# 移动App的REST API设计实践

# http://www.jianshu.com/p/23cccb3a90b1

## 通讯协议

一些只是对服务器数据进行CRUD操作的App，通常采用HTTP协议，为了安全也可以采用HTTPS协议。IM软件可以选择使用XMPP协议。

其他一些特有场景的App可能基于Socket自定义协议。

SOCKET是实现传输层协议的一种编程API，可以是TCP，也可以是UDP。

TCP --- 传输控制协议,提供的是面向连接、可靠的字节流服务。TCP提供超时重发，丢弃重复数据，检验数据，流量控制等功能，保证数据能从一端传到另一端。

UDP --- 用户数据报协议，是一个无连接的简单的面向数据报的运输层协议。UDP不提供可靠性，它只是把应用程序传给IP层的数据报发送出去，但是并不能保证它们能到达目的地。

在需要保证需要传输数据到对方的时候应该选择TCP协议，比如文件传输。而像在线视频播放这些，可以不用保证数据100%被接受的，可以用UDP，因为丢失一帧画面数据没什么影响，而且用UDP速度会更快。

最近也有听说用WebSocket做通讯的，可以在Web端和App端共用一个接口。

## 数据格式

比较通用的数据交互格式是JSON和XML。在现在，JSON格式使用的更为广泛，因为结构简单而且解析方便。

iOS5之后的SDK提供了NSJSONSerialization类解析JSON，而Android SDK本身就自带了org.json 的相关包来提供JSON解析的功能。如果在Android里面对JSON的解析或者序列化性能有要求，可以考虑使用android.utils包里面的JsonReader 和 JsonWriter类。它们是用流式解析的方式，不过使用更加繁琐。也可以考虑使用Google的Gson，阿里巴巴的fastjson，以及Jackson这些开源JSON处理的库，它们提供了更多的功能，也有更好的性能。

XML设计目标是“Extensible Markup Language”，可扩展的标记语言，而不是JSON的只是作为一个数据序列化的语言。XML格式也有自己的优点，比如你可以用通用的XPath查询特定的数据而不用用一个个嵌套的循环或者分支语句从一个复杂的数据中拿到一个特定的值。

## REST架构的API设计

可以参考阮一峰文章[http://www.ruanyifeng.com/blog/2014/05/restful\_api.html](http://www.ruanyifeng.com/blog/2014/05/restful_api.html" \t "_blank)

本文其余内容基本都是在谈REST架构的API

一些感想:

1. API域名与网站域名分开，比如使用 [https://api.example.com](https://api.example.com/" \t "_blank) 这样的方式
2. API请求中应该加入API版本号，比如在URL中加入版本号，如  
   [https://api.example.com/v1/，](https://api.example.com/v1/%EF%BC%8C" \t "_blank) 这样当API升级的或者改动的时候，可以保留旧的API服务器，把新的API服务器mount到[https://api.example.com/v2/](https://api.example.com/v2/" \t "_blank)上，这样使用旧的API的App也不会出现问题
3. API返回的数据量小的时候，没太多必要进行zip压缩。API返回数据大的时候，要考虑API设计是否适当。  
   4.写API要方便使用这些API开发的人员测试，比如写好文档，使用[https://helloreverb.com/developers/swagger](https://helloreverb.com/developers/swagger" \t "_blank) 这样的工具生成调试页面。

## 性能

要避免写API Server的时候出现一些低级的错误，比如数据库查询用了N＋1 Query。

其他性能的问题其实和Web开发大同小异，无非是横向和纵向的几种不同方式的扩展。

## 安全性和用户认证

如果使用第三方的API，通常采用Oauth协议或者SSO登录。

如果是自己开发，因为自己写HTTP请求不会和浏览器发送的时候自动维护一个Cookie，所以可以自己手动维护一个Token，代替Cookie的作用来进行用户验证。

对于Token等关键的数据，不要明文保存在设备本地，可以用iOS提供的Keychain这样的机制进行加密存储。

也有用Xauth协议的，类似Oauth的简化版。

安全方便，和Web开发一样，不要相信用户的任何数据，服务器端都应该做对应的认证。

此外，如果对一些数据有较高的安全需求，那么应该避免把秘密的数据用明文写在代码里，比如一些第三方Acess Key，可以在启动的时候动态请求，否则很容易被反编译获取。

Android一定要做好反编译工作。

## 学习

Github的API设计应该算得上优秀，值得参考：  
[https://developer.github.com/v3/](https://developer.github.com/v3/" \t "_blank)

此外，可以用Charles这些抓包工具，学习和参考别人的App与服务器的数据交互内容，Charles甚至可以在你的开发的移动设备上安装自签名证书，采用类似中间人攻击的方式来获取App采用HTTPS协议交互的数据的明文。

## 如果你不想写API

你可以使用[Parse.com](http://www.jianshu.com/p/parse.com" \t "_blank) 或者 [AVOSCloud](http://avoscloud.com/" \t "_blank) 这些BASS平台提供的服务。这些服务很适合一些不需要在服务器端提供复杂操作，以及前期的原型开发。

# RESTful API 设计指南

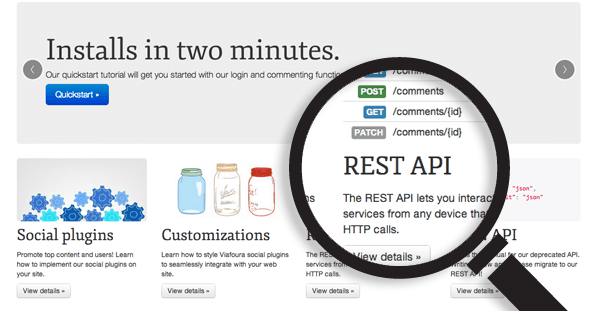
作者： [阮一峰](http://www.ruanyifeng.com/)

日期： [2014年5月22日](http://www.ruanyifeng.com/blog/2014/05/)

网络应用程序，分为前端和后端两个部分。当前的发展趋势，就是前端设备层出不穷（手机、平板、桌面电脑、其他专用设备......）。

因此，必须有一种统一的机制，方便不同的前端设备与后端进行通信。这导致API构架的流行，甚至出现["API First"](http://www.google.com.hk/search?q=API+first" \t "_blank)的设计思想。[RESTful API](http://en.wikipedia.org/wiki/Representational_state_transfer" \t "_blank)是目前比较成熟的一套互联网应用程序的API设计理论。我以前写过一篇[《理解RESTful架构》](http://www.ruanyifeng.com/blog/2011/09/restful.html" \t "_blank)，探讨如何理解这个概念。

今天，我将介绍RESTful API的设计细节，探讨如何设计一套合理、好用的API。我的主要参考了两篇文章（[1](http://codeplanet.io/principles-good-restful-api-design/" \t "_blank)，[2](https://bourgeois.me/rest/" \t "_blank)）。



## 一、协议

API与用户的通信协议，总是使用[HTTPs协议](http://www.ruanyifeng.com/blog/2014/02/ssl_tls.html" \t "_blank)。

## 二、域名

应该尽量将API部署在专用域名之下。

https://api.example.com

如果确定API很简单，不会有进一步扩展，可以考虑放在主域名下。

https://example.org/api/

## 三、版本（Versioning）

应该将API的版本号放入URL。

https://api.example.com/v1/

另一种做法是，将版本号放在HTTP头信息中，但不如放入URL方便和直观。[Github](https://developer.github.com/v3/media/" \l "request-specific-version" \t "_blank)采用这种做法。

## 四、路径（Endpoint）

路径又称"终点"（endpoint），表示API的具体网址。

在RESTful架构中，每个网址代表一种资源（resource），所以网址中不能有动词，只能有名词，而且所用的名词往往与数据库的表格名对应。一般来说，数据库中的表都是同种记录的"集合"（collection），所以API中的名词也应该使用复数。

举例来说，有一个API提供动物园（zoo）的信息，还包括各种动物和雇员的信息，则它的路径应该设计成下面这样。

* https://api.example.com/v1/zoos
* https://api.example.com/v1/animals
* https://api.example.com/v1/employees

## 五、HTTP动词

对于资源的具体操作类型，由HTTP动词表示。

常用的HTTP动词有下面五个（括号里是对应的SQL命令）。

* GET（SELECT）：从服务器取出资源（一项或多项）。
* POST（CREATE）：在服务器新建一个资源。
* PUT（UPDATE）：在服务器更新资源（客户端提供改变后的完整资源）。
* PATCH（UPDATE）：在服务器更新资源（客户端提供改变的属性）。
* DELETE（DELETE）：从服务器删除资源。

还有两个不常用的HTTP动词。

* HEAD：获取资源的元数据。
* OPTIONS：获取信息，关于资源的哪些属性是客户端可以改变的。

下面是一些例子。

* GET /zoos：列出所有动物园
* POST /zoos：新建一个动物园
* GET /zoos/ID：获取某个指定动物园的信息
* PUT /zoos/ID：更新某个指定动物园的信息（提供该动物园的全部信息）
* PATCH /zoos/ID：更新某个指定动物园的信息（提供该动物园的部分信息）
* DELETE /zoos/ID：删除某个动物园
* GET /zoos/ID/animals：列出某个指定动物园的所有动物
* DELETE /zoos/ID/animals/ID：删除某个指定动物园的指定动物

## 六、过滤信息（Filtering）

如果记录数量很多，服务器不可能都将它们返回给用户。API应该提供参数，过滤返回结果。

下面是一些常见的参数。

* ?limit=10：指定返回记录的数量
* ?offset=10：指定返回记录的开始位置。
* ?page=2&per\_page=100：指定第几页，以及每页的记录数。
* ?sortby=name&order=asc：指定返回结果按照哪个属性排序，以及排序顺序。
* ?animal\_type\_id=1：指定筛选条件

参数的设计允许存在冗余，即允许API路径和URL参数偶尔有重复。比如，GET /zoo/ID/animals 与 GET /animals?zoo\_id=ID 的含义是相同的。

## 七、状态码（Status Codes）

服务器向用户返回的状态码和提示信息，常见的有以下一些（方括号中是该状态码对应的HTTP动词）。

* 200 OK - [GET]：服务器成功返回用户请求的数据，该操作是幂等的（Idempotent）。
* 201 CREATED - [POST/PUT/PATCH]：用户新建或修改数据成功。
* 202 Accepted - [\*]：表示一个请求已经进入后台排队（异步任务）
* 204 NO CONTENT - [DELETE]：用户删除数据成功。
* 400 INVALID REQUEST - [POST/PUT/PATCH]：用户发出的请求有错误，服务器没有进行新建或修改数据的操作，该操作是幂等的。
* 401 Unauthorized - [\*]：表示用户没有权限（令牌、用户名、密码错误）。
* 403 Forbidden - [\*] 表示用户得到授权（与401错误相对），但是访问是被禁止的。
* 404 NOT FOUND - [\*]：用户发出的请求针对的是不存在的记录，服务器没有进行操作，该操作是幂等的。
* 406 Not Acceptable - [GET]：用户请求的格式不可得（比如用户请求JSON格式，但是只有XML格式）。
* 410 Gone -[GET]：用户请求的资源被永久删除，且不会再得到的。
* 422 Unprocesable entity - [POST/PUT/PATCH] 当创建一个对象时，发生一个验证错误。
* 500 INTERNAL SERVER ERROR - [\*]：服务器发生错误，用户将无法判断发出的请求是否成功。

状态码的完全列表参见[这里](http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html" \t "_blank)。

## 八、错误处理（Error handling）

如果状态码是4xx，就应该向用户返回出错信息。一般来说，返回的信息中将error作为键名，出错信息作为键值即可。

{ error: "Invalid API key" }

## 九、返回结果

针对不同操作，服务器向用户返回的结果应该符合以下规范。

* GET /collection：返回资源对象的列表（数组）
* GET /collection/resource：返回单个资源对象
* POST /collection：返回新生成的资源对象
* PUT /collection/resource：返回完整的资源对象
* PATCH /collection/resource：返回完整的资源对象
* DELETE /collection/resource：返回一个空文档

## 十、Hypermedia API

RESTful API最好做到Hypermedia，即返回结果中提供链接，连向其他API方法，使得用户不查文档，也知道下一步应该做什么。

比如，当用户向api.example.com的根目录发出请求，会得到这样一个文档。

{"link": { "rel": "collection <https://www.example.com/zoos>", "href": "<https://api.example.com/zoos>", "title": "List of zoos", "type": "application/vnd.yourformat+json" }}

上面代码表示，文档中有一个link属性，用户读取这个属性就知道下一步该调用什么API了。rel表示这个API与当前网址的关系（collection关系，并给出该collection的网址），href表示API的路径，title表示API的标题，type表示返回类型。

Hypermedia API的设计被称为[HATEOAS](http://en.wikipedia.org/wiki/HATEOAS" \t "_blank)。Github的API就是这种设计，访问[api.github.com](https://api.github.com/" \t "_blank)会得到一个所有可用API的网址列表。

{ "current\_user\_url": "<https://api.github.com/user>", "authorizations\_url": "<https://api.github.com/authorizations>", // ... }

从上面可以看到，如果想获取当前用户的信息，应该去访问[api.github.com/user](https://api.github.com/user" \t "_blank)，然后就得到了下面结果。

{ "message": "Requires authentication", "documentation\_url": "<https://developer.github.com/v3>" }

上面代码表示，服务器给出了提示信息，以及文档的网址。

## 十一、其他

（1）API的身份认证应该使用[OAuth 2.0](http://www.ruanyifeng.com/blog/2014/05/oauth_2_0.html" \t "_blank)框架。

（2）服务器返回的数据格式，应该尽量使用JSON，避免使用XML。

（完）