RESTful 架构基础

ImportNew Java后端 2019-10-31

来自 | 唐尤华

译自 | dzone.com/refcardz/rest-foundations-restful

上篇 | 终于有人把 Docker 讲清楚了

REST（Representational State Transfer）架构风格是一种世界观，把信息提升为架构中的一等公民。 通过 REST 可以实现系统的高性能、可伸缩、通用性、简单性、可修改性和可扩展等特性。这篇文章解释了主要的 HTTP 操作，对 HTTP 响应码进行描述，并列举相关开发库和框架。此外，本文还提供了额外的资源，对每个主题进行了更深入的探讨。

1. 简介

REST 架构风格不是一种可以购买的技术，也不是一个可以添加到软件开发项目中的开发库。 首先也是最重要的，REST 是一种世界观，把将信息提升为构建架构中的一等公民。

Roy Fielding 的博士论文“架构风格和基于网络的软件架构设计”介绍和整理了“RESTful”系统的思想和相关术语。 这是一篇学术论文，虽然使用正式语言，但是仍然易于理解并且提供了实践基础。

总结一下，RESTful 通过体系结构的特定选择能从部署的系统中获得理想特性。 尽管这种风格定义的约束细节并没有为所有场合设计，但是的确可以广泛适用。

由于 Web 对消费者偏好有多重影响，REST 风格的倡导者鼓励企业组织在其边界内使用相同原则，就像他们在面向外部客户的网页上做的那样。本文将讨论现代 REST Web 实现中的基本约束和属性。

1.1 基础概念

REST 表示什么含义？以无状态方式传输、访问和操作文本数据。当正确部署后，REST 为互联网上不同应用程序之间提供了一致的互操作性。无状态（stateless）这个术语至关重要，它使得应用程序可以用不可知的方式进行通信。RESTful API 通过统一资源定位符地址（URL）公开服务。URL 名称将资源的区分为接受内容或返回内容。RFC 1738中定义了 URL scheme，可以在这里找到: https://tools.ietf.org/rfc/rfc1738.txt

RESTful URL 类似于下面这个 library API：

```
http://fakelibrary.org/library
```

实际公开的不一定是某种任意的服务，而是代表对消费者有价值的信息资源。URL 作为资源句柄，可以请求、更新或删除内容。

开始把服务发布到某个地方，然后开始与 REST 服务进行交互。 返回的内容可能是 XML、JSON 格式，或者更确切地说是像 Atom 或自定义 MIME 类型等超媒体格式。虽然一般建议尽可能重用现有的格式，但是对正确设计的媒体类型正在变得越来越宽容。

需要请求资源的时候，客户机会发一个超文本传输协议（HTTP）GET 请求，例如在浏览器中键入一个 URL 然后点击回车，选择书签，或者点击锚引用链接。
通过编程方式与 RESTful API 交互，有数十个客户端 API 或工具可供选择。使用 curl 命令行工具，可以输入以下命令：

```
$ curl http://fakelibrary.org/library
```

```
1 $ curl http://fakelibrary.org/library
```

上面的命令使用默认格式，但你可能不需要这种格式的信息。幸运的是 HTTP 有一种机制，可以指定返回信息的格式。 在请求中指定 "Accept" 头，如果服务器支持这种格式，会以指定的格式返回。 这个过程称为内容协商，这是 HTTP 中未被充分利用的功能之一，可以使用一个类似于上面例子中的 curl 命令来指定：

```
1 $ curl –H "Accept:application/json" http://fakelibrary.org/library
```

由于资源名称与内容格式是独立的，从而让请求不同格式信息成为可能。虽然 REST 中的 “R” 的含义是 “表现”而非“资源”，但是应该在构建系统时允许客户端指定请求的内容格式，请牢记这一点。在我们的例子中 library API 可能包含以下 URL：

- http://fakelibrary.org/library：图书馆基本信息，搜索图书、DVD等相关资源基本功能的链接。
- http://fakelibrary.org/book：存放书籍的“信息空间”。从概念上说，这里可能会存放所有的书籍。显然，如果这个问题得到解决，我们不会希望返回所有图书，而是希望通过类别、搜索关键词等来检索图书。
- http://fakelibrary.org/book/category/1234：在书籍的信息空间里，我们可以指定类别浏览，例如成人小说、儿童书籍、园艺书籍等。使用杜威十进制图书分类法是可行的，但我们也可以想象自定义分组。问题的关键在于，这种“信息空间”可能是无限的，而且可能收到人们实际关心的信息类型影响。
- http://fakelibrary.org/book/isbn/978-0596801687：提到某本具体的书，应该包括书名、作者、出版商、系统中的拷贝数、可用拷贝数等信息。

> 译注：杜威十进制图书分类法由美国图书馆专家麦尔威·杜威发明，于1876年首次发表，历经22次的大改版。该分类法以三位数字代表分类码，共可分为10个大分类、100个中分类及1000个小分类。

就图书馆用户而言，上面提到的这些 URL 可能就是只读的，但是图书馆员使用应用程序时实际上可以操作这些资源。

例如添加一本新书，可以向 main/book 地址 POST 一个 XML。使用 curl 提交，看起来可能像这样：

```
1 $  curl –u username:password -d @book.xml -H "Content-type: text/xml" http://fakelibrary.org/book
```

此时，服务器可能会对提交的内容进行校验，创建与图书相关的记录，并返回响应代码201——表示已创建新资源。新资源的 URL 可以在响应的 Location 头中找到。

RESTful 请求一个重要特性：每次请求都包含了充足的状态信息来响应请求。这为服务器的可见性和无状态创造了条件，并为扩展系统和识别发送的请求内容提供了理想特性。对于缓存结果也非常有帮助。服务器地址和请求状态组合成可计算的 hash 键值，并形成一个结果集：

```
1 http://fakelibrary.org + /book/isbn/978-0596801687
```

接下来我们会先介绍 GET 请求。客户端在需要时发出 GET 请求获取指定资源。客户端可以在本地缓存请求结果，服务器可以在远程缓存结果，系统的中间层可以在请求链路中间缓存结果。这是一个与具体应用程序无关的特性，可以加入系统设计中。

正因为可以操作资源，也就意味着并不是每个人都可以这样做。我们完全可以建立一个防护模型，要求用户在操作前验证身份，证明他们具有该操作的授权。在本文的最后，将提供一些提升 RESTful 服务安全性的内容。

2. REST 和 SOAP 比怎么样?

> SOAP：简单对象访问协议（Simple Object Access Protocol）。是交换数据的一种协议规范，是一种轻量的、简单的、基于XML的协议。一条 SOAP 消息就是一个普通的 XML 文档，包含必需的 Envelope 元素、可选的 Header 元素、必需的 Body 元素和可选的 Fault 元素。

把 REST 与 SOAP 划等号是错误的。在这两者之间进行比较，带来的困扰远多于好处。简单来说，它们不是一回事。尽管可以用这两种方法解决许多架构问题，但是它们不能相互替换。

这种混淆很大程度上源于对“REST 是通过 URL 调用 Web 服务”这句话的误解。 这种观点与 RESTful 架构的功能相距甚远。如果不全面深入理解 RESTful 的架构实现，就很容易误解 REST 实践的本意。

利用 REST 的最佳方式，是将生产和消费过程中的信息与技术分离实现解耦，进而更好地管理系统，让架构具备以下特性：

- 高性能
- 可扩展
- 通用
- 简洁
- 可修改
- 可扩展

这并不是说，基于 SOAP 构建的系统不能具备上述特性。 而是当技术、组织或过程的复杂性造成不能在单个事务中完成请求的生命周期时，这种情况 SOAP 能够发挥最佳效果。

3. Richardson 成熟度模型

Leonard Richardson 引入了一种成熟度模型，部分阐述了 SOAP 与 REST 之间的区别，并提供一种对不同类型的系统进行分类的框架。许多人不恰当地称之为 “REST”。 可以将这种分类看作系统中不同 Web 技术组件紧密程度的度量标准：包括信息资源、HTTP 作为应用层协议和作超媒体作为控制媒介。

等级	采纳
0	基本上就是 SOAP。没有信息资源，HTTP 被用作传输协议，也没有超媒体的概念。 结论，REST 和 SOAP 是两种不同的方案
1	用到 URL 但并不总是作为信息资源使用，所有内容都通过 GET 请求，包括更新服务器状态的请求。大多数人刚接触 REST 时构建的系统通常是这样的
2	使用 URL 表示信息资源。HTTP 作为应用层协议，有时也用作内容协商。大多数面向 Internet 的 “REST” Web 服务实际上位于这个级别，因为它们只支持非超媒体格式
3	使用 URL 表示信息资源。HTTP 作为应用层协议，也用来进行内容协商。使用超媒体进行客户端的交互

称其为“成熟度模型”似乎意味着应该只构建“成熟度”最高的系统。这种看法是不合适的。第2级是有价值的，从2级向3级转变通常只是采用了一种新的 MIME 类型。然而，从0级到3级的转变要困难得多，因此增量式升级转变通常也会增值。

首先，确定希望公开哪些信息资源。采用 HTTP 作为处理这些信息资源的应用协议，包括内容协商。 接下来，当一切就绪时，使用基于超媒体的 MIME 类型，这样就可以充分享受 REST 的好处了。

4. 动词

动词是用来与服务器资源交互的方法或操作。RESTful 系统中有限的动词让刚接触该的使用者感到困惑和沮丧。看似武断和不必要的约束，目的是鼓励以应用程序无关的形式提供可预测的行为。通过明确、清晰地定义这些动词的行为，客户端可以在网络中断或故障时自主处理。

精心设计的 RESTful 系统主要使用4个 HTTP 动词。

4.1 GET

GET 请求是最常用的 Web 动词。GET 请求将命名资源从服务器传输到客户端。尽管客户端不需要知道请求的资源内容，但是请求返回的结果是带元数据标记的字节流，这表明客户端应该知道如何解释资源。在 Web 中通常用 “text/html” 或 “application/xhtml+xml” 表示。 正如之前提到的那样，只要服务器支持，客户端可以通过内容协商提前指定请求的返回格式。

GET 请求关键点之一，不要修改服务器端的任何内容。 这是一个基本的安全要求，也是不熟悉 REST 的开发者犯的最大错误之一。你可能会遇到这样的 URL：

```
http://example.com/res/action=update?data=1234
```

不要这样做！ 由于 GET 请求安全性允许缓存请求，这会让正在构建的 RESTful 系统陷入混乱。 GET 请求也意味着幂等性，即多次请求不会对系统产生任何影响。这是基于分布式基础设施的一个重要特性。如果进行 GET 请求时被打断，由于幂等性，客户端可以再次发起请求。这点非常重要。在设计良好的基础结构中，客户端可以从任意应用程序发起请求。虽然一定会有与应用程序相关的特定行为，但是加入与应用程序无关的行为越多，系统就会越有弹性，也更容易维护。

4.2 POST

在辨别 POST 和 PUT 动词意图的时候，情况开始变得不那么清晰。 根据定义，二者似乎都可以被客户端用来创建或更新服务器资源，然而它们的用途各有不同。

当无法预测请求创建的资源的标识时，客户端会使用 POST 请求。 在新增雇员、下订单或提交表单的时候，我们无法预测服务器将如何命名正在创建的资源。这就是为什么将资源提交给类似 Servlet 这样的程序处理。接下来，服务器会接受请求、校验请求、验证用户凭据等。成功处理后，服务器将返回 201 HTTP 响应代码，其中包含一个 “Location” 头，代表新创建的资源的位置。

注意： 有些人将 POST 视为创建资源的 GET 会话。他们会对创建的资源通过 body 返回200，而不是返回201。 这似乎是避免二次请求的一种快捷方式，但是这种做法混合了 POST 和 GET，让缓存资源的潜在影响变得微妙。 尽量避免因为走捷径而牺牲大局。短期看这似乎是值得的，但随着时间的推移，这些捷径叠加起来可能会带来不利的影响。

POST 动词的另一个主要用途是“追加（Append）”资源信息，即增量编辑或部分更新，而不是提交完整的资源。 这里应使用 PUT 操作。对已知资源使用 POST 更新，可用于向订单添加新送货地址或更新购物车中某个商品的数量。

由于是更新资源的部分信息，**POST 既不安全也不幂等**。

POST 的最后一种常见用法是提交查询。将查询的内容或表单内容进行 URL 编码后提交给服务执行查询。通常可以直接返回 POST 结果，因为没有与查询相关的标识。

注意： 建议将这样的查询转换为信息资源本身。如果采用 POST 查询，可以考虑采用 GET 请求，后者支持缓存。 你可以与其他人分享这个链接。

4.3 PUT

由于 HTML 表单目前还不支持 PUT，许多开发人员基本上会忽略 PUT 动词。 然而，PUT 有一个重要作用并且是 RESTful 系统完整愿景的一部分。

客户端可以向指定 URL 发 PUT 请求，服务器用请求中的数据执行覆盖操作。 PUT 请求在某种程度上是等幂的，而 POST 更新不是。

如果客户端在 PUT 覆盖请求时被打断，由于重新发送覆盖操不会造成任何后果，因此可以再次发送。 客户端具备管理状态能力，所以直接重发覆盖命令即可。

注意： 这种协议层处理并不意味着要取消更高级别（如应用层）的事务，但是同样地，它也是一种体系结构上理想的属性，可以在应用层以下使用。

如果客户端能够提前了解资源的标识，那么 PUT 也可用于创建资源。正如我们在 POST 部分中讨论的那样，通常不会出现这种情况。但是如果客户端能够控制服务器端信息空间，那么这种操作也是合理的。

4.4 DELETE

在公共网络上 DELETE 动词没有被广泛使用（谢天谢地!）。 然而，对于控制信息空间非常有用，它是资源生命周期中非常有用的一部分。

DELETE 请求意在实现等幂。可能由于网络故障 DELETE 请求被打断，这时我们希望客户端继续尝试。 第一次请求无论成功与否，资源都应该返回204（无指定内容）。对之前已删除的资源或不存在的资源可能需要一些额外处理，两种情况都应该返回404。一些安全策略要求为不存在的和已删除的资源返回404，这样 DELETE 请求就不会泄漏有关资源是否存在的信息。

还有另外三个没有广泛使用但是有价值的动词。

4.5 HEAD

HEAD 动词用来请求资源，但不实际检索。客户端可以通过 HEAD 检查资源是否存在，并检查资源相关的元数据。

4.6 OPTIONS

OPTIONS 动词也可以用来查询服务器相关资源的情况，方法是询问哪些其它动词可用于该资源。

4.7 PATCH

最新的动词 PATCH 直到2010年才正式采纳为 HTTP 的一部分。旨在提供一种标准化方式来表示部分更新。PATCH 请求通过标准格式让交互的意图更明确。这是推荐使用 PATCH 而非 POST 的原因，尽管 POST 可以用于任何事情。 IETF 发布了 RFC 文档，定义用于 PATCH 操作的 XML 和 JSON。

如果客户端 PATCH 请求的 header 中带 If-Match，则此部分为幂等更新。 可以重试中断的请求，因为如果第一次请求成功，那么 If-Match header 会不同于新状态。如果相同，则未处理原始请求可应用 PATCH。

5. 响应码

HTTP 响应码为我们在客户端和服务器之间的对话提供了丰富的请求状态信息。大多数人只熟悉一般意义上的200、403、404或者500，但是还有更多有用的代码可供使用。这里表格并不全面，但是它们涵盖了许多在 RESTful 环境中应该考虑使用的最重要代码。数字可按照以下类别分组：

- 1XX：信息类
- 2XX：操作成功
- 3XX：重定向
- 4XX：客户端错误
- 5XX：服务器错误

第一组响应码表明客户端的请求格式正确且处理成功。具体操作如下表所示：

响应代码	描述
200	OK。请求已成功执行，回应内容取决于所调用的动词。
201	已创建。请求已成功执行，在执行过程中创建了一个新资源。响应 body 为空或包含所创建资源的 URI。响应中的 Location 头也应该指向 URI
202	已接受。请求有效并已接受，但尚未得到处理。可通过请求轮询，响应结果中 URI 提供状态更新。这种方式支持异步 REST 请求
204	没有请求的内容。请求已成功处理，但服务器没有任何响应结果，客户端不应更新显示

表1 成功的客户端请求

响应代码	描述
301	永久移除。请求的资源不再位于指定的 URL，新的 Location 应该在响应头中返回。只有 GET 或 HEAD 请求应当重定向到新位置。如果可能，客户端应该更新自己保存的书签
302	已找到。请求的资源已在其它临时位置找到，并应该在响应头中返回该位置。只有 GET 或 HEAD 请求应该重定向到新位置。客户端无需更新书签，因为资源会返回对应的 URL。
303	参见其他信息。该响应码被 W3C 技术架构小组（TAG）重新解释成一种对非网络可寻址资源的有效请求方式。这是语义化 Web 中的一个重要概念，可以向个人、概念、组织等提供 URI。能够在 Web 中找到和不能在 Web 中找到的资源是有区别的。如果客户端收到的响应码是303而不是200，就能分辨这种差别。重定向的位置在响应的 Location 头中。头信息可能包含相关资源的文档引用，也可能包含相关资源的一些元数据

表2 — 客户端重定向请求

表3中的响应代码表示客户端请求无效，如果条件不发生变化，重新请求仍无法处理。这些故障可能有请求格式错误、未授权的请求、请求的资源不存在等。

响应代码	描述
405	方法不允许
406	请求不接受
410	资源不存在
411	需要指定长度
412	前提条件失败
413	Entity 太大
414	URI 超长
415	不支持的媒体类型
417	预期失败

表3 客户端请求错误

最后，表4中的响应代码表示服务器暂时无法处理客户端请求（可能仍然无效）。客户端应当在将来的某个时候重新请求。

响应代码	描述
500	内部服务器错误
501	未实现
503	服务不可用

表4 服务器处理请求错误

服务根据其自身功能要求具有不同程度的可扩展性。

注意： 试试响应代码418，它会返回简洁有力的回复："我是一个茶壶。"

5.1 REST 资源

5.1.1 论文

Fielding 博士的论文《架构的风格与基于网络的软件架构设计》是对 RESTful 思想的主要介绍：http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

5.1.2 RFC 规范

REST 常见用法的技术规范由**国际互联网工程任务组（IETF）定义，按照请求评议（RFC）**流程完善。规范由数字定义，并随着时间推移不时更新版本，以替换已经过时的文件。目前，这里有最新的相关 RFC 文件。

5.1.2.1 URI

RFC 3986定义了 URI 命名方案的通用语法。URI 是一种命名方案，包含了对其他如网址、支持名字子空间等编码方案。 网址：http://www.ietf.org/rfc/rfc3986.txt>

5.1.2.2 URL

Url 是 URI 的一种形式，其中嵌入了充足的信息（通常是访问方案和地址），用于解析和定位资源统一资源定位符。 网址：http://www.ietf.org/rfc/rfc1738.txt

5.1.2.3 IRI

国际化资源标识符（IRI）在概念上是一个用 Unicode 编码的 URI，用于在 Web 上使用的标识符中支持世界上各种语言的字符。IETF 选择创建一个新的标准，而不是改变 URI 方案本身，以避免破坏现有的系统并明确区分这两种方法。 那些支持 IRI 的人故意这样做。还定义了在 IRI 和 URI 之间进行转换的映射方案。网址<：http://www.ietf.org/rfc/rfc3987.txt

5.1.2.4 HTTP

HTTP 1.1版本定义了一个应用程序协议，用于操作通常以超媒体格式表示的信息资源。 虽然它是一个应用级协议，但通常不与应用程序绑定，由此产生了重要的体系结构优势。大多数人认为 HTTP 和超文本标记语言文（HTML）就是“Web”，但是 HTTP 在非面向文档的系统开发中也很有用。网址：http://www.ietf.org/rfc/rfc2616.txt

5.1.2.5 PATCH 格式

JavaScript 对象表示法（JSON）Patch 网址：https://www.ietf.org/rfc/rfc6902.txt
XML Patch 网址：https://www.ietf.org/rfc/rfc7351.txt

5.2 描述语言

人们对使用各种语言来描述 API 非常感兴趣，通过描述语言可以更容易地编写客户端和服务器文档，甚至生成骨架代码。 一些比较流行、有趣的描述语言包括：

5.2.1 RAML

RAML 是一种 YAML/JSON 语言，可以定义2级成熟度的 API。 它支持可重用模式和特性，通过模式和特性实现功能 API 设计的标准化。网址：http://raml.org

5.2.2 Swagger

Swagger 是另一种 YAML/JSON 语言，支持定义2级成熟度的 API。 它包含代码生成器、编辑器、API 文档可视化功能，能够与其他服务集成的。网址：http://swagger.io

5.2.3 Apiary.io

Apiary.io 是一个协作式的托管站点。它支持 Markdown 格式的 API 文档，可以围绕设计过程进行社交，并且支持模拟数据的托管实现，以便于在 API 实现之前对其进行测试。网址：http://apiary.io

5.2.4 Hydra-Cg

Hydra-Cg 是一种超媒体描述语言，通过像 JSON-LD 这样的标准方便地实现数据关联和并其它数据源的交互。 网址：http://www.hydra-cg.com

5.3 实现

有一些用于构建、生成和使用 RESTful 系统的库和框架。 虽然任何 Web 服务器都可以配置成提供 REST API，但有了这些框架、库和环境可以让过程变得更容易。

以下概述了一些主流的环境：

5.3.1 JAX-RS

JAX-RS 规范为 JEE 环境增加了对 REST 的支持。网址：https://jax-rs-spec.java.net

5.3.2 Restlet

Restlet API 是构建用于生产和消费 RESTful 系统的 Java API 先行者之一。它专注于为客户端和服务器生成一些非常干净、强大的 API。

Restlet Studio 是一个免费工具，能够在 RAML 和基于 swagger 的 API 描述之间进行转换，支持 Restlet、Node 和 JAX-RS 服务器和客户端的骨架和 Stub 代码。网址：http://restlet.org

5.3.3 NetKernel

Netkernel 是一个比较有趣的 RESTful 系统。它基于微内核，是支持各种架构风格环境的代表。Netkernel 受益于在软件体系结构中采用 Web 的经济属性。你可以把它想象成“在内部引入 REST”。 虽然任何基于 REST 的系统在外面看起来都一样，但在运行环境内部 NetKernel 看起来也一样。网址：http://netkernel.org

5.3.4 Play

两个主要的 Scala REST 框架之一。网址：https://www.playframework.com

5.3.5 Spray

两个主要的 Scala REST 框架之一。它设计成配合 Akka actor 模型一起工作。网址：http://spray.io

5.3.6 Express

两个主要的 Node.js REST 框架之一。网址：http://expressjs.com

5.3.7 hapi

两个主要的 Node.js REST 框架之一。网址：http://hapijs.com

5.3.8 Sinatra

Sinatra 是一个领域特定语言（DSL），用来在 Ruby 中创建 RESTful 应用程序。网址：http://www.sinatrarb.com

5.4 客户端

通过浏览器调用 REST API 是可行的，但是还有其它客户端可用于测试和构建面向资源的系统。

5.4.1 curl

curl 是流行的库和命令行工具之一，支持在各种资源上调用各种协议。网址：https://curl.haxx.se

5.4.2 httpie

httpie 是一个非常灵活和易用的客户端，支持通过 HTTP 与资源进行交互。网址：https://httpie.org

5.4.3 Postman

健全的 API 测试需要能够捕获和重播请求，支持各种身份验证和授权方案等功能。 以前的命令行工具允许这样做，但 Postman 是一个较新的桌面应用程序，让这些工作对于开发团队来说变得更容易。网址：https://www.getpostman.com

6. 书籍

- “RESTful Web APIs”：Leonard Richardson、Mike Amundsen 和 Sam Ruby，2013，O’Reilly 出版社
- “RESTful Web Services Cookbook”：Subbu Allamaraju，2010，O’Reilly 出版社
- “REST in Practice”： Jim Webber、Savas Parastatidis 和 Ian Robinson，2010，O’Reilly 出版社。 中文版《REST 实战(中文版)》
- “Restlet in Action” by Jerome Louvel and Thierry Boileau，2011，Manning 出版社
- “Resource-Oriented Architecture Patterns for Webs of Data (Synthesis Lectures on the Semantic Web: Theory and Technology)”：Brian Sletten，2013，Morgan & Claypool

---

-END-

推荐阅读

1. 淘宝为什么能抗住双 11 ？
2. 为什么 ?阿里规定超过 3 张表禁止 join
3. 理解 IntelliJ IDEA 的项目配置和 Web 部署
4. Java 开发中常用的 4 种加密方法
5. 团队开发中 Git 最佳实践

学Java，请关注公众号：Java后端

喜欢文章，点个在看

前后端分离开发，RESTful 接口如何设计

唐尤华 Java后端 2019-11-23

本文译者：唐尤华

本文编辑：江南一点雨（id：a_javaboy）

本文地址：https://urlify.cn/fqmyqa

REST（Representational State Transfer）架构风格是一种世界观，把信息提升为架构中的一等公民。通过 REST 可以实现系统的高性能、可伸缩、通用性、简单性、可修改性和可扩展等特性。这篇文章解释了主要的 HTTP 操作，对 HTTP 响应码进行描述，并列举相关开发库和框架。此外，本文还提供了额外的资源，对每个主题进行了更深入的探讨。

1. 简介

REST 架构风格不是一种可以购买的技术，也不是一个可以添加到软件开发项目中的开发库。首先也是最重要的，REST 是一种世界观，把将信息提升为构建架构中的一等公民。

Roy Fielding 的博士论文“架构风格和基于网络的软件架构设计”介绍和整理了“RESTful”系统的思想和相关术语。这是一篇学术论文，虽然使用正式语言，但是仍然易于理解并且提供了实践基础。

总结一下，RESTful 通过体系结构的特定选择能从部署的系统中获得理想特性。尽管这种风格定义的约束细节并没有为所有场合设计，但是的确可以广泛适用。

由于 Web 对消费者偏好有多重影响，REST 风格的倡导者鼓励企业组织在其边界内使用相同原则，就像他们在面向外部客户的网页上做的那样。本文将讨论现代 REST Web 实现中的基本约束和属性。

1.1 基础概念

REST 表示什么含义？以无状态方式传输、访问和操作文本数据。当正确部署后，REST 为互联网上不同应用程序之间提供了一致的互操作性。无状态（stateless）这个术语至关重要，它使得应用程序可以用不可知的方式进行通信。
RESTful API 通过统一资源定位符地址（URL）公开服务。URL 名称将资源的区分为接受内容或返回内容。RFC 1738 中定义了 URL scheme，可以在这里找到: https://tools.ietf.org/rfc/rfc1738.txt

RESTful URL 类似于下面这个 library API：

- http://fakelibrary.org/library

实际公开的不一定是某种任意的服务，而是代表对消费者有价值的信息资源。URL 作为资源句柄，可以请求、更新或删除内容。

开始把服务发布到某个地方，然后开始与 REST 服务进行交互。返回的内容可能是 XML、JSON 格式，或者更确切地说是像 Atom 或自定义 MIME 类型等超媒体格式。虽然一般建议尽可能重用现有的格式，但是对正确设计的媒体类型正在变得越来越宽容。

需要请求资源的时候，客户机会发一个超文本传输协议（HTTP）GET 请求，例如在浏览器中键入一个 URL 然后点击回车，选择书签，或者点击锚引用链接。

通过编程方式与 RESTful API 交互，有数十个客户端 API 或工具可供选择。使用 curl 命令行工具，可以输入以下命令：

```
curl http://fakelibrary.org/library
```

上面的命令使用默认格式，但你可能不需要这种格式的信息。幸运的是 HTTP 有一种机制，可以指定返回信息的格式。在请求中指定 "Accept" 头，如果服务器支持这种格式，会以指定的格式返回。这个过程称为内容协商，这是 HTTP 中未被充分利用的功能之一，可以使用一个类似于上面例子中的 curl 命令来指定：

```
curl –H "Accept:application/json" http://fakelibrary.org/library
```

由于资源名称与内容格式是独立的，从而让请求不同格式信息成为可能。虽然 REST 中的 “R” 的含义是 “表现”而非“资源”，但是应该在构建系统时允许客户端指定请求的内容格式，请牢记这一点。在我们的例子中 library API 可能包含以下 URL：

- http://fakelibrary.org/library：图书馆基本信息，搜索图书、DVD等相关资源基本功能的链接。
- http://fakelibrary.org/book：存放书籍的“信息空间”。从概念上说，这里可能会存放所有的书籍。显然，如果这个问题得到解决，我们不会希望返回所有图书，而是希望通过类别、搜索关键词等来检索图书。
- http://fakelibrary.org/book/category/1234：在书籍的信息空间里，我们可以指定类别浏览，例如成人小说、儿童书籍、园艺书籍等。使用杜威十进制图书分类法是可行的，但我们也可以想象自定义分组。问题的关键在于，这种“信息空间”可能是无限的，而且可能收到人们实际关心的信息类型影响。
- http://fakelibrary.org/book/isbn/978-0596801687：提到某本具体的书，应该包括书名、作者、出版商、系统中的拷贝数、可用拷贝数等信息。

> 译注：杜威十进制图书分类法由美国图书馆专家麦尔威·杜威发明，于1876年首次发表，历经22次的大改版。该分类法以三位数字代表分类码，共可分为10个大分类、100个中分类及1000个小分类。

就图书馆用户而言，上面提到的这些 URL 可能就是只读的，但是图书馆员使用应用程序时实际上可以操作这些资源。

例如添加一本新书，可以向 main/book 地址 POST 一个 XML。使用 curl 提交，看起来可能像这样：

```
curl –u username:password -d @book.xml -H "Content-type: text/xml" http://fakelibrary.org/book
```

此时，服务器可能会对提交的内容进行校验，创建与图书相关的记录，并返回响应代码201——表示已创建新资源。新资源的 URL 可以在响应的 Location 头中找到。

RESTful 请求一个重要特性：每次请求都包含了充足的状态信息来响应请求。这为服务器的可见性和无状态创造了条件，并为扩展系统和识别发送的请求内容提供了理想特性。对于缓存结果也非常有帮助。服务器地址和请求状态组合成可计算的 hash 键值，并形成一个结果集：

```
http://fakelibrary.org + /book/isbn/978-0596801687
```

接下来我们会先介绍 GET 请求。客户端在需要时发出 GET 请求获取指定资源。客户端可以在本地缓存请求结果，服务器可以在远程缓存结果，系统的中间层可以在请求链路中间缓存结果。这是一个与具体应用程序无关的特性，可以加入系统设计中。

正因为可以操作资源，也就意味着并不是每个人都可以这样做。我们完全可以建立一个防护模型，要求用户在操作前验证身份，证明他们具有该操作的授权。在本文的最后，将提供一些提升 RESTful 服务安全性的内容。

2. REST Vs SOAP

> SOAP：简单对象访问协议（Simple Object Access Protocol）。是交换数据的一种协议规范，是一种轻量的、简单的、基于XML的协议。一条 SOAP 消息就是一个普通的 XML 文档，包含必需的 Envelope 元素、可选的 Header 元素、必需的 Body 元素和可选的 Fault 元素。

把 REST 与 SOAP 划等号是错误的。在这两者之间进行比较，带来的困扰远多于好处。简单来说，它们不是一回事。尽管可以用这两种方法解决许多架构问题，但是它们不能相互替换。

这种混淆很大程度上源于对 “REST 是通过 URL 调用 Web 服务”这句话的误解。这种观点与 RESTful 架构的功能相距甚远。如果不全面深入理解 RESTful 的架构实现，就很容易误解 REST 实践的本意。

利用 REST 的最佳方式，是将生产和消费过程中的信息与技术分离实现解耦，进而更好地管理系统，让架构具备以下特性：

- 高性能
- 可扩展
- 通用
- 简洁
- 可修改

这并不是说，基于 SOAP 构建的系统不能具备上述特性。而是当技术、组织或过程的复杂性造成不能在单个事务中完成请求的生命周期时，这种情况 SOAP 能够发挥最佳效果。

3. Richardson 成熟度模型

Leonard Richardson 引入了一种成熟度模型，部分阐述了 SOAP 与 REST 之间的区别，并提供一种对不同类型的系统进行分类的框架。许多人不恰当地称之为 “REST”。可以将这种分类看作系统中不同 Web 技术组件紧密程度的度量标准：包括信息资源、HTTP 作为应用层协议和作超媒体作为控制媒介。

等级	采纳
0	基本上就是 SOAP。没有信息资源，HTTP 被用作传输协议，也没有超媒体的概念。 结论，REST 和 SOAP 是两种不同的方案
1	用到 URL 但并不总是作为信息资源使用，所有内容都通过 GET 请求，包括更新服务器状态的请求。大多数人刚接触 REST 时构建的系统通常是这样的
2	使用 URL 表示信息资源。HTTP 作为应用层协议，有时也用作内容协商。大多数面向 Internet 的 “REST” Web 服务实际上位于这个级别，因为它们只支持非超媒体格式
3	使用 URL 表示信息资源。HTTP 作为应用层协议，也用来进行内容协商。使用超媒体进行客户端的交互

称其为“成熟度模型”似乎意味着应该只构建“成熟度”最高的系统。这种看法是不合适的。第 2 级是有价值的，从 2 级向 3 级转变通常只是采用了一种新的 MIME 类型。然而，从 0 级到 3 级的转变要困难得多，因此增量式升级转变通常也会增值。

首先，确定希望公开哪些信息资源。采用 HTTP 作为处理这些信息资源的应用协议，包括内容协商。接下来，当一切就绪时，使用基于超媒体的 MIME 类型，这样就可以充分享受 REST 的好处了。

4. 动词

动词是用来与服务器资源交互的方法或操作。RESTful 系统中有限的动词让刚接触该的使用者感到困惑和沮丧。看似武断和不必要的约束，目的是鼓励以应用程序无关的形式提供可预测的行为。通过明确、清晰地定义这些动词的行为，客

户端可以在网络中断或故障时自主处理。

精心设计的 RESTful 系统主要使用 4 个 HTTP 动词。

4.1 GET

GET 请求是最常用的 Web 动词。GET 请求将命名资源从服务器传输到客户端。尽管客户端不需要知道请求的资源内容，但是请求返回的结果是带元数据标记的字节流，这表明客户端应该知道如何解释资源。在 Web 中通常用“text/html” 或 “application/xhtml+xml” 表示。正如之前提到的那样，只要服务器支持，客户端可以通过内容协商提前指定请求的返回格式。

GET 请求关键点之一，不要修改服务器端的任何内容。这是一个基本的安全要求，也是不熟悉 REST 的开发者犯的最大错误之一。你可能会遇到这样的 URL：

`http://example.com/res/action=update?data=1234`

不要这样做！ 由于 GET 请求安全性允许缓存请求，这会让正在构建的 RESTful 系统陷入混乱。GET 请求也意味着幂等性，即多次请求不会对系统产生任何影响。这是基于分布式基础设施的一个重要特性。如果进行 GET 请求时被打断，由于幂等性，客户端可以再次发起请求。这点非常重要。在设计良好的基础结构中，客户端可以从任意应用程序发起请求。虽然一定会有与应用程序相关的特定行为，但是加入与应用程序无关的行为越多，系统就会越有弹性，也更容易维护。

4.2 POST

在辨别 POST 和 PUT 动词意图的时候，情况开始变得不那么清晰。根据定义，二者似乎都可以被客户端用来创建或更新服务器资源，然而它们的用途各有不同。

当无法预测请求创建的资源的标识时，客户端会使用 POST 请求。在新增雇员、下订单或提交表单的时候，我们无法预测服务器将如何命名正在创建的资源。这就是为什么将资源提交给类似 Servlet 这样的程序处理。接下来，服务器会接受请求、校验请求、验证用户凭据等。成功处理后，服务器将返回 201 HTTP 响应代码，其中包含一个 “Location” 头，代表新创建的资源的位置。

注意： 有些人将 POST 视为创建资源的 GET 会话。他们会对创建的资源通过 body 返回200，而不是返回 201。这似乎是避免二次请求的一种快捷方式，但是这种做法混合了 POST 和 GET，让缓存资源的潜在影响变得微妙。尽量避免因为走捷径而牺牲大局。短期看这似乎是值得的，但随着时间的推移，这些捷径叠加起来可能会带来不利的影响。

POST 动词的另一个主要用途是“追加（Append）”资源信息，即增量编辑或部分更新，而不是提交完整的资源。这里应使用 PUT 操作。对已知资源使用 POST 更新，可用于向订单添加新送货地址或更新购物车中某个商品的数量。

由于是更新资源的部分信息，POST 既不安全也不幂等。

POST 的最后一种常见用法是提交查询。将查询的内容或表单内容进行 URL 编码后提交给服务执行查询。通常可以直接返回 POST 结果，因为没有与查询相关的标识。

注意： 建议将这样的查询转换为信息资源本身。如果采用 POST 查询，可以考虑采用 GET 请求，后者支持缓存。你可以与其他人分享这个链接。

4.3 PUT

由于 HTML 表单目前还不支持 PUT，许多开发人员基本上会忽略 PUT 动词。然而，PUT 有一个重要作用并且是 RESTful 系统完整愿景的一部分。

客户端可以向指定 URL 发 PUT 请求，服务器用请求中的数据执行覆盖操作。PUT 请求在某种程度上是等幂的，而 POST 更新不是。

如果客户端在 PUT 覆盖请求时被打断，由于重新发送覆盖操不会造成任何后果，因此可以再次发送。客户端具备管理状态能力，所以直接重发覆盖命令即可。

注意： 这种协议层处理并不意味着要取消更高级别（如应用层）的事务，但是同样地，它也是一种体系结构上理想的属性，可以在应用层以下使用。

如果客户端能够提前了解资源的标识，那么 PUT 也可用于创建资源。正如我们在 POST 部分中讨论的那样，通常不会出现这种情况。但是如果客户端能够控制服务器端信息空间，那么这种操作也是合理的。

4.4 DELETE

在公共网络上 DELETE 动词没有被广泛使用（谢天谢地!）。然而，对于控制信息空间非常有用，它是资源生命周期中非常有用的一部分。

DELETE 请求意在实现等幂。可能由于网络故障 DELETE 请求被打断，这时我们希望客户端继续尝试。第一次请求无论成功与否，资源都应该返回204（无指定内容）。对之前已删除的资源或不存在的资源可能需要一些额外处理，两种情况都应该返回404。一些安全策略要求为不存在的和已删除的资源返回404，这样 DELETE 请求就不会泄漏有关资源是否存在的信息。

还有另外三个没有广泛使用但是有价值的动词。

4.5 HEAD

HEAD 动词用来请求资源，但不实际检索。客户端可以通过 HEAD 检查资源是否存在，并检查资源相关的元数据。

4.6 OPTIONS

OPTIONS 动词也可以用来查询服务器相关资源的情况，方法是询问哪些其它动词可用于该资源。

4.7 PATCH

最新的动词 PATCH 直到 2010 年才正式采纳为 HTTP 的一部分。旨在提供一种标准化方式来表示部分更新。PATCH 请求通过标准格式让交互的意图更明确。这是推荐使用 PATCH 而非 POST 的原因，尽管 POST 可以用于任何事情。IETF 发布了 RFC 文档，定义用于 PATCH 操作的 XML 和 JSON。

如果客户端 PATCH 请求的 header 中带 If-Match，则此部分为幂等更新。可以重试中断的请求，因为如果第一次请求成功，那么 If-Match header 会不同于新状态。如果相同，则未处理原始请求可应用 PATCH。

5. 响应码

HTTP 响应码为我们在客户端和服务器之间的对话提供了丰富的请求状态信息。大多数人只熟悉一般意义上的200、403、404或者500，但是还有更多有用的代码可供使用。这里表格并不全面，但是它们涵盖了许多在 RESTful 环境中应该考虑使用的最重要代码。数字可按照以下类别分组：

- 1XX：信息类
- 2XX：操作成功
- 3XX：重定向
- 4XX：客户端错误
- 5XX：服务器错误

第一组响应码表明客户端的请求格式正确且处理成功。具体操作如下表所示：

响应代码	描述
200	OK。请求已成功执行，回应内容取决于所调用的动词。
201	已创建。请求已成功执行，在执行过程中创建了一个新资源。响应 body 为空或包含所创建资源的 URI。响应中的 Location 头也应该指向 URI
202	已接受。请求有效并已接受，但尚未得到处理。可通过请求轮询，响应结果中 URI 提供状态更新。这种方式支持异步 REST 请求
204	没有请求的内容。请求已成功处理，但服务器没有任何响应结果，客户端不应更新显示

表1 成功的客户端请求

响应代码	描述
301	永久移除。请求的资源不再位于指定的 URL，新的 Location 应该在响应头中返回。只有 GET 或 HEAD 请求应当重定向到新位置。如果可能，客户端应该更新自己保存的书签
302	已找到。请求的资源已在其它临时位置找到，并应该在响应头中返回该位置。只有 GET 或 HEAD 请求应该重定向到新位置。客户端无需更新书签，因为资源会返回对应的 URL。
303	参见其他信息。该响应码被 W3C 技术架构小组（TAG）重新解释成一种对非网络可寻址资源的有效请求方式。这是语义化 Web 中的一个重要概念，可以向个人、概念、组织等提供 URI。能够在 Web 中找到和不能在 Web 中找到的资源是有区别的。如果客户端收到的响应码是303而不是200，就能分辨这种差别。重定向的位置在响应的 Location 头中。头信息可能包含相关资源的文档引用，也可能包含相关资源的一些元数据

表2 — 客户端重定向请求

表 3 中的响应代码表示客户端请求无效，如果条件不发生变化，重新请求仍无法处理。这些故障可能有请求格式错误、未授权的请求、请求的资源不存在等。

响应代码	描述
405	方法不允许
406	请求不接受
410	资源不存在
411	需要指定长度
412	前提条件失败
413	Entity 太大
414	URI 超长
415	不支持的媒体类型
417	预期失败

表3 客户端请求错误

最后，表4中的响应代码表示服务器暂时无法处理客户端请求（可能仍然无效）。客户端应当在将来的某个时候重新请

求。

响应代码	描述
500	内部服务器错误
501	未实现
503	服务不可用

表4 服务器处理请求错误

服务根据其自身功能要求具有不同程度的可扩展性。

注意：试试响应代码 418，它会返回简洁有力的回复："我是一个茶壶。"

5.1 REST 资源

5.1.1 论文

- Fielding 博士的论文《架构的风格与基于网络的软件架构设计》是对 RESTful 思想的主要介绍：http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

5.1.2 RFC 规范

REST 常见用法的技术规范由**国际互联网工程任务组（IETF）定义，按照请求评议（RFC）**流程完善。规范由数字定义，并随着时间推移不时更新版本，以替换已经过时的文件。目前，这里有最新的相关 RFC 文件。

5.1.2.1 URI

RFC 3986 定义了 URI 命名方案的通用语法。URI 是一种命名方案，包含了对其他如网址、支持名字子空间等编码方案。网址：http://www.ietf.org/rfc/rfc3986.txt

5.1.2.2 URL

Url 是 URI 的一种形式，其中嵌入了充足的信息（通常是访问方案和地址），用于解析和定位资源统一资源定位符。网址：http://www.ietf.org/rfc/rfc1738.txt

5.1.2.3 IRI

国际化资源标识符（IRI）在概念上是一个用 Unicode 编码的 URI，用于在 Web 上使用的标识符中支持世界上各种语言

的字符。IETF 选择创建一个新的标准，而不是改变 URI 方案本身，以避免破坏现有的系统并明确区分这两种方法。那些支持 IRI 的人故意这样做。还定义了在 IRI 和 URI 之间进行转换的映射方案。网址：http://www.ietf.org/rfc/rfc3987.txt

5.1.2.4 HTTP

HTTP 1.1 版本定义了一个应用程序协议，用于操作通常以超媒体格式表示的信息资源。虽然它是一个应用级协议，但通常不与应用程序绑定，由此产生了重要的体系结构优势。大多数人认为 HTTP 和超文本标记语言文（HTML）就是“Web”，但是 HTTP 在非面向文档的系统开发中也很有用。网址：http://www.ietf.org/rfc/rfc2616.txt

5.1.2.5 PATCH 格式

JavaScript 对象表示法（JSON）Patch 网址：https://www.ietf.org/rfc/rfc6902.txt

XML Patch 网址：https://www.ietf.org/rfc/rfc7351.txt

5.2 描述语言

人们对使用各种语言来描述 API 非常感兴趣，通过描述语言可以更容易地编写客户端和服务器文档，甚至生成骨架代码。一些比较流行、有趣的描述语言包括：

5.2.1 RAML

RAML 是一种 YAML/JSON 语言，可以定义 2 级成熟度的 API。它支持可重用模式和特性，通过模式和特性实现功能 API 设计的标准化。网址：http://raml.org

5.2.2 Swagger

Swagger 是另一种 YAML/JSON 语言，支持定义2级成熟度的 API。它包含代码生成器、编辑器、 API 文档可视化功能，能够与其他服务集成的。网址：http://swagger.io

5.2.3 Apiary.io

Apiary.io 是一个协作式的托管站点。它支持 Markdown 格式的 API 文档，可以围绕设计过程进行社交，并且支持模拟数据的托管实现，以便于在 API 实现之前对其进行测试。网址：http://apiary.io

5.2.4 Hydra-Cg

Hydra-Cg 是一种超媒体描述语言，通过像 JSON-LD 这样的标准方便地实现数据关联和并其它数据源的交互。网址：http://www.hydra-cg.com

5.3 实现

有一些用于构建、生成和使用 RESTful 系统的库和框架。虽然任何 **Web** 服务器都可以配置成提供 REST API，但有了这些框架、库和环境可以让过程变得更容易。

以下概述了一些主流的环境：

5.3.1 JAX-RS

JAX-RS 规范为 JEE 环境增加了对 REST 的支持。网址：https://jax-rs-spec.java.net

5.3.2 Restlet

Restlet API 是构建用于生产和消费 RESTful 系统的 Java API 先行者之一。它专注于为客户端和服务器生成一些非常干净、强大的 API。

Restlet Studio 是一个免费工具，能够在 RAML 和基于 swagger 的 API 描述之间进行转换，支持 Restlet、 Node 和 JAX-RS 服务器和客户端的骨架和 Stub 代码。网址：http://restlet.org

5.3.3 NetKernel

Netkernel 是一个比较有趣的 RESTful 系统。它基于微内核，是支持各种架构风格环境的代表。Netkernel 受益于在软件体系结构中采用 Web 的经济属性。你可以把它想象成“在内部引入 REST”。虽然任何基于 REST 的系统在外面看起来都一样，但在运行环境内部 NetKernel 看起来也一样。网址：http://netkernel.org

5.3.4 Play

两个主要的 Scala REST 框架之一。网址：https://www.playframework.com

5.3.5 Spray

两个主要的 Scala REST 框架之一。它设计成配合 Akka actor 模型一起工作。网址：http://spray.io

5.3.6 Express

两个主要的 Node.js REST 框架之一。网址：http://expressjs.com

5.3.7 hapi

两个主要的 Node.js REST 框架之一。网址：http://hapijs.com

5.3.8 Sinatra

Sinatra 是一个领域特定语言（DSL），用来在 Ruby 中创建 RESTful 应用程序。网址：http://www.sinatrarb.com

5.4 客户端

通过浏览器调用 REST API 是可行的，但是还有其它客户端可用于测试和构建面向资源的系统。

5.4.1 curl

curl 是流行的库和命令行工具之一，支持在各种资源上调用各种协议。网址：https://curl.haxx.se

5.4.2 httpie

httpie 是一个非常灵活和易用的客户端，支持通过 HTTP 与资源进行交互。网址：https://httpie.org

5.4.3 Postman

健全的 API 测试需要能够捕获和重播请求，支持各种身份验证和授权方案等功能。以前的命令行工具允许这样做，但 Postman 是一个较新的桌面应用程序，让这些工作对于开发团队来说变得更容易。网址：https://www.getpostman.com

---

【END】

4. MySQL:数据库优化,可以看看这篇文章

5. 团队开发中 Git 最佳实践

喜欢文章,点个在看

前后端分离开发，RESTful 接口应该这样设计

唐尤华 Java后端 2月16日

译者 | 唐尤华

https://dzone.com/refcardz/rest-foundations-restful

前言

REST（Representational State Transfer）架构风格是一种世界观，把信息提升为架构中的一等公民。通过 REST 可以实现系统的高性能、可伸缩、通用性、简单性、可修改性和可扩展等特性。这篇文章解释了主要的 HTTP 操作，对 HTTP 响应码进行描述，并列举相关开发库和框架。此外，本文还提供了额外的资源，对每个主题进行了更深入的探讨。

1. 简介

REST 架构风格不是一种可以购买的技术，也不是一个可以添加到软件开发项目中的开发库。首先也是最重要的，REST 是一种世界观，把将信息提升为构建架构中的一等公民。

Roy Fielding 的博士论文“架构风格和基于网络的软件架构设计”介绍和整理了“RESTful”系统的思想和相关术语。这是一篇学术论文，虽然使用正式语言，但是仍然易于理解并且提供了实践基础。

总结一下，RESTful 通过体系结构的特定选择能从部署的系统中获得理想特性。尽管这种风格定义的约束细节并没有为所有场合设计，但是的确可以广泛适用。

由于 Web 对消费者偏好有多重影响，REST 风格的倡导者鼓励企业组织在其边界内使用相同原则，就像他们在面向外部客户的网页上做的那样。本文将讨论现代 REST Web 实现中的基本约束和属性。

1.1 基础概念

REST 表示什么含义？以无状态方式传输、访问和操作文本数据。当正确部署后，REST 为互联网上不同应用程序之间提供了一致的互操作性。无状态（stateless）这个术语至关重要，它使得应用程序可以用不可知的方式进行通信。RESTful API 通过统一资源定位符地址（URL）公开服务。URL 名称将资源的区分为接受内容或返回内容。RFC 1738 中定义了 URL scheme，可以在这里找到: https://tools.ietf.org/rfc/rfc1738.txt

RESTful URL 类似于下面这个 library API：http://fakelibrary.org/library

实际公开的不一定是某种任意的服务，而是代表对消费者有价值的信息资源。URL 作为资源句柄，可以请求、更新或删除内容。

开始把服务发布到某个地方，然后开始与 REST 服务进行交互。返回的内容可能是 XML、JSON 格式，或者更确切地说是像 Atom 或自定义 MIME 类型等超媒体格式。虽然一般建议尽可能重用现有的格式，但是对正确设计的媒体类型正在变

得越来越宽容。

需要请求资源的时候，客户机会发一个超文本传输协议（HTTP）GET 请求，例如在浏览器中键入一个 URL 然后点击回车，选择书签，或者点击锚引用链接。

通过编程方式与 RESTful API 交互，有数十个客户端 API 或工具可供选择。使用 curl 命令行工具，可以输入以下命令：

```
curl http://fakelibrary.org/library
```

上面的命令使用默认格式，但你可能不需要这种格式的信息。幸运的是 HTTP 有一种机制，可以指定返回信息的格式。在请求中指定 "Accept" 头，如果服务器支持这种格式，会以指定的格式返回。这个过程称为内容协商，这是 HTTP 中未被充分利用的功能之一，可以使用一个类似于上面例子中的 curl 命令来指定：

```
curl –H "Accept:application/json" http://fakelibrary.org/library
```

由于资源名称与内容格式是独立的，从而让请求不同格式信息成为可能。虽然 REST 中的 “R” 的含义是 “表现” 而非 “资源”，但是应该在构建系统时允许客户端指定请求的内容格式，请牢记这一点。在我们的例子中 library API 可能包含以下 URL：

1、http://fakelibrary.org/library：图书馆基本信息，搜索图书、DVD等相关资源基本功能的链接。

2、http://fakelibrary.org/book：存放书籍的“信息空间”。从概念上说，这里可能会存放所有的书籍。显然，如果这个问题得到解决，我们不会希望返回所有图书，而是希望通过类别、搜索关键词等来检索图书。

3、http://fakelibrary.org/book/category/1234：在书籍的信息空间里，我们可以指定类别浏览，例如成人小说、儿童书籍、园艺书籍等。使用杜威十进制图书分类法是可行的，但我们也可以想象自定义分组。问题的关键在于，这种“信息空间”可能是无限的，而且可能收到人们实际关心的信息类型影响。

4、http://fakelibrary.org/book/isbn/978-0596801687：提到某本具体的书，应该包括书名、作者、出版商、系统中的拷贝数、可用拷贝数等信息。

> 译注：杜威十进制图书分类法由美国图书馆专家麦尔威·杜威发明，于1876年首次发表，历经22次的大改版。该分类法以三位数字代表分类码，共可分为10个大分类、100个中分类及1000个小分类。

就图书馆用户而言，上面提到的这些 URL 可能就是只读的，但是图书馆员使用应用程序时实际上可以操作这些资源。

例如添加一本新书，可以向 main/book 地址 POST 一个 XML。使用 curl 提交，看起来可能像这样：

```
curl –u username:password -d @book.xml -H "Content-type: text/xml" http://fakelibrary.org/book
```

此时，服务器可能会对提交的内容进行校验，创建与图书相关的记录，并返回响应代码201——表示已创建新资源。新资源的 URL 可以在响应的 Location 头中找到。

RESTful 请求一个重要特性：每次请求都包含了充足的状态信息来响应请求。这为服务器的可见性和无状态创造了条件，并为扩展系统和识别发送的请求内容提供了理想特性。对于缓存结果也非常有帮助。服务器地址和请求状态组合成可计算的 hash 键值，并形成一个结果集：

```
http://fakelibrary.org + /book/isbn/978-0596801687
```

接下来我们会先介绍 GET 请求。客户端在需要时发出 GET 请求获取指定资源。客户端可以在本地缓存请求结果，服务器可以在远程缓存结果，系统的中间层可以在请求链路中间缓存结果。这是一个与具体应用程序无关的特性，可以加入系统设计中。

正因为可以操作资源，也就意味着并不是每个人都可以这样做。我们完全可以建立一个防护模型，要求用户在操作前验证身份，证明他们具有该操作的授权。在本文的最后，将提供一些提升 RESTful 服务安全性的内容。

2. REST Vs SOAP

> SOAP：简单对象访问协议（Simple Object Access Protocol）。是交换数据的一种协议规范，是一种轻量的、简单的、基于XML的协议。一条 SOAP 消息就是一个普通的 XML 文档，包含必需的 Envelope 元素、可选的 Header 元素、必需的 Body 元素和可选的 Fault 元素。

把 REST 与 SOAP 划等号是错误的。在这两者之间进行比较，带来的困扰远多于好处。简单来说，它们不是一回事。尽管可以用这两种方法解决许多架构问题，但是它们不能相互替换。

这种混淆很大程度上源于对 “REST 是通过 URL 调用 Web 服务”这句话的误解。这种观点与 RESTful 架构的功能相距甚远。如果不全面深入理解 RESTful 的架构实现，就很容易误解 REST 实践的本意。

利用 REST 的最佳方式，是将生产和消费过程中的信息与技术分离实现解耦，进而更好地管理系统，让架构具备以下特性：

1、高性能

2、可扩展

3、通用

4、简洁

5、可修改

这并不是说，基于 SOAP 构建的系统不能具备上述特性。而是当技术、组织或过程的复杂性造成不能在单个事务中完成请求的生命周期时，这种情况 SOAP 能够发挥最佳效果。

3. Richardson 成熟度模型

Leonard Richardson 引入了一种成熟度模型，部分阐述了 SOAP 与 REST 之间的区别，并提供一种对不同类型的系统进行分类的框架。许多人不恰当地称之为 “REST”。可以将这种分类看作系统中不同 Web 技术组件紧密程度的度量标准：包括信息资源、HTTP 作为应用层协议和作超媒体作为控制媒介。

等级	采纳
0	基本上就是 SOAP。没有信息资源，HTTP 被用作传输协议，也没有超媒体的概念。 结论，REST 和 SOAP 是两种不同的方案
1	用到 URL 但并不总是作为信息资源使用，所有内容都通过 GET 请求，包括更新服务器状态的请求。大多数人刚接触 REST 时构建的系统通常是这样的
2	使用 URL 表示信息资源。HTTP 作为应用层协议，有时也用作内容协商。大多数面向 Internet 的 “REST” Web 服务实际上位于这个级别，因为它们只支持非超媒体格式
3	使用 URL 表示信息资源。HTTP 作为应用层协议，也用来进行内容协商。使用超媒体进行客户端的交互

称其为“成熟度模型”似乎意味着应该只构建“成熟度”最高的系统。这种看法是不合适的。第 2 级是有价值的，从 2 级向 3 级转变通常只是采用了一种新的 MIME 类型。然而，从 0 级到 3 级的转变要困难得多，因此增量式升级转变通常也会增值。

首先，确定希望公开哪些信息资源。采用 HTTP 作为处理这些信息资源的应用协议，包括内容协商。接下来，当一切就绪时，使用基于超媒体的 MIME 类型，这样就可以充分享受 REST 的好处了。

4. 动词

动词是用来与服务器资源交互的方法或操作。RESTful 系统中有限的动词让刚接触该的使用者感到困惑和沮丧。看似武断和不必要的约束，目的是鼓励以应用程序无关的形式提供可预测的行为。通过明确、清晰地定义这些动词的行为，客户端可以在网络中断或故障时自主处理。

精心设计的 RESTful 系统主要使用 4 个 HTTP 动词。

4.1 GET

GET 请求是最常用的 Web 动词。GET 请求将命名资源从服务器传输到客户端。尽管客户端不需要知道请求的资源内容，但是请求返回的结果是带元数据标记的字节流，这表明客户端应该知道如何解释资源。在 Web 中通常用 “text/html” 或 “application/xhtml+xml” 表示。正如之前提到的那样，只要服务器支持，客户端可以通过内容协商提前指定请求的返回格式。

GET 请求关键点之一，不要修改服务器端的任何内容。这是一个基本的安全要求，也是不熟悉 REST 的开发者犯的最大错误之一。你可能会遇到这样的 URL：

`http://example.com/res/action=update?data=1234`

不要这样做！ 由于 GET 请求安全性允许缓存请求，这会让正在构建的 RESTful 系统陷入混乱。 GET 请求也意味着幂等性，即多次请求不会对系统产生任何影响。这是基于分布式基础设施的一个重要特性。如果进行 GET 请求时被打断，由于幂等性，客户端可以再次发起请求。这点非常重要。在设计良好的基础结构中，客户端可以从任意应用程序发起请求。虽然一定会有与应用程序相关的特定行为，但是加入与应用程序无关的行为越多，系统就会越有弹性，也更容易维护。

4.2 POST

在辨别 POST 和 PUT 动词意图的时候，情况开始变得不那么清晰。 根据定义，二者似乎都可以被客户端用来创建或更新服务器资源，然而它们的用途各有不同。

当无法预测请求创建的资源的标识时，客户端会使用 POST 请求。在新增雇员、下订单或提交表单的时候，我们无法预测服务器将如何命名正在创建的资源。这就是为什么将资源提交给类似 Servlet 这样的程序处理。接下来，服务器会接受请求、校验请求、验证用户凭据等。成功处理后，服务器将返回 201 HTTP 响应代码，其中包含一个 “Location” 头，代表新创建的资源的位置。

注意： 有些人将 POST 视为创建资源的 GET 会话。 他们会对创建的资源通过 body 返回200，而不是返回 201。这似乎是避免二次请求的一种快捷方式，但是这种做法混合了 POST 和 GET，让缓存资源的潜在影响变得微妙。尽量避免因为走捷径而牺牲大局。短期看这似乎是值得的，但随着时间的推移，这些捷径叠加起来可能会带来不利的影响。

POST 动词的另一个主要用途是“追加（Append）”资源信息，即增量编辑或部分更新，而不是提交完整的资源。这里应使用 PUT 操作。对已知资源使用 POST 更新，可用于向订单添加新送货地址或更新购物车中某个商品的数量。

由于是更新资源的部分信息，POST 既不安全也不幂等。

POST 的最后一种常见用法是提交查询。将查询的内容或表单内容进行 URL 编码后提交给服务执行查询。通常可以直接返回 POST 结果，因为没有与查询相关的标识。

注意： 建议将这样的查询转换为信息资源本身。如果采用 POST 查询，可以考虑采用 GET 请求，后者支持缓存。 你可以与其他人分享这个链接。

4.3 PUT

由于 HTML 表单目前还不支持 PUT，许多开发人员基本上会忽略 PUT 动词。 然而，PUT 有一个重要作用并且是 RESTful 系统完整愿景的一部分。

客户端可以向指定 URL 发 PUT 请求，服务器用请求中的数据执行覆盖操作。 PUT 请求在某种程度上是等幂的，而 POST 更新不是。

如果客户端在 PUT 覆盖请求时被打断，由于重新发送覆盖操不会造成任何后果，因此可以再次发送。客户端具备管理状态能力，所以直接重发覆盖命令即可。

注意： 这种协议层处理并不意味着要取消更高级别（如应用层）的事务，但是同样地，它也是一种体系结构上理想的属性，可以在应用层以下使用。

如果客户端能够提前了解资源的标识，那么 PUT 也可用于创建资源。正如我们在 POST 部分中讨论的那样，通常不会出现这种情况。但是如果客户端能够控制服务器端信息空间，那么这种操作也是合理的。

4.4 DELETE

在公共网络上 DELETE 动词没有被广泛使用（谢天谢地!）。然而，对于控制信息空间非常有用，它是资源生命周期中非常有用的一部分。

DELETE 请求意在实现等幂。可能由于网络故障 DELETE 请求被打断，这时我们希望客户端继续尝试。第一次请求无论成功与否，资源都应该返回204（无指定内容）。对之前已删除的资源或不存在的资源可能需要一些额外处理，两种情况都应该返回404。一些安全策略要求为不存在的和已删除的资源返回404，这样 DELETE 请求就不会泄漏有关资源是否存在的信息。

还有另外三个没有广泛使用但是有价值的动词。

4.5 HEAD

HEAD 动词用来请求资源，但不实际检索。客户端可以通过 HEAD 检查资源是否存在，并检查 资源相关的元数据。

4.6 OPTIONS

OPTIONS 动词也可以用来查询服务器相关资源的情况，方法是询问哪些其它动词可用于该资源。

4.7 PATCH

最新的动词 PATCH 直到 2010 年才正式采纳为 HTTP 的一部分。 旨在提供一种标准化方式来表示部分更新。PATCH 请求通过标准格式让交互的意图更明确。这是推荐使用 PATCH 而非 POST 的原因，尽管 POST 可以用于任何事情。 IETF 发布了 RFC 文档，定义用于 PATCH 操作的 XML 和 JSON。

如果客户端 PATCH 请求的 header 中带 If-Match，则此部分为幂等更新。 可以重试中断的请求，因为如果第一次请求成

功，那么 If-Match header 会不同于新状态。如果相同，则未处理原始请求可应用 PATCH。

5. 响应码

HTTP 响应码为我们在客户端和服务器之间的对话提供了丰富的请求状态信息。大多数人只熟悉一般意义上的200、403、404或者500，但是还有更多有用的代码可供使用。这里表格并不全面，但是它们涵盖了许多在 RESTful 环境中应该考虑使用的最重要代码。数字可按照以下类别分组：

1、1XX：信息类

2、2XX：操作成功

3、3XX：重定向

4、4XX：客户端错误

5、5XX：服务器错误

第一组响应码表明客户端的请求格式正确且处理成功。具体操作如下表所示：

响应代码	描述
200	OK。请求已成功执行，回应内容取决于所调用的动词。
201	已创建。请求已成功执行，在执行过程中创建了一个新资源。响应 body 为空或包含所创建资源的 URI。响应中的 Location 头也应该指向 URI
202	已接受。请求有效并已接受，但尚未得到处理。可通过请求轮询，响应结果中 URI 提供状态更新。这种方式支持异步 REST 请求
204	没有请求的内容。请求已成功处理，但服务器没有任何响应结果，客户端不应更新显示

表1 成功的客户端请求

响应代码	描述
301	永久移除。请求的资源不再位于指定的 URL，新的 Location 应该在响应头中返回。只有 GET 或 HEAD 请求应当重定向到新位置。如果可能，客户端应该更新自己保存的书签
302	已找到。请求的资源已在其它临时位置找到，并应该在响应头中返回该位置。只有 GET 或 HEAD 请求应该重定向到新位置。客户端无需更新书签，因为资源会返回对应的 URL。
303	参见其他信息。该响应码被 W3C 技术架构小组（TAG）重新解释成一种对非网络可寻址资源的有效请求方式。这是语义化 Web 中的一个重要概念，可以向个人、概念、组织等提供 URI。能够在 Web 中找到和不能在 Web 中找到的资源是有区别的。如果客户端收到的响应码是303而不是200，就能分辨这种差别。重定向的位置在响应的 Location 头中。头信息可能包含相关资源的文档引用，也可能包含相关资源的一些元数据

表2 — 客户端重定向请求

表 3 中的响应代码表示客户端请求无效，如果条件不发生变化，重新请求仍无法处理。这些故障可能有请求格式错误、未授权的请求、请求的资源不存在等。

响应代码	描述
405	方法不允许
406	请求不接受
410	资源不存在
411	需要指定长度
412	前提条件失败
413	Entity 太大
414	URI 超长
415	不支持的媒体类型
417	预期失败

表3 客户端请求错误

最后，表4中的响应代码表示服务器暂时无法处理客户端请求（可能仍然无效）。客户端应当在将来的某个时候重新请求。

响应代码	描述
500	内部服务器错误
501	未实现
503	服务不可用

表4 服务器处理请求错误

服务根据其自身功能要求具有不同程度的可扩展性。

注意：试试响应代码 418，它会返回简洁有力的回复："我是一个茶壶。"

5.1 REST 资源

5.1.1 论文

Fielding 博士的论文《架构的风格与基于网络的软件架构设计》是对 RESTful 思想的主要介绍：http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

5.1.2 RFC 规范

REST 常见用法的技术规范由**国际互联网工程任务组（IETF）定义，按照请求评议（RFC）**流程完善。规范由数字定义，并随着时间推移不时更新版本，以替换已经过时的文件。目前，这里有最新的相关 RFC 文件。

5.1.2.1 URI

RFC 3986 定义了 URI 命名方案的通用语法。URI 是一种命名方案，包含了对其他如网址、支持名字子空间等编码方案。网址：http://www.ietf.org/rfc/rfc3986.txt

5.1.2.2 URL

Url 是 URI 的一种形式，其中嵌入了充足的信息（通常是访问方案和地址），用于解析和定位资源统一资源定位符。网址：http://www.ietf.org/rfc/rfc1738.txt

5.1.2.3 IRI

国际化资源标识符（IRI）在概念上是一个用 Unicode 编码的 URI，用于在 Web 上使用的标识符中支持世界上各种语言的字符。IETF 选择创建一个新的标准，而不是改变 URI 方案本身，以避免破坏现有的系统并明确区分这两种方法。那些支持 IRI 的人故意这样做。还定义了在 IRI 和 URI 之间进行转换的映射方案。网址：http://www.ietf.org/rfc/rfc3987.txt

5.1.2.4 HTTP

HTTP 1.1 版本定义了一个应用程序协议，用于操作通常以超媒体格式表示的信息资源。虽然它是一个应用级协议，但通常不与应用程序绑定，由此产生了重要的体系结构优势。大多数人认为 HTTP 和超文本标记语言文（HTML）就是“Web”，但是 HTTP 在非面向文档的系统开发中也很有用。网址：http://www.ietf.org/rfc/rfc2616.txt

5.1.2.5 PATCH 格式

JavaScript 对象表示法（JSON）Patch 网址：https://www.ietf.org/rfc/rfc6902.txt

XML Patch 网址：https://www.ietf.org/rfc/rfc7351.txt

5.2 描述语言

人们对使用各种语言来描述 API 非常感兴趣，通过描述语言可以更容易地编写客户端和服务器文档，甚至生成骨架代码。一些比较流行、有趣的描述语言包括：

5.2.1 RAML

RAML 是一种 YAML/JSON 语言，可以定义 2 级成熟度的 API。 它支持可重用模式和特性，通过模式和特性实现功能 API 设计的标准化。网址：http://raml.org

5.2.2 Swagger

Swagger 是另一种 YAML/JSON 语言，支持定义2级成熟度的 API。它包含代码生成器、编辑器、 API 文档可视化功能，能够与其他服务集成的。网址：http://swagger.io

5.2.3 Apiary.io

Apiary.io 是一个协作式的托管站点。它支持 Markdown 格式的 API 文档，可以围绕设计过程进行社交，并且支持模拟数据的托管实现，以便于在 API 实现之前对其进行测试。网址：http://apiary.io

5.2.4 Hydra-Cg

Hydra-Cg 是一种超媒体描述语言，通过像 JSON-LD 这样的标准方便地实现数据关联和并其它数据源的交互。网址：http://www.hydra-cg.com

5.3 实现

有一些用于构建、生成和使用 RESTful 系统的库和框架。虽然任何 **Web** 服务器都可以配置成提供 REST API，但有了这些框架、库和环境可以让过程变得更容易。

以下概述了一些主流的环境：

5.3.1 JAX-RS

JAX-RS 规范为 JEE 环境增加了对 REST 的支持。 网址：https://jax-rs-spec.java.net

5.3.2 Restlet

Restlet API 是构建用于生产和消费 RESTful 系统的 Java API 先行者之一。 它专注于为客户端和服务器生成一些非常干净、强大的 API。

Restlet Studio 是一个免费工具，能够在 RAML 和基于 swagger 的 API 描述之间进行转换，支持 Restlet、 Node 和 JAX-RS 服务器和客户端的骨架和 Stub 代码。网址：http://restlet.org

5.3.3 NetKernel

Netkernel 是一个比较有趣的 RESTful 系统。 它基于微内核，是支持各种架构风格环境的代表。Netkernel 受益于在软件体系结构中采用 Web 的经济属性。你可以把它想象成“在内部引入 REST”。虽然任何基于 REST 的系统在外面看起来都一样，但在运行环境内部 NetKernel 看起来也一样。网址：http://netkernel.org

5.3.4 Play

两个主要的 Scala REST 框架之一。 网址：https://www.playframework.com

5.3.5 Spray

两个主要的 Scala REST 框架之一。 它设计成配合 Akka actor 模型一起工作。 网址：http://spray.io

5.3.6 Express

两个主要的 Node.js REST 框架之一。 网址：http://expressjs.com

5.3.7 hapi

两个主要的 Node.js REST 框架之一。 网址：http://hapijs.com

5.3.8 Sinatra

Sinatra 是一个领域特定语言（DSL），用来在 Ruby 中创建 RESTful 应用程序， 网址：http://www.sinatrarb.com

5.4 客户端

通过浏览器调用 REST API 是可行的，但是还有其它客户端可用于测试和构建面向资源的系统。

5.4.1 curl

curl 是流行的库和命令行工具之一，支持在各种资源上调用各种协议。网址：https://curl.haxx.se

5.4.2 httpie

httpie 是一个非常灵活和易用的客户端，支持通过 HTTP 与资源进行交互。网址：https://httpie.org

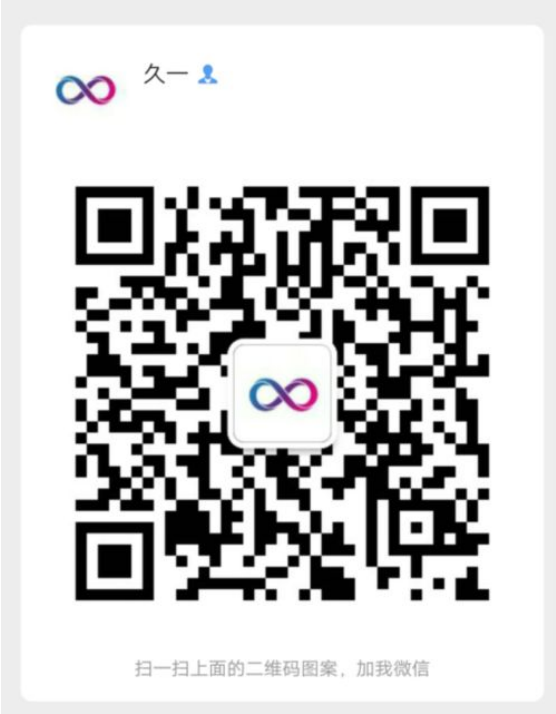
5.4.3 Postman

健全的 API 测试需要能够捕获和重播请求，支持各种身份验证和授权方案等功能。以前的命令行工具允许这样做，但 Postman 是一个较新的桌面应用程序，让这些工作对于开发团队来说变得更容易。网址：https://www.getpostman.com

-END-

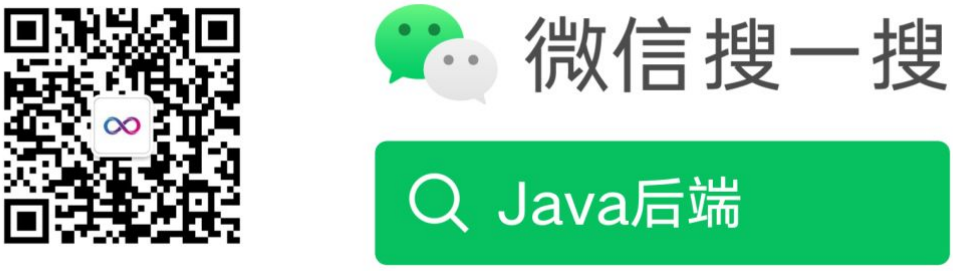
如果看到这里，说明你喜欢这篇文章，请**转发、点赞**。微信搜索「web_resource」关注本订阅号，欢迎添加小编微信「focusoncode」，每日朋友圈更新一篇高质量技术博文（无广告）。

↓扫描二维码添加小编↓



推荐阅读

1. Spring Boot + Vue 如此强大？
2. 编程能力与编程年龄
3. 安利一款 IDEA 中强大的代码生成利器
4. 如何获取靠谱的新型冠状病毒疫情



基于 Spring Boot 的 Restful 风格实现增删改查

Java后端 2019-09-05

作者 | 虚无境

链接 |cnblogs.com/xuwujing/p/8260935.html

前言

在去年的时候，在各种渠道中略微的了解了Spring Boot，在开发web项目的时候是如何的方便、快捷。但是当时并没有认真的去学习下，毕竟感觉自己在Struts和Spring MVC都用得不太熟练。不过在看了很多关于Spring Boot的介绍之后，并没有想象中的那么难，于是开始准备学习Spring Boot。

在闲暇之余的时候，看了下Spring Boot实战以及一些大神关于Spring Boot的博客之后，开始写起了我的第一个Spring Boot的项目。在能够对Spring Boot进行一些简单的开发Restful风格接口实现CRUD功能之后，于是便有了本篇博文。

Spring Boot介绍

Spring Boot是由Pivotal团队提供的全新框架，其设计目的是用来简化新Spring应用的初始搭建以及开发过程。该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置。
简单的来说就是，只需几个jar和一些简单的配置，就可以快速开发项目。
假如我就想简单的开发一个对外的接口，那么只需要以下代码就可以了。

一个主程序启动springBoot

```
@SpringBootApplication
public class Application {
    public static void main(String[] args)
{
        SpringApplication.run(Application.class, args);
    }
}
```

控制层

```
@RestController
public class HelloWorldController {
    @RequestMapping("/hello"
)
    public String index() {
        return "Hello World"
;
    }
 }
```

成功启动主程序之后，编写控制层，然后在浏览器输入 http://localhost:8080//hello 便可以查看信息。

感觉使用SpringBoot开发程序是不是非常的简单呢！

用SpringBoot实战的话来说：

> 这里没有配置，没有web.xml，没有构建说明，甚至没有应用服务器，但这就是整个应用程序了。SpringBoot会搞定执行应用程序所需的各种后勤工作，你只要搞定应用程序的代码就好。

基于SpringBoot开发一个Restful服务

一、开发准备

1.1 数据库和表

首先，我们需要在MySql中创建一个数据库和一张表

数据库的名称为 springboot，表名称为 t_user

脚本如下：

```
CREATE DATABASE `springboot`;

USE `springboot`;

DROP TABLE IF EXISTS `t_user`;

CREATE TABLE `t_user` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT 'id',
  `name` varchar(10) DEFAULT NULL COMMENT '姓名',
  `age` int(2) DEFAULT NULL COMMENT '年龄',
  PRIMARY KEY (`id`
)
) ENGINE=InnoDB AUTO_INCREMENT=12 DEFAULT CHARSET=utf8;
```

1.2 maven相关依赖

因为我们使用Maven创建的，所以需要添加SpringBoot的相关架包。

这里Maven的配置如下：

> springBoot最核心的jar
> spring-boot-starter ：核心模块，包括自动配置支持、日志和YAML；

```
    <dependencies
>
        <dependency
>
            <groupId>org.springframework.boot</groupId
```

```
>
            <artifactId>spring-boot-starter-web</artifactId>
>
        </dependency>
>
        <dependency>
>
            <groupId>org.springframework.boot</groupId>
>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
>
        </dependency>
>
        <dependency>
>
            <groupId>org.springframework.boot</groupId>
>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
>
        </dependency>
>


        <dependency>
>
            <groupId>org.springframework.boot</groupId>
>
            <artifactId>spring-boot-devtools</artifactId>
>
            <optional>true</optional>
>
        </dependency>
>

        <dependency>
>
            <groupId>org.springframework.boot</groupId>
>
            <artifactId>spring-boot-starter-test</artifactId>
>
            <scope>test</scope>
>
        </dependency>
>


          <!-- Spring Boot Mybatis 依赖 -->
        <dependency>
>
            <groupId>org.mybatis.spring.boot</groupId>
>
            <artifactId>mybatis-spring-boot-starter</artifactId>
>
            <version>${mybatis-spring-boot}</version>
```

```
    >
            </dependency
    >


            <!-- MySQL 连接驱动依赖 -->
            <dependency
    >
                <groupId>mysql</groupId
    >
                <artifactId>mysql-connector-java</artifactId
    >
                <version>${mysql-connector}</version
    >
            </dependency
    >


        </dependencies
    >
```

二、工程说明

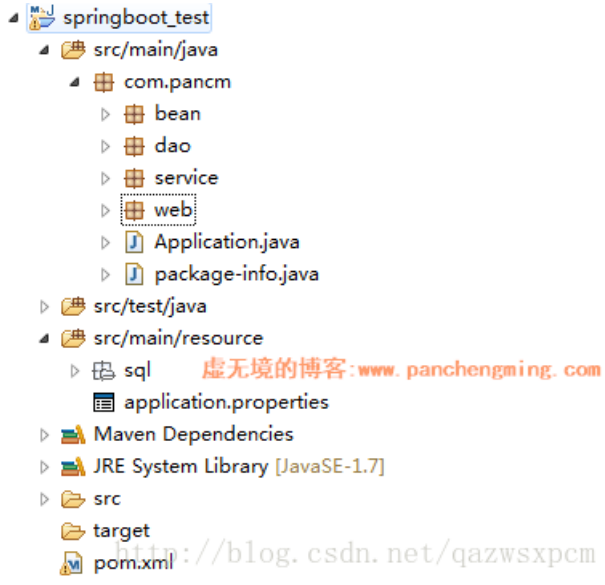
成功创建好数据库以及下载好相应架包之后。

我们来正式开发SpringBoot项目。

2.1工程结构图:

首先确定工程结构,这里我就简单的说明下了。

> com.pancm.web - Controller 层
> com.pancm.dao - 数据操作层 DAO
> com.pancm.bean - 实体类
> com.pancm.service - 业务逻辑层
> Application - 应用启动类
> application.properties - 应用配置文件,应用启动会自动读取配置



2.2 自定义配置文件

一般我们需要一些自定义的配置，例如配置jdbc的连接配置，在这里我们可以用 application.properties 进行配置。数据源实际的配置以各位的为准。

```
### 数据源配置
spring.datasource.url=jdbc:mysql://localhost:3306/springBoot?useUnicode=true&characterEncoding=utf8
spring.datasource.username=root
spring.datasource.password=123456
spring.datasource.driver-class-name=com.mysql.jdbc.Driver


### Mybatis 配置
# 配置为 com.pancm.bean 指向实体类包路径。
mybatis.typeAliasesPackage=com.pancm.bean
# 配置为 classpath 路径下 mapper 包下，* 代表会扫描所有 xml 文件。
mybatis.mapperLocations=classpath\:mapper/*.xml
```

三、代码编写

在创建好相关工程目录之后，我们开始来编写相应的代码。

3.1 实体类编写

由于我们这里只是用于测试，只在数据库中创建了一张t_user表，所以这里我们就只创建一个User实体类，里面的字段对应t_user表的字段。

示例代码如下：

```
 public class User {
     /** 编号 */
     private int id;
     /** 姓名 */
     private String name;
     /** 年龄 */
     private int age;

     public User()
{
     }
   public class User
{
     /** 编号 */
     private int id;
     /** 姓名 */
     private String name;
     /** 年龄 */
     private int age;

```

```
    public User()
{
    }
//   getter和 setter 略
}
```

3.2 Dao层编写

在以前的Dao层这块，hibernate和mybatis 都可以使用注解或者使用mapper配置文件。在这里我们使用spring的JPA来完成基本的增删改查。

说明:

一般有两种方式实现与数据库实现CRUD：

- 第一种是xml的mapper配置。
- 第二种是使用注解，@Insert、@Select、@Update、@Delete 这些来完成。本篇使用的是第二种。

```
@Mapper
public interface UserDao {

    /**
     * 用户数据新增
     */
     @Insert("insert into t_user(id,name,age) values (#{id},#{name},#{age})"
)
      void addUser(User user);

     /**
      * 用户数据修改
      */
     @Update("update t_user set name=#{name},age=#{age} where id=#{id}"
)
      void updateUser(User user);

     /**
      * 用户数据删除
     */
     @Delete("delete from t_user where id=#{id}"
)
     void deleteUser(int id);

      /**
     * 根据用户名称查询用户信息
     *
     */
    @Select("SELECT id,name,age FROM t_user where name=#{userName}"
)
    User findByName(@Param("userName") String userName);

   /**
     * 查询所有
```

```
     */
    @Select("SELECT id,name,age FROM t_user")

    List<User> findAll();


}
```

说明:

- mapper : 在接口上添加了这个注解表示这个接口是基于注解实现的CRUD。
- Results: 返回的map结果集，property 表示User类的字段，column 表示对应数据库的字段。
- Param:sql条件的字段。
- Insert、Select、Update、Delete:对应数据库的增、查、改、删。

3.3 Service 业务逻辑层

这块和hibernate、mybatis的基本一样。

代码如下：

接口

```
import com.pancm.bean.User;/** * * Title: UserService* Description:用户接口 * Version:1.0.0  * @author pancm */public
l();}
```

实现类

```
@Service
public class UserServiceImpl implements UserService {
    @Autowired
    private UserDao userDao;


    @Override
    public boolean addUser(User user)
{
        boolean flag=false
;
        try
{
            userDao.addUser(user);
            flag=true
;
        }catch(Exception e){
            e.printStackTrace();
        }
        return flag
;
    }
```

```
22
23      @Override
24      public boolean updateUser(User user)
25  {
26          boolean flag=false
27  ;
28          try
29  {
30              userDao.updateUser(user);
31              flag=true
32  ;
33          }catch(Exception e){
34              e.printStackTrace();
35          }
36          return flag
37  ;
38      }
39
40      @Override
41      public boolean deleteUser(int id)
42  {
43          boolean flag=false
44  ;
45          try
46  {
47              userDao.deleteUser(id);
48              flag=true
49  ;
50          }catch(Exception e){
51              e.printStackTrace();
52          }
53          return flag
    ;
      }

      @Override
      public User findUserByName(String userName)
  {
          return userDao.findByName(userName);
      }


      @Override
      public List<User> findAll()
  {
          return userDao.findAll();
      }
  }
```

3.4 Controller 控制层

控制层这块和springMVC很像，但是相比而言要简洁不少。

说明：

- RestController：默认类中的方法都会以json的格式返回。
- RequestMapping：接口路径配置。
- method：请求格式。
- RequestParam：请求参数。

具体实现如下：

```
@RestController
@RequestMapping(value = "/api/user"
)
public class UserRestController {
    @Autowired
    private UserService userService;

    @RequestMapping(value = "/user", method = RequestMethod.POST)
    public boolean addUser( User user) {
        System.out.println("开始新增...")
;
        return userService.addUser(user);
    }

    @RequestMapping(value = "/user", method = RequestMethod.PUT)
    public boolean updateUser( User user) {
        System.out.println("开始更新...")
;
        return userService.updateUser(user);
    }

    @RequestMapping(value = "/user", method = RequestMethod.DELETE)
    public boolean delete(@RequestParam(value = "userName", required = true) int userId) {
        System.out.println("开始删除...")
;
        return userService.deleteUser(userId);
    }


    @RequestMapping(value = "/user", method = RequestMethod.GET)
    public User findByUserName(@RequestParam(value = "userName", required = true) String userName) {
        System.out.println("开始查询...")
;
        return userService.findUserByName(userName);
    }


    @RequestMapping(value = "/userAll", method = RequestMethod.GET)
    public List<User> findByUserAge() {
        System.out.println("开始查询所有数据...")
;
        return userService.findAll();
```

```
    }
}
```

3.5 Application 主程序

SpringApplication 则是用于从main方法启动Spring应用的类。
默认，它会执行以下步骤：

1. 创建一个合适的ApplicationContext实例（取决于classpath）。
2. 注册一个CommandLinePropertySource，以便将命令行参数作为Spring properties。
3. 刷新application context，加载所有单例beans。
4. 激活所有CommandLineRunner beans。

直接使用main启动该类，SpringBoot便自动化配置了。

> ps:即使是现在我依旧觉得这个实在是太厉害了。

该类的一些注解说明：

> SpringBootApplication：开启组件扫描和自动配置。
> MapperScan: mapper 接口类扫描包配置

代码如下：

```
@SpringBootApplication
@MapperScan("com.pancm.dao")
public class Application {
    public static void main(String[] args)
{
        // 启动嵌入式的 Tomcat 并初始化 Spring 环境及其各 Spring 组件
        SpringApplication.run(Application.class, args);
        System.out.println("程序正在运行...")
;
    }
}
```

四、代码测试

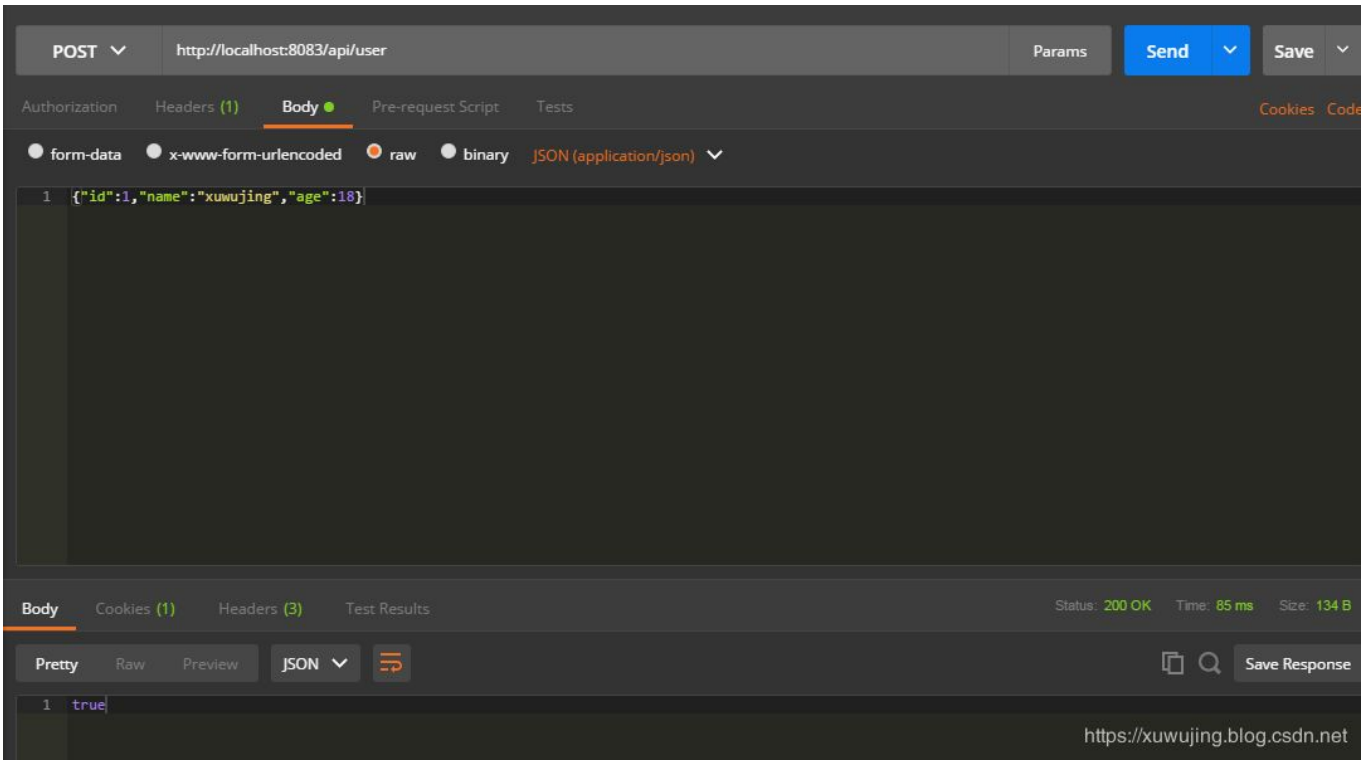
代码编写完之后，我们进行代码的测试。

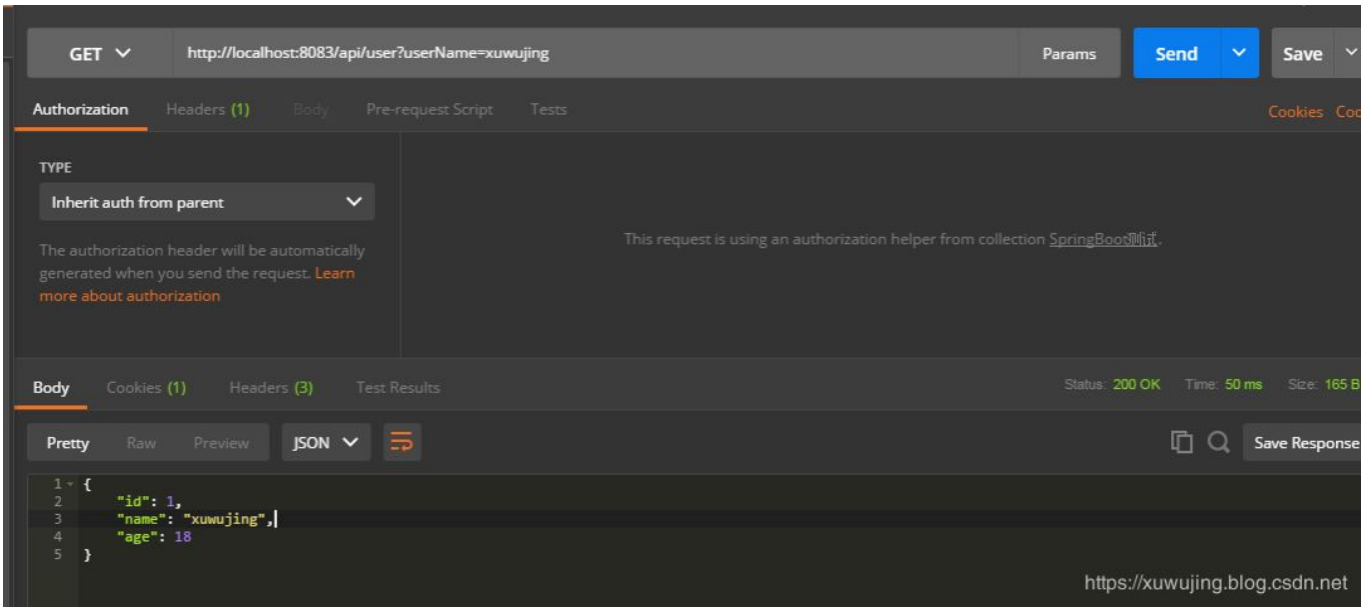
启动Application 之后，使用postman工具进行接口的测试。

postman的使用教程可以看这篇博客：

> http://www.panchengming.com/2017/04/24/pancm12/

测试结果如下:





这里只使用了一个get和post测试,实际方法都测试过了。

项目放到github上面去了:

> https://github.com/xuwujing/springBoot

如果喜欢本篇文章,欢迎转发、点赞。关注订阅号「Web项目聚集地」,回复「进群」即可进入无广告技术交流。

推荐阅读

1. 什么时候进行分库分表？

2. 从 Java 程序员的角度理解加密

3. IDEA 中使用 Git 图文教程

4. 一文梳理 Redis 基础

5. 优化你的 Spring Boot

6. 数据库不使用外键的 9 个理由

Web项目聚集地

微信扫描二维码，关注我的公众号

喜欢文章，点个在看

实战：SpringBoot & Restful API 构建示例

liuxiaopeng Java后端 2019-12-29

作者 | liuxiaopeng

链接 | cnblogs.com/paddix/p/8215245.html

在现在的开发流程中，为了最大程度实现前后端的分离，通常后端接口只提供数据接口，由前端通过Ajax请求从后端获取数据并进行渲染再展示给用户。我们用的最多的方式就是后端会返回给前端一个JSON字符串，前端解析JSON字符串生成JavaScript的对象，然后再做处理。

本文就来演示一下Spring boot如何实现这种模式，本文重点会讲解如何设计一个Restful的API，并通过Spring boot来实现相关的API。

不过，为了大家更好的了解Restful风格的API，我们先设计一个传统的数据返回接口，这样大家可以对比着来理解。

一、非Restful接口的支持

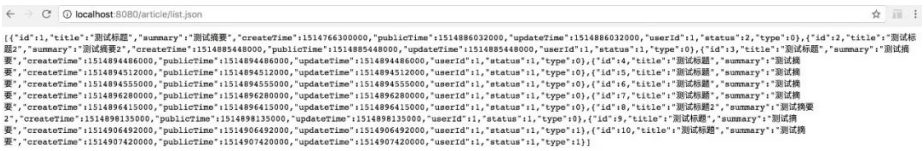
我们这里以文章列表为例，实现一个返回文章列表的接口，代码如下：

```
@Controller
@RequestMapping("/article")
public class ArticleController {

  @Autowired
  private ArticleService articleService;

  @RequestMapping("/list.json")
  @ResponseBody
  public List<Article> listArticles(String title, Integer pageSize, Integer pageNum) {
    if (pageSize == null) {
      pageSize = 10;
    }
    if (pageNum == null) {
      pageNum = 1;
    }
    int offset = (pageNum - 1) * pageSize;
    return articleService.getArticles(title, 1L, offset, pageSize);
  }
}
```

这个ArticleService的实现很简单，就是简单的封装了ArticleMapper的操作，ArticleService的实现类如下：

```
@Service
public class ArticleServiceImpl implements ArticleService {

  @Autowired
  private ArticleMapper articleMapper;

  @Override
  public Long saveArticle(@RequestBody Article article) {
    return articleMapper.insertArticle(article);
  }

  @Override
  public List<Article> getArticles(String title,Long userId,int offset,int pageSize) {
    Article article = new Article();
    article.setTitle(title);
    article.setUserId(userId);
    return articleMapper.queryArticlesByPage(article,offset,pageSize);
  }

  @Override
  public Article getById(Long id) {
    return articleMapper.queryById(id);
  }

  @Override
  public void updateArticle(Article article) {
    article.setUpdateTime(new Date());
    articleMapper.updateArticleById(article);
  }
}
```

运行Application.java这个类，然后访问：http://locahost:8080/article/list.json，就可以看到如下的结果：



ArticleServiceImpl这个类是一个很普通的类，只有一个Spring的注解@Service，标识为一个bean以便于通过Spring IoC容器来管理。我们再来看看ArticleController这个类，其实用过Spring MVC的人应该都熟悉这几个注解，这里简单解释一下：

- @Controller 标识一个类为控制器。
- @RequestMapping URL的映射。
- @ResponseBody 返回结果转换为JSON字符串。
- @RequestBody 表示接收JSON格式字符串参数。

通过这个三个注解，我们就能轻松的实现通过URL给前端返回JSON格式数据的功能。不过大家肯定有点疑惑，这不都是Spring MVC的东西吗？跟Spring boot有什么关系？

其实Spring boot的作用就是为我们省去了配置的过程，其他功能确实都是Spring与Spring MVC来为我们提供的，大家应该记得Spring boot通过各种starter来为我们提供自动配置的服务，我们的工程里面之前引入过这个依赖：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

这个是所有Spring boot的web工程都需要引入的jar包，也就是说只要是Spring boot的web的工程，都默认支持上述的功能。这

里我们进一步发现，通过Spring boot来开发web工程，确实为我们省了许多配置的工作。

二、Restful API设计

好了，我们现在再来看看如何实现Restful API。实际上Restful本身不是一项什么高深的技术，而只是一种编程风格，或者说是一种设计风格。

在传统的http接口设计中，我们一般只使用了get和post两个方法，然后用我们自己定义的词汇来表示不同的操作，比如上面查询文章的接口，我们定义了article/list.json来表示查询文章列表，可以通过get或者post方法来访问。而Restful API的设计则通过HTTP的方法来表示CRUD相关的操作。

因此，除了get和post方法外，还会用到其他的HTTP方法，如PUT、DELETE、HEAD等，通过不同的HTTP方法来表示不同含义的操作。下面是我设计的一组对文章的增删改查的Restful API：

接口URL	HTTP方法	接口说明
/article	POST	保存文章
/article/{id}	GET	查询文章列表
/article/{id}	DELETE	删除文章
/article/{id}	PUT	更新文章信息

这里可以看出，URL仅仅是标识资源的路劲，而具体的行为由HTTP方法来指定。

三、Restful API实现

现在我们再来看看如何实现上面的接口，其他就不多说，直接看代码：

```java
@RestController
@RequestMapping("/rest")
public class ArticleRestController {

  @Autowired
  private ArticleService articleService;

  @RequestMapping(value = "/article", method = POST, produces = "application/json")
  public WebResponse<Map<String, Object>> saveArticle(@RequestBody Article article) {
    article.setUserId(1L);
    articleService.saveArticle(article);
    Map<String, Object> ret = new HashMap<>();
    ret.put("id", article.getId());
    WebResponse<Map<String, Object>> response = WebResponse.getSuccessResponse(ret);
    return response;
  }

  @RequestMapping(value = "/article/{id}", method = DELETE, produces = "application/json")
  public WebResponse<?> deleteArticle(@PathVariable Long id) {
    Article article = articleService.getById(id);
    article.setStatus(-1);
    articleService.updateArticle(article);
    WebResponse<Object> response = WebResponse.getSuccessResponse(null);
    return response;
  }

  @RequestMapping(value = "/article/{id}", method = PUT, produces = "application/json")
  public WebResponse<Object> updateArticle(@PathVariable Long id, @RequestBody Article article) {
    article.setId(id);
    articleService.updateArticle(article);
    WebResponse<Object> response = WebResponse.getSuccessResponse(null);
    return response;
  }

  @RequestMapping(value = "/article/{id}", method = GET, produces = "application/json")
  public WebResponse<Article> getArticle(@PathVariable Long id) {
    Article article = articleService.getById(id);
    WebResponse<Article> response = WebResponse.getSuccessResponse(article);
    return response;
  }
}
```

我们再来分析一下这段代码，这段代码和之前代码的区别在于：

（1）我们使用的是@RestController这个注解，而不是@Controller，不过这个注解同样不是Spring boot提供的，而是Spring MVC4中的提供的注解，表示一个支持Restful的控制器。

（2）这个类中有三个URL映射是相同的，即都是/article/{id}，这在@Controller标识的类中是不允许出现的。这里的可以通过method来进行区分，produces的作用是表示返回结果的类型是JSON。

（3）@PathVariable这个注解，也是Spring MVC提供的，其作用是表示该变量的值是从访问路径中获取。

所以看来看去，这个代码还是跟Spring boot没太多的关系，Spring boot也仅仅是提供自动配置的功能，这也是Spring boot用起来很舒服的一个很重要的原因，因为它的侵入性非常非常小，你基本感觉不到它的存在。

四、测试

代码写完了，怎么测试？除了GET的方法外，都不能直接通过浏览器来访问，当然，我们可以直接通过postman来发送各种http请求。不过我还是比较支持通过单元测试类来测试各个方法。这里我们就通过Junit来测试各个方法：

```java
@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest(classes = Application.class)
public class ArticleControllerTest {

  @Autowired
  private ArticleRestController restController;

  private MockMvc mvc;

  @Before
  public void setUp() throws Exception {
    mvc = MockMvcBuilders.standaloneSetup(restController).build();
  }

  @Test
  public void testAddArticle() throws Exception {
    Article article = new Article();
    article.setTitle("测试文章000000");
    article.setType(1);
    article.setStatus(2);
    article.setSummary("这是一篇测试文章");
    Gson gosn = new Gson();
    RequestBuilder builder = MockMvcRequestBuilders
        .post("/rest/article")
        .accept(MediaType.APPLICATION_JSON)
        .contentType(MediaType.APPLICATION_JSON_UTF8)
        .content(gosn.toJson(article));

    MvcResult result = mvc.perform(builder).andReturn();
    System.out.println(result.getResponse().getContentAsString());
  }

  @Test
  public void testUpdateArticle() throws Exception {
    Article article = new Article();
    article.setTitle("更新测试文章");
    article.setType(1);
    article.setStatus(2);
    article.setSummary("这是一篇更新测试文章");
    Gson gosn = new Gson();
    RequestBuilder builder = MockMvcRequestBuilders
        .put("/rest/article/1")
        .accept(MediaType.APPLICATION_JSON)
        .contentType(MediaType.APPLICATION_JSON_UTF8)
        .content(gosn.toJson(article));
    MvcResult result = mvc.perform(builder).andReturn();
  }

  @Test
  public void testQueryArticle() throws Exception {
    RequestBuilder builder = MockMvcRequestBuilders
        .get("/rest/article/1")
        .accept(MediaType.APPLICATION_JSON)
        .contentType(MediaType.APPLICATION_JSON_UTF8);
    MvcResult result = mvc.perform(builder).andReturn();
    System.out.println(result.getResponse().getContentAsString());
  }

  @Test
  public void testDeleteArticle() throws Exception {
    RequestBuilder builder = MockMvcRequestBuilders
        .delete("/rest/article/1")
        .accept(MediaType.APPLICATION_JSON)
        .contentType(MediaType.APPLICATION_JSON_UTF8);
```

```
            .contentType(MediaType.APPLICATION_JSON_UTF8));
        MvcResult result = mvc.perform(builder).andReturn();
    }
}
```

执行结果这里就不给大家贴了，大家有兴趣的话可以自己实验一下。整个类要说明的点还是很少，主要这些东西都与Spring boot没关系，支持这些操作的原因还是上一篇文章中提到的引入对应的starter：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

因为要执行HTTP请求，所以这里使用了MockMvc，ArticleRestController通过注入的方式实例化，不能直接new，否则ArticleRestController就不能通过Spring IoC容器来管理，因而其依赖的其他类也无法正常注入。通过MockMvc我们就可以轻松的实现HTTP的DELETE/PUT/POST等方法了。

五、总结

本文讲解了如果通过Spring boot来实现Restful的API，其实大部分东西都是Spring和Spring MVC提供的，Spring boot只是提供自动配置的功能。

但是，正是这种自动配置，为我们减少了很多的开发和维护工作，使我们能更加简单、高效的实现一个web工程，从而让我们能够更加专注于业务本身的开发，而不需要去关心框架的东西。

- END -

推荐阅读

1. Java 线程有哪些不太为人所知的技巧与用法?

2. 关于 CPU 的一些基本知识总结

3. 发布没有答案的面试题，都是耍流氓

4. 什么是一致性 Hash 算法?

5. 团队开发中 Git 最佳实践

某小公司RESTful、共用接口、前后端分离、接口约定的实践

邵磊 Java后端 2019-10-10

作者 | 邵磊

链接 | juejin.im/post/59eafab36fb9a045076eccc3

前言

随着互联网高速发展，公司对项目开发周期不断缩短，我们面对各种需求，使用原有对接方式，各端已经很难快速应对各种需求，更难以提高效率。于是，我们不得不重新制定对接规范、开发逻辑以便快速上线项目。

我们的目标

- 尽可能的缩小沟通的成本，开最少的会，确定大部分的事。
- 花最少的时间写文档，保证90%的开发人员看懂所有内容。
- 哪怕不看文档，也能知道各种接口逻辑。
- 不重复写代码
- 尽可能的写高可读性的代码

我们做了哪些事

- 完成了前后端分离
- Android、ios、web共用一套接口
- 统一接口规范（post、put、get、patch、delete）
- 统一了调试工具
- 统一了接口文档

之前的我们

接口是这样子的：

接口地址	含义	请求方式
.../A项目/模块1/getProducts	获得产品	GET
.../A项目/模块1/addProduct	添加产品	POST
.../A项目/模块1/getProductDetail	获得产品详情	GET
.../A项目/模块1/editProduct	修改产品	POST

客户端请求是这样的：

- …/project/model1/getProducts?id=1&a=2&b=3&c=4&d=5……
- A页面->B页面（携带n个变量）->C页面（携带m个变量，包含i个A页面的变量）经常n>4
- 大部分请求是POST，至于put、patch、delete是什么鬼，关我屁事。
- 关于接口入参使用json，那完全是看开发心情。

出参是这样的：

```
{"message":"success","code":0,"data":具体内容}
```

其中data里包含数组可能是：

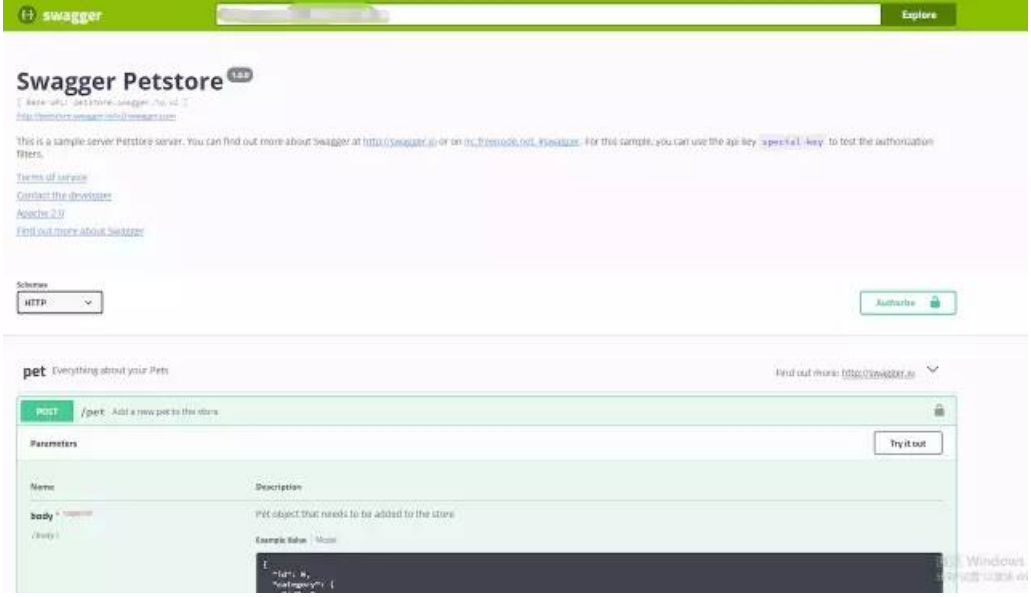
```
[{"a":"1","b":"1"},{"a":"1","b":"1"},{"a":"1","b":"1"},
```

{"a":"1","b":"1"}]即使下一个页面用到也不会使用id，而是把所有字段都传进去。A接口中，返回产品用product；B接口中使用good，多个接口很可能不统一。

客户端对接是这样子的：

- 安卓、ios一套;部分接口各自用一套；html5端一套。
- 客户端和后台是不停交流的

接口文档是这样的



- swagger
- 阿里的rap
- Word文档
- 其它

当然了，我觉得swagger和rap神器都是非常强大的，能够实现各种功能逻辑，但是考虑到开发人员掌握程度不通，复杂度较高，难以提高效率，我决定初期并不使用这两样神器。

后端是这样的

> …/Aproject/model1/getProducts ----接口
> …/Aproject/model1/Products.html ----页面
> …/Aproject/model1/Products.js ----静态资源

接口和静态资源缠在一块，毕竟很多页面可能是一位开发人员同时开发前端、后端，这里的弊端是，只需要自己清楚逻辑，很多做法临时应付，方案并不优雅，别人也很难看懂。一旦这位同事离职，很多说不清的逻辑就留给后人采坑了。

重构

下面步入正题，面对以上众多问题聊聊我是如何重新制定流程的

数据库约定

约定数据库里所有表必须包含名为id主键字段。可能有人会说，正常来说不是每张表里都应该有id主键吗？但是，我们项目中由于之前开发不严谨，部分表没有id主键，或者不为id的主键。这里我们采用分布式的全球唯一码来作为id。

api出参约定

约定所有出参里含list，且其他请求会用到这组list，则list里所有对象必须含id唯一标识。

入参约定

约定token身份认证统一传入参数模式，后端采用aop切面编程识别用户身份。其他参数一律为json。

resultfull接口约定

首先我们选择一个名词复数，比如产品

post方法

新增一条记录

比如 /products

则代表新增一条产品入参json如下：

```
{
    "name":"我是一款新产品"
,
    "price":100
,
    "kind":"我的分类"
,
    "pic":[一组图片]
,
     //等等还有很多
}
```

Java 代码control层

```
@ResponseBody
@RequestMapping(value = "/A项目/B模块/products", method = {RequestMethod.POST})
public ResultObject getProducts() {
    //具体逻辑
。
}
```

put方法

新增某条记录

比如 /products/1001

入参json如下：

```
{
    "name":"我是一款新产品"
,
    "price":100
,
    "kind":"我的分类"
,
    "pic":[一组图片]
,
     //等等还有很多
}
```

表示增加一条1001 id的记录

Java代码control层

```
  @ResponseBody
    @RequestMapping(value = "/A项目/B模块/products/{id}", method = {RequestMethod.PUT})
    public ResultObject putProducts(@PathVariable(value = "id") String id)
{
        //具体逻辑
。
    }
```

get方法

获得所有记录

/products 则代表获取所有产品

因为有分页，所以我们后面加了 ?page=1&pageSize=50

我们约定了所有名词复数，都会返回list，且list每个对象都有字段为id的唯一id。

比如

```
{
    "data":{"list":[{"id":"唯一id","其他很多字段":""},{"id":"唯一id","其他很多字段":""}],"page":1,其他字
,
    "code":0
,
    "message":"成功
"
}
```

/products/{id} 获取某个具体产品（一定比列表更详细）

比如某个具体产品里还包含一个list，如该产品推荐列表，则：/products/{id}/recommendations

假设它包含的不是一个list，而是对象，比如产品佣金信息，则：/products/{id}/Commission

这里我们以是否名词复数来判断是对象还是list.

Java代码control层

```
 @ResponseBody
@RequestMapping(value = "/A项目/B模块/products/{id}", method = {RequestMethod.GET})
public ResultObject putProducts(@PathVariable(value = "id") String id) {
    //具体逻辑
。
}
```

patch 方法

更新局部x产品y信息

入参是post方法时入参的子集，所有支持更新的参数会说明，并不是支持所有变量

如：/products/{id}

```
{
    "name":"我是一款新产品"
,
    "price":100
,
      //部分变
量
}
```

Java代码control层

```
@ResponseBody
@RequestMapping(value = "/A项目/B模块/products/{id}", method = {RequestMethod.PATCH})
public ResultObject putProducts(@PathVariable(value = "id") String id) {
    //具体逻辑
。
}
```

delete方法

删除某记录

如：/products/1001

删除1001产品。

Java代码control层

```
@ResponseBody
    @RequestMapping(value = "/A项目/B模块/products/{id}", method = {RequestMethod.DELETE})
    public ResultObject putProducts(@PathVariable(value = "id") String id)
{
        //具体逻辑
。
    }
```

其他说明

我们尽可能少的使用动词，但有一些行为需要使用动词，比如登录等。关于版本号，我们打算在模块后增加/v1/等标识。

权限约定

服务端要对用户角色进行判断，是否有权限执行某个逻辑。

前后端分离约定

后端以开发接口为主，不再参与页面开发，或者混合式jsp页面开发，统一以接口形式返回，前端通过js渲染数据以及处理逻辑。

共用接口

web、Android、ios使用统一接口，不在因为哪方特殊要求额外开放接口。

使用统一dao层生成工具

基于mybatis-generator改造成适合我们项目的dao层以及部分service层，内部共同维护共同使用。

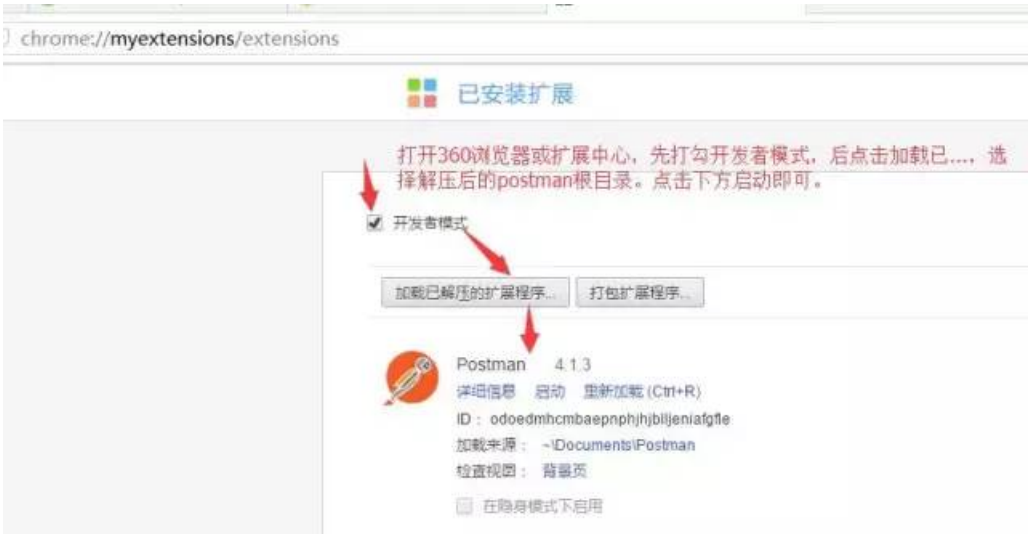
使用postman最为接口文档、调试工具

虽然有上文中介绍的rap和swagger都是特别牛的接口神器，但是我们还是选择了postman，开发人员将接口名称、说明、入参、出参，以及各种出参示例都存储，这样开发直接可以看得清接口含义。

我们建议使用浏览器插件，这里以360极速浏览器为例。插件下载地址：

> https://download.csdn.net/download/qq273681448/10033456

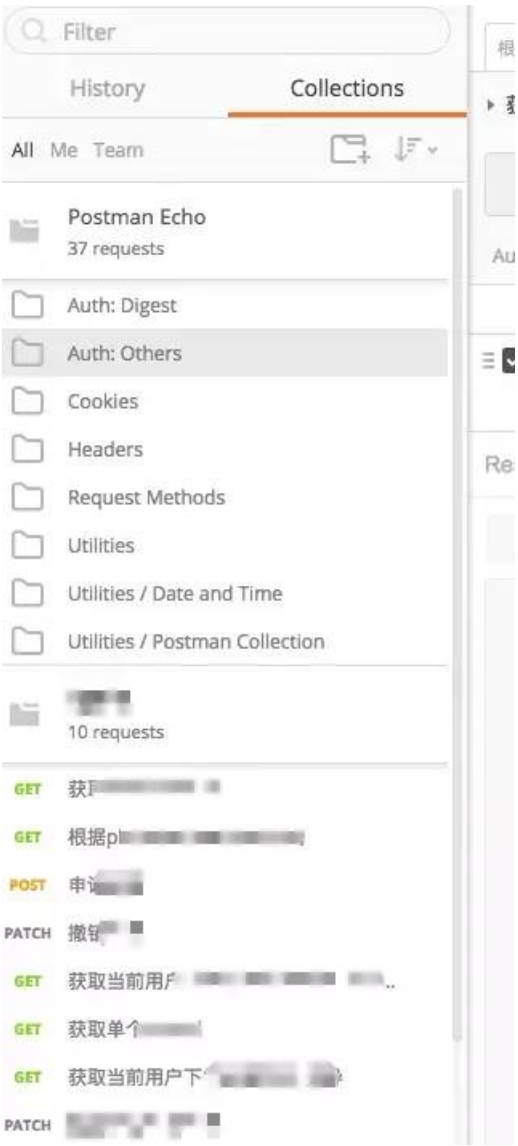
打开360浏览器扩展中心，然后勾选开发者模式，再点击加载已解压的扩展程序，选中压缩包解压后的目录，最后点击运行即可。



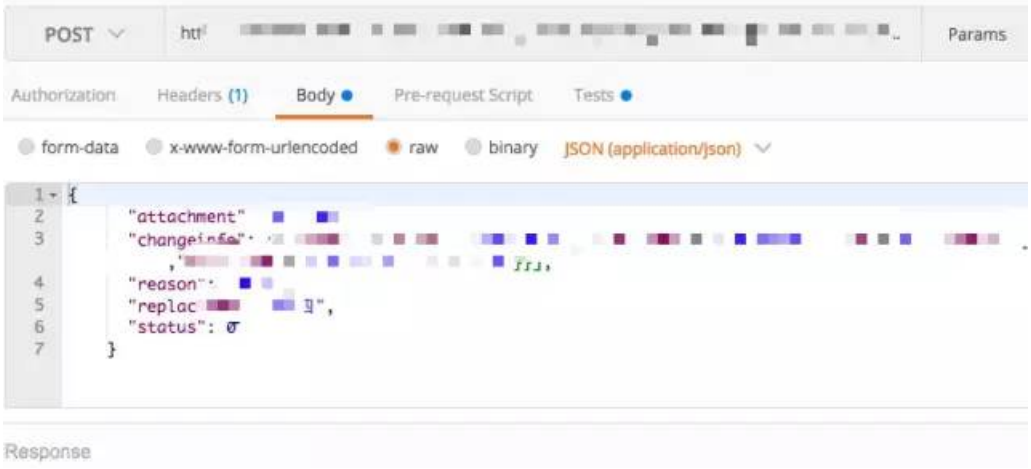
其中出参注释、及接口说明，写在tests里：

```
/*
这里是接口说明，和每个出参、入参的意思。
*/
```

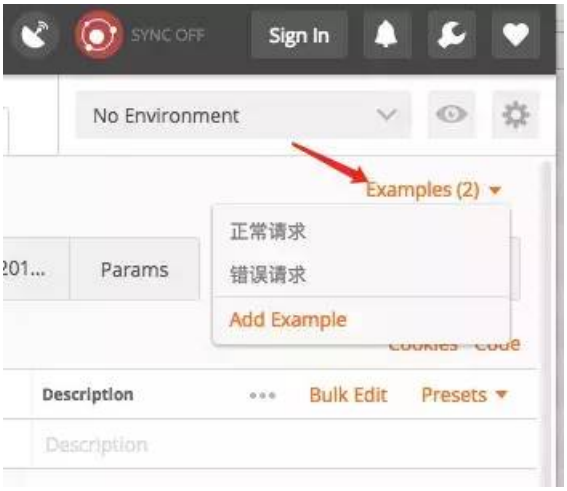
接口按模块划分为文件夹：



入参：



出参示例：



正常请求：

开发人员即可直接看到接口示例进行开发，而开发人员开发的时候，自己调用一次即可保存为postman文件，为了加快上线，我们允许将java中实体类变量定义的代码（含注释）直接复制粘贴出来。

js等静态资源缓存问题

从短期角度上讲，我的要求是减少js文件的变更，如果有变更，务必更改版本号。那么如何减少修改，我们的做法是将一部分js写在html内，反正前后端分离，大不了刷新一下cdn的节点缓存，毕竟大部分浏览器也不会主动缓存html文件（大部分浏览器会缓存js等文件）。

统一js请求框架

这里我们使用angular js的请求框架，因为我们内部对angularjs使用较多，比较熟悉，封装后的请求，可以自动弹窗错误请求，可复写错误回调。

目前效果

目前，我们客户端看到接口，大概能说出其意思，也能猜出一连串接口的含义，比如

如：/classes

可以看出它是获取班级列表接口，

猜到
/classes/id get获取id为id的班级详情
/classes/id patch 修改班级信息
/classes/id delete 删除班级信息

至于入参，patch是post的子集、put=patch、delete无入参。

而入参含义，直接打开postman可以直接查看每个字段的含义，并且，可以实时调取开发环境数据（非开发人员电脑），这里我们使用了多环境，详情可了解：

> https://juejin.im/post/59e1d92d51882578db27c2e1

前端使用统一封装后的js请求框架也加快了开发进度，不用造轮子。

开发人员，一般代码开发写好，使用postman自我测试，测试完成后，接口文档也就写好了。
测试人员想了解接口文档的也可以直接使用postman进行导入查看。

至此，我们交流成本下降了一大半，剩下开会的内容就是按ui分解需求或者按ui施工了。

总结

经过一番的折腾，开发进度总算快了点，也一定程度上达到了快速上线项目的效果。关于restful风格api，每个人都有自己的见解，只要内部约定清楚，能尽可能少的减少沟通，我觉得就是好的理解。至于接口工具，可能很多人会说为什么不用之前的，我觉得以后还是会用的，最好能做到插件自动化生成api，但是对java开发注释要求比较严格，随意慢慢来吧，毕竟后面我们还有很多路要走。

-END-

推荐阅读

1. 团队开发中 Git 最佳实践
2. Google 出品 Java 编码规范，强烈推荐！
3. 支付系统高可用架构设计实战
4. 重构你乱糟糟的代码
5. 如何设计 API 接口，实现统一格式返回？

喜欢文章，点个在看