RESTful Web Services

Created & Presented by
Sangeeta Joshi

Agenda

1. Introduction to REST
2. Basic Design Principles
3. Common MIME Types

RESTful Web Services

- REST (Representational State Transfer) was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation.

- REST is an architectural style for designing distributed systems.

- It is not a standard but a set of constraints, such as
  - being stateless
  - having a client/server relationship
  - a uniform interface.

- REST is not strictly related to HTTP, but it is most commonly associated with it.

RESTful Web Services

- REST defines a set of architectural principles .

Design Web services that focus on a system's resources,

- including how resource states are addressed

&

- transferred over HTTP

by a wide range of clients written in different languages.

- REST has emerged in the last few years alone as a predominant Web service design model.
- In fact, REST has had such a large impact on the Web that it has mostly displaced SOAP- and WSDL-based interface design because it's a considerably simpler style to use.

RESTful Web Services

- to **minimize the coupling** between client and server components in a distributed application.

- if your server is going to be used by **many different clients** that you do not have control over.

- if you want to be able to **update the server regularly** without needing to update the client software.

RESTful Web Services

- Roy Fielding first introduced “REST” in his academic dissertation,
 "Architectural Styles and the Design of Network-based Software Architectures,“

- It analyzes a set of software architecture principles that use the Web as a platform for distributed computing

- REST didn't attract this much attention at that time.

- Now, years after its introduction, major frameworks for REST have started to appear

- and are still being developed because it's slated, for example, to become an integral part of Java™ 6 through JSR-311.

RESTful Web Services

A concrete implementation of a REST Web service follows four basic design principles:

- *Use HTTP methods explicitly.*
- *Be stateless.*
- *Expose directory structure-like URIs.*
- *Transfer XML, JavaScript Object Notation (JSON), or both.*

Use HTTP methods explicitly

- One of the key characteristics of RESTful Web service
- It establishes a one-to-one mapping between CRUD operations and HTTP methods:
  - To create a resource on the server, use ***POST.***
  - To retrieve a resource, use ***GET***.
  - To change state of resource or to update use ***PUT.***
  - To remove or delete a resource, use ***DELETE.***

A. Use HTTP methods explicitly

- POST /users

  HTTP/1.1 Host : myserver Content-Type: application/xml

  <?xml version="1.0"?> <user> <name>Robert</name> </user>

- GET /users/Robert

  HTTP/1.1 Host: myserver Accept: application/xml

- PUT /users/Robert HTTP/1.1

- DELETE /users/Robert

Use Nouns instead of Verbs

- Follow REST guidelines for using HTTP methods explicitly
- Use nouns in URIs instead of verbs
- In a RESTful Web service, the verbs—POST, GET, PUT, and DELETE—are already defined by the protocol
- To keep the interface generalized and to allow clients to be explicit about the operations they invoke, the Web service should not define more verbs or remote procedures, such as /adduser or /updateuser.
- This general design principle also applies to the body of an HTTP request, which is intended to be used ***to <u>transfer resource state</u>***, not to carry the name of a remote method or remote procedure to be invoked.

B. **Be stateless**

- A complete, independent request doesn't require the server, to retrieve any kind of application context or state.
- A REST Web service application (or client) includes all parameters, context, and data needed by server-side component to generate a response. (within the HTTP headers and body of a request)
- Statelessness improves Web service performance and simplifies the design and implementation of server-side components because the absence of state on the server removes the need to synchronize session data with an external application.

C. **Expose directory structure-like URIs**

- REST Web service URIs should be intuitive to the point where they are easy to guess.
- Think of a URI as a kind of self-documenting interface that requires little, if any, explanation or reference for a developer to understand what it points to and to derive related resources.
- Structure of a URI should be straightforward, predictable, and easily understood.

C. **Expose directory structure-like URIs**

- One way to achieve this level of usability is to define directory structure-like URIs.
- This type of URI is hierarchical, rooted at a single path, and branching from it are subpaths that expose the service's main areas.
- According to this definition, a URI is not merely a slash-delimited string, but rather a tree with subordinate and superordinate branches connected at nodes.
- http://www.myservice.org/discussion/topics/{topic}

C. Expose directory structure-like URIs

Some additional guidelines about URI structure for a RESTful Web service are:

- Hide the server-side scripting technology file extensions (.jsp, .php, .asp), if any, so you can port to something else without changing the URIs.
- Keep everything lowercase.
- Substitute spaces with hyphens or underscores
- Avoid query strings as much as you can.
- Instead of using the 404 Not Found code if request URI is for a partial path, always provide a default page or resource as a response.
- URIs should be static so that when the resource changes or the implementation of the service changes, the link stays the same.

D. Transfer XML, JSON, or both

- A resource representation typically reflects the current state of a resource, and its attributes, at the time a client application requests it. Resource representations in this sense are mere snapshots in time.

- The objects in your data model are usually related in some way, and the relationships between data model objects (resources) should be reflected in the way they are represented for transfer to a client application.

```
<?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>
```

Common MIME types used by RESTful services

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

- This allows the service to be used by a variety of clients written in different languages running on different platforms and devices.

- Using MIME types and the HTTP Accept header is a mechanism known as *content negotiation*, which lets clients choose which data format is right for them and minimizes data coupling between the service and the applications that use it.

RESTful Web Services

• REST is not always the right choice.

•It is a way to design Web services with less dependence on proprietary middleware (for example, an application server) than the SOAP- and WSDL-based kind.

•REST is a return to the Web the way it was before the age of the big application server, through its emphasis on the early Internet standards, URI and HTTP.

• XML over HTTP is a powerful interface that allows internal applications, such as Asynchronous JavaScript + XML (Ajax)-based custom user interfaces, to easily connect, address, and consume resources.

•The great fit between Ajax and REST has increased the amount of attention REST is getting these days.

•Exposing a system's resources through a RESTful API is a flexible way to provide different kinds of applications with data formatted in a standard way.

Any Questions?