



下载APP

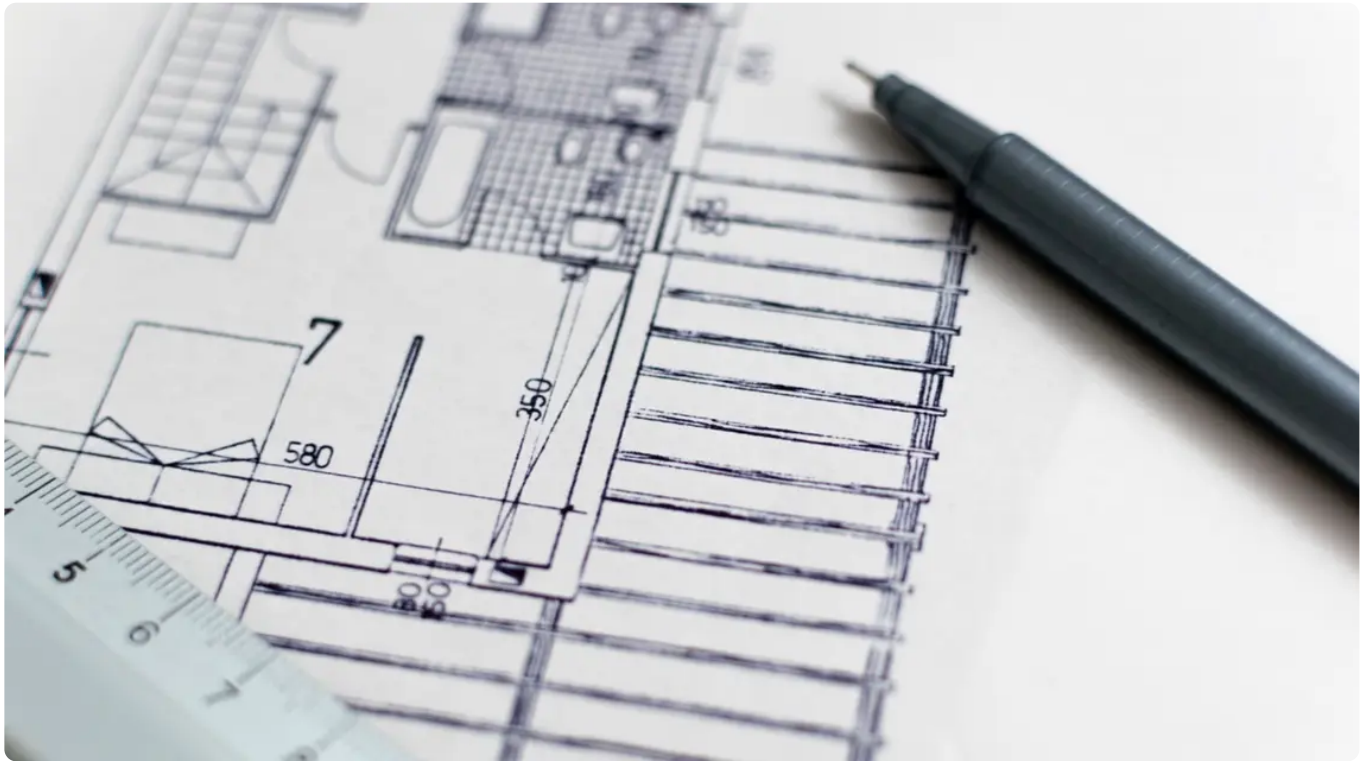


## 12 | API 风格（上）：如何设计RESTful API？

2021-06-22 孔令飞

《Go 语言项目开发实战》

[课程介绍 >](#)



讲述：孔令飞

时长 17:12 大小 15.76M



你好，我是孔令飞。从今天开始，我们就要进入实战第二站，开始学习如何设计和开发 Go 项目开发中的基础功能了。接下来两讲，我们一起来看下如何设计应用的 API 风格。

绝大部分的 Go 后端服务需要编写 API 接口，对外提供服务。所以在开发之前，我们需要确定一种 API 风格。API 风格也可以理解为 API 类型，目前业界常用的 API 风格有三种：REST、RPC 和 GraphQL。我们需要根据项目需求，并结合 API 风格的特点，确定使用哪种 API 风格，这对以后的编码实现、通信方式和通信效率都有很大的影响。

在 Go 项目开发中，用得最多的是 REST 和 RPC，我们在 IAM 实战项目中也使用了 R 和 RPC 来构建示例项目。接下来的两讲，我会详细介绍下 REST 和 RPC 这两种风格，如果你对 GraphQL 感兴趣，[GraphQL 中文官网](#)有很多文档和代码示例，你可以自行学习。



这一讲，我们先来看下 RESTful API 风格设计，下一讲再介绍下 RPC API 风格。

## RESTful API 介绍

在回答“RESTful API 是什么”之前，我们先来看下 REST 是什么意思：REST 代表的是表现层状态转移（REpresentational State Transfer），由 Roy Fielding 在他的论文 [《Architectural Styles and the Design of Network-based Software Architectures》](#) 里提出。REST 本身并没有创造新的技术、组件或服务，它只是一种软件架构风格，是一组架构约束条件和原则，而不是技术框架。

**REST 有一系列规范，满足这些规范的 API 均可称为 RESTful API。** REST 规范把所有内容都视为资源，也就是说网络上一切皆资源。REST 架构对资源的操作包括获取、创建、修改和删除，这些操作正好对应 HTTP 协议提供的 GET、POST、PUT 和 DELETE 方法。HTTP 动词与 REST 风格 CRUD 的对应关系见下表：

HTTP方法	行为	URI	示例说明
GET	获取资源列表	/users	获取用户列表
GET	获取一个具体的资源	/users/admin	获取 admin 用户的详细信息
POST	创建一个新的资源	/users	创建一个新用户
PUT	以整体的方式更新一个资源	/users/admin	更新 user 为 admin 的用户
DELETE	删除服务器上的一个资源	/users/admin	删除 user 为 admin 的用户

REST 风格虽然适用于很多传输协议，但在实际开发中，由于 REST 天生和 HTTP 协议相辅相成，因此 HTTP 协议已经成了实现 RESTful API 事实上的标准。所以，REST 具有以下核心特点：

- 以资源 (resource) 为中心，所有的东西都抽象成资源，所有的行为都应该是在资源上的 CRUD 操作。
- 资源对应着面向对象范式里的对象，面向对象范式以对象为中心。
- 资源使用 URI 标识，每个资源实例都有一个唯一的 URI 标识。例如，如果我们有一个用户，用户名是 admin，那么它的 URI 标识就可以是 /users/admin。

资源是有状态的，使用 JSON/XML 等在 HTTP Body 里表征资源的状态。

客户端通过四个 HTTP 动词，对服务器端资源进行操作，实现“表现层状态转化”。

无状态，这里的无状态是指每个 RESTful API 请求都包含了所有足够完成本次操作的信息，服务器端无须保持 session。无状态对于服务端的弹性扩容是很重要的。

因为怕你弄混概念，这里强调下 REST 和 RESTful API 的区别：**REST 是一种规范，而 RESTful API 则是满足这种规范的 API 接口。**

## RESTful API 设计原则

上面我们说了，RESTful API 就是满足 REST 规范的 API，由此看来，RESTful API 的核心是规范，那么具体有哪些规范呢？

接下来，我就从 URI 设计、API 版本管理等七个方面，给你详细介绍下 RESTful API 的设计原则，然后再通过一个示例来帮助你快速启动一个 RESTful API 服务。希望你学完这一讲之后，对如何设计 RESTful API 有一个清楚的认知。

## URI 设计

资源都是使用 URI 标识的，我们应该按照一定的规范来设计 URI，通过规范化可以使我们的 API 接口更加易读、易用。以下是 URI 设计时，应该遵循的一些规范：

资源名使用名词而不是动词，并且用名词复数表示。资源分为 Collection 和 Member 两种。

- Collection：一堆资源的集合。例如我们系统里有很多用户（User），这些用户的集合就是 Collection。Collection 的 URI 标识应该是 域名/资源名复数，例如 `https:// iam.api.marmotedu.com/users`。
- Member：单个特定资源。例如系统中特定名字的用户，就是 Collection 里的一个 Member。Member 的 URI 标识应该是 域名/资源名复数/资源名称，例如 `https:// iam.api.marmotedu.com/users/admin`。

URI 结尾不应包含/。

URI 中不能出现下划线 \_，必须用中杠线 - 代替（有些人推荐用 \_，有些人推荐用 -，统一使用一种格式即可，我比较推荐用 -）。

URI 路径用小写，不要用大写。

避免层级过深的 URI。超过 2 层的资源嵌套会很乱，建议将其他资源转化为?参数，比如：

[复制代码](#)

```
1 /schools/tsinghua/classes/rooma/students/zhang # 不推荐
2 /students?school=tsinghua&class=rooma # 推荐
```

这里有个地方需要注意：在实际的 API 开发中，可能你会发现有些操作不能很好地映射为一个 REST 资源，这时候，你可以参考下面的做法。

将一个操作变成资源的一个属性，比如想在系统中暂时禁用某个用户，可以这么设计 URI：`/users/zhangsan?active=false`。

将操作当作是一个资源的嵌套资源，比如一个 GitHub 的加星操作：

[复制代码](#)

```
1 PUT /gists/:id/star # github star action
2 DELETE /gists/:id/star # github unstar action
```

如果以上都不能解决问题，有时可以打破这类规范。比如登录操作，登录不属于任何一个资源，URI 可以设计为：`/login`。

在设计 URI 时，如果你遇到一些不确定的地方，推荐你参考 [Git@H@ub 标准 RESTful API](#)。

## REST 资源操作映射为 HTTP 方法

基本上 RESTful API 都是使用 HTTP 协议原生的 GET、PUT、POST、DELETE 来标识对资源的 CRUD 操作的，形成的规范如下表所示：

	Collection资源 (/users)	ember资源(/users/:username)
GET	获取一个Collection下所有Member的信息	获取一个Member的状态表征
PUT	用另外一个Collection替换这个Collection。不常用，不建议使用	更新一个Member的状态表征
POST	在Collection中新建一个Member	没有这类操作
DELETE	删除整个Collection，可以用来批量删除资源	删除这个Member

对资源的操作应该满足安全性和幂等性：

安全性：不会改变资源状态，可以理解为只读的。

幂等性：执行 1 次和执行 N 次，对资源状态改变的效果是等价的。

使用不同 HTTP 方法时，资源操作的安全性和幂等性对照见下表：

HTTP方法	是否安全	是否幂等
GET	是	是
POST	否	否
PUT	否	是
DELETE	否	是

在使用 HTTP 方法的时候，有以下两点需要你注意：

GET 返回的结果，要尽量可用于 PUT、POST 操作中。例如，用 GET 方法获得了一个 user 的信息，调用者修改 user 的邮件，然后将此结果再用 PUT 方法更新。这要求 GET、PUT、POST 操作的资源属性是一致的。

如果对资源进行状态 / 属性变更，要用 PUT 方法，POST 方法仅用来创建或者批量删除这两种场景。

在设计 API 时，经常会有批量删除的需求，需要在请求中携带多个需要删除的资源名，但是 HTTP 的 DELETE 方法不能携带多个资源名，这时候可以通过下面三种方式来解决：

发起多个 DELETE 请求。

操作路径中带多个 id，id 之间用分隔符分隔，例如：DELETE /users?ids=1,2,3。

直接使用 POST 方式来批量删除，body 中传入需要删除的资源列表。

其中，第二种是我最推荐的方式，因为使用了匹配的 DELETE 动词，并且不需要发送多次 DELETE 请求。

你需要注意的是，这三种方式都有各自的使用场景，你可以根据需要自行选择。如果选择了某一种方式，那么整个项目都需要统一用这种方式。

## 统一的返回格式

一般来说，一个系统的 RESTful API 会向外界开放多个资源的接口，每个接口的返回格式要保持一致。另外，每个接口都会返回成功和失败两种消息，这两种消息的格式也要保持一致。不然，客户端代码要适配不同接口的返回格式，每个返回格式又要适配成功和失败两种消息格式，会大大增加用户的学习和使用成本。

返回的格式没有强制的标准，你可以根据实际的业务需要返回不同的格式。本专栏 **第 19 讲** 中会推荐一种返回格式，它也是业界最常用和推荐的返回格式。

## API 版本管理

随着时间的推移、需求的变更，一个 API 往往满足不了现有的需求，这时候就需要对 API 进行修改。对 API 进行修改时，不能影响其他调用系统的正常使用，这就要求 API 变更做到向下兼容，也就是新老版本共存。

但在实际场景中，很可能会出现同一个 API 无法向下兼容的情况。这时候最好的解决办法是从一开始就引入 API 版本机制，当不能向下兼容时，就引入一个新的版本，老的版本则保留原样。这样既能保证服务的可用性和安全性，同时也能满足新需求。

API 版本有不同的标识方法，在 RESTful API 开发中，通常将版本标识放在如下 3 个位置：

URL 中，比如 `/v1/users`。

HTTP Header 中，比如 `Accept: vnd.example-com.foo+json; version=1.0`。

Form 参数中，比如 `/users?version=v1`。

我们这门课中的版本标识是放在 URL 中的，比如 `/v1/users`，这样做的好处是很直观，GitHub、Kubernetes、Etcd 等很多优秀的 API 均采用这种方式。

这里要注意，有些开发人员不建议将版本放在 URL 中，因为他们觉得不同的版本可以理解成同一种资源的不同表现形式，所以应该采用同一个 URI。对于这一点，没有严格的标准，根据项目实际需要选择一种方式即可。

## API 命名

API 通常的命名方式有三种，分别是驼峰命名法 (serverAddress)、蛇形命名法 (server\_address) 和脊柱命名法 (server-address)。

驼峰命名法和蛇形命名法都需要切换输入法，会增加操作的复杂性，也容易出错，所以这里建议用脊柱命名法。GitHub API 用的就是脊柱命名法，例如 [selected-actions](#)。

## 统一分页 / 过滤 / 排序 / 搜索功能

REST 资源的查询接口，通常情况下都需要实现分页、过滤、排序、搜索功能，因为这些功能是每个 REST 资源都能用到的，所以可以实现为一个公共的 API 组件。下面来介绍下这些功能。

分页：在列出一个 Collection 下所有的 Member 时，应该提供分页功能，例如 `/users?offset=0&limit=20`（limit，指定返回记录的数量；offset，指定返回记录的开始位置）。引入分页功能可以减少 API 响应的延时，同时可以避免返回太多条目，导致服务器 / 客户端响应特别慢，甚至导致服务器 / 客户端 crash 的情况。

过滤：如果用户不需要一个资源的全部状态属性，可以在 URI 参数里指定返回哪些属性，例如 `/users?fields=email,username,address`。

排序：用户很多时候会根据创建时间或者其他因素，列出一个 Collection 中前 100 个 Member，这时可以在 URI 参数中指明排序参数，例如 `/users?sort=age,desc`。



搜索：当一个资源的 Member 太多时，用户可能想通过搜索，快速找到所需要的 Member，或着想搜下有没有名字为 xxx 的某类资源，这时候就需要提供搜索功能。搜索建议按模糊匹配来搜索。

## 域名

API 的域名设置主要有两种方式：

🔗 <https://marmotedu.com/api>，这种方式适合 API 将来不会有进一步扩展的情况，比如刚开始 marmotedu.com 域名下只有一套 API 系统，未来也只有这一套 API 系统。

🔗 <https://iam.api.marmotedu.com>，如果 marmotedu.com 域名下未来会新增另一个系统 API，这时候最好的方式是每个系统的 API 拥有专有的 API 域名，比如：storage.api.marmotedu.com，network.api.marmotedu.com。腾讯云的域名就是采用这种方式。

到这里，我们就将 REST 设计原则中的核心原则讲完了，这里有个需要注意的点：不同公司、不同团队、不同项目可能采取不同的 REST 设计原则，以上所列的基本上都是大家公认的原则。

REST 设计原则中，还有一些原则因为内容比较多，并且可以独立成模块，所以放在后面来讲。比如 RESTful API 安全性、状态返回码和认证等。

## REST 示例

上面介绍了一些概念和原则，这里我们通过一个“Hello World”程序，来教你用 Go 快速启动一个 RESTful API 服务，示例代码存放在 🔗 [gopractisedemo/apistyle/ping/main.go](https://gopractisedemo.apistyle/ping/main.go)。

📄 复制代码

```
1 package main
2
3 import (
4     "log"
5     "net/http"
6 )
7
```



```
8 func main() {
9     http.HandleFunc("/ping", pong)
10    log.Println("Starting http server ...")
11    log.Fatal(http.ListenAndServe(":50052", nil))
12 }
13
14 func pong(w http.ResponseWriter, r *http.Request) {
15     w.Write([]byte("pong"))
16 }
```

在上面的代码中，我们通过 `http.HandleFunc`，向 HTTP 服务注册了一个 `pong` handler，在 `pong` handler 中，我们编写了真实的业务代码：返回 `pong` 字符串。

创建完 `main.go` 文件后，在当前目录下执行 `go run main.go` 启动 HTTP 服务，在一个新的 Linux 终端下发送 HTTP 请求，进行使用 `curl` 命令测试：

```
1 $ curl http://127.0.0.1:50052/ping
2 pong
```

[复制代码](#)

## 总结

这一讲，我介绍了两种常用 API 风格中的一种，RESTful API。REST 是一种 API 规范，而 RESTful API 则是满足这种规范的 API 接口，RESTful API 的核心是规范。

在 REST 规范中，资源通过 URI 来标识，资源名使用名词而不是动词，并且用名词复数表示，资源都是分为 Collection 和 Member 两种。RESTful API 中，分别使用 POST、DELETE、PUT、GET 来表示 REST 资源的增删改查，HTTP 方法、Collection、Member 不同组合会产生不同的操作，具体的映射你可以看下 **REST 资源操作映射为 HTTP 方法** 部分的表格。

为了方便用户使用和理解，每个 RESTful API 的返回格式、错误和正确消息的返回格式，都应该保持一致。RESTful API 需要支持 API 版本，并且版本应该能够向前兼容，我们可以将版本号放在 URL 中、HTTP Header 中、Form 参数中，但这里我建议将版本号放在 URL 中，例如 `/v1/users`，这种形式比较直观。

另外，我们可以通过脊柱命名法来命名 API 接口名。对于一个 REST 资源，其查询接口还应该支持分页 / 过滤 / 排序 / 搜索功能，这些功能可以用同一套机制来实现。API 的域名可以采用 `https://marmotedu.com/api` 和 `https://iam.api.marmotedu.com` 两种格式。

最后，在 Go 中我们可以使用 `net/http` 包来快速启动一个 RESTful API 服务。

## 课后练习

1. 使用 `net/http` 包，快速实现一个 RESTful API 服务，并实现 `/hello` 接口，该接口会返回 “Hello World” 字符串。
2. 思考一下，RESTful API 这种 API 风格是否能够满足你当前的项目需要，如果不满足，原因是什么？

期待在留言区看到你的思考和答案，也欢迎和我一起探讨关于 RESTful API 相关的问题，我们下一讲见！

分享给需要的人，Ta 订阅后你可得 **24 元现金奖励**

 赞 4     提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇   11 | 设计模式：Go 常用设计模式概述

下一篇   13 | API 风格（下）：RPC API 介绍

更多课程推荐

# 容器实战高手课

在实战中深入理解容器技术的本质

李程远

eBay 总监级工程师  
云平台架构师



涨价倒计时

今日订阅 **¥69**，7月20日涨价至 **¥129**

## 精选留言 (6)

写留言



管清麟

2021-07-01

我觉得老师可以专门开个gin的专栏，看了一下iam源码写的真好。

作者回复: 业界No.1 没有之一



6



pedro

2021-06-22

在过去的经验中，RESTful API对于动词性API不能很好的work，比如说修改密码，重置密码等，很难通过URL和HTTP方法表征出来。

但是对于Github，豆瓣等资源性API，大量的都是资源获取与删除，就特别适合RESTful。

作者回复: 是的，可以将这些动词抽象成一个属性



4

2

**cxn**

2021-06-22

URI中使用\_踩过坑,谷歌业务中URI中不能出现\_

展开 ▾

💬 1

👍 1

**h**

2021-06-29

``避免层级过深的 URI。超过 2 层的资源嵌套会很乱，建议将其他资源转化为?参数，比如：

/schools/tsinghua/classes/rooma/students/zhang # 不推荐

/students?school=tsinghua&class=rooma # 推荐``

老师这句话，我想起来了我一个接口，是和用户组相关的。...

展开 ▾

作者回复: 显示组用户：GET /groups/:gid

添加组用户：POST /groups/:gid/users/:uid

删除组用户：DELETE /groups/:gid/users?uids=1,2,3

/groups/:gid/users/:uid 这个是其实2层资源嵌套。

这样设计原因是：符合现实世界的分层关系，逻辑更合理



💬

👍

**nio**

2021-06-24

如果比较复杂一点的查询，比如需要join表的情况，如何做扩展比较好呢？

作者回复: 可以试试gorm的Joins方法。



💬

👍



2021-06-23

像RESTful 中 对某个资源的获取 api 中，针对某个特定资源的获取是尽量精简还是需要详细？

如 需求是 获取最近十条的最新数据

应该是 GET: /data?order=createdAt,desc

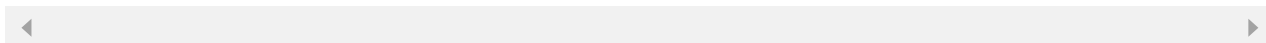
还是使用 GET: /last10data ...

展开 ✓

作者回复: 选择GET: /data?order=createdAt,desc。

前后端分离的时候，尽量设计的通用些。甚至可以不考虑前端。

前端需要的参数，是接口返回参数的一个子集。



1

