RAML规范

RAML全称是RESTful API Modeling Language（RESTful API建模语言），由MuleSoft公司成员主导设计。

RAML的本质上是一组API设计规范，截止到2017年3月13日，最新版本是1.0。它基于YAML 1.2语法规范，从它的名字上可以看出，RAML主要是在API的设计阶段使用，它能够对RESTful API进行建模，然后导出文档、HTML模板以及多种编程语言的SDK，对于前后端分离的架构很有帮助。

1. 文档头

RAML文档头部分描述了关于API的基本信息，还有类型和特征等辅助信息。

RAML API文档中的节点可以以任意顺序出现。而RAML的处理器在解析时必须保存节点顺序，如果包含数组，那么还要保存数组中元素的顺序。

下面GitHub v3公开API的一个RAML API定义的示例：

```
#%RAML 1.0
title: GitHub API
version: v3
baseUri: https://api.github.com
mediaType:  application/json
securitySchemes:
  oauth_2_0: !include securitySchemes/oauth_2_0.raml
types:
  Gist:  !include types/gist.raml
  Gists: !include types/gists.raml
resourceTypes:
  collection: !include types/collection.raml
traits:
securedBy: [ oauth_2_0 ]
/search:
  /code:
    type: collection
    get:
```

下表列出了RAML文档头中可以出现的节点：

节点名	说明

title	用于描述API的简短标题
description?	字符串类型的API的具体描述，可以使用 markdown[1] 格式。
version?	说明API的版本，如"v1"，字符串类型。
baseUri?	提供资源和服务的 根URIs[2]。可以是 模板URI[3].
baseUriParameters?	根Uri[4]（模板）使用的参数名 。
protocols?	此API支持的 协议[5]。
mediaType?	请求和响应报问题的 默认媒体类型[6]，如"application/json"。
documentation?	API的附加 文档[7]。
schemas?	在RAML 0.8中等价于"types"的别名，但新版本中不再推荐使用，因为"types"节点支持XML和JSON的schemas。
types?	(数据)类型[8]声明。
traits?	traits[9]的声明。

[1] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#markdown
[2] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#base-uri-and-base-uri-parameters
[3] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#template-uri
[4] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#base-uri-and-base-uri-parameters
[5] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#protocols
[6] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#default-media-types
[7] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#user-documentation
[8] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#defining-types
[9] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#resource-types-and-traits

resourceTypes?	API使用的 资源类型[10] 。
annotationTypes?	注解使用的 注解类型[11] 声明。
(<annotationName>)?	API使用的 注解[12] 。注解是用圆括号包裹的键值映射，括号中是注解的名字，而值是注解的实例。
securitySchemes?	对每个资源和方法使用的 安全schemes[13] 。
securedBy?	安全schemes[14] 。
uses?	导入扩展 库[15] .
/<relativeUri>?	以反斜杠开始的URIs相对路径，指代API资源。 资源节点[16] 总是以反斜杠开始的，它可以是根节点，也可以是子节点，例如 /users 和 /{groupId}。

"schemas" 和 "types" 节点是互斥的同义词：处理器不允许在根级别同时处理两个以上的节点。我们建议用"types"节点替代"schemas"节点，因为我们将在未来的RAML版本中移除"schemas"别名。

[10] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#resource-types-and-traits
[11] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#declaring-annotation-types
[12] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#annotations
[13] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#security-schemes
[14] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#applying-security-schemes
[15] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#libraries
[16] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#resources-and-nested-resources

1.1. 用户文档[17]

可选的**documentation**节点包含了各种文档，这些文档主要用做API的用户手册和和参考文档。例如文档可以清晰的描述API如何工作，并说明技术和业务的场景。

documentation节点的值是一个或多个documents，每个document都必须是包含了下面两个键值对的映射：

键值	描述
title	文档标题，必须是非空字符串。
content	文档内容。它必须是非空字符串，同时可以使用 markdown[18]格式。

示例如下：

```
#%RAML 1.0
title: ZEncoder API
baseUri: https://app.zencoder.com/api
documentation:
 - title: Home
   content: |
     Welcome to the _Zencoder API_ Documentation. The _Zencoder API_
     allows you to connect your application to our encoding service
     and encode videos without going through the web interface. You
     may also benefit from one of our
     [integration libraries](https://app.zencoder.com/docs/faq/basics/
libraries)
     for different languages.
 - title: Legal
   content: !include docs/legal.markdown
```

[17] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#user-documentation
[18] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#markdown

1.2. Base URI 与 Base URI参数[19]

可选的 **baseUri** 节点指定API的根URI，其值必须是一个字符串，同时还要符合 RFC2396[20] 或者 URI模板[21] 规范。

如果 baseUri 值是一个 URI模板[22] ，那么可以使用base URI参数：

URI Parameter	值
version	根级别版本的值

任何出现在baseUri中的URI模板参数都可以通过 **baseUriParameters** 节点在API定义根路径下进行描述。baseUriParameters节点具有和 uriParameters[23] 一样的结构和语义，除此之外它还指定了URI中的参数。

下面的 RAML API 定义使用了 URI模板[24] 作为根URI：

```
#%RAML 1.0
title: Salesforce Chatter REST API
version: v28.0
baseUri: https://na1.salesforce.com/services/data/{version}/chatter
```

下面的例子明确指定了一个 base URI 参数：

```
#%RAML 1.0
title: Amazon S3 REST API
version: 1
baseUri: https://{bucketName}.s3.amazonaws.com
baseUriParameters:
  bucketName:
    description: The name of the bucket
```

[19] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#base-uri-and-base-uri-parameters
[20] https://www.ietf.org/rfc/rfc2396.txt
[21] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#template-uri
[22] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#template-uri
[23] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#template-uris-and-uri-parameters
[24] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#template-uri

在baseAPI以一个或多个反斜杠（/)结束时，这些末尾的斜线会被忽略。例如下面两个资源的的相对路径是 `http://api.test.com/common/users` 和 `http://api.test.com/common/users/groups`。

```
baseUri: http://api.test.com/common/
/users:
  /groups:
```

下面的例子更复杂，它们的实际资源路径如下：`//api.test.com//common/`, `//api.test.com//common//users/`, and `//api.test.com//common//users//groups//`.

```
baseUri: //api.test.com//common//
/:
  /users/:
    /groups//:
```

1.3. 协议[25]

可选的 **protocols** 节点说明了API支持的协议。如果 protocaols 没有明确指定，那么一或多个protocols会被包含在baseUri节点中。protocals节点必须是非空的字符串数组，可以是HTTP和/或HTTPS，不区分大小写。

参见下方的示例：

```
#%RAML 1.0
title: Salesforce Chatter REST API
version: v28.0
protocols: [ HTTP, HTTPS ]
baseUri: https://na1.salesforce.com/services/data/{version}/chatter
```

1.4. 默认请求类型[26]

mediaType这个节点是可选的，它能设置默认的请求或响应类型，

[25] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#protocols
[26] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#default-media-types

mediaType节点必须是一个字符串序列，它用于说明该URL的内容类型。你可以在 RFC6838[27] 这个网址去看看支持的媒体类型有哪些。

下面给出了一个json类型的内容的RAML文档示例，这向用户说明：如果请求中没有明确指定媒体类型，那么此API只会接受和响应JSON格式的内容。

```
#%RAML 1.0
title: New API
mediaType: application/json
```

下面这个示例展示了一个可以同时接收和返回Json或xml的RAML片段。

```
#%RAML 1.0
title: New API
mediaType: [ application/json, application/xml ]
```

你可以明确指定哪些类型的内容（Json或xml）可用于哪种请求（POST或GET操作）。下面的片段说明了 `/list` 会返回一个JSON或XML的资源，而`/send`只会默认返回JSON类型的资源。详情参见 body[28]。

```
#%RAML 1.0
title: New API
mediaType: [ application/json, application/xml ]
types:
  Person:
  Another:
/list:
  get:
    responses:
      200:
        body: Person[]
/send:
  post:
    body:
      application/json:
        type: Another
```

[27] https://tools.ietf.org/html/rfc6838
[28] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#bodies

1.5. 默认的安全设置[29]

securedBy节点是可选的，它可以用来设置默认的安全schemes，从而为API的每一个资源的每一个方法添加保护。该节点的值可以是多个security scheme的name。详情参见 Applying Security Schemes[30]，里面说明了应用程序如何通过继承机制解析多个security schemes。

下面的示例展示了一个API，它允许通过OAuth 2.0或者OAuth 1.1协议进行访问：

```
#%RAML 1.0
title: Dropbox API
version: 1
baseUri: https://api.dropbox.com/{version}
securedBy: [ oauth_2_0, oauth_1_0 ]
securitySchemes:
  oauth_2_0: !include securitySchemes/oauth_2_0.raml
  oauth_1_0: !include securitySchemes/oauth_1_0.raml
```

2. RAML 数据类型

2.1. 简介[31]

RAML 1.0提出了**数据类型**的概念，它提供了一种便捷而有力的描述API数据的方式。数据类型可以对数据的类型进行声明，从而为其添加可校验的特性。

数据类型可以描述URI的资源、查询参数、请求或响应头，甚至是请求或响应报文体。数据类型可以是预建的或是自定义的。预建的类型可以用于描述出现在API的任何地方的数据。自定义类型可以通过继承的方式，由预建的类型进行衍生，然后像预建的类型那样使用。继承的类型无法创建任何循环依赖，但可以被内联继承。

[29] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#default-security
[30] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#applying-security-schemes
[31] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#introduction-1

下面展示了一个RAML示例，它定义了一个User类型，并且声明了firstname, lastname, 以及 age 三个属性，这三个属性分别使用了预建的string和number类型。最后，我们在payload中使用了这个User类型（ schema ）。

```
#%RAML 1.0
title: API with Types
types:
  User:
    type: object
    properties:
      firstname: string
      lastname:  string
      age:       number
/users/{id}:
  get:
    responses:
      200:
        body:
          application/json:
            type: User
```

RAML类型声明类似于JSON schema定义。事实上，RAML类型可以用于替代JSON和XML schemas，或者用于作为补充。RAML类型的语法被设计得更易于使用，并且比JSON和XML的schemas更简洁，甚至比它们更灵活且更具有表现力。下面的片段展示了多个类型声明的示例：

```
#%RAML 1.0
title: My API with Types
mediaType: application/json
types:
  Org:
    type: object
    properties:
      onCall: AlertableAdmin
      Head: Manager
  Person:
    type: object
    properties:
      firstname: string
      lastname:  string
      title?:    string
  Phone:
    type: string
```

```
    pattern: "[0-9|-]+"
  Manager:
    type: Person
    properties:
      reports: Person[]
      phone:  Phone
  Admin:
    type: Person
    properties:
      clearanceLevel:
        enum: [ low, high ]
  AlertableAdmin:
    type: Admin
    properties:
      phone: Phone
  Alertable: Manager | AlertableAdmin
/orgs/{orgId}:
  get:
    responses:
      200:
        body:
          application/json:
            type: Org
```

2.2. 概览

这一节是一个概览。

RAML类型系统的灵感来源于Java，同时又和XSD和Json Schemas类似。

RAML类型概览：

- Types和Java类很相似。
  - Types借鉴了JSON Schema，XSD，以及其他面向对象语言的类型的特性。
- 你可以通过继承其他类型来定义一个新的类型。
  - 和Java不同，RAML类型可以进行多继承。
- Types可以划分为四种：外部（扩展）类型、对象类型、数组类型、scalar（标量）类型。

- Types可以定义两种成员：**properties（属性）**和**facets（面）**。二者都可以被继承。
  - **Properties（属性）**非常常见，对象由属性组成。
  - **Facets（面）**是比较特别配置，你可以通过facet值的特征来描述类型。例如minLength（最小长度）和maxLength(最大长度）。
- 只有对象类型可以声明属性，但所有的类型都可以声明facets（面）。
- 你可以通过实现facets，给facets一个具体的值，从而指定scalar类型。
- 为了指定一个对象类型，你需要定义属性。

2.3. 定义类型[32]

类型可以通过继承API预定义类型来声明一个新的类型，在API的根节点下，**types**节点是可选的，你也可以直接包含另一个RAML库。你应该使用 键值对（map）[33]的方式来声明一个类型，就像下面这样：

```
types:
  Person: # key name
    # value is a type declaration
```

2.4. 类型声明[34]

类型声明可以通过添加功能性facets（例如属性）或非功能性的facets（例如描述），来引用、封装或者继承其他类型，同样，也可以使用指代其他类型的**类型表达式**。下面的表格展示了所有类型声明可以使用的facet：

Facet	描述
default?	类型的默认值。API请求如果没有找到实例的类型，例如一个查询参数没有被指定类型时，API必须将其指定为default中描述的一种默认类型。

[32] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#defining-types
[33] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#type-declarations
[34] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#type-declarations

	类似的，API响应如果没有指定实例类型，那么客户端必须将服务器响应的实例指定为default中描述的特定类型。URI参数则比较特殊，如果某个指定了默认facet的URI参数没有获取到，那么客户端必须用一个默认值来代替它。
schema?	等价于"type"的别名，RAML 0.8中已经不再建议使用。后续的RAML API版本中会将此Facet移除，并用"type"来替代它。"type"同时支持XML和Json。
type?	当前类型继承或封装的一个类型。它的值只能是：a) 用户自定义类型名； b) RAML预建类型名(object对象, array数组, 或者scalar类型； c) 一个内联（匿名）类型的声明。
example?	一个关于此类型如何使用的示例。可以通过文档生成器来生成一个此类的对象的值，在"examples"facet被定义的时候，此facet不可用。详情参见Examples[35]。
examples?	此类型的示例（多个）。详情参见Examples[36]。
displayName?	可选的facet，用于向阅读者展示一个友好的名称。
description?	类型的详细描述。它的值可以是字符串，也可以是 markdown[37] 格式。

[35] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#defining-examples-in-raml
[36] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#defining-examples-in-raml
[37] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#markdown

(<annotationName>)?	此API所使用的https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#annotations[注解]。每个注解都是用括号包围起来的键值对。
facets?	附加的一个Map，它会为每一个继承此类型的子类型添加此facets限制。详情参见 用户自定义Facets[38] 。
xml?	为此类型添加 类型实例的XML序列化[39]功能。
enum?	此类型可用的枚举值，可以是数组。当配置此facet之后，此类型的值只能是此facet列表的值中的其中之一。

"schema"和"type"这两个facets只能择一而使用，下面是两个错误的示例：

```
types:
  Person:
    schema: # invalid as mutually exclusive with `type`
    type: # invalid as mutually exclusive with `schema`
```

```
/resource:
  get:
    responses:
      200:
        body:
          application/json: # start type declaration
            schema: # invalid as mutually exclusive with `type`
            type: # invalid as mutually exclusive with `schema`
```

官方建议用"type"来代替"schema"，因为schema在后续RAML版本中不再建议使用，而且"type"标签同时支持XML和JSON schema。

[38] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#user-defined-facets
[39] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#xml-serialization-of-type-instances

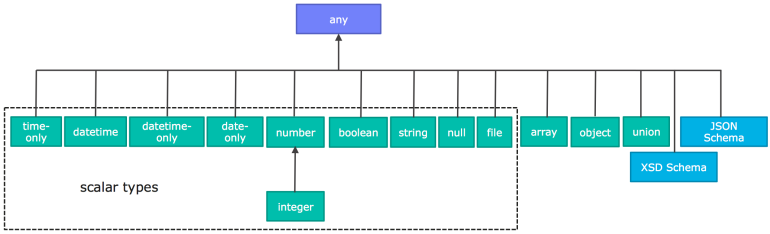
2.5. 预建类型

RAML类型系统定义了下列预建类型：

- any任意[40]
- object对象[41]
- array数组[42]
- union组合[43]类型表达式
- scalar类型[44]：number数字, boolean布尔, string字符串, date-only单日期, time-only单时间, datetime-only单日期时间, datetime日期时间, file文件, integer整型, 或者nil空。

作为附加的预建类型，RAML类型系统也允许定义 JSON或XML schema[45]。

下图展示了一个继承树，所有的类型都是由顶级类型 any 派生出来的：



"Any" 类型

任何类型的都是由 any类型派生出来的，所有类型都默认继承它（无论你是否显式继承）。上图中的基本类型都派生自any，any是所有类型的顶级父类。在

[40] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#the-any-type
[41] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#object-type
[42] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#array-type
[43] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#union-type
[44] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#scalar-types
[45] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#using-xml-and-json-schema

RAML中，any的角色类似于Java语言中的Object所扮演的角色，所有Java类型都直接或间接的继承自Object类。

any类型没有facets。

Object对象类型[46]

所有包含在继承树中的预建对象基类都可以在声明中使用下列facets：

Facet	描述
properties?	此类的实例可以或必须拥有的 属性[47]。
minProperties?	此类的实例所允许的此属性的最小数值。
maxProperties?	此类的实例所允许的此属性的最大数值。
additionalProperties?	此对象实例是否包含 附加的属性[48]。 **默认值：** `true`
discriminator?	由于联合或者继承会导致payloads包含一个模糊的类型，所以可能需要在运行时分辨一个类的具体类型。此facet的值可以是一个已声明的类型的##名。在内联（匿名）类中无法使用，也无法使用非scalar属性 进行辨别[49]。
discriminatorValue?	标识声明的类型，只能用于声明了`discriminator`facet的类型中。它的值必须能够在类型的层次中唯一标

[46] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#object-type
[47] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#property-declarations
[48] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#additional-properties
[49] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#using-discriminator

	识一个对象。此facet不支持内联类型声明。 **默认值：** 类型的名字。

对象类型必须显式继承自预建的object类型：

```
#%RAML 1.0
title: My API With Types
types:
  Person:
    type: object
    properties:
      name:
        required: true
        type: string
```

属性声明[50]

对象类型的属性由可选的**properties** facet进行定义。RAML规范把"properties" facet的值叫做 "属性声明"。属性声明是一个键值对，键是可以用于类型实例的有效属性名，值是类型名或内联（匿名）类型声明。

无论属性是必须的还是可选的，属性声明都可以被指定。

Facet	描述
required?	指定一个属性是否是必须的。 **默认值：** `true`.

下面的示例为一个对象类型声明了两个属性：

```
types:
  Person:
    properties:
      name:
```

[50] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#property-declarations

```
        required: true
        type: string
      age:
        required: false
        type: number
```

下列示例展示了一个通用的惯例：

```
types:
  Person:
    properties:
      name: string # equivalent to ->
                   # name:
                   #  type: string
      age?: number # optional property; equivalent to ->
                   # age:
                   #  type: number
                   #  required: false
```

在required facet作用于某个类型声明中的某个属性时，任何对于属性名的问题标记都是针对属性名的一部分，而不是作为一个可选属性的指示器。

```
types:
  profile:
    properties:
      preference?:
        required: true
```

profile类型具有一个叫做preference?的属性，它可以包含附加的问题标记。下列代码段展示了两种可选的使用preference?的方式：

```
types:
  profile:
    properties:
      preference?:
        required: false
```

或

```
types:
```

```
  profile:
    properties:
      preference??:
```

注意：.
对象类型不包含"属性" facet时，那么此对象就会被认为是无约束的对象，它可以包含任何类型的任何属性。

附加属性[51]

默认情况下，任何对象的实例都可以拥有附加的属性，而不仅仅是规范中的数据类型`properties` facet。下面的代码展示了之前章节声明的数据类型`Person`的对象实例。

```
Person:
  name: "John"
  age: 35
  note: "US" # valid additional property `note`
```

`note`属性没有明确在`Person`数据类型中声明，但它仍然有效，因为所有的附加类型都是默认生效的，而无论是否被显式声明。

为了约束附加属性，你可以设置 `additionalProperties` facet的值为`false`，你也可以指定正则表达式patterns来匹配需要设置的键，并为它们添加约束。后文中我们会把它们统称为`pattern##`。patterns是由成对的/字符来界定，就像下面这样：

```
#%RAML 1.0
title: My API With Types
types:
  Person:
    properties:
      name:
        required: true
        type: string
      age:
        required: false
```

[51] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#additional-properties

```
        type: number
      /^note\d+$/: # restrict any properties whose keys start
 with "note"
                   # followed by a string of one or more digits
        type: string
```

这一pattern属性可以为所有以"note"字符串开头的键添加附加属性约束。下面的示例中，note属性对于"US"是生效的，但对于同名的note属性则无效，因为它的值是一个数字类型而不是字符串类型。

```
Person:
  name: "John"
  age: 35
  note: 123 # not valid as it is not a string
  address: "US" # valid as it does not match the pattern
```

可以通过下列方式，强制所有被附加的属性都是字符串，而不管它们的键值是什么：

```
#%RAML 1.0
title: My API With Types
types:
  Person:
    properties:
      name:
        required: true
        type: string
      age:
        required: false
        type: number
      //: # force all additional properties to be a string
        type: string
```

如果pattern属性正则表达式同时匹配了一个已经被明确声明的属性，那么正则将让位于明确声明的属性。如果同时有两个正则表达式同时匹配了一个属性名，那么先声明的正则优先。

更进一步，如果对于给定的类型定义，additionalProperties是false(显式或内联方式指定)，那么就不允许使用partten属性；相反，如果additionalProperties是true（或未指定），那么则允许使用pattern属性。

对象类型的特性[52]

你可以通过继承其他对象类型来声明对象类型。子类会继承父类所有的属性。在下列示例中，Employee继承了父类Person的所有属性：

```
#%RAML 1.0
title: My API With Types
types:
  Person:
    type: object
    properties:
      name:
        type: string
  Employee:
    type: Person
    properties:
      id:
        type: string
```

子类还可以重写父类的属性，但有如下两个约束：1）父类中的必填属性无法在子类中改为可选属性； 2) 父类中声明的属性在子类中只能具象化为更明确的类型，而不能被修改为其他类型。

使用鉴别器[53]

当payloads由于组合或者继承的原因包含了一个模糊类型的时候，它通常能在运行时确定为不同类型的实例，这在payload被反序列化为静态类型语言时经常发生。

RAML处理器可以提供一种自动选择类型的机制，一个简单办法是通过关联的类型对象中某些唯一的特征来确定运行时类型。

你可以使用discriminator facet来设置对象属性的名字。该名字的对象属性会被用于鉴别更具体的类型。discriminatorValue可以用于保存标识某一具体对象的类型的值。默认情况下，discriminatorValue和类型名相同。

下面是使用discriminator的示例：

[52] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#object-type-specialization
[53] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#using-discriminator

```
#%RAML 1.0
title: My API With Types
types:
  Person:
    type: object
    discriminator: kind # refers to the `kind` property of object
 `Person`
    properties:
      kind: string # contains name of the kind of a `Person` instance
      name: string
  Employee: # kind can equal `Employee`; default value for
 `discriminatorValue`
    type: Person
    properties:
      employeeId: integer
  User: # kind can equal `User`; default value for `discriminatorValue`
    type: Person
    properties:
      userId: integer
```

```
data:
  - name: A User
    userId: 111
    kind: User
  - name: An Employee
    employeeId: 222
    kind: Employee
```

你也可以为每个类重写discriminatorValue。下面的示例通过小写字母来重新指定``discriminatorValue的默认值：

```
#%RAML 1.0
title: My API With Types
types:
  Person:
    type: object
    discriminator: kind
    properties:
      name: string
      kind: string
  Employee:
    type: Person
    discriminatorValue: employee # override default
```

```
    properties:
      employeeId: string
  User:
    type: Person
    discriminatorValue: user # override default
    properties:
      userId: string
```

```
data:
  - name: A User
    userId: 111
    kind: user
  - name: An Employee
    employeeId: 222
    kind: employee
```

discriminator和discriminatorValue都不能用于内联类型或者组合类型。

```
# valid whenever there is a key name that can identify a type
types:
  Device:
    discriminator: kind
    properties:
      kind: string
```

```
# invalid in any inline type declaration
application/json:
   discriminator: kind
   properties:
     kind: string
```

```
# invalid for union types
PersonOrDog:
   type: Person | Dog
   discriminator: hasTail
```

数组类型[54]

数组类型可以用方括号[]这种 类型表达式[55]来标识，也可以在type facet中使用array值来指定。如果你定义了一个顶级数组类型，例如Emails，那么你可以通过下列facet来对数组类型进行进一步的约束：

Facet	说明
uniqueItems?	布尔值。可以用于指示此数组的元素是否必须唯一。
items?	表明此数组的元素继承自哪里。可以引用已存在的类型，也可以引用内联类型声明[56]。
minItems?	数组中最少需要几个元素。此值必须大于等于0。 **默认值：** 0.
maxItems?	数组中最多能用几个元素，此值必须大于等于0。 **默认值：** 2147483647.

下列两个示例都是有效的：

```
types:
  Email:
    type: object
    properties:
      subject: string
      body: string
  Emails:
    type: Email[]
    minItems: 1
```

[54] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#array-type
[55] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#type-expressions
[56] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#type-declaration

```
    uniqueItems: true
    example: # example that contains array
      - # start item 1
        subject: My Email 1
        body: This is the text for email 1.
      - # start item 2
        subject: My Email 2
        body: This is the text for email 2.
```

```
types:
  Email:
    type: object
    properties:
      name:
        type: string
  Emails:
    type: array
    items: Email
    minItems: 1
    uniqueItems: true
```

type facet中使用 `Email[]`和使用`type: array`是等价的。`items` facet定义了每个数组元素都必须继承自`Email`类型。

Scalar类型[57]

RAML定义了一些预建的scalar类型，它们都必须遵从一些预定义的约束。

字符串string[58]

JSON字符串具有如下facets：

Facet	说明
pattern?	此字符串必须匹配的正则表达式。
minLength?	此字符串的最小长度，必须大于等于0。

[57] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#scalar-types
[58] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#string

	默认值： 0
maxLength?	此字符串的最大长度，必须大于等于0。 **默认值：** 2147483647

示例：

```
types:
  Email:
    type: string
    minLength: 2
    maxLength: 6
    pattern: ^note\d+$
```

数字Number[59]

任何JSON数字（ 整型integer[60]也算）都包含以下facets：

Facet	说明
minimum?	此参数的最小值，此facet只能用于number或者integer。
maximum?	此参数的最大值，此facet只能用于number或者integer。
format?	此值的格式，只能是 int32, int64, int, long, float, double, int16, int8 其中之一。
multipleOf?	如果数值能够被multipleOf中的值整除，那么它是一个有效值。

例如：

```
types:
  Weight:
```

[59] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#number
[60] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#integer

```
    type: number
    minimum: 3
    maximum: 5
    format: int64
    multipleOf: 4
```

整型Integer[61]

JSON numbers的子集，包含正整数和负整数。integer类型从 数值类型number[62]集成了它的facets。

```
types:
  Age:
    type: integer
    minimum: 3
    maximum: 5
    format: int8
    multipleOf: 1
```

布尔型Boolean[63]

JSON布尔类型没有任何facets。

```
types:
  IsMarried:
    type: boolean
```

日期Date[64]

必须支持如下日期类型：

Type	Description

[61] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#integer
[62] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#number
[63] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#boolean
[64] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#date

date-only	RFC3339[65]规范中的全日期符号，格式是yyyy-mm-dd。不支持时间与时区时间的符号。
time-only	RFC3339[66]规范中的时间部分，格式是 hh:mm:ss[.ff…]。不支持日期或者时区时间的符号。
datetime-only	将date-only与time-only结合，并通过T分割，格式为 yyyy-mm-ddThh:mm:ss[.ff…]。不支持时区时间。
datetime	下列格式之一的时间戳： 如果 *format* 未指定，或者指定了 rfc3339，那么使用 RFC3339[67]规范中date-time的格式，如果 *format*被指定为 rfc2616，那么则使用 RFC2616[68]规范定义的格式。

只有在类型是 *datetime* 的时候 *format* 这个facet才能够起作用，并且 *format* 的值必须是 rfc3339 或者 rfc2616 二者之一，任何其他的值都是无效的。

```
types:
  birthday:
    type: date-only # no implications about time or offset
    example: 2015-05-23
  lunchtime:
    type: time-only # no implications about date or offset
    example: 12:30:00
  fireworks:
    type: datetime-only # no implications about offset
    example: 2015-07-04T21:00:00
  created:
    type: datetime
    example: 2016-02-28T16:41:41.090Z
    format: rfc3339 # the default, so no need to specify
```

65 http://xml2rfc.ietf.org/public/rfc/html/rfc3339.html#anchor14
66 http://xml2rfc.ietf.org/public/rfc/html/rfc3339.html#anchor14
67 http://xml2rfc.ietf.org/public/rfc/html/rfc3339.html#anchor14
68 https://www.ietf.org/rfc/rfc2616.txt

```
  If-Modified-Since:
    type: datetime
    example: Sun, 28 Feb 2016 16:41:41 GMT
    format: rfc2616 # this time it's required, otherwise, the example
 format is invalid
```

文件File[69]

file类型可以包含从表单发送过来的内容。在这一类型用于web表单内容提交时，它应该是通过有效的JSON格式进行提交。文件内容应该编码为base64字符串。

Facet	说明
fileTypes?	文件中有效内容类型的字符串的列表。文件类型为 / 时必须是一个有效值。
minLength?	指定参数的最小字节数，此值必须大于等于0。 **默认值：** 0
maxLength?	指定参数的最大字节数，此值必须大于等于0。 **默认值：** 2147483647

```
types:
  userPicture:
    type: file
    fileTypes: ['image/jpeg', 'image/png']
    maxLength: 307200
  customFile:
    type: file
    fileTypes: ['*/*'] # #########
    maxLength: 1048576
```

[69] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#file

空类型Nil[70]

在RAML中，nil是一种scalar类型，它只允许nil（空）数据值。特别的，YAML中只允许YAML的null（或者等价的~），JSON中只允许JSON的null，XML中只允许XML的 xsi:nil。在头部，URI参数和查询参数中，nil类型只允许字符串值"nil"(大小写敏感)；反过来，如果在字符串中发现了"nil"值（大小写敏感），那么说明它的类型是nil，它将被反序列化为nil值。

在下列示例中，对象类型具有两个必填参数，name和comment，二者默认类型都是string。在example中，name被分配了一个字符串值，但comment是nil，但这是不被允许的，因为RAML只接收字符串。

```
types:
  NilValue:
    type: object
    properties:
      name:
      comment:
    example:
      name: Fred
      comment: # #######
```

下列示例展示了给comment一个nil类型。

```
types:
  NilValue:
    type: object
    properties:
      name:
      comment: nil
    example:
      name: Fred
      comment: # ######
```

下列示例展示了如何在组合中使用可空（nilable）属性：

```
types:
  NilValue:
```

[70] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#nil-type

```
    type: object
    properties:
      name:
      comment: nil | string # equivalent to ->
                            # comment: string?
    example:
      name: Fred
      comment: # #############
```

声明属性的类型为nil，意味着类型实例中缺乏该值。在RAML上下文中需要一个nil类型的值（相对于类型声明），在YAML中通常使用null。如果type是 `nil | number`，那么你可以使用 `enum: [ 1, 2, ~]`，或者更进一步 `enum: [ 1, 2, !! null ""]` ；在非内联符号中，你也可以完全忽略此值。

组合类型[71]

组合类型允许数据的类型从多个类型中择一而用。组合类型通过使用竖线(|)来连接多个类型来使用，这些被连接起来的类型被当做组合类型的超类。在下列示例中，Device类型可以是Phone或者Notebook这两个类型的其中之一。

```
#%RAML 1.0
title: My API With Types
types:
  Phone:
    type: object
    properties:
      manufacturer:
        type: string
      numberOfSIMCards:
        type: number
      kind: string
  Notebook:
    type: object
    properties:
      manufacturer:
        type: string
      numberOfUSBPorts:
        type: number
      kind: string
  Device:
```

[71] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#union-type

```
    type: Phone | Notebook
```

当且仅当它满足其中一个父类的全部约束时，联合类型的实例才是有效的。当且仅当实例是至少一个超类的有效实例，并且此超类可以由类层次中通过扩展全部组合类型来获取到，那么此实例才是有效的。当一个实例被通过操作这一扩展，并且为所有超类匹配其实例来进行反序列化操作时，从最左边开始处理到最右边；在首次找到成功匹配的基类时，那就用它来反序列化此实例。

下列示例定义了两个类型和一个包含了二者的第三个联合类型：

```
types:
  CatOrDog:
    type: Cat | Dog # elements: Cat or Dog
  Cat:
    type: object
    properties:
      name: string
      color: string
  Dog:
    type: object
    properties:
      name: string
      fangs: string
```

下列示例是一个有效的CatOrDog实例：

```
CatOrDog: # follows restrictions applied to the type 'Cat'
  name: Musia,
  color: brown
```

想象一下一个更复杂的联合类示例，它使用了多继承：

```
types:
   HasHome:
     type: object
     properties:
       homeAddress: string
   Cat:
     type: object
     properties:
       name: string
       color: string
```

```
  Dog:
    type: object
    properties:
      name: string
      fangs: string
  HomeAnimal: [ HasHome ,  Dog | Cat ]
```

这种情况下，HomeAnimal具有两个超类，HasHome和一个匿名联合类，它通过Dog | Cat这个类型表达式来定义。

对HomeAnimal类型的验证包含了对它的每一个父类的验证，以及联合类型中每一个元素类型的验证。在这种特殊情况下，处理器必须测试[HasHome, Dog]和[HasHome, Cat]是否是有效类型。

如果你继承了两个联合类型，处理器必须对每个可能的组合进行校验。例如，校验下述HomeAnimal类型时，处理器必须测试六种可能的组合：[HasHome, Dog], [HasHome, Cat], [HasHome, Parrot], [IsOnFarm, Dog], [IsOnFarm, Cat], and [IsOnFarm, Parrot]。

```
types:
  HomeAnimal: [ HasHome | IsOnFarm ,  Dog | Cat | Parrot ]
```

使用XML和JSON Schema[72]

RAML允许使用XML和JSON schema来描述API请求和响应的报文体，这一功能通过把schemas集成到数据类型系统中来实现。

下列示例展示了如何包含一个扩展的JSON schema到顶层类型定义中以及报文体声明中：

```
types:
  Person: !include person.json
```

```
/person:
  get:
    responses:
      200:
        body:
```

[72] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#using-xml-and-json-schema

```
          application/json:
            type: !include person.json
```

RAML处理器不允许对XML或JSON schema中定义的类型进行任何继承或专门化，也不允许他们出现在有效的 类型表达式[73] 中。因此，你无法定义这些类的子类，也无法为它们声明任何新的属性，无法添加约束，设置facets，也无法声明facets。但你可以通过添加annotations、examples、display name或者description来对它们进行简单封装。

下列示例展示了一个有效声明：

```
types:
  Person:
    type: !include person.json
    description: this is a schema describing person
```

下列示例展示了一个无效的类型声明，因为它继承了JSON schema的特征，并添加了附加属性：

```
types:
  Person:
    type: !include person.json
    properties: # invalid
      single: boolean
```

下面是另一个无效示例，因为Person在另一个类型中被当作一个属性类型来使用：

```
types:
  Person:
    type: !include person.json
    description: this is a schema describing person
  Board:
    properties:
      members: Person[] # invalid use of type expression '[]' and as a
 property type
```

RAML处理器必须能够解释、使用JSON schema和XML schema。

[73] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#type-expressions

XML schema或者JSON schema禁止用于不支持XML格式或JSON格式的媒体类型数据。XML和JSON schemas也禁止声明与查询参数、查询字符串、URI参数和报文头。

`schemas`、`types`节点和`schema`、`type`节点类似，它们是同义词，并且相互排斥。但你更应该使用`types`或者`type`，因为`schemas`和`schema`在未来的RAML版本中可能会被移除。

引用内部元素[74]

有时候，引用在schema中定义的元素非常重要。RAML允许你通过URL fragment进行引用，就像下面这样：

```
type: !include elements.xsd#Foo
```

在引用一个schema的内部元素时，RAML处理器必须对它进行一些特殊校验。RAML规范支持引用任何有效的JSON schema中的内部元素、任何全局定义元素、XML schemas中的复杂类型。但有如下限制：

- 校验XML或者JSON实例的内部元素时，必须对XML或者JSON schema进行同样的校验。
- 对于XML实例结构的判断，可以引用XSD中的复杂类型，但如果复杂类型没有在顶级XML元素中定义一个名字，那么此类型无法用于序列化XML实例。

2.6. 用户自定义Facets

Facets为类型添加了各种附加的约束，例如数字类型numbers的`minimum`和`maximum`，scalars类型的`enum` facet。除了RAML预建的facets之外，用户也可以根据需要，为各种类型自定义facets。

用户自定义facet可以在类型声明中使用`facets`这一facet进行声明（很绕是吧，就是这么绕）。`facets`的值是一个map。map的键是自定义facet的名字。其中的值代表可用于此自定义facet中的值。自定义facet声明的语法和 属性声明[75]的语法是一样的。facet根据不同的值对类型的实例进行约束，而不约束类型本身。

[74] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#references-to-inner-elements
[75] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#property-declarations

facet的名字不允许用左括号开始，从而与注解annotations进行区分。在类型type中，用户自定义的facet不能与类型的预建facets同名，也不允许与该类的继承树中的任何父类中的任何facet同名。

若类型中的facet声明为必填项，那么任何type的子类都必须为此facet定义一个值。

下列是一个示例，它为dates添加了约束，不允许dates是一个节假日：

```
#%RAML 1.0
title: API with Types
types:
  CustomDate:
    type: date-only
    facets:
      onlyFutureDates?: boolean # ##`PossibleMeetingDate`####
      noHolidays: boolean # ##`PossibleMeetingDate`####
  PossibleMeetingDate:
    type: CustomDate
    noHolidays: true
```

在此示例中，我们声明了`noHolidays`，并为它定义了一个对日期实例的约束，描述日期能否是节假日。任何继承此类型（CustomDate）的子类都必须为它设置一个值，要么true，要么false，就像上例中的`PossibleMeetingDate`。

用户自定义facets并不属于RAML规范的一部分，因此RAML处理器无需对它们进行标准化处理。RAML处理器可以选择处理或不处理用户自定义facets。在上面的例子中，RAML处理器无需赋予`noHolidays`任何含义，所以也不必纠结`PossibleMeetingDate`这一实例中的`noHolidays`的值到底是true还是false。

2.7. 鉴别默认类型[76]

RAML处理器必须能够鉴别通过如下规则声明的类型的默认类型：

- 当且仅当类型声明中包含了一个`properties`facet，那么它的默认类型就应该是`object`。下列示例展示了这一规则：

```
types:
```

[76] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#determine-default-types

```
  Person:
    type: object
    properties:
```

这一规则也可以用下面的格式：

```
types:
  Person:
    # #####`object`#######
    properties:
```

- 当且仅当类型声明既不包含properties facet，也不包含type或schema facet时，默认类型才是string。下列片段展示了这一规则：

```
types:
  Person:
    properties:
      name: # ####type#schema######`string`#
```

- 任何body节点如果不包含properties，type或schema，那么默认类型则是any。例如：

```
body:
  application/json:
    # #####`any`
```

如果已经定义了默认媒体类型，那么就可以不再声明，就像下面这样：

```
body:
  # #####`any`
```

当然，所有规则都可以被明确的重写到类型定义中：

```
types:
  Person:
    properties:
      name:
        type: number
```

2.8. 类型表达式Type Expressions[77]

类型表达式提供了强大的方式来引用和定义类型。类型表达式可以被用于任何可以使用type的地方。最简单的类型表达式就是一个类型的名字。通过使用类型表达式，你可以设计类型组合、数组、maps以及其他一些有趣的玩意。

表达式	说明
`Person`	最简单的类型表达式：一个简单类
`Person[]`	一个Person对象的数组
`string[]`	一个scalars字符串的数组
`string[][]`	一个scalars字符串的二维数组
`string | Person`	一个联合类型，要么它是一个string，要么它是一个Person
``(string | Person)[]``	

类型表达式可以被用于任何能够使用类型的地方：

```
#%RAML 1.0
title: My API With Types

types:
  Phone:
    type: object
    properties:
      manufacturer:
        type: string
      numberOfSIMCards:
        type: number
  Notebook:
    type: object
    properties:
      manufacturer:
        type: string
      numberOfUSBPorts:
        type: number
  Person:
    type: object
```

[77] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#type-expressions

```
    properties:
      devices: ( Phone | Notebook )[]
      reports: Person[]
```

你甚至可以从一个类型表达式进行继承：

```
#%RAML 1.0
title: My API With Types
types:
  Phone:
    type: object
    properties:
      manufacturer:
        type: string
      numberOfSIMCards:
        type: number
  Notebook:
    type: object
    properties:
      manufacturer:
        type: string
      numberOfUSBPorts:
        type: number
  Devices:
    type:  ( Phone | Notebook )[]
```

这个例子声明了两个复杂类型：Phone和Notebook。也声明了一个Phone和Notebook联合到一起的一个组合类的数组类（没错，表述起来就是这么绕），并为这个数组类定义了一个别名（type alias）Devices。你可以通过这种办法为其他复杂类型添加一个简单的名字，也可以为其加上其他附加的属性，例如description或者annotations。

语法[78]

类型表达式是预建类型或者自定义类型再结合某些符号的表达式，比如下面这些：

表达式组合	说明	例子

[78] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#grammar

type name	类型的名称，构建类型表达式的最基本的模块，它是一种最简单的类型表达式。	`number`:是一种预建类型 `Person`:是一种自定义类型

(type expression)	用括号来消除歧义。	`Person | Animal[]`
		
		`( Person | Animal )[]`

(type expression)[]	通过在类型表达式后面加上一对方括号作为后缀来定义一维数组，说明这个类型是一个表达式所代表的类型的数组类。	`string[]`:是一个字符串数组 `Person[][]`: 是一个Person实例的二维数组。
(type expression 1) | (type expression 2)	通过竖线|来连接两个类型表达式，表明它是一个联合类型（二选一）。联合操作符可以在一个表达式中被多次使用。	没有示例

2.9. 多继承Multiple Inheritance[79]

RAML类型支持多继承。它是通过一个类型序列来实现的。

```
types:
  Person:
    type: object
    properties:
      name: string
  Employee:
    type: object
    properties:
      employeeNr: integer
  Teacher:
    type: [ Person, Employee ]
```

[79] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#multiple-inheritance

上述例子中，Teacher同时继承了Person和Employee的约束。

只有在子类继承所有父类的约束时仍然有效的情况下，多继承才会被允许。因此，它无法继承多个不同的（预建类型的）初始类型，例如[number, string]。

下列示例中，子类Number3完全有效：

```
types:
  Number1:
    type: number
    minimum: 4
  Number2:
    type: number
    maximum: 10
  Number3: [ Number1, Number2]
```

在同样的示例中，如果把Number2的maximum值从10改成2，那么Number3则成为一个无效类型。

```
types:
  Number1:
    type: number
    minimum: 4
  Number2:
    type: number
    maximum: 2
  Number3: [ Number1, Number2] # ###################
```

联合类型[80]这一小节中展示了如何用多继承和联合类型来进行校验的另一个示例。

如果子类从至少两个父类中继承了同名的属性，那么有两种情况下子类会被认为是无效的：1) 当某个父类已经声明了"pattern" facet时，又定义了一个"pattern"。 2) 当另一个用户自定义facet具有相同的值时，又使用用户自定义的facet。这些情况下，我们认为这是一个无效类型声明。

[80] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#union-multiple-inheritance

2.10. 内联类型声明Inline Type Declarations[81]

你可以在任何可以使用类型表达式的地方使用内联/匿名类型。

```
#%RAML 1.0
title: My API With Types
/users/{id}:
  get:
    responses:
      200:
        body:
          application/json:
            type: object
            properties:
              firstname:
                type: string
              lastname:
                type: string
              age:
                type: number
```

2.11. 再RAML示例中定义一个示例[82]

请务必接受我们的安利：请在你的API文档中包含一个具有代表性的示例。RAML支持定义多个示例，或者一个简单的任一的类型声明的实例。

多个示例[83]

examples facet是可选的，它能够用于为类型声明添加附带的例子。它的值是一个键值对表示的map，每个键值对都唯一标识某个 单一示例[84]。

下列示例展示了**examples** facet的值：

```
message: # {key} - unique id
```

[81] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#inline-type-declarations
[82] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#defining-examples-in-raml
[83] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#multiple-examples
[84] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#single-example

```
  # example declaration
  title: Attention needed
  body: You have been added to group 274
record: # {key} - unique id
  # example declaration
  name: log item
  comment: permission check
```

单一示例[85]

example facet是可选的，它能够用于给某个类型声明附加一个类型实例的示例。有两种方式进行附加：为类型实例指定一个明确的说明，或者在map中附带一些facets。

为类型实例指定一个明确的说明[86]

例如：

```
title: Attention needed
body: You have been added to group 274
```

在map中附带一些facets[87]

map中可以包含下列附带的facets：

Facet	说明
displayName?	对使用者友好的示例的名字。如果示例是examples节点的一部分，那么默认值则是示例中已经定义的用于唯一标识它的键值。

[85] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#single-example
[86] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#as-an-explicit-description-of-a-specific-type-instance
[87] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#as-a-map-that-contains-additional-facets

description?	示例的详细描述。它的值是个字符串，也可以使用 markdown[88]格式。
(<annotationName>)?	用于此API的 注解[89]。注解是通过圆括号"("和")"括起来的键值对，键表示注解的名字，值表示注解的实例。
value	类型实例的真实示例。
strict?	是否要用类型声明对此实例进行校验（默认为true）。设置为false说明不必校验。

例子：

```
(pii): true
strict: false
value:
  title: Attention needed
  body: You have been added to group 274
```

在RAML中如何定义example/examples的示例[90]

下列片段展示了example和examples属性如何在不同级别的RAML API中使用的示例：

```
#%RAML 1.0
title: API with Examples

types:
  User:
    type: object
    properties:
      name: string
      lastname: string
    example:
```

[88] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#markdown
[89] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#annotations
[90] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#example-of-how-to-define-exampleexamples-in-raml

```
      name: Bob
      lastname: Marley
  Org:
    type: object
    properties:
      name: string
      address?: string
      value?: string
/organization:
  post:
    headers:
      UserID:
        description: the identifier for the user who posts a new
 organization
        type: string
        example: SWED-123 # single scalar example
    body:
      application/json:
        type: Org
        example: # single request body example
          value: # needs to be declared since instance contains
 a 'value' property
            name: Doe Enterprise
            value: Silver
  get:
    description: Returns an organization entity.
    responses:
      201:
        body:
          application/json:
            type: Org
            examples:
              acme:
                name: Acme
              softwareCorp:
                value: # validate against the available facets for the
 map value of an example
                  name: Software Corp
                  address: 35 Central Street
                  value: Gold # validate against an instance of the
 `value` property
```

2.12. 类型实例的XML序列化[91]

RAML通过在 类型声明[92]中附加xml节点来简化XML序列化这一过程的复杂性。此节点用于配置类型实例应该如何被序列化为XML。xml节点的值是一个包含下列节点的map：

Name	Type	说明
attribute?	boolean	true将此类型实例序列化为一个XML属性，只允许scalar类型，可以是true。 **默认值：** false
wrapped?	boolean	true表示将此类型实例封装为一个XML元素。可以是scalar类型的true，和attribute中的true类似。 **默认值：** false
name?	string	更改序列化出来的XML元素或属性名。 **默认值：** 类型或属性的名字。
namespace?	string	配置XML命名空间的名字。
prefix?	string	配置用于序列化为XML的前缀。

下列类型声明展示了xml节点的使用：

```
types:
```

[91] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#xml-serialization-of-type-instances

[92] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#type-declarations

```
  Person:
    properties:
      name:
        type: string
        xml:
          attribute: true # serialize it as an XML attribute
          name: "fullname" # attribute should be called fullname
      addresses:
        type: Address[]
        xml:
          wrapped: true # serialize it into its own ... XML element
  Address:
    properties:
      street: string
      city: string
```

上述示例可以序列化为下面的XML：

```
<Person fullname="John Doe">

     …
     ...

</Person>
```