FEATURED / REST

NEXT STORY
Windows process impersonation using RunAs, Windows APIs, and psexec

PREVIOUS STORY
Debugging a Java application... Remotely!

RECENT POSTS

Determine the "Readability" of a text with Python
23 JAN, 2019

Awesome APIs with Kotlin, Spring 5 and Swagger
3 FEB, 2018

dotenv for Java, Kotlin and the JVM
10 JAN, 2018

Node.js / Express app generator: with ES.next, API validation, documentation, logging, and more
11 APR, 2017

RESTful Design Principles

BY CDIMASCIO · SEPTEMBER 6, 2013

Here, we will outline the set of RESTful design principles that should be adhered to when creating a 'proper' RESTful service.

Let's start with the basics. **What is REST**?

REST = REpresentational State Transfer. REST is an architectural style for network based software that requires stateless, cacheable, client-server communication via a uniform interface between components.

The primary focus of this blog post is to introduce REST along with REST terminology, REST concepts, and some simple examples describing what REST looks like in practice. As a secondary

ARCHIVES

- January 2019
- February 2018
- January 2018
- April 2017
- January 2017
- February 2016
- January 2016
- September 2015
- June 2015
- May 2015
- March 2015
- February 2015
- January 2015
- November 2014
- August 2014
- July 2014
- June 2014
- May 2014
- April 2014
- March 2014
- February 2014

focus, I will address a topic that often confuses folks. Many folks often ask to compare REST vs SOAP. This comparison *does not* make sense. REST is an architectural style, while SOAP, like HTTP, is communication protocol. What does this mean? Well, REST and SOAP are not mutually exclusive. In theory, they can be used together. I would highly recommend against this usage, but their is nothing about the REST style that prohibits this case. During this post we'll touch on this. We'll also see why REST is so widely used over HTTP. All in all, the primary focus will remain an introduction to REST!

Let's get started. First off, I introduced a number of loaded terms in my one line description of REST. These terms were *stateless*, *cacheable*, *client-server* communication, and *uniform interface*. These represent the basic principles of REST. Let's briefly introduce these principles and their meaning within the context of REST.

Basic Principles

Client-Server Communication

Client-server architectures have a very distinct separation of concerns. All applications built in the RESTful style must also be client-server in princple.

Stateless

January 2014

December 2013

November 2013

October 2013

September 2013

August 2013

July 2013

DATABASE / DERBY
Quick Start with Apache Derby
28 JUL, 2013

DATABASE / JAVASCRIPT / REST
Create a REST API with Node.js, Express, and MongoDB
5 APR, 2014

SCALA / UNIT TESTING
Testing with Specs2 and Play ReactiveMongo
18 JAN, 2015

FEATURED / JAVA
Named Pipes with Java
11 JAN, 2014

Each each client request to the server requires that its state be fully represented. The server must be able to completely understand the client request without using any server context or server session state. It follows that all state must be kept on the client. We will discuss stateless representation in more detail later.

Cacheable

Cache constraints may be used, thus enabling response data to to be marked as cacheable or not-cachable. Any data marked as cacheable may be reused as the response to the same subsequent request.

Uniform Interface

All components must interact through a single uniform interface. Because all component interaction occurs via this interface, interaction with different services is very simple. The interface is the same! This also means that implementation changes can be made in isolation. Such changes, will not affect fundamental component interaction because the uniform interface is always unchanged. One disadvantage is that you are stuck with the interface. If an optimization could be provided to a specific service by changing the interface, you are out of luck as REST prohibits this. On the bright side, however, REST is

optimized for the web, hence incredible popularity of REST over HTTP!

The above concepts represent defining characteristics of REST and differentiate the REST architecture from other architectures like web services. It is useful to note that a REST service is a web service, but a web service is not necessarily a REST service.

Let's now dive into a bit more detail and discuss a variety of elements used to compose a RESTful system.

Resource and Resource Identifier:

A key abstraction of REST is the *resource*. A *resource* can be just about anything. It can be a document or an image, an object, a collection of other *resources*, and more. A *resource* is identify by its *resource identifier*. The *resource identifier* is often used when
multiple components communicate with one another. They are able to reference specific *resources* using the *resource identifier*.

In practice, resources are *nouns*. Resources and identified by URIs
e.g. This Car *resource* is identified by the *resource identifier*,
http://www.automart.com/cars/12345

Representation:

Components perform actions a *resource* by applying operation provided by the component's uniform interface.
A resource is represented by its current state or its intended state (assuming the action will modify the resource in some way). This representation includes a sequence of bytes and some description of those bytes. The format of a representation is defined as its media type.

In practice, *resources* are often represented as XML, JSON, RDF, and more

Basic Principles in Practice

Let's harken back to the Basic Principles section and describe how those principles can be applied in practice.

Client-server

HTTP is a client-server protocol. Why not use it with REST. Check!

Uniform Interface

REST is optimized for the web, thus HTTP is typically used. HTTP defines GET,POST, PUT, DELETE. Woah! That meets REST's requirement to provide a *uniform interface* for components.

Cacheable

HTTP provides a cache control mechanism. See here. Dang! HTTP just filled another REST requirement

Stateless

Hmm. Not quite so easy. Let's apply some rules to the uniform interface provided by HTTP

`GET` – Safe, Cacheable, Idempotent
`PUT` – Idempotent
`DELETE` – Idempotent
`HEAD` – Safe, Idempotent
`POST` – n/a

Cool! But what does that mean?

– Safe – the operation must not have side effects
– Cacheable – the result may be cached e.g. by a proxy server
– Idempotent – The operation must always return the same result

Check!

Rest Practical Usage

Let's now provide some example of in-practice usage:

Resource and Resource Identifiers

Example of resources are Car, Engine, Part. Each resource is identified by its *resource identifier*. For example:

Car:
http://www.automart.com/cars/12345

Part:
http://www.automart.com/part/12345

Part:
http://www.automart.com/engine/12345

Representation
Our Car with *resource identifier,*
http://www.automart.com/cars/12343
can be manipulated by a component via a uniform interface of GET, POST, PUT, DELETE.
Here are some examples:

GET returns a representation of a resource's state, For example, http://www.automart.com/cars/12343. An XML representation of that state might be:

[code language="xml"]

<Car>
<Make>Audi</Make>
<Model>A5</Model>
<Year>2013</Year>
</Car>

[/code]

or in JSON

[code language="javascript"]
{
"Make" : "Audi",
"Model" : "A5",

"Year" : 2013
}
[/code]

Representation with Linked Resources
Resource representations may contain links to other resources
e.g.

[code language="xml"]

<Car>
…
<Engine
uri="http://www.automart.com/engine/1242"/>
…
</Car>

[/code]

or in JSON

[code language="javascript"]
{
…
"engine" :
"http://www.automart.com/engine/1242"
…
}
[/code]

That's it!!
Thanks!

Tags: JAX-RS JSON REST RESTful XML

YOU MAY ALSO LIKE...

Create a REST API with Node.js, Express, and MongoDB

APRIL 5, 2014

Node.js / Express app generator: with ES.next, API validation, documentation, logging, and more

APRIL 11, 2017

Named Pipes with Java

JANUARY 11, 2014

10 RESPONSES

Comments 1 Pingbacks 9

SutoCom September 8, 2013 at 12:41 pm

Reblogged this on Sutoprise Avenue, A SutoCom Source.

Reply

LEAVE A REPLY

Comment

Name * **Email** *

Website

Post Comment