# Restful Web Services


# JAX-RS

**Note:**

**-----**

**New features in JAX-RS 2.0**

**|-@BeanParam**

**|-Programmatic Type Convertors**

**|-ParameterConverters**

**|-Custom Type Convertors**

**|-Content Handlers**

**|-Server Response and Exception Handling**

**|-Custom Marshalling and Un-marshalling using message body readers and writers**

**|-Exception Handling**

**|-JAX-RS Client**

**|-Caching Support**

**|-Asynchronous RESTful Services**

**|-Client API**

**|-Exceptions has been modified in JAX-RS 2.0 and new interfaces are added.**

## Details Content

|-3.5) HTTP Method and URI Matching

3.1 Development of RESTful Service
|-Implementations for JAX-RS
|-Support of Java to develop the RESTful services
|-Use case (DTDC Courier with Local Courier vendor)
|-The basic set things we need to develop web services and RESTful with comparison
|-Steps to Develop RESTful with the help of RESTful principles
|-Ways to send the data to server over Http req
|- Differences between the GET and POST
|- Difference between JERSEY and RESTEasy Implementations

3.2 RESTEasy Implementation
|-Set-up to work with RESTEasy
|-Configuring the JAX-RS RESTEasy Run time
|-Accessing the Application using Browser or Rest Clients
|-Bootstrapping Mechanisms for RESTEasy Runtime
|-1. Init-Param [resteasy.scan=true]
|-2. Control Scopes [javax.ws.rs.Application=Our Application class]
|-3. Customize the resource URI [resteasy.servlet.mapping.prefix]
|-4. No web.xml [Works with JEE-5 Enable container only]
|-Purpose of the JAX-RS API
(Or)
Purpose of JAX-RS Runtime which is specific to the Vendor Implementation

3.3 JERSEY Implementation
|-Set up to work with JERSEY
|-Bootstrapping mechanisms for JAX-RS ServletContainer Which is specific to JERSEY
|-1. Auto scan
|-Difference between the JERSEY Runtime (ServletContainer) and RESTEasy Runtime (HttpServletDispatcher)
a) Concrete Resource (classes scanning) scanning
b) Jersey packages Scanning including sub packages
|-2. Custom Application subclass and Controlling Scopes
|-3. Custom ResourceConfig subclass and Controlling Scopes
|-4. Servlet Plugability Mechanism (ServletContainer will be automatically added by JERSEY)
a) JAX-RS API Application class without an Application subclass
b) JAX-RS API Application with a custom Application subclass or ResourceConfig subclass
|-5. without web.xml
a) Using @ApplicationPath with Custom Application
b) Using @ApplicationPath with Custom ResourceConfig
|-Internal Code for without web.xml Using @ApplicationPath with Custom Application or Custom ResourceConfig
|-Configuring the JAX-RS Runtime which will works agnostic to the Implementation
(Or)
Recommended Bootstrapping or best Runtime bootstrapping mechanism

|-Internal Req flow of RESTful services

3.4 Developing REST Resource Using Interfaces & Abstract Classes
|-REST Resource Using Interfaces
|-Advantages of REST development using Interfaces
|-REST Resource Using Abstract classes
|-Use case related for need of Abstract classes in REST
|-Differences between developing the REST Resources using Interfaces and Abstract classes

3.5 HTTP Method and URI Matching
|-@Path
|-Binding URIs
|-@Path Expressions
|-Template Parameters
|-Problems with Query Params
|-Benefits of @Path Expressions
|-Differences between the Query and Template Params
|-Regular Expressions
|-Matrix Parameters
|-Differences between the Query, Path and Matrix Params and when to use them
|-Sub Resource Locators
|-Full Dynamic Dispatching
|-Terminology in REST Resource

4. JAX-RS Injection
|-Def of JAX-RS Injection
|-Basics
|-@QueryParam (We already discussed)
|-@PathParam and @MatrixParam
|-1) More Than One Path Parameter & Scope of Path Parameters
|-2) Single PathSegment and Matrix Parameters
|-3) Matching with multiple PathSegments and Matrix Parameters (List<PathSegment>)
|-1) Using one List<PathSegment>
|-2) Using multiple List<PathSegment>
|-Limitations of Annotations Approach
|-Programmatic URI Information
|-1) Accessing all the path segment params of URI (Possible only uisng Programatic Approch)
|-2) Accessing all tha matrix params of all the Path segments of URI
[(OR) we can use multiple List<PathSegment>]
|-3) Accessing all the Query params of URI (or) Programmatic Query Parameter Information
|-4) Programmatic URI Information with Regular Expressions
|-Differences between PathParam and List<PathParam> and UriInfo
|-Comparisons between Programatic UriInfo over Annotations
|-@FormParam
|-@HeaderParam

|-@CookieParam
|-HttpHeaders (or) Programatic Access to Headers and Cookies (or) Raw Headers and Cookies
|-@BeanParam
|-Common Functionality
  |-Automatic Java Type Conversion
    |-Introduction
    |-Automatic Java Type Conversion criterias/conditions
      |-1) Primitive type conversion
      |-2) Java object conversion
      |-3) Collections
          |-Internal Working Flow of Java object Convertion with Collection
      |-4) ParamConverters or CustomParamConverter or Low Level API Converters or Programatic Converters
          |-a) ParamConverters or CustomParamConverter Without Reg Exp
            |-Purpose of @Provider
            |-Final Working Flow
          |-b) CustomParamConverter and Reg Exp With Collection
          |-c) CustomParamConverter with Path and Matrix Params
      |-5) Conversion failures
  |-@DefaultValue
  |-Working with Date obj
    |-1) Constructor with a single String parameter (or) static method named valueOf() that takes a single String argument.
    |-2) Using ParamConverter
  |-@Encoded

5 JAX-RS Content Handlers (Modelling RESTful as object to object)
  |-Intriduction
    |-Definition Content Handlers
  |-1) Built-in Content Marshalling
      |-Purpose of Built-in Marshallers
      |-1) javax.ws.rs.core.StreamingOutput (Used for to send Response/Write operation)
          |-Advantage of making OutputStream as a Callback mechanism
      |-2) java.io.InputStream
      |-3) java.io.Reader
      |-4) byte[]
      |-5) String
      |-6) char[]
      |-7) MultivaluedMap<String, String> and Form Input
      |-8) java.io.File
          |-Use Case
          |-Internal working flow of File in case of JAX-RS:
          |-General probelms while working with Files in REStful
          |-Checking practially whether the temp file created or not
      |-9) javax.xml.transform.Source
      |-Working with different types of return types
  |-2) Custom Content Handlers or Custom Marshallers and Unmarshallers

## History of the RESTful Web Services

➔Web Services has been released in 1998 and real web services starts begun in 2000.
➔The initial notation of RESTful Web services begun in the year of 2006.
➔SOAP Bases services are called as Web Services.
➔Non-Based Services are called as RESTful Services.
➔Now currently industry is using Cloud enable RESTfulWeb Services. Like AWS cloud.
➔In order to work with RESTful Web Services Sun Microsystem has provided API called as JAX-RS (Java API for RESTful Web Services) and it has provided one Impl called as JERSEYand it has not successful because lot of features are missing, that's where 3rd party libraries has provided Impl for the JAX-RS API and one of the popular 3rd party vendor Impl is JBOSS RESTEasy.

### JAX-RS API

|-JAX-RS 1.0 API
|-JAX-RS 1.1 API
|-JAX-RS 2.0 API

### JAX-RS API 1.0/1.1Implementations

|-JERSEY by Sun called as JSR 311 Reference Implementation
|-JBOSS RESTEasy

→The missing features are taken from the JBOSS RESTEasy and Oracle (The acquisition of Sun Microsystems by Oracle Corporation was completed on January 27, 2010.) made it as JAX-RS 2.0 and released in the Year of May 2013 and in 2014 it has been adapted by most of the industry at 2014.

### JAX-RS API 2.0 Implementations

|-JERSEY by Oracle called as JSR 339 Reference Implementation
(jsr-java spacifiaction resource)
|-JBOSS RESTEasy

➔JERSEY RESTful Web Services framework is an open source framework for developing RESTful Web Services in Java. It provides support for JAX-RS APIs and serves as a JAX-RS (JSR 311 & JSR 339) Reference Implementation.
➔JSR 339 Group is a Java Community Process (JCP) where Implementation for the JAX-RS will be decided with collaborative to the Vendors that is the reason Sun Ms made Sun Reference Impl as JERSEY Reference Impl for JAX-RS 1.0 as JSR 311 and for JAX-RS 2.0 as JSR 339
➔JSR means JERSEY.
➔The JAX-RS A PIsserves as a JAX-RS (JSR 311 & JSR 339) Reference Implementations.
➔Now no one using JAX-RS 1.0 all the industry is using only JAX-RS 2.0 API only.
➔JAX-RS 2.0 and JERSEY (JSR 339) Implementation has been adapted by the industry and it has more popular like RESTful means JERSEY and JERSEY means RESTful.

## Purpose of RESTful Web Services:

➔SOAP Bases services are called as Web Services.

➔Non-Based Services are called as RESTful Web Services.

➔Distributed tech is meant for distributing the business functionality without re-building the application that is there in our own system to the business users so that better through-put will be achieved and scalability of the business will increase.

➔But Distributed tech will not be interoperable that's where we need to go for Interoperable Distributed Tech which is called as Web Services.

➔The Problem with Web Services is that Web Services restricts or it imposes some rules (By WSI Organization) without using their own protocols or own data formats to achieve the interoperability on the heterogeneous systems.

➔The WSI Organization imposes some rules to achieve the interoperability such as

|-We must use SOAP and it made as interoperable data format protocol which is agnostic to the Language and agnostic the transport protocol

|-It recommends Http made as transport protocol

|-XML as data exchanging the data

➔Interoperability means

|-Independent of the Plat form

|-Independent of the language

➔That means Web Services is an interoperable tech if and if we agree the rules that are imposed by the WSI Organization other it cannot be interoperable. That means for the sake of interoperability Web Services has defined as lot rules and restrictions.

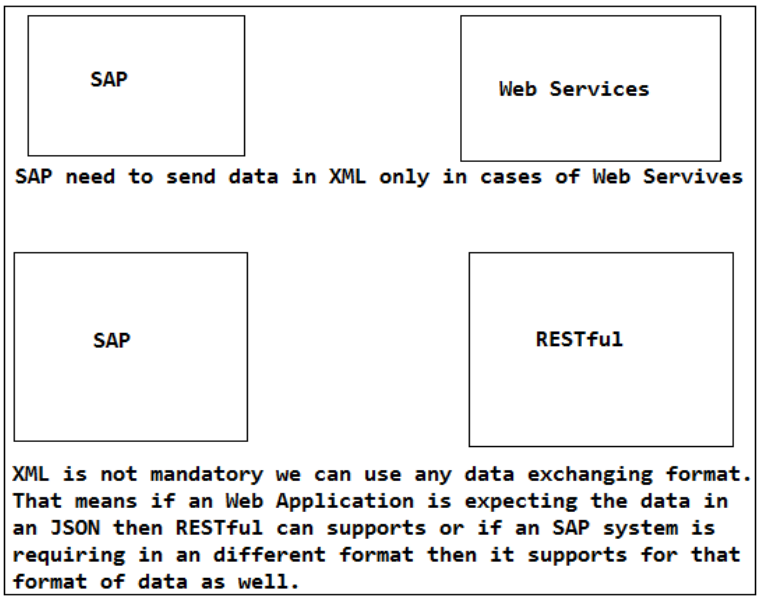
➔But in comes to the RESTful services it doesn't imposes any kind of restrictions or rules to achieve the interoperability that means

|-We can use any data format language like xml, SOAP over xml, JSON,text, binary and own data formats etc. That means we don't have contractual restrictions to achieve the interoperability. That means we have freedom of modelling the data format language. So we can send any format so that it can return own formats.

|-RESTful not imposes ant restriction to use SOAP protocol rather it is optional that means RESTful is Non-SOAP based service

|-RESTful recommends Http protocol but if we want we use any protocol.

**Note:**
JSON or JavaScript Object Notation is a lightweight text-based open standard designed for human-readable data interchange.




➔The concept of the RESTful services is introduced by Roy Fielding stating to achieve the interoperability we don't need SOAP protocol rather we can any data exchanging protocol.
→Roy Fielding is one of the Http protocol Author and he developed the Http-Server in Apache Foundation.

**Principles of RESTful Services (Adopted from Principle of Web):**

➔Web distributed tech and is very successful and it is Prevalent(spreading in each and every corner of the world).

➔So if we can adopt the principles of the Web to our Application then our services then our services also becomes success.

➔That means we develop the Distributed Application based on the same principles that Web is working then our applications also become successful.

➔That's where he drafted the thesis and submitted the university in the year of 2000 that's where RESTful services evolved.

➔The concept of is introduced by the students of the University for intranet then it became evolved as Web.

➔That means adapting the Web principles to our applications so that our applications also become so successful as like Web and in a distributed manner interoperable manner.

**Principle of Web (RESTful):**

**➔Unique Addressable URI (Unique Resource Identifier)**
**➔Uniform, Constrained Interfaces**
|-Advantages of Uniform, Constrained Interfaces
1. Familiarity
2. Interoperability
3. Scalability

**➔ Representation-Oriented**
**➔Stateless Communicate**

|-Advantages
1. Avoids the Thread-Starvation Issues

➔**HATEOAS (Hypermedia As The Engine Of Application State)**

|-Advantages of HATEOAS
1. The engine of application state
2. Avoids/Eliminates (WSDL) the Service Description Documentation

➔**Caching**

## 1. Unique Address URI:

➔Anything in the world has unique address for example to access the JSP page we need address URL and can access using Http.

➔That Address URI is the key factor to access the resources on Web.

➔For example if there is a SAP System and it wants to talk with Web Application we need to develop the Web Service using JEE and SAP can access the Web Service SAP needs

1. End-point URL of the Web Service
2. SOAP Action of the Web Service otherwise it cannot access

➔That means by using URL itself we cannot access the Web Service rather we need SOAP Action as well but we can Access the Web resource using Unique URL itself directly (like jsp can be accessible uniquely using URL which is unique for that source).

➔That means when we are integrating with Web Services we need to know the URL and SOAP Action to integrate with the Systemsthat means which is difficult but in case of Web we can access the resource using URI itself directly.

➔So we don't have unique addressable URI to integrate systems directly so integration becomes complicated in cases of Web Services.

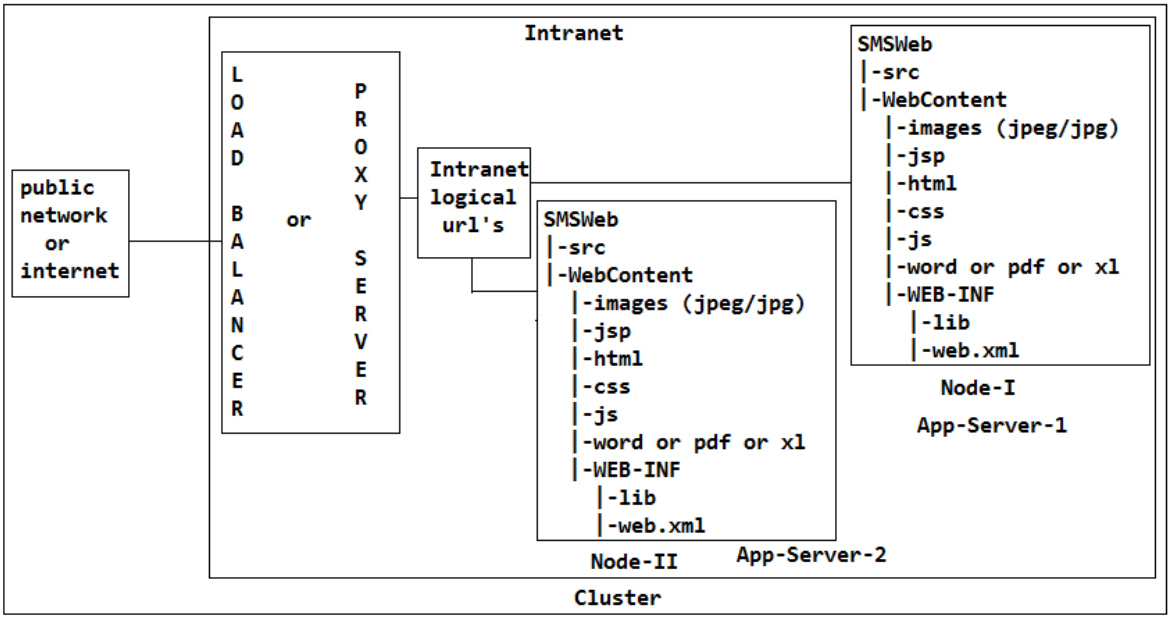
➔So if we adapt this Web principle to our applications then integration efforts becomes easy so that we can access the service uniquely and quickly without any additional efforts, that's where this principle is adapted for the RESTful services.

➔If we adapt this principle to the RESTful then it easy to integrate desperate (different) systems without any SOAP action or its additional addressing factors even they are different technologies like SAP, EJB, CORBA, .Net etc we can directly access using Unique URI if we adapt this web principle to RESTful.

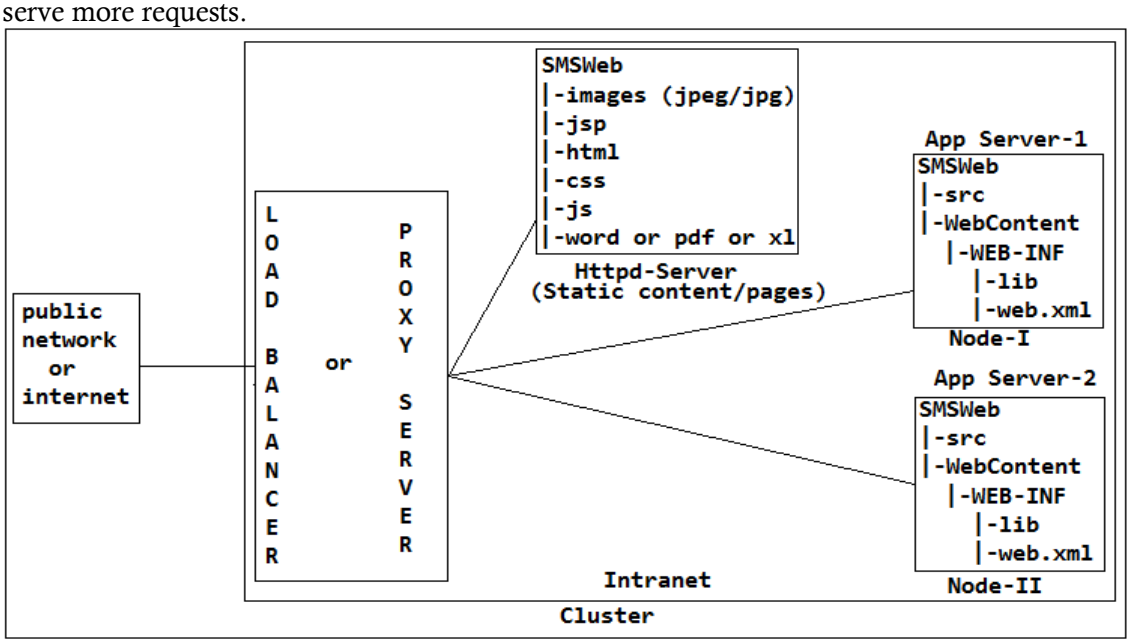
➔In production the applications will be deployed in a cluster environment. Cluster is a collection of nodes.

➔We will deploy the applications in multiple server in cluster Environment to get High Availability that means to distribute the load across the multiple Application Server's or to facilitate more no. of requests to be processed to avoid the performance issues we distribute the applications across multiple Application servers with one logical url form the end-user prospective.

➔Whenever the request comes to the LOAD-BALANCER it will checks whether in which Application Server or in which Node the load is less then forwards to that Node so that performance and high availability for requests to access the resources.




➔For example whenever we send a request to access the static page like image logo then also request goes to the LOAD BALANCER and checks which node is free then forwards the request to that Node then that node will process and gives the response.
➔But the problem here is just to access the static pages the LOAD BALANCER need to check which node is free and then forwards the request which is time taking process and if this type static content requests comes then performance of the application will be degraded because with static pages will not render any dynamic data that's where Httpd-Server will be used to render the static pages so that we can avoid the static requests to the cluster so that high availability of the cluster will be achieved so that it can serve more requests.




➔For example if we send the request to "/home.jsp"(which not static jsp and will be there in the App-Server- or App-Server-2) then request comes to the LOAD BALANCER then it will checks whether load

is there on Node-I or Node-II the which has less load if Node-I has less load then jsp will gets translated in Node-I as _service(req,res) and will be called then res will be displayed.

Header part in the browser
<header src=image/logo.jpg>
</header>

➔That means whenever the request comes as GET will be directed to the Httpd-Server then it checks in the directory for the "logo.jpg" will renders back to the browser over the Http so that performance of the JEE containers will not be degraded.

➔Httpd-Server or directory server (because it looks for the static resources in Directory) which will manages by serves the static resources as output over the Http protocol. That means anything will be treated as resource by the Http.

➔For example if we wanted to change the log then we need to send the request as POST so don't need to stop the JEE Containers (App-Server's) and we no need to stop the Httpd-Server as well rather it gets modified easily which is called as run time behaviour of the Httpd-Server and we wanted to delete the logo then send request as DELETE then it gets deleted by the Httpd-Server.

➔With this Httpd-Server concept there will be server's called as Content Management server adding some API's over Httpd-Server is called as Content Management server which is used management the static resources upload/download in the production systems.

➔Anything that gets accessed (static/dynamic) over the Http protocol is called as web-resource.

➔In internet we will see while downloading the resources then it shows as click here to download over FTP-Protocol or click here to download over Http-protocol that means Httpd-Server will be there in the back-end and gets downloaded.

➔Httpd-Server is light is light compared to the FTP-Server because FTP will gets client info to download or to access the resources but Http protocol is not.
➔That means Http protocol is state less and FTP is state full protocol that's where Httpd-server performance is more and it less weight because it doesn't carry any client info.

## What is RESTful or why REST why not the SOAP and explain the principles of the RESTful?

### 2. Uniform, Constrained Interfaces:

➔Interface means the way we can communicate that means how can we communicate to the object.

Ex:

```
Interface RailReservation {
        Receipt bookTicket(PassengerInfo, JourneyInfo);
        Status getStatus(String pnr);
        String cancelTicket(String ticketNo);
}
```

➔Interfaces means how do we provide the services to the external world so that peoples can interact with us, that means we can have any methods with any name to expose as web services.

➔But in RESTful services there is constraint to write the service methods that means we don't have freedom to write any method name rather it restricts us to write method names as follows.

➔So that we can access the service with Unique URI directly without any additional URI otherwise we need additional URI's soap Action in case of web services.

➔The REST principle of a constrained interface is perhaps the hardest pill for an experienced CORBA or SOAP developer to swallow.

➔The idea behind it is that you stick to the finite set of operations of the application protocol you're distributing your services upon.

➔This means that you don't have a soap "action" parameter in your URI.

➔ But if we don't have soap action we cannot access the service that where we need use only constrained (or conditional) methods of HTTP for our web services Interface across the world uniformly which is called as "Uniform-ConstrainedInterface".

➔HTTP has a small, fixed set of operational methods. Each method has a specific purpose and meaning.

1. GET
2. PUT
3. POST
4. DELETE
5. HEAD
6. OPTIONS
7. TRACE
8. CONNECT

Assume

```
Interface RailReservation {
        Receipt post(PassengerInfo, JourneyInfo);
        Status get(String pnr);
        String delete(String ticketNo);
}
```

➔So we can directly access the service over the Http request directly with Unique URL with post request directly using Http standard methods so that we don't need soap action. That means we can book a ticket directly with unique URL itself.

**GET**

➔It is a read-only operationor read only access to a resource.
➔It is used to query the server for specific informationand no matter how many times you apply the operation repeatedly, the result is always the same hence it will cache's the data in the Client side so that performance of the application will not degrades.

**PUT**

➔Used to update a new resource.

**POST**

➔Used to update a existing resource or create a new resource.

**DELETE**

➔Used to remove a resource.

**OPTIONS**

➔Used to get the supported operations on a resource.

**HEAD**

➔ It is exactly like GET except that instead of returning a response body, it returns only a response code and any headers associated with the request.

➔There are other HTTP methods (like TRACE and CONNECT), but they are unimportant when designing and implementing RESTful web services.

➔You may be scratching your head and thinking, "How is it possible to write a distributed service with only four to six methods?" Well…SQL only has four operations: SELECT, INSERT, UPDATE, and DELETE. JMS and other Message Oriented Middleware (MOM) really only have two logical operations: send and receive.

➔How powerful are these tools? For both SQL and JMS, the complexity of the interaction is confined purely to the data model.

➔The addressability and operations are well defined and finite and the hard stuff is delegated to the data model (in the case of SQL) or the message body (in the case of JMS).

**Advantages of Uniform, Constrained Interface:**

1. Familiarity
2. Interoperability
3. Scalability

### 1. Familiarity:

➔If you have a URI that points to a service, you know exactly which methods are available on that resource.
➔You don't need an IDL-like file describing which methods are available.
➔You don't need stubs. All you need is an HTTP client library.
➔If you have a document that is composed of links to data provided by many different services, you already know which method to call to pull in data from those links.

### 2. Interoperability:

➔HTTP is a very ubiquitous (present, appearing, or found everywhere) protocol.
➔Most programming languages have an HTTP client library available to them.
➔So, if our web service is exposed over HTTP, there is a very high probability that people who ever want to use your service will be able to do so without any additional requirements beyond being able to exchange the data formats the service is expecting.
➔But with CORBA or Web Services we have to install vendor-specific client libraries as well as loads and loads of IDL or WSDL-generated stub code.

➔That means we have to build a Consumer with client libraries which is specific to that vendor.

➔So with Web Services (WS-I set of specifications) or CORBA, we are getting vendor specific interoperability only and to get this vendor specific interoperability we have to make sure that your client and server are using the same specification version of the protocol otherwise interoperability is not achieved in cases of Web services.

➔For example we developed the provider (may be Java or .Net) using SOAP1.1 and rpc-encoded then consumer (Java consumer or .Net) also need to have same protocol versions but if the provider changes the protocol versions then the consumer that we developed will not works that means we need re-build the consumer again to access the provider because vendor to vendor.

➔The WS-* specification common for all the services but all features of the WS-* the vendors may not supports that means if vendor protocols and Impl vendor has been changed then interoperability may not

be guaranteed because that vendor may not supports for the consumer that we have already developed so again we need re-build the consumer because changes form one vendor to another protocol and formats versions may not supports for any consumer.

➔That means with Web Services we are getting only vendor specific interoperability but not the complete application interoperability, But with REST over HTTP, we don't need to worry about either of these things and can just focus on understanding the data format of the service that if have a Http connection then it is enough we don't need anything so we can achieve application interoperability, rather than vendor interoperability.

### 3. Scalability:

➔Because REST constrains you to a well-defined set of methods, you have predictable behaviour that can have incredible performance benefits.

➔GET is the strongest example that means when surfing the Internet, have you noticed that the second time you browse to a specific page it comes up faster, this is because your browser caches already visited pages and images.

➔That means once we get the response form the server if we again send the request to the server it will gets the data from the Cache so that additional overhead (Re-computing same request and sending response back) on the server will gets reduced so that performance will high.

➔HTTP has a fairly rich and configurable protocol for defining caching semantics. Because GET is a read method that is both idempotent and safe (GET will not modifies it selects or read only), browsers and HTTP proxies can cache responses to servers, and this can save a huge amount of network traffic and hits to your website.

➔If add HTTP caching semantics to our web services so that we can have an incredibly rich way of defining caching policies for our services.

➔It doesn't end with caching, though. Consider both PUT and DELETE. Because they are idempotent, neither the client nor the server has to worry about handling duplicate message delivery. This saves a lot of bookkeeping and complex code.

### 3. Representation-Oriented

➔The third architectural principle of REST is that our services should be representation-oriented.
➔Each service is addressable through a specific URI and representations are exchanged between the client and service with a GET operation, you are receiving a representation of the current state of that resource but it cannot modify the state because it read only.

➔A PUT or POST passes a representation of the resource to the server so that the underlying resource's state can change.

➔Whenever the consumer sends the request to the Provider then it will takes xml over SOAP as input gives res as xml over SOAP but not any other formats.

➔For example if our client system wants the data in the form text then we need to convert the xml data to an text data which is another additional computational process in case of web services because xml parsing take time.

➔But if we go for RESTful we can get in format or any MIME type as res so that we can avoid additional conversions so that performance will increase.

➔Another example is the if our Ajax based application needs the data in the form of java scripts object then we need to use DOM parsing logic, if we use web services.

➔In a RESTful system, the complexity of the client-server interaction is within the representations being passed back and forth. These representations could be XML, JSON, YAML, or really any format we can come up with HTTP.

➔The representation is the message body of your request or response. An HTTP message body may be in any format the server and client want to exchange. HTTP uses the Content-Type header to tell the client or server what data format it is receiving.

➔The Content-Type header value string is in the Multipurpose Internet Mail Extension (MIME) format. The MIME format is very simple:

type/subtype; name=value; name=value...

➔type is the main format family and subtype is a category. Optionally, the MIME type can have a set of name/value pair properties delimited by the ";" character. Some examples are:

text/plain
text/html
application/xml
text/html; charset=iso-8859-1

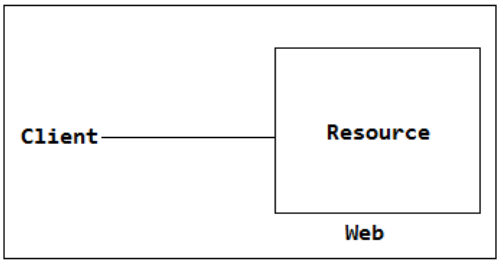
➔Representation-oriented(In what format or representation we want the data it gives in that format or orientation RESTful gives the data) means the resource will not only works with specific data formats or a data type rather our RESTful resource can present the data in any format so that integration efforts will become easy due to the Http and Http contains MIME types.

➔For example:One of the more interesting features of HTTP that leverages MIME types is the capability of the client and server to negotiate the message formats being exchanged between them. While not used very much by your browser, HTTP content negotiation (negotiation means supports for converts to any format because of MIME types are there with Http itself) is a very powerful tool when writing web services. With the Accept header, a client can list its preferred response formats like. Ajax clients can ask for JSON, Java for XML, Ruby for YAML.

➔Another thing this is very useful for is versioning of services. The same service can be available through the same URI with the same methods (GET, POST, etc.), and all that changes is the MIME type.

➔For example, the MIME type could be application/vnd+xml for an old service while newer services could exchange application/vnd+xml; version=1.1 MIME types. All in all, because REST and HTTP have a layered approach to addressability, method choice, and data format, you have a much more decoupled protocol that allows your service to interact with a wide variety of different clients in a consistent way.

## 4. Communicate Stateless




➔Whenever the client sends the request to access the web resource that is there in the server and server cannot store the state of the Client because Http is stateless.

➔That means if Server needs the state of the client Http will not gives all the client info to store in the server side for referring the same state in future but Http will not gives any state to store on the server because it is stateless.

➔That means it cannot carry any data, that means in a session we can have multiple operations in that for one operation only the data will be available but for another operation the previous state will not be available which is called as stateless.

➔Http is stateless but we can make Http as stateful so that we can store the data in the server side so that we don't get the data from the client for each and every operation using use client state remembering techniques or session management techniques like

1. Cookies
2. Hidden Form Field
3. URL Rewriting
4. Http Session

➔So by using these session management techniques we can remember the state of the client so that we can consecutively communication between the server and Client.

➔If more no.of (1-lack req's) users are accessing the resources then for 1-lack req's the server will maintains the state of the clients which will leads to the scalability of the application becomes less because RAM memory has been full because state of the clients has been stored by the server with Http Session hence there no computational resources are available due to which further requests cannot accommodate the server.

➔But when comes to the RESTful it is completely against to the stateful that completely RESTful is stateless communication so that server will not loads any client info related to the client in the server so that enough amount of computational resources are available to accommodate more no. of users so that scalability of the application becomes high.

➔But we need to remember the data of the client for the server side but if make server as stateful then server cannot scale-up that where RESTful always sends the state of the client to the server repeatedly whenever server needs by using session Caching mechanisms at the Client side so that server need not to worry about the client data rather client itself sends the data using caching mechanisms at the Client (Browser) side due to which scalability of the server becomes more.

➔That means in RESTful state of the client cannot be remembered by the server rather client itself remembers the state using Caching mechanisms at the client side. So that Server becomes stateless i.e thin server and client becomes thick client.

➔That means our web also stateless and RESTful also becomes stateless because Http session caching mechanisms at the client side. That is the reason this feature has been adopted for the RESTful from the web.

➔When we talk about statelessness, In REST, stateless means that there is no client session data stored on the server.

➔That means server will not records and manages the state of the client to expose resources to the external world.

➔If there needs to be session-specific data, it should be held and maintained by the client and transferred to the server with each request as needed.

➔A service provider or resource layer that does not have to maintain client sessions so for the server easier to scale-up in a clustered environment.

➔If we look in to the older many distributed applications had a fat GUI client written in Visual Basic, Power Builder, or Visual C++ talking RPCs to a middle tier that sat in front of a database.

➔The server was stateless and just processed data.The fat (thick) client held all session state.

➔The problem with this architecture was an IT operations one that means we cannot perform multiple operations in faster manner so it was very hard for operations to upgrade, patch, and maintain client GUIs in large environments.

➔But Web applications solved this problem because the applications could be delivered from a central server and rendered by the browser and we started maintaining client sessions on the server because of the limitations of the browser.

➔But from 2008, with the growing popularity of Ajax, Flex, and Java FX, the browsers are sophisticated enough to maintain their own session state like their fat-client as we did like in mid-'90s. So now we can now go back to that stateless scalable middle tier that we enjoyed in the past without any problems.

## Advantages stateless communication:

➔Avoids the Thread-Starvation Issues:

➔One of the key factor of the Web is stateless i.e Http is Stateless so server will not stores any data of the Client hence we can avoid the "Thread Starvation issue" at the server side.

➔Thread Starvation issues meansif server tries to store the client data in the server then server resources will gets consumed that means server memory resources gets consumed,if more requests are coming and there is no resources will be available to accommodate the new requests but server tries to accommodate more requests by using thread switching hence thread switching keep on happening and there are no resources (no memory) available to compute (process)requests the where no other operations will not get executed by any of the thread which is called as "Thread-Starvation".

➔To avoid this we can use De-Hydration concept. But these will not happen in RESTful or Web because these are completely stateless.

## 5. HATEOAS (Hypermedia as the Engine of Application State)

➔For example if we accessed the web site and in order to know about the web site we don't need to read the documentation of the web site rather they will provides some hyperlinks as navigations which will tells the features of web site so that user can easily understands and can use that site comfortably without any documentation of the website.

➔Similarly the RESTful also adopted this principle, that means the RESTful not only sends res rather it even sends some Hypermedia (Hyperlinks) along with res so that client can do easily do the next course of action or easily interact with the resource without any documentation of the resource that he is trying to access.

➔For example if we send the request to get the order details then it will displays all list of orders and along with the list of order we can provided Hypermedia (Hyperlinks) to the each and every order so that client can easily know the details of each and every list of orders.

List of Order:

| Sno | OrderName | For Order Details |
|---|---|---|
| 1 | Dell | ClickHere |
| 2 | Mobile | ClickHere |
| 3 | Modem | ClickHere |

➔Similarly we can provide any Hypermedia (Hyperlinks) like edit the order, to create a new order so that client can easily can do the next course of actions easily without any documentation to access the resource.

## Use-Case Related to HATEOAS:

➔The final principle of REST is the idea of using Hypermedia As the Engine of Application State (HATEOAS).
➔Hypermedia is a document-centric approach with the added support for embedding links to other services and information within that document format.
➔HATEOAS means Addressability using hyperlinks within the data format received from a service.
➔One of the uses of hypermedia and hyperlinks is composing complex sets of information from disparate sources.
➔The information could be within a company intranet or dispersed across the Internet. Hyperlinks allow us to reference and aggregate additional data without bloating our responses. ➔The e-commerce order in Addressability is an example of this:

```
<order id="111">
        <customer>http://customers.myintranet.com/customers/32133</customer>
        <order-entries>
        <order-entry>
        <quantity>5</quantity>
        <product>http://products.myintranet.com/products/111</product>
        ...
</order>
```

➔In that example, links embedded within the document allowed us to bring in additional information as needed. Aggregation isn't the full concept of HATEOAS, though. The more interesting part of HATEOAS is the "engine."

## Advantages of HATEOAS:

➔The engine of application state:

Let us say we are on amazon.com buying a book, we follow a series of links and fill out one or two forms before your credit card is charged. You transition through the ordering process by examining and interacting with the responses returned by each link you follow and each form you submit.

➔The server guides you through the order process by embedding where you should go next within the HTML data it provides your browser.

➔This is very different from the way traditional distributed applications work. Older applications usually have a list of precanned services they know exist and interact with a central directory server to locate these services on the network.

➔HATEOAS is a bit different because with each request returned from a server it tells you what new interactions you can do next, as well as where to go to transition the state of your applications.

➔For example, let's say we wanted to get a list of products available on a web store. We do an HTTP GET on http://example.com/webstore/products and receive back:

```
<products>
        <product id="123">
        <name>headphones</name>
        <price>$16.99</price>
</product>
<product id="124">
        <name>USB Cable</name>
        <price>$5.99</price>
</product>
...
</products>
```

➔This could be problematic if we had thousands of products to send back to our client. We might overload it, or the client might wait forever for the response to finish downloading. We could instead list only the first five products and provide a link to get the next set:

```
<products>
        <link rel="next" href="http://example.com/webstore/products?startIndex=5"/>
        <product id="123">
        <name>headphones</name>
        <price>$16.99</price>
</product>
...
</products>
```

➔When first querying for a list of products, clients don't have to know they're only getting back a list of five products. The data format can tell them that they didn't get a full set and to get the next set follow a specific link. Following the next link could get them back a new document with additional links:

```
<products>
        <linkrel="previous" href="http://example.com/webstore/products?startIndex=1"/>
        <link rel="next" href="http://example.com/webstore/products?startIndex=5"/>
        <product id="128">
        <name>stuff</name>
        <price>$16.99</price>
</product>
```

...
</products>

➔In this case, there is the additional state transition of previous so that clients can browse an earlier part of the product list.

➔The next and previous links seem a bit trivial, but imagine if we had other transition types like payment, inventory, or sales.

➔This sort of approach gives the server a lot of flexibility, as it can change where and how state transitions happen on the fly.

## 2. Avoids/Eliminates (WSDL) the Service Description Documentation

➔So with Hypermedia (Hyperlinks) we don't need any WSDL document describing of the provider or a Resource that we are trying to access.So now can avoid the WSDL because we don't need SOAP, we don't need MEF and we don't need to describe the services that are there at the provider because all the info we are trying put in the form Hypermedia (Hyperlinks) so we don't need any document.

➔By looking at this principle of Web the REST identifies the key architectural principles of why the Web is so prevalent and scalable. The next step in the evolution of the Web is to apply these principles to the semantic Web and the world of web services.

➔So we adopt this Web Principles to the world of Web services which will becomes or which will results RESTful services.

➔REST offers a simple, interoperable, and flexible way of writing web services that can be very different than the RPC mechanisms like CORBA and WS-*.

➔Hence Adopting the Web Principles to the Web Services=RESTful That's where RESTful services follows the Web Principles.

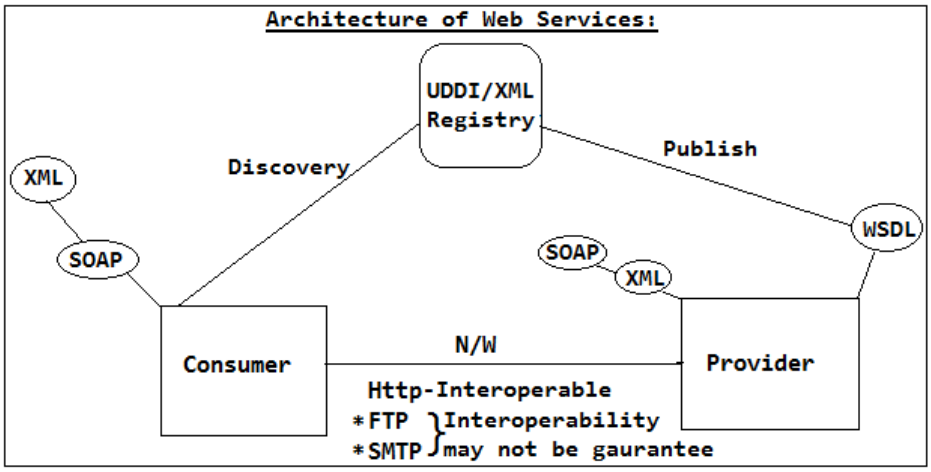
## Architecture of RESTful Services (Representational State Transfer Services):

### What is RESTful service can u please explain?

➔RESTful services are one form of the Web Services that means the new architectural style of developing the Web services is called as RESTful services.

➔The Web Services are bounded with set of rules and standards unless until we bound to the standards that provided by the WS-I we cannot develop the web services.

➔That means the programming is surrounded with standards and protocol specifications but when it comes to the RESTful services there is no standards in developing the RESTful.
➔Hence there is no WS-I organization for the RESTful services rather it is kind of Architectural style over which we can develop the applications that is the reason it is called as new architectural style of developing the Web services.




➔Provider is seems to be a restricted service which will expects the data in an specific format surrounded with standards and protocol specifications but RESTful accessed in an Uniform (easy/simple) way as like Web hence we should not call it as Provider (Service) rather we need to called as Resources.

➔Hence we can refer the services that we are trying to develop using REST is called as REST-Service/Resource/Web-Resource because these can accessed like a web resource just by using single URI hence we will name Consumer as Client.

➔RESTful is not requires any data formats or SOAP protocols hence in the architecture we will completely eliminate the SOAP and MEF.

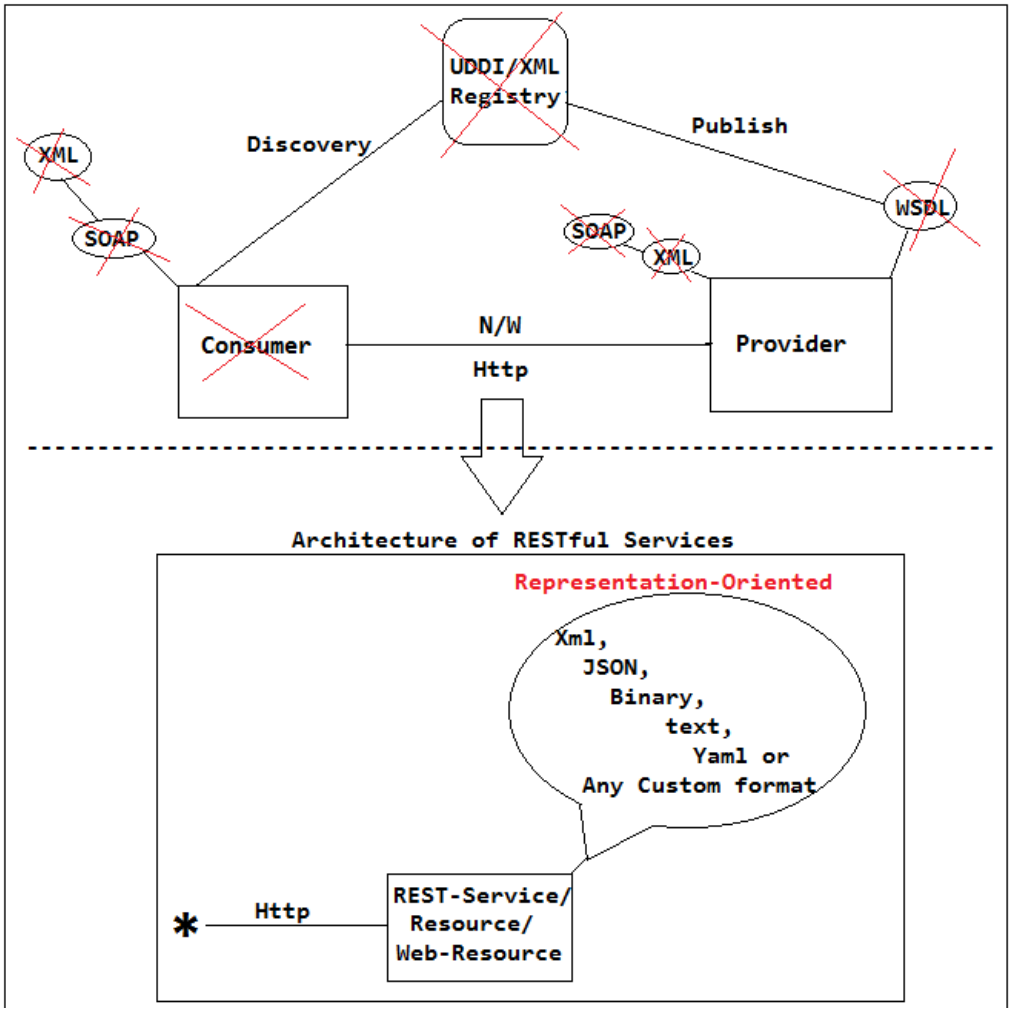
➔Resource can accept xml, JSON, binary, text, Yaml or any custom format etc that we wish to access the resource that means the resource not only serve the one data format rather it can serve of the data representational that we are looking for which is called Representational Format/Representation-Oriented.

➔We can any format but recommended to Xml, JSON etc. But it not recommended to use binary because interoperability will not be guaranteed.

➔If client wants to access the Resource then we can use any protocol in RESTful but JAX-RS API has been designed based on the Http protocol only hence we need to use only Http only to access the Resources.

➔If we client wants to access the Resource description language is not required like WSDL hence we need to discover or we no need to publish hence we don't need UDDI-registry in case of RESTful.

➔Hence we can access the Resource by simply sending the request as like web hence who will be cable of sending the Http request can acts as Client hence there is development of the Client (Consumer or Stub) as well which is called as "New Architectural Style of Developing the Web Services" which is called as "RESTful services".




**➔What is the RESTful services and how does the RESTful services different form Web Services?**

Ans: Explain the Architecture

## 3. HTTP methods and Http Status codes:

HTTP methods:

Idempotent: $AA^T = A^TA = I$

➔The above property is called as Matrix Idempotent property. That means if we multiply A with A-Transpose and A-Transpose with which results I which means same result.

➔Similarly Idempotent in Http means no matter how many times we are applying the operation, the result is always the same. The act of reading an HTML document shouldn't change the document.

Example:
➔If we placed the order in the Flip-Kart and we send the requestto get the order details of that particular shipment details then it shows the details and if we send again with the same order-id to get the details of the order that we placed then also same data will be displayed which means GET operation will effects on the resource that is stored on the server side.

**Safe:**
➔Safe means that invoking a GET does not change the state of the server at all.
➔This means that GET will not change state of the server side operation so it will not affect the server.

**Idempotent Methods:**

➔Methods can also have the property of "idempotence" in that (aside from error or expiration issues) the side-effects of N > 0 identical requests is the same as for a single request.

➔The methods GET, HEAD, PUT and DELETE share this property.
➔The methods OPTIONS and TRACE SHOULD NOT have side effects.
➔Idempotent means if a single execution of the entire sequence/operation always yields a result that is not changed by a re-execution of all, or part, of that sequence/operation. In other words a sequence that never has side effects is idempotent, by definition (provided that no concurrent operations are being executed on the same set of resources).

**Safe Methods:**
➔The convention has been established that the GET and HEAD methods SHOULD NOT have the significance of taking an action other than retrieval.
➔These methods ought to be considered "safe". This allows user agents to represent other methods, such as POST, PUT and DELETE, in a special way, so that the user is made aware of the fact that a possibly unsafe action is being requested.

**GET:**
➔GET is a read-only operation and we should not use update operation using the GET because it the standards of Http protocols so if follow these we can get the Http features to our RESTful.
It is used to query the server for specific information. It is both an idempotent and safe operation.

**PUT:**
➔It is usually modelled as an insert or update. But most of the times it used for Update operation only but not for insert. In order to update a resource we need to send the identity as part of the PUT req.

➔PUT requests that the server store the message body sent with the request under the location provided in the HTTP message. It is also idempotent. When using PUT, the client knows the identity of the resource it is creating or updating. It is idempotent because sending the same PUT message more than once has no effect on the underlying service. An analogy is an MS Word document that you are editing. No matter how many times you click the Save button, the file that stores your document will logically be the same document.

➔In PUT always must we need to send identity to update the existing resource but in case of GET we don't to send the identity because we are doing read only operation.

➔For example if we wanted to update the order we need to send the order-id and to do this update we need to send PUT request with order-id over the URL.

➔If we wanted to update the logo that is there in the Httpd-Server we need to use PUT.

**DELETE:**
➔DELETE is used to remove resources.
➔It is idempotent as well because once we send the request to the delete then it will deletes the resource at the server side and if we send the request with same data again or multiple times then it will not affect server again because resource already has been deleted hence there is no effect for consecutive requests that are coming with same data hence it is Idempotent.

**POST:**
➔It is used to create a new resource.
➔POST is the only Non-Idempotent and unsafe operation of HTTP.
➔Each POST method is allowed to modify the service in a unique way that means always it will creates a new resource for each and every request in the server side hence it is not Idempotent. We may or may not send information with the request and we may or may not receive information from the response.

**HEAD:**
➔HEAD is exactly like GET except that instead of returning a response body, it returns only a response code and any headers associated with the request.

**OPTIONS:**
➔OPTIONS is used to request information about the communication options of the resource you are interested in.
➔It allows the client to determine the capabilities of a server and a resource without triggering any resource action or retrieval.
➔That means it used to for getting the application details like content type of the resource.
➔We can use any methods for any operation but Http recommends to use only specific operations for specific methods that meansStandard methods for Standards operations only So that we can achieve the best modelling in the RESTful service.
➔There are other HTTP methods (like TRACE and CONNECT), but they are unimportant when designing and implementing RESTful web services.

## What are the Idempotent Http methods and what are Non-Idempotent?

**Http Response Status codes:**
➔Whenever we sent a request asking for the response form the server then always it will not be success that means in order to distinguish success and failure of the response we will use Http-Status codes.
➔In case of Web Services we will uses a fault in case of failure but in case of success faults will not be there similarly in RESTful also we have Http-Status codes which will give the info about the success and failure of the server response.
➔Method to method the status codes will differ because each method serves different purpose of operation.
➔HTTP, Hypertext Transfer Protocol, is the method by which clients (i.e. you) and servers communicate.
➔When someone clicks a link, types in a URL or submits out a form, their browser sends a request to a server for information.
➔It might be asking for a page, or sending data, but either way, that is called an HTTP Request.
➔When a server receives that request, it sends back an HTTP Response, with information for the client.

➔Usually, we will get common Response codes - 404, indicating a page was not found.
➔There are a fair few more status codes sent by servers, and the following is a list of the current ones in HTTP 1.1, along with an explanation of their meanings.
➔HTTP Response Codes indicate whether a specific HTTP requests has been successfully completed. Responses are grouped in five classes which are called as Types of Http Response

**Status codes:**
➔Types of Http response Status codes:

1. Informational response Status codes
2. Successful response Status codes
3. Redirection response Status codes
4. Client Error response Status codes
5. Server Error response Status codes

➔The HTTP response status codes for the GET (retrieve), POST (create), PUT (modify/update), and DELETE operations are given below

**GET:**
➔The conditional GET method is intended to reduce unnecessary network usage by allowing cached entities to be refreshed without requiring multiple requests or transferring data already held by the client.
➔The response to a GET request is cacheable if and only if it meets the requirements for HTTP caching

**POST:**
➔The action performed by the POST method might not result in a resource that can be identified by a URI.
➔In this case, either 200 (OK) or 204 (No Content) is the appropriate response status.
➔If a resource has been created on the origin server, the response SHOULD be 201 (Created) and contain an entity which describes the status of the request and refers to the new resource, with URI or Location of the resource to access.
➔Responses to this method are not cacheable, unless the response includes appropriate Cache-Control or Expires header fields. However, the 303 response can be used to direct the user agent to retrieve a cacheable resource.

**Summary:**
➔200 (OK)-Operation success and there in no response to display but it can displays status success (OK) message

➔204 (No Content)-Operation was success and there in no response to display (No Content)

➔201 (Created)-New resource has been created on the origin server and sends response in response body content as the Resource that was created and along with URI to access that resource.

➔303 (Cache)-Client can get or retrieve the response from a cacheable resource at the client side.

**PUT:**
➔It is used for to create/update the resource.
➔The PUT method requests that the enclosed entity be stored under the supplied Request-URI. ➔If the Request-URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server.
➔If the Request-URI does not point to an existing resource, and that URI is capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI.
➔If a new resource is created, the origin server MUST inform the user agent via the 201 (Created) response.

➔If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request.
➔If the resource could not be created or modified with the Request-URI, an appropriate error response SHOULD be given that reflects the nature of the problem. That means the request not understand or no implemented so it will MUST return a 501 (Not Implemented) response.

**Summary:**
➔200 (OK)-Existing resource updated successfully and there in no response to display but it can displays as status success (OK) message

➔204 (No Content)-Existing resource updated successfully and there in no response to display (No Content)

➔201 (Created/Update)-If requested resource is not available to update then it it serves a New resource has been created on the origin server and sends response in response body content as the Resource that was created and along with URI to acess that resource.

➔501- Simply we can say Requested HTTP operation not supported.

**DELETE**
➔The DELETE method requests that the origin server delete the resource identified by the Request-URI.
➔This method MAY be overridden by human intervention (or other means) on the origin server.
➔The client cannot be guaranteed that the operation has been carried out, even if the status code returned from the origin server indicates that the action has been completed successfully. However, the server SHOULD NOT indicate success unless, at the time the response is given, it intends to delete the resource or move it to an inaccessible location.

➔200 (OK)-The request was processed successfully, but no response body is needed but displays status successful (OK) as response

➔202 (Accepted)-If the action has not yet been enacted (completed)

➔204 (No Content)-If the action has been enacted (completed) but the response does not include an entity.

➔If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries SHOULD be treated as stale. Responses to this method are not cacheable.

# JAX-RS API(Restful Service)

➔JAX-RS Stands for Java API for XML Restful service.

➔**REST is a new architectural style that defines set of rules that can be applied on distributed system, which will lead to interoperable distributed application.**

➔**Rest was first introduced by Roy Fielding in 2000,** who is one of the principal authors of the HTTP Specification and co-founder of Apache HTTP Server.

➔REST stands for Representational State Transfer which is built to work best on Web. In the REST technology information on the server is considered as Resource which developers can access in a Uniform way using Web URI's and HTTP .

➔REST uses HTTP as its communication protocol. It has two parts Client and Server.

➔If you want to get the Information from the Server you need to access via HTTP Protocol. **REST technology exposes data and its functionality as resources to the client's.**

➔**As it uses HTTP protocol, anything that is exposed over HTTP is called a Resource.**

➔The resources are generally exposed as static resources (it could be file, image, css or JavaScript etc) which can be accessed by simple http methods like GET, POST, DELETE and PUT.

➔**In case of Web Services, even those are using the HTTP protocol internally you cannot access them as simple Web resources over the HTTP protocol directly. Unlike those REST exposes your functionality as services over HTTP as simple Web resources without needing of any SOAP.**

➔REST allows resources to have different representations like text, html, xml, and json etc. The rest client can ask specific representation via the HTTP protocol (content negotiation).

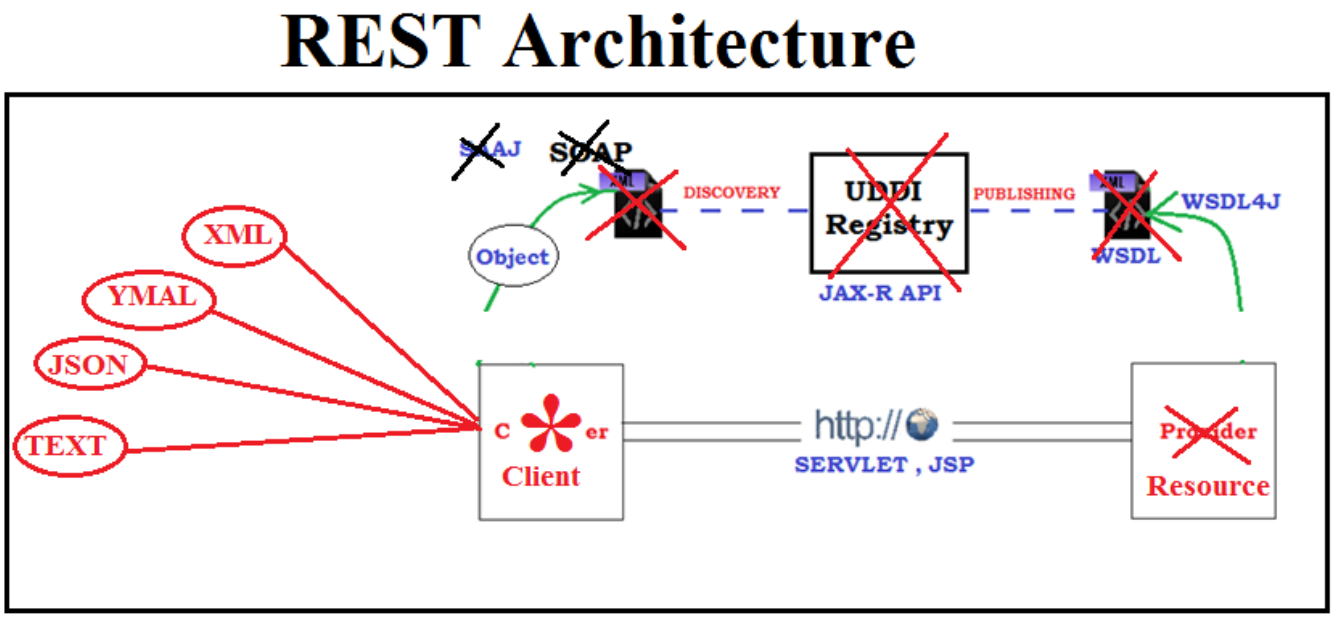
- ➔GET – Defines reading of the resources without side-effects. The resource is never changed via GET request.
- ➔PUT – creates a new resource
- ➔POST – updates existing resource
- ➔DELETE – removes the resource

So, if you observe the HTTP Methods are representing the CURD operations like create, update, delete and retrieve.

## JAVA Support for REST

➔Java has provided JAX-RS API to develop Restful services.
➔Initially it has released JAX-RS 1.1 API ,Later on it revised and released on more API JAX-RS 2.0 .
➔For which there are several implementation like JERSEY(Sun/Oracle) and RESTEASY(JBoss).

## Introduction to Rest:




**Rest Architecture:-**

➔There are two parts of the Restful services.
1.Client
2.Resourse class

**Client:**

➔Client is anyone who can send the request over http protocol, because there is no message exchanging format or no such additional parameter are provided in the request format.
➔Only the http request is required. Because the total rest architecture is Http centric. So the interoperable is more. Client can be in any technology or any platform.

**Resource:**

➔It is not called as provider just like in webservices because it does not contain any standard, any specification so this is called recourse.
➔It is called Web Recourse because we build it on web protocol principle.

### Q.How can my client and recourse class can communicate with each other?

➔For communicate in between client and recourse class we need a network.
➔The network must on should be a http protocol. We cannot use any other protocol.
➔We can send the data as an object over the http protocol network to the recourse class.
➔No need of SOAP over XML to send the data.

➔There is fixed set of methods are present inside the recourse class that is well known to others and no message exchanging formats are there so no need of WSDL is in restful services.
➔As there is no WSDL so no UDDI registry are required.

**HTTP(Hypertext Transfer Protocol):**

➔The http contains seven different methods these are
1.GET 2.POST 3.PUT 4.HEAD
5..DELET 6.TRACE 7.OPTIONS

➔Among these seven only 4 methods are supported by the restful webservices.
these methods are GET,POST,PUT,DELET

## STATUS CODE OF HTTP PROTOCOL:-

➔Status code shows the server response.
➔It is generally a three digit number, where the first digit show about the “Status-Code defines the class of response" and the last two digit does not defines any categorization role.
➔Some status code:

| | |
|---|---|
| 200 | Ok (The request is OK) |
| 201 | The request is complete, and a new resource is created. |
| 204 | A status code and a header are given in the response, but there is no entity-body in the reply.(content) |
| 303 | The requested page can be found under a different url |
| 404 | The server can not found the request page. |

The most popular rest API vender (3rd party) which provide standard documentation are
1.Ramel
2. Swagger

## Flow of Restful webservices:

### Q.How request will manage?

➔When the client send the request the request is coming through an client side socket.
➔The data is communicated through an interconnected network over an http protocol.
➔The data is received by the server side socket. And then it goes to recourse class.
➔The data flow in between the client and resource is in the form of "Stream of data" and to extract the data we use object stream of data
➔The resource class contains methods which have complex business logic within it.

### Q.How can you extract or read the data from http request?

➔In servlet generally we use request.getParameter generally in restful services we cannot read the data or extract by “request.getParameter"
➔we use "request.getParameter" in two cases i.e when we used form data or query string. In rest browser set these entire input control name in a specific form and send into the server.
➔In two ways client can send the request to the Resource class

1. By submitting form
2.URL form encoding

➔In rest client is creating the request form manually in the request body part, not submitting data in the form. Stream is the abstraction of java application to the world.

### Q.Why we cannot call the servet as web resource class?

➔Yes we can, but here the problem is servlet can receive input stream of data which is raw representation of data.
➔Which violate the representation orientation principle of restful which means that web resource can receive any type of request but servlet can't.
➔JAX-RS API gives or provides several classes and interfaces that can makes easy to obey all the principles.

## Difference between JERSEY and RESTEasy Implementations:

### RESTEasy:

JBossRESTeasy is an Implementation for the JAX-RS API and that works with only JBoss Application Server only out deployments will not supports that means if we develop the application using JBoss it will not works in other containers and in order to work without side deployments servers JBoss has provided some additional jar dependencies. Even though they provided the jar dependencies to deploy in any container but it most of the features works better with JBoss only like security works better with JBoss server only.

When we are working RESTeasy with JBoss Application server we no need to download binary distribution bczJBoss itself by default contains the binaries.

### JERSEY:

JERSEY is an Reference Implementation that is given by Sun Ms. (Oracle) and it works with any Application server or Servlet container (like Tomcat, Web Logic, WebSphere, Glassfish, JBossetc) bcz JERSEY is completely JAX-RS API compliant that is the reason industry preferred to work with JERSEY rather JBoss.

Glassfish server by default supports for the JERSEY Implementation hence when we are working with Glassfish we no need to download the binary distribution.

Why you are using JERSEY?

(i) It is completely JAX-RS API complaint.

(ii) It works with any application server that means agnostic to the application server.

## JBOSS RestEasy Implementation

### Resyeasy Approach-1(Option-1)

```
@Path("/plan")
public class PlanResource {

        @Produces("text/plain")
        @GET
        public String getPlanName(@QueryParam("planNo")int planNo){
                return "Plan No :"+planNo;
        }
}
```

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
  <display-name>RestApp7-1</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
        <servlet>
            <servlet-name>resteasy</servlet-name>
            <servlet-class>org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher</servlet-class>
            <init-param>
                <param-name>resteasy.scan</param-name>
                <param-value>true</param-value>
            </init-param>
            <load-on-startup>1</load-on-startup>
        </servlet>
        <servlet-mapping>
            <servlet-name>resteasy</servlet-name>
            <url-pattern>/*</url-pattern>
        </servlet-mapping>
</web-app>
```

**http://localhost:8086/RestApp7-1/plan?planNo=11**

**Environment**

**Server:JBoss AS 7.1.1**
**RestEasyJars:resteasy-jaxrs-3.0.5**
**JRE:1.6/1.7**
**Servlet:2.5**

## Resyeasy Approach-1(Option-2)

**You can add prefix also so that you can filter all the request coming to runtime servlet**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5">
 <display-name>RestApp7-1</display-name>
 <welcome-file-list>
  <welcome-file>index.html</welcome-file>
 </welcome-file-list>
      <servlet>
            <servlet-name>resteasy</servlet-name>
            <servlet-class>org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher</servlet-class>
            <init-param>
                  <param-name>resteasy.scan</param-name>
                  <param-value>true</param-value>
            </init-param>
            <init-param>
                  <param-name>resteasy.servlet.mapping.prefix</param-name>
                  <param-value>/api</param-value>
            </init-param>
            <load-on-startup>1</load-on-startup>
      </servlet>
      <servlet-mapping>
            <servlet-name>resteasy</servlet-name>
            <url-pattern>/api/*</url-pattern>
      </servlet-mapping>
</web-app>
```

**http://localhost:8086/Resteasy-II/api/plan?planNo=11**

➔At the time of the creating the object of HttpServletDispatcher init() will be called .
➔Inside init() it read these two parameters
➔If you don't write resteasy.scan then it will understand you don't want resource to be expose. So unless until you write resteasy.scan there is no resource will be exposed.
➔ So rather than going to each and every class if you remove @path then resteasy.scan=false

## 2nd Approach(Option-1)

```
@Path("/plan")
public class PlanResource {
	@GET
	@Produces("text/plain")
	public String getPlanName(@QueryParam("planNo")int planNo){
		return "Plan No :"+planNo;
	}
}
```

```
@ApplicationPath("/")
public class PlanResourceApplication extends Application{}
```

**http://localhost:7777/RestfulFirstApplication/plan?planNo=11**

**Environment**
**Server:wildfly-10.1.0**
**RestEasy Jar:resteasy-jaxrs-3.1.1**
**JRE:1.8**
**Servlet:3.0**

**Benefits**

➔This approach is implementation vendor independent
➔So across all the implementation it will going to work.
➔JBoss & Wildfly both it will work.
➔All the servlet container all the implementation it will work.

## 2nd Approach(Option-2)

```
@Path("/plan")
public class PlanResource {
        @GET
        @Produces("text/plain")
        public String getPlanName(@QueryParam("planNo")int planNo){
                return "Plan No :"+planNo;
        }
}
```

```
//@ApplicationPath("/")
public class PlanResourceApplication extends Application{}
```

```
<web-app >
 <display-name>RestfulApplication-1</display-name>
 <servlet>
        <servlet-name>resteasyApp</servlet-name>
        <servlet-class>com.rfa.applications.PlanResourceApplication</servlet-class>
 </servlet>
 <servlet-mapping>
        <servlet-name>resteasyApp</servlet-name>
        <url-pattern>/*</url-pattern>
 </servlet-mapping>
</web-app>
```

**Environment**
**Server:wildfly-10.1.0**
**RestEasy Jar:resteasy-jaxrs-3.1.1**
**JRE:1.8**
**Servlet:3.0**

**Internals**

➔It will go and checks classpath is there any clas annotated with @ApplicationPath (No).
➔Then it will goes to web.xml and search is there any servlet that is being configure with Application class .
➔Resteasy servlet container can identify is it servlet or application class because that class is extending from Application.
➔Quickly read the configuration and with this configure HttpServletDispatcher with /*

## 2nd Approach(Option-3)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0">
  <display-name>RestfulApplication-1</display-name>
  <servlet>
        <servlet-name>resteasyApp</servlet-name>
        <servlet-class>javax.ws.rs.core.Application</servlet-class>
  </servlet>
  <servlet-mapping>
        <servlet-name>resteasyApp</servlet-name>
        <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

## 2nd Approach(Option-4)

```
@Path("/plan")
public class PlanResource {
        @GET
        @Produces("text/plain")
        public String getPlanName(@QueryParam("planNo")int planNo){
                return "Plan No :"+planNo;
        }
}
```

```
import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;
@ApplicationPath("/rest")
public class PlanResourceApplication extends Application{}
```

http://localhost:7777/RestfulApplication-2/rest/plan?planNo=11

## 2nd Approach(Option-5)

```
<web-app  version="3.0">
 <display-name>RestfulApplication-1</display-name>
 <servlet>
       <servlet-name>resteasyApp</servlet-name>
       <servlet-class>com.rfa.applications.PlanResourceApplication</servlet-class>
       <init-param>
                  <param-name>resteasy.servlet.mapping.prefix</param-name>
                  <param-value>/rest</param-value>
            </init-param>
            <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
       <servlet-name>resteasyApp</servlet-name>
       <url-pattern>/rest/*</url-pattern>
 </servlet-mapping>
</web-app>
```

**Benefits**

➔With out modifying any class also I can change url's

# 3rd Approach (Option-1)

## Managing the Scope (Singleton) and restrict the Resources to expose

```
@Path("/plan")
public class PlanResource {
	@GET
	@Produces("text/plain")
	public String getPlanName(@QueryParam("planNo")int planNo){
		return "Plan No :"+planNo+"-"+this.hashCode();
	}
}
```

```
@ApplicationPath("/")
public class PlanResourceApplication extends Application{
	private Set<Object> singletons;
	private Set<Class<?>> classes;
	public PlanResourceApplication() {
		singletons = new HashSet<Object>();
		classes = new HashSet<Class<?>>();
		singletons.add(new PlanResource());
		System.out.println("Inside PlanResourceApplication()");
	}
	@Override
	public Set<Object> getSingletons() {
		System.out.println("Inside getSingletons()");
		return singletons;
	}
	@Override
	public Set<Class<?>> getClasses() {
		System.out.println("Inside getClasses()");
		return classes;
	}
}
```

**Inside PlanResourceApplication()**

**Inside getClasses()**
**Inside getClasses()**
**Inside getSingletons()**
**Inside getSingletons()**

**Output**

**Plan No :15-13201248**
**Plan No :15-13201248**
**Plan No :15-13201248**

## Internals

➔**Resteasy servlet container initialize will loads the HttpServletDispatcher.**

➔Application class contains the complete information about all the resources that are there in my application
➔Application class is coming form jax-rs api . and it is concrete class.
➔We need to override 2 methods
getSingletons()
getClasses()
➔These two are contractual methods.
➔Then we need to decide which class we need singletone which class we need to non singletone.
➔Based on which we need to set the objects or set of classes.
➔by default these methods return empty.
➔HttpServletDispatcher will create the object only once.
➔ant will call getSingleton() any no of time.
➔If you want same object to be return then instantiate the HashSet object inside the constructor so that it will created once because PlanResourceApplication class will created only once because init() will instantiate this object.
➔If you have any non singleton class then add to getClasses ().
➔If you write only empty class without overriding getSingletons() then internally in configuration it will take resteasy.sacn=true.(means all the classes it will expose).
➔If you want few classes singleton few want to be non singleton then
➔add singletons to getSingleton()
➔add non singletone to getClasses()
➔because if you overwride getSingleton method resteasy.sacn will not work we need to manually add to getClassed()

## 3rd Approach (Option-2)

Managing the Scope (Non-Singleton)/Request Scope

```
@Path("/plan")
public class PlanResource {
        @GET
        @Produces("text/plain")
        public String getPlanName(@QueryParam("planNo")int planNo){
                return "Plan No :"+planNo+"-"+this.hashCode();
        }
}
```

```
@ApplicationPath("/")
public class PlanResourceApplication extends Application{}
```

**Plan No: 15-1447880**
**Plan No: 15-24707617**
**➔By default it is request scope if we are not overriding getSingletons().**

# JAX-RS RI Implementation(Jersey)

## Approach-1

```
@Path("/plan")
public class PlanResource {
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	public String getPlanDetails(@QueryParam("planNo")int planNo){
		return "Plan No:"+planNo;
	}

}
```

```
<web-app version="3.0">
    <servlet>
        <servlet-name>jersey</servlet-name>
        <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>jersey.config.server.provider.packages</param-name>
            <param-value>com.jfa.resources</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>jersey</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

**http://localhost:8085/JerseyFirstApp/plan?planNo=15**

**Server: Tomcat 7 /Tomcat 8**
**Servlet: 3.0Version (Work with any servlet version)**
**Jdk-1.7/1.8**
**Jax-ri jars: jaxrs-ri-2.4.1**

➔Servlet container will go to the package and scan for all the class
➔If the classes are annotated with @Path then it reads that annotations and configure in metadata.
➔When the request comes to application then tomcat servlet container will forward the request to jersey ServletContainer.
➔But Jersey servlet container will not dispatches the request to our resource. Because by default autoscan is not there.

**Approach-2**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0">
    <servlet>
        <servlet-name>jersey</servlet-name>
        <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>jersey.config.server.provider.packages</param-name>
            <param-value>com.jfa.resources</param-value>
        </init-param>
        <init-param>
            <param-name>jersey.config.server.provider.scanning.recursive</param-name>
            <param-value>true</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>jersey</servlet-name>
        <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>
</web-app>
```

**http://localhost:8085/JerseyFirstApp/rest/plan?planNo=15**

**Approach-3**

```
@Path("/plan")
public class PlanResource {
     @GET
     @Produces(MediaType.TEXT_PLAIN)
     public String getPlanDetails(@QueryParam("planNo")int planNo){
          return "Plan No:"+planNo;
     }

}
```

```
@Path("/product")
public class ProductResource {
     @GET
     @Produces(MediaType.TEXT_PLAIN)
     public double getProductPrice(@QueryParam("product-code")String productCode){
          return 33.67;
     }
}
```

```
<web-app version="3.0">
    <servlet>
        <servlet-name>jersey</servlet-name>
        <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>jersey.config.server.provider.classnames</param-name>
            <param-value>com.jfa.resources.PlanResource;com.jfa.resources.ProductResource</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>jersey</servlet-name>
        <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>
</web-app>
```

http://localhost:8085/JerseyFirstApp/rest/product?product-code=SJ-211

http://localhost:8085/JerseyFirstApp/rest/plan?planNo=15

**Approach-4**

```
@Path("/plan")
public class PlanResource {
     @GET
     @Produces(MediaType.TEXT_PLAIN)
     public String getPlanDetails(@QueryParam("planNo")int planNo){
          return "Plan No:"+planNo;
     }

}
```

```
@ApplicationPath("/")
public class ResourceApplication extends Application {}
```

```
http://localhost:8085/JerseyFirstApp/plan?planNo=15
```

**Approach-5**

```
@ApplicationPath("/rest")
public class ResourceApplication extends Application {}
```

```
http://localhost:8085/JerseyFirstApp/rest/plan?planNo=15
```

**Approach-6**

```
<web-app version="3.0">
     <servlet>
          <servlet-name>com.jfa.applications.ResourceApplication</servlet-name>
     </servlet>
     <servlet-mapping>
          <servlet-name>com.jfa.applications.ResourceApplication</servlet-name>
          <url-pattern>/rest/*</url-pattern>
     </servlet-mapping>
</web-app>
```

```
//@ApplicationPath("/rest")
public class ResourceApplication extends Application {

}
```

**http://localhost:8085/JerseyFirstApp/rest/plan?planNo=15**

**Approach-7**

```
<web-app version="3.0">
    <servlet>
        <servlet-name>javax.ws.rs.core.Application</servlet-name>
    </servlet>
    <servlet-mapping>
        <servlet-name>javax.ws.rs.core.Application</servlet-name>
        <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>
</web-app>
```

➔**This is read by jersey ServletContextInitialize.**

## Q.What is the difference between config and Application?

➔When we change from resteasy to jersy , resteasy people provide "httpServletDispatcher" but in jersy the servlet is different and it is called as "servletContainer".
➔The init parameters also changed from one vender to another vender. so the total configuration file also changed. So it is implementation vender specific bootstrapping.
➔If we go for Application then it is not implementation vender specific.becouse this class is coming from jax-rs api not from resteasy or jersy.

## Q. Managing scope of resources?

➔The scope of the recourses are managed by the developer. we have to say to the jax-rs runtime to create one object for all the request or for each and every request one object.
➔As per jax-rs api programmer has to manage the scope of resources manually by using web.xml whether you want one object or multiple object for one request.
➔Generally two types of scope is there "scope to request" or "scope to the application" .(request and singleton).

**Approach-8**

```
@ApplicationPath("/rest")
public class ResourceApplication extends Application {
     private Set<Class<?>> classes;
     private Set<Object> singletons;

     public ResourceApplication() {
          classes = new HashSet<Class<?>>();
          singletons = new HashSet<Object>();
          singletons.add(new PlanResource());
          classes.add(ProductResource.class);
     }

     @Override
     public Set<Class<?>> getClasses() {
          return classes;
     }

     @Override
     public Set<Object> getSingletons() {
          return singletons;
     }
}
```

**Approach-9**

```
@ApplicationPath("/")
public class ResourceApplication extends ResourceConfig{

     public ResourceApplication() {
          register(ProductResource.class); //Non Singleton (scope=request)
          register(new PlanResource());     //Singleton
     }

}
```

➔ResourceConfig class extending from Application and overridden all the methods as empty Set.
➔It has provided methods

**register(Object) //registerInstances(Object,Object….)**

➔This is for singleton classes

**register(Class) //register Classes(Class,Class,Class…)**

➔This is for Non singleton classes
➔If you pass the class/Object it will add those values to HashSet.
➔This approach is only in Jersey implementation

## Approach-10

Make a resource as singleton without using annotation.

```
@Path("/plan")
public class PlanResource {
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	public String getPlanDetails(@QueryParam("planNo")int planNo){
		return "Plan No:"+planNo+"-"+this.hashCode();
	}
}
```

```
//@ApplicationPath("/")
public class MyApplication extends Application{
	private Set<Object> singletons;
	public MyApplication() {
		singletons=new HashSet<>();
		singletons.add(new PlanResource());
	}
	@Override
	public Set<Object> getSingletons() {
		return singletons;
	}
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0">
	<servlet>
		<servlet-name>jersey</servlet-name>
		<servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
		<init-param>
			<param-name>javax.ws.rs.Application</param-name>
			<param-value>com.jfa.applications.MyApplication</param-value>
		</init-param>
		<load-on-startup>1</load-on-startup>
	</servlet>
	<servlet-mapping>
		<servlet-name>jersey</servlet-name>
		<url-pattern>/*</url-pattern>
	</servlet-mapping>
</web-app>
```

```
Plan No: 159-30542472
Plan No: 159-30542472
```

## Create a Application Using All the HTTP Methods (FreechargeWallet)

```
@Path("/freecharge")
public class FreeChargeWallet {
     private static Map<String,Account> accountMap=null;
     private Account accountDto=null;
     @POST
     @Consumes(MediaType.APPLICATION_XML)
     @Produces(MediaType.APPLICATION_XML)
     public StreamingOutput register(InputStream io) throws ParserConfigurationException, SAXException, IOException{
          Account account=buildRegister(io);
          OutputStreamRegisterHandler osh=new OutputStreamRegisterHandler(account.getAccountNo(),account.getMobileNo());
          return osh;
     }
```

### POST Operation

```
public Account buildRegister(InputStream io) throws ParserConfigurationException, SAXException, IOException{
          DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
          DocumentBuilder builder=factory.newDocumentBuilder();
          Document accountDoc=builder.parse(io);
          Node firstNode=null;
          NodeList list=null;
          Node node=null;
          if (accountDoc!=null){
               firstNode=accountDoc.getFirstChild();
               list=firstNode.getChildNodes();
               accountDto=new Account();
               accountMap=new ConcurrentHashMap<String,Account>();

               for (int i = 0; i < list.getLength(); i++) {
                    node=list.item(i);

                    if (node.getNodeType()==Node.ELEMENT_NODE) {
                         String name=node.getNodeName();
                         String value=node.getFirstChild().getTextContent();
                         if (name.equals("member-name")) {
                              accountDto.setMemberName(value);
                         }
                         if (name.equals("email-address")) {
                              accountDto.setEmailAddress(value);
                         }
                         if (name.equals("mobile-no")) {
                              accountDto.setMobileNo(value);
                         }
                         if (name.equals("password")) {
                              accountDto.setPassword(value);
                         }
                         if (name.equals("balance")) {
                              accountDto.setBalance(Float.parseFloat(value));
                         }
                    }
               }
          }
          accountDto.setAccountNo(UUID.randomUUID().toString());
          accountMap.put(accountDto.getMobileNo(),accountDto);
          return accountDto;
     }
```

```
private class OutputStreamRegisterHandler implements StreamingOutput{
          private String accountNo=null;
          private String mobileNo=null;
          public OutputStreamRegisterHandler(String accountNo, String mobileNo) {
               this.accountNo = accountNo;
               this.mobileNo = mobileNo;
          }
          @Override
          public void write(OutputStream os) throws IOException, WebApplicationException {
               StringBuffer buffer=new StringBuffer();
               buffer.append("<account-info>").append("<account-no>")
               .append(accountNo).append("</account-no>")
               .append("<mobile-no>").append(mobileNo)
               .append("</mobile-no>").append("</account-info>");

               os.write(buffer.toString().getBytes());
               os.close();
          }
     }
```

**Input Post Request**

```
<?xml version="1.0" encoding="utf-8"?>
<register>
     <member-name>Dhananjaya</member-name>
     <mobile-no>9040010697</mobile-no>
     <email-address>mail4dhananjaya@gmail.com</email-address>
     <password>realspeed</password>
     <balance>0</balance>
</register>
```

**Out Put Request**

```
<account-info>
    <account-no>bfdd70c1-df16-438c-bd1f-3cf5afa7500d</account-no>
    <mobile-no>9040010697</mobile-no>
</account-info>
```

**PUT Operation**

```
@PUT
@Consumes(MediaType.APPLICATION_XML)
@Produces(MediaType.APPLICATION_XML)
public StreamingOutput addCash(InputStream is) throws Exception{
     accountDto=getAddCash(is);
     OutputStreamAddCashHandler addCashHandler=new
                         OutputStreamAddCashHandler(accountDto.getAccountNo(),
                              accountDto.getMobileNo(), accountDto.getBalance());
     return addCashHandler;
}
```

```
public Account getAddCash(InputStream is) throws Exception{
          DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
          DocumentBuilder builder=factory.newDocumentBuilder();
          Document addCashDoc=builder.parse(is);
          Node firstNode=null;
          NodeList list=null;
          Node node=null;
          if (addCashDoc!=null) {
               firstNode=addCashDoc.getFirstChild();
               list=firstNode.getChildNodes();
               String name=null;
               String value=null;
               for (int i = 0; i < list.getLength(); i++) {
                    node=list.item(i);
                    if (node.getNodeType()==Node.ELEMENT_NODE) {
                         name=node.getNodeName();
                         value=node.getFirstChild().getTextContent();
                         if (name.equals("mobile-no")) {
                              if (accountMap.containsKey(value)) {
                                   accountDto=accountMap.get(value);
                              }
                              else{
                                   throw new Exception("Mobile No is not valid");
                              }
                         }
                         if (name.equals("balance")){
                              accountDto.setBalance(Double.parseDouble(value));
                         }
                    }

               }
          }
          return accountDto;
     }
```

```
private final class OutputStreamAddCashHandler implements StreamingOutput{
     private String accountNo=null;
     private String mobileNo=null;
     private double balance=0.00;


     public OutputStreamAddCashHandler(String accountNo, String mobileNo, double balance) {
          this.accountNo = accountNo;
          this.mobileNo = mobileNo;
          this.balance = balance;
     }
     @Override
     public void write(OutputStream os) throws IOException, WebApplicationException {
          StringBuffer buffer=new StringBuffer();
          buffer.append("<add-cash-info>").append("<mobile-no>").append(mobileNo).append("</mobile-no>")
                    .append("<account-no>").append(accountNo).append("</account-no>").append("<balance>")
                    .append(balance).append("</balance>").append("</add-cash-info>");
          os.write(buffer.toString().getBytes());
          os.close();
     }
}
```

**Input PUT Request**

```
<?xml version="1.0" encoding="utf-8"?>
<add-cash>
     <mobile-no>9040010697</mobile-no>
     <card-no>1345-0998-1335-9867</card-no>
     <account-holder-name>Dhananjaya Samanta Singhar</account-holder-name>
     <cvv>622</cvv>
     <expiry>12/18</expiry>
     <balance>3068</balance>
</add-cash>
```

**Output PUT Request**

```
<add-cash-info>
     <mobile-no>9040010697</mobile-no>
     <account-no>bfdd70c1-df16-438c-bd1f-3cf5afa7500d</account-no>
     <balance>3068.0</balance>
</add-cash-info>
```

**GET Operation**

```
@GET
@Produces(MediaType.TEXT_PLAIN)
public double getBalance(@QueryParam("mobile-no")String mobileNo) throws Exception{
     double balance=0.00;
     if (accountMap.containsKey(mobileNo)) {
          accountDto=accountMap.get(mobileNo);
          balance=accountDto.getBalance();
     }else{
          throw new Exception("Mobile No is not valid");
     }
     return balance;
}
```

**Input GET Request**
**http://localhost:8085/JerseySecondApp/freecharge?mobile-no=9040010697**

**Delete Operation**

```
     @DELETE
     @Produces(MediaType.TEXT_PLAIN)
     public String deActivate(@QueryParam("mobile-no")String mobileNo) throws Exception{
          if (accountMap.containsKey(mobileNo)) {
               accountMap.remove(mobileNo);
          }else{
               throw new Exception("Mobile no is not valid");
          }
          return "Account "+mobileNo+" De-activated";
     }
}
```

**Input DELETE Request**
**http://localhost:8085/JerseySecondApp/freecharge?mobile-no=9040010697**

# Annotation Inheritance

```
public abstract class FreechargeWallet {
     static Map<String,Account> accountMap=new ConcurrentHashMap<String,Account>();
     @POST
     @Consumes({MediaType.APPLICATION_XML,MediaType.TEXT_PLAIN})
     @Produces({MediaType.APPLICATION_XML,MediaType.TEXT_PLAIN})
     public StreamingOutput register(InputStream io)
                    throws ParserConfigurationException, SAXException, IOException{
          final Account account=buildAccount(io);
          account.setAccountNo(UUID.randomUUID().toString());
          accountMap.put(account.getMobileNo(),account);

          return new StreamingOutput() {
               @Override
               public void write(OutputStream os) throws IOException, WebApplicationException {
                    os.write(response(account));

               }
          };
     }
     public abstract Account buildAccount(InputStream io);
     public abstract byte[] response(Account account);
}
```

```
@Path("/m-freecharge")
public class MFreechargeWallet extends FreechargeWallet {
     @Override
     public Account buildAccount(InputStream io) {
          Account account=null;
          int ch=-1;
          ByteArrayOutputStream bos=new ByteArrayOutputStream();
          try {
               while ((ch=io.read())!=-1) {
                    bos.write(ch);
               }
               io.close();

               //Dhananjaya,9040010697,mail4dhananjaya@gmail.com,realspeed,0
               String allInputs=bos.toString();

               //input['Dhananjaya','9040010697','mail4dhananjaya@gmail.com','realspeed',0]
               String [] inputs=allInputs.split(",");

               account=new Account();

               account.setMemberName(inputs[0]);
               account.setMobileNo(inputs[1]);
               account.setEmailAddress(inputs[2]);
               account.setPassword(inputs[3]);
               account.setBalance(Double.parseDouble(inputs[4]));
               bos.close();
          } catch (IOException e) {
               e.printStackTrace();
          }
          return account;
     }
     @Override
     public byte[] response(Account account) {
          StringBuffer buffer=new StringBuffer();
          buffer.append(account.getAccountNo()).append(",").append(account.getMobileNo());
          return buffer.toString().getBytes();
     }
}
```

```
@Path("/b-freecharge")
public class B2BFreechargeWallet extends FreechargeWallet{
     @Override
     public Account buildAccount(InputStream io) {
          DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
          DocumentBuilder builder;
          Document accountDoc=null;
          try {
               builder = factory.newDocumentBuilder();
               accountDoc = builder.parse(io);
          } catch (SAXException | IOException |ParserConfigurationException e) {
               e.printStackTrace();
          }

          Node firstNode=null;
          NodeList list=null;
          Account accountDto=null;
          Node node=null;
          firstNode=accountDoc.getFirstChild();
          list=firstNode.getChildNodes();
          accountDto=new Account();
               for (int i = 0; i < list.getLength(); i++) {
                    node=list.item(i);

                    if (node.getNodeType()==Node.ELEMENT_NODE) {
                         String name=node.getNodeName();
                         String value=node.getFirstChild().getTextContent();
                         if (name.equals("member-name")) {
                              accountDto.setMemberName(value);
                         }
                         if (name.equals("email-address")) {
                              accountDto.setEmailAddress(value);
                         }
                         if (name.equals("mobile-no")) {
                              accountDto.setMobileNo(value);
                         }
                         if (name.equals("password")) {
                              accountDto.setPassword(value);
                         }
                         if (name.equals("balance")) {
                              accountDto.setBalance(Float.parseFloat(value));
                         }
                    }
               }
          return accountDto;
     }

     @Override
     public byte[] response(Account account) {
          StringBuffer buffer=new StringBuffer();
          buffer.append("<account-info>").append("<account-no>").append(account.getAccountNo()).append("</account-no>")
          .append("<mobile-no>").append(account.getMobileNo()).append("</mobile-no>").append("</account-info>");
          return buffer.toString().getBytes();
     }
}
```

Input XMl

```
<?xml version="1.0" encoding="utf-8"?>
<register>
     <member-name>Dhananjaya</member-name>
     <mobile-no>9040010697</mobile-no>
     <email-address>mail4dhananjaya@gmail.com</email-address>
     <password>realspeed</password>
     <balance>0</balance>
</register>
```

Output XML

```
<account-info>
    <account-no>cb593ac9-552c-4f0f-83d1-e34b3b90504c</account-no>
    <mobile-no>9040010697</mobile-no>
</account-info>
```

**InputCSV**

```
Dhananjaya,9040010697,mail4dhananjaya@gmail.com,realspeed,0
```

**Output CSV**

```
a7d60dbd-5b73-4c98-a271-da0e89766ce3,9040010697
```

## How JAX-RS resolve if two GET Request comes to one single resource?

### Resteasy Implementation

```
@Path("/doctor")
public class DoctorResource {
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	public String getContactNo(@QueryParam("doctor-name")String doctorName){
		return "9853001547";
	}

	@GET
	@Produces(MediaType.TEXT_PLAIN)
	public String getSpecialization(@QueryParam("doctor-name")String doctorName){
		return "Cardiology";
	}
}
```

```
<web-app version="3.0">
  <display-name>URI</display-name>
 <servlet>
    <servlet-name>javax.ws.rs.core.Application</servlet-name>
    <init-param>
        <param-name>resteasy.servlet.mapping.prefix</param-name>
        <param-value>/api</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
    <servlet-name> javax.ws.rs.core.Application</servlet-name>
    <url-pattern>/api/*</url-pattern>
 </servlet-mapping>
</web-app>
```

➔Multiple resource methods match request "GET /doctor". Selecting one.
http://localhost:7777/URI/api/doctor?doctor-name=Dhananjaya

Note:

1. In Resteasy Implementation, we can't expect which method will call it depends on vendor Most of the time resteasy calls the last matched uri

2. In Jersey Implementation it is not possible ,Server throwing an Exception ModelValidationException:Validation of the application resource model has failed during application initialization

## Sub Path

```
@Path("/doctor")
public class DoctorResource {
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	@Path("/contact")
	public String getContactNo(@QueryParam("doctor-name")String doctorName){
		return "9853001547";
	}

	@GET
	@Produces(MediaType.TEXT_PLAIN)
	@Path("specialization")
	public String getSpecialization(@QueryParam("doctor-name")String doctorName){
		return "Cardiology";
	}
}
```

http://localhost:7777/URI/api/doctor/contact?doctor-name=Dhananjaya

http://localhost:7777/URI/api/doctor/specialization?doctor-name=Dhananjaya

### Q.What is the other way to send the data rather than sending the data in Queryparam ?

➔We can send the data in URI without using QueryParameter.

### Q.How to send the data in the URI? How template works?

➔Write the path in the templete parameter.
➔The URI followed some standard conversion and the entered URI take as a string type.
As per JAX-RS it define that the URI input are given by the programmer and the additional placeholder are also provided by the progarammer (path(**"/uber/{x}/{y}"**)) here the X and Y are provided by the programmer and also the placeholder.
➔In this standard format we have to follow whenever we send the URL to the path.
➔We have to only give the URI as standard format and the JAX-RS make tokenized the URL find he value of X and Y. They must follow the position.
➔The position of the parameter are called **Template.**
➔Template means every component must on should be same format. rather than we tokenized the URI to find the input if we follow the Template we can easily fetch the data from the Recourse class.

## Template Parameter

**Ex-1**

```
@Path("/doctor/{dName}")
public class DoctorResource {
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  public String getContactNo(@PathParam("dName")String doctorName){
     return "9853001547";
  }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0">
  <display-name>TemplateParameters</display-name>
 <servlet>
     <servlet-name>javax.ws.rs.core.Application</servlet-name>
 </servlet>
 <servlet-mapping>
     <servlet-name> javax.ws.rs.core.Application</servlet-name>
     <url-pattern>/api/*</url-pattern>
 </servlet-mapping>
</web-app>
```

**http://localhost:8085/TemplateParameters/api/doctor/dName=Dhananjaya**

### Q.If you write the @Path at method level what is the method will be called? Or When do we call a method as sub resource method?

➔When a method annotated with @Path along with http method designator that particular method is called as resource method.

**Ex-2:**

```
@Path("/doctor")
public class DoctorResource {
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  @Path("/contact/{dName}")
  public String getContactNo(@PathParam("dName")String doctorName){
     return "9853001547";
  }

  @GET
  @Produces(MediaType.TEXT_PLAIN)
  @Path("/specialization/{dName}")
  public String getSpecializarion(@PathParam("dName")String doctroName){
       return "Cardiology";
  }
}
```

```
http://localhost:8085/TemplateParameters/api/doctor/specialization/dName=Dhananjaya

http://localhost:8085/TemplateParameters/api/doctor/contact/dName=Dhananjaya
```

### Q.When a class is called as Root Resourse class?

➔When the class is annotated with @Path then that class is called Root resource class.

## Sub Resourse Locator

### Q.Is it easy to write all the method inside one single resourse class?

➔**Yes** possible but there are lot of problem

1. mentainability of code difficult.
2. undersatanding the business logic will difficult.

➔**To** overcome from this problem we have to separate the method into several classes.

### Q. How can we break the class in to multiple classes?

➔Resource method can directly accessible by the root URI but the sub resource method cannot directly accessible by root URI it accessible by sub resource URI.

### Q. What is sub resource method?

➔Along with the root resource URI if we add sub resource URI in accessing the method over http method designator then it is called sub resource method.

➔This method cannot be access directly because it is not annotated with any http method designator.
➔In this method the resource method will identify.
➔It helps in locating the other class in which the resource method is there to whom we need to dispatch the request.

**Ex-1**

```
@Path("airtel")
public class AirtelService{

	@GET
	@Produces(MediaType.TEXT_PLAIN)
	@Path("/balance")
	public double getBalance(@QueryParam("mobile")String mobile){
		return 389.00;
	}

	@GET
	@Produces(MediaType.TEXT_PLAIN)
	@Path("/data")
	public String getDataPlan(@QueryParam("mobile")String mobile){
		return "300 MB";
	}

	@Path("/subscriber")
	public AirtelSubscriberService getSubscriber(){
		return new AirtelSubscriberService();
	}
}
```

```
public class AirtelSubscriberService {
	@GET
	@Produces(MediaType.TEXT_HTML)
	public String getSubscriber(@QueryParam("mobile")String mobile){
		return "Dhananjaya";
	}
}
```

```
@ApplicationPath("/")
public class MyApplication extends Application {}
```

http://localhost:7777/SubResourceLocator/airtel/subscriber?mobile=9437215211

http://localhost:7777/SubResourceLocator/airtel/balance?mobile=9437215211

http://localhost:7777/SubResourceLocator/airtel/data?mobile=9437215211

**Ex-2**

```
@Path("airtel")
public class AirtelService{

	@GET
	@Produces(MediaType.TEXT_PLAIN)
	@Path("/balance/{mobile}")
	public double getBalance(@PathParam("mobile")String mobile){
		return 389.00;
	}

	@GET
	@Produces(MediaType.TEXT_PLAIN)
	@Path("/data/{mobile}")
	public String getDataPlan(@PathParam("mobile")String mobile){
		return "300 MB";
	}

	@Path("/subscriber/{mobile}")
	public AirtelSubscriberService getSubscriber(){
		return new AirtelSubscriberService();
	}
}
```

```
public class AirtelSubscriberService {
	@GET
	@Produces(MediaType.TEXT_HTML)
	public String getSubscriber(@PathParam("mobile")String mobile){
		return "Dhananjaya";
	}
}
```

http://localhost:7777/SubResourceLocator/airtel/subscriber/mobile=9437215211

http://localhost:7777/SubResourceLocator/airtel/balance/mobile=9437215211

http://localhost:7777/SubResourceLocator/airtel/data/mobile=9437215211

## Dynamic Sub Resource Locator

```
@Path("airtel")
public class AirtelService{

     @GET
     @Produces(MediaType.TEXT_PLAIN)
     @Path("/balance/{mobile}")
     public double getBalance(@PathParam("mobile")String mobile){
          return 389.00;
     }

     @GET
     @Produces(MediaType.TEXT_PLAIN)
     @Path("/data/{mobile}")
     public String getDataPlan(@PathParam("mobile")String mobile){
          return "300 MB";
     }

     @Path("/{subscribe}/{mobile}")
     public Object subscribe(@PathParam("subscribe")String type){
          if(type.equals("4g"))
               return new AirtelDataPackSubscribeService();
          else if (type.equals("music"))
               return new AirtelMusicPackSubscribeService();
          return null;
     }
}
```

```
public class AirtelDataPackSubscribeService {
     @PUT
     @Produces(MediaType.TEXT_HTML)
     public String activate4g(@PathParam("mobile")String mobile){
          return "4G Activated in "+mobile;
     }
}
```

```
public class AirtelMusicPackSubscribeService {
     @PUT
     @Produces(MediaType.TEXT_HTML)
     public String activateMusic(@PathParam("mobile")String mobile){
          return "Devotional music pack activated in "+mobile;
     }
}
```

```
@ApplicationPath("/")
public class MyApplication extends Application {}
```

*http://localhost:7777/SubResourceLocator/airtel/4g/mobile=9437215211*
*http://localhost:7777/SubResourceLocator/airtel/music/mobile=9437215211*
*http://localhost:7777/SubResourceLocator/airtel/balance/mobile=9437215211*
*http://localhost:7777/SubResourceLocator/airtel/data/mobile=9437215211*

## JAX-RS injection

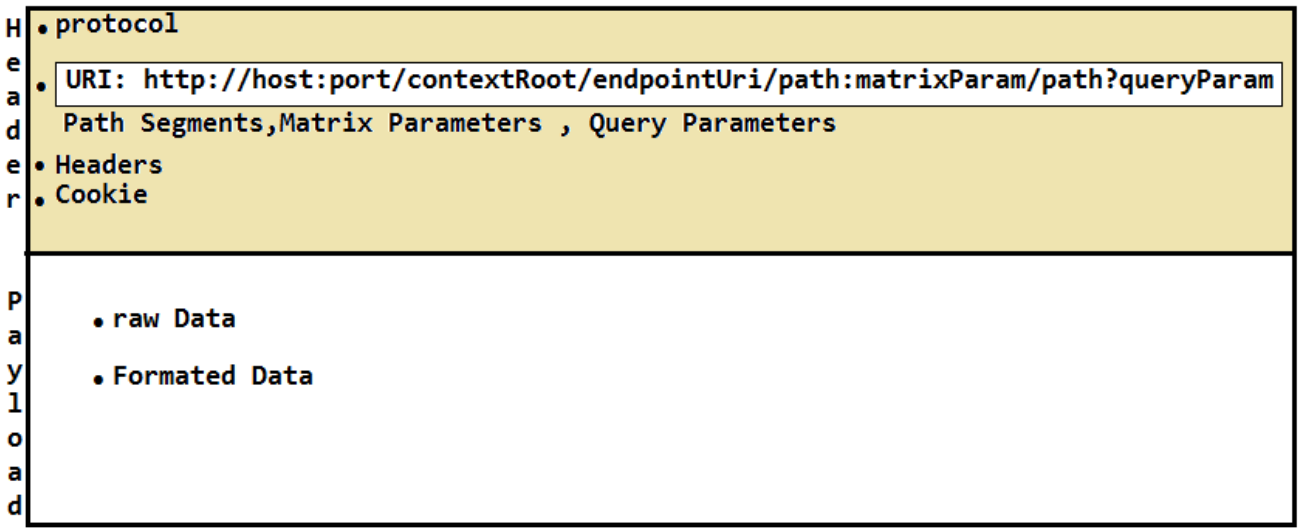
➔**This process of extracting the data from the resource is called JAX-RS injection.**
➔**Our** resource class is distributed and interoperable component and our resource class contains resource method and which contains business logic.
➔But our method needs data as input. we pass the data as input via a http request.
➔There are several parts in the http request part

1.header
2.Body

➔The header part contains several parts

1. headers data
2. URI
3. cookies

➔The body contains row data or form data.
➔There are various process of sending the data to the resource class.
➔The client has various ways to send the data either in the URI, Cookies or header or body.
➔Generally in JAX-RS injection we don't write the logic for pulling the data from request, generally we instruct the JAX-RS to inject the data and which annotation it use are managed by JAX-RS.
➔Some set of annotation are provided by JAX-Rs people for indicating the run time that which data we want to send using which request format.
➔Using this annotation we know how to read the data from the request is called as JAX-RS injection.




## Q. How many ways you can receive the data as part of request?

➔**As part of URI, as part of cookies, as part of header and as part of body.**

## Q. How do we send the data as part of the URI of the http request?

➔There are two ways we can receive the data as part the input in the URI

1. QueryParam
2.PathParam ((Templet parameter))

## Q.What is Difference Between QueryParam and PathParam:

| @QueryParam: | @PathParam: |
|---|---|
| ➔It is very difficult to remember the parameter name to constructing the URI and pass as an input in case of QueryParam.<br>Ex:<br>http://lh:8080/productWeb/productName=tv&manufacture=LG<br>➔In this case the end-user have to remember the parameter name and use it in proper place. | ➔It is very easy to remember the parameter name to communicating the URI and pass as an input in the case of PathParam<br>Ex:<br>http://lh:8080/productWeb/**product/tv/LG** |
| ➔The end-user have to remember the syntax to write the query as part of the query string such as where to use "?" and "productName=product value" and "&".<br>➔Which is difficult to understand as per a non technical end user. | ➔But in PathParam it is easy to write the URI for an end user.<br>➔There is no such grammar present for writing the URI.<br>➔We have to only send the parameter name and Parameter value separated with a "/'. |
| ➔You must should be write the query Parameter at the last of the URI.<br>➔The scope of the QueryParam is whole request. | ➔Incase of PathParam it is easy to write the parameter. |
| ➔QueryParam are optional.<br>➔In the absence of QueryParam also my request also matching the resource. | ➔PathParam is mandatory.<br>➔The general URI partten is http//lh:8080/*/*/product<br>➔So for accessing the resourse we have to mandatory match the URI. |
| ➔QueryParam are **multi value** it can have any parameter can pass as input with the same product name and same manuf name. | ➔PathParam is **single valued** we can send only one request with the same param name and param value as part of the URI. |

## Simplify the URL

```
@Path("/doctor/{dName}")
public class DoctorResource {
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  @Path("/contact")
  public String getContactNo(@PathParam("dName")String doctorName){
     return "9853001547";
  }

  @GET
  @Produces(MediaType.TEXT_PLAIN)
  @Path("/specialization")
  public String getSpecializarion(@PathParam("dName")String doctroName){
       return "Cardiology";
  }
}
```

**http://localhost:8085/TemplateParameters/api/doctor/liza/specialization**

**http://localhost:8085/TemplateParameters/api/doctor/liza/contact**

## Regular Expression Injection

➔As part of the PathParameter we can define regular expression.
➔Suppose we want only the integer type and limited number of digit then we will go for Regular Expression.
➔The **default regular expression is (.+)**
**Ex:**@Path("/specialization/{dId:\\d{3}}")

## Using Regular Expression

```
@Path("/doctor")
public class DoctorResource {
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  @Path("/specialization/{dId:\\d{3}}")
  public String getSpecializarion(@PathParam("dId")String doctroId){
       return "Doctro ID:"+doctroId+" Specialization Cardiology ";
  }
}
```

```
http://localhost:8085/TemplateParameters/api/doctor/specialization/111
```

## Scope of Path Parameter

➔We can write the @PathParam in two different place that means the scope of that PathParam is class level and method level.
➔The class level declaration is also called as Global declaration and the method level declaration is also called as local declaration.
➔If we define the parameter at class then it is called as Global declaration.
➔If we define the parameter at method level then it is called as local declaration.

```
@Path("/doctor/{dName}")
public class DoctorResource {
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  @Path("/contact/{dName}")
  public String getContactNo(@PathParam("dName")String doctorName){
     return "Doctor Name:"+doctorName+" Contact No:9853001547";
  }

  @GET
  @Produces(MediaType.TEXT_PLAIN)
  @Path("/specialization")
  public String getSpecializarion(@PathParam("dName")String doctroName){
       return "Cardiology";
  }
}
```

```
http://localhost:8085/TemplateParameters/api/doctor/dhananjaya/contact/liza

Output
-------
Doctor Name:liza Contact No:9853001547
```

## Q. When we write the @PathParam in class level?

➔When the particular input is required in both the method then it's better to declare the @PathParam in the global level.

Ex:Path ("/icici/{accountNo}")

## JAXRS Injection Examples

### Example#1

```
@Path("/drCar")
public class DrCar {
     @GET
     @Produces(MediaType.TEXT_PLAIN)
     @Path("/search/{type}/{manufacturer}/{year}")
     public String search(@PathParam("type")String type,
               @PathParam("manufacturer")String manufacturer,@PathParam("year")int year){
          return "Type :"+type+" Manufacturer :"+manufacturer+" Year :"+year;
     }
}
```

**http://localhost:7777/JAXRSInjection/api/drCar/search/suden/maruti/2015**

**Type :suden Manufacturer :maruti Year :2015**

### Example#2 (Problem without @DefaultValue)

```
@Path("/drCar")
public class DrCar {
     @GET
     @Produces(MediaType.TEXT_PLAIN)
     @Path("/search/{type}/{manufacturer}")
     public String search(@PathParam("type")String type,
          @PathParam("manufacturer")String manufacturer,
          @DefaultValue("2015")@QueryParam("year")int year){
          return "Type :"+type+" Manufacturer :"+manufacturer+" Year :"+year;
     }
}
```

```
http://localhost:7777/JAXRSInjection/api/drCar/search/suden/maruti

Type :suden Manufacturer :maruti Year :2015
```

## Example#3
## @DefaultValue

```
@Path("/drCar")
public class DrCar {
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	@Path("/search/{type}/{manufacturer}")
	public String search(@PathParam("type")String type,
		@PathParam("manufacturer")String manufacturer,
		@DefaultValue("2015")@QueryParam("year")int year){
		return "Type :"+type+" Manufacturer :"+manufacturer+" Year :"+year;
	}
}
```

```
http://localhost:7777/JAXRSInjection/api/drCar/search/suden/maruti

Type :suden Manufacturer :maruti Year :2015
```

## Matrix Parameter

➔It is the new addition of the http1.1 version.
➔**It is one of the difference between http 1.0 and 1.1 version.**
➔Matrix parameter is something related to JAX-RS .
➔A specification path location of the URI where we can add some additional data as part of the segment can be possible by **Matrix parameter.**
➔The scope of the Matrix Parameter not in URL level, the scope is local to the path segment to which we want to pass the value as input to the resource.

## Example#4
## @MatrixParameter

```
@Path("/drCar")
public class DrCar {
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	@Path("/search/{type}/{manufacturer}/{year}")
	public String search(@PathParam("type")String type,
			@PathParam("manufacturer")String manufacturer,
			@PathParam("year")int year,@MatrixParam("color")String color){
		return "Type :"+type+" Manufacturer :"+manufacturer+" Year :"+year+" Color : "+color;
	}
}
```

```
http://localhost:7777/JAXRSInjection/api/drCar/search/suden/maruti;color=red/2015

 Type :suden Manufacturer :maruti Year :2015 Color : red

http://localhost:7777/JAXRSInjection/api/drCar/search/suden;color=black/maruti;color=red/2015

Type :suden Manufacturer :maruti Year :2015 Color : black
```

## PathSegment

➔Annotation are good enough for rapid application development.
➔ But there are some limitations for this JAX-RS provide classes which we need to provided data manually to the resource that is called programmatic approach.
➔JAX-RS provided one class called **Path Segment**.
➔A specific segment of your URI can be represented by one path segment. It extract the positional parameter value and replace it to the PathSegment.
➔If we use Path Segment then the whole information that is attached to the position is extracted.

**Limitation:**

➔Multiple places or different scopes if the path parameters are there we can only can access the last one that is the limitation with @Pathparam
➔By using @PathParam we can't access the matrix parameters located at the specific position
➔Alternate is programmatic API(PathSegment) or form URIInfo we can get the pathsegments

## Example#5

```
@Path("/drCar")
public class DrCar {
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    @Path("/search/{type}/{manufacturer}/{year}")
    public String search(@PathParam("type")PathSegment pathSegmentType,
                         @PathParam("manufacturer")PathSegment pathSegmentManufacturer,
                         @PathParam("year")int year){

        String typeColor=pathSegmentType.getMatrixParameters().getFirst("color");
        String manufacturerColor=pathSegmentManufacturer.getMatrixParameters().getFirst("color");

        return "Type :"+pathSegmentType+" Manufacturer :"+pathSegmentManufacturer+
                  " Year :"+year+" Type Color : "+typeColor +" Manufacturer Color"+manufacturerColor;
    }
}
```

```
http://localhost:7777/JAXRSInjection/api/drCar/search/suden;color=black/maruti;color=red/2015
```

```
Type :suden;color=black Manufacturer :maruti;color=red Year :2015 Type Color : black Manufacturer Color: red
```

## Q. How to extract the Matrix Parameter at the Run time?

➔To resolve the collision of giving the same segment to the different path Parameter we should go for Path Segment.
➔JAX-RS provide one special Map called **Multi valued Map.**
➔PathSegment used Multivalve Map. Where key is the String and value is Multi value String.
➔RX support multivalue matrix parameter.

## Example#6

## Read Multiple matrix parameter from PathSegment

```
@Path("/drCar")
public class DrCar {
     @GET
     @Produces(MediaType.TEXT_PLAIN)
     @Path("/search/{type}/{manufacturer}/{year}")
     public String search(@PathParam("type")PathSegment pathSegmentType,
          @PathParam("manufacturer")PathSegment pathSegmentManufacturer,
          @PathParam("year")int year){

          String type=pathSegmentType.getPath();
          String manufacturer=pathSegmentManufacturer.getPath();

          String typeAllValues="Type ="+type+" "+getAllParameters(pathSegmentType);
          String manufacturerAllValues="Manufacturer ="+manufacturer+"
                    "+getAllParameters(pathSegmentManufacturer);
          return typeAllValues+" \t"+manufacturerAllValues+" \t Year ="+year;
     }

     public String getAllParameters(PathSegment pathSegment){
          StringBuffer buffer=new StringBuffer();
          MultivaluedMap<String,String> mvp=pathSegment.getMatrixParameters();
          for (String key : mvp.keySet()) {
               buffer.append(key+":");
               List<String> values=mvp.get(key);
               for (String value: values) {
                    buffer.append(value+" ");
               }
               buffer.append(";");
          }
          return buffer.toString();
     }
}
```

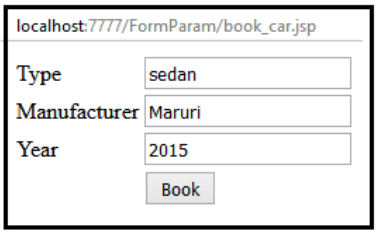
```
http://localhost:7777/JAXRSInjection/api/drCar/search/suden;color=black ;color=blue/maruti;color=red;color=green/2015

Type =suden color:black  blue ;   Manufacturer =maruti color:red green ;      Year =2015
```

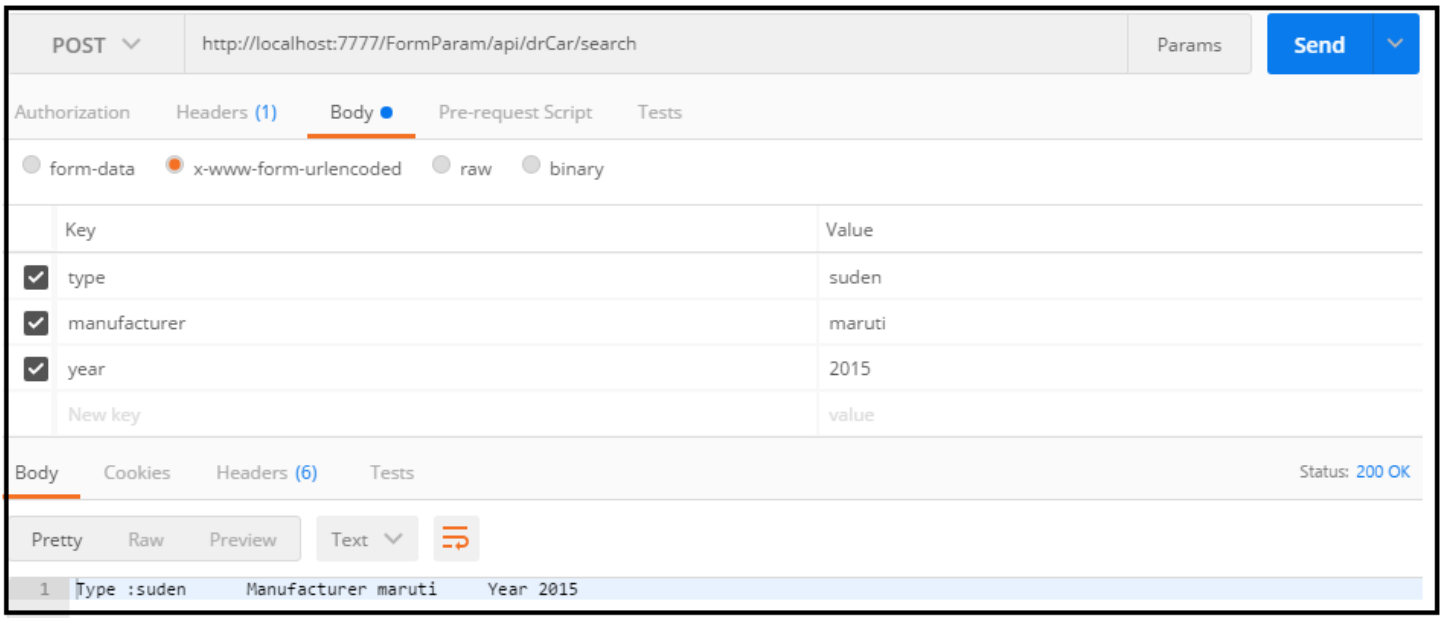
## @FormParam

➔By Using @FormParam annotation our restful our restful web services would accept html form parameters sent by the client in the POST request and bind them to the method variables.
➔Generally we send the formatted data in the body by  which any one can use it properly.
➔The data are present in formatted way that's why we don't required to use String Input in the resource class method.
➔If the data is coming from the request body part in a well formed manner (form) based then we have to use @FormParam.

## Example#7

```
@Path("/drCar")
public class DrCar {
     @POST
     @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
     @Produces(MediaType.TEXT_PLAIN)
     @Path("/search")
     public String search(@FormParam("type")String type,
                          @FormParam("manufacturer")String manufacturer,
                          @FormParam("year")int year){
          return "Type :"+type+" \t Manufacturer "+manufacturer+" \t Year "+year;
     }
}
```

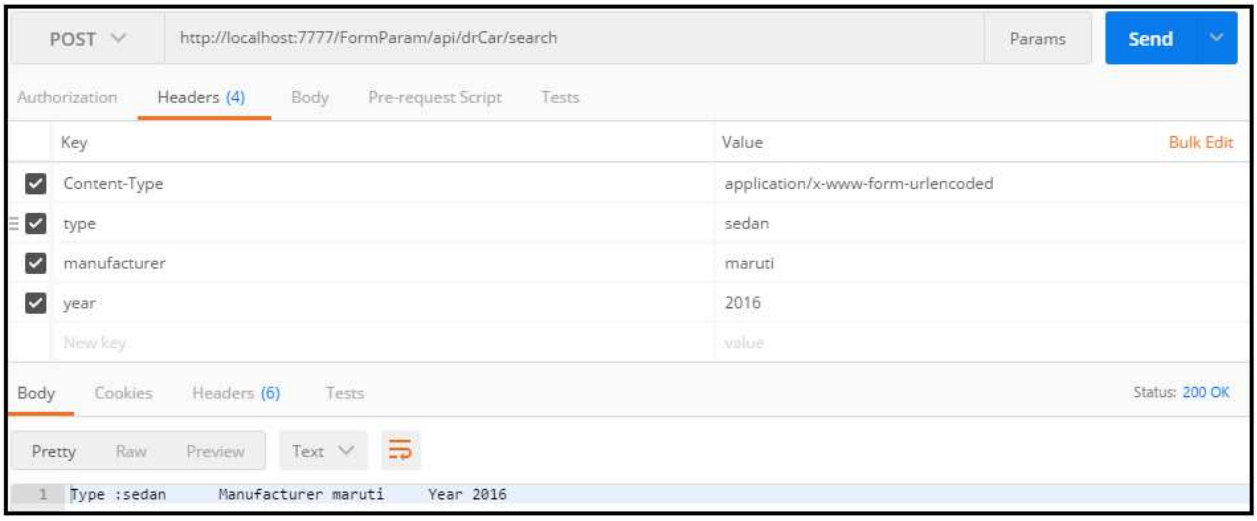



## Example#8

## @HeaderParam

```
@Path("/drCar")
public class DrCar {
	@POST
	@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
	@Produces(MediaType.TEXT_PLAIN)
	@Path("/search")
	public String search(@HeaderParam("type")String type,
			@HeaderParam("manufacturer")String manufacturer,
			@HeaderParam("year")int year){
		return "Type :"+type+" \t Manufacturer "+manufacturer+" \t Year "+year;
	}
}
```



## CookiesParam

➔Servers can store state information in cookies on the client, and can retrieve that information when the client makes its next request.
➔Many web applications use cookies to set up a session between the client and the server.
➔These cookie values are transmitted back and forth between the client and server via cookie headers.
➔When the client try to communicate with the server then in the request part the data comes as a part of cookies.
➔When the path of the cookies match with the path of the domain then automatically the cookies come into the Resourse class.
➔At the resource side we read the cookies by annotating the parameter by using @CookieParam.

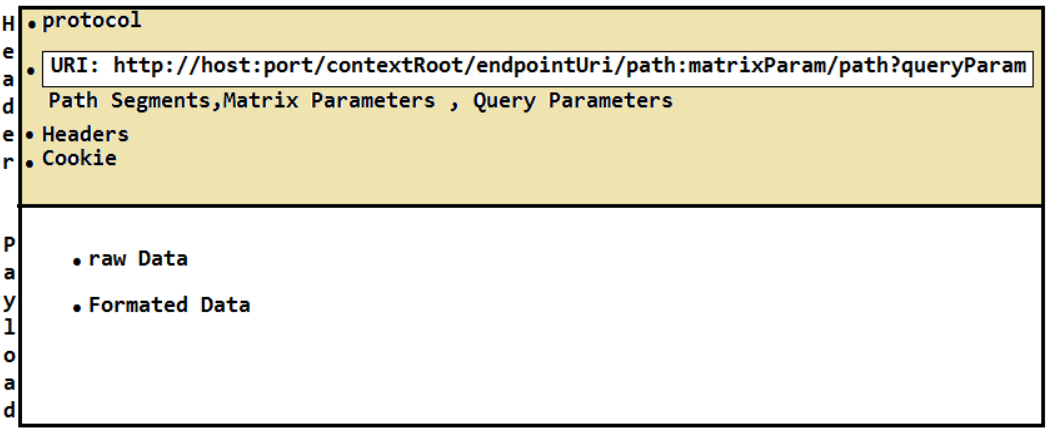
## @Cookie

```
@Path("/drCar")
public class DrCar {
     @GET
     @Produces(MediaType.TEXT_PLAIN)
     @Path("/search/{type}/{manufacturer}/{year}")
     public String search(@PathParam("type")String type,
                          @PathParam("manufacturer")String manufacturer,
                          @PathParam("year")int year,
                          @CookieParam("city") String city){

          return "Type :"+type+" \t Manufacturer "+manufacturer+" \t Year "+year+" Cookie:"+city;
     }

     @GET
     @Produces(MediaType.TEXT_PLAIN)
     @Path("/setCookie")
     public Response getAllEployees(){
       NewCookie cookie= new NewCookie("city", "Hyderabad");
       return Response.ok().cookie(cookie).build();
     }
}
```

http://localhost:7777/FormParam/api/drCar/setCookie

http://localhost:7777/FormParam/api/drCar/search/sedan/maruti/2015

Type :sedan  Manufacturer: maruti  Year: 2015  Cookie: Hyderabad



**Q. What all are annotation are you write in JAX-RS recourses class?**

| | |
|---|---|
| **Path Segments** | 1. @PathParam<br>2. PathSegments(Programmatic)<br>3. URIInfo(C)<br><br>**Limitation:**<br>➔Multiple places or different scopes if the path parameters are there we can only can access the last one that is the limitation with @Pathparam<br>➔By using @PathParam we can't access the matrix parameters located at the specific position<br>➔Alternate is programmatic API(PathSegment) or form URIInfo we can get the pathsegments |
| **Matrix Parameters** | 1. @MatrixParam<br>2. URIInfo(C) |
| **QueryParameters** | 1. @QueryParam<br>2. URIInfo(C) |
| **HttpHeaders** | 1. @HeaderParam<br>2. HttpHeader |
| **Cookies** | 1. @CookieParam<br>2. HttpHeader |
| **Form Request Body** | 1. @FormParam<br>**Programmatically**<br>2. InputStream<br>3. Reader<br>4. byte[]<br>5. String<br>6. File<br>7. MultiValueMap |

**URIInfo and HttpHeaders are Predefined Content Classes in JAX-RS**

## Receive Multiple value by @QueryParam

```
@Path("/drCar")
public class DrCar {
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	@Path("/search/{type}/{manufacturer}")
	public String search(@PathParam("type")String type,
			@PathParam("manufacturer")String manufacturer,
			@QueryParam("year")List<Integer> year){

		return "Type :"+type+"\t Manufacturer :"+manufacturer+"\t Year :"+year;
	}
```

```
http://localhost:7777/JAXRSInjection/api/drCar/search/suden/maruti?year=2015&year=2016&year=2017

Type :suden	Manufacturer :maruti	Year :[2015, 2016, 2017]
```

## @BeanParam

```
public class SearchBean {
	@PathParam("type")
	protected String type;
	@PathParam("manufacturer")
	protected String manufacturer;
	@PathParam("year")
	protected int year;
	@CookieParam("city")
	//Setters & Getters
}
```

```
@Path("/drCar")
	public class DrCar {
		@GET
		@Produces(MediaType.TEXT_PLAIN)
		@Path("/search/{type}/{manufacturer}/{year}")
		public String search(@BeanParam SearchBean searchBean){

			return "Type :"+searchBean.getType()+"
				\t Manufacturer "+searchBean.getManufacturer()+"
				\t Year "+searchBean.getYear()+" Cookie:"+searchBean.getCity();
		}

		@GET
		@Produces(MediaType.TEXT_PLAIN)
		@Path("/setCookie")
		public Response getAllEployees(){
		  NewCookie cookie= new NewCookie("city", "Bhubaneswar");
		  return Response.ok().cookie(cookie).build();
		}
	}
```

```
http://localhost:7777/FormParam/api/drCar/setCookie

http://localhost:7777/FormParam/api/drCar/search/sedan/maruti/2015

Type :sedan	Manufacture: maruti	Year:2015 Cookie:Bhubaneswar
```

## @Context

### Get QueryParameters,PathParameters,MatrixParameters from URIInfo using @ContextParam

```
@Path("/drCar")
public class DrCar {
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	@Path("/search/{type}/{manufacturer}")
	public String search(@Context UriInfo info){

		//getting QueryParameters
		MultivaluedMap<String,String>queryParameters=info.getQueryParameters();
		String qParam=getAllParameters(queryParameters);

		//getting PathParameters
		MultivaluedMap<String,String>pathParameters=info.getPathParameters();
		String pParam=getAllParameters(pathParameters);

		//getting MatrixParameters
		MultivaluedMap<String, String> matrixParameters=null;
		StringBuffer matrixs=new StringBuffer();
		List<PathSegment>pathSegments=info.getPathSegments();


		for (PathSegment pathSegment : pathSegments) {
			matrixParameters=pathSegment.getMatrixParameters();
			matrixs.append(getAllParameters(matrixParameters));
		}

		return qParam+"\n"+pParam+"\n"+matrixs.toString();
	}
	public String getAllParameters(MultivaluedMap<String,String> mvp){
		StringBuffer buffer=new StringBuffer();
		for (String key : mvp.keySet()) {
			buffer.append(key+":");
			List<String> values=mvp.get(key);
			for (String value: values) {
				buffer.append(value).append(" ");
			}
			buffer.append("\n");
		}
		return buffer.toString();
	}
}
```

http://localhost:7777/JAXRSInjection/api/drCar/search/suden;color=black;color=red/maruti?year=2015&year=2016&year=2017

**year:2015 2016 2017**
**type:suden**
**manufacturer:maruti**
**color:black red**

## Get Headers and Cookies from HttpHeaders @Context

```
@Path("/drCar")
public class DrCar {
     @GET
     @Produces(MediaType.TEXT_PLAIN)
     @Path("/search/{type}/{manufacturer}")
     public String search(@Context HttpHeaders headers){
          MultivaluedMap<String, String>requestHeader=headers.getRequestHeaders();
          String rHeaders=getAllParameters(requestHeader);

          StringBuffer buffer=new StringBuffer();
          Map<String, Cookie>cookie=headers.getCookies();
          for (String key : cookie.keySet()) {
               buffer.append("Key : "+key).append("\n");
               buffer.append(cookie.get(key));
          }
          return "Headers: \n "+rHeaders+"\n Cookies :\n"+buffer.toString();
     }
     public String getAllParameters(MultivaluedMap<String,String> mvp){
          StringBuffer buffer=new StringBuffer();
          for (String key : mvp.keySet()) {

               List<String> values=mvp.get(key);
               for (String value: values) {
                    buffer.append(key+":");
                    buffer.append(value).append(" ");
               }
               buffer.append("\n");
          }
          return buffer.toString();
     }

     @GET
     @Produces(MediaType.TEXT_PLAIN)
     @Path("/setCookie")
     public Response getAllEployees(){
       NewCookie cookie= new NewCookie("user", "Dhananjaya");
       NewCookie cookie1= new NewCookie("password", "realspeed");
       return Response.ok().cookie(cookie).cookie(cookie1).build();
     }
}
```

**http://localhost:7777/JAXRSInjection/api/drCar/setCookie**

**http://localhost:7777/JAXRSInjection/api/drCar/search/suden/maruti**

```
Headers:
 Accept:*/*
Accept-Encoding:gzip, deflate, sdch, br
Accept-Language:en-US,en;q=0.8,or;q=0.6,hi;q=0.4
Cache-Control:no-cache
Connection:keep-alive
Content-Type:application/x-www-form-urlencoded
Cookie:city=Bhubaneswar; user=Dhananjaya; password=realspeed
Host:localhost:7777
manufacturer:maruti
Postman-Token:0cef602e-598c-b970-c363-7f5233019bc5
type:sedan
User-Agent:Mozilla/5.0 (Windows NT 6.2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/57.0.2987.133
Safari/537.36
year:2016

 Cookies :
Key : password=realspeedKey
    : user=Dhananjaya
```

## Automatic Type Conversion

### What is Automatic parameter conversion?

➔In most of the cases when we inject value in to the attributes of the parameter those need not to be String .
➔You can take any java primitive types as parameter or attribute into which JAX-RS runtime can perform the injection.
➔There are some predefined supported types into which JAX-RS can perform injection.
➔Any java primitive types can be taken as candidate for resource injection .
➔It can be any class type as well , but there are certain rule that class has to follow.
➔Class should have a single argument constructor
Or
➔class should have a valueOf(String) method which is takes string and convert it to object.

### When we should go for automatic type convertor?.

➔In some cases we don't have the source code of the classes like API provided classes these classes are not following under the automatic type conversion rule then JAX-RS runtime will not able to inject parameterizable arguments then we need to write our own custom parameter convertor and we need pass it to JAX-RS runtime.

### Accessing Multiple Values of PathParameters

```
@Path("/drCar/{year}")
public class DrCar {
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	@Path("/search/{type}/{manufacturer}/{year}")
	public String search(@PathParam("type")String type,
			@PathParam("manufacturer")String manufacturer,
			@PathParam("year")List<Integer> year){
		return "Type :"+type+" \t Manufacturer "+manufacturer+" \t Year "+year;
	}
}
```

```
http://localhost:7777/FormParam/api/drCar/2015/search/sedan/maruti/2017

Type :sedan     Manufacturer maruti     Year [2015, 2017]
```

## Get the Object of data with Constrictor with one String argument

```
@Path("/drCar")
public class DrCar {
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	@Path("/search")
	public String search(@QueryParam("keyword")Criteria criteria) {
		return criteria.toString();
	}
}
```

```
public class Criteria {
	protected String type;
	protected String manufacturer;
	protected int year;
	public Criteria(String criteria){
		String[] splits=criteria.split(",");
		if (splits!=null && splits.length==3) {
			type=splits[0];
			manufacturer=splits[1];
			year=Integer.parseInt(splits[2]);
		}
	}
	@Override
	public String toString() {
		return "Criteria [type=" + type + ",
		manufacturer=" + manufacturer + ", year=" + year + "]";
	}
}
```

```
http://localhost:7777/FormParam/api/drCar/search?keyword=sedan,maruti,2017

Criteria [type=sedan, manufacturer=maruti, year=2017]
```

## Get the Object of data with static valueOf method

```
public class Criteria {
     protected String type;
     protected String manufacturer;
     protected int year;
               public static Criteria valueOf(String keyword){
               String type=null;
               String manufacturer=null;
               int year=0;
               String[] splits=keyword.split(",");
               if (splits!=null && splits.length==3) {
                    type=splits[0];
                    manufacturer=splits[1];
                    year=Integer.parseInt(splits[2]);
               }
               Criteria criteria=new Criteria();
               criteria.type=type;
               criteria.manufacturer=manufacturer;
               criteria.year=year;
               return criteria;
          }

          @Override
          public String toString() {
               return "Criteria [type=" + type + ",
               manufacturer=" + manufacturer + ", year=" + year + "]";
          }
     }
```

```
@Path("/drCar")
public class DrCar {
     @GET
     @Produces(MediaType.TEXT_PLAIN)
     @Path("/search")
     public String search(@QueryParam("keyword")Criteria criteria) {
          return criteria.toString();
     }
}
```

http://localhost:7777/FormParam/api/drCar/search?keyword=sedan,maruti,2014

Criteria [type=sedan, manufacturer=maruti, year=2014]

## CustomParameterTypeConvertor

### How Custom Parameterizable Convertor works?

➔We can't write any class and we can't call it as a parameterizable convertor it must ensure should be implements from ParamConvertor this is the standard interface provided by JAX-RS API.
➔And we need to override 2 method public <Object> fromString(String param) and public String toString().
➔Then we need to write one provider class that class should annotate with @Provider and that provide should register with JAX-RS runtime.
➔Then JAX-RS runtime will talks to the provider by passing a class type as input and ask for which request which custom parameterizable convertor class is to be use.
➔
➔Then JAX-RS runtime will takes care of call these two methods when you send the string value but your method is expecting Object as a input as it can't convert ,then it is going to pass that String parameter as a input to your convertor class and ask the convertor to convert String into Object and return .
➔Then our convertor will converts and returns the object to JAX-RS Runtime and JAX-RS runtime will pass it as an argument to the method of our resource class.

```
@Path("/agency")
public class TravelAgency {
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	@Path("/getPlace/{address}")
	public String getPlaces(@PathParam("address")Address address){
		return address.toString();
	}
}
```

```
public class Address {
	private String street;
	private String city;
	private int pin;
	//Getters & Setters
	//toString()
}
```

```
@Provider
public class CustomParameterConvertorProvider implements ParamConverterProvider {
	@Override
	public ParamConverter getConverter(Class classType, Type rawType, Annotation[] annotations) {
		ParamConverter obj=null;
		if(classType.isAssignableFrom(Address.class)){
		 obj=new CustomParameterConvertor();
		}
		return obj;
	}
}
```

```
public class CustomParameterConvertor implements ParamConverter<Address>{
     Address address=null;
     @Override
     public Address fromString(String in) {

          String [] splits=in.split(",");
          address=new Address();
          if (splits!=null && splits.length==3) {
               address.setStreet(splits[0]);
               address.setCity(splits[1]);
               address.setPin(Integer.parseInt(splits[2]));
          }
          if (splits!=null && splits.length==2) {
               address.setCity(splits[1]);
               address.setPin(Integer.parseInt(splits[2]));
          }
          if (splits!=null && splits.length==1) {
               address.setPin(Integer.parseInt(splits[2]));
          }
          return address;
     }
     @Override
     public String toString(Address address) {
          return address.toString();
     }

}
```

```
@ApplicationPath("/api")
public class CPCApplication extends Application{
     private Set<Object> singletons;
     private Set<Class<?>> classes;

     public CPCApplication() {
          singletons=new HashSet<Object>();
          classes=new HashSet<Class<?>>();
          singletons.add(new TravelAgency());
          singletons.add(new CustomParameterConvertorProvider());
          classes.add(CustomParameterConvertor.class);
     }

     @Override
     public Set<Class<?>> getClasses() {
          return classes;
     }

     @Override
     public Set<Object> getSingletons() {
          return singletons;
     }
```

```
http://localhost:7777/CustomParamConvertor/api/agency/getPlace/ameerpet,hyderabad,50038

Address [street=ameerpet, city=hyderabad, pin=50038]
```

**Request Using InputStream**

```
@Path("aadhar")
public class AadharRegistration {
	@POST
	@Produces(MediaType.TEXT_PLAIN)
	@Consumes(MediaType.TEXT_PLAIN)
	@Path("/is")
	public String register(InputStream is) throws IOException{
		StringBuffer buffer=new StringBuffer();
		BufferedInputStream bis=new BufferedInputStream(is);
		int ch=0;
			while ((ch=bis.read())!=-1) {
				buffer.append((char)ch);
			}
			bis.close();
			is.close();
		return buffer.toString();
  }
}
```

```
@ApplicationPath("/")
public class CHApplication extends Application{
}
```

```
http://localhost:7777/ContentHandler/aadhar/is

name=Dhananjaya,Age=27,City=Bhubaneswar
```

**Send data as part of body**

**With content Type as text/plain**

**Request Using Reader**

```
@POST
@Produces(MediaType.TEXT_PLAIN)
@Consumes(MediaType.TEXT_PLAIN)
@Path("/reader")
public String register(Reader reader) throws IOException{
     String line=null;
     BufferedReader bReader=null;
     StringBuffer buffer=null;

     bReader=new BufferedReader(reader);
     buffer=new StringBuffer();
          while ((line=bReader.readLine())!=null) {
               buffer.append(line);
          }
          reader.close();
          bReader.close();
     return buffer.toString();
}
```

http://localhost:7777/ContentHandler/aadhar/reader

Send in Body
name=Dhananjaya,Age=27,City=Bhubaneswar

**Request Using byte[]**

```
@POST
@Produces(MediaType.TEXT_PLAIN)
@Consumes(MediaType.TEXT_PLAIN)
@Path("/byte")
public String register(byte[] data) throws IOException{
     StringBuffer buffer=null;
     buffer=new StringBuffer();
          if (data!=null && data.length>0){
               for (byte b : data) {
                    buffer.append((char)b);
               }
          }
     return buffer.toString();
}
```

http://localhost:7777/ContentHandler/aadhar/byte

name=Dhananjaya,Age=27,City=Bhubaneswar

**Request Using String**

```
@POST
@Produces(MediaType.TEXT_PLAIN)
@Consumes(MediaType.TEXT_PLAIN)
@Path("/string")
public String register(String data) throws IOException{
     return data;
}
```

http://localhost:7777/ContentHandler/aadhar/string

name=Dhananjaya,Age=27,City=Bhubaneswar

**Request Using File**

```
@POST
     @Produces(MediaType.TEXT_PLAIN)
     @Consumes(MediaType.MULTIPART_FORM_DATA)
     @Path("/upload")
     public String register(File file) throws IOException{
          StringBuffer buffer=null;
          FileInputStream fis=null;
          int c=0;
          buffer=new StringBuffer();
               if (file!=null){
                    fis=new FileInputStream(file);
                    while((c=fis.read())!=-1) {
                         buffer.append((char)c);
                    }
               }
          return buffer.toString();
  }
```

http://localhost:7777/ContentHandler/aadhar/upload

save in a txt file and send binary data from postman
name=Dhananjaya,Age=27,City=Hyderabad

## Request Using MultiValueMap

```
@POST
@Produces(MediaType.TEXT_PLAIN)
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
@Path("/form")
public String register(MultivaluedMap<String, String> formData) throws IOException {
     StringBuffer buffer = null;
     buffer = new StringBuffer();
     for (String key : formData.keySet()) {
          buffer.append(" Key :").append(key)
          .append("  value :").append(formData.getFirst(key))
          .append("\n");
     }
     return buffer.toString();
}
```

http://localhost:7777/ContentHandler/aadhar/form

send form data through x-www-form-urlencoded in postman

Key :city  value :odisha
Key :name  value :Dhananjaya
Key :age  value :27

## Cusom content Handler

➔For returning the returning value from the resource the possible return type is "streamingOutput, byte[], file or String"
➔For giving input to the resource class the possible return type is "byte[], string, file, inputStream and MultiValueMap".
➔If the data is send as part of the request body then we receive the data as any of the one data type that support by the JAX-RS API.
➔My resource class contains complex business logic and for performing these business logic we need data .and the data are coming from the request body.
➔The client may send in xml format. And the resource class understand the xml data but when the client change the format xml to json then we have to modify the resource class.
➔For one one request format we have to write one one resource method.
➔It is not easy to support multiple content type because the business logic is expose to the format of the data.
➔To overcome this problem we are not going to read the data in raw format we are read the data from the resource class as **object of data.**
➔Then if we change the format to xml to json then we don't need to change the required type in the resource class.
➔So the return type we take as object of data.
➔When the client send the request to the resource class then it comes to the JAX-RS runtime .And JAX-RS runtime identify the resource class method by the help of root resource URI and sub resource URI and the content type.
➔If the content type is application/xml then it goes to the particular method.
➔The JAX-RS runtime does not know how to convert the xml data to object type data.
➔The conversion of xml representation data to object type is known by programmer.
➔The clients request is coming to the JAX-RS runtime and it does not know how to convert the data to the object format so it takes the help of **MessageBodyReader** .
➔Message body reader helps the data to convert into object it act as a converter.
➔To converting the input data to the object type the Message body reader wants several input these are.

1. MediaType
2. InputStream
3.RawType
4. ClassType
5.Annotation[]
6. http header request (for additional information related to the data)

➔Message body reader read the data from the request body and convert it to the object type and send it to the resource class.

## Custom Content Handler (Content Type=Application/XML)

```
@Path("/amazon")
public class Amazon {
	@POST
	@Consumes(MediaType.APPLICATION_XML)
	@Produces(MediaType.APPLICATION_XML)
	@Path("/add")
	public Acknowledge addProduct(Product product){
		Acknowledge acknowledge=new Acknowledge();
		acknowledge.setAckNo(product.getProductName()+""+product.getProductCode());
		acknowledge.setMessage("Product has been added");
		return acknowledge;
	}
}
```

```
@XmlRootElement(name="product")
@XmlType(name="product")
@XmlAccessorType(XmlAccessType.FIELD)
public class Product {
	@XmlElement(name="product-code")
	private int productCode;
	@XmlElement(name="product-name")
	private String productName;
	private String category;
	private float price;
	//getter & setters
}
```

```
@XmlRootElement(name="acknowledge")
@XmlType(name="acknowledge")
@XmlAccessorType(XmlAccessType.FIELD)
public class Acknowledge {
	@XmlElement(name="ack-no")
	private String ackNo;
	private String message;
	//setters and getters
}
```

```
@ApplicationPath("/api")
public class AmazonApplication extends
Application{
}
```

## MessageBodyReader

```
@Provider
@Consumes(MediaType.APPLICATION_XML)
public class AmazonJAXBMessageReader implements MessageBodyReader<Object>{
     @Override
     public boolean isReadable(Class<?> classType, Type rawType, Annotation[] annotations, MediaType mediaType)
{
          if (classType.isAnnotationPresent(XmlRootElement.class)) {
               return true;
          }
          return false;
     }
     @Override
     public Object readFrom(Class<Object> classType, Type rawType, Annotation[]   annotations, MediaType
mediaType,MultivaluedMap<String, String> multiValues,     InputStream is) throws IOException,
WebApplicationException {
          Object object=null;
          try {
               System.out.println("****JAX-B readFrom()****");
               JAXBContext context=JAXBContext.newInstance(classType);
               Unmarshaller unmarshaller=context.createUnmarshaller();
               object=unmarshaller.unmarshal(is);
          } catch (JAXBException e) {
               throw new WebApplicationException(e);
          }
          return object;
     }
}
```

## MessageBodyWriter

```
@Provider
@Produces(MediaType.APPLICATION_XML)
public class AmazonJAXBMessageWriter implements MessageBodyWriter<Object> {

     @Override
     public long getSize(Object obj, Class<?> classType, Type rawType, Annotation[] annotations, MediaType mediaType) {
          return 0;
     }

     @Override
     public boolean isWriteable(Class<?> classType, Type rawType, Annotation[] annotations, MediaType mediaType) {

          if (classType.isAnnotationPresent(XmlRootElement.class)) {
               return true;
          }
          return false;
     }

     @Override
     public void writeTo(Object obj, Class<?> classType, Type rawType, Annotation[] annotations, MediaType mediaType,
               MultivaluedMap<String, Object> multiMap, OutputStream os) throws IOException, WebApplicationException {
          try {
               System.out.println("****JAX-B writeTo()****");
               JAXBContext context = JAXBContext.newInstance(classType);
               Marshaller marshaller = context.createMarshaller();
               marshaller.marshal(obj, os);
          } catch (JAXBException e) {
               throw new WebApplicationException(e);

          }

     }
}
```

**Input and Output**

```
http://localhost:7777/XMLRequest/api/amazon/add
```

Input

```
<?xml version="1.0" encoding="utf-8"?>
<product>
     <product-code>j7</product-code>
     <product-name>Samsung Galaxy</product-name>
     <category>mobile</category>
     <price>8990</price>
</product>
```

Output

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<acknowledge>
    <ack-no>Samsung Galaxy0</ack-no>
    <message>Product has been added</message>
</acknowledge>
```

## Control Flow:

➔First it will check what is the resource class? and what is the resource method?

➔What does the class excepting(object or what)?

➔JAX-RS runtime goes to the Message body reader and give the xml as input and ask Message body reader to convert it to object type.

➔There are multiple Message Body Reader for converting various different format to object. (xml or Json…)

➔JAX-RS runtime does not know which MessageBodyReader is converting which message body format.

➔So it takes the help of provider to identify the particular MessageBodyReader which support incoming request media Type.

## Message body Writer

➔**S**imilarly from converting the object type to xml message format then our JAX-RS runtime never no to how to convert the object to xml so it takes help of Message Body Writer class.

➔MessageBodyWrite is a class which knows how to convert object to any form of data.
Because our resource class send object as output and it comes to the JAX-RS run time and it does not know how to convert then it takes help of MessageBodyWriter. For converting Message body writer need some input.

## Q.How can we convert a xml into an object?

➔For converting the xml to object we take the help of JAX-B .
➔So take the MessageBodyReader as JAX-B message Body reader as class name.
➔And my class must and should be implement from a standard interface through which our JAX-Rs runtime can easily understand our class.
➔So my class should be implements from MessageBodyReader.

In this class we have to write two methods

1. for verifying (**public boolean isReadable()**)
2. for converting **public object readFrom()**

similarly my message Body Writer class also impliments MessageBodyWriter and override methods.

1. for verifying (**public boolean isWritable()**)
2. for converting **public object writeTo()**

## Custom Content Handler (our own format)

Like(name=dhananjaya;address=Bhubaneswar)

```
@Path("/amazon")
public class Amazon {
	@POST
	@Consumes(MediaType.TEXT_PLAIN)
	@Produces(MediaType.TEXT_PLAIN)
	@Path("/register")
	public AccountInfo register(Membership membership){
		AccountInfo acc=new AccountInfo();
		acc.setMemberId(String.valueOf(new Random().nextInt(1000000)));
		acc.setMobile(membership.getMobile());
		return acc;
	}
}
```

```
@KeyValueType
public class AccountInfo {
	private String memberId;
	private String mobile;
	//Setters and Getters
}
```

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface KeyValueType {

}
```

```
@KeyValueType
public class Membership {
	private String memberName;
	private String password;
	private String gender;
	private String mobile;
	//setters and getters
}
```

```
http://localhost:7777/CustomContentHandler/api/amazon/register
memberName=Dhananjaya;password=realspeed;gender=male;mobile=9040010697

memberId=376585;mobile=9040010697
```

## MessageBodyReader

```
@Provider
@Consumes(MediaType.TEXT_PLAIN)
public class AmazonKeyValueMessageBodyReader implements MessageBodyReader<Object>{
	@Override
	public boolean isReadable(Class<?> classType, Type rawType, Annotation[] annotations, MediaType mediaTypes) {
		if (classType.isAnnotationPresent(KeyValueType.class)) {
			return true;
		}
		return false;
	}
	@Override
	public Object readFrom(Class<Object> classType, Type rawType, Annotation[] annotations, MediaType mediaTypes,
		MultivaluedMap<String, String> requestHeaders, InputStream is) throws IOException, WebApplicationException {
		Object obj=null;
		System.out.println("*******readFrom(method Called)*******");
		String rawData=getRawData(classType, is);
		Map<String,String> buildDataMap=buildRequestDataMap(rawData);
		obj=getObject(buildDataMap, classType);
		return obj;
	}
	private String getRawData(Class<Object> classType,InputStream is){
		StringBuffer buffer=new StringBuffer();
		int c=0;
		try {
			while ((c=is.read())!=-1) {
				buffer.append((char)c);
			}
		} catch (IOException e) {
			e.printStackTrace();
		}
		return buffer.toString();
	}
	private Map<String,String> buildRequestDataMap(String rawData){
		Map<String,String> requestDataMap=null;
		String[] dataTokens=rawData.split(";");
		requestDataMap=new HashMap<String,String>();
		for (String s : dataTokens) {
			String [] tokens=s.split("=");
			requestDataMap.put(tokens[0],tokens[1]);
		}
		return requestDataMap;
	}
	private Object getObject(Map<String,String> buildDataMap,Class<Object> classType){
		Object obj=null;
		Set<String> attributeNames=buildDataMap.keySet();
		try{
			obj=classType.newInstance();
			for (String attributeName : attributeNames) {
				String value=buildDataMap.get(attributeName);
				Method[] methods=classType.getDeclaredMethods();
				for (Method method : methods) {
					if (method.getName().equalsIgnoreCase("set"+attributeName)) {
						method.invoke(obj,value);
						break;
					}
				}
			}
		}catch(Exception e){
			throw new WebApplicationException(e);
		}
		return obj;
	}
}
```

## MessageBodyWriter

```
@Provider
@Produces(MediaType.TEXT_PLAIN)
public class AmazonKeyValueMessageBodyWriter implements MessageBodyWriter<Object>{

    @Override
    public long getSize(Object obj, Class<?> classType, Type rawType, Annotation[] annotations, MediaType mediaType) {
        return 0;
    }

    @Override
    public boolean isWriteable(Class<?> classType, Type rawType, Annotation[] annotations, MediaType mediaType) {

        if (classType.isAnnotationPresent(KeyValueType.class)) {
            return true;
        }
        return false;
    }

    @Override
    public void writeTo(Object obj, Class<?> classType, Type rawType, Annotation[] annotations, MediaType mediaType,
            MultivaluedMap<String, Object> responseHeaders, OutputStream os) throws IOException, WebApplicationException {
        System.out.println("*******writeTo(method Called)*******");
        writeData(obj,os);
    }
    private void writeData(Object obj,OutputStream os){
        Class<?> classType=obj.getClass();
        Field[] fields=classType.getDeclaredFields();
        StringBuffer buffer=new StringBuffer();
        Method[] methods=classType.getDeclaredMethods();
        boolean isFirst=true;
        try {
            for (Field field : fields){
                for (Method method : methods) {
                    if (method.getName().equalsIgnoreCase("get"+field.getName())) {
                        String value=(String)method.invoke(obj,null);
                        if (isFirst) {
                            buffer.append(field.getName()).append("=").append(value);
                            isFirst=false;
                        }else{
                            buffer.append(";").append(field.getName()).append("=").append(value);
                        }
                    }
                }
            }
            os.write(buffer.toString().getBytes());
            os.close();
        } catch (Exception e) {
            throw new WebApplicationException(e);
        }
    }
}
```

## Content Handling(json)

**Q.If the user is sending the JSON as input how cans we receive the data in object format?**

➔We don't have to write my own the MessageBodyReader and Writer to exchange the data because Jersey and Resteasy are integrated with Jackson.
➔If you write the JAX-B binding class as parameter as input and return type then it is support me to exchanging the data.
➔**Enable Support for Jax-B/JSON we should use in Jersey init param as pojoMappingFeature=true**

```
@XmlRootElement(name="product")
@XmlType(name="product")
@XmlAccessorType(XmlAccessType.FIELD)
public class Product {
     @XmlElement(name="product-code")
     private int productCode;
     @XmlElement(name="product-name")
     private String productName;
     private String category;
     private float price;
     //setters and getters
}
```

```
@POST
     @Consumes({MediaType.APPLICATION_XML,MediaType.APPLICATION_JSON})
     @Produces({MediaType.APPLICATION_XML,MediaType.APPLICATION_JSON})
     @Path("/add")
     public Acknowledge addProduct(Product product){
          Acknowledge acknowledge=new Acknowledge();

     acknowledge.setAckNo(product.getProductName()+""+product.getProductCode());
          acknowledge.setMessage("Product has been added");
          return acknowledge;
     }
```

**Input & output**

```
<product>
     <product-code>j7</product-code>
     <product-name>Samsung Galaxy</product-name>
     <category>mobile</category>
     <price>45567</price>
</product>
```

```
{
  "message": "Product has been added",
  "ack-no": "Samsung Galaxy0"
}
```


**Q.What is representation Oriented?(How many client can interact with resource)**

➔Different different client can interact with the resource. Different request format can be send request to the request (xml or json).
➔Rest support representational state transfer.
➔The client can interact with the resource with any of the format.

**Q.What is Custom content handler and what is the purpose of Custom content handler?**

➔When we want to interact with the recourses by searching the data, the request can read the data from the request body.
➔But when it used to request the data from the request body it has to send the standard data type or content type that are supported by JAX-RS runtime(byte[],MultiValueMap, file, InputStream ,Reader).
➔If we use any of the datatype then the data the resource is going to read is became raw data and the business logic that we written became tightly coupled.
➔To overcome this problem and to make our resource class Representation oriented instead of represent the class as raw data we can represent our resource class which support any kind of input.
➔We have to make our resource class who can read the data from message body by object format.
➔That is called Custom content Handler.Becouse JAX-RS does not know how to exchange object to data and data to object.
➔So we write our own Custom Content Handler.

**Q.How the Message Body Reader registered with JAX-RS runtime?**

➔At the time when we deploy the application the http servlet request dispatcher is created. The servlet dispatcher is going to call init ().
➔init() checks whether the auto scan is enable or not.(resteasy.scan is true or not if we use Application Path then it is always true) .
➔The httpServletDispatcher goes to all the classes present in the class path including the jars looking for the classes which are annotated with @Path and @Provider annotation.
➔It scans the resources and registers the Meta data, identify the provider register the provider into JAX-RS runtime.
➔After it register into the JAX-RS runtime it checks one class after another class and checks which class is allow to handle xml or json among the group of Reader and Writer class by the help of isReadable().

## Context Resolver

➔**Our** JAX-RS runtime contain multiple MessageBodyReader and MessageBodyWritera and all the MessageBodyReader and Writer interface are present in the JVM memory.
➔For all the reader and writer class we have to create the context object(e.g. JAXBContext). Instead of creating object for all the reader and writer class we go for Context Factory. and all the classes are depends upon the factory class for object .As we use one object our JVM memory will going to invest less. For this activity **Context Resolver** comes into picture.

### Q.Why we use Context Resolver and how it works?

➔It helps in optimize the performance of the JVM and it use less JVM memory.
➔It helps in identifying the write context object to use across all the resources.
➔Don't create JAX-B reader or writer class object across the class of the application or don't create in each and every class.
➔Instantiate this objects from context resolver and allow the classes to call the resolver. Because all classes need the same object.
➔If your restful service is integrated with other technology that technology specific object should be distrusted across your application, for this reason we use Context Resolver.

```
@Path("/amazon")
public class Amazon {
     @POST
     @Consumes({MediaType.APPLICATION_XML,MediaType.APPLICATION_JSON})
     @Produces({MediaType.APPLICATION_XML,MediaType.APPLICATION_JSON})
     @Path("/add")
     public Acknowledge addProduct(Product product){
          Acknowledge acknowledge=new Acknowledge();

     acknowledge.setAckNo(product.getProductName()+""+product.getProductCode());
          acknowledge.setMessage("Product has been added");
          return acknowledge;
     }
}
```

```
@XmlRootElement(name="product")
@XmlType(name="product")
@XmlAccessorType(XmlAccessType.FIELD)
public class Product {
     @XmlElement(name="product-code")
     private int productCode;
     @XmlElement(name="product-name")
     private String productName;
     private String category;
     private float price;
     //setters and getters
}
```

```
@XmlRootElement(name="acknowledge")
@XmlType(name="acknowledge")
@XmlAccessorType(XmlAccessType.FIELD)
public class Acknowledge {
     @XmlElement(name="ack-no")
     private String ackNo;
     private String message;
     //setters and getters
}
```

```
@ApplicationPath("/api")
public class AmazonApplication extends Application{
}
```

```java
@Provider
public class JAXBContextResolver implements ContextResolver<JAXBContext>{
     JAXBContext jContext=null;

     public JAXBContextResolver() {
          try {
               this.jContext=
          JAXBContext.newInstance(new Class[]{Product.class,Acknowledge.class});
          } catch (JAXBException e) {
               new WebApplicationException(e);
          }
     }
     @Override
     public JAXBContext getContext(Class<?> classType) {
          if (classType.isAssignableFrom(Product.class)||
classType.isAssignableFrom(Acknowledge.class)) {
               return jContext;
          }
               return null;
     }
}
```

```java
@Provider
@Consumes(MediaType.APPLICATION_XML)
public class AmazonJAXBMessageBodyReader implements MessageBodyReader<Object>{
     @Context
     private Providers providers;
     @Override
     public boolean isReadable(Class<?> classType, Type rawType, Annotation[] annotations, MediaType mediaType) {
          if (classType.isAnnotationPresent(XmlRootElement.class)) {
               return true;
          }
          return false;
     }
     @Override
     public Object readFrom(Class<Object> classType, Type rawType, Annotation[] annotations, MediaType mediaType,
               MultivaluedMap<String, String> requestHeaders, InputStream is) throws IOException, WebApplicationException {
          Object object=null;
          try {
               System.out.println("****JAX-B readFrom()****");
               ContextResolver<JAXBContext> resolver=providers.getContextResolver(JAXBContext.class,MediaType.APPLICATION_XML_TYPE);
               JAXBContext jContext=(JAXBContext)resolver.getContext(classType);
               //JAXBContext jContext=JAXBContext.newInstance(classType);
               Unmarshaller unmarshaller=jContext.createUnmarshaller();
               object=unmarshaller.unmarshal(is);
          } catch (JAXBException e) {
               throw new WebApplicationException(e);
          }
          return object;
     }
}
```

➔For working with context resolver our class must on should **implements** from the **Context Resolver** .And it create JAXB Context object.
➔Context Resolver also register as Provider.
➔For context object we need go for JAX-RS runtime and ask him for Context Resolver because the context resolver knows about the context object our JAX-RS runtime knows about the ContextResolver it does not know anything about the Context object.
➔JAX-RS return the ContextResolver to us for this we have to register the ContextResolver with JAX-Rs runtime.
➔Context Resolver register inside the Provides in the JAX-Rs runtime.

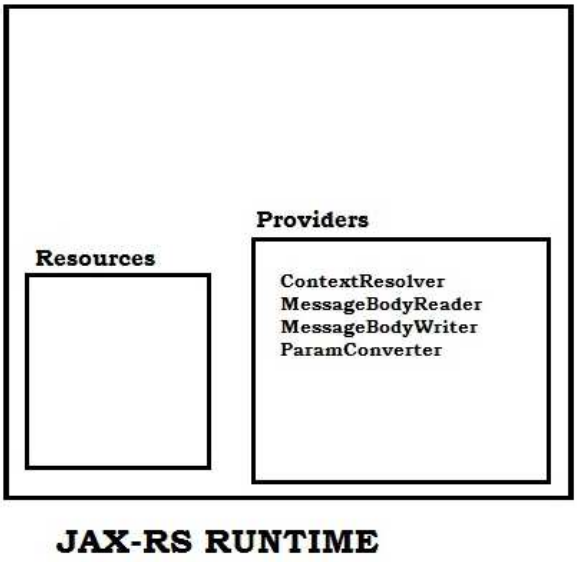
```java
@Provider
@Produces(MediaType.APPLICATION_XML)

public class AmazonJAXBMessageBodyWriter implements MessageBodyWriter<Object> {
     @Context
     private Providers providers;
     @Override
     public long getSize(Object obj, Class<?> classType, Type rawType, Annotation[] annotations, MediaType mediaType) {
          return 0;
     }
     @Override
     public boolean isWriteable(Class<?> classType, Type rawType, Annotation[] annotations, MediaType mediaType) {

          if (classType.isAnnotationPresent(XmlRootElement.class)) {
               return true;
          }
          return false;
     }
     @Override
     public void writeTo(Object obj, Class<?> classType, Type rawType, Annotation[] annotations, MediaType mediaType,
               MultivaluedMap<String, Object> requestHeaders, OutputStream os)
               throws IOException, WebApplicationException {
          try {
               ContextResolver<JAXBContext> resolver = providers.getContextResolver(JAXBContext.class,
                         MediaType.APPLICATION_XML_TYPE);
               // JAXBContext jContext=JAXBContext.newInstance(classType);
               JAXBContext jContext = (JAXBContext) resolver.getContext(classType);
               Marshaller marshaller = jContext.createMarshaller();
               marshaller.marshal(obj, os);
          } catch (JAXBException e) {
               throw new WebApplicationException(e);
          }
     }
}
```

```
http://localhost:7777/CustomContentHandler/api/amazon/add

{"Content-Type":"application/xml","Accept":"application/json"}

<product>
     <product-code>j7</product-code>
     <product-name>Samsung Galaxy</product-name>
     <category>mobile</category>
     <price>45567</price>
</product>


{
  "message": "Product has been added",
  "ack-no": "Samsung Galaxy0"
}
```

## Q.Total how many Provider are there in the JAX-RS Api?

➔There are five Provider are there in JAX-RS rutime.

1. MessageBodyReader
2. MessageBodyWriter
3. ContextResolver
4. ParamConverter Provider.
5. ----------



## Context Responses

➔As part of the dispatching the response back to the client, we may not only send data rather we may need to send response headers or cookies or status code back to the client.
➔For customize the response we need to use JAX-RS API provided to return responses .This is possible for Context Response.

## Responses using javax.ws.rs.core.Response

➔You can build custom responses using the javax.ws.rs.core.Response and ResponseBuilder classes. If you want to do your own streaming, your entity response must be an implementation of javax.ws.rs.core.StreamingOutput. See the java doc for more information.

## Server Response

```
@Path("/uber")
public class Uber {
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	@Path("/fare/{source}/{destination}")
	public Response getBaseFare(@PathParam("source") String source,
				@PathParam("destination") String destination){
		ResponseBuilder builder=Response.status(201);
		Response response=builder.build();
		return response;
	}
}
```

```
@ApplicationPath("/api")
public class SRApplication extends Application {
}
```

http://localhost:7777/ServerResponse/api/uber/fare/hyd/bbsr

## Http Status Codes

- **1×× Informational**
- 100 Continue
- 101 Switching Protocols
- 102 Processing

- **2×× Success**
- 200 OK
- 201 Created
- 202 Accepted
- 203 Non-authoritative Information
- 204 No Content
- 205 Reset Content
- 206 Partial Content
- 207 Multi-Status
- 208 Already Reported
- 226 IM Used

- **3×× Redirection**
- 300 Multiple Choices
- 301 Moved Permanently
- 302 Found
- 303 See Other
- 304 Not Modified
- 305 Use Proxy
- 307 Temporary Redirect
- 308 Permanent Redirect

- **4×× Client Error**
- 400 Bad Request
- 401 Unauthorized
- 402 Payment Required
- 403 Forbidden
- 404 Not Found
- 405 Method Not Allowed
- 406 Not Acceptable
- 407 Proxy Authentication Required
- 408 Request Timeout
- 409 Conflict
- 410 Gone
- 411 Length Required
- 412 Precondition Failed
- 413 Payload Too Large
- 414 Request-URI Too Long
- 415 Unsupported Media Type
- 416 Requested Range Not Satisfiable
- 417 Expectation Failed
- 418 I'm a teapot
- 421 Misdirected Request
- 422 Unprocessable Entity
- 423 Locked
- 424 Failed Dependency
- 426 Upgrade Required
- 428 Precondition Required
- 429 Too Many Requests
- 431 Request Header Fields Too Large
- 444 Connection Closed Without Response
- 451 Unavailable For Legal Reasons
- 499 Client Closed Request

- **5×× Server Error**
- 500 Internal Server Error
- 501 Not Implemented
- 502 Bad Gateway
- 503 Service Unavailable
- 504 Gateway Timeout
- 505 HTTP Version Not Supported
- 506 Variant Also Negotiates
- 507 Insufficient Storage
- 508 Loop Detected
- 510 Not Extended
- 511 Network Authentication Required
- 599 Network Connect Timeout Error

To understand How to work with Response Here I am taking a Bank Example

## Ex-1

### Sending the output through Response Object

```
@Path("/netbanking")
public class NetBanking {
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	@Path("/balance/{accountNo}")
	public Response getBalance(@PathParam("accountNo")String accountNo){
		Response response=null;
		ResponseBuilder builder=null;
		builder=Response.ok("Your Balance is :"+333.00);
		response=builder.build();
		return response;
	}
}
```

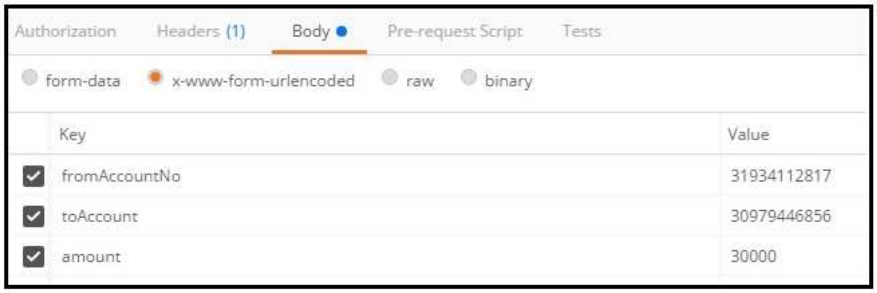
```
http://localhost:7777/ServerResponse/api/netbanking/balance/31934112817

Your Balance is :333.0
```

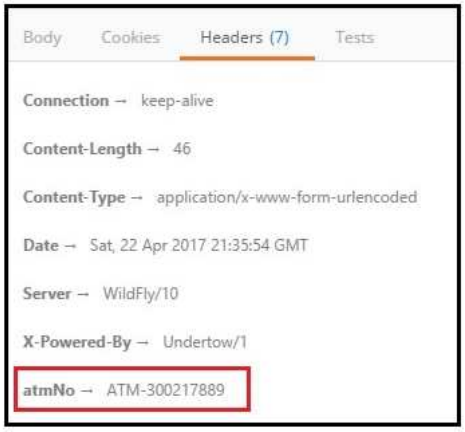
## Ex-2

### Sending the data in HttpHeader using Response

```
@Path("/netbanking")
public class NetBanking {
	@PUT
	@Produces(MediaType.APPLICATION_FORM_URLENCODED)
	@Path("/transfer")
	public Response transfer(@FormParam("fromAccountNo")String fromAccountNo,
				@FormParam("toAccount")String toAccount,
				@FormParam("amount")double amount){
		Response response=null;
		ResponseBuilder builder=null;
		builder=Response.status(201);
		builder.entity(amount+" Successfully transfered to "+toAccount);
		builder.header("atmNo", "ATM-300217889");
		response=builder.build();
		return response;
	}
}
```

http://localhost:7777/ServerResponse/api/netbanking/transfer



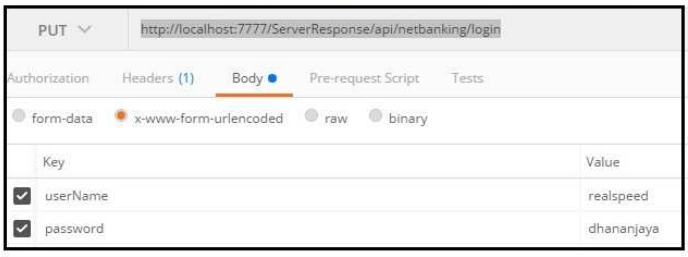

30000.00 Successfully transfered to 30979446856

## Ex-3

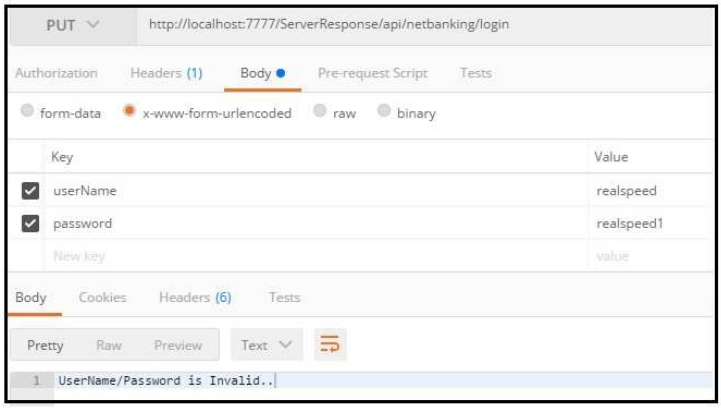
## Sending the various HttpStatus code as per condition

```
@Path("/netbanking")
public class NetBanking {
	@PUT
	@Produces(MediaType.APPLICATION_FORM_URLENCODED)
	@Path("/login")
	public Response transfer(@FormParam("userName")String userName,
				@FormParam("password")String password){
		Response response=null;
		ResponseBuilder builder=null;
		if (userName.equals("realspeed") && password.equals("dhananjaya")) {
			builder=Response.status(200);
			builder.entity("Welcome "+userName);
		}
		else{
			builder=Response.status(401);
			builder.entity("UserName/Password is Invalid..");
		}
		response=builder.build();
		return response;
	}
}
```

http://localhost:7777/ServerResponse/api/netbanking/login




Welcome realspeed



## Ex-4

### Sending the Cookie using Response

```
@Path("/netbanking")
public class NetBanking {
     @PUT
     @Produces(MediaType.APPLICATION_FORM_URLENCODED)
     @Path("/login")
     public Response transfer(@FormParam("userName")String userName,
                              @FormParam("password")String password,
                              @FormParam("remember")boolean remember){
          Response response=null;
          ResponseBuilder builder=null;
          if (userName.equals("realspeed") && password.equals("dhananjaya")) {
               builder=Response.status(200);
               builder.entity("Welcome "+userName);
               if (remember==true) {
               builder.cookie(new NewCookie("userName",userName),
                              new NewCookie("password",password));
               }
          }
          else{
               builder=Response.status(401);
               builder.entity("UserName/Password is Invalid..");
          }
          response=builder.build();
          return response;
     }
}
```

## Ex-5

### How to work with Collection type .

```
@Path("/netbanking")
public class NetBanking {
     @GET
     @Produces({MediaType.APPLICATION_JSON})
     @Path("/services/{accountNo}")
     public Response getServices(@PathParam("accountNo")String accountNo){
          Response response=null;
          ResponseBuilder builder=null;
          List<Service> services= null;
          if (accountNo.equals("A-30979446856")) {
               services=new ArrayList<Service>();
               services.add(new Service("sms",true));
               services.add(new Service("netBanking",true));
               services.add(new Service("check",false));
               builder=Response.status(200);
               builder.entity(new GenericEntity<List<Service>>(services,List.class));
               response=builder.build();
               return  response;
          }
          builder=Response.status(401);
          builder.entity("Not a valid Account No");
          response=builder.build();
          return response;
     }

}
```

http://localhost:7777/ServerResponse/api/netbanking/services/A-30979446856

```
[
  {
    "serviceName": "sms",
    "status": true
  },
  {
    "serviceName": "netBanking",
    "status": true
  },
  {
    "serviceName": "check",
    "status": false
  }
]
```

**Ex-6**

**If we are using Collection type as return type then Produces should not be XML Type because XML contains only one root element ,If we send collection type then multiple root element will be created so JAX-RS will throw an Exception.**

```
@GET
     @Produces({MediaType.APPLICATION_XML})
     @Path("/service/{accountNo}")
     public List<Service> getServices1(@PathParam("accountNo")String accountNo){
          Response response=null;
          ResponseBuilder builder=null;
          List<Service> services= null;
               services=new ArrayList<Service>();
               services.add(new Service("sms",true));
               services.add(new Service("netBanking",true));
               services.add(new Service("check",false));
               return  services;
     }

}
```

```
com.sun.xml.bind.v2.runtime.IllegalAnnotationsException: 1
counts of IllegalAnnotationExceptions
com.sr.dto.Service does not have a no-arg default constructor.
     this problem is related to the following location:
          at com.sr.dto.Service
```

## 1.Client API with @PathParam

**Resource**

```
@Path("/netbanking")
public class NetBanking {
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	@Path("/balance/{accountNo}")
	//http://localhost:7777/ServerResponse/api/netbanking/balance/ac3001
	public Response getBalance(@PathParam("accountNo")String accountNo){
		Response response=null;
		ResponseBuilder builder=null;
		builder=Response.ok("Your Balance is :"+333.00);
		response=builder.build();
		return response;
	}
}
```

**Client**

```
public class ClientRequest {
	public static void main(String[] args) {
		ClientBuilder builder=ClientBuilder.newBuilder();
		Client client=builder.build();
		WebTarget target=client.target("http://localhost:7777/ServerResponse/api/netbanking");
		target=target.path("/balance/{accountNo}");
		target=target.resolveTemplate("accountNo","9040010697");
		Response response=target.request().get();
		if (response.getStatus()==200) {
			String out=response.readEntity(String.class);
			System.out.println(out);
		}
		else{
			System.out.println(response.getStatus());
		}
	}
}
```

## 2.Client API with@PathParam and @QueryParam

Resource

```
@Path("/netbanking")
public class NetBanking {
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	@Path("/balance/{accountNo}")
	public Response getBalance(@PathParam("accountNo")String accountNo,@QueryParam("mobile")String mobileNo){
		Response response=null;
		ResponseBuilder builder=null;
		builder=Response.ok("Dear "+mobileNo+" Your Balance is :"+333.00);
		response=builder.build();
		return response;
	}
}
```

ClientAPI

```
public class ClientRequest {
	private final String BASEURI="http://localhost:7777/ServerResponse/api/netbanking";
	public void getBalance(String accountNo,String mobile) {
		ClientBuilder builder=ClientBuilder.newBuilder();
		Client client=builder.build();
		WebTarget target=client.target(BASEURI);
		target=target.path("/balance/{accountNo}");
		target=target.resolveTemplate("accountNo",accountNo);
		target=target.queryParam("mobile",mobile);

		Response response=target.request().get();
		if (response.getStatus()==200) {
			String out=response.readEntity(String.class);
			System.out.println(out);
		}
		else{
			System.out.println(response.getStatus());
		}
	}
}
```

```
public class RSClientTest {
	public static void main(String[] args) {
		ClientRequest request=new ClientRequest();
		request.getBalance("ac1001","9040010697");
	}
}
```

## 3.Send Request in one line

```
public class ClientRequest {
     private final String BASEURI = "http://localhost:7777/ServerResponse/api/netbanking";

     public void getBalance(String accountNo, String mobile) {

          /* ClientBuilder builder=ClientBuilder.newBuilder(); Client
           client=builder.build();
           WebTarget target=client.target(BASEURI);
           target=target.path("/balance/{accountNo}");
           target=target.resolveTemplate("accountNo",accountNo);
           target=target.queryParam("mobile",mobile);
            Response resp=target.request().get();*/

          /*Response response=ClientBuilder.newBuilder().build()
                        .target(BASEURI).path("/balance/{accountNo}")
                        .resolveTemplate("accountNo",accountNo)
                        .queryParam("mobile",mobile).request().get();*/

          Response response = ClientBuilder.newClient().target(BASEURI).path("/balance/{accountNo}")
                    .resolveTemplate("accountNo", accountNo).queryParam("mobile", mobile).request().get();

          if (response.getStatus() == 200) {
               String out = response.readEntity(String.class);
               System.out.println(out);
          } else {
               System.out.println(response.getStatus());
          }
     }
}
```

## Header Data

Resource

```
@Path("/doctor")
public class DoctorResource {
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  @Path("/contact/{dName}/{dept}")
  public String getContactNo(@PathParam("dName")String doctorName,
                             @PathParam("dept")String department,
                             @HeaderParam("user")String user){
     return "Doctor Name: "+doctorName+" Dept:"+department+"
               Contact No:9853001547 and user is "+user;
  }
```

```
public class ClientRequest {
     private final String BASEURI = "http://localhost:8085/TemplateParameters/api/doctor";

     public void getContact(String doctorName, String dept,String user) {

          Response response=ClientBuilder.newClient().target(BASEURI)
                         .path("/contact/{dName}/{dept}")
                         .resolveTemplate("dName", doctorName)
                         .resolveTemplate("dept", dept).request()
                         .header("user",user).get();

          if (response.getStatus() == 200) {
               String out = response.readEntity(String.class);
               System.out.println(out);
          } else {
               System.out.println(response.getStatus());
          }
     }
}
```

## Cookie

**Resource**

```
@Path("/doctor")
public class DoctorResource {
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  @Path("/contact/{dName}/{dept}")
  public String getContactNo(@PathParam("dName")String doctorName,
                             @PathParam("dept")String department,
                             @CookieParam("user")String user) {
     return "Doctor Name: "+doctorName+" Dept:"+department+"
               Contact No:9853001547 and user is "+user;
  }
```

```
public class ClientRequest {
     private final String BASEURI = "http://localhost:8085/TemplateParameters/api/doctor";

     public void getContact(String doctorName, String dept,String user) {

          Response response=ClientBuilder.newClient().target(BASEURI)
                         .path("/contact/{dName}/{dept}")
                         .resolveTemplate("dName", doctorName)
                         .resolveTemplate("dept", dept).request()
                          .cookie("user",user).get();
          if (response.getStatus() == 200) {
               String out = response.readEntity(String.class);
               System.out.println(out);
          } else {
               System.out.println(response.getStatus());
          }
     }
}
```

## FormParam

```
@Path("/drCar")
public class DrCar {
     @POST
     @Produces(MediaType.APPLICATION_FORM_URLENCODED)
     @Path("/search")
     public String search(@FormParam("model")String model,
                          @FormParam("manufacturer")String manufacturer,
                          @FormParam("year")String year) {
          return "Model :"+model+" manufacturer "+manufacturer+" year"+year;
     }
}
```

```
public class ClientRequest {
     private final String BASEURI = "http://localhost:7777/FormParam/api/drCar";
     public void getContact(String model, String manufacturer,String year) {

          Form form=new Form();
          form.param("model", model);
          form.param("manufacturer", manufacturer);
          form.param("year", year);

          Response response=ClientBuilder
                           .newClient().target(BASEURI).path("/search")
                           .request().post(Entity.form(form));

          if (response.getStatus() == 200) {
               String out = response.readEntity(String.class);
               System.out.println(out);
          } else {
               System.out.println(response.getStatus());
          }
     }
}
```


**Object Of data (XML)**

**Provider**

```java
@XmlRootElement(name="account-info")
@XmlType
@XmlAccessorType(XmlAccessType.FIELD)
public class AccountInfo {
     private String accountNo;
     private double amount;
     //Setter & Getter
}
```

```java
@XmlRootElement(name="transaction-info")
@XmlType
@XmlAccessorType(XmlAccessType.FIELD)
public class TransactionIn {
     private String transactionId;
     private double amount;
     private String status;
     //Setters & Getters
}
```

```java
@Path("/netbanking")
public class Netbanking {
     @PUT
     @Consumes({MediaType.APPLICATION_XML,MediaType.APPLICATION_JSON})
     @Produces({MediaType.APPLICATION_XML,MediaType.APPLICATION_JSON})
     @Path("/transfer")
     public  void transfer(final AccountInfo accountInfo,
                         @Suspended final AsyncResponse asycResponse){
          new Thread(){
               public void run(){
                    TransactionIn info=new TransactionIn();
                    info.setTransactionId(UUID.randomUUID().toString());
                    info.setAmount(accountInfo.getAmount());
                    asycResponse.resume(Response.ok(info).build());
               }
          }.start();
     }
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
id="WebApp_ID" version="3.0">
     <servlet>
          <servlet-name>jersey</servlet-name>
          <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
          <init-param>
               <param-name>jersey.config.server.provider.packages</param-name>
               <param-value>com.jsa.resources</param-value>
          </init-param>
     <!-- <init-param>
               <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
               <param-value>true</param-value>
          </init-param> -->
          <load-on-startup>1</load-on-startup>
          <async-supported>true</async-supported>
     </servlet>
     <servlet-mapping>
          <servlet-name>jersey</servlet-name>
          <url-pattern>/api/*</url-pattern>
     </servlet-mapping>
</web-app>
```

**Client**

```
    public void transfer(String acNo,double amount){
         AccountInfo ac=new AccountInfo();
         ac.setAccountNo(acNo);
         ac.setAmount(amount);
         Response response=ClientBuilder.newClient().target(BASEURI).path("/transfer")
                   .request().put(Entity.entity(ac, MediaType.APPLICATION_XML));
         if (response.getStatus() == 200) {
              String out = response.readEntity(String.class);
              System.out.println(out);
         } else {
              System.out.println(response.getStatus());
         }
    }
}
```

```
public class RSClientTest {
     public static void main(String[] args) {
          ClientRequest request=new ClientRequest();
          request.transfer("30979446856", 3000.00);
     }
}
```

## Key Value Pair (Custom Content Handler)

```
@KeyValueType
public class Membership {
private String memberName;
private String password;
private String gender;
private String mobile;
//Setters & Getters
}
```

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface KeyValueType {

}
```

```
@KeyValueType
public class AccountInfo {
private String memberId;
private String mobile;
//Setters & Getters
}
```

```
public class ClientRequest {
     private final String BASEURI = "http://localhost:7777/CustomContentHandler/api/amazon";
          public void register(String memberShip){
          Response response=ClientBuilder.newClient().target(BASEURI).path("/register")
                    .request().post(Entity.entity(memberShip, MediaType.TEXT_PLAIN));

          if (response.getStatus() == 200) {
               String out = response.readEntity(String.class);
               System.out.println(out);
          } else {
               System.out.println(response.getStatus());
          }
     }
}
```

```
public class RSClientTest {
     public static void main(String[] args) {
          ClientRequest request=new ClientRequest();
          request.register("memberName=Dhananjaya;
                              password=realspeed;gender=male;mobile=9040010697");
     }
}
```

## Async Response (Server Side)

```java
@XmlRootElement(name="account-info")
@XmlType
@XmlAccessorType(XmlAccessType.FIELD)
public class AccountInfo {
     private String accountNo;
     private double amount;
     //Setters & Getters
}
```

```java
@XmlRootElement(name="transaction-info")
@XmlType
@XmlAccessorType(XmlAccessType.FIELD)
public class TransactionIn {
     private String transactionId;
     private double amount;
     private String status;
```

```java
@Path("/netbanking")
public class Netbanking {
     @PUT
     @Consumes(MediaType.APPLICATION_XML)
     @Produces(MediaType.APPLICATION_XML)
     @Path("/transfer")
     public  void transfer(final AccountInfo accountInfo,
                    @Suspended final AsyncResponse asycResponse){
          new Thread(){
               public void run(){
                    TransactionIn info=new TransactionIn();
                    info.setTransactionId(UUID.randomUUID().toString());
                    info.setAmount(accountInfo.getAmount());
                    asycResponse.resume(Response.ok(info).build());
               }
          }.start();
     }
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID"
version="3.0">
     <servlet>
          <servlet-name>jersey</servlet-name>
          <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
          <init-param>
               <param-name>jersey.config.server.provider.packages</param-name>
               <param-value>com.jsa.resources</param-value>
          </init-param>
          <async-supported>true</async-supported>
     </servlet>
     <servlet-mapping>
          <servlet-name>jersey</servlet-name>
          <url-pattern>/api/*</url-pattern>
     </servlet-mapping>
</web-app>
```

## Async API Client side (Future)

```
@Path("/bse")
public class BseStockExchange {
     @GET
     @Produces(MediaType.TEXT_PLAIN)
     @Path("/{stockName}")
     public double getStockPrice(@PathParam("stockName")String stockName){
          return new Random().nextInt(99999);
     }
}
```

```
public class ClientRequest {
     private final String BASEURI = "http://localhost:8085/AsyncWebApp/api/bse";
     public void reload(List<String> stockNames) throws InterruptedException, ExecutionException{
          new Thread(){
               @Override
               public void run() {
                    Future<Response> fResp=null;
                    List<Double> prices=new ArrayList<Double>();
                    for (String stockName : stockNames) {
                         fResp=ClientBuilder.newClient()
                              .target(BASEURI).path(stockName)
                              .resolveTemplate("stockName", stockName)
                              .request().async().get();
                         Response response = null;
                         try {
                              response = fResp.get();
                         } catch (InterruptedException | ExecutionException e) {}
                         double price=response.readEntity(Double.class);
                         prices.add(price);
                         System.out.println("Stock Price : "+price);
                    }
               }

          }.start();
     }
}
```

```
public class RSClientTest {
     public static void main(String[] args)
               throws InterruptedException, ExecutionException {
          ClientRequest request=new ClientRequest();
          List<String>stockNames=new ArrayList<String>();
          stockNames.add("cipla");
          stockNames.add("huge");
          stockNames.add("frog");
          request.reload(stockNames);
     }
}
```

## Async API Client Side Invocation Handler

```
public class ClientRequest {
     private final String BASEURI = "http://localhost:8085/AsyncWebApp/api/bse";
     public void reloadCallBack(List<String> stockNames) {
          new Thread(){
               @Override
               public void run() {
                    for (String stockName : stockNames) {
                         ClientBuilder.newClient()
                              .target(BASEURI).path(stockName)
                              .resolveTemplate("stockName", stockName)
                              .request().async().get(new CallBackHandler());
                    }
               }

          }.start();
     }
```

```
     private final class CallBackHandler implements InvocationCallback<Response>{
          @Override
          public void completed(Response response) {
               double price=response.readEntity(Double.class);
               System.out.println("Stock Price : "+price);
          }
          @Override
          public void failed(Throwable throwable) {

          }
     }
}
```

## Caching

### Expires

```
@Path("/bluedart")
public class Bluedart {
     @GET
     @Produces(MediaType.TEXT_PLAIN)
     @Path("{awbNo}")
     public Response track(@PathParam("awbNo")String awbNo){

          Calendar calender=Calendar.getInstance(TimeZone.getTimeZone("GMT"));
          //calender.set(year, month, date, hourOfDay, minute, second);
          calender.set(2017, 05, 10, 00, 00, 00);
          Date expireDt=Calendar.getInstance().getTime();
          Response response=Response.ok("Your awbNo is "+awbNo+"
                         and status is in-transit").expires(expireDt).build();
          return response;
     }

}
```

http://localhost:7777/Cache/api/bluedart/35

### Cache-Control

```
@Path("/bluedart")
public class Bluedart {
     @GET
     @Produces(MediaType.TEXT_PLAIN)
     @Path("{awbNo}")
     public Response getDetails(@PathParam("awbNo")String awbNo){

          CacheControl cc=new CacheControl();
          cc.setMaxAge(200000);
          cc.setMaxAge(300);
          cc.setPrivate(true);
          cc.setNoStore(true);
          cc.setMustRevalidate(false);
          Response response=Response.ok("Your awbNo is "+awbNo+"
                    and status is in-transit").cacheControl(cc).build();
          return response;
     }
}
```