RESTFul API 设计

- 分 享 人：郭凯
- 分享日期：2018-05-17

目录

CONTENTS

01 关于REST

02 为什么需要RESTFul API

03 RESTFul API 设计

04 RESTFul API 文档

05 Tips

01

关于REST

01 关于REST

起源：

REST这个词，是Roy Thomas Fielding在他2000年的博士论文中提出的

Roy Thomas Fielding：他是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。

“我这篇文章的写作目的，就是想在符合架构原理的前提下，理解和评估以网络为基础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 “
---Roy Thomas Fielding

01 关于REST

REST 定义：

(Resources) Representational State Transfer

（资源）表述性状态传递

01 关于REST

REST 三要素：Resources

Resources：网络上的一个具体信息。它可以是一段文本、一张图片、一首歌曲、一种服务。你可以用一个URI或者URL指向它，每种资源对应一个特定的URI。要获取这个资源，访问它的URI就可以。（资源和表述是一对多的关系）

01 关于REST

REST 三要素：Representation

Representation："资源"是一种信息实体，它可以有多种外在表现形式。我们把"资源"具体呈现出来的形式，叫做它的"表现层"，比如，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，它的具体表现形式，应该在HTTP请求的头信息中用Accept字段指定。

01 关于REST

REST 三要素：State

State：客户端和服务器的交互过程中势必涉及到数据和状态的变化。如果客户端想要操作服务器，必须通过某种手段，让服务器端发生"状态转化"（State Transfer）。客户端用到的手段是HTTP协议。具体来说，就是HTTP协议里面五个常用表示操作方式的动词：GET、POST、PUT/PATCH、DELETE。（HTTP动词不止这5个）

01 关于REST

总结：

REST：互联网软件的架构原则（约束条件）
RESTFul：一种架构风格，一个架构符合REST原则，就称它为RESTful架构

特点：

（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动词，对服务器端资源进行操作，实现"表现层状态转化"。

02

为什么需要RESTFul API

02 为什么需要RESTFul API

微服务

微服务架构（Microservice Architecture）是一种架构概念，旨在通过将功能分解到各个离散的服务中以实现对解决方案的解耦，
微服务的目的是有效的拆分应用（围绕着业务领域组件），实现敏捷开发和部署。

随着组件拆分、服务解耦，各组件之间的调用均是通过接口实现，REST可以让这些拆分后的服务风格统一、便于维护和管理

02 为什么需要RESTFul API

01 SOA (Service Oriented Architecture)

中立性，语言无关，协议跨平台

集中式的服务架构，需要ESB（企业服务总线）（消息传递中间件）统一规范

共用组件，模块相互依赖

服务粒度比较粗

注重组件（模块）可共享（可重用性）

02 微服务

中立性，语言无关，协议跨平台

分布式的服务架构，没有集中统一的总线架构（APIGateway）

服务单独开发和部署，无相互依赖

服务粒度比较细

专注组件（模块）解耦

02 为什么需要RESTFul API

01 Web Service

Web Service 是一组平台独立的，低耦合的web的应用程序。

工作方式：HTTP+XML

协议：SOAP

特点：跨平台、跨语言

02 RESTFul API

RESTFul API是符合REST架构原则的一组应用程序接口（应用程序）。

工作方式：HTTP+Json/Xml/Atom

协议：HTTP

特点：跨平台、跨语言

02 为什么需要RESTFul API

RESTFul API优点

可伸缩性强（HTTP无状态协议、token集中存储）
轻量，直接基于http，不在需要任何别的诸如消息协议
资源描述与视图的松耦合

充分利用 HTTP 协议本身语义：
URL具有很强可读性的，具有自描述性（看Url就知道要什么）
充分使用了Http协议中的动词（看HTTP Method就知道干什么）

02 为什么需要RESTFul API

谁在用

Gmail、Github、Twitter、Facebook、Elastic Search ...

03

RESTFul API 设计

03 RESTFul API 设计

REST 三要素

Resources

Representation

State

03 RESTFul API 设计

设计三要素

Url	-------->	Resources
Content-Type(Response)	-------->	Representation
HTTP Verbs	-------->	State

03 RESTFul API 设计

设计三要素：URL

应该采用Https协议
应该尽量将API部署在专用域名之下，如：api.github.com

版本号建议放在accept头里面或者url后面
Accept: application/vnd.github.v3+json
Accept: application/json;version=v3
或者
https://api.example.com/v1/

03 RESTFul API 设计

设计三要素：URL

URL使用名词而不是动词，且推荐用复数

Good Example：

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

嵌套结构

GET /friends/{group_id}/profile/{friend_id}

03 RESTFul API 设计

设计三要素：URL

Bad Example:

GET	/getProducts
GET	/listOrders
GET	/getOrderById?orderId=1
POST	/updateOrderById?orderId=1
POST	/deleteOrderById?orderId=1
GET	/deleteOrderById?orderId=1
GET	/updateOrderById?orderId=1

03 RESTFul API 设计

设计三要素：Content-Type

这里的Content-Type指的是Response的Content-Type而不是Request的Content-Type

根据Http请求的Accept头部属性协商，优先采用 application/json
（HttpMessageConverter）
（RequestMapping中的produces属性）

03 RESTFul API 设计

设计三要素：HTTP Verbs

常用的HTTP动词有下面五个

GET（SELECT）：从服务器取出资源（一项或多项）；	幂等
POST（CREATE）：在服务器新建一个资源。	非幂等
PUT（UPDATE）：在服务器更新资源（完整更新）。	幂等
PATCH（UPDATE）：在服务器更新资源（部分更新）。	非幂等
DELETE（DELETE）：从服务器删除资源	幂等

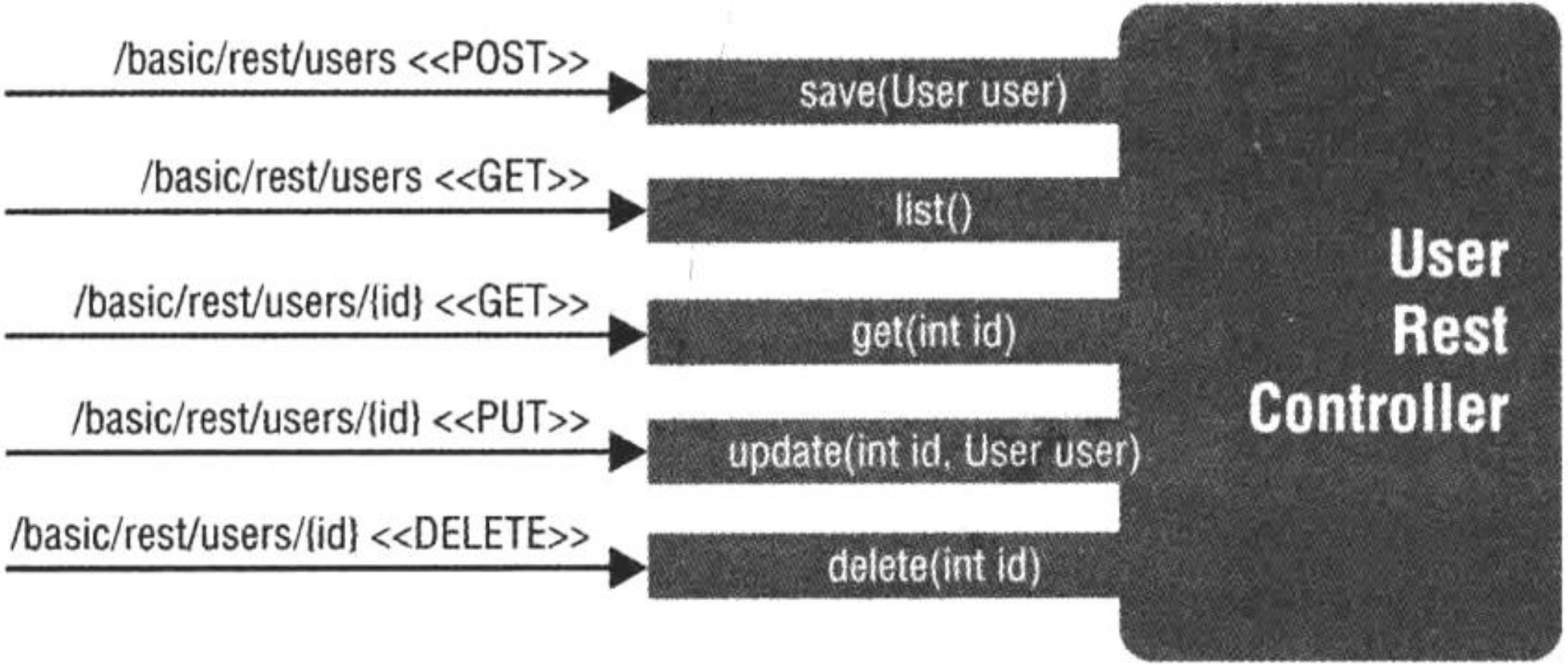
03 RESTFul API 设计

设计三要素：HTTP Verbs

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。

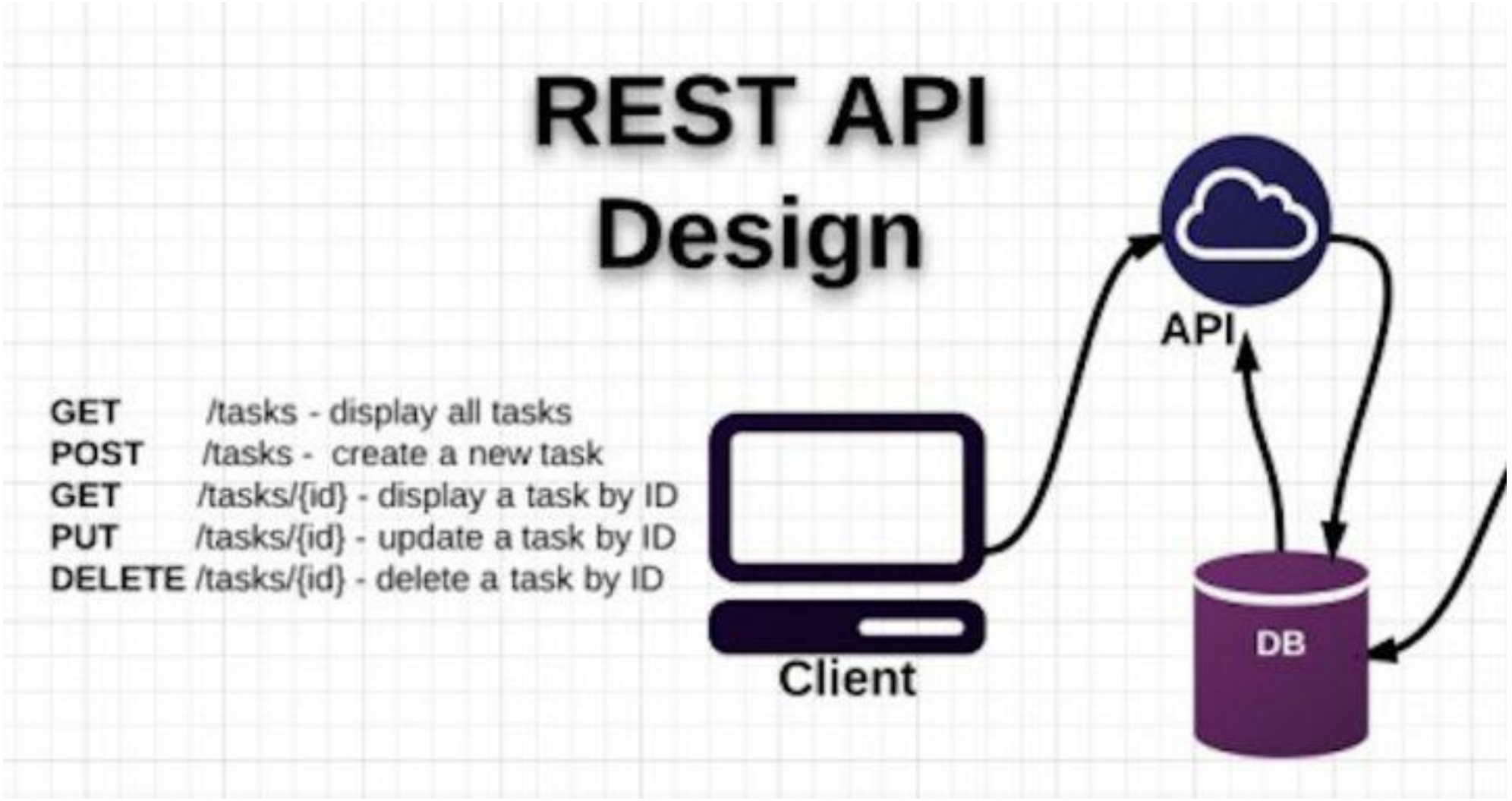
必须保持HEAD和GET的语义（安全）：（不会对资源状态有所改变）
除了使用GET请求获取资源的表述之外，想要做别的事情，例如删除资源，都是对Web及其规则的滥用

03 RESTFul API 设计



图来源：《Spring入门经典》

03 RESTFul API 设计



图来源： 怎样用通俗的语言解释REST，以及RESTful

03 RESTFul API 设计

认证和授权

认证：authentication
授权：authorization

认证：验证用户身份合法性，即用户是谁（比如登录）
授权：在已经登录（或者已注册用户）前提下授予某些资源访问权限，即用户可以做什么（前提是认证通过）

oauth2.0
授权码模式（authorization code）
简化模式（implicit）
密码模式（resource owner password credentials）
客户端模式（client credentials）

最好将认证相关的东西放在头中，包括app_id:token，base64编码，然后把signature放在参数里面

03 RESTFul API 设计

认证和授权

认证：authentication
授权：authorization

认证：验证用户身份合法性，即用户是谁（比如登录）
授权：在已经登录（或者已注册用户）前提下授予某些资源访问权限，即用户可以做什么（前提是认证通过）

一般服务较多的话，可以将认证和授权做成单独的服务，和资源服务区分开

03 RESTFul API 设计

认证和授权

API 的授权应该使用 OAuth 2.0框架

oauth2.0的四种模式：
授权码模式（authorization code）
简化模式（implicit）
密码模式（resource owner password credentials）
客户端模式（client credentials）

03 RESTFul API 设计

返回结果

数据 格式：尽量采用json
状态码：参考HTTP状态码

接口返回值可以参考Spring封装的ResponseEntity：

```
private final HttpHeaders headers;

private final T body;

private final HttpStatus statusCode;
```

03 RESTFul API 设计

结果过滤

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

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

03 RESTFul API 设计

设计进阶： REST 成熟度模型

也叫Richardson成熟度模型
该模型把 REST 服务按照成熟度划分成 4 个层次：

第一个层次（Level 0）的 Web 服务只是使用 HTTP 作为传输方式，实际上只是远程方法调用（RPC）的一种具体形式。SOAP 和 XML-RPC 都属于此类。

第二个层次（Level 1）的 Web 服务引入了资源的概念。每个资源有对应的标识符和表达。

03 RESTFul API 设计

设计进阶： REST 成熟度模型

第三个层次（Level 2）的 Web 服务使用不同的 HTTP 方法来进行不同的操作，并且使用 HTTP 状态码来表示不同的结果。如 HTTP GET 方法来获取资源，HTTP DELETE 方法来删除资源。

第四个层次（Level 3）的 Web 服务使用 HATEOAS。在资源的表达中包含了链接信息。客户端可以根据链接来发现可以执行的动作。

03 RESTFul API 设计

设计进阶： REST 成熟度模型

HATEOAS：Hypermedia as the engine of application state
超媒体即应用状态引擎

超文本利用引用链接其他不同类型（内含声音、图片、动画）的文件，这些具有多媒体操作的超文本和多媒体在信息浏览环境下的结合，它是超级媒体的简称

通过HATEOAS，客户端与应用服务器完全通过超媒体动态提供的网络应用进行交互。除了对超媒体的一般理解之外，REST客户端不需要关于如何与应用程序或服务器进行交互的预先知识，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

03 RESTFul API 设计

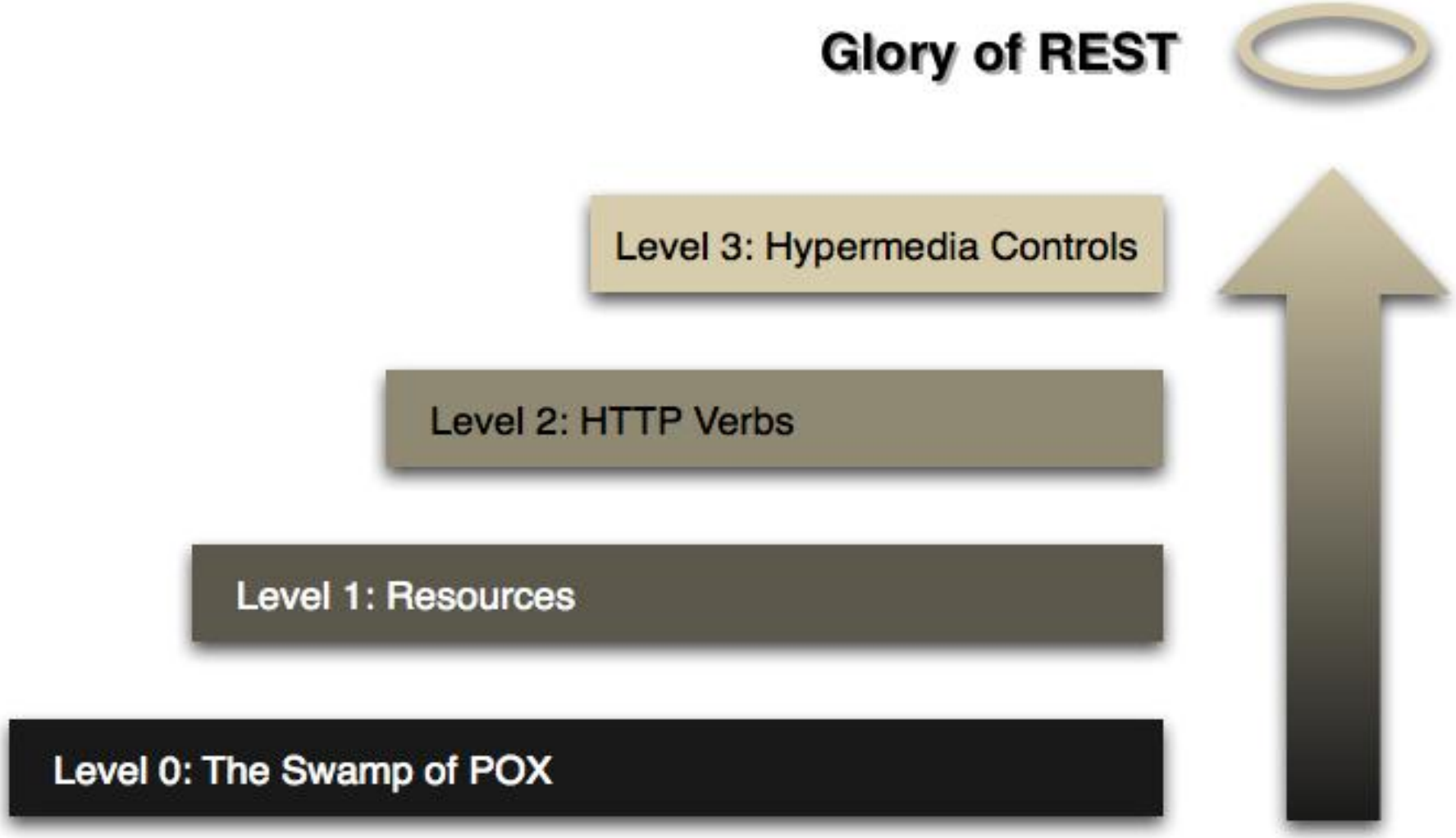
设计进阶： REST 成熟度模型

HATEOAS示例：

访问api.github.com


```
{
  "current_user_url": "https://api.github.com/user",
  "emails_url": "https://api.github.com/user/emails",
  ...
  "team_url": "https://api.github.com/teams",
  "user_url": "https://api.github.com/users/{user}",
}
```

03 RESTFul API 设计



图来源： Richardson成熟度模型

04

API文档

04 API文档

MkDocs

gitbook

swagger

MkDocs

静态站点生成器
python编写
Git（可选）+Markdown
支持搜索（中文支持很差）

发布到自己域名下

GitBook

静态站点生成器
node.js编写
Git+Markdown

托管到gitbook.com
或者发布到自己的域名

Swagger UI

动态生成可视化优雅文档
支持与API资源进行交互
支持多客户端
配置json/yaml即可
可以跟服务端代码集成（SpringFox)(不推荐）

一般跟Mkdocs或者GitBook文档放在同一个域名不同目录下即可

搜索

MkDocs生成的文档对英文搜索支持较好（跨页面）

推荐采用Elastic Search自定义站内搜索

部署

文档线上地址最好以developer开头

https://developer.github.com
https://developers.google.com
https://developer.mozilla.org

05

Tips

05 Tips

5.1 统一接口返回值

建议：多个业务线统一接口返回值，
方便各个微服务之间相互调用的时候对接口返回结果进行处理
方便前端JavaScript处理
方便api网关在限流或者熔断时候返回统一默认兜底数据

05 Tips

5.2 安全

1、认证和授权

Oauth2.0

2、参数校验

防止恶意攻击

提升用户体验

05 总结回顾

5.3 单元测试

驱动和验证功能实现

保护已有的功能不被破坏

建议：测试驱动开发

05 Tips

5.4 日志

建议：

1、日志框架采用slf4j+logback/log4j

2、日志分级、分文件打印

3、dev/test/prod不同环境最低打印日志级别区分开

4、异常根据需要要么在业务逻辑层捕获，要么抛出，由最外层的统一异常捕获，然后用程序定义返回码返回给用户提示，且所有异常日志必须打印在app-warn.log或者app-error.log中，脚本同步告警，不能吞掉异常

05 Tips

5.4 日志

5、除查询外，增删改接口必须在进入方法之后立即打印用户传入的参数，避免用户纠纷

6、日志中用户敏感信息打印注意脱敏

7、打印接口调用时间，定期分析QPS/TPS，及早发现问题以及考虑是否扩容等

8、打印调用Ip，方便后期扩展（Ip限流、Ip黑名单）

参考文档：

1、《Spring实战》
2、《Spring入门经典》
3、《REST实战》
4、阮一峰：理解RESTFul架构
5、阮一峰：RESTFul API设计指南
6、成 富：使用 Spring HATEOAS 开发 REST 服务
7、知乎话题：怎样用通俗的语言解释REST，以及RESTful
8、知乎话题：SOA和微服务架构的区别

THANK YOU!

说明：