西安电子科技大学

构件与中间件技术

第三部分 SOA与Web Service

§3-3 REST与RESTful API

徐悦甦

ysxu@xidian.edu.cn

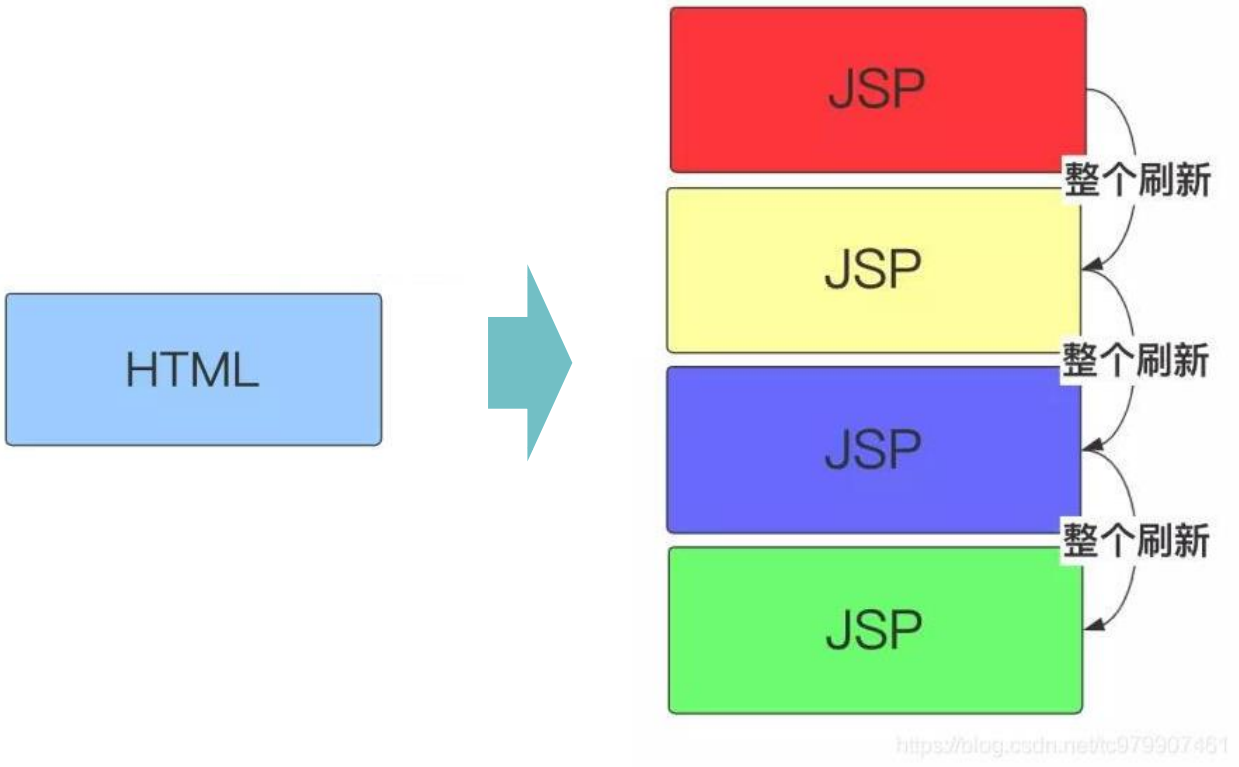
西安电子科技大学

目录

目录

RESTful API出现动机与起源

- 在互联网发展初期，移动端还未出现，页面请求和并发量不高；
- 动态页面（JSP）可满足Web开发需求。

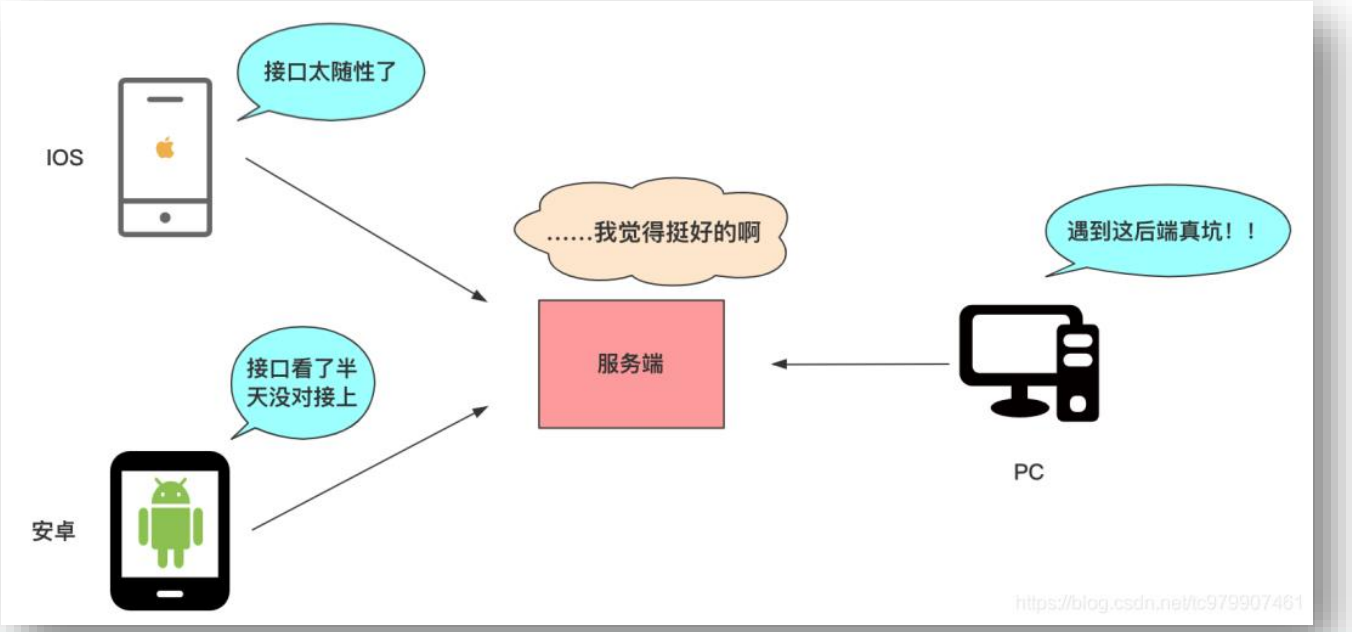


RESTful API出现动机与起源

■但是随着互联网和移动设备的发展，人们对Web应用的使用需求也增加，传统的动态页面由于低效率而渐渐被HTML + JavaScript/Ajax的前后端分离所取代；

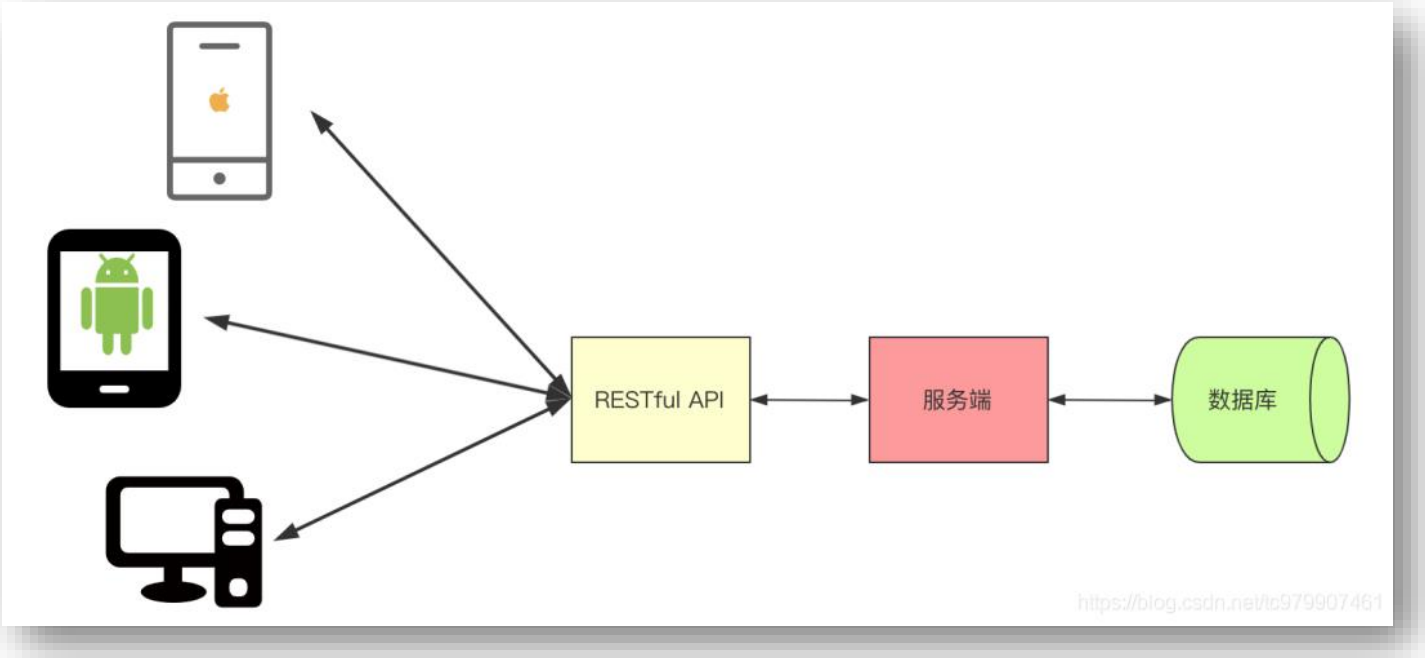
■同时，**安卓**、**iOS**、**小程序**等形式客户端层出不穷，客户端的种类多元化，而客户端和服务端就需要接口进行通信；

■从而产生了接口的**规范性问题**。



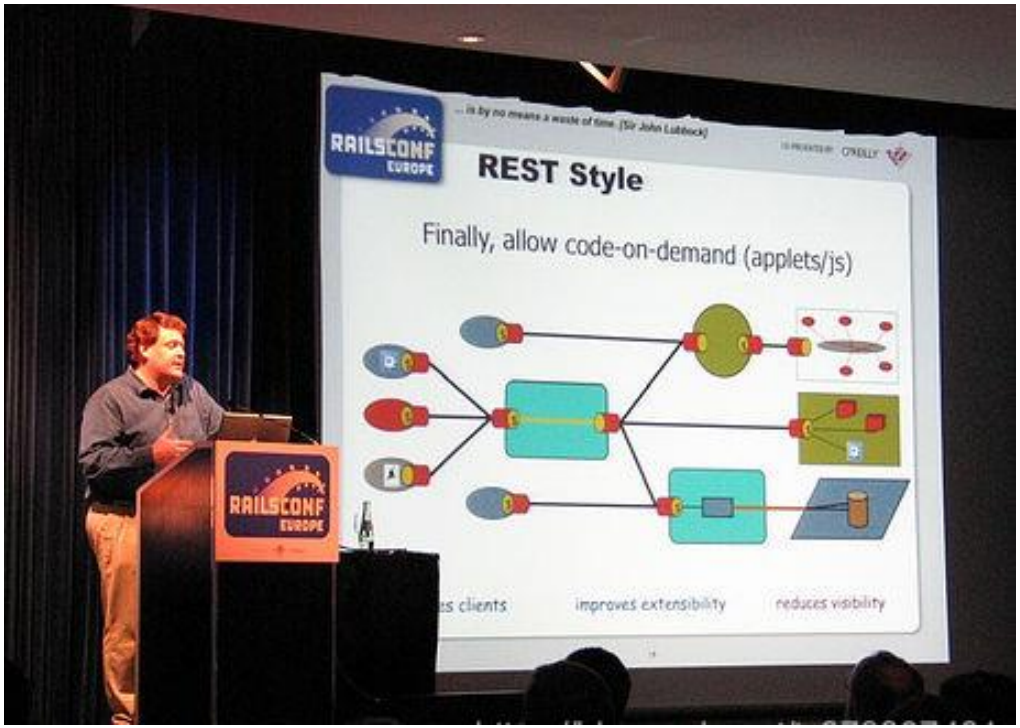
RESTful API出现动机与起源

- Web开发对于接口的需求：
  - 结构清晰、符合标准、易于理解、扩展方便；
  - RESTful风格的接口(RESTful API) 逐渐被实践应用而变得流行。



RESTful API出现动机与起源

- REST（**RE**presentational **S**tate **T**ransfer）由Dr. Roy Thomas Fielding在2000年的博士论文 “*Architectural Styles and the Design of Network-based Software Architectures*”中提出。



HTTP协议（1.0版和1.1版）的主要设计者、Apache服务器软件的作者之一、Apache基金会的第一任主席。

RESTful API出现动机与起源

ics.uci.edu/~fielding/pubs/dissertation/top.htm

应用 百度一下，你就知道 西安电子科技大学 微软 Bing 搜索 -... 首页 - 知乎 F 搜 微博-随时 AOL Search 全渠道搜索_思谋上...

UNIVERSITY OF CALIFORNIA, IRVINE

Architectural Styles and the Design of Network-based Software Architectures

DISSERTATION

submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in Information and Computer Science

by

Roy Thomas Fielding

2000

Dissertation Committee:
Professor Richard N. Taylor, Chair
Professor Mark S. Ackerman
Professor David S. Rosenblum

PDF Editions

1-column for viewing online
2-column for printing

Table of Contents

Dedication
Acknowledgments

Quote: 我写这篇文章的目的是：在符合架构原理前提下，理解和评估基于网络的应用软件的架构设计，得到一个功能强、性能好、适宜通信的架构。

REST

- **REST**：Representational State Transfer，表现层状态转化
  - 一个架构符合REST原则，就称之为RESTful架构。
- **REST中的要素**
  - **1. 资源（Resources）**
    - “表现层状态转化”中，“表现层”指“资源”（Resources）的“表现层”；
    - **“资源”**，网络上的一个**实体**，或者说是网络上的一个**具体信息**，可以是一段文本、一张图片、一首歌曲、一种服务
    - 每种资源对应一个特定的URI （统一资源标识符），每一个资源拥有唯一的URI作为网络地址。

REST

- **2. 表现层（Representation）**
  - "资源"具体呈现出来的形式，叫做它的"**表现层**"（Representation）
  - 例如，文本可以用txt格式表现，也可以用HTML格式、XML格式、JSON格式表现，甚至可以采用二进制格式；
  - 图片可以用JPG格式表现，也可以用PNG格式表现
- **3. 状态转化（State Transfer）**
  - 访问一个网站，就代表了客户端和服务器的一个交互过程，过程中涉及到数据和状态的变化。
  - 互联网通信协议：HTTP协议作为无状态协议（严格来说是HTTP1.0/1.1），所有的状态都保存在服务器端。
  - 如果客户端想要继续与服务器交互，必须通过某种途径，让服务器端发生“状态转化”（State Transfer）；
  - 状态转化建立在表现层之上，故称为"表现层状态转化"。

REST

- HTTP协议的无状态性
  - **无状态**：当浏览器给服务器发送请求时,服务器响应客户端请求。但是当同一个浏览器再次发送请求时，服务器并不知道这就是刚才的浏览器；
  - 总结：协议对于交互性场景没有记忆能力
  - HTTP 1.0与1.1为无状态协议，HTTP 2.0严格意义上为有状态协议
  - 目前应用最多的是HTTP 1.1
  - 所以出现了Cookie、Session
  - 思考：为什么HTTP 1.x版本设计为无状态协议?

有状态：

A：你今天中午吃的啥?

B：吃的大盘鸡。

A：味道怎么样呀?

B：还不错，挺好吃的。

无状态：

A：你今天中午吃的啥?

B：吃的大盘鸡。

A：味道怎么样呀?

B：？？？啊？啥？啥味道怎么样?

所以需要cookie

A：你今天中午吃的啥?

B：吃的大盘鸡。

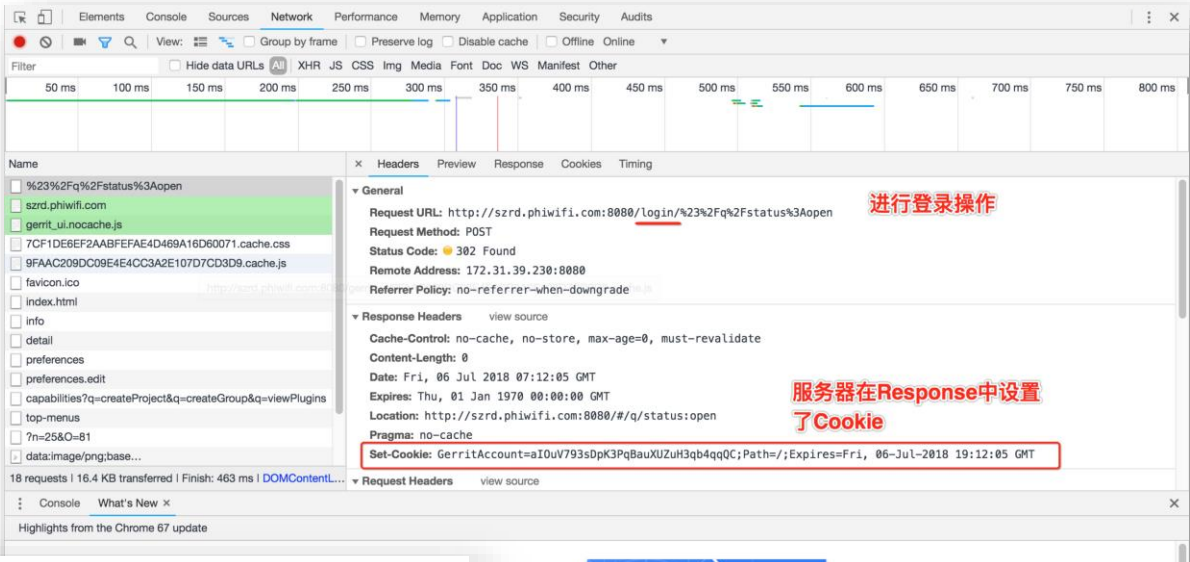
A：你今天中午吃的大盘鸡味道怎么样呀?

B：还不错，挺好吃的。

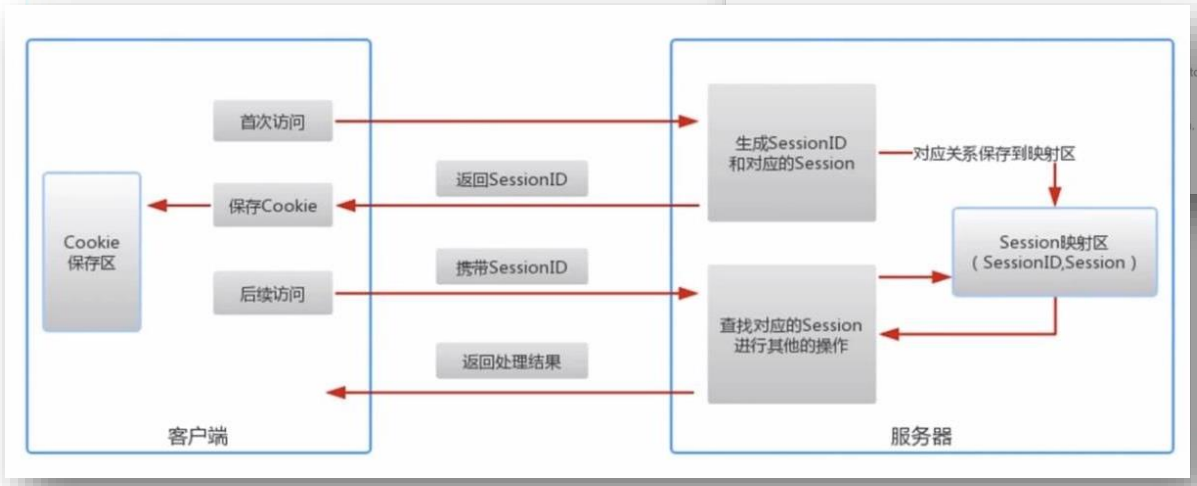
REST

- HTTP协议的无状态性

Cookie



Session



RESTful API设计

RESTful API中URL设计

- 一个完整的URL组成由以下几个部分构成：

```
URI = scheme "://" host ":" port "/" path [ "?" query ][ "#" fragment ]
```

  - **scheme**: 指底层用的协议，如http、https、ftp
  - **host**: 服务器的IP地址或者域名
  - **port**: 端口，http默认为80端口
  - **path**: 访问资源的路径，就是各种web 框架中定义的route路由
  - **query**: 查询字符串，为发送给服务器的参数，在这里更多发送数据分页、排序等参数。
  - **fragment**: 锚点，定位到页面的资源

RESTful API设计

- Path设计

  /{version}/{resources}/{resource-id}

  - version：API版本号，有些版本号也会放置于HTTP头信息中;
  - resources：资源，推荐用小写英文单词;
  - resource-id：资源的id，访问或操作该资源。

  - **Path设计习惯**
    - 不用大写字母，所有单词使用英文且小写。
    - 连字符用中杠"-“而不用下杠”_"
    - 正确使用 "/"表示层级关系,URL的层级不要过深，越靠前的层级应该越稳定
    - 结尾不要包含正斜杠分隔符"/"
    - URL中不出现动词，用请求方式表示动作
    - 资源表示用复数不要用单数
    - 不使用文件扩展名

RESTful API设计

■ 资源操作

- 对资源操作的动作对应HTTP协议中的操作方式：GET、POST、PUT、DELETE、PATCH等；

HTTP动作	对应数据库中的SQL	动作语义
Get	SELECT	从服务器取出资源
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）
Delete	DELETE	从服务器删除资源

RESTful API设计

■ 资源操作

➢ 对资源操作的动作对应HTTP协议中的操作方式：GET、POST、PUT、DELETE、PATCH等；

HTTP动作	对应数据库中的SQL	动作语义
PATCH	UPDATE	在服务器更新资源（客户端提供改变的属性）
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

➢ 资源操作总结：

– 1. 每一个URI代表一种资源；

– 2. 客户端和服务器之间，传递这种资源的某种表现层；

– 3. 客户端通过HTTP动作，对服务器端资源进行操作，实现"表现层状态转化"。

RESTful API设计

- 资源操作举例一
  - 本地用户增删查改，非Restful API

```
//添加用户
http://localhost/createuser

//删除id为1的用户
http://localhost/deleteuser?userid=1

//获取用户列表
http://localhost/getuser

//获取id为1的用户
http://localhost/getuser?userid=1

//更新id为1的用户
http://localhost/updateuser?userid=1
```

RESTful API设计

- 资源操作举例一
  - 本地用户增删查改，Restful API
  - **资源**：用户，使用URL描述资源：http://localhost/user
  - **表现层**：Json、XML或其它

```
//查询用户列表
GET请求：http://localhost/user
//查询id为1的用户
GET请求：http://localhost/user/1
//创建一个用户
POST请求：http://localhost/user
//修改id为1的用户
PUT请求：http://localhost/user/1
//删除id为1的用户
DELETE请求：http://localhost/user/1
```

RESTful API设计

- 资源操作举例二

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

RESTful API设计

■ 状态码（Status Codes）

- 服务器向用户返回的状态码和提示信息，RESTful API基于HTTP协议实现，故状态码即为HTTP协议的状态码；
- HTTP状态代码为3位，分为5个类别。

状态码	语义
1xx	相关信息
2xx	操作成功
3xx	重定向
4xx	客户端错误
5xx	服务器错误

RESTful API设计

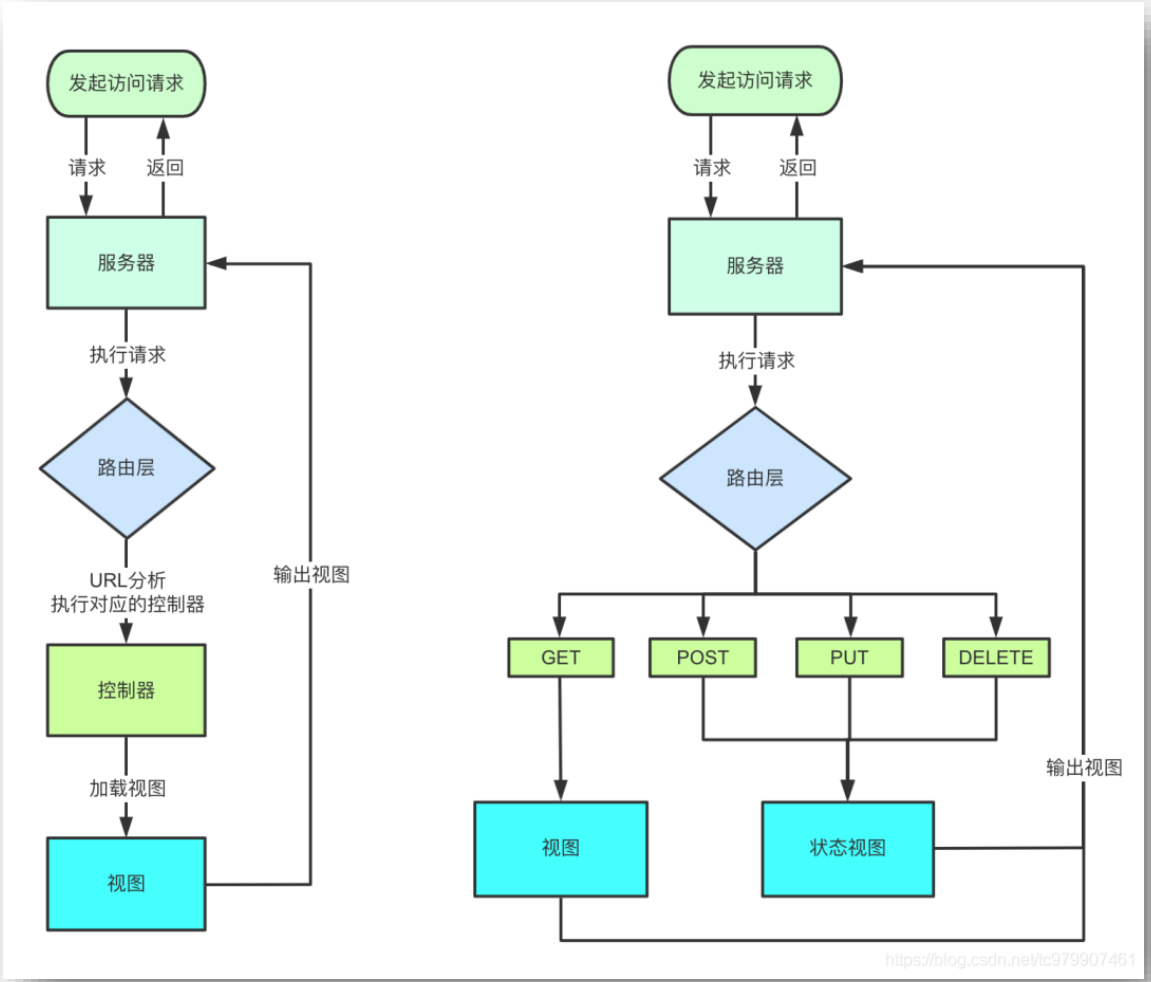
■ 状态码（Status Codes）

- 服务器向用户返回的状态码和提示信息，RESTful API基于HTTP协议实现，故状态码即为HTTP协议的状态码。

状态码	HTTP 操作	语义
200 OK	GET,PUT	服务器成功返回用户请求的数据
201 CREATED	POST, PUT, PATCH	用户新建或修改数据成功
202 Accepted	POST,PUT,DELETE,PATCH	一个请求已经进入后台排队（异步任务）
204 NO CONTENT	DELETE,PUT,PATCH	用户操作数据成功，但没有返回结果
301	GET	资源已被移除
400 INVALID REQUEST	GET,POST,PUT,DELETE,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
401 Unauthorized	GET,POST,PUT,DELETE,PATCH	用户没有权限（令牌、用户名、密码错误）
403 Forbidden	GET,POST,PUT,DELETE,PATCH	用户得到授权（与401错误相对），但是访问是被禁止的
404 NOT FOUND	GET,POST,PUT,DELETE,PATCH	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
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）
422 Unprocesable entity	POST/PUT/PATCH]	当创建一个对象时，发生一个验证错误
410 Gone	Get	用户请求的资源被永久删除，且不会再得到的
500 INTERNAL SERVER ERROR	GET,POST,PUT,DELETE,PATCH	服务器发生错误，用户将无法判断发出的请求是否成功

RESTful API设计

- RESTful API和传统API显著区别
  - **1. 请求流程**



RESTful API设计

- RESTful API和传统API显著区别
  - **2. 请求方式**
    - 在非RESTful风格的API中，通常使用GET请求和POST请求完成增删改查以及其他操作
      - 查询和删除一般使用GET方式请求，更新和插入一般使用POST请求；
      - 从请求方式上无法知道API的动作，所有在URL上都会有操作的动词来表示API进行的动作，例如：query，add，update，delete等等。
    - RESTful风格的API则要求在URL上都以名词的方式出现，请求方式对应动作。
    - 如果某些动作HTTP动词表示不了，将动作做成一种资源。比如网上汇款，从账户1向账户2汇款500元，错误的URI是：

```
POST /accounts/1/transfer/500/to/2
```

RESTful API设计

- 正确的写法是把动词transfer改成名词transaction，资源不能是动词，但是可以是一种服务。

```
POST /transaction HTTP/1.1
Host: 127.0.0.1
from=1&to=2&amount=500.0
```

RESTful API设计

RESTful API和传统API显著区别

3. Hypermedia API

- RESTful API提倡Hypermedia API，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。

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

- 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哪个API；
- 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。

RESTful API设计实例

- 设计一个简单的RESTful API实例
  - 环境：SpringBoot+MyBatis+Maven

```
package com.restfuldemo.pojo;

public class Dog {
private int id; //唯一id标识
private String name; //名称
private int age; //年龄
//省略get set
 }
```

API	非RESTful	RESTful
获取dog	/dogs/query/{dogid}	GET： /dogs/{dogid}
插入dog	/dogs/add	POST： /dogs
更新dog	dogs/update/{dogid}	PUT：/dogs/{dogid}
删除dog	/dods/delete/{dogid}	DELETE：/dogs/{dogid}

  - 方法

RESTful API设计实例

- Mapper类：

```
package com.restfuldemo.mapper;

import com.restfuldemo.pojo.Dog;
import org.apache.ibatis.annotations.*;
import java.util.List;

@Mapper
public interface DogMapper {
    @Select("select * from dog")
    List<Dog> getAllDog();
    @Select("select * from dog where id=#{id}")
    Dog getDogById(@Param("id") int id);
    @Insert("insert into dog (name,age) values (#{name},#{age})")
    boolean addDog(Dog dog);
    @Update("update dog set name=#{name},age=#{age} where id=#{id}")
    boolean updateDog(Dog dog);
    @Delete("delete from dog where id=#{id}")
    boolean deleteDogById(int id);
}
```

RESTful API设计实例

Controller类

```
package com.restfuldemo.controller;

import com.restfuldemo.mapper.DogMapper;
import com.restfuldemo.pojo.Dog;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import java.util.Arrays;
import java.util.List;

@RestController
public class TestController {

    @Autowired(required = false)
    DogMapper dogMapper;

    @GetMapping("dogs")
    public List<Dog> getDogs() {
      return dogMapper.getAllDog();
    }

    @GetMapping("dogs/{id}")
    public Dog getDogById(@PathVariable("id") int id)
    {
      Dog dog=dogMapper.getDogById(id);
      return dog;
    }

    @PostMapping("dogs")
    public boolean addDog(Dog dog) {
      return dogMapper.addDog(dog);
    }

    @PutMapping("dogs/{id}")
    public boolean updateDog(@PathVariable("id")int
    id,@RequestParam("name")String name, @RequestParam(
    "age") int age) {
        Dog dog=dogMapper.getDogById(id);
        dog.setName(name);
        dog.setAge(age);
        return dogMapper.updateDog(dog);
    }

    @DeleteMapping("dogs/{id}")
    public Boolean deleteDog(@PathVariable("id") int
    id){
        return dogMapper.deleteDogById(id);
    }
}
```

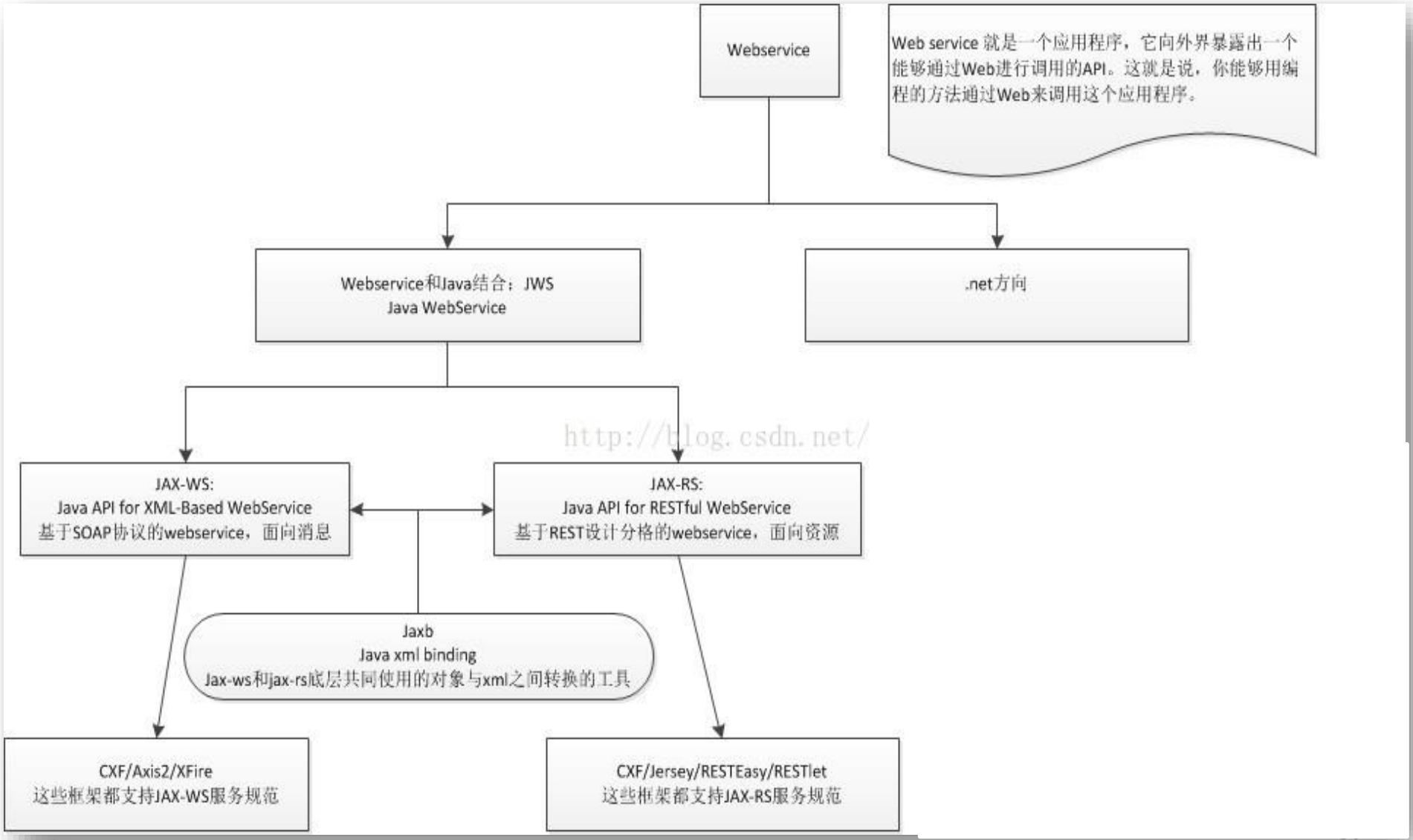
REST与SOAP

- RESTful API及对应协议可以看作Web Service的一种轻量级实现（相对于SOAP）
  - 分别对应一个JSR

    注：JSR:Java Specification Requests

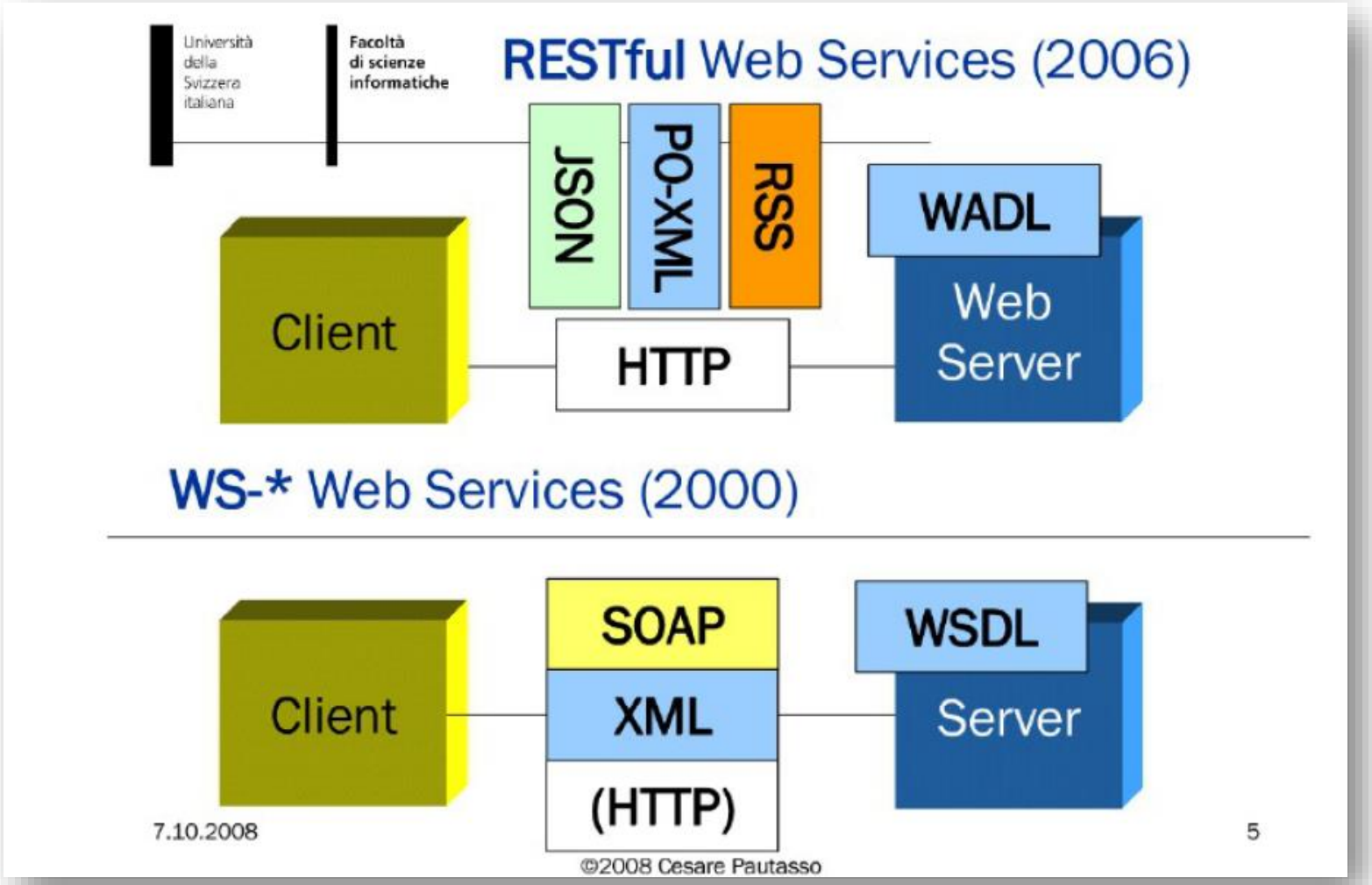
    - JSR-224 (JAX-WS：Java API for XML-Based Web Services)，使用SOAP协议，使用WSDL(Web Service Description Language)来描述；
    - JSR-311 (JAX-RS：The Java API for RESTful Web Services)，使用WADL(Web Application Description Language)描述；
      - ✓ WSDL面向连接的描述，WSDL2.0接口独立;
      - ✓ WADL面向资源的描述，WADL基于HTTP。

REST与SOAP


Webservice
Web service 就是一个应用程序，它向外界暴露出一个能够通过Web进行调用的API。这就是说，你能够用编程的方法通过Web来调用这个应用程序。
Webservice和Java结合：JWS
Java WebService
.net方向
http://blog.csdn.net/
JAX-WS:
Java API for XML-Based WebService
基于SOAP协议的webservice，面向消息
JAX-RS:
Java API for RESTful WebService
基于REST设计分格的webservice，面向资源
Jaxb
Java xml binding
Jax-ws和jax-rs底层共同使用的对象与xml之间转换的工具
CXF/Axis2/XFire
这些框架都支持JAX-WS服务规范
CXF/Jersey/RESTEasy/RESTlet
这些框架都支持JAX-RS服务规范

REST与SOAP

- 轻量性处处体现



REST与SOAP[5]

- SOAP (Simple Object Access Protocol) 是一个严格定义的信息交换协议，用于在Web Service中把远程调用和返回封装成机器可读的格式化数据；
- SOAP并不假定传输数据的下层协议，因此必须设计为能在各种协议上运行；
- 即使绝大多数SOAP是运行在HTTP上，使用URI标识服务，SOAP也仅仅使用POST方法发送请求，用一个唯一的URI标识服务的入口；
  - 以图书馆在线查询管理系统为例，服务提供者必须为每一本书提供一个内部标识，然后可能定义一个listBooks操作来返回一系列图书，一个getBook操作来返回指定的图书，一个createBook操作来向数据库加入新增的图书，一个deleteBook操作来删除作废的图书，每个操作都有各自的参数。
  - 问题：deleteBook操作也要用POST方法来发送，而HTTP协议有更和逻辑的DELETE方法可用。

REST与SOAP[5]

- REST设计：REST为每一个资源（图书）指定一个唯一的URI，而用HTTP的方法GET、POST、PUT、DELETE直观地表示获取、创建、更新和删除图书；
  - 同时图书集合也是和单本的图书不同的资源，如果用/books来代表图书列表，/books/ID来代表标识为ID的图书，那么对/books的GET操作就代表返回整个图书列表，对/books/ID的DELETE操作代表删除指定的图书。

雅虎新闻搜索应用的WADL描述示例[7]

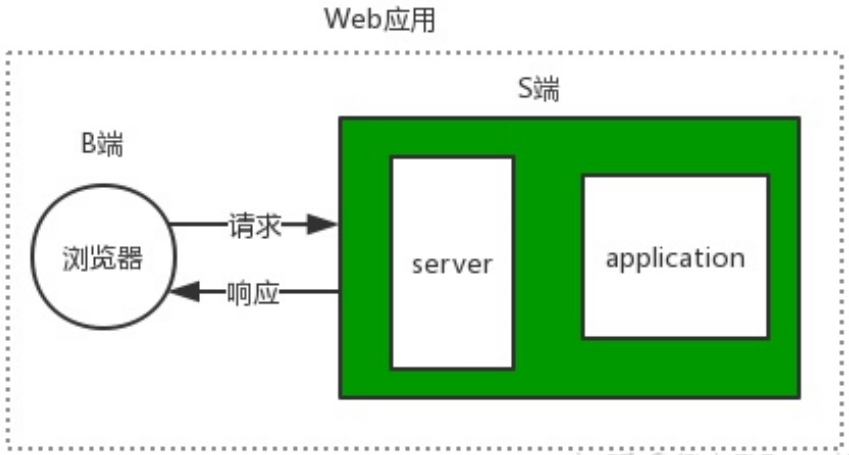
```
1.  <xml version="1.0"?>
2.  <application xmlns:xsi="http:///2001/XMLSchema-instance"
3.      xsi:schemaLocation="wadl/2006/10 wadl.xsd"
4.      xmlns:tns="urn:yahoo:yn" xmlns:xsd="http:///2001/XMLSchema"
5.      xmlns:yn="urn:yahoo:yn" xmlns:ya="urn:yahoo:api" xmlns="wadl/2006/10">
6.      <grammars>
7.          <include href="NewsSearchResponse.xsd" />
8.          <include href="Error.xsd" />
9.      grammars>
10.
11.     <resources base="">
12.         <resource path="newsSearch">
13.             <method name="GET" id="search">
14.                 <request>
15.                     <param name="appid" type="xsd:string" style="query" required="true" />
16.                     <param name="query" type="xsd:string" style="query" required="true" />
17.                     <param name="type" style="query" default="all">
18.                         <option value="all" />
19.                         <option value="any" />
20.                         <option value="phrase" />
21.                     param>
22.                     <param name="results" style="query" type="xsd:int" default="10" />
23.                     <param name="start" style="query" type="xsd:int" default="1" />
24.                     <param name="sort" style="query" default="rank">
25.                         <option value="rank" />
26.                         <option value="date" />
27.                     param>
28.                     <param name="language" style="query" type="xsd:string" />
29.                 request>
30.                 <response>
31.                     <representation mediaType="application/xml" element="yn:ResultSet" />
32.                     <fault status="400" mediaType="application/xml" element="ya:Error" />
33.                 response>
34.             method>
35.         resource>
36.     resources>
37. application>
```

- 第2-5行：开始一个应用的描述，定义在应用描述的其他地方要用到的XML命名空间。
- 第6-9行：定义服务要用到的XML语法（grammars），这里包含了两个XMLSchema文件的引用。
- 第16-45行：描述了雅虎新闻搜索的Web资源和这个资源支持的HTTP方法。
- 第18-43行：描述了 "search" GET方法，
- 第19-36行：描述了方法的输入，
- 第37-42行：描述了方法可能的输入，

REST与SOAP

Web应用

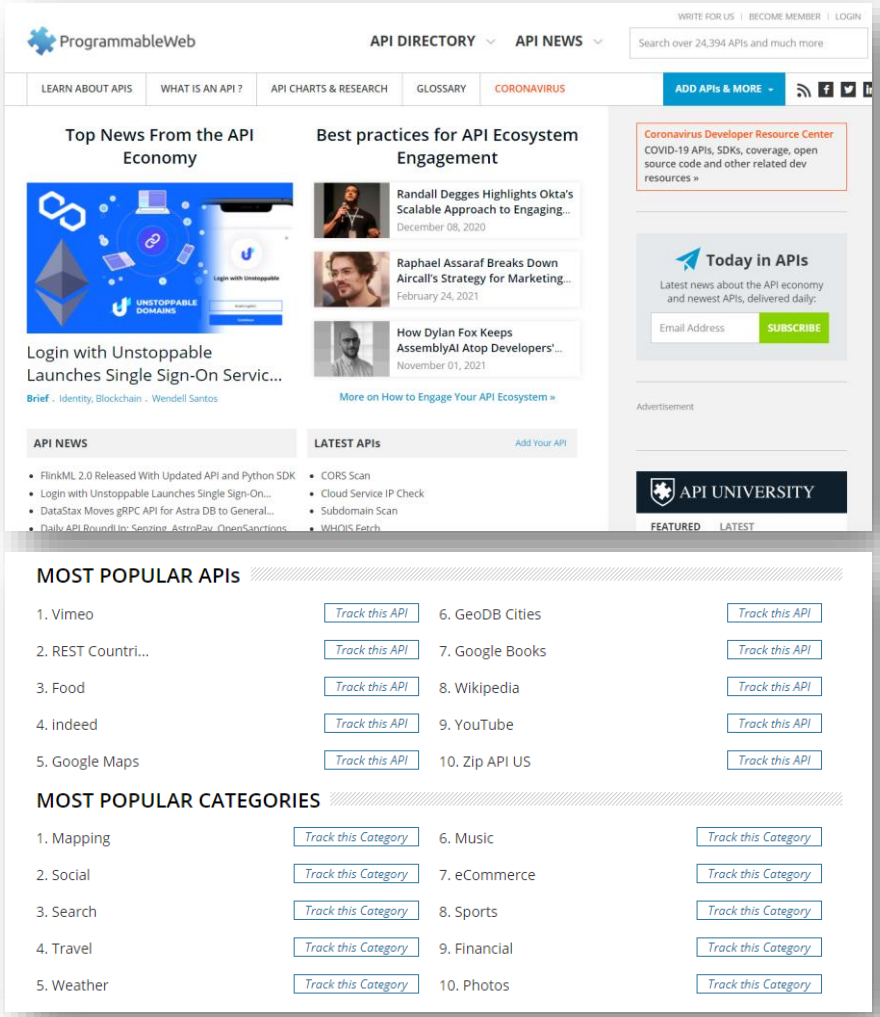
- 延伸讨论：Web 应用（Web APP, Web Application）
- 可以通过Web访问的应用程序，用户只需浏览器即可，不需要再安装其他软件；
- Web应用!=网站，网站是Web应用的展现形式，浏览器是Web应用运行的环境。
- 举例：小说阅读器，Web视频播放器，Web邮箱，网络地图，管理信息系统，网络游戏，学校的教务管理系统



Restful API：平台

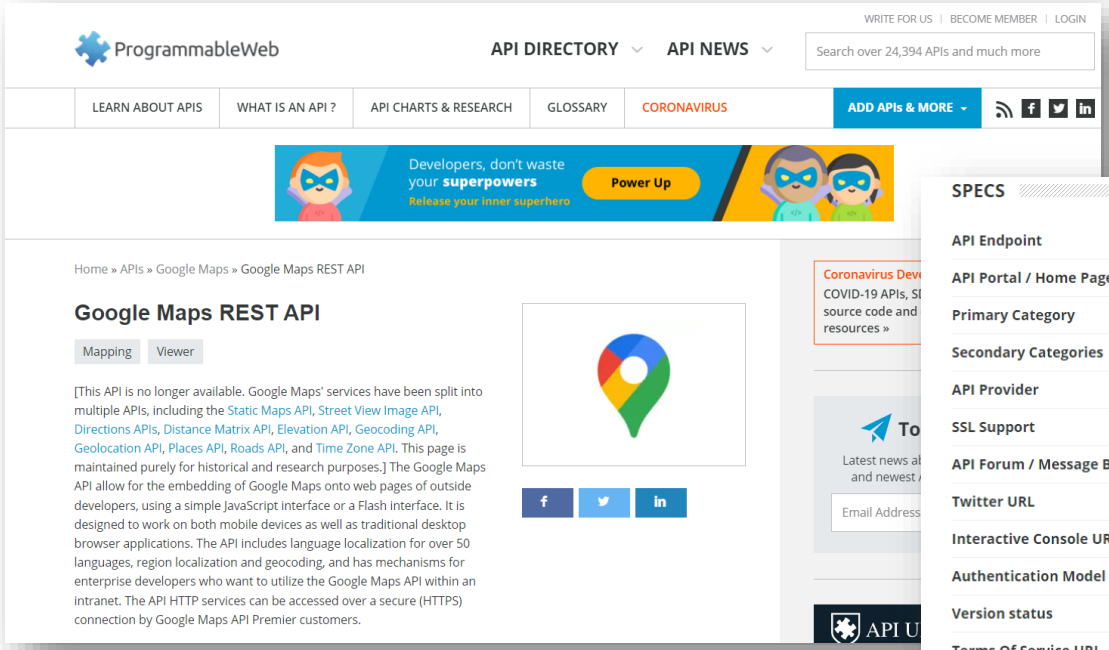
ProgrammableWeb

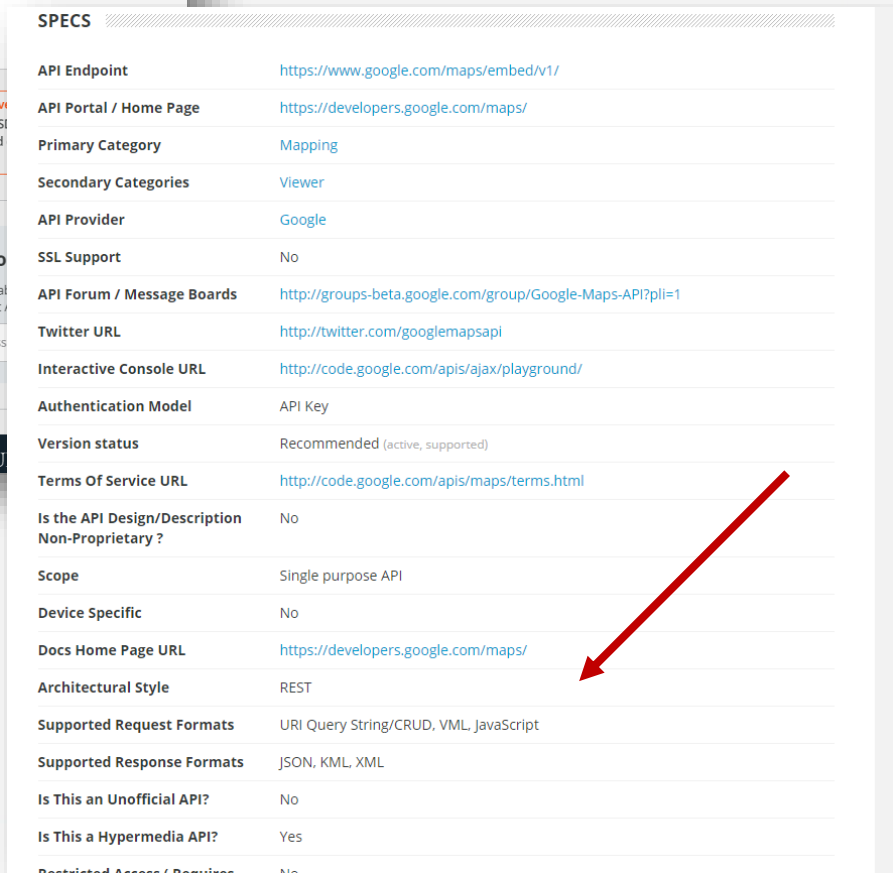
- ProgrammableWeb
  - https://www.programmableweb.com/



Restful API：平台

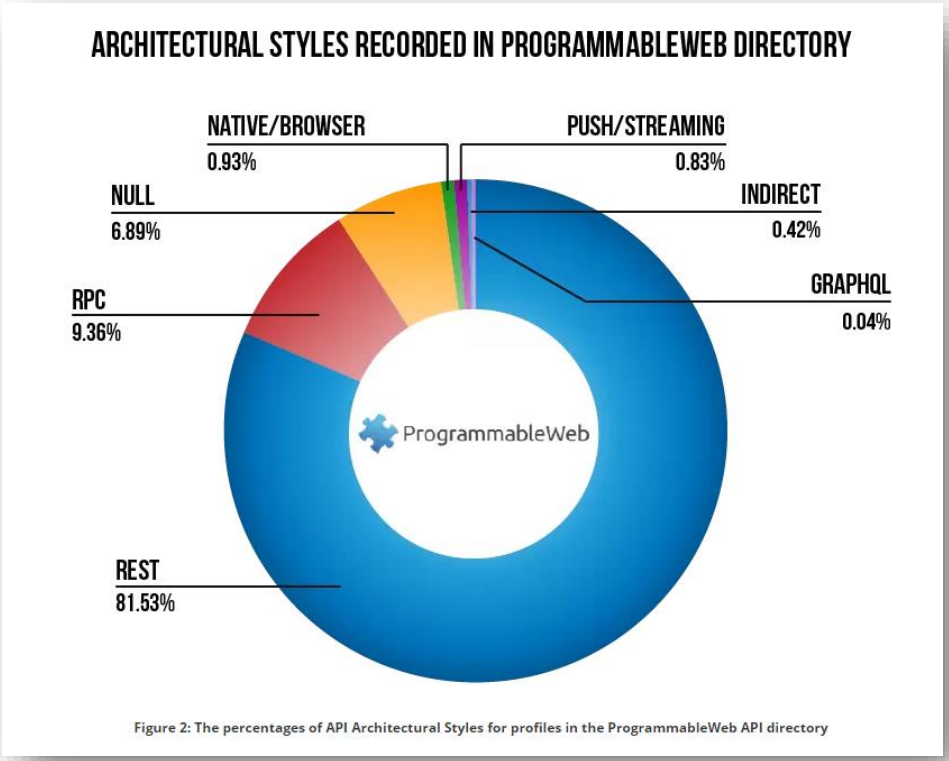
- 举例：Google Map API





Restful API：平台

- API架构分析：RESTful API占绝对优势[4]

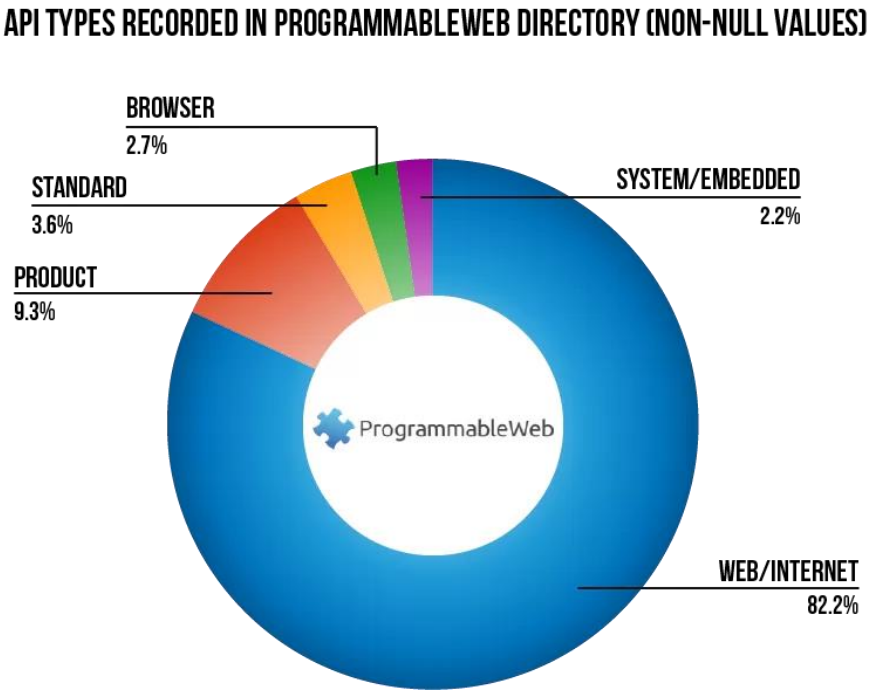


Wendell Santos. Which API Types and Architectural Styles are Most Used?. *https://www.programmableweb.com/news/which-api-types-and-architectural-styles-are-most-used/research/2017/11/26*, 2017

- **REST**
- **RPC (Remote Procedure Call)** 、Web services、SOAP、XML-RPC
- **Native/Browser** - APIs that are accessible on the local system, runtime or OS APIs that are mashable into Web and mobile apps. For example, JavaScript APIs found in the Web browser or a device specific API like one for the accelerometer in a smartphone.
- **Push/Streaming** - Realtime APIs where data is streamed back to the calling system using a technology like Websockets or Webhooks.
- **Indirect** - An API is only accessible indirectly through an SDK. An example is the Evernote API that is only made accessible via the various language SDKs[4].

Restful API：平台

API类型分析：Web占绝对优势[4]



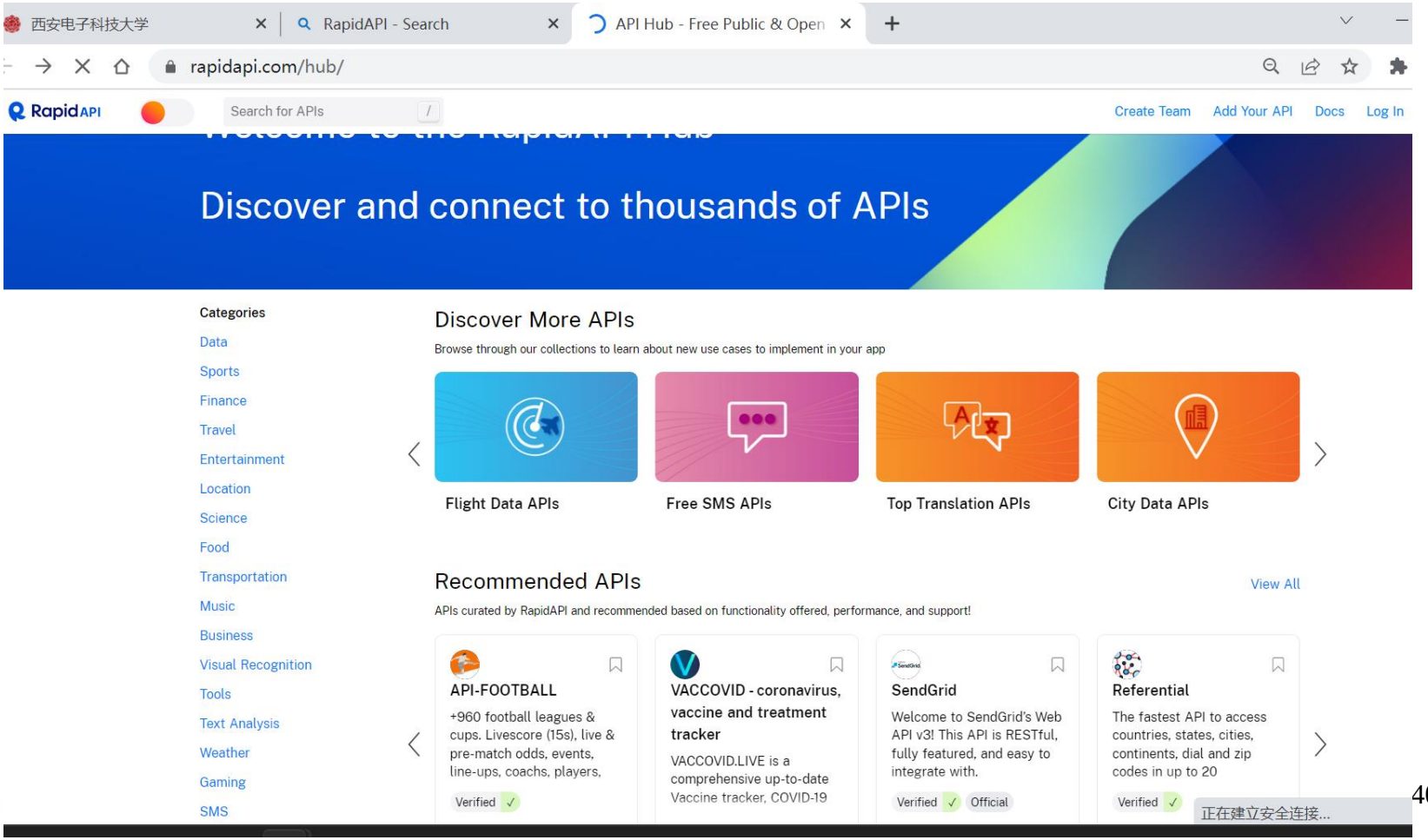
Wendell Santos. Which API Types and Architectural Styles are Most Used?. *https://www.programmableweb.com/news/which-api-types-and-architectural-styles-are-most-used/research/2017/11/26*, 2017

- **Web/Internet**
- **Product**
- **Standard** - Standard APIs (or proposed standards) are non-browser API specifications that other APIs might comply with.
- **Browser** - These are APIs that can be found in the browser itself for enabling Web apps. For example, Mozilla's implementation of the Web Audio API.
- **System/Embedded** - These are APIs that are mashable into Web apps but that might be specific to a local operating system or device.

Restful API：平台

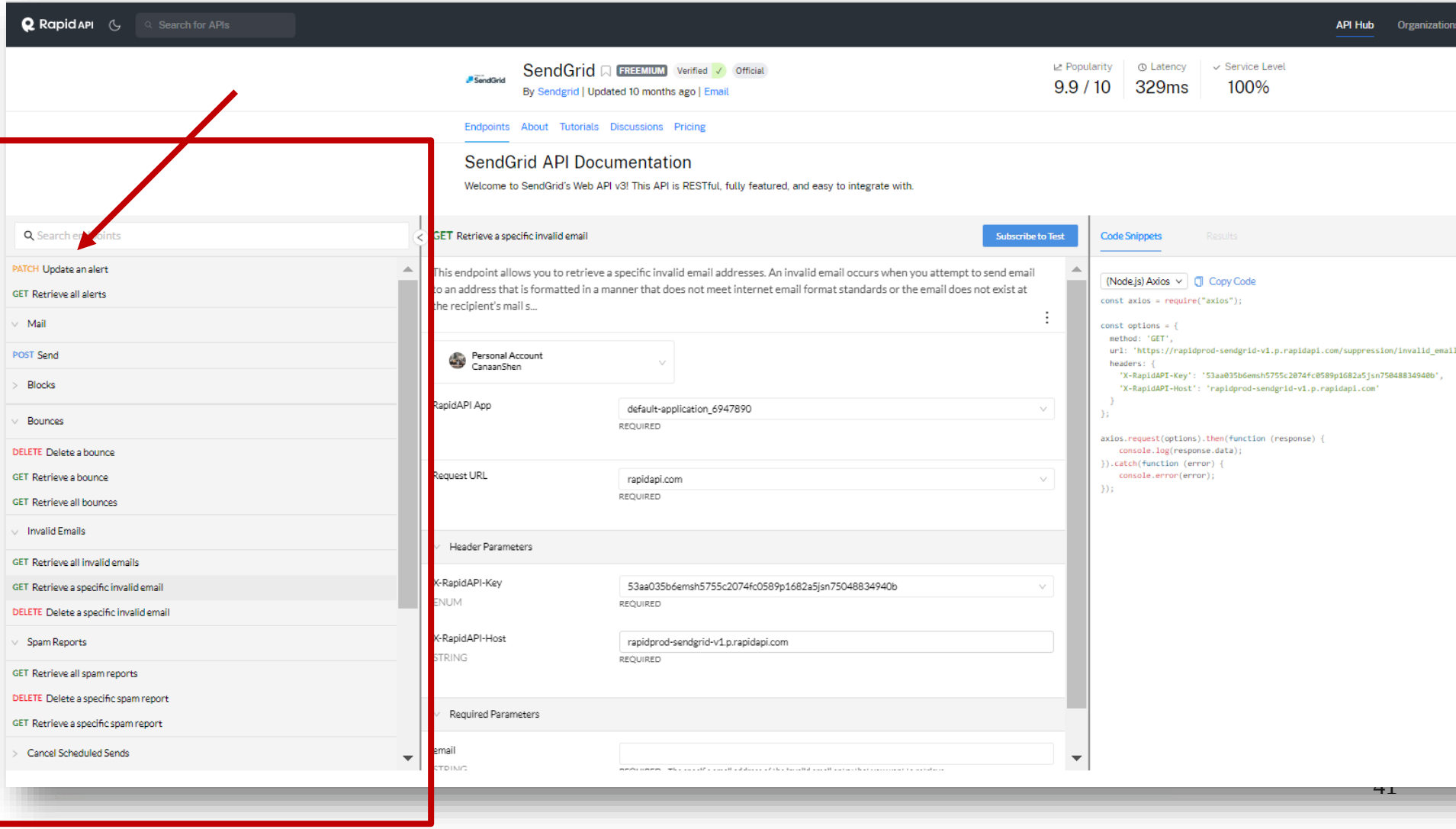
RapidAPI

- https://rapidapi.com/hub/



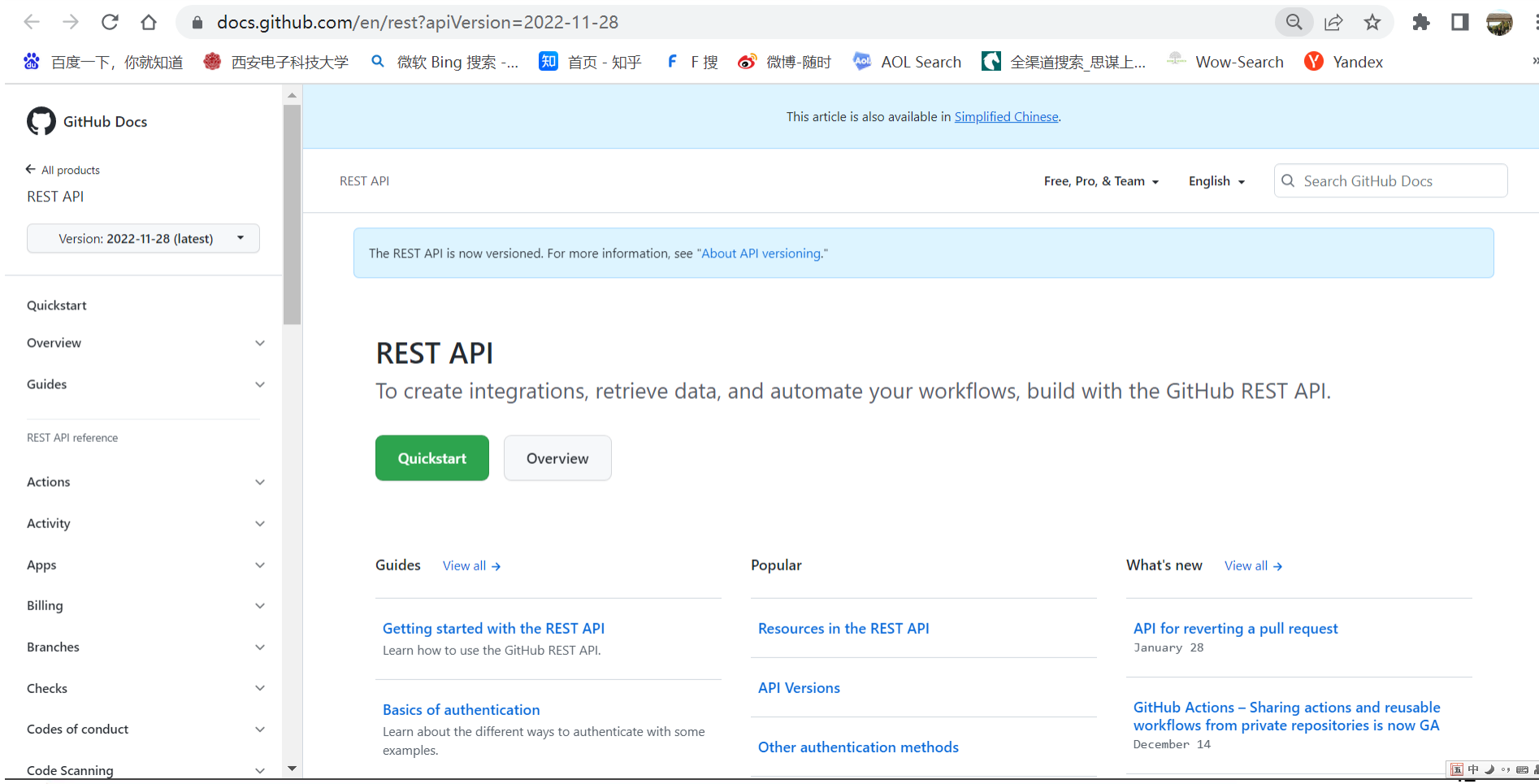
Restful API：平台

- https://rapidapi.com/sendgrid/api/sendgrid/



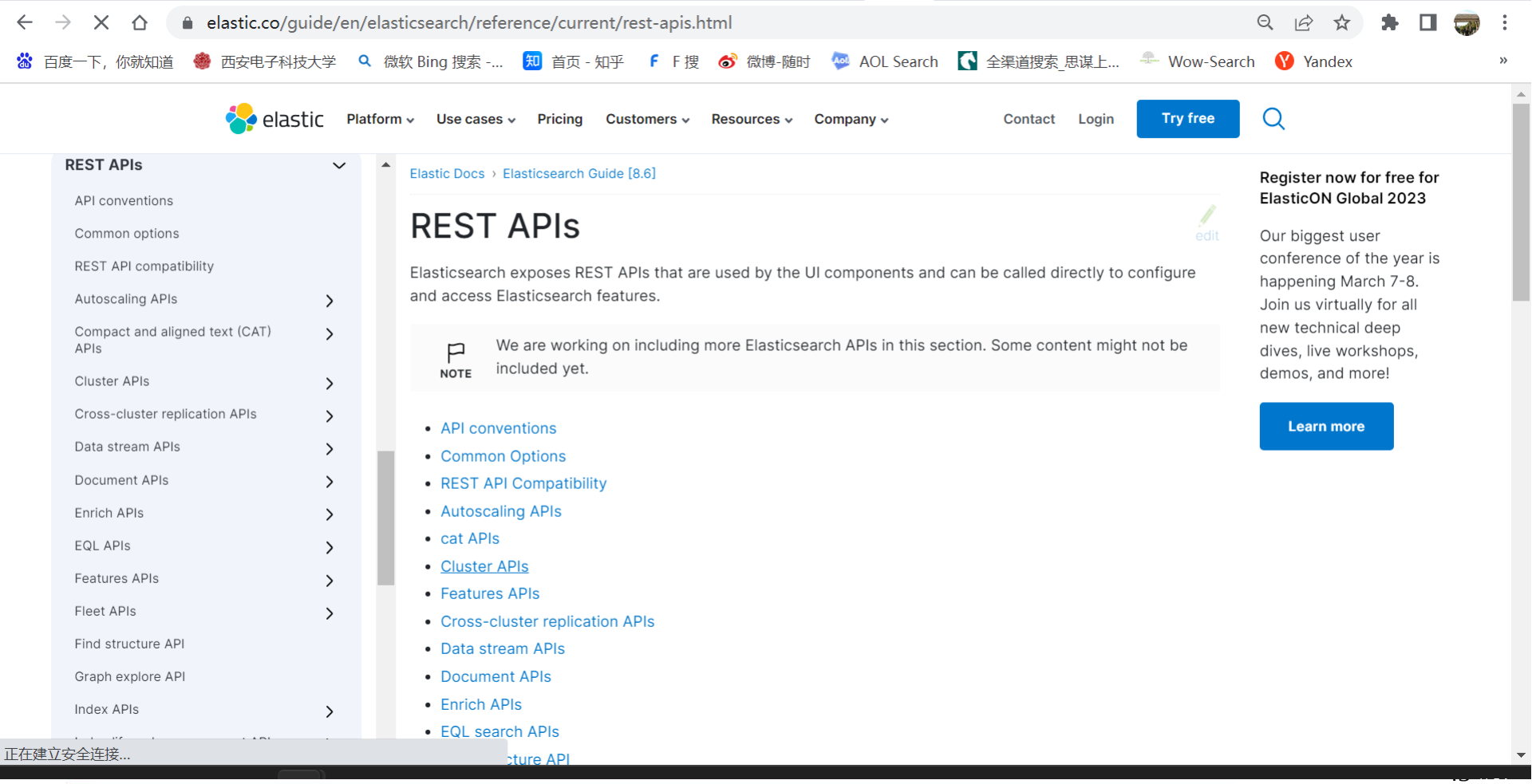
Restful API：厂商

Github



Restful API：厂商

Elasticsearch



总结

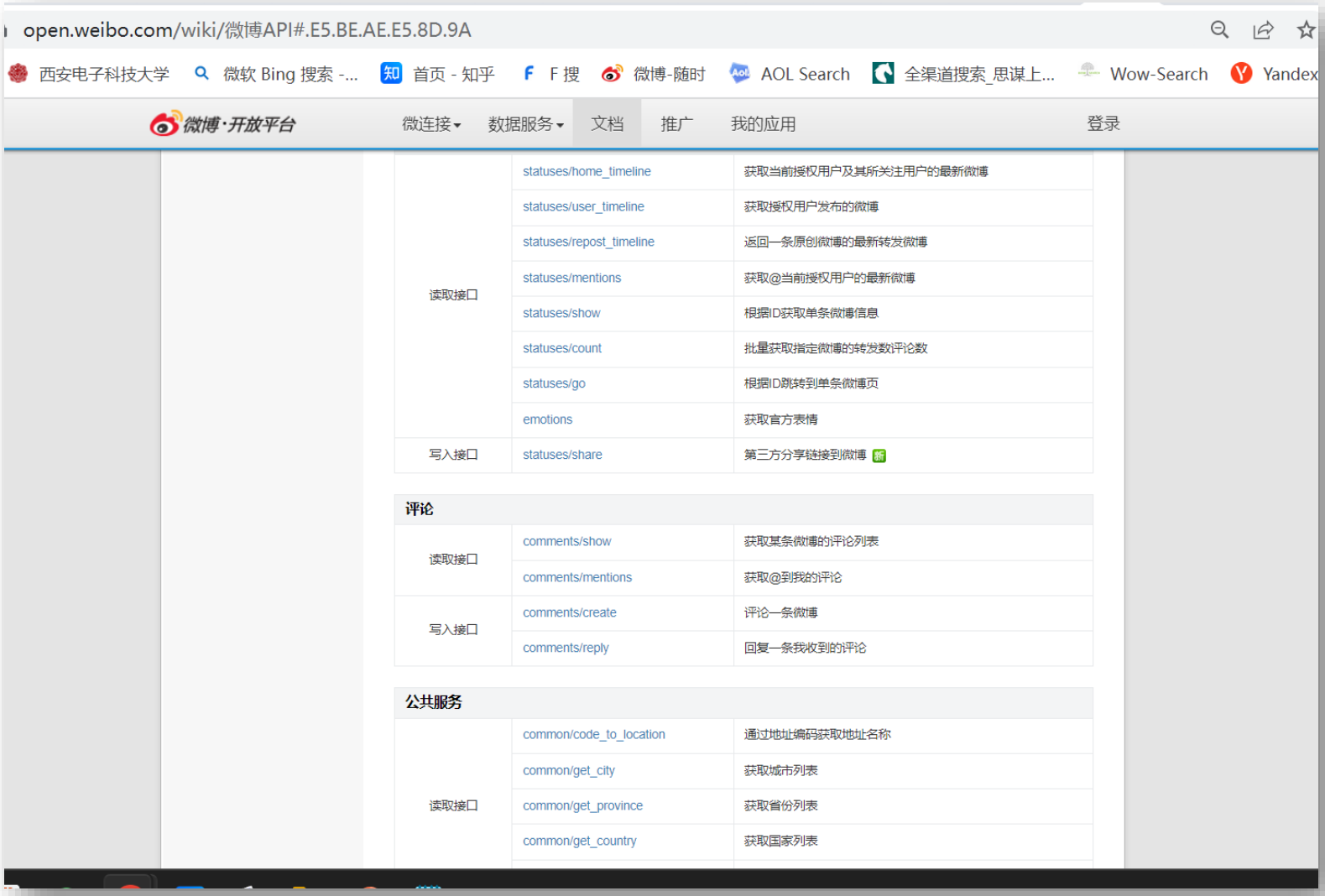
- RESTful API是一种针对分布式应用，特别是Web应用的软件架构风格、设计风格，本身不是标准，提供了一组设计原则和约束条件；
- 同时存在其他的架构风格
  - 分布式对象：RMI、EJB、CORBA
  - 远程过程调用：SOAP、RPC
- Github 的API设计为标准的RESTful API，可以参照
- RESTful API缺点
  - 操作方式：RESTful API根据GET、POST、PUT、DELETE 区分操作资源动作，而HTTP Method 本身不可直接见，如果将动作放到URL的path上反而清晰可见，更利于理解；
  - 某些浏览器对GET,POST之外的请求支持不友好，需要额外处理；
  - 过分强调资源，而实际业务API可能有各种需求比较复杂，仅使用资源的增删改查可能并不能有效满足使用需求。

延伸

- Web API/Open API并不等同于Restful API
- 一部分Web API并没有采用Restful风格，而采用了传统风格
  - 一方面适合业务场景才是第一位；另一方面，Restful API也存在缺点
  - 新浪微博Web API即没有采用Restful风格，但也被大量使用且效果不错

延伸

- https://open.weibo.com/wiki/%E5%BE%AE%E5%8D%9AAPI



延伸

- REST中文资源
  - http://restful.p2hp.com/

课程外参考资料

1. AWeiLoveAndroid. 深入理解什么是RESTful API. https://www.jianshu.com/p/84568e364ee8
2. 阮一峰. RESTful API 设计指南. http://www.ruanyifeng.com/blog/2014/05/restful_api.html
3. 吉诺比利20. RESTful风格API详解. https://blog.csdn.net/tc979907461/article/details/111386432
4. Wendell Santos. Which API Types and Architectural Styles are Most Used?. https://www.programmableweb.com/news/which-api-types-and-architectural-styles-are-most-used/research/2017/11/26, 2017
5. 赶路人. WSDL和WADL区别. https://blog.csdn.net/liuxiao723846/article/details/51611183#commentBox, 2016
6. 林海峰. Web应用. https://zhuanlan.zhihu.com/p/140527886, 2019
7. yong_kang. WADL简介. http://blog.chinaunix.net/uid-9789791-id-1997442.html, 2011

西安电子科技大学

微服务

谢谢！