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

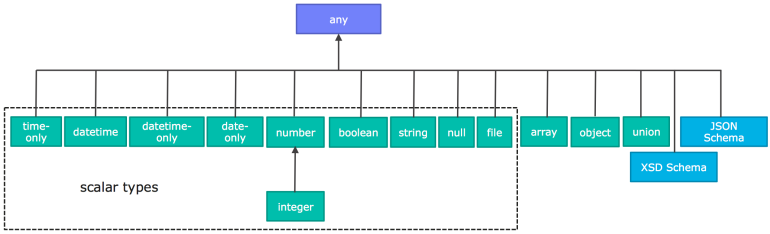
2.5. 预建类型

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:
        required: true
```

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

```
        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
        type: number
      /^note\d+$/: # restrict any properties whose keys start
 with "note"
```

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

```
                    # 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
```

If a pattern property regular expression also matches an explicitly declared property, the explicitly declared property definition prevails. If two or more pattern property regular expressions match a property name in an instance of the data type, the first one prevails.

Moreover, if `additionalProperties` is `false` (explicitly or by inheritance) in a given type definition, then explicitly setting pattern properties in that definition is not allowed. If `additionalProperties` is `true` (or omitted) in a given type definition, then pattern properties are allowed and further restrict the additional properties allowed in that type.

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[52] Object Type Specialization

You can declare object types that inherit from other object types. A sub-type inherits all the properties of its parent type. In the following example, the type `Employee` inherits all properties of its parent type `Person`.

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

A sub-type can override properties of its parent type with the following restrictions: 1) a required property in the parent type cannot be changed to optional in the sub-type, and 2) the type declaration of a defined property in the parent type can only be changed to a narrower type (a specialization of the parent type) in the sub-type.

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

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[53] Using Discriminator

When payloads contain ambiguous types due to unions or inheritance, it is often impossible to discriminate the concrete type of an individual object at runtime. For example, when deserializing the payload into a statically typed language, this problem can occur.

A RAML processor might provide an implementation that automatically selects a concrete type from a set of possible types, but a simpler alternative is to store some unique value associated with the type inside the object.

You set the name of an object property using the `discriminator` facet. The name of the object property is used to discriminate the concrete type. The `discriminatorValue` stores the actual value that might identify the type of an individual object. By default, the value of `discriminatorValue` is the name of the type.

Here’s an example that illustrates how to use `discriminator`:

```
#%RAML 1.0
title: My API With Types
types:
  Person:
    type: object
    discriminator: kind # refers to the `kind` property of object
  `Person`
```

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

```
    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
```

You can also override the default for `discriminatorValue` for each individual concrete class. The following example replaces the default value of `discriminatorValue` in initial caps with lowercase:

```
#%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
```

Neither `discriminator` nor `discriminatorValue` can be defined for any inline type declarations or union types.

```
# 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
```

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[54] Array Type

Array types are declared by using either the array qualifier `[]` at the end of a type expression[55] or `array` as the value of a `type` facet. If you are defining a top-level array type, such as the `Emails` in the examples below, you can declare the following facets in addition to those previously described to further restrict the behavior of the array type.

Facet	Description
uniqueItems?	Boolean value that indicates if items in the array MUST be unique.
items?	Indicates the type all items in the array are inherited from. Can be a reference to an existing type or an inline type declaration[56].
minItems?	Minimum amount of items in array. Value MUST be equal to or greater than 0. **Default:** `0`.

[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

maxItems?	Maximum amount of items in array. Value MUST be equal to or greater than 0. **Default:** `2147483647`.

Both of the following examples are valid:

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

Using `Email[]` is equivalent to using `type: array`. The `items` facet defines the `Email` type as the one each array item inherits from.

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[57] Scalar Types

RAML defines a set of built-in scalar types, each of which has a predefined set of restrictions.

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[58] String

A JSON string with the following additional facets:

Facet	Description

[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

pattern?	Regular expression that this string should match.
minLength?	Minimum length of the string. Value MUST be equal to or greater than 0. **Default:** `0`
maxLength?	Maximum length of the string. Value MUST be equal to or greater than 0. **Default:** `2147483647`

Example:

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

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[59] Number

Any JSON number including integer[60] with the following additional facets:

Facet	Description
minimum?	The minimum value of the parameter. Applicable only to parameters of type number or integer.
maximum?	The maximum value of the parameter. Applicable only to parameters of type number or integer.
format?	The format of the value. The value MUST be one of the following: int32, int64, int, long, float, double, int16, int8
multipleOf?	A numeric instance is valid against "multipleOf" if the result of dividing the instance by this keyword’ s value is an integer.

[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

Example:

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

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[61] Integer

A subset of JSON numbers that are positive and negative multiples of 1. The integer type inherits its facets from the number type[62].

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

[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

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[63] Boolean

A JSON boolean without any additional facets.

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

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[64] Date

The following date type representations MUST be supported:

[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

Type	Description
date-only	The "full-date" notation of RFC3339[65], namely `yyyy-mm-dd`. Does not support time or time zone-offset notation.
time-only	The "partial-time" notation of RFC3339[66], namely hh:mm:ss[.ff...]. Does not support date or time zone-offset notation.
datetime-only	Combined date-only and time-only with a separator of "T", namely yyyy-mm-ddThh:mm:ss[.ff...]. Does not support a time zone offset.
datetime	A timestamp in one of the following formats: if the *format* is omitted or set to `rfc3339`, uses the "date-time" notation of RFC3339[67]; if *format* is set to `rfc2616`, uses the format defined in RFC2616[68].

The additional facet *format* MUST be available only when the type equals *datetime*, and the value MUST be either `rfc3339` or `rfc2616`. Any other values are invalid.

```
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
```

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

```
  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
  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
```

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[69] File

The **file** type can constrain the content to send through forms. When this type is used in the context of web forms it SHOULD be represented as a valid file upload in JSON format. File content SHOULD be a base64-encoded string.

Facet	Description
fileTypes?	A list of valid content-type strings for the file. The file type `/` MUST be a valid value.
minLength?	Specifies the minimum number of bytes for a parameter value. The

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

	value MUST be equal to or greater than 0. **Default:** 0
maxLength?	Specifies the maximum number of bytes for a parameter value. The value MUST be equal to or greater than 0. **Default:** 2147483647

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

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[70] Nil Type

In RAML, the type `nil` is a scalar type that allows only nil data values. Specifically, in YAML it allows only YAML's `null` (or its equivalent

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

representations, such as ~), in JSON it allows only JSON's `null`, and in XML it allows only XML's `xsi:nil`. In headers, URI parameters, and query parameters, the `nil` type only allows the string value "nil" (case-sensitive); and in turn an instance having the string value "nil" (case-sensitive), when described with the `nil` type, deserializes to a nil value.

In the following example, the type of an object and has two required properties, `name` and `comment`, both defaulting to type `string`. In `example`, `name` is assigned a string value, but comment is nil and this is *not* allowed because RAML expects a string.

```
types:
  NilValue:
    type: object
    properties:
      name:
      comment:
    example:
      name: Fred
      comment: # Providing no value here is not allowed.
```

The following example shows the assignment of the `nil` type to `comment`:

```
types:
  NilValue:
    type: object
    properties:
      name:
      comment: nil
    example:
      name: Fred
      comment: # Providing a value here is not allowed.
```

The following example shows how to represent nilable properties using a union:

```
types:
  NilValue:
    type: object
    properties:
```

```
      name:
      comment: nil | string # equivalent to ->
                            # comment: string?
    example:
      name: Fred
      comment: # Providing a value or not providing a value here is
allowed.
```

Declaring the type of a property to be `nil` represents the lack of a value in a type instance. In a RAML context that requires *values* of type `nil` (vs just type declarations), the usual YAML `null` is used, e.g. when the type is `nil | number` you may use `enum: [ 1, 2, ~ ]` or more explicitly/verbosely `enum: [ 1, 2, !!null "" ]`; in non-inline notation you can just omit the value completely, of course.

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[71]Union Type

A union type is used to allow instances of data to be described by any of several types. A union type is declared via a type expression that combines 2 or more types delimited by pipe (`|`) symbols; these combined types are referred to as the union type' s super types. In the following example, instances of the `Device` type may be described by either the `Phone` type or the `Notebook` type:

```
#%RAML 1.0
```

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

```
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:
    type: Phone | Notebook
```

An instance of a union type is valid if and only if it meets all restrictions associated with at least one of the super types. More generally, an instance of a type that has a union type in its type hierarchy is valid if and only if it is a valid instance of at least one of the super types obtained by expanding all unions in that type hierarchy. Such an instance is deserialized by performing this expansion and then matching the instance against all the super types, starting from the left-most and proceeding to the right; the first successfully-matching base type is used to deserialize the instance.

The following example defines two types and a third type which is a union of those two types.

```
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
```

The following example of an instance of type `CatOrDog` is valid:

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

Imagine a more complex example of a union type used in a multiple inheritance type expression:

```
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 ]
```

In this case, type `HomeAnimal` has two super types, `HasHome` and an anonymous union type, defined by the following type expression: `Dog | Cat`.

Validating the `HomeAnimal` type involves validating the types derived from each of the super types and the types of each element in the union type. In this particular case, the processor MUST test that types `[HasHome, Dog]` and `[HasHome, Cat]`are valid types.

If you are extending from two union types a processor MUST perform validations for every possible combination. For example, to validate the `HomeAnimal` type shown below, the processor MUST test the six possible

combinations: `[HasHome, Dog ]`, `[HasHome, Cat ]`, `[HasHome, Parrot]`, `[IsOnFarm, Dog ]`, `[IsOnFarm, Cat ]`, and `[IsOnFarm, Parrot]`.

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

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[72]**Using XML and JSON Schema**

RAML allows the use of XML and JSON schemas to describe the body of an API request or response by integrating the schemas into its data type system.

The following examples show how to include an external JSON schema into a root-level type definition and a body declaration.

```
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
```

A RAML processor MUST NOT allow types that define an XML or JSON schema to participate in type inheritance or specialization, or effectively in any type expression[73]. Therefore, you cannot define sub-types of these types to declare new properties, add restrictions, set facets, or declare facets. You can, however, create simple type wrappers that add annotations, examples, display name, or a description.

The following example shows a valid declaration.

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

The following example shows an invalid declaration of a type that inherits the characteristics of a JSON schema and adds additional properties.

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

Another invalid case is shown in the following example of the type `Person` being used as a property type.

```
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
```

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

A RAML processor MUST be able to interpret and apply JSON Schema and XML Schema.

An XML schema, or JSON schema, MUST NOT be used where the media type does not allow XML-formatted data, or JSON-formatted data, respectively. XML and JSON schemas are also forbidden in any declaration of query parameters, query string, URI parameters, and headers.

The nodes "schemas" and "types", as well as "schema" and "type", are mutually exclusive and synonymous for compatibility with RAML 0.8. API definitions should use "types" and "type", as "schemas" and "schema" are deprecated and might be removed in a future RAML version.

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[74] References to Inner Elements

Sometimes it is necessary to refer to an element defined in a schema. RAML supports that by using URL fragments as shown in the example below.

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

When referencing an inner element of a schema, a RAML processor MUST validate an instance against that particular element. The RAML specification supports referencing any inner elements in JSON schemas that

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

are valid schemas, any globally defined elements, and complex types in XML schemas. There are only a few restrictions:

- Validation of any XML or JSON instance against inner elements follows the same restrictions as the validation against a regular XML or JSON schema.
- Referencing complex types inside an XSD is valid to determine the structure of an XML instance, but since complex types do not define a name for the top-level XML element, these types cannot be used for serializing an XML instance.