Representational State Transfer (REST)

Representational State Transfer is an architectural style that build on certain principles using the current web fundamentals. It generally runs on HTTP. It makes a stateless transfer. REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements. REST has been applied to describe desired web architecture, to identify existing problems, to compare alternative solutions, and to ensure that protocol extensions would not violate the core constraints that make the Web successful. Fielding used REST to design HTTP 1.1 and Uniform Resource Identifiers (URI).

The REST architectural style is also applied to the development of web services as an alternative to other distributed communication types such like SOAP. REST is often used in mobile applications, social networking Web sites, mashup tools and automated business processes.

There are 5 basic fundamentals of REST services which are created for the Web.

1. Everything is a Resource.
2. Every Resource is Identified by a Unique Identifier.
3. Use Simple and Uniform Interfaces.
4. Communication is Done by Representation.
5. Every Request is Stateless.

Everything is a Resource

The first important point in REST is to think in terms of resources rather than physical files. You access the resources over some URI. for example:

- http://www.mysite.com/pictures/logo.png - Image Resource;
- http://www.mysite.com/index.html - Static Resource;
- http://www.mysite.com/Customer/1001 - Dynamic Resource returning XML or JSON content;
- http://www.mysite.com/Customer/1001/Picture - Dynamic Resource returning an image.

Unique Identifier

Informations are reached by using unique identifiers. In REST, we add one more constraint to the current URI: in fact, every URI should uniquely represent every item of the data collection. For example you can see below unique URI format for each customer is defined.

http://www.mysite.com/Customer/dupont

http://www.mysite.com/Customer/smith

and orders of customer ” dupont ” is defined like following:

http://www.mysite.com/Customer/dupont/Orders

Simple and Uniform Interfaces

To request and send data to those resources some HTTP methods are used. These are the HTTP methods :

GET - List the members of the collection (one or several)

PUT - Update a member of the collection

POST - Create a new entry in the collection

DELETE - Delete a member of the collection

Then, at URI level, you can define the type of collection, like this:

http://www.mysite.com/Customer to identify the customers or

http://www.mysite.com/Customer/1234/Orders to access a given order.

This combination of HTTP method and URI replace a list of English-based methods, like GetCustomer / InsertCustomer / UpdateOrder / RemoveOrder.

Representations

What you are sending via the network is actually a representation of the actual resource data.

The main representation schemes are XML and JSON but it could be CVS as well.

For example, here is how a customer data is retrieved with a GET method:

An example with XML format :

```
<Customer>
<ID>1234</ID>
<Name>Dupond</Name>
<Address>Tree street</Address>
</Customer>
```

Below is a simple JSON snippet for creating a new customer record with name and address:

```
{Customer: {"Name":"Dupont", "Address":"Tree street"}}
```

As a result to this data transmitted with a POST command, the RESTful server will return the just-created ID.

Clarity of this data format is one of the reasons why preferably to use JSON format instead of XML or any format.

Stateless

In REST concept every request is independent and each request doesn’t have any information about the previous request. Every request should be an independent request so that we can enhance the requests being sent. As REST doesn’t use much memory like SOAP so that we can make more independent requests. This Scalability in terms is important because a server request is unaware of any state we store and makes resource management easier.

Of course, there are some disadvantages of being stateless. Client has to add all the necessary information in request' which increases network traffic. It also highlights consistency in the behavior of the application server which could be handled difficultly because there may be many different Clients, the requests may come from the different content.

Independent request means that each request doesn’t carry out any state information about another. A classic example of statelessness is the use of the HTTP protocol. HTTP protocol does not carry out data between requests so that this is usually achieved by using some kind of server variables to carry out the data using some programming languages. For example in Asp.Net it can be achieved by using variables such as viewstate, session, caching, query string vice versa.

There is one more important fact that need to be outlined: **Cacheable**. HTTP response by the client "cache" can be entertainment, so it sent Server RESPONSE to indicate whether the case is cacheable, it is important in terms of performance.

What are the advantages of rest;

- It is lightweight, can be easily extended.
- Inbound, outbound data size is very small.

- It is easy to design and easy implementation, it does not need any extra tools.
- Works over HTTP, platform independent.

REST and RESTful

Representational state transfer (REST) is a style of software architecture. As described in a dissertation by Roy Fielding, REST is an "architectural style" that basically exploits the existing technology and protocols of the Web.

RESTful is typically used to refer to web services implementing such an architecture. Some says that "REST" is an architectural paradigm. "RESTful" describes using that paradigm. More precisely the term Representational State Transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation. Conforming to the REST constraints is referred to as being ‘RESTful’.

Prev | Up | Next
Chapter 5. Architectures for Enterprise Level | Home | Portal architecture - one of the SOA variants