HTTP RESTful规范

概述

API作为服务对外暴露的接口协议，应该保持稳定且遵守一定的标准。提供统一、规范的API，能方便其他系统的接入，对外提供一致的体验，有助于与其他系统集成。目前服务对外的API，大多数遵循HTTP协议，本规范从HTTP协议入手，给出RESTFul设计风格的一些规范和建议。

HTTP的RESTful API设计的主要原则

- REST API 是围绕资源设计的，资源是客户端可以访问的任何类型的对象、数据或服务。
- 资源有一个标识符(identifier)，它是唯一标识该资源的 URI。例如，特定客户订单的 URI 可能是：

```
https://adventure-works.com/orders/1
```

- 客户端通过交换资源表示与服务交互。许多 Web API 使用 JSON 作为交换格式。例如，对上面列出的 URI 的 GET 请求可能会返回此响应正文：

```
{"orderId":1,"orderValue":99.90,"productId":1,"quantity":1}
```

- REST API 使用统一的接口，这有助于将客户端和服务实现解耦。对于基于 HTTP 构建的 REST API，统一接口包括使用标准 HTTP 动词对资源执行操作。最常见的操作是 GET、POST、PUT、PATCH 和 DELETE。
- REST API 使用无状态请求模型。HTTP协议本身是无状态的，在各个请求之间保持状态是不可行的，由于请求的无状态，客户端也不应该依赖HTTP执行的顺序。唯一存储信息的地方是资源本身，每个请求都应该是一个原子操作。此约束使 Web 服务具有高度可扩展性，因为无需在客户端和特定服务器之间保留任何关联。任何服务器都可以处理来自任何客户端的任何请求。

以资源为中心来设计API

采用RESTFul的风格，API设计以资源以资源为中心来设计API接口。通常URI采用名词而非动词。资源的接口跟服务内部的实现尽量保持独立，一个接口访问的数据可能在数据库中几张表中。RESTFul的标准，不限定客户端用任何方式发送API请求，即客户端可以用任何语言来发送HTTP请求和解析HTTP响应。关注web API公开的实体。例如在一个电子商务系统中，通过HTTP POST请求来创建订单。HTTP响应返回订单是否创建成功。URI基于名词（资源）而不是动词（对资源的操作）。

```
https://adventure-works.com/orders       //
https://adventure-works.com/create-order //
```

在接口中避免使用具体的名称，推荐使用通用名称加上具体类目的形式。例如/orders/1获取第1个订单的数据。在某些情形下，我们也需要/orsers/1/consumers这种形式，但不推荐写太多的嵌套，例如/orders/1/consumers/3/orders，这种后期维护较困难。

通常URI集合采用复数形式，返回某个资源的集合，如下所示，返回所有订单的集合。

```
https://adventure-works.com/orders
```

而对于单个实体，采用参数化的URI路径，一般web框架也支持这种写法。例如请求某个具体的用户，path定义的路由为/customers/{id}。

使用HTTP的方法来定义API的操作

- GET 获取特定资源状态，返回的HTTP响应中包括请求资源的详情。
- POST 创建特定的资源，请求的HTTP响应包含需要创建资源的详情。
- PUT 创建或者更改特定的资源，请求的HTTP响应包含创建/修改的资源。
- PATCH 修改特定资源的部分，请求的HTTP响应包含修改的部分。
- DELETE 删除特定的资源。

特定请求的效果取决于资源是集合还是单个项目。下表总结了假设的电子商务服务下的RESTful的一些特殊约定，作为参考示例。

资源	POST	GET	PUT	DELETE
/customers	创建一个新的客户	查询所有客户	批量更新客户	删除所有客户
/customers/1	不允许的操作	查询客户1的详情	更新客户1的详情（如果存在）	删除客户1
/customers/1/orders	为客户1创建订单	查询客户1的所有订单	批量更新客户1的订单	删除客户1的所有订单

POST PUT与PATCH解释

- POST用于创建资源，服务器为新的资源分配URI并返回给客户端。通常POST也用于创建一系列资源的集合。
- PUT用于创建或者更新资源，客户端指定资源的URI，请求的正文包含资源的完整表示。如果有此URI存在，则替换它。一般不用于创建资源的集合。对于存在的资源，就更新。对于不存在的资源就创建。
- PATCH用于对现有资源进行部分更新。客户端指定资源的URI。请求正文应用于资源的一组更改。

API版本

API设计添加上版本号，版本号一般有以下2种形式：

- 将API版本号放到URL中，例如https://adventure-works.com/api/v1/consumers
- 将API版本号放到URL的query参数中，例如https://adventure-works.com/api/consumer?version=1

URLAPI

schema

URL的设计，应遵循一定的规则，使接口容易阅读和使用，“/”分隔符用于划分资源层级。对于RESTFul来说，“/”用于分割资源，避免在结尾添加“/”。

连字符使用“-”，例如consumer-id可读性较好，是推荐的方式，而consumer_id不推荐。

URL建议使用小写字母的形式。

URL中不要包括文件的扩展形式，例如.json,对类型信息有特殊要求，建议放在query参数或者请求的Accept中，例如otype=json或者Accept: application/json。

URL结构

URL需要便于阅读和组织，对于未来云，各个产品可以通过URL的path来区分。例如容器平台的前端可以是/k8s/k8s-fe，对于后端可以是/k8s/k8s-be。

URL长度

尽管HTTP/1.1没有定义请求行的长度，但是为了兼容性和诸多客户端能正确处理URL，不建议URL太长，根据stackoverflow的讨论，建议URL长度不要超过2000个字符。讨论参考这里https://stackoverflow.com/questions/417142/what-is-the-maximum-length-of-a-url-in-different-browsers/417184#417184

幂等性

HTTP中定义的一些方法，有幂等性，即请求一次和多次是一样对请求的资源有相同的作用。下表给出了使用HTTP方法的幂等性。

方法	描述	幂等性
GET	获取一个对象的值	是
PUT	创建一个新对象或者替换一个对象	是
DELETE	删除一个对象	是
POST	创建一个新对象	否
HEAD	返回一个GET响应的元信息。一般支持GET的API也需要支持HEAD方法	是
PATCH	修改一个对象的部分值	否
OPTIONS	获取一个请求的信息	是

返回格式类型

通常情况下，响应的返回格式为json数据。对于某些需要特殊格式的响应，服务端需要根据请求的格式正确设置返回体的格式。如果请求没有指定响应的格式，默认返回json形式，对于其他格式，在响应头中正确设置媒体的格式。为了节省带宽，服务端应该尽量返回必要的数据，减少无用数据。

参考的响应格式

默认响应体反馈json格式的数据。不同的服务，响应格式字段会根据实际场景遵循不同的格式。这里参考微软的建议，给出参考的返回格式的字段。

```
{
  "code""",
  "message": ""
  "details"[
      #
  ]
  #
}
```

用户登录和鉴权

用户在未登录时，不能完成一些需要权限操作的请求。在用户登录后，需要将登录状态保持，并在后续的请求中传递登录状态。存储登录状态一般有使用session的方式或者cookie的方式。服务端可以采用jwt签发token，也可以采用oauth2的方式对资源进行限制权限。

支持大数据分块传输

针对大型的二进制文件，API应该支持应支持Accept-Ranges头，客户端可以使用HEAD方法获取文件的信息，并在接下来的GET请求中使用Accept-Ranges获取分块数据。例如

```
HEAD https://adventure-works.com/products/10?fields=productImage HTTP/1.1


HTTP/1.1 200 OK

Accept-Ranges: bytes
Content-Type: image/jpeg
Content-Length: 4580


GET https://adventure-works.com/products/10?fields=productImage HTTP/1.1
Range: bytes=0-2499
```

状态码

在HTTP应答中，状态码定义了服务端是否正常完成、出现错误等。客户端根据状态码做出进一步处理。HTTP状态码分成以下几个区间。我们应该按照RFC规范设定返回的状态码。

- 2XX：请求正常处理
- 3XX：重定向，请求的资源位置发生变化
- 4XX：客户端发送的请求有错误
- 5XX：服务端出现错误

GET方法

成功的 GET 方法通常会返回 HTTP 状态代码 200 (OK)。如果找不到资源，该方法应返回 404（未找到）。

POST方法

如果 POST 方法创建新资源，则返回 HTTP 状态代码 201（已创建）。新资源的 URI 包含在响应的 Location 标头中。响应正文包含资源的表示。

如果该方法进行了一些处理但没有创建新资源，则该方法可以返回 HTTP 状态代码 200 并将操作的结果包含在响应正文中。或者，如果没有要返回的结果，该方法可以返回没有响应正文的 HTTP 状态代码 204（无内容）。

如果客户端将无效数据放入请求中，服务器应返回 HTTP 状态码 400（Bad Request）。响应正文可以包含有关错误的附加信息或指向提供更多详细信息的 URI 的链接。

PUT方法

如果 PUT 方法创建了一个新资源，它会返回 HTTP 状态代码 201（已创建），就像 POST 方法一样。如果该方法更新现有资源，则返回 200（正常）或 204（无内容）。在某些情况下，可能无法更新现有资源。在这种情况下，请考虑返回 HTTP 状态代码 409（冲突）。

考虑实现批量 HTTP PUT 操作，该操作可以批量更新集合中的多个资源。PUT 请求应指定集合的 URI，请求正文应指定要修改的资源的详细信息。

PATCH方法

通过 PATCH 请求，客户端以*补丁文档*的形式向现有资源发送一组更新。服务器处理补丁文档以执行更新。补丁文档并未描述整个资源，仅描述了一组要应用的更改。PATCH 方法的规范（RFC 5789）没有定义补丁文档的特定格式。格式必须从请求中的媒体类型推断出来。

JSON 可能是 Web API 最常见的数据格式。有两种主要的基于 JSON 的补丁格式，称为JSON 补丁和JSON 合并补丁。

JSON 合并补丁稍微简单一些。补丁文档与原始 JSON 资源具有相同的结构，但仅包含应更改或添加的字段子集。此外，可以通过null在补丁文档中指定字段值来删除字段。（如果原始资源可以具有显式空值，则合并补丁不适合。）

以下是处理 PATCH 请求时可能遇到的一些典型错误情况，以及相应的 HTTP 状态代码。

错误的场景	HTTP状态码
不支持PATCH的文档格式	415（不支持的媒体类型）
格式错误的PATCH文件	400（错误请求）
PATCH文档有效，但是无法更改当前资源的状态	409（冲突）

DELETE方法

如果删除操作成功，Web 服务器应以 HTTP 状态码 204（无内容）响应，表明该过程已成功处理，但响应正文不包含更多信息。如果资源不存在，Web 服务器可以返回 HTTP 404（未找到）。

异步操作

有时 POST、PUT、PATCH 或 DELETE 操作可能需要一些需要一段时间才能完成的处理。如果在向客户端发送响应之前等待完成，可能会导致无法接受的延迟。如果是这样，请考虑使操作异步。返回 HTTP 状态代码 202（已接受）以指示请求已被接受进行处理但尚未完成。

您应该公开一个返回异步请求状态的端点，以便客户端可以通过轮询状态端点来监视状态。在 202 响应的 Location 标头中包含状态端点的 URI。例如：

```
HTTP/1.1 202 Accepted
Location: /api/status/12345
```

如果客户端向此端点发送 GET 请求，则响应应包含请求的当前状态。或者，它还可以包括预计完成时间或取消操作的链接。

```
HTTP/1.1 200 OK
Content-Type: application/json
{
    "status":"In progress",
    "link": { "rel":"cancel", "method":"delete", "href":"/api/status/12345" }
}
```

如果异步操作创建了新资源，则状态端点应在操作完成后返回状态代码 303（参见其他）。在 303 响应中，包含一个 Location 标头，提供新资源的 URI：

```
HTTP/1.1 303 See Other
Location: /api/orders/12345
```

错误处理

服务端或者客户端错误时，除了返回对应的HTTP状态码，还应该将错误的详细信息返回。通常错误信息需要包含以下部分。error：表示发生错误。code：表示本次错误的错误码。message：本次错误的描述信息。details：错误的详情字段和描述信息，建议为一个数组。

避免泄露个人信息【禁止】

对于用户账户、手机号、家庭住址等个人信息，通常不建议放在URL的参数中。通常请求的URL会被记录到日志中，因此我们避免在设计API时将个人的API信息放在URL的请求参数中。

分页和过滤

请求某个资源的聚合时，需要限定最大的返回数量，避免服务收到请求过多的数据，导致服务端处理一次请求代价过大。在设计API时，通常需要限定一次请求的最大数量，避免服务端受到拒绝服务的攻击。在请求资源过多，推荐使用分页的方式，例如`/orders?page=10&page-size=10`请求第10页。一次返回某个资源的全部字段，通常也是一种浪费，我们建议在请求参数加上字段过滤，例如`/orders?fields=ProductID,Quantity`。

限流

设计API接口是，需要加上限流，同时在返回的headers中记录限流的状态。限流的状态包括当前的请求总次数，还剩余的请求次数，限流重新生效的时间。使用3个headers来表示限流。

```
X-Ratelimit-Limit: 18000
X-Ratelimit-Remaining: 17995
X-Ratelimit-Reset: 1590570990
```

返回API的描述列表

使用API描述列表，无需在提前了解系统的信息的情况下，获取系统提供的能力的API列表。API列表通常包括接口的简单描述和链接。参考github，访问api.github.com可以获取github的接口和链接。

```
{
current_user_url: "https://api.github.com/user",
current_user_authorizations_html_url: "https://github.com/settings/connections/applications{/client_id}",
authorizations_url: "https://api.github.com/authorizations",
code_search_url: "https://api.github.com/search/code?q={query}{&page,per_page,sort,order}",
commit_search_url: "https://api.github.com/search/commits?q={query}{&page,per_page,sort,order}",
...
}
```

其他建议

本建议规范跟网校的HTTP API参考规范差别较大，请根据实际使用场景遵循网校的规范，并在网校规范未明确界定的场景下，参考本规范建议。网校的规范参考这里。https://yach-doc-shimo.zhiyinlou.com/docs/cDHTjJCtjx6jQxKG/ 〈网校研发接口规范-V2.0（公示期）〉

参考

1. https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design
2. https://github.com/Microsoft/api-guidelines/blob/master/Guidelines.md
3. http://developer.github.com/v3/#current-version
4. https://cloud.google.com/apis/design