	2014-03-14	Initial release of the formal document.
1.1	2012-08-22	Release of Swagger 1.1
1.0	2011-08-10	First release of the Swagger Specification