Learn REST: A Tutorial

A fast-training course for **REST** - *Re*presentational *S*tate *T*ransfer, a new approach to systems architecture and a lightweight alternative to web services

1. What is REST?

REST stands for **Re**presentational **S**tate **T**ransfer. (It is sometimes spelled "ReST".) It relies on a stateless, client-server, cacheable communications protocol -- and in virtually all cases, the HTTP protocol is used.

REST is *an architecture style* for designing networked applications. The idea is that, rather than using complex mechanisms such as CORBA, RPC or SOAP to connect between machines, simple HTTP is used to make calls between machines.

- In many ways, the World Wide Web itself, based on HTTP, can be viewed as a REST-based architecture.

RESTful applications use HTTP requests to post data (create and/or update), read data (e.g., make queries), and delete data. Thus, REST uses HTTP for all four CRUD (Create/Read/Update/Delete) operations.

REST is a lightweight alternative to mechanisms like RPC (Remote Procedure Calls) and Web Services (SOAP, WSDL, et al.). Later, we will see how much more simple REST is.

- Despite being simple, REST is fully-featured; there's basically nothing you can do in Web Services that can't be done with a RESTful architecture.

REST is not a "standard". There will never be a W3C recommendataion for REST, for example. And while there are REST programming frameworks, working with REST is so simple that you can often "roll your own" with standard

TABLE OF CONTENTS

YOUR HOST

DR. M. ELKSTEIN

A software engineering expert with years of hands-on experience. I have developed dozens of web sites, and have taught numerous programming and computer science courses in academia and in the industry. I provide consultancy services, and my clients range from small

library features in languages like Perl, Java, or C#.

BY DR. M. ELKSTEIN 47 COMMENTS

2. REST as Lightweight Web Services

As a programming approach, REST is a lightweight alternative to Web Services and RPC.

Much like Web Services, a REST service is:

- Platform-independent (you don't care if the server is Unix, the client is a Mac, or anything else),
- Language-independent (C# can talk to Java, etc.),
- Standards-based (runs on top of HTTP), and
- Can easily be used in the presence of firewalls.

Like Web Services, REST offers no built-in security features, encryption, session management, QoS guarantees, etc. But also as with Web Services, these can be added by building on top of HTTP:

- For security, username/password tokens are often used.
- For encryption, REST can be used on top of HTTPS (secure sockets).
- ... etc.

One thing that is *not* part of a good REST design is cookies: The "ST" in "RE**ST**" stands for "State Transfer", and indeed, in a good REST design operations are self-contained, and each request carries with it (transfers) all the information (state) that the server needs in order to complete it.

BY DR. M. ELKSTEIN 10 COMMENTS

3. How Simple is REST?

Let's take a simple web service as an example: querying a phonebook application for the details of a given user. All we have is the user's ID.

Using Web Services and SOAP, the request would look something like this:

```
<?xml version="1.0"?>
<soap:Envelope
```

startups to the largest banks in my country.

VIEW MY COMPLETE PROFILE

```
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-enc
oding">
 <soap:body pb="http://www.acme.com/phonebook">
  <pb:GetUserDetails>
   <pb:UserID>12345</pb:UserID>
  </pb:GetUserDetails>
 </soap:Body>
</soap:Envelope>
```

(The details are not important; this is just an example.) The entire shebang now has to be sent (using an HTTP POST request) to the server. The result is probably an XML file, but it will be embedded, as the "payload", inside a SOAP response envelope.

And with REST? The query will probably look like this:

```
http://www.acme.com/phonebook/UserDetails/12345
```

Note that this isn't the request body -- it's just a URL. This URL is sent to the server using a simpler GET request, and the HTTP reply is the raw result data -- not embedded inside anything, just the data you need in a way you can directly use.

- It's easy to see why Web Services are often used with libraries that create the SOAP/HTTP request and send it over, and then parse the SOAP response.
- With REST, a simple network connection is all you need. You can even test the API directly, using your browser.
- Still, REST libraries (for simplifying things) do exist, and we will discuss some of these later.

Note how the URL's "method" part is not called "GetUserDetails", but simply "UserDetails". It is a common convention in REST design to use *nouns* rather than *verbs* to denote simple *resources*.

The letter analogy

A nice analogy for REST vs. SOAP is mailing a letter: with SOAP, you're using an envelope; with REST, it's a postcard. Postcards are easier to handle (by the receiver), waste less paper (i.e., consume less bandwidth), and have a short content. (Of course, REST requests aren't really limited in length, esp. if they use POST rather than GET.)

But don't carry the analogy too far: unlike letters-vs.-

postcards, REST is every bit as secure as SOAP. In particular, REST can be carried over secure sockets (using the HTTPS protocol), and content can be encrypted using any mechanism you see fit. Without encryption, REST and SOAP are both insecure; with proper encryption in place, both are equally secure.

BY DR. M. ELKSTEIN 21 COMMENTS

4. More Complex REST Requests

The previous section included a simple example for a REST request -- with a single parameter.

REST can easily handle more complex requests, including multiple parameters. In most cases, you'll just use HTTP GET parameters in the URL.

For example:

```
http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe
```

If you need to pass long parameters, or binary ones, you'd normally use HTTP POST requests, and include the parameters in the POST body.

As a rule, GET requests should be for read-only queries; they should not change the state of the server and its data. For creation, updating, and deleting data, use POST requests. (POST can also be used for read-only queries, as noted above, when complex parameters are required.)

- In a way, this web page (like most others) can be viewed as offering services via a REST API; you use a GET request to read data, and a POST request to post a comment -- where more and longer parameters are required.

While REST services might use XML in their *responses* (as one way of organizing structured data), REST *requests* rarely use XML. As shown above, in most cases, request parameters are simple, and there is no need for the overhead of XML.

- One advantage of using XML is type safety. However, in a stateless system like REST, you should *always*

verify the validity of your input, XML or otherwise!

BY DR. M. ELKSTEIN 39 COMMENTS

Home Continues...

Subscribe to: Posts (Atom)