RESTful API 介绍

什么是 API

API（Application Programming Interface，应用程序编程接口）是一些预先定义的函数或者接口，目的是提供应用程序与开发人员基于某软件或硬件得以访问一组例程的能力，而又无须访问源码，或理解内部工作机制的细节。

要实现一个 API 服务器，首先要考虑两个方面：API 风格和媒体类型。Go 语言中常用的 API 风格是 RPC 和 REST，常用的媒体类型是 JSON、XML 和 Protobuf。在 Go API 开发中常用的组合是 gRPC + Protobuf 和 REST + JSON。

REST 简介

REST 代表表现层状态转移（REpresentational State Transfer），由 Roy Fielding 在他的 论文 (https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm 中提出。REST 是一种软件架构风格，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 规范中有如下几个核心：

1. REST 中一切实体都被抽象成资源，每个资源有一个唯一的标识 —— URI，所有的行为都应该是在资源上的 CRUD 操作
2. 使用标准的方法来更改资源的状态，常见的操作有：资源的增删改查操作
3. 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。

REST 风格虽然适用于很多传输协议，但在实际开发中，REST 由于天生和 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。在 HTTP 协议中通过 POST、DELETE、PUT、GET 方法来对应 REST 资源的增、删、改、查操作，具体的对应关系如下：

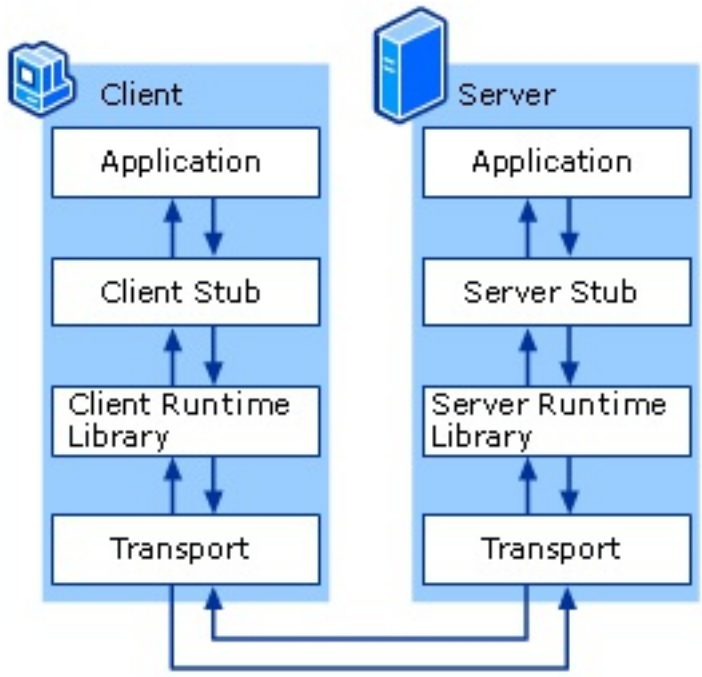
HTTP 方法	行为	URI	示例说明
GET	获取资源列表	/users	获取用户列表
GET	获取一个具体的资源	/users/admin	获取 admin 用户的详细信息
POST	创建一个新的资源	/users	创建一个新用户
PUT	以整体的方式更新一个资源	/users/1	更新 id 为 1 的用户
DELETE	删除服务器上的一个资源	/users/1	删除 id 为 1 的用户

RPC 简介

根据维基百科的定义：远程过程调用（Remote Procedure Call，RPC）是一个计算机通信协议。该协议允许运行于一台计算机的程序调用另一台计算机的子程序，而程序员无须额外地为这个交互作用编程。

通俗来讲，就是服务端实现了一个函数，客户端使用 RPC 框架提供的接口，调用这个函数的实现，并获取返回值。RPC 屏蔽了底层的网络通信细节，使得开发人员无须关注网络编程的细节，而将更多的时间和精力放在业务逻辑本身的实现上，从而提高开发效率。

RPC 的调用过程如下（图片来自 How RPC Works (https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc738291(v=ws.10)）：



1. Client 通过本地调用，调用 Client Stub
2. Client Stub 将参数打包（也叫 Marshalling）成一个消息，然后发送这个消息
3. Client 所在的 OS 将消息发送给 Server
4. Server 端接收到消息后，将消息传递给 Server Stub
5. Server Stub 将消息解包（也叫 Unmarshalling）得到参数
6. Server Stub 调用服务端的子程序（函数），处理完后，将最终结果按照相反的步骤返回给 Client

> Stub 负责调用参数和返回值的流化（serialization）、参数的打包解包，以及负责网络层的通信。Client 端一般叫 Stub，Server 端一般叫 Skeleton。

REST vs RPC

在做 API 服务器开发时，很多人都会遇到这个问题 —— 选择 REST 还是 RPC。RPC 相比 REST 的优点主要有 3 点：

1. RPC+Protobuf 采用的是 TCP 做传输协议，REST 直接使用 HTTP 做应用层协议，这种区别导致 REST 在调用性能上会比 RPC+Protobuf 低
2. RPC 不像 REST 那样，每一个操作都要抽象成对资源的增删改查，在实际开发中，有很多操作很难抽象成资源，比如登录操作。所以在实际开发中并不能严格按照 REST 规范来写 API，RPC 就不存在这个问题
3. RPC 屏蔽网络细节、易用，和本地调用类似

> 这里的易用指的是调用方式上的易用性。在做 RPC 开发时，开发过程很烦琐，需要先写一个 DSL 描述文件，然后用代码生成器生成各种语言代码，当描述文件有更改时，必须重新定义和编译，维护性差。

但是 REST 相较 RPC 也有很多优势：

1. 轻量级，简单易用，维护性和扩展性都比较好
2. REST 相对更规范，更标准，更通用，无论哪种语言都支持 HTTP 协议，可以对接外部很多系统，只要满足 HTTP 调用即可，更适合对外，RPC 会有语言限制，不同语言的 RPC 调用起来很麻烦
3. JSON 格式可读性更强，开发调试都很方便
4. 在开发过程中，如果严格按照 REST 规范来写 API，API 看起来更清晰，更容易被大家理解

> 在实际开发中，严格按照 REST 规范来写很难，只能尽可能 RESTful 化。

其实业界普遍采用的做法是，内部系统之间调用用 RPC，对外用 REST，因为内部系统之间可能调用很频繁，需要 RPC 的高性能支撑。对外用 REST 更易理解，更通用些。当然以现有的服务器性能，如果两个系统间调用不是特别频繁，对性能要求不是非常高，以笔者的开发经验来看，REST 的性能完全可以满足。本小册不是讨论微服务，所以不存在微服务之间的高频调用场景，此外 REST 在实际开发中，能够满足绝大部分的需求场景，所以 RPC 的性能优势可以忽略，相反基于 REST 的其他优势，笔者更倾向于用 REST 来构建 API 服务器，本小册正是用 REST 风格来构建 API 的。

媒体类型选择

媒体类型是独立于平台的类型，设计用于分布式系统间的通信，媒体类型用于传递信息，一个正式的规范定义了这些信息应该如何表示。HTTP 的 REST 能够提供多种不同的响应形式，常见的是 XML 和 JSON。JSON 无论从形式上还是使用方法上都更简单。相比 XML，JSON 的内容更加紧凑，数据展现形式直观易懂，开发测试都非常方便，所以在媒体类型选择上，选择了 JSON 格式，这也是很多大公司所采用的格式。

小结

本小节介绍了软件架构中 API 的实现方式，并简单介绍了相应的技术，通过对比，得出本小册所采用的实现方式：API 风格采用 REST，媒体类型选择 JSON。通过本小节的学习，读者可以了解小册所构建 API 服务器核心技术的选型和原因。