**REST名称**  
1 REST -- REpresentational State Transfer  
首先，之所以晦涩是因为前面主语被去掉了，全称是 Resource Representational State Transfer：通俗来讲就是：资源在网络中以某种表现形式进行状态转移。分解开来：  
Resource：资源，即数据（前面说过网络的核心）。比如 newsfeed，friends等；  
Representational：某种表现形式，比如用JSON，XML，JPEG等；  
State Transfer：状态变化。通过HTTP动词实现。

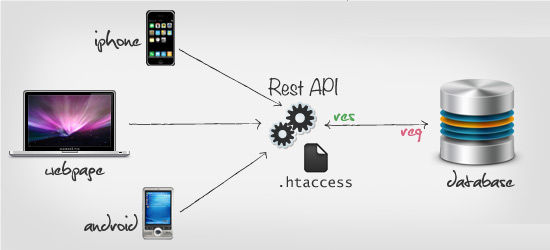
**REST架构风格**

REST是一种基于互联网的网络架构指导风格。同时它也是基于http协议的。和HTTP协议特点类似，Restful接口具有以下特点：

* 客户端-服务器模式
* 通过url定位资源：即面向资源，所有的事物都抽象为资源
* **无状态**：客户端和服务器之间的每次交互都必须携带所有信息
* **超媒体驱动**：即通过服务器返回的超链接进行资源态的跃迁
* **统一接口**：

1. 基本的http方法（POST，GET，PUT，DELETE）来实现对服务器资源的操作，即CURD。
2. http头信息
3. http响应状态码
4. http内容协商机制
5. http缓存机制
6. http客户身份认证机制

**RESTful API**  
实用的是如何正确地理解 RESTful架构和设计好RESTful API。

**首先为什么要用RESTful结构呢？**  
大家都知道"古代"网页是前端后端融在一起的，比如之前的PHP，JSP等。在之前的桌面时代问题不大，但是近年来移动互联网的发展，各种类型的Client层出不穷，RESTful可以通过一套统一的接口为 Web，iOS和Android提供服务。另外对于广大平台来说，比如Facebook platform，微博开放平台，微信公共平台等，它们不需要有显式的前端，只需要一套提供服务的接口，于是RESTful更是它们最好的选择。在RESTful架构下：  
  
**Server的API如何设计才满足RESTful要求?**  
1. URI使用名词而不是动词，且推荐用复数。  
BAD

* /getProducts
* /listOrders
* /retrieveClientByOrder?orderId=1

GOOD

* GET /products : will return the list of all products
* POST /products : will add a product to the collection
* GET /products/4 : will retrieve product #4
* PATCH/PUT /products/4 : will update product #4

2. 保证 HEAD 和 GET 方法是安全的，不会对资源状态有所改变（污染）。比如严格杜绝如下情况：  
GET /deleteProduct?id=1  
3. 资源的地址推荐用嵌套结构。比如：  
GET /friends/10375923/profile  
UPDATE /profile/primaryAddress/city6. 警惕返回结果的大小。如果过大，及时进行分页（pagination）或者加入限制（limit）。HTTP协议支持分页（Pagination）操作，在Header中使用 Link 即可。  
4. 使用正确的HTTP Status Code表示访问状态：[HTTP/1.1: Status Code Definitions](https://link.zhihu.com/?target=http%3A//www.w3.org/Protocols/rfc2616/rfc2616-sec10.html)  
5. 在返回结果用明确易懂的文本（String。注意返回的错误是要给人看的，避免用 1001 这种错误信息），而且适当地加入注释。  
6. 关于安全：自己的接口就用https，加上一个key做一次hash放在最后即可。考虑到国情，HTTPS在无线网络里不稳定，可以使用Application Level的加密手段把整个HTTP的payload加密。有兴趣的朋友可以用手机连上电脑的共享Wi-Fi，然后用Charles监听微信的网络请求（发照片或者刷朋友圈）。  
如果是平台的API，可以用成熟但是复杂的OAuth2，新浪微博这篇：[授权机制说明](https://link.zhihu.com/?target=http%3A//open.weibo.com/wiki/%25E6%258E%2588%25E6%259D%2583%25E6%259C%25BA%25E5%2588%25B6%25E8%25AF%25B4%25E6%2598%258E)

**各端的具体实现**  
如上面的图所示，Server统一提供一套RESTful API，web+ios+android作为同等公民调用API。各端发展到现在，都有一套比较成熟的框架来帮开发者事半功倍。

-- Server --  
推荐： Spring MVC 或者 Jersey 或者 Play Framework  
教程：  
[Getting Started · Building a RESTful Web Service](https://link.zhihu.com/?target=https%3A//spring.io/guides/gs/rest-service/)

-- Android --  
推荐： RetroFit ( [Retrofit](https://link.zhihu.com/?target=http%3A//square.github.io/retrofit/) ) 或者 Volley ( [mcxiaoke/android-volley · GitHub](https://link.zhihu.com/?target=https%3A//github.com/mcxiaoke/android-volley) Google官方的被block，就不贴了 )   
教程：  
[Retrofit โ Getting Started and Create an Android Client](https://link.zhihu.com/?target=https%3A//futurestud.io/blog/retrofit-getting-started-and-android-client/)  
[快速Android开发系列网络篇之Retrofit](https://link.zhihu.com/?target=http%3A//www.cnblogs.com/angeldevil/p/3757335.html)

-- iOS --  
推荐：RestKit ( [RestKit/RestKit · GitHub](https://link.zhihu.com/?target=https%3A//github.com/RestKit/RestKit) )  
教程：  
[Developing RESTful iOS Apps with RestKit](https://link.zhihu.com/?target=http%3A//code.tutsplus.com/tutorials/restkit_ios-sdk--mobile-4524)

-- Web --  
推荐随便搞！可以用重量级的AngularJS，也可以用轻量级 Backbone + jQuery 等。  
教程：[http://blog.javachen.com/2015/01/06/build-app-with-spring-boot-and-gradle/](https://link.zhihu.com/?target=http%3A//blog.javachen.com/2015/01/06/build-app-with-spring-boot-and-gradle/)

Web端不再用之前典型的PHP或JSP架构，而是改为前端渲染和附带处理简单的商务逻辑（比如AngularJS或者BackBone的一些样例）。Web端和Server只使用上述定义的API来传递数据和改变数据状态。格式一般是JSON。

iOS和Android同理可得。由此可见，Web，iOS，Android和第三方开发者变为平等的角色通过一套API来共同消费Server提供的服务。