Dao Data Engine REST API Reference

概述

当前版本的 Dao Data Engine 底层采用 arangodb 作为 Graph Database。Dao Data Engine REST API 采用arangodb提供的 Foxx Microservice 作为框架进行开发。DataEngine API 当前版本没有提供鉴权认证功能，默认 DataEngine 运行在内网环境中。后续会加入对 API 调用的鉴权管理，将会采用 Foxx 提供的 API KEY 和 Session 机制。

通用部分

URL模板

DE REST API 中 URL 模板如下：

```
http://主机信息/_db/数据库名/API信息
```

- 在默认安装的开发环境下，主机信息通常是：`localhost:8529`
- 默认数据库为：`_system`

其中 API 信息部分又采用如下格式模板：

```
/版本号/API大类/调用信息
```

对于当前版本的Graph大类 API，具体的 URL 模板为：

```
/v1/g/stub/key/path?参数表
```

- `v1` 是当前API版本号：1.0
- `g` 是当前API的分类：**Graph**
- `stub/key` 用于在 Graph 结构中定位一个**锚点**，从而把网状结构转化为树状结构。然后可以以此**锚点**为根，进行树状遍历，找到最后的叶节点。
  - stub = `_uuid`，key 的值为 nodes 中的 `uuid` 值。所有资源的全局唯一标示。
  - stub = `_key`, key 的值就是 nodes collection 中的 `_key` 值，在该 collection 中唯一。
  - stub = 外部 collection 的名称，`stub/key` 就是当前数据库中其他 collection 中某个 document 的 `_id`。DataEngine 通过 nodes 中的 `ref` 字段索引到其在 Graph 中的位置。
- `path` 部分是以 Graph Edge Name 作为路径的寻址方式。比如：`/stub/key/trunk/branch/leaf`
- 参数表目前版本就提供了一个 `s` 参数，具体取值如下：
  - `s=..`，当前操作是对path指向的Graph Edge数据，即DataEngine中的link数据。
  - `s=.`，当前操作的是graph中指向原始数据的node，而不是原始数据本身。
  - `s=collectionName`，指定数据源名称是某个collection。通常用于在指定数据源中创建新的数据。
  - `s=selection`，用于HTTP GET中选择性的获取数据而不是对象全部的数据。例如某个数据对象有大量的字段，我们只要求返回name, age, sex三个字段信息，参数为：`s=name,age,sex`。注意字段间用“,”分隔。

HTTP Body数据格式

格式统一起见，DE REST API中body统一采用JSON数据格式，即HTTP `Content-Type: application/json`。API中需要body内容的HTTP Method有两个：POST和PUT。

arangodb foxx框架中主要采用 joi做数据验证。当前版本开发中对body字段没有做太多验证，只要是json格式即可。

HTTP GET: 获取数据

Header格式

```
GET /v1/g/:root/:key/*path?s=selection
```

- `v1`: 版本号 version 1.0
- `g`： 类别 Graph
- `:root`: Graph中绝对寻址某个节点用于以此为Root的相对位置寻址。
  - `_uuid`: 用资源的全局唯一ID定位root。对应Graph.nodes.uuid值。
  - `_key`: 用Graph.nodes中的唯一标示 `_key` 寻址。
  - `collectionName`: 用原始数据寻址， `:root/:key` 就是原始数据的 `document-handle`
- `:key`: 参考 `:root` 说明，此参数对应不同 `:root` 参数，有不同的取值。
- `*path`: 使用link name进行寻址。 `*path` 代表后面是一个层次序列，例如： `branch1/branch2/leaf` 。
  - 特殊取值“.”: 当前节点只有一个out link时，可以不用指定名字，比如： `qrcode/./model` 。需要注意的是，如果有多个*out link*时， `.` 等同于 `any` ，是第一个找到的*out link*。 `.` 可以进行重复盲查，比如： `log/first/./././error` ，可以用于单线列表式的数据结构快速寻址。
  - 特殊取值“..”: 当前节点的后向寻址，而不是简单的回退到上一级节点。比如： `父亲/儿子/../母亲` ， `儿子` 节点有两个*in link*， `父亲` 和 `母亲` ， `..` 操作就是寻找*link name*为 `母亲` 的*in link*。 `..` 也可以级联进行寻址： `张三/儿子/../../张家` ，但是这种级联不是简单回溯，而是等同于 `张三/儿子/.././../张家` ，是对 `儿子` 后向寻址任意一个*in link*（可能是 `父亲` 也可能是 `母亲` ），然后再从*父母*后向寻址到 `张家` 。
- `s=selection`: 这个参数用于指定GET想要返回的内容。
  - `s` 参数不存在或者为空时: 完整获取数据对象。这是默认的情况。
  - `s=.`: 获取当前的节点数据，而不是所指向的原始数据。
  - `s=..`: 获取前序节点与当前节点的Link数据。
  - `s=选取的属性列表`： 选择性的获取数据而不是对象全部的数据。例如某个数据对象有大量的字段，我们只要求返回name, age, sex三个字段信息，参数为： `s=name,age,sex` 。注意字段间用“,”分隔。
- 特殊格式：当 `*path` 部分不存在时，表示采用绝对寻址方式直接对节点操作。与上述带有 `*path` 格式情况对比，不需要相对 `path` 寻址，不支持 `s=..` 获取Link数据的参数，其他都一样。

返回值

- 200：找到指定的数据并返回。
  - 请求完整原始数据时，以数据原本格式返回原始数据，可以是普通字符串格式，也可以是JSON格式。
  - 选择性返回数据： `s=_key,name`
    - `{"_key":"gary","name":"Gary"}`
  - 请求节点数据时，返回例示：
    - `{"_id":handler,"_key":key,"ref":source_id,"type":type,"data":data}`
  - 请求连接数据时，返回结果实例：

```
{
"_id":"work_de_links/195801828559",
"_key":"195801828559",
"_rev":"195801828559",
"_from":"work_de_nodes/root",
"_to":"work_de_nodes/195801631951",
"name":"alice"
}
```

- 404: 在此API代码中，遇到任何异常统一返回如下错误:
  - `{"error":"The route is not viable."}`

HTTP POST: 创建数据

Header格式

`POST /v1/g/:root/:key/*path/leaf?s=source`

- `v1`: 版本号 version 1.0
- `g`： 类别 Graph
- `:root`: Graph中绝对寻址某个节点用于以此为Root的相对位置寻址。
  - `_uuid`: 用资源的全局唯一ID定位root。对应Graph.nodes.uuid值。
  - `_key`: 用Graph.nodes中的唯一标示 `_key` 寻址。
  - `collectionName`: 用原始数据寻址， `:root/:key` 就是原始数据的 `document-handle`
- `:key`: 参考 `:root` 说明，此参数对应不同 `:root` 参数，有不同的取值。
- `*path`: 使用link name进行寻址。 `*path` 代表后面是一个层次序列，例如： `branch1/branch2/branch3` 。
- `leaf`: 待新建的叶节点**Link**名称。
- `s=source`: 这个参数用于指定所创建数据对象的类型或位置。
  - `s` 参数不存在或者为空时: 这是默认的情况，与下述 `s=.` 等效。
  - `s=.`: 使用**body**数据创建新的**node**，同时创建以 `leaf` 为名字的**Link**，从 `*path` 指向的节点连接新建的节点。注：此参数不创建外部数据，但可以创建 `type="_self"` 的内部数据： `data=内部数据` 。
  - `s=..`: 创建以当前 `*path` 为起始节点、以**body**参数中指向的节点为终结点、以**leaf**为名称的链接。这种情况下只会创建链接，而不会创建节点和原始数据。
  - `s=collection`： 这种情况下会使用**body**中的数据，在外部**collection**中创建数据对象，同时创建一个指向该新建数据对象的**node**，并创建一个从 `*path` 节点指向新的**node**的名称为**leaf**的链接。
- 特殊格式：下面的操作直接针对DataEngine底层数据操作，通常不推荐。执行时会严格检查数据对象是否与Links、Nodes的定义一致。
  - “ `POST /v1/g/._` ”：使用**body**数据直接创建新的节点。新节点可以在内部存放数据，也可以指向外部数据。本操作不会创建Link和外部数据。
  - “ `POST /v1/g/.._` ”：使用**body**数据直接创建新的链接，创建过程不产生新的节点和外部数据。

Body数据格式

- 外部数据对象：当前版本仅支持同一个数据库的其他 `collection` 中的 `document` 。**body**采用application/json数据格式。其中作为该数据对象键值的 `_key` 可以自行定义，也可以由系统自动产生。只是 `_key` 的定义要符合**arangodb**的命名规则。
- **node**数据对象：采用**DataEngine**定义的数据结构，如下所示。
  - `{"_key": Key, "ref": Ref, "type": Type, "uuid": UUID, "data": Data}`
- **link**数据对象：这个数据对象不采用**link**内部的数据结构，而是采用 `{type: Key}` 格式，具体有如下几种形式：
  - `{"_uuid": UUID}`: 被连接节点的UUID值。
  - `{"_key": Key}`: 被连接节点的**document key**
  - `{"_ref": "Collection/Key"}`: 被连接节点的外部引用数据对象的**document-handle**。
- 特殊格式“ `POST /v1/g/.._` ”下**link**数据对象则采用内部数据结构，即：
  - `{"_key": Key, "_from": From, "_to": To, "name": Name}`

返回值

- 200：正确创建数据对象，并返回摘要信息。如：
  - `{"_id":"work_de_links/202120627671","_rev":"202120627671","_key":"202120627671"}`
- 404: 在此API代码中，遇到任何异常统一返回如下错误:

- `{"error":"The route is not viable."}`

HTTP PUT: 更改数据

Header格式

`PUT /v1/g/:root/:key/*path?s=source`

- `v1`: 版本号 version 1.0
- `g`：类别 Graph
- `:root`: Graph中绝对寻址某个节点用于以此为Root的相对位置寻址。
  - `_uuid`: 用资源的全局唯一ID定位root。对应Graph.nodes.uuid值。
  - `_key`: 用Graph.nodes中的唯一标示 `_key` 寻址。
  - `collectionName`：用原始数据寻址， `:root/:key` 就是原始数据的 `document-handle`
- `:key`：参考 `:root` 说明，此参数对应不同 `:root` 参数，有不同的取值。
- `*path`: 使用link name进行寻址。 `*path` 代表后面是一个层次序列，例如： `branch1/branch2/branch3` 。
- `s=source`: 这个参数用于指定所创建数据对象的类型或位置。
  - `s=.`: 使用**body**数据更新当前 `*path` 指向的**node**数据。此操作仅改变**node**内部数据，不会更改外部的原始数据和链接数据。
  - `s=..`: 使用**body**中的数据更新当前 `*path` 末端的**link**数据。需要注意的是，由于**link**内部数据中只有 `name` 字段能够被修改，也就是说只有 `{"name": newName}` 才真正起作用。
  - `s` 未设置或者为空：这是默认情况，使用**body**中的数据更新原始数据。**body**中的数据会更新原始数据已有字段、增加原始数据中没有的字段。注：在**node**内部**data**字段保存的数据可以是任意形式。
- 特殊格式：当 `*path` 部分不存在时，表示采用绝对寻址方式直接对节点操作。与上述带有 `*path` 格式情况对比，不需要相对 `path` 寻址，不支持 `s=..` 操作Link数据的参数，其他都一样。即当 `s=.` 时更新**Graph Node**数据；当 `s` 不存在时，更新原始数据。

Body数据格式

- 外部数据对象：当前版本仅支持同一个数据库的其他 `collection` 中的 `document` 。**body**采用application/json数据格式。其中作为该数据对象键值的 `_key` 可以自行定义，也可以由系统自动产生。只是 `_key` 的定义要符合**arangodb**的命名规则。
- 内部数据对象：直接在 `node.data` 中存放数据时，**body**可以为任意格式，只不过当 `node.data` 和 `body` 数据都是**object**类型时，执行 `union update` 操作；是其他类型数据时，直接执行覆盖替换操作。
- **node**数据对象：采用**DataEngine**定义的数据结构，如下所示。不过由于系统限制， `_key` 是无法被修改的。
  - `{"_key": Key, "ref": Ref, "type": Type, "uuid": UUID, "data": Data}`
- **link**数据对象：这个数据对象采用**link**内部的数据结构，但是由于系统限制，整个**link**内部数据中，只有 `name` 是可以被更改的。因此有效的数据格式是 `{"name": newName}` 。

返回值

- 200：正确创建数据对象，并返回摘要信息。如下所示。其中 `_rev` 与 `_oldRev` 是更新前后的**revision**号。
  - `{"_id":"male/jerry","_rev":"214392413514","_oldRev":"202177512919","_key":"jerry"}`
- 404: 在此API代码中，遇到任何异常统一返回如下错误:
  - `{"error":"The route is not viable."}`

HTTP DELETE: 删除数据

Header格式

`PUT /v1/g/:root/:key/*path?s=source`

- `v1`: 版本号 version 1.0
- `g`： 类别 Graph
- `:root`: Graph中绝对寻址某个节点用于以此为Root的相对位置寻址。
  - `_uuid`: 用资源的全局唯一ID定位root。对应Graph.nodes.uuid值。
  - `_key`: 用Graph.nodes中的唯一标示 `_key` 寻址。
  - `collectionName`: 用原始数据寻址， `:root/:key` 就是原始数据的 `document-handle`
- `:key`: 参考 `:root` 说明，此参数对应不同 `:root` 参数，有不同的取值。
- `*path`: 使用link name进行寻址。 `*path` 代表后面是一个层次序列，例如： `branch1/branch2/branch3` 。
- `s=source`: 这个参数用于指定所创建数据对象的类型或位置。
  - `s=..`: 仅仅删除当前 `*path` 最后的叶节点与前序节点间的链接。
  - `s=.`: 删除当前 `*path` 指向的节点，同时自动删除与该节点相连的所有链接。此操作不影响外部原始数据，但是会同时删除内部数据。
  - `s` 未设置或者为空：这是默认情况，首先会删除 `*path` 指向的节点及该节点关联的所有链接，然后如果该**node**指向外部数据，则会同时删除外部数据对象。
- 特殊格式：当 `*path` 部分不存在时，表示采用绝对寻址方式直接对节点操作。与上述带有 `*path` 格式情况对比，不需要相对 `path` 寻址，不支持 `s=..` 删除Link数据的参数，其他都一样。即当 `s=.` 时删除**Graph Node**及相关链接；当 `s` 不存在或者为空时，同时还会删除原始数据。

返回值

- 200：成功删除数据对象后，返回
  - `{"success": true`
- 404: 在此API代码中，遇到任何异常统一返回如下错误:
  - `{"error":"The route is not viable."}`