**Restful Web Services**

* REST is about resources and how to represent resources in different ways.
* REST is about client-server communication.
* REST is about how to manipulate resources.
* REST offers a simple, interoperable and flexible way of writing web services that can be very different from other techniques.
* Comes from Roy Fielding’s Thesis study.
* **Java support for REST**
* Java has provided JAX-RS API to develop RESTFUL services. Initially it has released jaxrs 1.1 api.
* Later on it revised and released one more api jaxrs 2.0. for which there are several implementations like **Jersey** (SUN/ORACLE) and **Rest Easy** (Jboss).

REST stands for representation state transfer. It is a new Architectural style that defines set of rules that can be applied on distributed system, which will lead to interoperable distributed applications.

REST architecture has been highly influenced from web architecture, Roy Fielding one of the principle authors of HTTP specification and co-founder of the Apache HTTP server project, as part of his doctorial thesis has defined several architecture principles that defines REST.

**REST is NOT!**

* A protocol.
* A standard.
* A replacement for SOAP.

**REST**

* Representational State Transfer
* Architectural style (technically not a standard)
* Idea: a network of web pages where the client progresses through an application by selecting links
* When client traverses link, accesses new resource (i.e., transfers state)
* Uses existing standards, e.g., HTTP
* REST is an architecture all about the Client-Server communication.

**An Architectural Style**

* REST is the architecture of the Web as it works today and, so it is already used in the web!
* It is an software architectural model which is used to describe distributed systems like WWW (World Wide Web).
* It has been developed in parallel with HTTP protocol.

**REST Characteristics**

* Resources: Application state and functionality are abstracted into resources.
* URI: Every resource is uniquely addressable using URIs.
* Uniform Interface: All resources share a uniform interface for the transfer of state between client and resource, consisting of
* Methods: Use only HTTP methods such as GET, PUT, POST, DELETE, HEAD
* Representation
* Protocol (The constraints and the principles)
* Client-Server
* Stateless
* Cacheable
* Layered

**Principles**

1. **Addressable resource:** Every resource has to have a unique directly addressable URI so that we can reach directly. To make integration efforts much easy it is very important to have a direct reachable address. If you see in an **Jee** most of the distributed applications will not have direct addressable URI, like **Ejb** or **Web Services**, due to which other applications has to understand the specific access technique to integrate.
2. **Uniform, Constrained Interfaces:** It is about exposing your service over a finite set of operations of the application protocols. In case of a Web Service if you have to access an operation on a Web service, we need to send an extra parameter called **“action”** or **“saopaction”.** But if you rely on application protocols methods you don’t need to use a extra **“action**” or **“soapaction”** in case of HTTP it has fixed set of methods with predefined meaning as follows.

**HTTP Methods**

* + GET – safe, idempotent, cacheable
  + PUT - idempotent
  + POST
  + DELETE - idempotent
  + HEAD
  + OPTIONS

1. **GET:** GET is a read-only operation. It is used for query the server for some information. It is safe and idempotent method.
2. **PUT:** PUT is modeled as an insert or update. It is also idempotent, as client will always send the identity of the resource he is trying to create/modify on the server.
3. **POST:** POST is modeled as an insert or update. It is non-idempotent and non-safe as client will send the identity of the resource, sending POST multiple times will a new resource every time.
4. **HEAD:** It is exactly like GET except that instead of returning the response body.
5. **DELETE:** It is used for removing the resource on the server.
6. **OPTIONS:** OPTIONS is used to request information about communication options of the resource you are interested in.

**Constrained interface has many more advantages like below**

**Familiarity:** If you have an URI to access the resource, you know which methods are exactly there on that resource. You don’t need an **IDL** or **WSDL** to know the information about the service.

**Interoperability:** HTTP is universal protocol. Most of the programming language have support for HTTP protocol. This indicates if your service is built on HTTP it will be interoperable by just sending the data in the format requested by the service. We don’t need any additional standards like **SOAP** or **WS-\*** standards.

**Scalability:** As the operations are fixed, we can predict the behavior of those method, let’s day a GET always returns the information of the resource, it never modifies the resource. This indicate if we access the resource and for the next access to the resource if there is not **PUT** or **POST** or **DELETE** requests in-between we don’t need to send the request to the server rather we can cache and return the information from client.

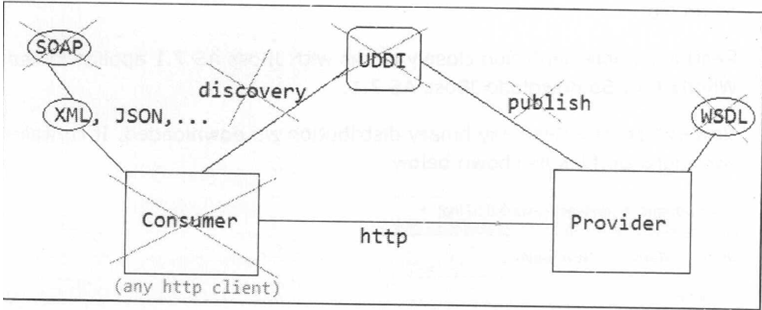
**Representation Oriented:** Different clients wants different representation of the resource, let’s say a JavaScript client want the resource to present in **JSON** format. A typical java application may want the resource to be presented in **XML**. In which representation we requested the server will present the resource in that specific representation.

**Communicate Stateless:** Nothing will be stored on the server to remember the client state. Everything will be stored at the client level only. This makes application highly scalable

**HATEOAS:** Hypermedia as the engine of application state. The resource as part of response returned to the client will even send more hyperlink indicate, what is the nest course of action you can do with the resource. This elimates the need of **WSDL** or **IDL** describing the resource, which means with no prior knowledge of the resource we should be able to interact with the service

**Architecture**

**JAX-RS API** has been designed completely based on http protocol. So we cannot build REST resource on any other protocol apart from HTTP.



**Status Codes**

HTTP status codes returned in the response header

**200 OK** – The resource was read, updated or deleted

**201 Created** – The resource was created

**400 Bad Request** – The data sent in the request was bad

**403 Not Authorized** – The principal named in the request was not authorized to perform this action

**404 Not Found** – The resource does not exist

**409 Conflict** – A duplicate resource cloud not be created

**500 Internal Service Error** – A service error occurred

**Restful Services Annotations**

**@Path --** It identifies the URI path. It can be specified on class or method.

**@Pathparam --** Itrepresents the parameter of the URI path.

**@GET --** It specifies method responds to GET request.

**@POST --** Itspecifies method responds to POST request.

**@PUT --** It specifies method responds to PUT request.

**@HEAD --** Itspecifies method responds to HEAD request.

**@DELETE --** specifies method responds to DELETE request.

**@OPTIONS --** It specifies method responds to OPTIONS request.

**@FormParam --** It represents the parameter of the form.

**@QueryParam --** It represents the parameter of the query string of an URL.

**@HeaderParam --** It represents the parameter of the header.

**@CookieParam --** It represents the parameter of the cookie.

**@Produces -- It** defines media type for the response such as XML, PLAIN, JSON etc. It defines the media type that the methods of a resource class or MessageBodyWriter can produce.

**@Consumes --** It defines the media type that the methods of a resource class or MessageBodyReader can produce.