Căn bản về RESTful Web services

Alex Rodriguez (arodrigu@us.ibm.com)
Kỹ sư phần mềm
IBM

07 03 2013
(Xuất bản lần đầu tiên vào ngày 22 05 2009)

REST (Representational State Transfer) đã được chọn sử dụng rộng rãi thay cho Web service dựa trên SOAP và WSDL. Bằng chứng quan trọng của sự thay đổi này chính là việc các công ty dẫn đầu trong lĩnh vực cung cấp dịch vụ mạng 2.0 như Yahoo, Google và Facebook đã phản đối các giao thức dựa trên SOAP hoặc WSDL và ủng hộ phương thức hướng đến tài nguyên và dễ sử dụng đối với các dịch vụ của họ. Trong bài viết này, Alex Rodriguez sẽ giới thiệu với các bạn các nguyên lý cơ bản của REST.

Nhập môn

REST định nghĩa các quy tắc kiến trúc để bạn thiết kế Web services chú trọng vào tài nguyên hệ thống, bao gồm các trạng thái tài nguyên được định dạng như thế nào và được chuyển tải qua HTTP thông qua số lượng lớn người dùng và được viết bởi những ngôn ngữ khác nhau. Nếu tính theo số dịch vụ mạng sử dụng, REST đã nổi lên trong vài năm qua như là một mô hình thiết kế dịch vụ chiếm ưu thế. Trong thực tế, REST đã có những ảnh hưởng lớn và gần như thay thế SOAP và WSDL vì nó đơn giản và dễ sử dụng hơn rất nhiều.

REST không thu hút được nhiều sự chú ý khi lần đầu tiên giới thiệu vào năm 2000 bởi Roy Fielding trong luận án của ông "Architectural Styles and the Design of Network-based Software Architectures" (Phong cách kiến trúc và thiết kế kiến trúc phần mềm dựa trên mạng) tại Đại học California. Luận án đã phân tích một loạt các nguyên tắc kiến trúc phần mềm sử dụng Web như là một nền tảng tính toán phân tán (xem Tài nguyên liên kết tới luận án). Đến nay, vài năm sau đó, đã xuất hiện các framework chủ đạo cho REST và chúng vẫn đang được tiếp tục phát triển, nó đang được xem xét để đưa vào trong bộ Java™ 6 thông qua tiêu chuẩn JSR-311.

Bài viết này chỉ ra rằng khi REST được biết đến nhiều hơn thì việc cụ thể hóa một Web service REST sẽ tuân thủ theo bốn nguyên tắc thiết kế cơ bản sau:

- Sử dụng các phương thức HTTP một cách rõ ràng
- Phi trạng thái
- Hiển thị cấu trúc thư mục như URIs
- Chuyển đổi JavaScript Object Notation (JSON) và XML hoặc cả hai.

Các phần sau đây sẽ mở rộng dựa trên bốn nguyên lý này và đề xuất một nhân tố kỹ thuật cơ bản giải thích vì sao chúng quan trọng đối với các nhà thiết kế dịch vụ mạng REST.

Sử dụng **các** phương thức **HTTP** một **cách rõ ràng**

Một đặc tính quan trọng của dịch Web service RESTful là sử dụng một cách rõ ràng các phương thức HTTP theo cách một giao thức được xác định bởi RFC 2616. Ví dụ HTTP GET được xác định như là một phương thức sinh ra số liệu được sử dụng có chủ đích bởi các ứng dụng người dùng để thu thập tài nguyên, dữ liệu từ một máy chủ, hoặc thực thi một truy vấn mà máy chủ sẽ tìm kiếm và phản hồi cùng với một gói thông tin tương thích.

REST yêu cầu các nhà phát triển sử dụng phương thức HTTP một cách rõ ràng theo cách tương thích với giao thức chuẩn. Nguyên lý thiết kế REST cơ bản này thiết lập một ánh xạ 1-1 giữa các hành động tạo, đọc, cập nhật và xoá (CRUD) các quá trình vận hành và các phương thức HTTP. Theo cách ánh xạ này thì:

- Để tạo một tài nguyên trên máy chủ, bạn cần sử dụng phương thức POST.
- Để truy xuất một tài nguyên, sử dụng GET.
- Để thay đổi trạng thái một tài nguyên hoặc để cập nhật nó, sử dụng PUT.
- Để huỷ bỏ hoặc xoá một tài nguyên, sử dụng DELETE.

Một lỗ hổng trong thiết kế vốn có trong các Web API là việc sử dụng các phương thức HTTP mà không có mục đích trước. Ví dụ lệnh URI trong một lệnh HTTP GET thường xác định một tài nguyên cụ thể. Hoặc một chuỗi truy vấn trong một lệnh URI bao gồm một nhóm các tham số xác định tiêu chí tìm kiếm được máy chủ sử dụng để tìm các tài nguyên phù hợp. Ít nhất điều này cho thấy HTTP/1.1 miêu tả GET như thế nào. Nhưng có nhiều trường hợp Web APIs không được tốt lắm, do sử dụng HTTP GET để khởi động một vài tác vụ trên máy chủ — ví dụ, thêm các bản ghi vào một cơ sở dữ liệu. Trong các trường hợp này, phương thức GET-yêu-cầu-URI đã không được sử dụng đúng đắn hoặc chưa sử dụng đầy đủ. Nếu Web API sử dụng GET để tiến hành các thủ tục ra lệnh từ xa thì nó sẽ có dạng như sau:

```
GET /adduser?name=Robert HTTP/1.1
```

Đây không phải là mẫu thiết kế hấp dẫn vì phương pháp nói trên hỗ trợ phương thức thay đổi trạng thái trên HTTP GET. Nói cách khác, phương thức yêu cầu HTTP GET nói trên có những tác động phụ. Nếu được xử lý thành công, kết quả của yêu cầu (trong ví dụ này) là để tạo thêm một người dùng mới vào kho dữ liệu. Vấn đề ở đây chỉ về mặt ngữ nghĩa. Các Web server được thiết kế để phản hồi lại các yêu cầu HTTP GET bằng cách truy vấn dữ liệu phù hợp với đường dẫn (hoặc câu truy vấn) theo URI và phản hồi những dữ liệu này hoặc những thông tin đại diện, chứ không phải để thêm một dữ liệu vào database. Từ góc độ mục đích sử dụng giao thức đó và theo tiêu chuẩn Web server HTTP/1.1 thì việc sử dụng GET theo cách này tồn tại mâu thuẫn.

Về mặt ngữ nghĩa, có nhiều vấn đề khi sử dụng GET là để khởi động một sự xoá bỏ, sửa đổi, hoặc ghi thêm vào cơ sở dữ liệu, hoặc để thay đổi trạng thái máy chủ theo một cách nào đó. Nó dùng các công cụ Web cache (các đường dẫn) và các công cụ tìm kiếm để làm thay đổi máy chủ một cách không chủ định thông qua đường dẫn. Một cách đơn giản để vượt qua vấn đề hay xảy ra này là di chuyển tên và giá trị các tham số yêu cầu trên URI vào các thẻ XML. Các thẻ kết quả, một đại

diện XML của một chủ thể được tạo ra, có thể được gửi vào một nhóm HTTP POST, những nơi mà yêu cầu URI là chủ thể sinh ra có chủ đích (xem ví dụ 1 và 2).

Ví dụ 1. Trước

```
GET /adduser?name=Robert HTTP/1.1
```

Ví dụ 2. Sau

```
POST /users HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
  <name>Robert</name>
</user>
```

Phương pháp trên là ví dụ của yêu cầu RESTful: sử dụng đúng HTTP POST và bao gồm cả tải trọng trong phần thân câu lệnh. Đối với phía đầu tiếp nhận, câu lệnh có thể được xử lý bằng cách thêm tài nguyên vào hệ thống đó như một phần tài nguyên phụ được định dạng trong lệnh URI; trong trường hợp này tài nguyên mới phải được thêm vào như là một nhánh con của `/users`. Quan hệ bao hàm giữa thực thể mới và nhánh cha, như đã xác định trong yêu cầu POST, tương tự như cách tệp đã thuộc thư mục cha. Máy khách thiết lập quan hệ giữa thực thể và nhánh cha, và xác định URI của thực thể mới trong yêu cầu POST.

Ứng dụng máy khách sau đó có thể nhận kết nối của nguồn sử dụng URI mới, lưu ý rằng ít nhất về mặt logic, nguồn được đặt dưới `/users`, như trong ví dụ 3.

Ví dụ 3. Lệnh HTTP GET

```
GET /users/Robert HTTP/1.1
Host: myserver
Accept: application/xml
```

Sử dụng GET theo cách này rất rõ ràng vì GET chỉ dành cho truy cập dữ liệu. GET là một phương thức mà không có hiệu ứng phụ, như là một đặc tính riêng *không thay đổi giá trị*.

Tương tự, những thay đổi của phương thức Web cũng cần được ứng dụng trong các trường hợp khi một thao tác cập nhật được hỗ trợ qua HTTP GET, như thể hiện trong ví dụ 4.

Ví dụ 4. Thực hiện lệnh Cập nhật thông qua HTTP GET

```
GET /updateuser?name=Robert&amp;newname=Bob HTTP/1.1
```

Câu lệnh này thay đổi thuộc tính (hoặc đặc tính) `name` của dữ liệu. Có thể dùng chuỗi truy vấn (query) để dùng cho những thao tác như thế này, và ví dụ 4 là một ví dụ đơn giản, mẫu phương-pháp-dấu-hiệu-như-là-chuỗi-truy-vấn (query-string-as-method-signature) có thể không hoạt động khi sử dụng đối với các thao tác phức tạp hơn. Do mục tiêu của chúng ta là làm rõ việc sử dụng các phương thức HTTP, nên cách tiếp cận RESTful là gửi một yêu cầu HTTP PUT để cập nhật tài nguyên, thay vì HTTP GET, cho những lý do tương tự chỉ ra ở trên (xem ví dụ 5).

Ví dụ 5. Lệnh HTTP PUT

```
PUT /users/Robert HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
  <name>Bob</name>
</user>
```

Sử dụng PUT để thay đổi dữ liệu gốc, cho thấy cách làm rõ ràng hơn, phù hợp với các nguyên lý của REST và các khái niệm của các phương thức HTTP. Lệnh PUT trong ví dụ 5 rõ ràng ở chỗ nó chỉ ra dữ liệu được cập nhật bằng cách xác định nó trong câu lệnh URI, và nó chuyển một đại diện mới của các thuộc tính dữ liệu cần chuyển đổi như là nhóm không chặt chẽ các tên tham số và giá trị trên lệnh URI. Ví dụ 5 cũng có hiệu ứng từ việc đổi tên tài nguyên từ `Robert` sang `Bob`, và trong việc các thay đổi của URI sang `/users/Bob`. Trong một dịch vụ mạng REST, lệnh tiếp theo của tài nguyên sử dụng URI cũ sẽ sinh ra lỗi căn bản 404 Not Found.

Như là một nguyên tắc thiết kế chung, nó giúp theo sát các hướng dẫn sử dụng REST để sử dụng các phương pháp HTTP một cách rõ ràng bằng cách sử dụng các danh từ trong URIs thay vì động từ. Trong một Web service RESTful, các động từ — POST, GET, PUT, và DELETE — đã được định nghĩa bởi giao thức. Và tốt nhất, để giữ giao diện được khái quát hoá và cho phép người dùng hiểu rõ các thao tác mà họ gọi thì Web service không nên đưa ra nhiều động từ hoặc các thủ tục remote từ xa, như `/adduser` hoặc `/updateuser`. Nguyên tắc thiết kế chung này cũng áp dụng đối với phần thân câu lệnh HTTP, được sử dụng có chủ ý để chuyển trạng thái tài nguyên, không mang tên của một phương thức hay thủ tục remote từ xa.

Phi trạng thái

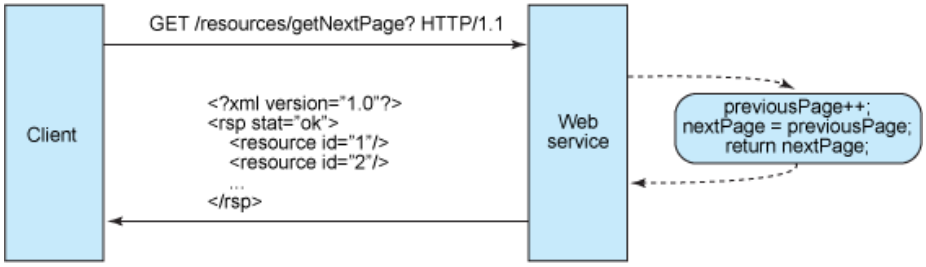
Các Web service REST cần được điều chỉnh về quy mô để đáp ứng được các yêu cầu ngày càng cao về chất lượng thực hiện. Các khu vực lưu trữ của máy chủ với khả năng cân bằng tải và vượt qua sự mất mát, các bức ngăn (tường lửa) và các cổng được sắp xếp theo một phương thức đặc thù nhằm tạo ra một cấu trúc dịch vụ bền vững cho phép chuyển tiếp yêu cầu từ một máy chủ tới máy chủ khác khi cần để giảm tổng thời gian phản hồi của một yêu cầu Web service. Sử dụng máy chủ trung gian nhằm nâng cao mức yêu cầu dịch vụ mạng REST của khách hàng để gửi các yêu cầu hoàn chỉnh và độc lập, có nghĩa là gửi các yêu cầu bao gồm tất cả dữ liệu cần thiết để đáp ứng sao cho các thành phần trong các máy chủ trung gian có thể gửi tiếp đi, gửi theo tuyến và cân bằng tải mà không cần các trạng thái được kiểm soát bên trong giữa các yêu cầu.

Một yêu cầu hoàn chỉnh, độc lập không đòi hỏi máy chủ để thu thập được bất kỳ ngữ cảnh hoặc trạng thái của ứng dụng nào trong lúc xử lý yêu cầu. Một ứng dụng (hoặc máy khách) Web service RESTchứa ở phần đầu và phần thân trang HTTP của một yêu cầu tất cả các tham số, ngữ cảnh và dữ liệu cần thiết bởi thành phần bên ngoài máy chủ để đưa ra một phản hồi. Phi trạng thái theo nghĩa này nâng cao tính hiệu quả của dịch vụ Web, đơn giản hoá thiết kế và sự thi hành của các thành phần của máy chủ vì khi máy chủ không có trạng thái sẽ huỷ bỏ nhu cầu để đồng bộ hoá các mảng dữ liệu với một ứng dụng bên ngoài.

Hình 1 minh hoạ một dịch vụ trạng thái, từ đó một ứng dụng có thể yêu cầu trang sau trong một tập hợp các trang kết quả, giả sử rằng dịch vụ theo sát ứng dụng dừng lại ở nơi trong khi điều chỉnh tập

hợp đó. Đối với thiết kế trạng thái, dịch vụ gia tăng và lưu giữ một `previousPage` (trang trước) thay đổi ở nơi để có thể phản hồi các lệnh tiếp theo.

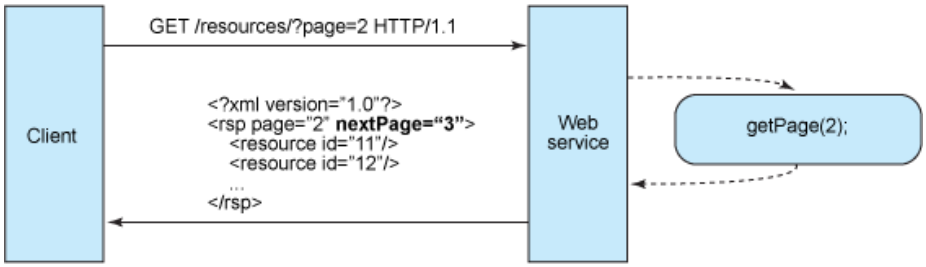
Hình 1. Thiết kế trạng **thái**



Dịch vụ trạng thái như thế này trở nên phức tạp. Trong môi trường Nền tảng Java, Phiên bản Doanh nghiệp (EE), dịch vụ trạng thái yêu cầu rất cẩn thận lúc ban đầu để lưu trữ hiệu quả và cho phép đồng bộ hoá dữ liệu session (phiên làm việc) qua một hệ thống container Java EE. Trong môi trường này, có một vấn đề quen thuộc đối với các chuyên viên phát triển servlet/JavaServer Pages (JSP) và Enterprise JavaBeans (EJB), những người này thường gặp khó khăn khi tìm gốc rễ nguyên nhân của `java.io.NotSerializableException` trong khi tái tạo session. Liệu nó được chuyển bởi thành phần chứa Servlet trong khi `HttpSession` được tái tạo hoặc chuyển đi bởi thành phần chứa EJB trong khi sao bản EJB trạng thái, đó là vấn đề mà có thể làm các chuyên viên phát triển mất nhiều ngày để xác định mấu chốt một đối tượng mà không thực thi `Serializable`, đôi khi trong một đồ thị phức tạp của các đối tượng mà đóng góp nên trạng thái của máy chủ. Ngoài ra, phần tối ưu hoá làm đội thêm chi phí ảnh hưởng đến hiệu quả của máy chủ.

Mặt khác, các thành phần máy chủ phi trạng thái ít phức tạp hơn để thiết kế, viết và phân bổ thông qua máy chủ được cân bằng tải. Dịch vụ phi trạng thái không chỉ hoạt động tốt hơn, nó còn chuyển hầu hết vai trò duy trì trạng thái sang ứng dụng ở máy khách. Trong một dịch vụ mạng RESTful, máy chủ chịu trách nhiệm đưa ra các phản hồi và cung cấp một giao diện cho phép máy khách duy trì trạng thái ứng dụng của chính nó. Ví dụ, trong yêu cầu tập hợp trang kết quả, máy khách sẽ gồm số trang thực tế khi truy xuất thay vì đơn giản chỉ là yêu cầu tiếp *theo* (xem hình 2).

Hình 2. Thiết kế **phi** trạng **thái**



Một dịch Web phi trạng thái sinh ra một phản hồi liên kết với số trang tiếp theo trong một tổng thể và để máy khách làm những gì mà nó cần để giữ giá trị này ở mức nhất định. Khía cạnh này của thiết kế dịch vụ Web RESTful có thể được tách thành hai phần trách nhiệm như là mức phân chia cao nhất mà chỉ rõ một dịch vụ phi trạng thái có thể được duy trì như thế nào.

Máy chủ

- Tạo ra các phản hồi bao gồm các đường dẫn tới nguồn tài nguyên cho phép các ứng dụng điều hướng giữa các tài nguyên liên quan. Loại phản hồi này nhúng các liên kết. Tương tự, nếu các yêu cầu đối với máy chủ hoặc các kho tài nguyên, thì các phản hồi RESTful Web service điển hình có thể bao gồm các đường dẫn đến các máy con hoặc các tài nguyên phụ sao cho những phản hồi này được duy trì kết nối.
- Tạo ra các phản hồi mà xác định chúng có thể lưu trữ hoặc không phải để nâng cao được hiệu quả bằng cách giảm số lượng yêu cầu đối với các tài nguyên trùng nhau và bằng cách loại trừ một vài yêu cầu toàn bộ. Máy chủ làm được như vậy bằng cách gộp một phản hồi phần đầu HTTP Last - Modified (lần sửa gần nhất) (giá trị ngày) và Cache-Control (bộ điều khiển lưu trữ).

Ứng dụng **máy khách**

- Sử dụng phần đầu phản hồi Cache-Control (bộ điều khiển lưu trữ tạm) để xác định lưu trữ tài nguyên (lập một vùng sao chép nội bộ) hay không. Máy khách cũng đọc phần đầu phản hồi Last-Modified (lần sửa gần nhất) và gửi lại giá trị ngày vào phần đầu If-Modified-Since (nếu-sửa) để truy vấn máy chủ xem tài nguyên có thay đổi không. Việc này được gọi là truy vấn có điều kiện, và hai phần đầu đi với nhau trong phản hồi của máy chủ là mã 304 chuẩn (không sửa đổi) và bỏ qua tài nguyên thực được yêu cầu nếu nó không thay đổi. Mã phản hồi HTTP 304 có nghĩa rằng máy khách có thể sử dụng an toàn một vùng sao lưu nội bộ, lưu giữ một bản sao mới nhất của tài nguyên đại diện, hiệu quả bằng cách vượt qua yêu cầu GET tiếp theo cho đến khi tài nguyên thay đổi.
- Gửi các yêu cầu hoàn chỉnh có thể được đáp ứng độc lập bởi các yêu cầu khác. Điều này đòi hỏi máy khách sử dụng toàn bộ các phần đầu HTTP như chỉ định bởi giao diện dịch vụ mạng và để gửi các đại diện tài nguyên hoàn chỉnh trong phần giữa của yêu cầu. Máy khách gửi yêu cầu lập một vài giả thuyết về các yêu cầu trước đó, sự tồn tại của một vùng của máy chủ, khả năng của máy chủ để thêm các ngữ cảnh vào yêu cầu, hoặc về các trạng thái ứng dụng mà được giữ giữa các yêu cầu.

Sự hợp tác này giữa ứng dụng máy khách và máy chủ là cần thiết để có một phi trạng thái trong một Web service RESful. Nó nâng cao hiệu quả bằng cách tiết kiệm băng thông và tối thiểu hoá trạng thái ứng dụng về phía máy chủ.

Đưa **ra** cấu **trúc** thư mục giống **URIs**

Từ điểm hiện có của tài nguyên địa chỉ ứng dụng máy khách, các đường dẫn xác định tính hiện thực của Web service REST như thế nào, và được sử dụng theo cách các chuyên viên thiết kế có thể tham gia. Tính năng thứ ba của Web service RESful về tất cả đường dẫn.

Các địa chỉ Web service REST nên có tính hiện thực theo nghĩa rằng chúng dễ dàng đối với người dùng. Có thể nghĩ rằng một địa chỉ đường dẫn như là giao diện tự đóng gói mà đòi hỏi ít lý giải hoặc tham chiếu, nếu có, đối với một nhà phát triển để hiểu nó nhắm đến điểm gì và phân phát tài nguyên liên quan. Cuối cùng, cấu trúc của một địa chỉ nên rõ ràng, có thể đoán được và dễ hiểu.

Một cách để đạt được mức độ sử dụng này là xác định cấu trúc thư mục giống URIs. Loại URI này có thứ bậc, có điểm khởi nguồn tại một đường dẫn đơn giản, và có nhánh đi ra là các nhánh phụ thể hiện các vùng chính của dịch vụ. Theo định nghĩa này, một URI không chỉ là một chuỗi bị cắt

không giới hạn, mà còn là một cây với các nhánh chính và nhánh dọc nối với nhau tại các nút. Ví dụ, trong một thảo luận dịch vụ nhỏ thu thập các chủ đề từ Java tới bài viết, bạn có thể định nghĩa một tập hợp được cấu trúc bởi URIs giống như sau:
`http://www.myservice.org/discussion/topics/{topic}`

Phần gốc, `/discussion`, có một nút `/topics` bên dưới nó. Phía dưới là một chuỗi tên các chủ đề, như chuyện xã hội, kỹ thuật, v.v.., mỗi chủ đề chỉ ra một mạch thảo luận. Trong cấu trúc này, dễ dàng kéo các mạch thảo luận bằng cách gõ một vài thứ sau /topics/.

Trong một vài trường hợp, đường dẫn tới một tài nguyên cho mượn chính nó đặc biệt tốt với cấu trúc giống cây thư mục. Ví dụ, tài nguyên được cấu trúc bởi ngày, điều mà phối hợp rất tốt để sử dụng một cú pháp phân cấp.

Ví dụ này trực quan vì nó dựa trên các nguyên tắc:
`http://www.myservice.org/discussion/2008/12/10/{topic}`

Phần đầu tiên trong đường dẫn là năm có bốn chữ số, phần thứ hai là ngày có hai chữ số, và phần thứ ba là tháng có hai chữ số. Có vẻ hơi ngu ngốc khi giải thích theo cách này, nhưng đây là mức độ đơn giản. Con người và máy móc có thể dễ dàng sinh ra các cấu trúc URIs giống như vậy vì chúng dựa trên các nguyên tắc. Bổ sung vào các phần đường dẫn trong các khe của một cú pháp làm cho chúng tốt hơn vì có một mẫu xác định từ đó để soạn chúng.
`http://www.myservice.org/discussion/{year}/{day}/{month}/{topic}`

Một vài hướng dẫn bổ sung để lưu ý trong khi nói về cấu trúc địa chỉ của Web service RESTful là:

- Giấu các đuôi tài liệu mở rộng của bản gốc trong máy chủ (.jsp, .php, .asp), nếu có, vì vậy bạn có thể giấu một số thứ mà không cần thay đổi địa chỉ Urls.
- Để mọi thứ là chữ thường.
- Thay thế các khoảng trống bằng gạch chân hoặc hoặc gạch nối (một trong hai loại).
- Tránh các chuỗi yêu cầu càng nhiều càng tốt.
- Thay vì sử dụng mã (404 Not Found) khi yêu cầu địa chỉ cho một phần đường dẫn, luôn luôn cung cấp một trang mặc định hoặc tài nguyên như một phản hồi.

Các địa chỉ URIs nên giữ nguyên để khi tài nguyên thay đổi hoặc khi tiến hành thay đổi dịch vụ, đường liên kết cũng sẽ giữ nguyên. Việc này cho phép đánh dấu lại vị trí đang đọc. Nó cũng rất quan trọng vì mối liên quan giữa các tài nguyên mà được mã hoá trong các địa chỉ được giữ nguyên độc lập với các mối liên quan đại diện khi chúng được lưu trữ.

Chuyển đổi **XML, JSON,** hoặc cả **hai**

Một tài nguyên đại diện điển hình phản ánh trạng thái hiện tại của một tài nguyên, và các thuộc tính của nó, tại thời điểm một ứng dụng máy khách yêu cầu nó. Các đại diện tài nguyên theo nghĩa này là chỉ các bản tóm tắt lúc đó. Điều này có thể đơn giản như một đại diện của một bản ghi trong một cơ sở dữ liệu, bao gồm một tổng thể giữa tên các cột và các thẻ XML, nơi các giá trị thành phần trong XML bao gồm các giá trị dòng. Hoặc nếu hệ thống có một mô hình dữ liệu, thì theo định nghĩa này một tài nguyên đại diện là một bản tóm tắt các thuộc tính của những thứ trong mô hình dữ liệu hệ thống. Đây là những điều bạn muốn Web services REST mang đến.

Những trở ngại cuối cùng, đi kèm với thiết kế Web service RESTful, phải làm với định dạng dữ liệu mà ứng dụng và trao đổi dịch vụ trong mức đáp ứng yêu cầu/phản hồi hoặc trong phần thân của HTTP. Đây chính là điều nó làm để giữ mọi thứ đơn giản, có thể đọc được và kết nối được.

Các chủ thể trong mô hình dữ liệu của bạn thường liên quan đến nhau theo cách nào đó, và mối liên hệ giữa mô hình dữ liệu chủ thể (tài nguyên) nên được phản ánh theo cách chúng được đại diện để chuyển đến một ứng dụng máy khách. Trong dịch vụ chuỗi thảo luận, một ví dụ của đại diện tài nguyên được kết nối có thể bao gồm một chủ đề thảo luận gốc và các thuộc tính của nó, và các đường dẫn được nhúng vào các phản hồi nhất định của chủ đề đó.

Ví dụ **6.** Đại diện **XML** của một chuỗi thảo luận

```
<?xml version="1.0"?>
<discussion date="{date}" topic="{topic}">
  <comment>{comment}</comment>
  <replies>
    <reply from="joe@mail.com" href="/discussion/topics/{topic}/joe"/>
    <reply from="bob@mail.com" href="/discussion/topics/{topic}/bob"/>
  </replies>
</discussion>
```

Cuối cùng, để đưa đến khả năng yêu cầu loại nội dung cụ thể cho các ứng dụng máy khách mà phù hợp nhất với chúng, hãy cấu trúc dịch vụ của bạn sao cho nó tận dụng được phần đầu chấp nhận HTTP sẵn có bên trong, nơi giá trị của phần đầu là một loại MIME. Một vài loại MIME thông thường được sử dụng bởi dịch vụ RESTful được thể hiện trong Bảng 1.

Bảng **1.** Loại **MIME** phổ biến được sử dụng bởi dịch vụ **RESTful**

MIME-Type	Content-Type
JSON	application/json
XML	application/xml
XHTML	application/xhtml+xml

Nó cho phép dịch vụ được nhiều khách hàng khác nhau sử dụng, viết bằng các ngôn ngữ khác nhau, chạy trên nền và thiết bị khác nhau. Sử dụng kiểu MIME và phần đầu HTTP Accept là một cơ chế được biết như là nội *dung* thương thuyết, cái mà cho phép máy khách chọn định dạng dữ liệu nào là đúng với chúng và tối thiểu hoá sự nối lại giữa dịch vụ và các ứng dụng mà sử dụng nó.

Kết luận

REST không phải là sự lựa chọn luôn đúng. Đã có trường hợp khi thiết kế Web service, nó có sự kém độc lập đối với phần mềm trung gian (ví dụ: một ứng dụng máy chủ) so với loại dựa trên SOAP hặc WSDL. Có nghĩa REST là một sự quay lại con đường mạng trước thời đại ứng dụng máy chủ rất lớn, kể cả ấn tượng các chuẩn của internet thời kỳ đầu, địa chỉ và HTTP. Bạn vừa nghiên cứu nguyên tắc gọi là thiết kế giao diện RESTful, XML so với HTTP là một giao diện vượt trội cho phép các ứng dụng bên trong, như các giao diện người dùng dựa trên Asynchronous JavaScript + XML (Ajax), dễ dàng kết nối, xác định, và tiêu thụ tài nguyên. Thực tế, sự tương thích lớn giữa Aax và REST đã nhận được rất nhiều sự chú ý gần đây.

Đưa một tài nguyên hệ thống thông qua một RESTful API là một cách linh động để cung cấp các loại ứng dụng khác nhau với dữ liệu đã được định dạng theo cách tiêu chuẩn. Nó giúp đáp ứng các yêu cầu tích hợp, điều rất quan trọng để xây dựng hệ thống khi dữ liệu được kết hợp dễ dàng (mashups) và để mở rộng hoặc xây dựng trên một gói hệ thống căn bản. Dịch vụ RESTful có thể rất lớn đối với một số thứ. Bài viết này hướng dẫn dựa vào nguyên lý cơ bản trên và hy vọng bằng cách nào đó sẽ khuyến khích bạn khám phá chủ đề này.

Tài nguyên

Học tập

- Đọc chương 5 luận án của Roy Fielding Architectural Styles and the Design of Network-based Software Architectures (Phong cách kiến trúc và thiết kế kiến trúc phần mềm dựa trên mạng)."
- Tìm thêm thông tin tại RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1
- Đọc sách *RESTful Web Services*.
- Địa chỉ SOA and Web services zone trên trang web của IBM developerWorks, đăng hàng trăm thông tin về các bài viết phong phú thông tin và các bài giảng về giới thiệu, trung cấp, hoặc cao cấp để làm sao phát triển ứng dụng dịch vụ mạng.
- Thực hành tại IBM SOA Sandbox! Nâng cao kỹ năng SOA của bạn thông qua thực hành, kinh nghiệm thực tế cùng với các bài nhập môn IBM SOA.
- Trang web IBM SOA đưa ra một tổng thể của SOA và làm thế nào IBM có thể giúp bạn ở đây.
- Hãy tham gia các sự kiện kỹ thuật developerWorks và web quảng bá .
- Tìm các sách cùng hoặc khác chủ đề kỹ thuật tại Hiệu sách Safari.
- Đọc thêm các tài liệu Web services on demand demo.

Lấy sản phẩm **và công** nghệ

- Tải về JSR 311: JAX-RS: The Java API for RESTful Web Services.
- Lấy Jersey (GlassFish).
- Tải về Restlet, the REST framework for Java.
- Tải về Phiên bản đánh giá sản phẩm của IBM và bạn có các công cụ phát triển ứng dụng và các phần mềm trung gian từ DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Thảo luận

- Tham gia vào cộng đồng các chuyên viên phát triển bằng cách gia nhập vào developerWorks blogs.

Đôi nét về **tác** giả

Alex Rodriguez

Alex Rodriguez là một kỹ sư phần mềm tại IBM chuyên về công nghệ Java và dịch vụ mạng REST. Ông đã lập trình bằng Java từ JDK 1.1.7B và chuyên về thiết kế và phát triển phần mềm Java dựa trên EE.