About Jerome Versrynge

Introduction To REST Concepts

by Jerome Versrynge on October 25th, 2012 | **Filed in:** Software Development **Tags:** RESTful Web Services

Introduction

This post aims at demystifying the REST (Representational State Transfert) web design concepts. REST is based on a client server model. REST is a set of principles describing how standards can be used to develop web applications, for example. Its main purpose is to anticipate on common implementation issues and organize the relationship between logical clients and servers. You could call it a set of best practices!

In practice, REST provides guidance on how to implement web application interfaces to the web. Typically, one says a web application is constructed in a REST-like way or not. REST is often associated (or implemented) with HTTP, but it could be implemented with other technologies too. REST is platform and language independent.

Roy Fielding, the inventor of REST, says REST aims at achieving the following:

- *Generality Of Interfaces* – All web applications should implement their interfaces the same way. By sharing the same convention, other applications know how to call yours, and you know how to call theirs. Minimal learning curve for each new application.
- *Independent Deployment of Components* – Once an application and its REST interfaces have been implemented and deployed, one must be able to implement or re-implement, and deploy any REST interfaces without having to rewrite or modify existing ones.
- *Encapsulate Legacy Systems* – Existing applications which are not implemented in a REST-like way can be wrapped with REST

interfaces, making them REST-like applications.

- *Intermediary Components To Reduce Interaction Latency* – For example, in order to handle traffic, it is common to distribute user/client requests to several physical servers (which is not to be confused with logical servers). This is transparent for users. Since REST uses interfaces, implementing or adding extra layered components, such as physical servers to handle a peak of client requests, is easy.
- *Emphasizing The Scalability Of Component Interactions* – This is complementary to the previous point.
- *Enforce Security* – Exchanging information over the Internet can be risky. Hackers can use it to twist the system. REST principles eliminate many of those risks.

Concepts

- *Resource* – A logical resource is any concept (car, dog, user, invoice…) which can be addressed and referenced using a global identifier. Typically, each resource is accessible with a URI when implementing REST over HTTP (for example: http://www.mysite.com/invoice/34657).
- *Server* – A logical server is where resources are located, together with any corresponding data storage features. Such servers do not deal with end user interfaces (GUI).
- *Client* – Logical clients make *requests* to logical servers to perform operations on their resources. For example, a client can request the state of the resource, create a resource, update a resource, delete a resource, etc… Clients do not possess resources or corresponding data storage features. However, they deal with end user interfaces (GUI).
- *Request and Responses* – The interactions between client and servers is organized with requests from client to server, and responses to requests from server back to client. Requests can contain representations of the resource.
- *Representation* – A representation is a document representing the current status of a resource. It can also be the new desired status when a client makes a request to update a resource, for example.

Principles

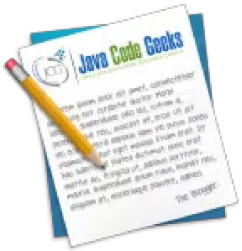
Here are some principles applicable in REST-like applications:

- *The state of a resource remains internal to the server, not the client* – The client can request it, or update it with requests made to the server.
- *No client context saved on the server between requests* – The server must not store the status of a client. Otherwise, this would break the scalability objective of REST when reaching a couple million users. Remember that requests can be distributed to several physical servers, which could cause physical resource consumption issues.
- *Client requests contain all information to service it* – No matter which request is sent by a client to a server, it must be complete enough for the server to process it.
- *Session states are stored on the client side* – If necessary, any information about the status of the commur between a logical server and a logical client must be held on the client side.
- *Multiple representations of a resource can coexist* – The chosen format used to represent the state of a resource in requests and responses is free (XML, JSON…). Multiple formats can be used.

Newsletter

Join our Newsletter to gain exclusive access to the latest news in the Java world, as well as insights about Android, Scala, Groovy and other related technologies. As an **extra bonus**, by joining you will get our **brand new e-books**, published by Java Code Geeks and their JCG partners for your reading pleasure!

Join Us



With **368,437** Aug unique visitors and over **500** authors we are placed among the top Java related sites around. Constantly being on the lookout for partners; we encourage you to join us. So If you have a blog with unique and interesting

- *Responses explicitly indicate their cacheability* – When a server returns a response to a request, the information it contains may or may not be cached by the client. If not, the client should make new requests to obtain the latest status of a resource, for example.
- *Code on Demand* – This is an optional feature in REST. Clients can fetch some extra code from the server to enrich their functionalities. An example is Javascript.

About session states, implementing a login logout (i.e., authentication) system between a physical server and a physical client requires saving session information on the server side. Otherwise, if it were saved on the client side, it could be hacked from the client side.

There is a general agreement that whatever 'resource' is required to implement authentication between the client and the server is considered out-of-scope for REST. These authentication resources do not have to follow REST principles (see here for more details).

REST Over HTTP

When implementing REST over HTTP, the logical REST client is typically a web browser and the logical REST server is a web server. The REST API (or service) must be hypertext driven.

About resource IDs:

- The preference is given to nouns rather than verbs to indicate the type of a resource (cat, dog, car…).
- The unique ID of a resource is a URI, for example: http://www.mysite.com/invoice/34657.
- A group of resources can also be accessed with a URI, for example: http://www.mysite.com/user/7723/invoices.

It is also considered good practice to use URIs in resource representations when a resource refers to another resource. For example, in a XML document representing a resource:

```
1 <dog self='www.mysite.com/dog/923' >
2     <name>Lassie</name>
3     <owner ref='www.mysite.com/owner/411' />
4 </dog>
```

In order to perform operations on resources, simple HTTP is used to make calls between machines. HTTP knows several types of calls: PUT, GET, POST, DELETE, HEAD, CONNECT, PATCH, TRACE and OPTIONS.

However, REST only uses four: PUT, GET, POST and DELETE.

- GET – Clients can request the status of a resource by making an HTTP GET request to the server, using the resource's URI. REST requires that this operation does not produce any side effect to the resource's status (nullipotent).
- PUT – Creates a new resource. Since the client does not know the next invoice number, the URI can be: http://www.mysite.com/invoice. If the resource is already created, it is not recreated. In other words, a REST PUT on http://www.mysite.com/invoice/841 (for example) is (and must be) idempotent. Invoice 841 must not be created multiple times if clients call that PUT several times.

- POST – REST requires POST client requests to update the corresponding resource with information provided by the client, or to create this resource if it does not exist. This operation is not idempotent.
- DELETE- This operation removes the resource forever. It is idempotent.

REM: Implementing a http://www.mysite.com/invoice/add URI is not considered a REST compliant pratice.

The format (JSON, XML…) used to return representations of resources is set in the media type of the server response (Multipurpose Internet Mail Extensions – MIME).

In order to handle success or errors issues, HTTP REST recommends using one of the HTTP Status Code.

Additional Read

- http://www.infoq.com/articles/rest-introduction

Reference: Introduction To REST Concepts from our JCG partner Jerome Versrynge at the Technical Notes blog.

You might also like:

- Introduction To JavaEE Concepts
- Introduction To Git Concepts
- How should REST services be documented?
- Why REST is so important
- ETags for REST with Spring

Related Whitepaper:

Share and enjoy!

Leave a Reply

Name (Required)

Mail (will not be published) (Required)

Website

2 + = five

Notify me of followup comments via e-mail

Sign me up for the newsletter!

Submit Comment

Knowledge Base

- Examples
- Resources
- Tutorials
- Whitepapers

Partners

- Mkyong

The Code Geeks Network

- Java Code Geeks
- .NET Code Geeks
- Web Code Geeks

Hall Of Fame

- “Android Full Application Tutorial” series
- GWT 2 Spring 3 JPA 2 Hibernate 3.5 Tutorial
- Android Game Development Tutorials
- Android Google Maps Tutorial
- Android Location Based Services Application – GPS location
- Funny Source Code Comments
- Java Best Practices – Vector vs ArrayList vs HashSet
- Android JSON Parsing with Gson Tutorial
- Android Quick Preferences Tutorial

About Java Code Geeks

JCGs (Java Code Geeks) is an independent online community focused on creating the ultimate Java to Java developers resource center; targeted at the technical architect, technical team lead (senior developer), project manager and junior developers alike. JCGs serve the Java, SOA, Agile and Telecom communities with daily news written by domain experts, articles, tutorials, reviews, announcements, code snippets and open source projects.

License

Java Code Geeks content is offered under Creative Commons Attribution-ShareAlike 3.0 Unported License. For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page. Any of the above conditions can be waived if you get written permission from Java Code Geeks. Nothing in this license impairs or restricts the author's moral rights.

Email Format

- html
- text

Subscribe