BDH Class Notes - Web Services

venkatesh V
venakteshv@cmi.ac.in

March 2020

1 Introduction

Having discussed the advances in data storage and computation, we now turn our attention to building useful big data applications. Towards this purpose, it is insufficient to understand how to manage big data as we have learned so far using distributed computing platforms such as Hadoop HDFS, programming models such as map-reduce, and data stores such as NoSQL. The missing piece is about handling interoperability of applications and distributed systems. We leverage the advances in the area of web technologies to build services that expose application features built over heterogeneous distributed systems. In this article, we discuss the design and implementation of web services.

2 Interoperability of Distributed Sytems

Different distributed systems present interesting challenges of interoperability. The systems may be heterogeneous by nature. The heterogeneity might be driven not only by the nature of computing systems and programming languages/scripts/models used but also the data representations used within each system. For example, let us say, we would like to use both the google maps functionality and Facebook's face recognition technology. In an ideal world, both these companies expose their features as "services" that any developer may use in building his own application. In the present era, our applications depend on large and complex heterogeneous distributed systems.

3 A Brief Survey of Interoperability Solutions

We may consider file passing as a simple means of communication between two distributed systems. However, this has several limitations. For instance, we need to design common file formats and semantics of serializing the data into a file. Moreover, the encryption, compression, encoding and transmission of the file are concerns that every system developer needs to solve. Hence, a middleware platform will be very useful from separations of concerns and a

reuse perspective. By separation of concerns, we mean the application developer can worry about the application details without worrying about the "big data" management details. The "big data" related features can be transferred to the middleware community.

Instead of transferring files, transferring objects would solve the impedance mismatch problem. Thus, passing objects will only be more natural for client-server communication. Some technologies use remote procedure calls. In other words, they invoke methods on a remote object. Another architectural decision passes "messages".

Object Management Group's (OMG) Common Object Broker Architecture (CORBA) is an open standards based solution proposed to meet this objective. CORBA is a standards-based, vendor neutral and language agnostic solution for interoperability. A server makes objects available for clients to use through a middleware layer named Object Request Broker (ORB). Client requests for an object (served by the server) through the ORB API. The ORB API supplies the object instance after communicating and obtaining that instance from the server often over a network. Notice that ORBs make the sharing of objects seamless and easy. Given the standard architectural specification, vendor-specific code could be provided by the specific programming language community.

Soon, vendor specific implementations were out in the market. Microsoft implemented DCOM. Sun Microsystems came up with RMI. In the early 2000's, DCOM and RMI were popular. The central idea remains the same across both DCOM and RMI. The Remote Method Invocation (RMI) model used RMI registry to advertise the services. First, the client "discovers" the service it needs. Then, it directly talks to the RMI server to "invoke" that service which is essentially a method call on an object. DCOM supports remote objects by running on a protocol called Object Remote Procedure Call (ORPC). A DCOM client calls the exposed methods of a DCOM server by acquiring a pointer to one of the server object's interfaces. In this course, we will not go any deeper into OMG, DCOM or RMI. Instead, we focus on building web services which has revolutionized the industry, of-late.

4 Web and its Disruptions

With the broad penetration of internet, it made sense to make method invocations over the web. This led to the idea of "web services".

Early static web was developed in 1990 at European Organization for Nuclear Research (CERN). NCSA Mosaic 1.0 was the first browser, released by the National Center for Supercomputer Applications (NCSA). So, how does it work? First, we need to write code in the Hyper Text Mark Up (HTML) language. Then, we move it to a machine which has a software module named web server running on it. Web servers wait for client requests, usually sent as Hyper Text Transfer Protocol (HTTP) requests. In return, the server sends the HTML back to the client. The client then renders the HTML on the screen.

Static web pages were extremely limiting our abilities to implement online

transactions, maintain sessions, remember data over sessions and so on. This called for a more powerful technology which could render HTML output dynamically (i.e., programmatically). Apache Httpd 1.0 web server allowed Common Gateway Interface (CGI) which enabled the dynamic web. Microsoft proposed Active Server Pages (ASP). The details of coding with CGI or ASP is beyond the scope of this course. However, I encourage you to spend few minutes looking at some sample ASP and CGI scripts.

A web server's responsibility is to receive requests for web pages and serve them. Soon, technologies evolved to do more complex server-side processing. Full-stack applications could be coded on the server end. Such servers came to be known as Application Servers (also called App Servers). The technologies in this space went through a churn from being together in a single module to an architecture where a web server still receives web requests and forwards them to another app server which in-turn may work with several other app servers to complete the task. Needless to mention that availability of cloud data stores greatly improved our ability to solve bigger and more interesting business problems.

5 Web Services

A "service" is a software component provided through an (often, network-accessible) endpoint. Service consumer and provider use messages to exchange invocation request and response information in the form of self-containing documents. A *Web service* is an application that exposes certain application features (services) over the web. They are used for application-to-application integration.

A Web Resource is a named object that is accessible over the web. Uniform Resource Identifiers (URI), are used to identify them. A web page is an example of a web resource. An Uniform Resource Locator (URL), say http://www.google.com locates the webpage. In the context of web pages, URLs and URIs are used interchangeably. This is because, in this context, a resource is identified by its location. However, unique ids such as ISBN strings of books may serve as URIs.

5.1 Application Interoperability with Web Services

Let us assume that we need to create an application where we need to show meanings of words. We do not need to re-implement a dictionary. Instead, we may just use an existing dictionary. Oxford Dictionary offers a web service through which we can easily access word senses. The URL https://od-api.oxforddictionaries.com/api/v2/entries/en-us/ubiquitous returns a Javascript Object Notation (JSON) document containing the definitions. Part of the response containing the definition is shown below.

```
{
```

```
  "definitions": [
        "present, appearing, or found everywhere" ]
}
```

You may use these service to build your own applications.

5.2 REST API

The example of Oxford Dictionary discussed above is an example of REST API. Representational State Transfer (REST) was proposed by Roy Fielding in the year 2000. The central idea is to identify resources, and then use a representation of the resource that can be transferred over the web.

Each word is seen as a web resource that is represented using JSON format where we capture all data elements describing the term such as the definitions, usage and synonyms.

We use Hypertext Transfer Protocol (HTTP) methods to transfer the response. By default, when nothing is specified, we assume the HTTP Get method is invoked when we access the URL. In this document, we use the notation HTTP GET /ubiquitous to represent the firing of URL http://od-api.oxforddictionaries.com/api/v2/entries/en-us/ubiquitous over the web. The server returns the JSON response as a result.

REST allows the Create/Retrieve/Update/Delete (CRUD) operations through the HTTP methods. The HTTP Get method is mapped to the retrieval action. Instead, if you intend to create a new resource, you perform HTTP Post and send the representation of the resource along with the post request. For instance, HTTP POST /ubiquitous would create a new instance of the resource on the server. HTTP Put is used to update a resource. If you wish to add a new usage of the term ubiquitous, you would create a new representation which contains the update and submit as a HTTP Put request. Similarly, you may use HTTP Delete to remove the resource from the server. Thus you could manage the states of the resource through transferring its representation.

An idempotent HTTP method is an HTTP method that can be called many times without different outcomes. Notice that Get, Put and Delete are idempotent. Irrespective of how many ever times you invoke these methods, you get the same response from the server. Invoking Post multiple times might (not necessarily, depends on application design) create multiple instances of the resource.

To understand this better, refer to the example discussed at the restfulapi.net[1]. In our word sense case, ubiquitous is a singleton. However, in reality, we could have different requirements. The example at this website discusses creating several devices and configurations through REST service. HTTP POST /devices creates a new device. HTTP GET /devices/id will retrieve it.

A HTTP response code is also sent along with each response. For instance, a 500 error code means that there was some internal server error while serving the request. In fact, all 5xx errors indicate server failures. Similarly, 4xx errors

[1] https://restfulapi.net/rest-api-design-tutorial-with-example/

indicate that the request was not well formed. For more details on HTTP response codes, refer restfulapi.net[2].

6 Designing REST API

There are four major steps in the designing of REST APIs:

- Identify the object model
- Create Model URIs
- Determine Representations
- Assign HTTP Methods

6.1 Identifying Object Model

Assuming that we are designing the REST API for an ecommerce application. Say, we have several products to sell. These "Products" become a candidate for resource. Similarly, we need to identify the other objects, say users, reviews, etc. Coming with an object model is the subject of class diagram design in UML.

6.2 Create Model URIs

Next, we are interested in creating the model URIs. HTTP GET /product/id retrieves a product. To create a product, we do a HTTP Post /product. Remember, we submit an XML or a JSON representation of the resource along with the post request. In our case, we need to design the representation and that is our third step. Often, we include version numbers in our URL for maintenance reasons. So, you will see URLs such as https://od-api.oxforddictionaries.com/api/v2/entries/en-us/ubiquitous. Since we are concerned about resources, the URIs always contain nouns and never verbs.

6.3 Determine Representations

To create or update a resource, client sends the resource representation, usually (not necessarily) in XML or JSON format. For a product, the representation may contain details such as name, price, description and so on. Here is an example:

```
<product xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <name type="xs:string">N95 Mask</name>
  <price type="xs:int">100</price>
  <description type="xs:string">
    98% 3-layer filtration of pollution
```

[2]https://restfulapi.net/http-status-codes/

```
    particles PM2.5 99% filtration of
    bacteria, tested by Nelson Laboratories
  </description>
</product>
```

For a JSON example, refer the restfulapi website[3].

6.4 Assign HTTP Methods

The last step is to assign the http methods for the resource actions. To add products in the server, we use Post method. To update, we use Put and to retrieve, we use Get method. Since our products are uniquely identified by name, we could have a Get method such as HTTP GET /products/n95nelsonmask. This URI could map to an URL http://mywebsite/products/n95nelsonmask. On the other hand, if we model users as non-singleton, we will use HTTP GET /users/id to retrieve a specific user resource. An interesting fact to notice here is that the interaction between client and server is stateless. By stateless, we mean that the server need not store any information at its end to use across successive client requests. It is the client's responsibility to send all the necessary information along with each request. This design helps in keeping the server design lean (demanding less memory) and simple.

6.5 Implementing REST API

Java^{TM} API for RESTful Web Services (JAX-RS) delivers API for RESTful Web Services development in Java SE and Java EE. JAX-RS is implemented in Jersey. It provides all the required libraries to implement REST API. The annotations based implementation simplifies the web services development. You are encouraged to read the Jersey documentation[4] to learn web services implementation with Jersey.

7 REST API in the Real World

It is easy to spot REST api in the real world. They have in fact become a defacto standard for web services implementation. The developer tools in chrome browser lets you inspect the URIs and responses. Figure 1 shows an example from the google news website.

8 Summary

Interoperability of applications is central to the success of big data systems. Web Services play a key role in language-neutral, vendor-neutral application

[3] https://restful-api-design.readthedocs.io/en/latest/resources.html
[4] https://eclipse-ee4j.github.io/jersey.github.io/documentation/latest/getting-started.html

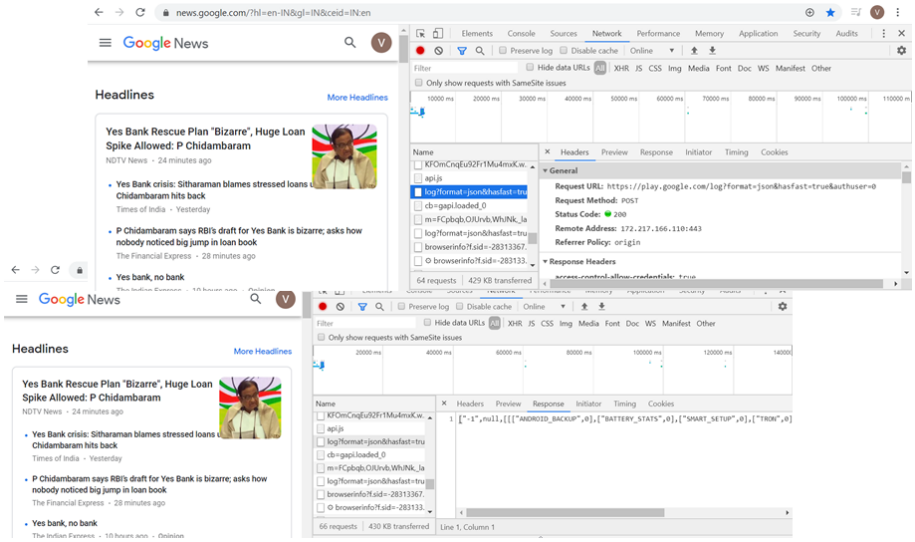

Figure 1: A snapshot of REST API usage in Google News.

interoperability. In this lecture, we discussed REST API for implementing web services. They can be easily implemented using Jersey-like libraries.