Introduction

如何编写基于OpenAPI规范的API文档

Introduction
如何编写基于OpenAPI规范的API文档
- 前言
  - 编写目的
  - 涉及范围
- 第1章 简介
  - 1.1 Swagger
  - 1.2 OpenAPI规范
  - 1.3 为啥要使用OpenAPI规范？
  - 1.4 如何编写API文档？
    - 1.4.1 语言：JSON vs YAML
    - 1.4.2 编辑器
- 第2章 从零开始
  - 2.1 最简单的例子
    - 2.1.1 OpenAPI规范的版本号
    - 2.1.2 API描述信息
    - 2.1.3 API的URL
    - 2.1.4 API的操作（operation）
  - 2.2 定义一个API操作
    - 2.2.1 添加一个`路径`（path）
    - 2.2.2 在路径中添加一个HTTP方法
    - 2.2.3 定义`响应`（response）类型
    - 2.2.4 定义响应内容
  - 2.3 定义`请求参数`（query parameters）
    - 2.3.1 在*get*方法中增加请求参数
    - 2.3.2 添加分页参数
  - 2.4 定义`路径参数`（path parameter）
    - 2.4.1 添加一个 *get /persons/{username}* 操作
    - 2.4.2 定义路径参数 *username*
    - 2.4.3 定义响应消息
  - 2.5 定义`消息体参数`（body parameter）
    - 2.5.1 添加一个 *post /persons* 操作
    - 2.5.2 定义消息体参数
    - 2.5.3 定义响应消息
- 第3章 文档瘦身
  - 3.1 简化数据模型
    - 3.1.1 添加`定义`（definitions）项
    - 3.1.2 增加一个可重用的（对象）`定义`
    - 3.1.3 引用一个`定义`来增加另一个`定义`
    - 3.1.4 在响应消息中使用`定义`
      - 3.1.4.1 get/persons
      - 3.1.4.2 get/persons/{username}
    - 3.1.5 在参数中使用`定义`
      - 3.1.5.1 post /persons
  - 3.2 简化响应消息
    - 3.2.1 定义可重用的HTTP 500 响应

3.2.2 增加一个Error定义
3.2.3 定义一个可重用的响应消息
3.2.4 使用已定义的响应消息
3.2.4.1 get /users
3.2.4.2 post/users
3.2.4.3 get/users/{username}
3.3 简化参数定义
3.3.1 路径参数只定义一次
3.3.2 定义可重用的参数
3.3.2.1 定义可重用的参数
3.3.2.2 使用定义参数
3.3.2.2.1 get /persons
3.3.2.2.2 get 和 delete /persons/{username}
3.3.2.2.3 get /persons/{username}/friends
第4章 深入了解一下
4.1 私人定制
4.1.1 字符串（Strings）长度和格式
4.1.2 日期和时间
4.1.3 数字类型和范围
4.1.4 枚举类型
4.1.5 数值的大小和唯一性
4.1.6 二进制数据
4.2 高级数据定义
4.2.1 读写操作同一定义的数据
4.2.2 组合定义确保一致性
4.2.3 数据模型的继承（TODO）
第5章 输入输出模型
5.1 高级参数定义
5.1.1 必带参数和可选参数
5.1.1.1 定义必带参数和可选参数
5.1.1.2 定义必带属性和可选属性
5.1.2 带默认值的参数
5.1.2.1 定义参数的默认值
5.1.2.2 定义属性的默认值
5.1.3 带空值的参数
5.1.4 参数组
5.1.5 消息头（Header）参数
5.1.6 表单参数
5.1.7 文件参数
5.1.8 参数的媒体类型
5.2 高级响应消息定义
5.2.1 不带消息体的响应消息
5.2.2 响应消息中的必带参数和可选参数
5.2.3 响应消息头
5.2.4 默认响应消息
5.2.5 响应消息的媒体类型
5.3 定义某个参数只存在于响应消息中
第6章 不要让API裸奔
6.1 定义安全
6.1.1 基础鉴权（Basic Authentication）
6.1.2 API秘钥鉴权（API Key）
6.1.3 Oauth2鉴权
6.1.3.1 流程（Flow）和URL
6.1.3.2 作用范围（scope）
6.2 使用安全定义
6.2.1 基础鉴权
6.2.1.1 API级别
6.2.1.2 操作级别

6.2.2 API秘钥鉴权
6.2.3 Oauth2鉴权
6.3 使用多种安全配置
6.3.1 安全定义
6.3.2 全局安全配置
6.3.3 覆盖全局配置
第7章 让文档的可读性更好
7.1 分类标签（Tags）
7.1.1 单标签
7.1.2 多标签
7.2 无处不在的描述文字（Descriptions）
7.2.1 安全项的描述
7.2.2 模式（Schema）的描述
7.2.3 属性的描述
7.2.4 参数的描述

前言

编写目的

本文介绍如何使用Swagger编写API文档。通过阅读本文，你可以：

- 了解swagger是什么
- 掌握使用swagger编写API文档的基本方法

涉及范围

- 本文包括对swagger specification（以下译作"规范"）的介绍，如何使用swaager协议编写出功能完整、结构清晰的API文档，以及项目实践中需要注意的问题。
- swagger的生态完整，从文档生成、编辑、测试到各种语言的代码自动生成，都有很多开源工具支持。本文中**不介绍**这些工具的使用。

第1章 简介

1.1 Swagger

> **The World's Most Popular Framework for APIs.**
>
> **Starting January 1st 2016 the Swagger Specification has been donated（捐赠） to the Open API Initiative (OAI) and is the foundation of the OpenAPI Specification.**

Swagger（丝袜哥）给人第一印象就是【最（hen）流（niu）行（bai）】，不懂Swagger咱就out了。它的官方网站是http://swagger.io/。

Swagger是一个简单但功能强大的API表达工具。它具有地球上最大的API工具生态系统，数以千计的开发人员，使用几乎所有的现代编程语言，都在支持和使用Swagger。使用Swagger生成API，我们可以得到交互式文档，自动生成代码的SDK以及API的发现特性等。

现在，Swagger已经帮助包括Apigee, Getty图像, Intuit, LivingSocial, McKesson, 微软, Morningstar和PayPal等世界知名企业建立起了一套基于RESTful API的完美服务系统。

2.0版本已经发布，Swagger变得更加强大。值得感激的是，Swagger的源码100%开源在github。

1.2 OpenAPI规范

OpenAPI规范是Linux基金会的一个项目，试图通过定义一种用来描述API格式或API定义的语言，来规范RESTful服务开发过程。OpenAPI规范帮助我们描述一个API的基本信息，比如：

- 有关该API的一般性描述
- 可用路径（/资源）
- 在每个路径上的可用操作（获取/提交...）
- 每个操作的输入/输出格式

目前V2.0版本的OpenAPI规范（也就是SwaggerV2.0规范）已经发布并开源在github上。该文档写的非常好，结构清晰，方便随时查阅。关于规范的学习和理解，本文最后还有个彩蛋。

1.3 为啥要使用OpenAPI规范？

- OpenAPI规范这类API定义语言能够帮助你更简单、快速的表述API，尤其是在API的设计阶段作用特别突出
- 根据OpenAPI规范编写的二进制文本文件，能够像代码一样用任何VCS工具管理起来
- 一旦编写完成，API文档可以作为：
  - 需求和系统特性描述的根据
  - 前后台查询、讨论、自测的基础
  - 部分或者全部代码自动生成的根据
  - 其他重要的作用，比如开放平台开发者的手册...

1.4 如何编写API文档？

1.4.1 语言：JSON vs YAML

我们可以选择使用JSON或者YAML的语言格式来编写API文档。但是个人建议使用YAML来写，原因是**它更简单**。一图胜千言，先看用JSON写的文档：

```
{
    "swagger": "2.0",
    "info": {
        "version": "1.0.0",
        "title": "Simple API",
        "description": "A simple API to learn how to write OpenAPI Specification"
    },
    "schemes": [
        "https"
    ],
    "host": "simple.api",
    "basePath": "/openapi101",
    "paths": {
        "/persons": {
            "get": {
                "summary": "Gets some persons",
                "description": "Returns a list containing all persons.",
                "responses": {
                    "200": {
```

```
                        "description": "A list of Person",
                        "schema": {
                            "type": "array",
                            "items": {
                                "properties": {
                                    "firstName": {
                                        "type": "string"
                                    },
                                    "lastName": {
                                        "type": "string"
                                    },
                                    "username": {
                                        "type": "string"
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

再来看看同一份API文档的YAML实现：

```
swagger: "2.0"

info:
  version: 1.0.0
  title: Simple API
  description: A simple API to learn how to write OpenAPI Specification

schemes:
  - https
host: simple.api
basePath: /openapi101

paths:
  /persons:
    get:
      summary: Gets some persons
      description: Returns a list containing all persons.
      responses:
        200:
          description: A list of Person
          schema:
            type: array
            items:
              required:
                - username
              properties:
                firstName:
                  type: string
                lastName:
                  type: string
                username:
```

```
            type: string
```

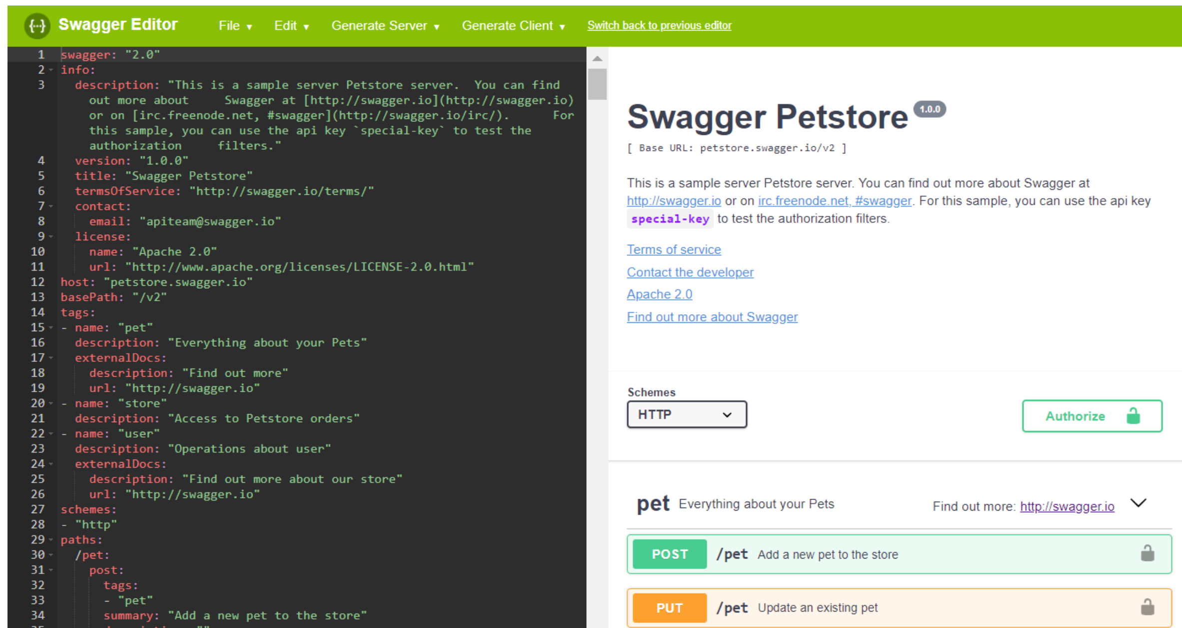
对于普通人来说，似乎用YAML更能够简化书写和阅读。这里我们并没有非此即彼的选择问题，因为：

- 几乎所用支持OpenAPI规范的工具都支持YAML
- 有很多的工具可以实现YAML-JSON之间的转换

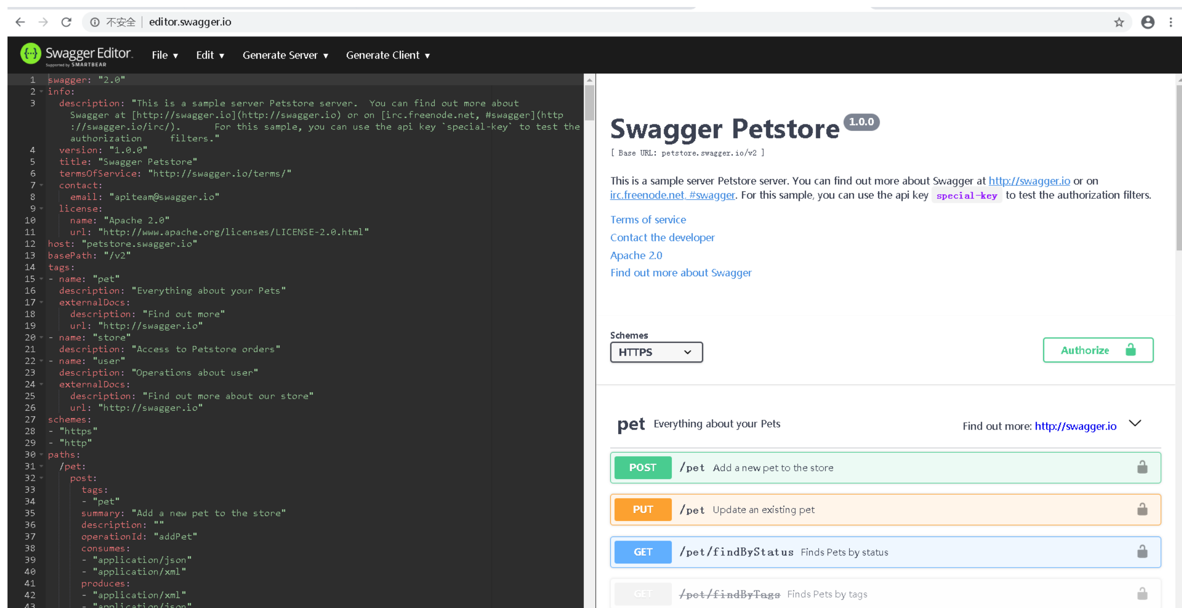
所以，用自己喜欢的方式书写即可。（后面的示例文档也都是用YAML来写的。强烈推荐使用YAML。）

1.4.2 编辑器

编写API文档，其实我们只是在写一个简单的文本文件。我们可以用任何最普通的文档编辑器来写。但是为了提高效率，还是建议使用专业的编辑工具。众多工具中，最好的选择是Swagger Editor，它能够提供语法高亮、自动完成、即时预览等功能，非常强大。



左边编辑API文档，右边实时预览。下面这张动图展示编辑提升功能：



我们可以使用在线版本来编辑，也可以非常简单的本地部署，更多细节请参考另一篇文档Swagger环境搭建.md以及Swagger Editor开源仓库上的说明。

第2章 从零开始

这一章主要介绍API的基本组成部分，包括提供给API消费者（所有可能访问API的个体，下简称"消费者"）的的不同HTTP请求方法、路径，请求和消息体中的参数，以及返回给消费者的不同HTTP状态及响应消息体。

2.1 最简单的例子

我们从一个最简单（几乎没有东西）的API文档开始：

```yaml
swagger: "2.0"

info:
  version: 1.0.0
  title: Simple API
  description: A simple API to learn how to write OpenAPI Specification

schemes:
  - https
host: simple.api
basePath: /openapi101

paths: {}
```

这个文档的内容分成四部分，下面分别来说明。

2.1.1 OpenAPI规范的版本号

首先我们要通过一个`swagger`属性来声明OpenAPI规范的版本。

```yaml
swagger: "2.0"
```

你没看错，是`swagger`，上面已经介绍了，OpenAPI规范是基于Swagger的，在未来的版本中，这个属性可能会换成别的。 目前这个属性的值，暂时只能填写为`2.0`。

2.1.2 API描述信息

然后我们需要说明一下API文档的相关信息，比如API文档版本（注意不同于上面的规范版本）、API文档名称已经可选的描述信息。

```yaml
info:
  version: 1.0.0
  title: Simple API
  description: A simple API to learn how to write OpenAPI Specification
```

2.1.3 API的URL

作为web API，一个很重要的信息就是用来给消费者使用的`根URL`，可以用协议（http或者https）、主机名、根路径来描述：

```yaml
schemes:
  - https
host: simple.api
basePath: /openapi101
```

这这个例子中，消费者把`https://simple.api/open101`作为根节点来访问各种API。因为和具体环境有关，不涉及API描述的根本内容，所以这部分信息是可选的。

2.1.4 API的操作（operation）

这个例子中，我们没有写API的操作，用一个YAML的空对象`{}`先占个位置。

2.2 定义一个API操作

如果我们要展示一组用户信息，可以这样描述：

```yaml
swagger: "2.0"

info:
  version: 1.0.0
  title: Simple API
  description: A simple API to learn how to write OpenAPI Specification

schemes:
  - https
host: simple.api
basePath: /openapi101

paths:
  /persons:
    get:
      summary: Gets some persons
      description: Returns a list containing all persons.
      responses:
        200:
          description: A list of Person
          schema:
            type: array
            items:
              required:
                - username
              properties:
                firstName:
                  type: string
                lastName:
                  type: string
                username:
                  type: string
```

2.2.1 添加一个`路径`（path）

我们添加一个`/persons`的`路径`，用来访问一组用户信息：

```yaml
paths:
  /persons:
```

2.2.2 在路径中添加一个HTTP方法

在每个`路径`中，我们可以添加任意的HTTP动词来操作所需要的资源。

比如需要展示一组用户信息，我们可以在`/persons`路径中添加`get`方法，同时还可以填写一些简单的描述信息（summary）或者说明该方法的一段长篇大论（description）。

```
    get:
      summary: Gets some persons
      description: Returns a list containing all persons.
```

这样一来，我们调 *get* *https://simple.api/open101/persons*方法就能获取一个用户信息列表了。

2.2.3 定义`响应`（response）类型

对于每个方法（或操作），我们都可以在`响应`(responses)中添加任意的HTTP状态码（比如*200 OK*或者 *404 Not Found*等）。这个例子中我们添加上`200`的响应：

```
      responses:
        200:
          description: A list of Person
```

2.2.4 定义响应内容

*get /persons*这个接口返回一组用户信息，我们通过响应消息中的`模式`（schema）属性来描述清楚具体的返回内容。

一组用户信息就是一个用户信息对象的`数组`（array），每一个数组元素则是一个用户信息`对象`（object），该对象包含三个string类型的属性：`姓氏`、`名字`、`用户名`，其中`用户名`必须提供（required）。

```
          schema:
            type: array
            items:
              required:
                - username
              properties:
                firstName:
                  type: string
                lastName:
                  type: string
                username:
                  type: string
```

2.3 定义`请求参数`（query parameters）

用户太多，我们不想一股脑全部输出出来。这个时候，分页输出是个不错的选择，我们可以通过添加请求参数来实现。

```
swagger: "2.0"

info:
  version: 1.0.0
  title: Simple API
  description: A simple API to learn how to write OpenAPI Specification

schemes:
  - https
```

```yaml
host: simple.api
basePath: /openapi101

paths:
  /persons:
    get:
      summary: Gets some persons
      description: Returns a list containing all persons. The list supports
paging.
#START##########################################################################
##
      parameters:
       - name: pageSize
         in: query
         description: Number of persons returned
         type: integer
       - name: pageNumber
         in: query
         description: Page number
         type: integer
# END
############################################################################

      responses:
        200:
          description: A list of Person
          schema:
            type: array
            items:
              required:
                - username
              properties:
                firstName:
                  type: string
                lastName:
                  type: string
                username:
                  type: string
```

2.3.1 在*get*方法中增加请求参数

首先我们在 *get* 方法中增加一个`参数`属性：

```yaml
paths:
  /persons:
    get:
      summary: Gets some persons
      description: Returns a list containing all persons. The list supports
paging.
#START##########################################################################
##
      parameters:
# END
############################################################################
```

2.3.2 添加分页参数

在参数列表中，我们添加两个名字（name）分别叫做`pageSize`和`pageNumber`的整型（integer）参数，并作简单描述：

```
      parameters:
#START############################################################################
##
        - name: pageSize
          in: query
          description: Number of persons returned
          type: integer
        - name: pageNumber
          in: query
          description: Page number
          type: integer
# END
##############################################################################
      responses:
```

这样一来，消费者就可以通过 *get /persons?pageSize=20&pageNumber=2* 来访问第2页的用户信息（不超过20条）了。

2.4 定义`路径参数`（path parameter）

有时候我们想要根据用户名来查找用户信息，这时我们需要增加一个接口操作，比如可以添加一个类似 */persons/{username}* 的操作来获取用户信息。注意，*{username}* 是在请求路径中的参数。

```
swagger: "2.0"

info:
  version: 1.0.0
  title: Simple API
  description: A simple API to learn how to write OpenAPI Specification

schemes:
  - https
host: simple.api
basePath: /openapi101

paths:
  /persons:
    get:
      summary: Gets some persons
      description: Returns a list containing all persons. The list supports
paging.
      parameters:
        - name: pageSize
          in: query
          description: Number of persons returned
          type: integer
        - name: pageNumber
          in: query
          description: Page number
          type: integer
      responses:
        200:
          description: A list of Person
```

```
          schema:
            type: array
            items:
              required:
                - username
              properties:
                firstName:
                  type: string
                lastName:
                  type: string
                username:
                  type: string

#START##########################################################################
##
  /persons/{username}:
    get:
      summary: Gets a person
      description: Returns a single person for its username
      parameters:
        - name: username
          in: path
          required: true
          description: The person's username
          type: string
      responses:
        200:
          description: A Person
          schema:
            required:
              - username
            properties:
              firstName:
                type: string
              lastName:
                type: string
              username:
                type: string
        404:
          description: The Person does not exists.
# END
############################################################################
```

2.4.1 添加一个 *get /persons/{username}* 操作

首先我们在 */persons* 路径后面，增加一个 */persons/{username}* 的路径，并定义一个 *get*（操作）方法。

```
swagger: "2.0"

info:
  version: 1.0.0
  title: Simple API
  description: A simple API to learn how to write OpenAPI Specification

schemes:
  - https
```

```
host: simple.api
basePath: /openapi101

paths:
  /persons:
                 username:
                   type: string
#START##########################################################################
##
  /persons/{username}:
    get:
      summary: Gets a person
      description: Returns a single person for its username
# END
############################################################################
```

2.4.2 定义路径参数 *username*

因为 *{username}* 是路径参数，我们需要先像请求参数一样将它添加到 *parameters* 属性中，注意名称应该同上面大括号（ *{ }* ）里面的名称一致。并通过 `in` 这个属性，来表示它是一个路径（path）参数。

```
      parameters:
        - name: username
          in: path
          required: true
          description: The person's username
          type: string
```

定义路径参数时很容易出现的问题就是忘记：`required: true`，Swagger的自动完成功能中没有包含这个属性定义。 如果没有写 *require* 属性，默认值是 **false**，也就是说 *username* 参数时可选的。可事实上，作为路径参数，它是必需的。

2.4.3 定义响应消息

别忘了获取单个用户信息也需要填写 *200* 响应消息，响应消息体的内容就是之前描述过的用户信息（用户信息列表中的一个元素）：

```
      responses:
        200:
          description: A Person
          schema:
            required:
              - username
            properties:
              firstName:
                type: string
              lastName:
                type: string
              username:
                type: string
```

当然，API的提供者会对 *username* 进行校验，如果查无此人，应该返回 *404* 的异常状态。所以我们再加上 *404* 状态的响应：

```
        404:
          description: The Person does not exists.
```

2.5 定义`消息体参数`（body parameter）

当我们需要添加一个用户信息时，我们需要一个能够提供 *post /persons* 的API操作。

```
swagger: "2.0"

info:
  version: 1.0.0
  title: Simple API
  description: A simple API to learn how to write OpenAPI Specification

schemes:
  - https
host: simple.api
basePath: /openapi101

paths:
  /persons:
    get:
      summary: Gets some persons
      description: Returns a list containing all persons. The list supports
paging.
      parameters:
       - name: pageSize
         in: query
         description: Number of persons returned
         type: integer
       - name: pageNumber
         in: query
         description: Page number
         type: integer
      responses:
        200:
          description: A list of Person
          schema:
            type: array
            items:
              required:
                - username
              properties:
                firstName:
                  type: string
                lastName:
                  type: string
                username:
                  type: string
#START################################################################################
##
    post:
      summary: Creates a person
      description: Adds a new person to the persons list.
      parameters:
        - name: person
```

```yaml
          in: body
          description: The person to create.
          schema:
            required:
              - username
            properties:
              firstName:
                type: string
              lastName:
                type: string
              username:
                type: string
      responses:
        204:
          description: Persons succesfully created.
        400:
          description: Persons couldn't have been created.
# END
############################################################################
  /persons/{username}:
    get:
      summary: Gets a person
      description: Returns a single person for its username.
      parameters:
        - name: username
          in: path
          required: true
          description: The person's username
          type: string
      responses:
        200:
          description: A Person
          schema:
            required:
              - username
            properties:
              firstName:
                type: string
              lastName:
                type: string
              username:
                type: string
        404:
          description: The Person does not exists.
```

2.5.1 添加一个 *post /persons* 操作

首先在 */persons* 路径下廷加一个 *post* 操作：

```yaml
paths:
  /persons:
    post:
      summary: Creates a person
      description: Adds a new person to the persons list.
```

2.5.2 定义消息体参数

接下来我们给 *post* 方法添加参数，通过 `in` 属性显式说明参数是在 *body* 中的。参数的定义参考 *get /persons/{username}* 的 *200* 响应消息体参数，也就是包含用户的姓氏、名字、用户名。

```
      parameters:
        - name: person
          in: body
          description: The person to create.
          schema:
            required:
              - username
            properties:
              firstName:
                type: string
              lastName:
                type: string
              username:
                type: string
```

2.5.3 定义响应消息

最后不要忘记定义 *post* 操作的响应消息。

```
      responses:
        204:
          description: Persons succesfully created.
        400:
          description: Persons couldn't have been created.
```

第3章 文档瘦身

现在我们已经学会了编写API文档的基本方法。不过上面的例子中存在一些重复，这对于程序员的嗅觉来说，就是代码的“坏味道”。这一章我们一起学习如何通过抽取可重用的定义（definitions）来简化API文档。

3.1 简化数据模型

我们认真观察第2章最后输出的API文档，很容易发现 *Person* 的定义出现了三次，非常的不 `DRY` ☹。

```
swagger: "2.0"

info:
  version: 1.0.0
  title: Simple API
  description: A simple API to learn how to write OpenAPI Specification

schemes:
  - https
host: simple.api
basePath: /openapi101

paths:
  /persons:
    get:
      summary: Gets some persons
```

```
      description: Returns a list containing all persons. The list supports
paging.
      parameters:
       - name: pageSize
         in: query
         description: Number of persons returned
         type: integer
       - name: pageNumber
         in: query
         description: Page number
         type: integer
      responses:
        200:
          description: A list of Person
          schema:
            type: array
            items:
#START 第1次定义
###################################################################
              required:
                - username
              properties:
                firstName:
                  type: string
                lastName:
                  type: string
                username:
                  type: string
# END  第1次定义
###################################################################
    post:
      summary: Creates a person
      description: Adds a new person to the persons list.
      parameters:
        - name: person
          in: body
          description: The person to create.
          schema:
#START 第2次定义
###################################################################
            required:
              - username
            properties:
              firstName:
                type: string
              lastName:
                type: string
              username:
                type: string
# END  第2次定义
###################################################################
      responses:
        204:
          description: Persons succesfully created.
        400:
          description: Persons couldn't have been created.
  /persons/{username}:
    get:
```

```
      summary: Gets a person
      description: Returns a single person for its username.
      parameters:
        - name: username
          in: path
          required: true
          description: The person's username
          type: string
      responses:
        200:
          description: A Person
          schema:
#START 第3次定义
###################################################################
            required:
              - username
            properties:
              firstName:
                type: string
              lastName:
                type: string
              username:
                type: string
# END  第3次定义
###################################################################
        404:
          description: The Person does not exists.
```

现在，我们通过可重用的`定义`（definition）来重构这个文档：

```
swagger: "2.0"

info:
  version: 1.0.0
  title: Simple API
  description: A simple API to learn how to write OpenAPI Specification

schemes:
  - https
host: simple.api
basePath: /openapi101

paths:
  /persons:
    get:
      summary: Gets some persons
      description: Returns a list containing all persons. The list supports
paging.
      parameters:
       - name: pageSize
         in: query
         description: Number of persons returned
         type: integer
       - name: pageNumber
         in: query
         description: Page number
         type: integer
```

```
      responses:
        200:
          description: A list of Person
          schema:
#START##########################################################################
##
            $ref: "#/definitions/Persons"
# END
############################################################################
    post:
      summary: Creates a person
      description: Adds a new person to the persons list.
      parameters:
        - name: person
          in: body
          description: The person to create.
          schema:
#START##########################################################################
##
            $ref: "#/definitions/Person"
# END
############################################################################
      responses:
        204:
          description: Persons succesfully created.
        400:
          description: Persons couldn't have been created.
  /persons/{username}:
    get:
      summary: Gets a person
      description: Returns a single person for its username.
      parameters:
        - name: username
          in: path
          required: true
          description: The person's username
          type: string
      responses:
        200:
          description: A Person
          schema:
#START##########################################################################
##
            $ref: "#/definitions/Person"
# END
############################################################################
        404:
          description: The Person does not exists.

#START 新增定义
####################################################################
definitions:
  Person:
    required:
      - username
    properties:
      firstName:
        type: string
```

```
        lastName:
          type: string
        username:
          type: string
  Persons:
    type: array
    items:
      $ref: "#/definitions/Person"
# END 新增定义####################################################################
```

文档简化了很多。这得益于[OpenAPI规范中关于定义（definition）的章节中]允许我们“一处定义，处处使用”。

3.1.1 添加`定义`（definitions）项

我们首先在API文档的尾部添加一个`定义`（definitions）项（其实它也可以放在文档的任意位置，只不过大家习惯放在文档末尾）：

```
        404:
          description: The Person does not exists.
#START############################################################################
##
definitions:
# END
############################################################################
```

3.1.2 增加一个可重用的（对象）`定义`

然后我们增加一个 *Person* 对象的定义：

```
definitions:
#START############################################################################
##
  Person:
    required:
      - username
    properties:
      firstName:
        type: string
      lastName:
        type: string
      username:
        type: string
# END
############################################################################
```

3.1.3 引用一个`定义`来增加另一个`定义`

在定义项中，我们可以立即引用刚才定义好的 *Person* 来增加另一个`定义`，*Persons*。*Persons* 是一个 *Person* 对象的数组。与之前直接定义的不同之处是，我们增加了一个`引用`（*reference*）属性，也就是 *$ref* 来引用 *Person* 。

```
  Persons:
    type: array
    items:
      $ref: "#/definitions/Person"
```

3.1.4 在响应消息中使用`定义`

一旦定义好了 *Person* ，我们可以把原来在响应消息中相应的定义字段替换掉。

3.1.4.1 get/persons

原来：

```
      responses:
        200:
          description: A list of Person
          schema:
            type: array
            items:
              required:
                - username
              properties:
                firstName:
                  type: string
                lastName:
                  type: string
```

现在：

```
      responses:
        200:
          description: A list of Person
          schema:
            $ref: "#/definitions/Persons"
```

3.1.4.2 get/persons/{username}

原来：

```
      responses:
        200:
          description: A Person
          schema:
            required:
              - username
            properties:
              firstName:
                type: string
              lastName:
                type: string
              username:
                type: string
```

现在：

```
    responses:
      200:
        description: A Person
        schema:
          $ref: "#/definitions/Person"
```

3.1.5 在参数中使用`定义`

不仅仅在消息中可以使用`定义`，在参数中也可以使用。

3.1.5.1 post /persons

原来：

```
  post:
    summary: Creates a person
    description: Adds a new person to the persons list.
    parameters:
      - name: person
        in: body
        description: The person to create.
        schema:
          required:
            - username
          properties:
            firstName:
              type: string
            lastName:
              type: string
            username:
              type: string
```

现在：

```
  post:
    summary: Creates a person
    description: Adds a new person to the persons list.
    parameters:
      - name: person
        in: body
        description: The person to create.
        schema:
          $ref: "#/definitions/Person"
```

3.2 简化响应消息

我们看到了`引用`（*$ref*）的作用，接下来我们再把它用到响应消息的定义中：

```
swagger: "2.0"

info:
  version: 1.0.0
  title: Simple API
  description: A simple API to learn how to write OpenAPI Specification
```

```
schemes:
  - https
host: simple.api
basePath: /openapi101

paths:
  /persons:
    get:
      summary: Gets some persons
      description: Returns a list containing all persons. The list supports
paging.
      parameters:
       - name: pageSize
         in: query
         description: Number of persons returned
         type: integer
       - name: pageNumber
         in: query
         description: Page number
         type: integer
      responses:
        200:
          description: A list of Person
          schema:
            $ref: "#/definitions/Persons"
#START##########################################################################
##
        500:
          $ref: "#/responses/Standard500ErrorResponse"
# END
############################################################################
    post:
      summary: Creates a person
      description: Adds a new person to the persons list.
      parameters:
        - name: person
          in: body
          description: The person to create.
          schema:
            $ref: "#/definitions/Person"
      responses:
        204:
          description: Persons succesfully created.
        400:
          description: Persons couldn't have been created.
#START##########################################################################
##
        500:
          $ref: "#/responses/Standard500ErrorResponse"
# END
############################################################################

  /persons/{username}:
    get:
      summary: Gets a person
      description: Returns a single person for its username.
      parameters:
        - name: username
```

```
          in: path
          required: true
          description: The person's username
          type: string
      responses:
        200:
          description: A Person
          schema:
            $ref: "#/definitions/Person"
        404:
          description: The Person does not exists.
#START##########################################################################
##
        500:
          $ref: "#/responses/Standard500ErrorResponse"
# END
############################################################################

definitions:
  Person:
    required:
      - username
    properties:
      firstName:
        type: string
      lastName:
        type: string
      username:
        type: string
  Persons:
    type: array
    items:
      $ref: "#/definitions/Person"
#START##########################################################################
##
  Error:
    properties:
      code:
        type: string
      message:
        type: string

responses:
  Standard500ErrorResponse:
    description: An unexpected error occured.
    schema:
      $ref: "#/definitions/Error"
# END
############################################################################
```

3.2.1 定义可重用的HTTP 500 响应

发生HTTP 500错误时，假如我们希望每一个API操作都返回一个带有错误码（error code）和描述信息（message）的响应，我们可以这样做：

```
paths:
  /persons:
```

```
    get:
      summary: Gets some persons
      description: Returns a list containing all persons. The list supports
paging.
      parameters:
       - name: pageSize
         in: query
         description: Number of persons returned
         type: integer
       - name: pageNumber
         in: query
         description: Page number
         type: integer
      responses:
        200:
          description: A list of Person
          schema:
            $ref: "#/definitions/Persons"
#START##########################################################################
##
        500:
          description: An unexpected error occured.
          schema:
            properties:
              code:
                type: string
              message:
                type: string
# END
############################################################################
    post:
      summary: Creates a person
      description: Adds a new person to the persons list.
      parameters:
        - name: person
          in: body
          description: The person to create.
          schema:
            $ref: "#/definitions/Person"
      responses:
        204:
          description: Persons succesfully created.
        400:
          description: Persons couldn't have been created.
#START##########################################################################
##
        500:
          description: An unexpected error occured.
          schema:
            properties:
              code:
                type: string
              message:
                type: string
# END
############################################################################

  /persons/{username}:
```

```
    get:
      summary: Gets a person
      description: Returns a single person for its username.
      parameters:
        - name: username
          in: path
          required: true
          description: The person's username
          type: string
      responses:
        200:
          description: A Person
          schema:
            $ref: "#/definitions/Person"
        404:
          description: The Person does not exists.
#START##########################################################################
##
        500:
          description: An unexpected error occured.
          schema:
            properties:
              code:
                type: string
              message:
                type: string
# END
############################################################################
```

3.2.2 增加一个Error定义

按照“一处定义、处处引用”的原则，我们可以在定义项中增加 *Error* 的定义：

```
definitions:
  Person:
    required:
      - username
    properties:
      firstName:
        type: string
      lastName:
        type: string
      username:
        type: string
  Persons:
    type: array
    items:
      $ref: "#/definitions/Person"
#START##########################################################################
##
  Error:
    properties:
      code:
        type: string
      message:
        type: string
```

```
# END
############################################################################
```

而且我们也学会了使用`引用`（*$ref*），所以我们可以这样写：

```
paths:
  /persons:
    get:
      summary: Gets some persons
      description: Returns a list containing all persons. The list supports
paging.
      parameters:
       - name: pageSize
         in: query
         description: Number of persons returned
         type: integer
       - name: pageNumber
         in: query
         description: Page number
         type: integer
      responses:
        200:
          description: A list of Person
          schema:
            $ref: "#/definitions/Persons"
        500:
          description: An unexpected error occured.
          schema:
#START################################################################################
##
            $ref: "#/definitions/Error"
# END
############################################################################
    post:
      summary: Creates a person
      description: Adds a new person to the persons list.
      parameters:
        - name: person
          in: body
          description: The person to create.
          schema:
            $ref: "#/definitions/Person"
      responses:
        204:
          description: Persons succesfully created.
        400:
          description: Persons couldn't have been created.
        500:
          description: An unexpected error occured.
          schema:
#START################################################################################
##
            $ref: "#/definitions/Error"
# END
############################################################################
  /persons/{username}:
    get:
```

```
      summary: Gets a person
      description: Returns a single person for its username.
      parameters:
        - name: username
          in: path
          required: true
          description: The person's username
          type: string
      responses:
        200:
          description: A Person
          schema:
            $ref: "#/definitions/Person"
        404:
          description: The Person does not exists.
        500:
          description: An unexpected error occured.
          schema:
#START################################################################################
##
            $ref: "#/definitions/Error"
# END
############################################################################
```

3.2.3 定义一个可重用的响应消息

上面的文档中，还是有一些重复的内容。我们可以根据OpenAPI规范中的responses章节的描述，通过定义一个可重用的响应消息，来进一步简化文档。

```
definitions:
  Person:
    required:
      - username
    properties:
      firstName:
        type: string
      lastName:
        type: string
      username:
        type: string
  Persons:
    type: array
    items:
      $ref: "#/definitions/Person"
  Error:
    properties:
      code:
        type: string
      message:
        type: string
#START################################################################################
##
responses:
  Standard500ErrorResponse:
    description: An unexpected error occured.
    schema:
      $ref: "#/definitions/Error"
```

```
# END
############################################################################
```

注意：响应消息中引用了 *Error* 的定义。

3.2.4 使用已定义的响应消息

我们还是通过`引用`（*$ref*）来使用一个已经定义好的响应消息，比如：

3.2.4.1 get /users

```
      responses:
        200:
          description: A list of Person
          schema:
            $ref: "#/definitions/Persons"
#START##########################################################################
##
        500:
          $ref: "#/responses/Standard500ErrorResponse"
# END
############################################################################
```

3.2.4.2 post/users

```
      responses:
        204:
          description: Persons succesfully created.
        400:
          description: Persons couldn't have been created.
#START##########################################################################
##
        500:
          $ref: "#/responses/Standard500ErrorResponse"
# END
############################################################################
```

3.2.4.3 get/users/{username}

```
      responses:
        200:
          description: A Person
          schema:
            $ref: "#/definitions/Person"
        404:
          description: The Person does not exists.
#START##########################################################################
##
        500:
          $ref: "#/responses/Standard500ErrorResponse"
# END
############################################################################
```

3.3 简化参数定义

类似数据模型、响应消息的简化，参数定义的简化也很容易。

```
swagger: "2.0"

info:
  version: 1.0.0
  title: Simple API
  description: A simple API to learn how to write OpenAPI Specification

schemes:
  - https
host: simple.api
basePath: /openapi101

paths:
  /persons:
    get:
      summary: Gets some persons
      description: Returns a list containing all persons. The list supports
paging.
#START##########################################################################
##
      parameters:
       - $ref: "#/parameters/pageSize"
       - $ref: "#/parameters/pageNumber"
# END
############################################################################
      responses:
        200:
          description: A list of Person
          schema:
            $ref: "#/definitions/Persons"
        500:
          $ref: "#/responses/Standard500ErrorResponse"
    post:
      summary: Creates a person
      description: Adds a new person to the persons list.
      parameters:
        - name: person
          in: body
          description: The person to create.
          schema:
            $ref: "#/definitions/Person"
      responses:
        204:
          description: Person succesfully created.
        400:
          description: Person couldn't have been created.
        500:
          $ref: "#/responses/Standard500ErrorResponse"

  /persons/{username}:
#START##########################################################################
##
    parameters:
      - $ref: "#/parameters/username"
# END
############################################################################
    get:
```

```
      summary: Gets a person
      description: Returns a single person for its username.
      responses:
        200:
          description: A Person
          schema:
            $ref: "#/definitions/Person"
        404:
          $ref: "#/responses/PersonDoesNotExistResponse"
        500:
          $ref: "#/responses/Standard500ErrorResponse"
    delete:
      summary: Deletes a person
      description: Delete a single person identified via its username
      responses:
        204:
          description: Person successfully deleted.
        404:
          $ref: "#/responses/PersonDoesNotExistResponse"
        500:
          $ref: "#/responses/Standard500ErrorResponse"

  /persons/{username}/friends:
#START##########################################################################
##
    parameters:
      - $ref: "#/parameters/username"
# END
############################################################################
    get:
      summary: Gets a person's friends
      description: Returns a list containing all persons. The list supports
paging.
#START##########################################################################
##
      parameters:
       - $ref: "#/parameters/pageSize"
       - $ref: "#/parameters/pageNumber"
# END
############################################################################
      responses:
        200:
          description: A person's friends list
          schema:
            $ref: "#/definitions/Persons"
        404:
          $ref: "#/responses/PersonDoesNotExistResponse"
        500:
          $ref: "#/responses/Standard500ErrorResponse"

definitions:
  Person:
    required:
      - username
    properties:
      firstName:
        type: string
      lastName:
```

```
        type: string
      username:
        type: string
  Persons:
    type: array
    items:
      $ref: "#/definitions/Person"
  Error:
    required:
      - code
      - message
    properties:
      code:
        type: string
      message:
        type: string

responses:
  Standard500ErrorResponse:
    description: An unexpected error occured.
    schema:
      $ref: "#/definitions/Error"
  PersonDoesNotExistResponse:
    description: Person does not exist.

#START##########################################################################
##
parameters:
  username:
    name: username
    in: path
    required: true
    description: The person's username
    type: string
  pageSize:
    name: pageSize
    in: query
    description: Number of persons returned
    type: integer
  pageNumber:
    name: pageNumber
    in: query
    description: Page number
    type: integer
# END ############################################################################
```

3.3.1 路径参数只定义一次

如果我们现在想要删除一个用户的信息，就需要增加一个 *delete /persons/{username}* 的操作，可以这样：

```
  /persons/{username}:
    get:
      summary: Gets a person
      description: Returns a single person for its username.
      parameters:
```

```
#START##########################################################################
##
        - name: username
          in: path
          required: true
          description: The person's username
          type: string
# END
############################################################################
      responses:
        200:
          description: A Person
          schema:
            $ref: "#/definitions/Person"
        404:
          $ref: "#/responses/PersonDoesNotExistResponse"
        500:
          $ref: "#/responses/Standard500ErrorResponse"
    delete:
      summary: Deletes a person
      description: Delete a single person identified via its username
      parameters:
#START##########################################################################
##
        - name: username
          in: path
          required: true
          description: The person's username
          type: string
# END
############################################################################
      responses:
        204:
          description: Person successfully deleted.
        404:
          $ref: "#/responses/PersonDoesNotExistResponse"
        500:
          $ref: "#/responses/Standard500ErrorResponse"
```

但是上面两次对参数 *username* 的定义，却让人有点难受。好消息是我们可以定义路径级别的参数（之前都是定义在操作级别。）

```
#START##########################################################################
##
  /persons/{username}:
    parameters:
      - name: username
        in: path
        required: true
        description: The person's username
        type: string
# END
############################################################################
    get:
      summary: Gets a person
      description: Returns a single person for its username.
      responses:
```

```
        200:
          description: A Person
          schema:
            $ref: "#/definitions/Person"
        404:
          $ref: "#/responses/PersonDoesNotExistResponse"
        500:
          $ref: "#/responses/Standard500ErrorResponse"
    delete:
      summary: Deletes a person
      description: Delete a single person identified via its username
      responses:
        204:
          description: Person successfully deleted.
        404:
          $ref: "#/responses/PersonDoesNotExistResponse"
        500:
          $ref: "#/responses/Standard500ErrorResponse"
```

3.3.2 定义可重用的参数

如果我们想根据用户名查找该用户的朋友圈，可以添加一个 *get /persons/{username}/friends* 的操作。根据前面所学的内容，第一反应应该这样写：

```
  /persons/{username}/friends:
    parameters:
      - name: username
        in: path
        required: true
        description: The person's username
        type: string
    get:
      summary: Gets a person's friends
      description: Returns a list containing all persons. The list supports
paging.
      parameters:
       - name: pageSize
         in: query
         description: Number of persons returned
         type: integer
       - name: pageNumber
         in: query
         description: Page number
         type: integer
      responses:
        200:
          description: A person's friends list
          schema:
            $ref: "#/definitions/Persons"
        404:
          $ref: "#/responses/PersonDoesNotExistResponse"
        500:
          $ref: "#/responses/Standard500ErrorResponse"
```

可以看到，关于 *username*、*pageSize*、*pageNumber* 的定义跟前面的 */person/{username}*、 *get /persons* 中的定义重复。如何消除重复呢？

3.3.2.1 定义可重用的参数

根据3.1和3.2中的内容，我们可以参考[OpenAPI规范]，融汇贯通。

```
parameters:
  username:
    name: username
    in: path
    required: true
    description: The person's username
    type: string
  pageSize:
    name: pageSize
    in: query
    description: Number of persons returned
    type: integer
  pageNumber:
    name: pageNumber
    in: query
    description: Page number
    type: integer
```

3.3.2.2 使用定义参数

借助万能的`引用`（*$ref*），这都是小菜一碟。比如：

3.3.2.2.1 get /persons

原来：

```
  /persons:
    get:
      summary: Gets some persons
      description: Returns a list containing all persons. The list supports
paging.
      parameters:
#START##########################################################################
##
       - name: pageSize
         in: query
         description: Number of persons returned
         type: integer
       - name: pageNumber
         in: query
         description: Page number
         type: integer
# END
############################################################################
```

现在：

```
  /persons:
    get:
      summary: Gets some persons
      description: Returns a list containing all persons. The list supports
paging.
      parameters:
#START##########################################################################
##
       - $ref: "#/parameters/pageSize"
       - $ref: "#/parameters/pageNumber"
# END
############################################################################
```

3.3.2.2.2 get 和 delete /persons/{username}

原来：

```
  /persons/{username}:
    parameters:
#START##########################################################################
##
      - name: username
        in: path
        required: true
        description: The person's username
        type: string
# END
############################################################################
```

现在：

```
  /persons/{username}:
    parameters:
#START##########################################################################
##
      - $ref: "#/parameters/username"
# END
############################################################################
```

3.3.2.2.3 get /persons/{username}/friends

原来：

```
  /persons/{username}/friends:
    parameters:
#START##########################################################################
##
      - name: username
        in: path
        required: true
        description: The person's username
        type: string
# END
############################################################################
    get:
      summary: Gets a person's friends
```

```
      description: Returns a list containing all persons. The list supports
paging.
      parameters:
#START################################################################################
##
        - name: pageSize
          in: query
          description: Number of persons returned
          type: integer
        - name: pageNumber
          in: query
          description: Page number
          type: integer
# END
############################################################################
```

现在：

```
  /persons/{username}/friends:
    parameters:
#START################################################################################
##
      - $ref: "#/parameters/username"
# END
############################################################################
    get:
      summary: Gets a person's friends
      description: Returns a list containing all persons. The list supports
paging.
      parameters:
#START################################################################################
##
        - $ref: "#/parameters/pageSize"
        - $ref: "#/parameters/pageNumber"
# END
############################################################################
```

第4章 深入了解一下

通过前面的练习，我们可以写出一篇结构清晰、内容精炼的API文档了。可是OpenAPI规范还给我们提供了更多的便利和惊喜，等着我们去了解和掌握。这一章主要介绍用于定义属性和数据模型的高级方法。

4.1 私人定制

> Primitive data types in the Swagger Specification are based on the types supported by the JSON-Schema Draft 4. Models are described using the Schema Object which is a subset of JSON Schema Draft 4. OpenAPI Specification Data Types

使用 JSON Schema Draft 4，我们可以定义任意类型的各种属性，举例说明。

4.1.1 字符串（Strings）长度和格式

当定义个字符串属性时，我们可以定制它的长度及格式：

属性	类型	描述
minLength	number	字符串最小长度
maxLength	number	字符串最大长度
pattern	string	正则表达式 (如果你暂时还不熟悉正则表达式，可以看看 Regex 101)

如果我们规定用户名是长度介于8~64，而且只能由小写字母和数字来构成，那么我们可以这样写：

```
username:
  type: string
  pattern: "[a-z0-9]{8,64}"
  minLength: 8
  maxLength: 64
```

4.1.2 日期和时间

日期和时间的处理参考 RFC 3339 ，我们唯一要做的就是写对格式：

格式	属性包含内容	属性示例
date	ISO8601 full-date	2016-04-01
dateTime	ISO8601 date-time	2016-04-16T16:06:05Z

如果我们在 *Person* 的定义中增加 `生日` 和 `上次登录时间` 时间戳，我们可以这样写：

```
dateOfBirth:
  type: string
  format: date
lastTimeOnline:
  type: string
  format: dateTime
```

如果想深入掌握API中处理日期和时间的方法，我们应该继续阅读 Jason Harmon 写的文章《 5 laws of API dates and times 》。

4.1.3 数字类型和范围

当我们定义一个数字类型的属性时，我们可以规定它是一个整型、长型、浮点型或者双浮点型。

名称	类型	格式
integer	integer	int32
long	integer	int64
float	number	float
double	number	double

和字符串一样，我们也可以定义数字属性的范围，比如：

属性	类型	描述
minimum	number	最小值
maximum	number	最大值
exclusiveMinimum	boolean	数值必须 > 最小值
exclusiveMaximum	boolean	数值必须 < 最大值
multipleOf	number	数值必须是multipleOf的整数倍

如果我们规定 *pageSize* 必须是整数，必须 > 0 且 <=100，还必须是 10 的整数倍，可以这样写：

```
pageSize:
  name: pageSize
  in: query
  description: Number of persons returned
  type: integer
  format: int32
  minimum: 0
  exclusiveMinimum: true
  maximum: 100
  exclusiveMaximum: false
  multipleOf: 10
```

4.1.4 枚举类型

我们还可以定义枚举类型，比如定义 *Error* 时，我们可以这样写：

```
      code:
        type: string
        enum:
          - DBERR
          - NTERR
          - UNERR
```

code 的值只能从三个枚举值中选择。

4.1.5 数值的大小和唯一性

数字的大小和唯一性通过下面这些属性来定义：

属性	类型	描述
minItems	number	数值中的最小元素个数
maxItem	number	数值中的最大元素个数
uniqueItems	boolean	标示数组中的元素是否唯一

比如我们定义一个用户数组 *Persons*，希望返回的用户信息条数介于10~100之间，而且不能有重复的用户信息，我们可以这样写：

```
  Persons:
    properties:
      items:
        type: array
        minItems: 10
        maxItems: 100
        uniqueItems: true
        items:
          $ref: "#/definitions/Person"
```

4.1.6 二进制数据

可以用 *string* 类型来表示二进制数据：

格式	属性包含
byte	Base64编码字符
binary	任意十进制的数据序列

比如我们需要在用户信息中增加一个头像属性（*avatarBase64PNG*）用base64编码的PNG图片来表示，可以这样写：

```
      avatarBase64PNG:
        type: string
        format: byte
```

4.2 高级数据定义

4.2.1 读写操作同一定义的数据

有时候我们读取资源信息的内容会比我们写入资源信息的内容（属性）更多，这很常见。是不是意味着我们必须专门为读取资源和写入资源分别定义不同的数据模型呢？幸运的是，OpenAPI规范中提供了 *readOnly* 字段来帮我们解决整问题。比如：

```
      lastTimeOnline:
        type: string
        format: dateTime
        readOnly: true
```

上面这个例子中，`上次在线时间`（*lastTimeOnline* ）是 *Person* 的一个属性，我们获取用户信息时需要这个属性。但是很明显，在创建用户时，我们不能把这个属性 *post* 到服务器。于是我们可以把它标记为 *readOnly*。

4.2.2 组合定义确保一致性

一致性设计是在编写API文档时需要重点考虑的问题。比如我们在获取一组用户信息时，需要同时获取 `页面信息`（

totalItems, totalPage, pageSize, currentPage）等，而且这些信息**必须**在根节点上。

怎么办呢？首先想到的做法就是：

```
  PagedPersonsV1:
    properties:
```

```yaml
      items:
        type: array
        items:
          $ref: "#/definitions/Person"
      totalItems:
        type: integer
      totalPages:
        type: integer
      pageSize:
        type: integer
      currentPage:
        type: integer
```

如果其他API操作也需要这些`页面信息`，那就意味着这些属性必须一遍又一遍的定义。不仅重复体力劳动，而且还很危险：比如忘记了其中的一两个属性，或者需要添加一个新的属性进来，那就是霰弹式的修改，想想都很悲壮。

稍微好一点的做法，就是根据前面学习的内容，把这几个属性抽取出来，建立一个 *Paging* 模型，"一处定义、处处使用"：

```yaml
  PagedPersonsV2:
    properties:
      items:
        type: array
        items:
          $ref: "#/definitions/Person"
      paging:
        $ref: "#/definitions/Paging"

  Paging:
    properties:
      totalItems:
        type: integer
      totalPages:
        type: integer
      pageSize:
        type: integer
      currentPage:
        type: integer
```

但是，页面属性都不再位于 **根节点**！与我们前面设定的要求不一样了。怎么破？

[JSON Schema v4 property](#)中定义的`allOf`，能帮我们解围：

```yaml
  PagedPersons:
    allOf:
      - $ref: "#/definitions/Persons"
      - $ref: "#/definitions/Paging"
```

上面这个例子表示，*PagedPersons* 根节点下，具有将 *Persons* 和 *Paging* **展开** 后的全部属性。

`allOf`同样可以使用行内的数据定义，比如：

```yaml
  PagedCollectingItems:
    allOf:
      - properties:
          items:
            type: array
            minItems: 10
            maxItems: 100
            uniqueItems: true
            items:
              $ref: "#/definitions/CollectingItem"
      - $ref: "#/definitions/Paging"
```

4.2.3 数据模型的继承（TODO）

目前各工具支持程度不高，待续

第5章 输入输出模型

这一章主要介绍如何定义高度精确化的参数和响应消息等。

5.1 高级参数定义

5.1.1 必带参数和可选参数

我们已经知道使用关键字`required`来定义一个必带参数。

5.1.1.1 定义必带参数和可选参数

在一个参数中，`required`是一个 `boolean` 型的可选值。它的默认值是 *false* 。

比如在某个操作中，*username* 是必填参数：

```yaml
  username:
    name: username
    in: path
#START########################################################################
##
    required: true
# END
############################################################################
    description: The person's username
    type: string
```

5.1.1.2 定义必带属性和可选属性

根据定义，`required`是一个字符串列表，列表中包含各必带参数名。如果某个参数在这张列表中找不到，那就说明它不是必带参数。如果没有定义`required`，就说明所有参数都是可选。如果`required`定义在一个HTTP请求上，这说明所有的请求参数都是必填。

在 *POST*、*persons* 中有 *Person* 的定义，在这里 *username* 这个属性是必带的，我们可以指定它为`required`，其他非必带字段则不指定：

```yaml
 Person:
#START########################################################################
##
```

```
    required:
      - username
# END
############################################################################
    properties:
      firstName:
        type: string
      lastName:
        type: string
      username:
        type: string
        pattern: '[a-z0-9]{8,64}'
        minLength: 8
        maxLength: 64
      dateOfBirth:
        type: string
        format: date
      lastTimeOnline:
        type: string
        format: date-time
        readOnly: true
      avatarBase64PNG:
        type: string
        format: byte
        default: data:image/png;base64,i……
      spokenLanguages:
        $ref: '#/definitions/SpokenLanguages'
```

5.1.2 带默认值的参数

通过关键字`default`，我们可以定义一个参数的默认值。当这个参数不可得（请求未带或者服务器未返回）时，这个参数就取默认值。因此设定了某个参数的默认值后，它是否`required`就没意义了。

5.1.2.1 定义参数的默认值

我们定义参数 *pageSize* 的默认值为 *20* ，那么如果请求时没有填写 *pageSize* ，服务器也会默认返回 *20* 个元素。

```
  pageSize:
    name: pageSize
    in: query
    description: Number of persons returned
    type: integer
    format: int32
    minimum: 0
    exclusiveMinimum: true
    maximum: 100
    exclusiveMaximum: false
    multipleOf: 10
#START################################################################################
##
    default: 20
# END
############################################################################
```

5.1.2.2 定义属性的默认值

我们在定义 *Person* 对象时，希望给每个用户一个默认头像，也就是要给 *avatarBase64PNG* 属性一个默认值。

默认头像：

```
  Person:
    required:
      - username
    properties:
      firstName:
        type: string
      lastName:
        type: string
      username:
        type: string
        pattern: '[a-z0-9]{8,64}'
        minLength: 8
        maxLength: 64
      dateOfBirth:
        type: string
        format: date
      lastTimeOnline:
        type: string
        format: date-time
        readOnly: true
      avatarBase64PNG:
        type: string
        format: byte
#START##########################################################################
##
        default: data:image/png;base64,iVBORw0KGgoAAAANSUhEUgA......rkJggg==
# END
############################################################################
      spokenLanguages:
        $ref: '#/definitions/SpokenLanguages'
```

5.1.3 带空值的参数

在 *GET /persons* 时，如果我们想添加一个参数来过滤“是否通过实名认证”的用户，应该怎么做呢？首先想到的是这样：*GET /persons?page=2&includeVerifiedUsers=true* ，问题是 *includeVerifiedUsers* 语义已经如此清晰，而让 “=true”显得很多余。我们能不能直接用：*GET /persons?page=2&includeVerifiedUsers* 呢？

要做到这种写法，我们需要一个关键字 `allowEmptyValue`。我们定义 *includeVerifiedUsers* 时允许它为空。那么如果我们请求 *GET /persons?page=2&includeVerifiedUsers* 则表示需要过滤“实名认证”用户，如果我们直接请求 *GET /persons?page=2* 则表示不过滤：

```
  includeNonVerifiedUsers:
    name: includeNonVerifiedUsers
    in: query
    type: boolean
    default: false
#START############################################################################
##
    allowEmptyValue: true
# END
############################################################################
```

5.1.4 参数组

设计API的时候，我们经常会遇到在 *GET* 请求中需要携带一组请求参数的情况。如何在API文档章呈现呢？很简单，我们只需要设定 `参数类型（type）` 为 `array`，并选择合适的 `组合格式（collectionFormat）` 就行了。

COLLECTIONFORMAT	描述	
csv (default value)	Comma separated values（逗号分隔） `foo,bar`	
ssv	Space separated values（空格分隔） `foo bar`	
tsv	Tab separated values（反斜杠分隔） `foo\tbar`	
pipes	Pipes separated values（竖线分隔） `foo\	bar`
multi	单属性可以取多个值，比如 `foo=bar&foo=baz`. 只适用于查询参数和表单参数。	

比如我们想根据多种参数（*username* ，*firstname* ，*lastname* ，*lastTimeOnline* ）等来对 *Person* 进行带排序的查询。我们需要一个这样的API请求： *GET /persons?sort=-lastTimeOnline|+firtname|+lastname* 。用于排序的参数是 *sort* ，`+` 表示升序，`-` 表示降序。

相应的API文档，可以这样写：

```
  sortPersons:
    name: sort
    in: query
    type: array
    uniqueItems: true
    minItems: 1
    maxItems: 3
    collectionFormat: pipes
    items:
      type: string
      pattern: '[-+](username|lastTimeOnline|firstname|lastname)'
```

现在我们就能搞定 *GET /persons?sort=-lastTimeOnline|+firtname|+lastname* 这种请求了。当然，我们还可以指定排序的默认值，锦上添花。

```
  sortPersons:
    name: sort
    in: query
    type: array
```

```
    uniqueItems: true
    minItems: 1
    maxItems: 3
    collectionFormat: pipes
    items:
      type: string
      pattern: '[-+](username|lastTimeOnline|firstname|lastname)'
#START################################################################################
##
    default:
      - -lastTimeOnline
      - +username
# END
############################################################################
```

5.1.5 消息头（Header）参数

参数，按照位置来分，不仅仅包含路径参数、请求参数和消息体参数，还包括消息头参数和表单参数等。比如我们可以在HTTP请求的消息头上加一个 *User-Agent*（用于跟踪、调试或者其他），可以这样定义它：

```
  userAgent:
    name: User-Agent
    type: string
    in: header
    required: true
```

然后像使用其他参数一样使用它：

```
paths:
  /persons:
    parameters:
      - $ref: '#/parameters/userAgent'
```

5.1.6 表单参数

有些 *js-less-browser* 的老浏览器不支持 *POST* JSON数据，比如在创建用户时，只能够以这样个格式请求：

```
POST /js-less-persons

username=apihandyman&firstname=API&lastname=Handyman
```

没有问题，丝袜哥可以搞定。我们只需要把各个属性的 `in` 关键字定义为 `formData`，然后设置 `consumes` 的媒体类型为 `application/x-www-form-urlencoded` 即可。

```
    post:
      summary: Creates a person
      description: For JS-less partners
#START################################################################################
##
      consumes:
        - application/x-www-form-urlencoded
# END
############################################################################
```

```
      produces:
        - text/html
      parameters:
        - name: username
#START###########################################################################
##
          in: formData
# END
#############################################################################
          required: true
          pattern: '[a-z0-9]{8,64}'
          minLength: 8
          maxLength: 64
          type: string
        - name: firstname
#START###########################################################################
##
          in: formData
# END
#############################################################################
          type: string
        - name: lastname
          in: formData
          type: string
        - name: dateOfBirth
#START###########################################################################
##
          in: formData
# END
#############################################################################
          type: string
          format: date
      responses:
        '204':
          description: Person succesfully created.
```

5.1.7 文件参数

当我们要处理一个请求，输入类型是 **文件** 时，我们需要：

- 使用 *multipart/form-data* 媒体类型；
- 设置参数的 `in` 关键字为 *formData*；
- 设置参数的 `类型（type）` 为 *file*。

比如：

```
  /images:
    parameters:
      - $ref: '#/parameters/userAgent'
    post:
      summary: Uploads an image
      consumes:
        - multipart/form-data
      parameters:
        - name: image
          in: formData
          type: file
```

```
      responses:
        '200':
          description: Image's ID
          schema:
            properties:
              imageId:
                type: string
```

有时候我们想限定输入文件的类型（后缀），很不幸的是，根据现在V2.0的规范暂时还做不到☹

> The spec doesn't allow specifying a content type for specific form data parameters. It's a limitation of the spec. *Ron Ratovsky comment in Swagger UI 609 issue*

5.1.8 参数的媒体类型

一个API可以消费各种不同的媒体类型，比如说最常见的是 *application/json* 类型的数据，当然这不是API唯一支持的类型。我们可以在**文档的根节点**或者一个**操作的根节点**下添加关键字 `consumes`，来定义这个操作能够消费的媒体类型。

比如我们的API全部都接受JSON和YAML的数据，那我们可以在文档的根节点下添加：

```
consumes:
  - application/json
  - application/x-yaml
```

如果某个操作（比如上传图片的操作）很特殊，它可以通过自己添加 `consumes` 来覆盖全局设置：

```
  /images:
    parameters:
      - $ref: '#/parameters/userAgent'
    post:
      summary: Uploads an image
#START###########################################################################
##
      consumes:
        - multipart/form-data
# END
#############################################################################
      parameters:
        - name: image
          in: formData
          type: file
      responses:
        '200':
          description: Image's ID
          schema:
            properties:
              imageId:
                type: string
```

5.2 高级响应消息定义

5.2.1 不带消息体的响应消息

不带消息体的响应很常见，比如HTTP *204* 状态响应本身就表示服务器返回不带任何消息内容的成功消息。

要定义一个不带消息体的响应很简单，我们只需要写响应状态和描述就行了：

```
    post:
      summary: Creates a person
      description: Adds a new person to the persons list.
      parameters:
        - name: person
          in: body
          required: true
          description: The person to create.
          schema:
            $ref: '#/definitions/Person'
      responses:
#START############################################################################
##
        '204':
          description: Person succesfully created.
# END
############################################################################
```

5.2.2 响应消息中的必带参数和可选参数

与请求消息中类似，我们使用`required`参数来表示，比如请求一个用户信息时， 服务器必须返回*username*，可以这样写：

```
  Person:
#START############################################################################
##
    required:
      - username
# END
############################################################################
    properties:
      firstName:
        type: string
      lastName:
        type: string
      username:
        type: string
        pattern: '[a-z0-9]{8,64}'
        minLength: 8
        maxLength: 64
      dateOfBirth:
        type: string
        format: date
      lastTimeOnline:
        type: string
        format: date-time
        readOnly: true
```

5.2.3 响应消息头

API的返回结果不仅仅体现下HTTP状态和响应消息体，还可以在响应消息头上做文章。比如我们可以限定一个API的使用次数和使用时间段，在响应消息头中，增加一个属性*X-Rate-Limit-Remaining* 来表示API可调用的剩余次数，增加另一个属性 *X-Rate-Limit-Reset* 来表示API的有效截止时间。

```
    post:
      summary: Creates a person
      description: Adds a new person to the persons list.
      parameters:
        - name: person
          in: body
          required: true
          description: The person to create.
          schema:
            $ref: '#/definitions/Person'
      responses:
        '204':
          description: Person succesfully created.
          headers:
#START##########################################################################
##
            X-Rate-Limit-Remaining:
              type: integer
            X-Rate-Limit-Reset:
              type: string
              format: date-time
# END
############################################################################
```

美中不足的是，对于这种响应消息头的修改，目前2.0规范暂时还不支持“一次定义、处处使用”☹

5.2.4 默认响应消息

我们在定义响应消息时，通常会列举不同的HTTP状态结果。如果有些状态不在我们API文档的定义范围（比如服务器需要返回 *993* 的状态），该怎么处理呢？这时需要通过关键字`default`来定义一个默认响应消息，用于各种**定义之外**的状态响应，比如：

```
    delete:
      summary: Deletes a person
      description: Delete a single person identified via its username
      responses:
        '204':
          description: Person successfully deleted.
          headers:
            X-Rate-Limit-Remaining:
              type: integer
            X-Rate-Limit-Reset:
              type: string
              format: date-time
        '404':
          $ref: '#/responses/PersonDoesNotExistResponse'
        '500':
          $ref: '#/responses/Standard500ErrorResponse'
#START##########################################################################
##
        default:
          $ref: '#/responses/TotallyUnexpectedResponse'
# END
############################################################################
```

目前这个配置也不支持“一次定义，处处使用”。☹

5.2.5 响应消息的媒体类型

与请求消息一样，我们也可以定义响应消息所支持的媒体类型，不同的是我们要用到关键字 `produces`（与请求消息中的 `consumes` 相对，由此可见，API文档描述的主体是**服务提供者**）。

比如，我们可以在文档的根路径下全局设置：

```
produces:
  - application/json
  - application/x-yaml
```

也可以在某个操作的根路径下覆盖设置：

```
  /images/{imageId}:
    parameters:
      - $ref: '#/parameters/userAgent'
    get:
      summary: Gets an image
      parameters:
        - name: imageId
          in: path
          required: true
          type: string
#START###############################################################################
##
      produces:
        - image/png
        - image/gif
        - image/jpeg
        - application/json
        - application/x-yaml
# END
############################################################################
      responses:
        '200':
          description: The image
          headers:
            X-Rate-Limit-Remaining:
              type: integer
            X-Rate-Limit-Reset:
              type: string
              format: date-time
        '404':
          description: Image do not exists
          headers:
            X-Rate-Limit-Remaining:
              type: integer
            X-Rate-Limit-Reset:
              type: string
              format: date-time
        '500':
          $ref: '#/responses/Standard500ErrorResponse'
        default:
          $ref: '#/responses/TotallyUnexpectedResponse'
```

5.3 定义某个参数只存在于响应消息中

如前章节4.2.1中已经提到的，定义一个对象，其中某个属性我们只希望在响应消息中携带，而不希望在请求消息中携带，应该用`readOnly`关键字来表示。考虑到内容的完整性，这里再介绍一下。

比如 *Person* 对象中的 *lastTimeOnline* 这个属性，注册用户时我们不需要填写，但是在获取用户信息时，需要提供给服务消费者：

```
  Person:
    required:
      - username
    properties:
      firstName:
        type: string
      lastName:
        type: string
      username:
        type: string
        pattern: '[a-z0-9]{8,64}'
        minLength: 8
        maxLength: 64
      dateOfBirth:
        type: string
        format: date
      lastTimeOnline:
        type: string
        format: date-time
#START###########################################################################
##
        readOnly: true
# END
#############################################################################
```

第6章 不要让API裸奔

这一章主要介绍API文档中如何描述安全相关的内容。

6.1 定义安全

安全相关内容的定义一般放在API文档根目录下的`securityDefinition`中，它包括一组具体的命名安全项，每一个命名安全定义可能包括下面三种安全类型之一：*basic*，*apiKey* ，*oauth2*。

6.1.1 基础鉴权（Basic Authentication）

要定义一个基础（*basic*）鉴权，我们只需要将`type`设置为 *basic* 即可：

```
securityDefinitions:
  UserSecurity:
    type: basic
  AdminSecurity:
    type: basic
  MediaSecurity:
    type: basic
```

这个例子中，我们定义了三种安全说明（*UserSecurity*，*AdminSecurity*， *MediaSecurity*），都属于基础鉴权。

6.1.2 API秘钥鉴权（API Key）

要定义一个API秘钥鉴权，我们需要：

- 设置`type`为 *apiKey*
- 通过关键字`in`指示api秘钥所在位置。通常api秘钥会放在消息头、请求参数或者消息体中
- 给安全项命个名字

```
securityDefinitions:
  UserSecurity:
    type: apiKey
    in: header
    name: SIMPLE-API-KEY
  AdminSecurity:
    type: apiKey
    in: header
    name: ADMIN-API-KEY
  MediaSecurity:
    type: apiKey
    in: query
    name: MEDIA-API-KEY
```

在这个例子中,我们定义了三个`apiKey`类型的安全项：

- *UserSecurity* 定义了一个名为*SIMPLE-API-KEY* 的参数在消息头（header）
- *AdminSecurity* 定义了一个名为 *ADMIN-API-KEY* 的参数在消息头 （header）
- *MediaSecurity* 定义了一个名为*MEDIA-API-KEY*的参数在请求参数中

6.1.3 Oauth2鉴权

6.1.3.1 流程（Flow）和URL

当我们定义个 *Oauth2* 类型的安全项上，我们通常会定义*Oauth2* 的流程（flow）和并根据选定的流程配置相应的`鉴权地址（authorizationUrl）`和/或`令牌地址（tokenUrl）`。

流程	所需要的URL
implicit	authorizationUrl（鉴权地址）
password	tokenUrl（令牌地址）
application	tokenUrl
accessCode	authorizationUrl and tokenUrl

比如：

```
securityDefinitions:
  OauthSecurity:
    type: oauth2
    flow: accessCode
    authorizationUrl: 'https://oauth.simple.api/authorization'
    tokenUrl: 'https://oauth.simple.api/token'
```

在这个例子中，我们定义了一个 *Oauth2* 的安全项，配置的流程是 *accessCode* 方式，同时配置了鉴权地址和令牌地址。

6.1.3.2 作用范围（scope）

我们借助关键字`scopes`并通过哈希键值对来还可以配置 *Oauth2* 安全项的作用范围（scope），键值对的**键**表示作用范围名称；**值**是它的相关描述，比如：

```
securityDefinitions:
  OauthSecurity:
    type: oauth2
    flow: accessCode
    authorizationUrl: 'https://oauth.simple.api/authorization'
    tokenUrl: 'https://oauth.simple.api/token'
    scopes:
      admin: Admin scope
      user: User scope
      media: Media scope
```

在这个例子中，我们给 *OauthSecurity* 安全项添加了三个作用范围（*admin*，*user*，*media*）。

6.2 使用安全定义

现在我们已经在*securityDefinition* 中定义好了安全项，现在我们可以将将它们应用到文档中了。使用的时候，我们通过`security`关键字，把安全项添加进去。

6.2.1 基础鉴权

6.2.1.1 API级别

```
securityDefinitions:
  UserSecurity:
    type: basic
  AdminSecurity:
    type: basic
  MediaSecurity:
    type: basic
#START############################################################################
##
security:
  - UserSecurity: []
# END
############################################################################
paths:
  /persons:
```

在这个例子中，我们在API文档的根路径下直接使用了安全项 *UserSecurity*，它的作用范围是整个API文档。

6.2.1.2 操作级别

比如我们在添加或者修改用户信息时，需要进行管理员鉴权，可以在 *POST /persons* 操作中增加安全项：

```
    post:
      summary: Creates a person
      description: Adds a new person to the persons list.
#START##########################################################################
##
      security:
        - AdminSecurity: []
# END
############################################################################
```

而在上传图片时，需要进行媒体操作鉴权，可以在 *POST /images* 操作中增加安全项：

```
  /images:
    parameters:
      - $ref: '#/parameters/userAgent'
    post:
      summary: Uploads an image
#START##########################################################################
##
      security:
        - MediaSecurity: []
# END
############################################################################
```

6.2.2 API秘钥鉴权

使用方式和基础鉴权一样，可以在API级别和操作级别使用：

```
securityDefinitions:
  UserSecurity:
    type: apiKey
    in: header
    name: SIMPLE-API-KEY
  AdminSecurity:
    type: apiKey
    in: header
    name: ADMIN-API-KEY
  MediaSecurity:
    type: apiKey
    in: query
    name: media-api-key
#START##########################################################################
##
security:
  - UserSecurity: []
# END
############################################################################
paths:
  /persons:
    post:
      summary: Creates a person
      description: Adds a new person to the persons list.
      security:
        - AdminSecurity: []
```

6.2.3 Oauth2鉴权

Oauth2 鉴权的使用和上面的两种鉴权方式基本相同，不同之处在于我们可以指定它的哪一个作用范围（scope）。

比如API级别的鉴权：

```
securityDefinitions:
  OauthSecurity:
    type: oauth2
    flow: accessCode
    authorizationUrl: 'https://oauth.simple.api/authorization'
    tokenUrl: 'https://oauth.simple.api/token'
    scopes:
      admin: Admin scope
      user: User scope
      media: Media scope
#START########################################################################
##
security:
  - OauthSecurity:
    - user
# END
############################################################################
paths:
  /persons:
```

操作级别的鉴权：

```
    post:
      summary: Creates a person
      description: Adds a new person to the persons list.
      security:
        - OauthSecurity:
          - admin
```

在这个例子中，作用范围 *admin* 将覆盖全局配置的作用范围 *user*。

6.3 使用多种安全配置

OpenAPI规范并没有限定我们只能使用一种安全项。下面的例子将展示如何使用多种安全配置。

6.3.1 安全定义

```
securityDefinitions:
  OauthSecurity:
    type: oauth2
    flow: accessCode
    authorizationUrl: 'https://oauth.simple.api/authorization'
    tokenUrl: 'https://oauth.simple.api/token'
    scopes:
      admin: Admin scope
      user: User scope
  MediaSecurity:
    type: apiKey
    in: query
```

```
    name: media-api-key
  LegacySecurity:
    type: basic
```

这个例子中，我们定义了三种鉴权方式。

6.3.2 全局安全配置

```
security:
  - OauthSecurity:
    - user
  - LegacySecurity: []
```

这个配置的意思是用户可以通过两种方式中的**任意一种**来访问我们提供的API接口。

6.3.3 覆盖全局配置

```
    post:
      summary: Creates a person
      description: Adds a new person to the persons list.
      security:
        - OauthSecurity:
          - admin
        - LegacySecurity: []
```

在 *POST /persons* 操作中，*OauthSecurity* 的作用范围被覆写为*admin*。此时用户可以通过*admin* 的 *Oauth2* **或者** *legacySecurity* 来鉴权使用这个操作。

```
  /images:
    parameters:
      - $ref: '#/parameters/userAgent'
    post:
      summary: Uploads an image
      security:
        - MediaSecurity: []
```

在 *POST/images* 操作中，用*MediaSecurity* 整体覆写了全局安全项，用户只能通过 *MediaSecurity* 鉴权使用这个操作。

第7章 让文档的可读性更好

7.1 分类标签（Tags）

通过关键字`tags`我们可以对文档中接口进行归类，`tags`的本质是一个字符串列表。`tags`定义在文档的根路径下。

7.1.1 单标签

比如说 *GET /persons* 属于**用户（Person）**这个分类的，那么我们可以给它贴个标签：

```yaml
paths:
  /persons:
    parameters:
      - $ref: '#/parameters/userAgent'
    get:
      summary: Gets some persons
      description: Returns a list containing all persons. The list supports
paging.
      operationId: searchUsers
#START############################################################################
##
      tags:
        - Persons
# END
############################################################################
```

7.1.2 多标签

一个操作也可以同时贴几个标签，比如：

```yaml
  /js-less-consumer-persons:
    parameters:
      - $ref: '#/parameters/userAgent'
    post:
      summary: Creates a person
      description: For JS-less partners
      operationId: createUserJS
      deprecated: true
      tags:
        - JSLess
        - Persons
```

贴上标签后，在Swagger Editor和Swagger UI中能够自动归类，我们可以按照标签来筛选接口，试试吧？

7.2 无处不在的描述文字（Descriptions）

`description`这个属性几乎是无处不在，为了提高文档的可读性，我们应该在必要的地方都加上描述文字。

7.2.1 安全项的描述

```yaml
securityDefinitions:
  OauthSecurity:
    description: New Oauth security system. Do not use MediaSecurity or
LegacySecurity.
    type: oauth2
    flow: accessCode
    authorizationUrl: 'https://oauth.simple.api/authorization'
    tokenUrl: 'https://oauth.simple.api/token'
    scopes:
      admin: Admin scope
      user: User scope
  MediaSecurity:
    description: Specific media security for backward compatibility. Use
OauthSecurity instead.
```

```
      type: apiKey
      in: query
      name: media-api-key
    LegacySecurity:
      description: Legacy security system for backward compatibility. Use
OauthSecurity instead.
      type: basic
```

7.2.2 模式（Schema）的描述

每一种模式（Schema），都会有一个标题（title）和一段描述，比如：

```
definitions:
  Person:
    title: Human
    description: A person which can be the user itself or one of his friend
```

7.2.3 属性的描述

比如：

```
    properties:
      firstName:
        description: first name
        type: string
```

7.2.4 参数的描述

```
paths:
  /persons:
    post:
      parameters:
        - name: person
          in: body
          required: true
          description: The person to create.
          schema:
            $ref: '#/definitions/Person'
parameters:
  username:
    name: username
    in: path
    required: true
    description: The person's
```