RAML 规范研究

1 RAML 是什么

RAML（RESTful API Modeling Language 即 RESTful API 建模语言）是对 RESTful API 的一种简单和直接的描述。它是一种让人们易于阅读并且能让机器对特定的文档能解析的语言。RAML 是基于 YAML，符合 1.2 版本规范，能帮助设计 RESTful API 和鼓励对 API 的发掘和重用，依靠标准和最佳实践从而编写更高质量的 API。通过 RAML 定义，因为机器能够看得懂，所以可以衍生出一些附加的功能服务，像是解析并自动生成对应的客户端调用代码、服务端代码结构， API 说明文档。

我们知道 Web Service 有相应的 WSDL 来描述它相应的 Schema，WSDL 就相当于对当前的服务做了一个描述，Client 端可以据此生成相应的 Proxy 代码, 因此 WSDL 可以帮助 Client 更容易的消费服务。对于 RESTful API，却没有相应的“REST WSDL”，在这种情况下，RAML 应运而生，它可以对我们的 API 做完整的描述，不管是对人或者是机器，都能以相对友好的方式使用。

shiyuanstone 2014 年 10 月 8 日

2 RAML 规范

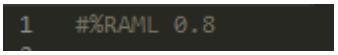
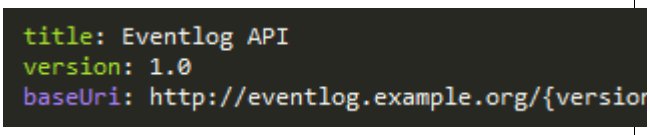
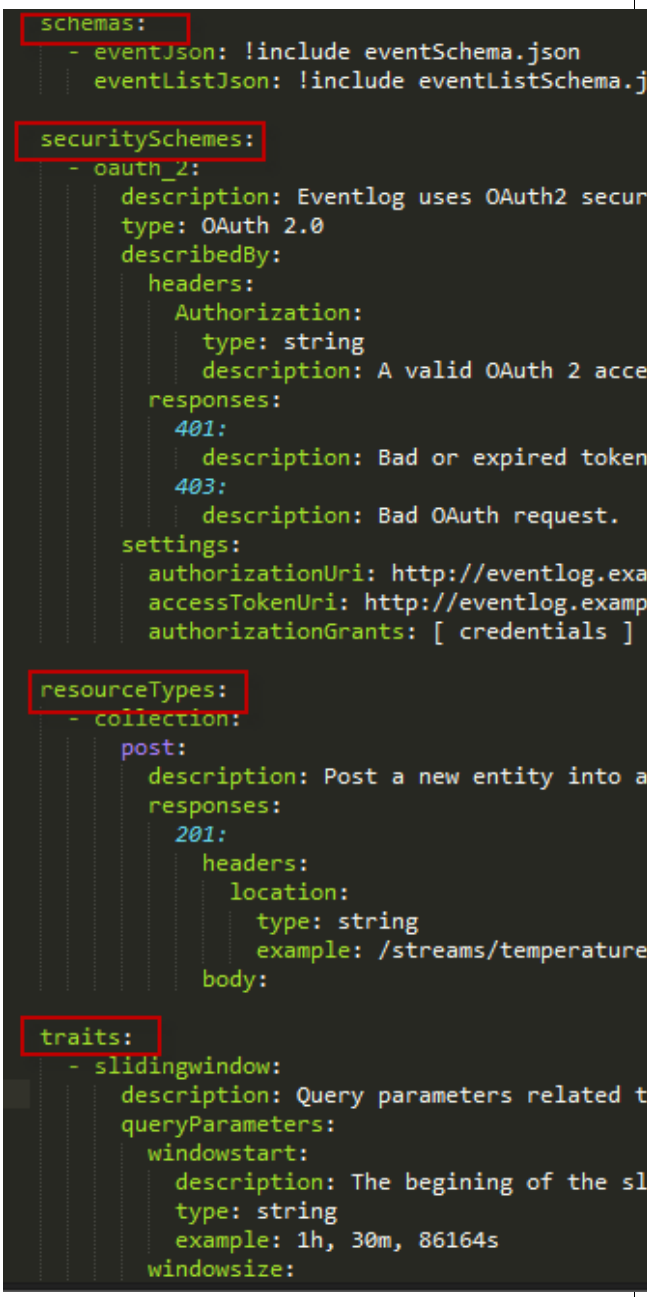
2.1 RAML 文件构成

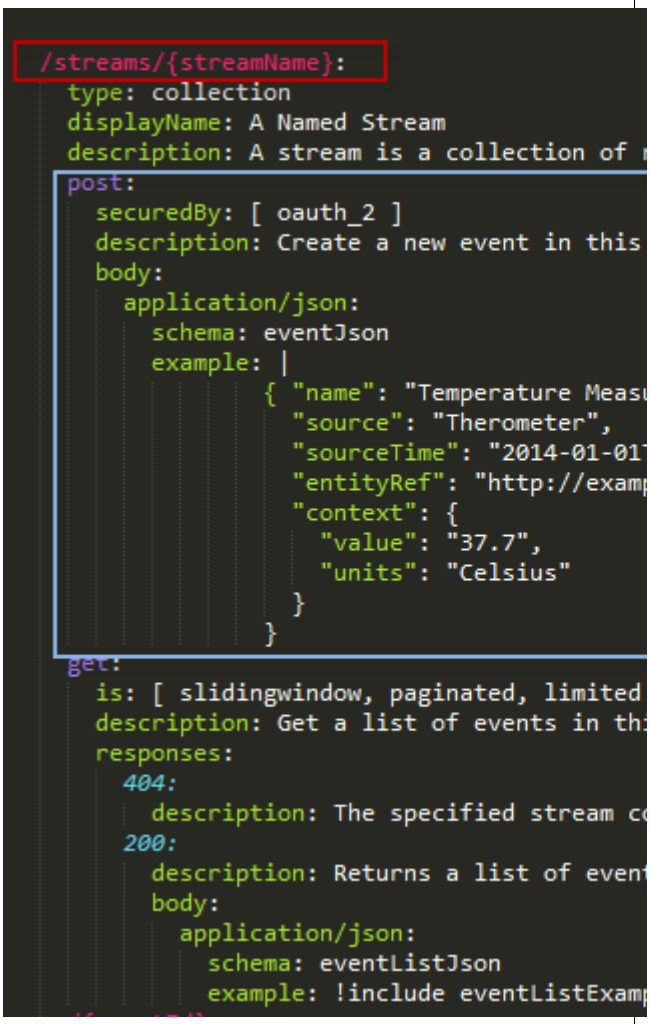
```
#%RAML 0.8

title: World Music API
baseUri: http://example.api.com/{version}
version: v1
traits:
  - paged:
      queryParameters:
        pages:
          description: The number of pages to return
          type: number
  - secured: !include http://raml-example.com/secured.yml
/songs:
  is: [ paged, secured ]
  get:
    queryParameters:
      genre:
        description: filter the songs by genre
  post:
  /{songId}:
    get:
      responses:
        200:
          body:
            application/json:
              schema: |
                { "$schema": "http://json-schema.org/schema",
                  "type": "object",
                  "description": "A canonical song",
                  "properties": {
                    "title":  { "type": "string" },
                    "artist": { "type": "string" }
                  },
                  "required": [ "title", "artist" ]
                }
            application/xml:
    delete:
      description: |
        This method will *delete* an **individual song**
```

上图是一个 RAML 一个简单的例子。下面介绍一下 RAML 的基本语法。完成一个 RAML 具体步骤参见：http://raml.org/docs.html

整个 RAML 文档可简单划分为“版本声明”、“API 元数据定义”、“公用属性定义”和“资源方法定义”四部分构成。

元素	说明	示例

RAML 版本声明	每一个 RAML 文件在第一行都必须是 RAML 版本的声明	
API 元数据定义	整个 API 的标题、版本、API 基础 url、相关文件（documentation 属性）	
公用属性定义	Schemas：用于定义请求参数或者响应结构； securitySchemas：定义 api 访问的安全机制； ressourceTypes：定义该属性如同定义一个资源，可以对其进行描述（declaration），方法和属性的定义。在使用该资源类型的资源会继承其性质（例如方法）。 traits：定义该属性如同定义一个方法，可以包括方法级属性，包括描述（declaration），头，查询字符串参数，和响应。使用这些 traits 的方法会继承其中的属性。	

资源（resource）和方法（Action）定义		

```
/streams/{streamName}:
  type: collection
  displayName: A Named Stream
  description: A stream is a collection of
  post:
    securedBy: [ oauth_2 ]
    description: Create a new event in this
    body:
      application/json:
        schema: eventJson
        example: |
                 { "name": "Temperature Measu
                   "source": "Therometer",
                   "sourceTime": "2014-01-01
                   "entityRef": "http://examp
                   "context": {
                     "value": "37.7",
                     "units": "Celsius"
                   }
                 }
  get:
    is: [ slidingwindow, paginated, limited
    description: Get a list of events in thi
    responses:
      404:
        description: The specified stream co
      200:
        description: Returns a list of event
        body:
          application/json:
            schema: eventListJson
            example: !include eventListExamp
```

2.2 YAML 语法介绍

RAML 是基于 YAML，符合 1.2 版本规范，所以文件的格式说明直接参考 YAML 语法。

YAML 作为一种比 XML，或者 JSON 都更为简单易读的序列化语言，正越来越多地被人们所接受和喜欢，并应用于软件项目的开发中，比如：现实生活中的数据上程序中的序列化表示，以及系统中的配置文件的书写。

YAML 的数据组织主要依靠的是空白，缩进，分行等结构。这使得 YAML 语言很容易上手。

用“#”表示文档行注释的开始。如下图

```
#%RAML 0.8

# Example event logging API as described in InfoQ article in March 2014.
```

“>”的作用，以缩进对齐来判断是否为一段文字，也就是说，一旦缩进与上一行不一致，则认为是一个新行。 下面 node1 的例子中，第一行“Ther... door”， 第二行“ "Please... floor"”， 第三行“So...So2”。

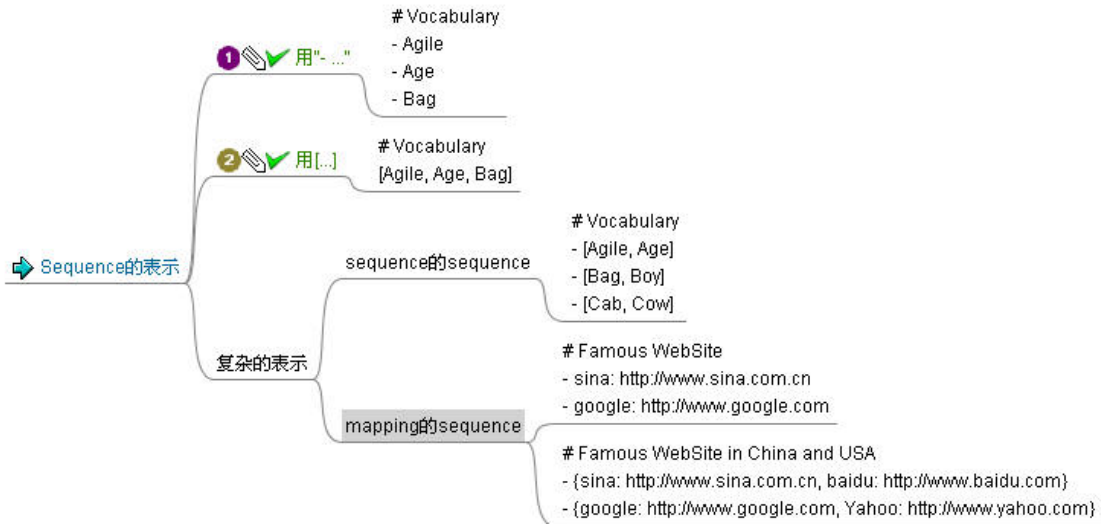
```
node1: >
  Ther once was a man from Darjeeling
  Who got on a bus bound for Ealing
  It said on the door
    "Please don't spit on the floor"
  So he carefully spat on the ceiling
  So2
```

用“|” 表示之后的文字，每一行均为一个新行。

```
/streams/{streamName}:
  type: collection
  displayName: A Named Stream
  description: A stream is a collection of related events. A Named Stream
  post:
    securedBy: [ oauth_2 ]
    description: Create a new event in this stream.
    body:
      application/json:
        schema: eventJson
        example: |
                { "name": "Temperature Measurement",
                  "source": "Therometer",
                  "sourceTime": "2014-01-01T16:56:54+11:00",
                  "entityRef": "http://example.org/thermometers/99981",
                  "context": {
                    "value": "37.7",
                    "units": "Celsius"
                  }
                }
```

YAML 的设计者认为在配置文件中所要表达的数据内容有三种类型：标量(Scalar，如字符串和整数等)、序列（Sequence，如数组）和 Mapping（类似 hash 的 key/value pair）。sequence 型主要是用来表示数组类型的数据。

下图描述了 YAML 中 Sequence 型数据的表示法



mapping 数据类型主要用来表示 key: value 对类型的数据。YAML 描述方式见下图：

用“-”来表示一些序列的项（Sequence），如下图里的产品（product）有两样东西（Basketball 和 Super Hoop）组织为一个序列；用“：”来表示一对项目（Map）里的栏目（Key）和其相应的值（Value），比如下图发票里的时间（date）的值是 2001-01-23，这就是一个 Map。

```
invoice: 34843
date : 2001-01-23
bill-to: &id001
 given : Chris
 family : Dumars
 address:
 lines: |
 458 Walkman Dr.
 Suite #292
 city : Royal Oak
 state : MI
 postal : 48046
ship-to: *id001
product:
 - sku : BL394D
 quantity : 4
 description : Basketball
 price : 450.00
 - sku : BL4438H
 quantity : 1
 description : Super Hoop
 price : 2392.00
tax : 251.42
total: 4443.52
comments: >
 Late afternoon is best.
 Backup contact is Nancy
 Billsmer @ 338-4338.
```

使用“!include”来引入外部的文件（包括 RAML，YAML 和普通的 txt 文件），这样就可以将一个很大的 RAML 分成很多个部分。注意：被引入的文件不能再引用其他文件，即被引用的文件中不能再包括“!include”属性。

这些就是 YAML 里最重要的语法了。如果想知道其他语法的细节可以参看 YAML 官方网页里的参考卡片（reference card）：http://www.yaml.org/refcard.html。

注意缩进：缩进只能是两个空格为一级，不能是其他字符。

2.3 RAML 属性定义

2.3.1 API 元数据定义

元素	说明	示例

title	必填。简要描述 api	`#%RAML 0.8`
		`title: Salesforce Chatter REST API`
		`version: v28.0`
		`protocols: [ HTTP, HTTPS ]`
		`baseUri: https://na1.salesforce.com`

version	版本	
baseUri	API 访问的基础路径。	

	REST API 的所有资源定义都是相基础访问路径的。	
protocols	当前 API 所支持的访问协议，如果没有定义该属性值，则默认认为仅支持 baseUri 所采用的协议。	
mediaType	API 响应的默认媒体类型	`mediaType: application/json`

documentation	API 定义中可以包含一系列文档用于指导用户如何使用。	两种实现方式
		1：直接在 RAML 文档中定义
		`documentation:`
		`- title: Home`
		`content: |`
		`Welcome to the _Zencoder API_ Documentati`
		`allows you to connect your application to`
		`and encode videos without going through t`
		`may also benefit from one of our`
		`[integration libraries](https://app.zenco`
		`for different languages.`
		2：使用“!include”属性引入外部文件
		`documentation:`
		`- title: Getting Started`
		`content: !include github-3-getting-started.md`
		`- title: Basics of Authentication`
		`content: !include github-3-basics-of-authentication.md`

2.3.2 元属性

RAML 规范中，用于描述像 URI 参数、查询参数（queryParameters）、表单参数（formParameters）、请求体（body）、请求和响应的头等集合属性的属性，我们称之为元属性。

元素名称	是否必须	说明

displayName	可选参数	指定参数的显示名称，如果没有定义，则显示属性的名称。
description	可选参数	对指定的属性进行描述，例如对属性的用法、含义等进行描述
type	可选参数	默认为 string 类型
enum	可选参数	仅适用于 string 类型参数。提供参数有效值得枚举，必须为一个数组。如果定义了枚举类型，则 API 客户端和服务器必须对定义了该参数的属性进行验证，如果没有匹配值，则会报错。
pattern	可选参数	仅适用于 string 类型参数。该参数为正则表达式，string 类型参数必须匹配
minLength	可选参数	仅适用与 string 类型参数，定义参数的最小长度
maxLength	可选参数	仅适用于 String 类型参数，定义参数的最大长度
minimum	可选参数	仅适用于 number 和 integer 类型参数，定义参数的最小值
maximum	可选参数	仅适用于 number 和 integer 类型参数，定义参数的最大值
example	可选参数	定义示例。
repeat	可选参数	表明参数可以重复，如果这个参数可以被多次使用，那么 repeat 的值为‘true’，否则则为‘false’。
required	可选参数	（除非有特别说明），用来表明该参数是否必需，如果是则为‘true’，否则为‘false’。
default	可选参数	用来定义参数的默认值。

举例如下图中查询参数“pagenumber”和“pagesize”的定义。

```
queryParameters:
  pagenumber:
    description: The page number of the result-set to return.
    type: integer
    minimum: 0
  pagesize:
    description: The number of rows in a page request.
    type: integer
    maximum: 100
```

2.3.3 资源定义

所有的资源标示符都是相对路径，必须以“/”开头。资源定义分为顶级资源和嵌套资源，嵌套资源即在已有资源定义内再定义资源。顶级资源的 url 路径相对于 baseUrl 属性。

```
#%RAML 0.8
title: GitHub API
version: v3
baseUri: https://api.github.com
/gists:
  displayName: Gists
  description: A collection of gist
  /public:
    displayName: Public Gists
```

定义资源的相对路径可以使用含有参数的模板路径。下面的例子为定义一个顶级资源“/jobs”和一个含有变量的嵌套资源“/{jobId}”。

```
#%RAML 0.8
title: ZEncoder API
version: v2
baseUri: https://app.zencoder.com/api/{version}
/jobs: # its fully-resolved URI is https://app.zencoder.com/api/{version}/jobs
  displayName: Jobs
  description: A collection of jobs
  /{jobId}: # its fully-resolved URI is https://app.zencoder.com/api/{version}/jobs/{jobId}
    description: A specific job, a member of the jobs collection
```

可以使用“uriParameters”属性对资源标识符中的变量进行详细定义。

```
#%RAML 0.8
title: GitHub API
version: v3
baseUri: https://api.github.com
/user:
  displayName: Authenticated User
/users:
  displayName: Users
  /{userId}:
    displayName: User
    uriParameters:
      userId:
        displayName: User ID
        type: integer
```

2.3.4 方法定义

在 RESTful API 中，所有的方法操作都是针对一个资源的。方法的名称必须是 HTTP 1.1 规

规范中定义的 method 中的一种。比较常用方法的有 GET,, POST, PUT, DELETE。

下图为查询和修改一本书的资源方法定义示例。其中 get 方法的响应体和 put 方法的请求体为 json 字符串，通过“body”属性定义。“body”属性作为一个 hashmap，其中必须包括媒体类型作为 key，如“application/json”、“text/xml”。

```
/{bookId}:
  displayName: Book
  uriParameters:
    bookId:
      description: |
        Book Identifier
      required: true
  get:
    description: Retrieve book-related information.
    responses:
      200:
        body:
          application/json:
            example:  |
              {
                "bookId": 1,
                "title": "Angel & Demons",
                "isbn": "0-671-02735-2",
                "authors": [
                  "Dan Brown"
                ]
              }
  put:
    description: Update book information.
    body:
       application/json:
        example: |
          {
           "title": "Inferno",
           "isbn": "978-0-385-53785-8",
           "authors": [
            "Dan Brown"
           ]
          }
    responses:
      204:
        description: |
          The book has been successfully updated.
      404:
        description: |
          Unable to find book with that identifier.
```

form 表单参数

```
#%RAML 0.8
title: Amazon simple storage API
version: 1
baseUri: https://{destinationBucket}.s3.amazonaws.com
/:
  post:
    description: The POST operation adds an object to a specified bucket using HTML forms.
    body:
      application/x-www-form-urlencoded:
        formParameters:
          AWSAccessKeyId:
            description: The AWS Access Key ID of the owner of the bucket who grants an Anonymous user access for a
            type: string
          acl:
            description: Specifies an Amazon S3 access control list. If an invalid access control list is specified,
            type: string
          file:
            - type: string
              description: Text content. The text content must be the last field in the form.
            - type: file
              description: File to upload. The file must be the last field in the form.
```

查询参数

```
#%RAML 0.8
title: GitHub API
version: v3
baseUri: https://api.github.com
/users:
  get:
    description: Get a list of users
    queryParameters:
      page:
        type: integer
      per_page:
        type: integer
```

2.3.5 重用属性定义

资源和方法的定义存在重复性，比如一个 API 需要 OAuth 认证，这个 API 的所有资源和方法的定义必须包含“*access_token*”查询参数。针对这种场景可以定义重用属性，让其他资源或方法通过继承的方式来获取属性的设置。

2.3.5.1资源类型 resourceTypes

定义资源类型如同定义一个资源，可以对其进行描述（declaration）、方法和属性的定义。定义资源时通过“type”属性定义所继承的资源类型，一个资源属于一个或零个资源类型。使用该资源类型的资源会继承其属性设置。

资源类型定义

```yaml
resourceTypes:
  - collection:
      post:
        description: Post a new entity into a collection.
        responses:
          201:
            headers:
              location:
                type: string
                example: /streams/temperatures/12345
            body:
```

资源类型使用

```yaml
/streams/{streamName}:
  type: collection
  displayName: A Named Stream
  description: A stream is a collection of related events. A Named Stream has been
  post:
    securedBy: [ oauth_2 ]
    description: Create a new event in this stream.
    body:
      application/json:
        schema: eventJson
        example: |
                { "name": "Temperature Measurement",
                  "source": "Therometer",
                  "sourceTime": "2014-01-01T16:56:54+11:00",
                  "entityRef": "http://example.org/thermometers/99981",
                  "context": {
                    "value": "37.7",
                    "units": "Celsius"
                  }
                }
```

2.3.5.2方法特征 traits

定义 trait 如同定义一个方法，可以包括方法级属性，包括描述（declaration），HTTP 头，查询字符串参数和响应的定义。定义资源时通过“is” 属性定义所使用的 trait，一个方法可以拥有零个或多个 trait。使用 trait 的方法会继承其中的属性设置。

traits 定义

```
traits:
  - slidingwindow:
      description: Query parameters related to retrieving a sliding window of timestamped entities
      queryParameters:
        windowstart:
          description: The begining of the sliding window expressed as a time interval from now rep
          type: string
          example: 1h, 30m, 86164s
        windowsize:
          description: The end of the sliding window expressed as a time interval from the start as
          type: string
          example: 10s, 1h, 25m
  - paginated:
      queryParameters:
        pagenumber:
          description: The page number of the result-set to return.
          type: integer
          minimum: 0
        pagesize:
          description: The number of rows in a page request.
          type: integer
          maximum: 100
  - limited:
      queryParameters:
        limit:
          description: A general limit on the number of rows to return in any request.
          type: integer
          default: 100
```

trait 使用

```
/streams/{streamName}:
  type: collection
  displayName: A Named Stream
  description: A stream is a collection of related events. A Named Stream has been defined
  get:
    is: [ slidingwindow, paginated, limited ]
    description: Get a list of events in this stream.
    responses:
      404:
        description: The specified stream could not be found.
      200:
        description: Returns a list of events.
        body:
          application/json:
            schema: eventListJson
            example: !include eventListExample.json
```

2.3.5.3结构模型 schemas

API 方法的入参或者返回结果如果为 json 或者 xsm 格式，通常需要“schema”属性来定义该格式类型对应的结构。“schema”属性值既可以是结构模型的真实定义，也可以是之前所定义结构模型的名称。

直接定义结构模型：

```
/jobs:
  displayName: Jobs
  post:
    description: Create a Job
    body:
      text/xml:
        schema: |
          <xs:schema attributeFormDefault="unqualified"
                     elementFormDefault="qualified"
                     xmlns:xs="http://www.w3.org/2001/XMLSchema">
            <xs:element name="api-request">
              <xs:complexType>
                <xs:sequence>
                  <xs:element type="xs:string" name="input"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:schema>
      application/json:
        schema: |
          {
            "$schema": "http://json-schema.org/draft-03/schema",
            "properties": {
                "input": {
                    "required": false,
                    "type": "string"
                }
            },
            "required": false,
            "type": "object"
          }
```

先定义结构模型，应用时通过“schema”属性设置所使用的结构模型的名称。

```
schemas:
  - eventJson: !include eventSchema.json #引入外部文件        schema定义
    eventListJson: !include eventListSchema.json

/streams/{streamName}:
  type: collection
  displayName: A Named Stream
  description: A stream is a collection of related events. A Named Stream has been defined by the user
  get:
    is: [ slidingwindow, paginated, limited ]
    description: Get a list of events in this stream.
    responses:
      404:
        description: The specified stream could not be found.
      200:
        description: Returns a list of events.
        body:
          application/json:                                  schema使用
            schema: eventListJson
            example: !include eventListExample.json
```

2.3.5.4安全认证模型 securitySchemes

假如一个 API 需要 OAuth 安全认证，这个 API 的所有资源和方法的定义必须包含“*access_token*”查询参数。针对这种场景可以定义重用属性，让其他资源或方

法通过继承的方式来获取属性的设置。

```
securitySchemes:
  - oauth_2:
      description: Eventlog uses OAuth2 security scheme only.
      type: OAuth 2.0
      describedBy:
        headers:
          Authorization:
            type: string
            description: A valid OAuth 2 access token.
        responses:
          401:
            description: Bad or expired token.
          403:
            description: Bad OAuth request.
      settings:
        authorizationUri: http://eventlog.example.org/oauth2/authorize
        accessTokenUri: http://eventlog.example.org/oauth2/token
        authorizationGrants: [ credentials ]
```

定义方法时通过“securedBy”是指定安全认证模型，“securedBy”属性值为集合类型。

```
/streams/{streamName}:
  type: collection
  displayName: A Named Stream
  description: A stream is a collection of related events. A Named Stream
  post:
    securedBy: [ oauth_2 ]
    description: Create a new event in this stream.
    body:
      application/json:
        schema: eventJson
        example: |
                { "name": "Temperature Measurement",
                  "source": "Therometer",
                  "sourceTime": "2014-01-01T16:56:54+11:00",
                  "entityRef": "http://example.org/thermometers/99981",
                  "context": {
                    "value": "37.7",
                    "units": "Celsius"
                  }
                }
```

3 RAML 支持工具

3.1 RAML 资源

官方网站：http://raml.org

规范文档：http://raml.org/spec.html

设计器：https://github.com/mulesoft/api-designer

shiyuanstone 2014 年 10 月 8 日

解析器：

JS 解析器：https://github.com/raml-org/raml-js-parser，生成浏览器可访问的 API 文档

java 解析器：https://github.com/raml-org/raml-java-parser 。基于该解析器我们可以开发出生成 API 文档，、API 调用客户端代码以及将 API 定义导入到服务中心的功能。