API Guideline

背景知识

App Infra对外暴露的服务包括两种形式，一种是UI（用户直接操作管理平台），另外一种是API（用户通过服务账号直接访问服务的接口）。如果Application Infra的产品缺少一个统一的API规范将会出现下面这些问题：

- 第三方开发者调用不同服务的接口时体验不一致
- 前端或者Bff层代码封装困难

由于上面的原因，我们需要定义一个统一的API规范来帮助开发者设计简单、一致易用的API接口。

REST API

本设计指引基于REST API， 并根据现状，增加更多的约定，以期开发者获得更加一致性的预期。例如，我们会通过约定好所有对于资源创建时间的请求参数必须用createAt，通过这样的约定，接口通信的双方都能明白对方想要表达的意思。

REST API简单讲就是可单独寻址的”资源“(API中的“名词”)的“集合”。资源通过资源名称被引用，并通过一组“方法”(也称为“动词”或“操作”)进行控制。（资源+动作）

资源名称

在面向资源的API设计中，资源通常被构建为***资源层次结构***。

示例：

```
https://kubernetes.devops.i.sz.shopee.io/openapi/v1/tenants/{tenantId}/projects/{projectName}
```

如上所示，是一个完整的api。我们可以把它拆分成以下部分:

API 服务名称	api服务模块 (required)	当前api版本 (required)	父资源集合 ID	父资源 ID	子资源集合 ID	子资源 ID
kubernetes.devops.i.sz.shopee.io	/openapi	/v1	/tenants	/{tenantId}	/projects	{projectName}

其中:

`tenants/{tenantId}/projects/{projectName}` 为资源名称。资源名称通常遵循REST网址规则，其行为与网络文件路径非常相似。

- 集合资源: `/tenants` , "集合"是一种特殊的资源，包含相同资源的子资源列表。集合的资源ID称为**集合ID**。
  - 集合资源必须使用复数形式
  - 必须使用首字母驼峰体
  - 应该避免过于笼统的词语，如: values、elements、items、objects、resources... 如果必须使用，请添加限定词，如应该用rowValues而不是values
- 单一资源: `/tenants/{tenantId}` , 单一资源名称由**集合ID**和**资源ID**构成，按分层方式组织，并以正斜杠/ 分隔。
- 有层级的集合资源:`/tenants/{tenantId}/projects` , 该示例展示了一个有父子层级关系的资源如何确定其资源名称。
- 有层级的单一资源: `/tenants/{tenantId}/projects/{projectName}` , 这里注意到我们分别用Id和Name来描述两种单一资源。在该api的实际使用中，tenantId为一个数字id，而projectName为project的实际名称。我们希望尽可能的通过更符合语义的名称字段，使接口更加一目了然。(如果需要tenant的名称字段，则应该通过响应正文`displayName`返回)

标准方法

我们约定的标准方法只包含以下几种:

标准方法	HTTP 映射	HTTP 请求正文	HTTP 响应正文
List	GET <collection URL>	无	资源列表
Get	GET <resource URL>	无	资源
Create	POST <collection URL>	资源	资源
Update	PUT or PATCH <resource URL>	资源	资源
Delete	DELETE <resource URL>	不适用	空

大多数情况下，应该尽量使用标准方法。它们的含义是简单并且明确的，最不会造成误解。

(warn: 使用标准方法的api禁止在请求网址路径上出现任何方法(动词verbs)。如：`https://kubernetes.devops.i.sz.shopee.io/openapi/v1/tenants/{tenantId}/delete`为不符合规范的api endpoint。方法应该映射http方法上)

List

- **List** 方法必须使用 HTTP GET动词
- 其对应的资源名称必须是**集合资源**
- 不能有请求正文
- 其他剩余的请求消息字段应该映射到网址查询参数即query中

示例:

api:

`https://kubernetes.devops.i.sz.shopee.io/openapi/v1/tenants`

请求正文:

无

响应正文:

```
{
  code: 0,
  message: "string",
  data: {
    "tenants": [
      {
        "id": 0,
        "name": "string",
        "detail": "string",
        "createdAt": "2021-07-27T08:10:44.270Z",
        "updatedAt": "2021-07-27T08:10:44.270Z"
      }
    ],
    "total": 0
  }
}
```


Get

- 网址路径就是对应的资源名称，请求的必须是一个**单一资源**
- 其他剩余请求消息字段应该映射到网址查询参数即query中
- 不能有请求正文
- 返回的资源应该映射到响应正文

示例：

api:

`https://kubernetes.devops.i.sz.shopee.io/openapi/v1/tenants/1001`

请求正文:

无

相应正文:


```
{
  code: 0,
  message: "string",
  data: {
    {
      "id": 0,
      "name": "string",
      "detail": "string",
      "createdAt": "2021-07-27T08:10:44.270Z",
      "updatedAt": "2021-07-27T08:10:44.270Z"
    }
  }
}
```


Create

- 网址路径应该具有**集合资源**，以指定在哪个集合资源下创建新的**单一资源**
- 包含资源的请求消息必须映射到请求正文中
- 其他剩余的请求消息字段应该映射到网址查询参数即query中
- 返回的资源应该映射到响应正文

示例：

`https://kubernetes.devops.i.sz.shopee.io/openapi/v1/tenants`

请求正文：


```
{
  "name": "string",
  "detail": "string"
}
```


响应正文:

```
{
  code: 0,
  message: "string",
  data: {
    {
      "id": 0,
      "name": "string",
      "detail": "string",
      "createdAt": "2021-07-27T08:10:44.270Z",
      "updatedAt": "2021-07-27T08:10:44.270Z"
    }
  }
}
```

Update

- 如果是更新资源的部分字段，使用HTTP PATCH
- 如果是完整更新资源，使用HTTP PUT
- 包含资源的请求消息必须映射到请求正文
- 其他剩余的请求消息字段应该映射到网址查询参数即query中
- 响应消息必须是更新的资源本身

示例:

`https://kubernetes.devops.i.sz.shopee.io/openapi/v1/tenants/{tenantId}/projects/{projectName}/applications/{appName}/deployments/{deployName}/resources`

请求正文:

```
{
  "appInstanceName": "string",
  "clusterName": "string",
  "phases": [
    {
      "phase": "string",
      "resource": [
        {
          "container": "string",
          "cpuLimit": 0,
          "memLimit": 0
        }
      ]
    }
  ]
}
```

响应正文:

```
{
  code: 0,
  message: "string",
  data: {
    {//update deploy response}
  }
}
```

Delete

- 网址路径就是对应的资源名称
- 其他剩余请求消息字段应该映射到网址查询参数即query中
- 不能有请求正文
- 如果 Delete 方法立即移除资源，则应该返回空响应
- 如果 Delete 方法仅将资源标记为已删除，则应该返回更新后的资源。
- 对 Delete 方法的调用在效果上应该是幂等的，但不需要产生相同的响应。任意数量的 Delete 请求都应该导致资源（最终）被删除，但只有第一个请求会产生成功代码。

示例:

`https://kubernetes.devops.i.sz.shopee.io/openapi/v1/tenants/1001`

请求正文:

无

相应正文:

```
{
  code: 0,
  message: "string",
  data: {}
}
```

自定义方法

自定义方法是指 5 个标准方法之外的 API 方法。这些方法应该仅用于标准方法不易表达的功能。通常情况下，API 设计者应该尽可能优先考虑使用标准方法，而不是自定义方法。

对于自定义方法，它们**应该**使用以下通用 HTTP 映射：

`https://service.name/v1/some/resource/name:customVerb`

注意：使用 : 而不是 / 将自定义动词与资源名称分开以便支持任意路径。

- 自定义方法一般应该使用 HTTP POST 动词，因为该动词具有最灵活的语义，但作为替代 get 或 list 的方法（如有可能，可以使用 GET）除外。
- 自定义方法不应该使用 HTTP PATCH，但可以使用其他 HTTP 动词。
- 接收与自定义方法关联的资源或集合的资源名称的请求消息字段应该映射到网址路径。
- 网址路径**必须**以包含冒号（后跟自定义动词）的后缀结尾。

示例

`Cancel`	`:cancel`	`POST`	取消一个未完成的操作，例如 `operations.cancel`。
`BatchGet`	`:batchGet`	`GET`	批量获取多个资源。
`Move`	`:move`	`POST`	将资源从一个父级移动到另一个父级，例如 `folders.move`。
`Search`	`:search`	`GET`	List 的替代方法，用于获取不符合 List 语义的数据，例如 `services.search`。
`Undelete`	`:undelete`	`POST`	恢复之前删除的资源，例如 `services.undelete`。

标准字段

- name：相应的资源名
- displayName: 资源的现实名称
- parent：父级资源名
- createAt：创建资源的时间戳
- updateAt：更新资源的时间戳
- deleteAt：删除资源的时间戳
- expireAt：资源过期的时间戳
- stratAt: 标记某个时间段开始的时间戳
- endAt: 标记某个时间段结束的时间戳
- filterBy: List方法的过滤参数
- searchBy: 用于搜索方法
- offset: List方法分页参数，指定查询的开始位置
- limit: List方法分页参数，指定查询的长度
- total: List方法返回的资源总数（与分页无关的资源总数）
- orderBy: 指定List请求的结果排序

orderBy

`orderBy=name`

可选参数。

用于指示返回数据的排序顺序和排序方向的一系列指标和维度。

- 排序顺序由所列指标和维度指定，其中指标和维度的排序优先级从左到右依次降低。
- 排序方向默认为升序。您可以通过为所请求的字段添加desc后缀将排序方向改为降序。

通过对查询结果排序，您可以解决许多数据方面的疑问。

使用`orderBy`参数时，请注意以下事项：

1. 默认情况下，字符串将按 en-US 语言区域的字母顺序升序排序。 默认情况下，数字将按数值顺序升序排序。 默认情况下，日期将按日期顺序升序排序。
2. 如果列表查询返回的items数组的成员对象有多级结构，如 `[{ foo: { bar: 1} }, { foo: { bar: 2} }`，需求对次级以上的属性如 `bar` 进行排序，则可以使用 keyPath，如 `orderBy=foo.bar`

filterBy

`filterBy=name==jack,name==tom`

可选参数。

`filterBy`查询字符串参数可以限制由您的请求返回的数据。要使用`filterBy`参数，请提供要用作过滤依据的维度或指标，然后紧接着提供过滤条件表达式。

注：如果列表查询返回的items数组的成员对象有多级结构，如 `[{ foo: { bar: 1} }, { foo: { bar: 2} }`，需求对次级以上的属性如 `bar`进行过滤，则可以使用 keyPath，如 `filterBy=foo.bar==1`

通过对查询进行过滤，能够对要（或不要）包含在结果中的行加以限制。结果中的每行都要针对过滤条件进行测试：如果满足过滤条件要求，该行会保留下来；如果不能满足过滤条件要求，该行会被丢弃。

- **网址编码：**发起请求时请对过滤条件运算符进行URL编码。
- **维度过滤：**过滤操作发生在对任何维度进行汇总之前，确保返回的指标只代表相关维度的总和。
- **指标过滤：**对指标的过滤操作发生在对指标进行汇总之后。
- **有效组合：**您可以过滤非查询组成部分的维度或指标，条件是请求中的所有维度/指标与过滤条件是有效组合。

过滤条件语法：

单个过滤条件使用以下格式：

`key <operator> value`

在这一语法中：

- key - 过滤所依据的维度或指标的名称。例如：name会依据name指标进行过滤。
- operator - 定义要使用的过滤条件匹配的类型。维度或指标分别有相对应的特定运算符。
- value - 说明要包含在结果中或从结果中排除的值。表达式使用正则表达式语法。

过滤条件运算符

维度对应的过滤条件运算符有六种，指标对应的过滤条件运算符也有六种。运算符必须经过网址编码，才能纳入网址查询字符串。

指标过滤条件

运算符	说明	网址编码格式	示例
==	等于	%3D%3D	返回名字为“jack”的所有用户：`filterBy=name==jack`
!=	不等于	!%3D	返回名字为“jack”的所有用户：`filterBy=name!=jack`
>	大于	%3E	返回年龄大于25岁的所有用户：`filterBy=age>jack`
<	小于	%3C	返回年龄小于25岁的所有用户：`filterBy=age<jack`
>=	大于或等于	%3E%3D	返回年龄大于等于25岁的所有用户：`filterBy=age>=jack`
<=	小于或等于	%3C%3D	返回年龄小于等于25岁的所有用户：`filterBy=age<=jack`

维度过滤条件

	运算符	说明	网址编码格式	示例
1	==	完全匹配	%3D%3D	对城市为“Irvine”的指标进行汇总：`filterBy=city==Irvine`
2	!=	非匹配	!%3D	对城市不是“Irvine”的指标进行汇总：`filterBy=city!=Irvine`
3	=@	包含子字符串	%3D@	对城市包含“York”的指标进行汇总：`filterBy=city=@York`
4	!@	不包含子字符串	!@	对“city”不包含“York”的指标进行汇总：`filterBy=city!@York`
5	=~	包含与正则表达式匹配的内容	%3D~	对“city”以“New”开头的指标进行汇总：`filterBy=city=~^New.*`（^将字符串的开头部分限定为某一格式。）

6	!~	与正则表达式不匹配	!~	对“city”不以“New”开头的指标进行汇总：`filterBy=city!~New.*`

过滤条件表达式

下面是过滤条件表达式要遵守的一些重要规则：

- **网址预留字符** - &等字符必须以常规方式进行网址编码。
- **预留字符** - 如果英文分号和英文逗号出现在表达式中，那么必须加上反斜杠进行转义。 英文分号`\;` 英文逗号`\,`
- **正则表达式** - 您还可以在过滤条件表达式中使用正则表达式，即使用=~和!~运算符。其语法类似于 Perl 正则表达式，不过有如下额外规则：
  - 最长 128 个字符 - 如果正则表达式长度超过 128 个字符，服务器会返回`400 Bad Request`状态代码。
  - 区分大小写 - 正则表达式匹配区分大小写。

合并过滤条件

过滤条件可以使用 OR 和 AND 布尔值逻辑进行合并。这样可以有效地扩展过滤条件表达式 128 个字符的限制。

OR

OR运算符使用英文逗号 (`,`) 进行定义。它的运算优先级高于AND运算符，并且在同一个表达式中不能用于组合维度和指标。

示例：（均须经过网址编码）

国家/地区为（“United States”或 [OR]“Canada”）：`country==United States,country==Canada`

AND

AND 运算符使用英文分号 (`;`) 进行定义。它的运算优先级低于 OR，并且在同一个表达式中可以组合维度和指标。

示例：（均须经过网址编码）

国家地区为“United States”并且 (AND) 浏览器为 Firefox：`country==United States;browser==Firefox`

国家/地区为“United States”并且 (AND) 语言不以“en”开头：`country==United States;language!~^en.*`

searchBy

`searchBy=shopee`

可选参数。

在某些场景下，你可能希望对数据进行全字段或者约定好的多个字段应用搜索，例如常见的模糊搜索。如果使用`filterBy`会显得非常繁琐。这个时候，`searchBy` 就非常适合这个场景。

响应正文

响应需要注意两部分：状态码 和 Response Body。

- 状态码: 状态码严格按照 Http 的状态码。
- 响应体：包括 **正确响应** 和 **错误响应** 。

响应体格式

为了更统一地处理返回数据和错误，我们规范了响应正文的标准结构:

正确响应

```
{
  code: 0, //
  message: "string", //
  data: { //

  }
}
```

```
{
  code: 110202, // ,
  status: 403,  //
  reason: "PEMISSION_DENIED",
  message: "[PEMISSION_DENIED] no permission to check", //
  metadata: { }  //
}
```

code 和 reason: 错误码。code 和 reason 通过 enum 定义，一一对应。如 PERMISSION_DENIED = 110202

message: 可读的错误信息在message字段返回。请尽量确保message的可读性。

data: 如果请求成功，需要通过响应正文返回的数据，通过data字段返回。如果不是必要，不要在data字段下再包裹一层结构，避免接收者需要通过例如 data.data 这样去调用。

错误码设计

错误由 6 位整型数组成，每两位表示一个域。包括三部分：

- 服务名：前两位
- 模块名： 中间两位
- 具体错误码：后两位

服务名

服务名由前两位组成，从 10 开始。已用服务名：

10 ： 表示 k8s 平台的 auth 服务

11 ： 表示 k8s 平台的 platformapi 服务

12 ： 表示 network 的 gateway 服务

模块名

模块名为中间两位，从 00-99，自行定义。可以按照根据自己服务的 domain 来定义，如 user 模块为01。

具体错误码

最后两位定义具体的错误码，从 00-99 可自行定义。

> 建议 错误码和 reason 定义为一一对应的枚举值，这样第三方可以选择使用数值的 code 也可以选择使用字符表示的 reason。