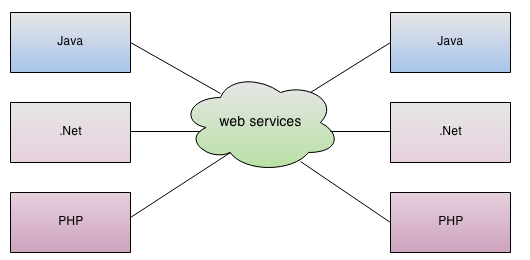
# **Java WEB Services**

**Web Service:** A Web Service is can be defined by following ways:

* It is a client server application or application component for communication.
* method of communication between two devices over network.
* It is a software system for interoperable machine to machine communication.
* It is a collection of standards or protocols for exchanging information between two devices or application.

**Web Service Components:** There are three major web service components.

1. SOAP: Simple Object Access Protocol (SOAP) is a XML-based **protocol** for accessing web services
2. WSDL: Web Services Description Language (WSDL) is a xml **document** containing information about web services such as method name, method parameter and how to access it. WSDL is a part of UDDI. It acts as an **interface** between web service applications.
3. UDDI: Universal Description, Discovery and Integration(UDDI) is a XML based framework for describing, discovering and integrating web services

**Advantage of SOAP Web Services:**

* **WS Security**: SOAP defines its own security known as WS Security.
* **Language and Platform independent**: SOAP web services can be written in any programming language and executed in any platform.

**Disadvantage of Web Services:**

* **Slow:** SOAP uses XML format that must be parsed to be read with many standards that must be followed. So it is slow and consumes more bandwidth and resource.
* **WSDL Dependent:** SOAP uses WSDL and doesn't have any other mechanism to discover the service.

**Advantages of RESTful Web Services**

* **Fast**: RESTful Web Services are fast because there is no strict specification like SOAP. It consumes less bandwidth and resource.
* **Language and Platform independent**: RESTful web services can be written in any programming language and executed in any platform.
* **Can use SOAP**: RESTful web services can use SOAP web services as the implementation.
* **Permits different data format**: RESTful web service permits different data format such as Plain Text, HTML, XML and JSON.

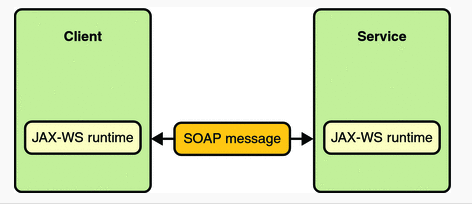
**SAOP Vs REST**

|  |  |  |
| --- | --- | --- |
| **No.** | **SOAP** | **REST** |
| 1) | SOAP is a **protocol**. | REST is an **architectural style**. |
| 2) | SOAP stands for **Simple Object Access Protocol**. | REST stands for **REpresentational State Transfer**. |
| 3) | SOAP **can't use REST** because it is a protocol. | REST **can use SOAP** web services because it is a concept and can use any protocol like HTTP, SOAP. |
| 4) | SOAP **uses services interfaces to expose the business logic**. | REST **uses URI to expose business logic**. |
| 5) | **JAX-WS** is the java API for SOAP web services. | **JAX-RS** is the java API for RESTful web services. |
| 6) | SOAP **defines standards**to be strictly followed. | REST does not define too much standards like SOAP. |
| 7) | SOAP **requires more bandwidth** and resource than REST. | REST **requires less bandwidth** and resource than SOAP. |
| 8) | SOAP **defines its own security**. | RESTful web services **inherits security measures** from the underlying transport. |
| 9) | SOAP **permits XML** data format only. | REST **permits different** data format such as Plain text, HTML, XML, JSON etc. |
| 10) | SOAP is **less preferred** than REST. | REST **more preferred** than SOAP. |

**Java Web Services Tutorial**

**JAX**-**WS**: Java API for XML Web Services, is a set of APIs for creating web services in XML format (SOAP). JAX-WS provides many annotations to simplify the development and deployment for both web service clients and web service providers (endpoints).

In general words, “web service endpoint” is a service which published outside for user to access; where “web service client” is the party who access the published service.

There are mainly two types of web services.

1. SOAP web services (use Apache CXF 3.1.x with JAX-WS)
   1. RPC Style (Remote Procedure Call)
   2. Document Style
2. RESTful web services (use Jersey with JAX-RX)
   1. Jersey Style
   2. RESTeasy Style

* Apache CXF 3.1.x for SOAP Web Service WSDL (Web Service Description Language)
* Jersey for JAX-RX for ReST Web Service WADL (Representational State Transfer)

Test URL: http://localhost:8030/employee/svc/ws/Employee?wsdl

There are tons of other stuff you can explore in Apache CXF like Creating dynamic clients, interceptors, leveraging other transport protocol, webservice over https etc.

**Handling Session in SOAP Web Service**

Following four steps you need to implements:

1. Add the @Resource (defined by Common Annotations for the Java Platform, JSR 250) to the top of your Web Service.
2. Add a variable of type WebServiceContext that will have the context injected into it.
3. Using the Web Service context, get the HttpSession object.
4. Save objects in the HttpSession using the setAttribute method and retrieve saved object using getAttribute. Objects are identified by a string value you assign.

Coding Example:

@WebService

public class ShoppingCart {

**@Resource**  // Step 1

**private WebServiceContext wsContext;**  // Step 2

public int addToCart(Item item) {

// Find the HttpSession

**MessageContext mc = wsContext.getMessageContext();** // Step 3

**HttpSession session = ((javax.servlet.http.HttpServletRequest)mc.get(MessageContext.SERVLET\_REQUEST)).getSession();**

if (session == null)

throw new WebServiceException("No HTTP Session found");

// Get the cart object from the HttpSession (or create a new one)

**List<Item> cart = (List<Item>)session.getAttribute("myCart");** // Step 4

if (cart == null)

cart = new ArrayList<Item>();

// Add the item to the cart (note that Item is a class defined

// in the WSDL)

cart.add(item);

// Save the updated cart in the HTTPSession (since we use the same

// "myCart" name, the old cart object will be replaced)

**session.setAttribute("myCart", cart);**

// return the number of items in the stateful cart

return cart.size();

}

}

**Web Service Security**

#### **Wire-level security**

* Assurance between client and web service that they are the only one communicating.
* Data encryption.
* Assurance that received message is same as sent message.

#### **By User authentication and authorization**

* Authentication (viz. login to application) is appropriate credentials to gain access.
* Authorization is users-role security. Users might be restricted to some resources based on their roles.

**Example: Authentication and authorization managed by container**

Include following code in the web.xml descriptor

<security-role>

<description>Admin role</description>

<role-name>admin</role-name>

</security-role>

<security-constraint>

<web-resource-collection>

<web-resource-name>UserRoleSecurity</web-resource-name>

<url-pattern>/calc</url-pattern>

</web-resource-collection>

<auth-constraint>

<role-name>admin</role-name>

</auth-constraint>

<user-data-constraint>

<transport-guarantee>CONFIDENTIAL</transport-guarantee>

</user-data-constraint>

</security-constraint>

<login-config>

<auth-method>BASIC</auth-method>

</login-config>

**Example: Authentication at Application Level with JAX-WS**

On the **web service client** site, just put your “username” and “password” into request header.

Map<String, Object> req\_ctx = ((BindingProvider)port).getRequestContext();

req\_ctx.put(BindingProvider.ENDPOINT\_ADDRESS\_PROPERTY, WS\_URL);

Map<String, List<String>> headers = new HashMap<String, List<String>>();

headers.put("Username", Collections.singletonList("mkyong"));

headers.put("Password", Collections.singletonList("password"));

req\_ctx.put(MessageContext.HTTP\_REQUEST\_HEADERS, headers);

On the **web service server** site, get the request header parameters via WebServiceContext.

@Resource

WebServiceContext wsctx;

@Override

public String method() {

MessageContext mctx = wsctx.getMessageContext();

//get detail from request headers

Map http\_headers = (Map) mctx.get(MessageContext.HTTP\_REQUEST\_HEADERS);

List userList = (List) http\_headers.get("Username");

List passList = (List) http\_headers.get("Password");

//...

That’s all, now, your deployed JAX-WS is supported application level authentication

**Example: Wire-level security using HTTPS**

To make web service more secure, we can use HTTPS instead of HTTP. It addresses three security services over transport services that HTTP provides; Peer authentication, confidentiality, and integrity.

**Example: Securing web service under tomcat**

Next step is to secure the web service or to enable https. To do this, go to Tomcat’s conf directory and edit server.xmlfile.  
But first things first, we will need to create a digital certificate. We can use Java’s keytool utility to generate the same.

The command would be like: keytool -genkey -alias tomcat -keyalg RSA

By default a digital certificate file with name .keystore shall be created in the user’s home directory. Now to configure this file to enable https, we shall edit connector configuration in server.xml as mentioned above. The new configuration shall be like:

*server.xml (partial)*

|  |  |
| --- | --- |
| 1 | <Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true" |

|  |  |
| --- | --- |
| 2 | maxThreads="150" scheme="https" secure="true" clientAuth="false" |

|  |  |
| --- | --- |
| 3 | sslProtocol="TLS" keystoreFile="/fusman/.keystore" /> |

OR:

<Connector SSLEnabled="true" URIEncoding="UTF-8" keyAlias="tomcat" keystoreFile="${catalina.base}/conf/.keystore" port="448" protocol="org.apache.coyote.http11.Http11NioProtocol" scheme="https" secure="true"/>

<Connector URIEncoding="UTF-8" port="8030" redirectPort="448"/>

<!-- Define an AJP 1.3 Connector on port 8009 -->

<Connector port="8009" protocol="AJP/1.3" redirectPort="8448" />

And the Web Service URL will be:

<https://localhost:448/usman-ws-birthday/svc/ws/BirthdayControl?wsdl>

Form

<http://localhost:8030/usman-ws-birthday/svc/ws/BirthdayControl?wsdl>

C:\Users\fusman>keytool -genkey -alias tomcat -keyalg RSA

Enter keystore password:

Re-enter new password:

What is your first and last name?

[Unknown]: farkalit

What is the name of your organizational unit?

[Unknown]: Vertex

What is the name of your organization?

[Unknown]: Vertex

What is the name of your City or Locality?

[Unknown]: Gurgaon

What is the name of your State or Province?

[Unknown]: Haryana

What is the two-letter country code for this unit?

[Unknown]: IN

Is CN=farkalit, OU=Vertex, O=Vertex, L=Gurgaon, ST=Haryana, C=IN correct?

[no]: y

Enter key password for <tomcat>

(RETURN if same as keystore password):

C:\Users\fusman>

For keytool detail read

<http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/keytool.html>

**Service Oriented Architecture**

The Service Oriented Architecture is an architectural design which includes collection of services in a network which communicate with each other. The complication of each service is not noticeable to other service. The service is a kind of operation which is well defined, self-contained that provides separate functionality such as checking customer account details, printing bank statements etc and does not depend on the state of other services.

**Why to use SOA?**

* SOA is widely used in market which responds quickly and makes effective changes according to the market situations.
* The SOA keep secret the implementation details of the subsystems.
* It allows interaction of new channels with customers, partners and suppliers.
* It authorizes the companies to select software or hardware of their choice as it acts as platform independence.

**Features**

* SOA uses interfaces which solves the difficult integration problems in large systems.
* SOA communicates customers, providers and suppliers with messages by using the XML schema.
* It uses the message monitoring to improve the performance measurement and detects the security attacks.
* As it reuses the service, there will be lower software development and management costs.

**Advantages**

* SOA allows reuse the service of an existing system alternately building the new system.
* It allows plugging in new services or upgrading existing services to place the new business requirements.
* It can enhance the performance, functionality of a service and easily makes the system upgrade.
* SOA has capability to adjust or modify the different external environments and large applications can be managed easily.
* The companies can develop applications without replacing the existing applications.
* It provides reliable applications in which you can test and debug the independent services easily as compared to large number of code.

**Disadvantages**

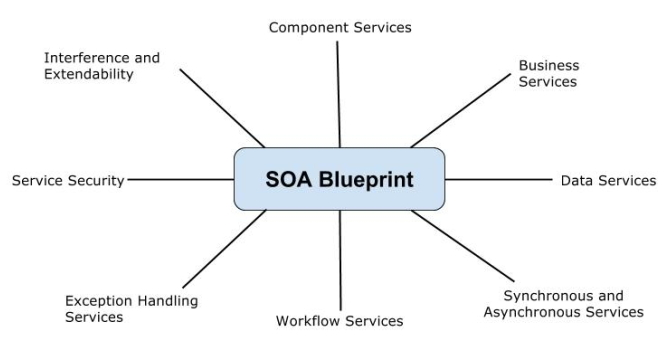
* SOA requires high investment cost (means large investment on technology, development and human resource).
* There is greater overhead when a service interacts with another service which increases the response time and machine load while validating the input parameters.
* SOA is not suitable for GUI (graphical user interface) applications which will become more complex when the SOA requires the heavy data exchange.

**The SOA blueprint contains some following goals:**

* Requirements of design principles
* Specific tasks of design principles
* Interaction of services
* Details of integration scenario
* Templates for the specific tasks

**SOA Blueprints Concepts**

The following figure shows SOA blueprint with different concepts:



**Considerations in SOA**

There are some considerations must be covered in SOA:

### **Infrastructure**

* Accessible of requirements
* Performance requirements
* Platform for system

### **Architecture**

* Models of domain and service
* Organization of services
* Process of integrating the structure
* Quality of the service
* Message exchange patterns

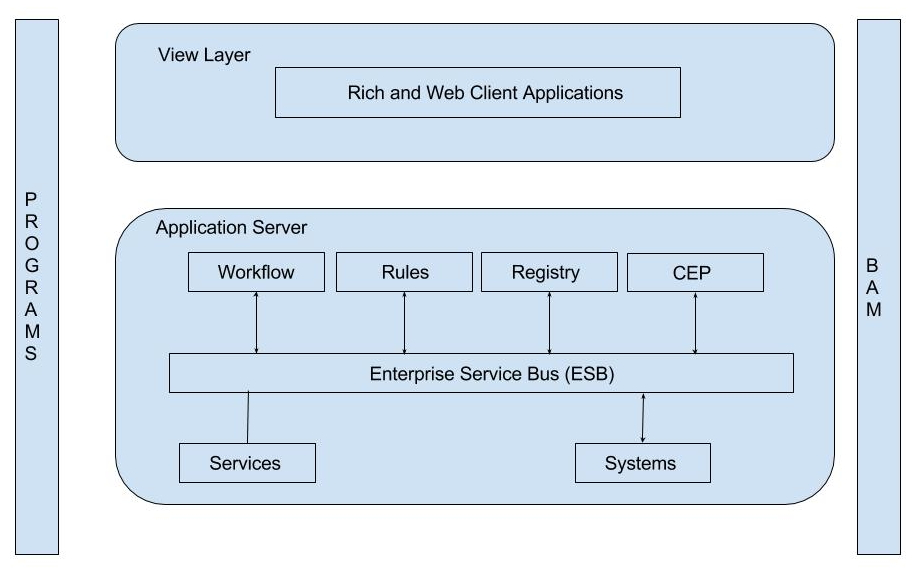
### **Development**

* Design guidelines for project development
* Required tools for project
* Validation and modification required things
* Handling errors
* Security for service access

### **Administration**

* Managing and building
* Testing and deploying the project
* Location of data stored and registering the application

The following figure shows SOA blueprint with different classes:



SOA contains the main functions of blueprint which are called as *Programs* and *BAM*.

## **Java Web Services Book Reading**

SEI: Service Endpoint Interface. The implementation is called the

SIB: Service Implementation Bean.

Just: Write all the code of chapter1 from 1 to 4 examples. Start the java class (No need to start Tomcat Server).

Goto browser and give http://localhost:8030/ts?wsdl (You will see all the wsdl on the browser)

WSDL contains portType (Which is equivalent to java interface/SEI)

WSDL contains bindings (Which is equivalent to java implemented class/SIB)

Note: commercial products available for tracking the SOAP traffic. are SOAP scope, NetSniffer, and Wireshark.

@SOAPBinding(style = Style.RPC): This annotation requires that the service use only very simple types such as string and integer.

@SOAPBinding(style = Style.DOCUMENT): By contrast, the Teams service uses richer data types, which means that Style.DOCUMENT, the default, should replace Style.RPC.The document style does

require more setup.

F:\Usman\NetBeansProjects\usman-webservice-reading\chapter1\target\classes>wsgen -cp . com.farkalit.webreading.chapter1.team.Teams (It generates all the artifacts in javax folder)

>javax.GetTeam.class

>javax.GetTeamResponse.class

>javax.GetTeams.class

>javax.GetTeamsResponse.class

First run the TeamPublisher class then run the command:

F:\Usman\NetBeansProjects\usman-webservice-reading\chapter1\target\classes> wsimport -p teamsC -keep http://localhost:8030/teams?wsdl This utility generates various classes in the subdirectory

teamsC (the -p flag stands for package) (option -keep indicates that the source files should be kept)

>teamsC.GetTeam.class (and other 19 class=Total 20 classes)

Endpoint publisher has been single threaded and, therefore, capable of handling only one client request at a time:

At issue, then, is how to make the Endpoint publisher multithreaded. The JWS framework supports Endpoint multithreading without forcing the

programmer to work with difficult, error-prone constructs such as the synchronized block or the wait and notify method invocations.

>http://localhost:8030/teams?xsd=1 will show XSD (For Document Style Web Service)

>http://localhost:8030/ts?xsd=1 will show XSD (Provided its javawx artifacts has been generated)

Here is XSD:

<xs:schema xmlns:tns="http://ts.chapter1.webreading.farkalit.com/" xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0" targetNamespace="http://ts.chapter1.webreading.farkalit.com/">

<xs:element name="getTimeAsElapsed" type="tns:getTimeAsElapsed"/>

<xs:element name="getTimeAsElapsedResponse" type="tns:getTimeAsElapsedResponse"/>

<xs:element name="getTimeAsString" type="tns:getTimeAsString"/>

<xs:element name="getTimeAsStringResponse" type="tns:getTimeAsStringResponse"/>

<xs:complexType name="getTimeAsString">

<xs:sequence/>

</xs:complexType>

<xs:complexType name="getTimeAsStringResponse">

<xs:sequence>

<xs:element name="return" type="xs:string" minOccurs="0"/>

</xs:sequence>

</xs:complexType>

<xs:complexType name="getTimeAsElapsed">

<xs:sequence/>

</xs:complexType>

<xs:complexType name="getTimeAsElapsedResponse">

<xs:sequence>

<xs:element name="return" type="xs:long"/>

</xs:sequence>

</xs:complexType>

</xs:schema>

% wsimport -keep -p awsClient http://ecs.amazonaws.com/AWSECommerceService/AWSECommerceService.wsdl

OR

% wsimport -keep -p awsClient http://ecs.amazonaws.com/AWSECommerceService/2013-08-01/AWSECommerceService.wsdl Recall that the -p awsClient part of the command generates a

package (and, therefore, a subdirectory) named awsClient.

@WebParam(name = "MarketplaceDomain", targetNamespace = "http://webservices.amazon.com/AWSECommerceService/2013-08-01")

String marketplaceDomain,

String awsAccessKeyId,

String associateTag,

String xmlEscaping,

String validate,

ItemSearchRequest shared,

List<ItemSearchRequest> request,

Holder<OperationRequest> operationRequest,

Holder<List<Items>> items);

"http://webservices.amazon.com/AWSECommerceService/2008-03-03",

>A SOAP message is a one-way transmission from a sender to a receiver; hence, the fundamental message exchange pattern (MEP) for SOAP is one way.

>MEPs such as request/response and solicit/response can be put together in suitable ways to support more expansive conversational patterns as needed.

>JWS provides a handler framework that allows application code to inspect and manipulate outgoing and incoming SOAP messages.

>A SOAP message handler provides a mechanism for intercepting the SOAP message in both the request and response of the Web Service.

>SOAP message handlers are used to intercept the SOAP message as they make their way from the client to the end-point service and vice-versa.

These handlers intercept the SOAP message for both the request and response of the Web Service.

> A few typical scenarios where you would be using SOAP Message handlers are: to encrypt and decrypt messages, to support logging,

caching and in some cases auditing, and in rare cases to provide transaction management as well.

>The JWS handler framework thus encourages the chain of responsibility pattern, which Java servlet programmers encounter when using filters.

>A handler can be injected into the framework in two steps:

1. One step is to create a handler class, which implements the Handler interface in the javax.xml.ws.handler package. The class that implements either the LogicalHandler or the SOAPHandler interface needs to define three methods for either interface type, including handleMessage, which gives the programmer access to the underlying message. The other two shared methods are handleFault and close.

2. The other step is to place a handler within a handler chain.

>JWS has two different ways to throw SOAP faults. The simplest way is to extend the Exception class. The other way, which takes more work, is to throw a fault from a handler.

>Similarities between Handlers and CXF Interceptors: JAX-WS handlers are internally implemented in CXF by use of interceptors, so by definition anything that can be done with the former can be done with the latter.

>The most important difference is that CXF Interceptors are split into two main categories: inbound and outgoing interceptors which are used to collect the server incoming message and the outgoing response. On the other hand when using JAX-WS handlers, the same handlers are activated both on request and reply.

> WebService Attachment (File and Image) (byte64 & MTOM/Image)

## **REST Web Service (REpresentational State Transfer):**

1. There is a **Resource** e.g.; The gross national product of Lithuania in 2001 is a resource, as is the Modern Jazz Quartet etc.

2. Every Resource have at least one **Representation**. In the web, Representations are MIME type. e.g.; text/html, image/jpeg; etc.

3. Resources have **State**. For example, Ernie Bank’s baseball accomplishments changed during his career with the Chicago

Cubs from 1953 through 1971 and culminated in his 1977 induction into the Baseball Hall of Fame.

So, we say that:

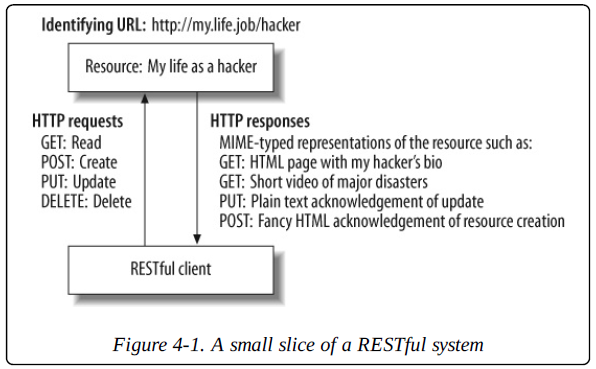
In a RESTful request targeted at a resource, the resource itself remains on the service machine. The requester typically receives a representation of the resource if the request succeeds. It is the representation that transfers from the service machine to the

requester machine. In different terms, a RESTful client issues a request that involves a resource, for instance, a request to read the resource. If this read request succeeds, a typed representation (for instance, text/html) of the resource is transferred from the

server that hosts the resource to the client that issued the request. The representation is a good one only if it captures the resource’s state in some appropriate way.

At the core of the RESTful approach is the insight that HTTP, despite the occurrence of Transport in its name, is an API and not simply a transport protocol. **HTTP** has its well-known verbs, officially known as methods.

|  |  |
| --- | --- |
| **HTTP verb** | **Meaning in CRUD terms** |
| POST | *Create* a new resource from the request data |
| GET | *Read* a resource |
| PUT | *Update* a resource from the request data |
| DELETE | *Delete* a resource |



HTTP also has standard response codes, such as 404 to signal that the requested resource could not be found, and 200 to signal that the request was handled successfully. In short, HTTP provides request verbs and MIME types for client requests and status codes (and MIME types) for service responses.

**Table shows some of the many HTTP status codes.**

|  |  |  |
| --- | --- | --- |
| **HTTP status code** | **Official reason** | **Meaning** |
| 200 | OK | Request OK. |
| 400 | Bad request | Request malformed. |
| 403 | Forbidden | Request refused. |
| 404 | Not found | Resource not found. |
| 405 | Method not allowed | Method not supported. |
| 415 | Unsupported media type | Content type not recognized. |
| 500 | Internal server error | Request processing failed. |

In general, status codes in the range of 100–199 are informational; those in the range of 200–299 are success codes; codes in the range of 300–399 are for redirection; those in the

range of 400–499 signal client errors; and codes in the range of 500–599 indicate server errors.

REST Client

1. REST Jersey client
2. HttpClient for REST services

Examples of REST Web Service:

**package** com.usman.jersey.first;

**import** javax.ws.rs.GET;

**import** javax.ws.rs.Path;

**import** javax.ws.rs.Produces;

**import** javax.ws.rs.core.MediaType;

*// The browser requests per default the HTML MIME type.*

*//Sets the path to base URL + /hello*

@Path("/hello")

**public** **class** **Hello** {

*// This method is called if TEXT\_PLAIN is request*

@GET

//@Path("{text}")

@Produces(MediaType.TEXT\_PLAIN)

**public** String sayPlainTextHello(@PathParam("text") String name) {

**return** "Hello Jersey"+ **name**;

}

}

REST Clint:

**package** com.usman.jersey.first.client;

**import** java.net.URI;

**import** javax.ws.rs.client.Client;

**import** javax.ws.rs.client.ClientBuilder;

**import** javax.ws.rs.client.WebTarget;

**import** javax.ws.rs.core.MediaType;

**import** javax.ws.rs.core.Response;

**import** javax.ws.rs.core.UriBuilder;

**import** org.glassfish.jersey.client.ClientConfig;

**public** **class** **Test** {

**public** **static** **void** main(String**[]** args) {

ClientConfig config = **new** ClientConfig();

Client client = ClientBuilder.newClient(config);

WebTarget target = client.target(getBaseURI());

String response = target.path("rest").

path("hello").request().

accept(MediaType.TEXT\_PLAIN).

get(Response.class)

.toString();

String plainAnswer= target.path("rest").path("hello/usmani").request()

.accept(MediaType.TEXT\_PLAIN).get(String.class);

System.out.println(response);

System.out.println(plainAnswer);

}

**private** **static** URI getBaseURI() {

**return** UriBuilder.fromUri("http://localhost:8080/com.usman.jersey.first").build();

}

}