Restful Services New Batch (25-Feb-2019) @11:00 AM
==

Pre-Requisites

Core Java

- OOPS
- Collections
- Exception Handling
- I/O streams
- Generics
- Annotations
- java.net package
- java.lang package

JDBC & Hibernate - Good to have
Spring - Required for Spring with Rest integration (3 days)
SOAP Webservices - Not required.

Servlets - Mandatory to know

- Servlet
- GenericServlet
- HttpServlet
- HttpServletRequest
- HttpServletResponse
- ServletConfig
- ServletContext
- RequestDispatcher
- Listeners
- Asynchronus Servlet
- WebServer & Application Server
- Deployment Process

XML Technologies - Required

- XML
- XSD
- JAX-P
- JAX-B
- JSON

Note-1: XML, XSD, JAX-P and XSD already convered in soap webservices.

Note-2 : JSON, JACKSON AND GSON will be covered in this course

Course Content

What is Distributed application
Why we need Distributed applications

What is Intereoperability
What are Distributed Technologies available
CORBA, RMI, EJB and Webservices
Why we need Restful Services
How Restful services are different from Webservices

JAX-RS 2.0 API

- JERSEY implementation
- REST EASY implementation
- JSON (jackson and Gson)
- Resource with Jersey
- Resource with RestEasy
- Jersey Bootstrappings
- RestEasy BootStrappings
- JAX-RS Injections
- Converters
- java.net client
- Apache Http Client
- Jersey client
- RestEasy client
- Spring with Rest integration
- RestTemplate
- SOAP UI and POSTMAN
- SWAGGER
- Exception Handling in Restful services
- Security in Restful Services (SSL)
- OAuth security
- Interview Quesstions

Course Duration : 1 month
Daily 1:30 mins
Timings : 11:00 AM - 1:00 PM

Facebook Group Name : Ashok IT School
My Email : ashok.javatraining@gmail.com
Youtube Channel : Ashok IT School

26-Feb-2019

What is Distributed application ?

Distributed applications (distributed apps) are applications or software that runs on multiple computers within a network at the same time and can be stored on servers or with cloud computing.

Unlike traditional applications that run on a single system, distributed applications run on multiple systems simultaneously for a single task or job.

To develop Distributed applications we need Distributed technologies.

- CORBA (Common request broker architecture)
- RMI (Remote method invocation)
- EJB (Enterprise java beans)
- Webservices

What is Webservice ?

- Webservice is a distributed technology which is used to develop distributed applications with intereoperability.

What are restful services?

Restful services are used to develop distributed applications with intereoperability.

What is intereoperability ?

Platform independent and language independent
java --- c#
java --- python
python --- java
java ---- C++

Why we need to know restful services ?

Now a days every application having business requirements
to communicate with other applications.

We can communicate from one app to another app using webservices.

There are some challenges involved in working with webservices.

To overcome challenges in webservice, Restful Services came into picture.

How restful services are different from webservices ?

Real adoptability is not available in webservices
Real Intereoperability is not available in webservice

Webservices specifications provided by WS-I.

WS-I (Webservices intereoperability)-non profitable org

WS-I released below 2 specifications to achieve intereoperability.

1) B.P 1.0 (Basic Profile) specification

- Sun adopted B.P 1.0 and provided jax-rs api.
- jax-rpc api (java api for xml remote procedural call)

2) B.P 1.1 specification
- Sun adopted B.P 1.1 and provided JA-WS API.
- jax-ws api(java api for xml webservices)

The above jax-rpc and jax-ws apis are partial. We need implementations to develop webservice.

JAX-RPC API IMPLEMENTATIONS

SI (SUN IMPLEMENTATIONS)
APACHE AXIS
ORACLE WEBLOGIC
IBM WEBSPHERE ETC...

JAX-WS API IMPLEMENTATIONS

RI (SUN REFERENCE IMPLEMENTATION)
APACHE AXIS 2
APACHE CXF
ORACLE WEBLOGIC
IBM WEBSPHERE
JBOSS ETC....

Using api and implementation we can develop both consumer and provider.

If we develop web service using jax-rpc or jax-ws API then that is called as soap webservice.

soap webservices also called as big web services.

27-Feb-2019

When webservices came into market every body started adopting and using webservices to develop distributed applications.

One PHD student is not happy with Webservices and he started doing his analysis and identified loop holes available in webservices.

That PHD student is Roy Fielding.

As part of his analysis he identified it is very difficult
to understand webservices and very difficult to adopt webservices.

Real intereoperability and Real adoptability is not available in webservices.

If you want to access webservice you should know what is soap and wsdl.

To overcome above challenges he identified that internet is running successfully. What makes internet is real adoptable and real intereoperable. If i can use same principles what internet is using then i can also achieve real intereoperability and real adoptability.

With that motivation he identified some principles and he released white paper with Rest Architecture principles.

Initially nobody is intrested to use Roy Fielding principles.

It took few years to bring lime light for Restful services.

Below are the Rest Architecture principles
----
1) Unique Addressbility
2) Uniform constraint interfaces (http methods)
3) Message oriented representation
4) Communication stateless
5) HATEOS (Hyper Media as an engine for state of an application)

Rest Architecture principles
----
1) Unique Addressbility : Every resource should bind to unique address

When we create a java class (ReportDao.java) in project we are only going to use that class.

When we create a servlet in our project, then we are not going to use that class in our project.

That servlet is going to access by others because servlet is a distributed component.

How others can access our Servlet ? - They should address our servlet.

Address of servlet is URL of Servlet.

Similarly when we create Rest component in project, we are not going to use that component in our project.

Rest component is a distibuted component.

Rest component is going to access by others or other applications.

To access our rest component they should know address of our rest component.

Every Rest component should bind to unique address.

To bind Rest component to url pattern we will use @Path annotation like below.

Note : If we use @Path annotation at class level then only that class be considered as Resource class other wise it would be like a normal java class.

```
@Path("/paytm")
public class PaytmResource{

    public String getBalance(long phno){
   //logic
   return "100.78 rs";
```

```
 }
}
```

Address : http://192.168.1.0:8080/Webapp/paytm

2) Uniform constraint interfaces

Uniform - same
constraint - limited

This is used to achieve real adoptability

When we develop distributed component, other persons or other
applications are going to access our distributed component.

When they can access our component ? - If they have the address they can access.

Only address is sufficient ?- No

Along with address they should know what is there in resource then only then can access our resource.

If we write our own methods with our names then it would be very difficult for others to access our component.

That's why what ever the methods we are writing in resource we should bind them to http designator methods.

GET, POST, PUT and DELETE.

```
@Path("/paytm")
public class PaytmResource{

   @GET
   public double getBalance(long phno){
 return 100.56;
   }

   @POST
   public boolean addBalance(long phno){
    //logic
   return true;
   }

}
```
----
http://192.8.9.0.1:8080/App/insurance

```
@Path("/insurance")
public class InsuranceResource{
     @GET
     public String m1(long ssn){
  //logic
  return "Approved";
     }

     @POST
     public String applyForPlan(String planName){
  //logic
  return "Success";
     }
}
```

```
@Path("/erail")
public class ERailResource{

     @GET - http designitor method
     public String m2(long ssn){
  return "Approved";
     }

     @POST
     public String m1(String planName){
  return "Success";
     }
}
```

Note : When we bind our methods to http methods like above then every body can understand our methods easily because then no need to know our method names. They need to know only http methods which are universal methods.

-to data - post request
-to get data --- get request
-to update data --- put request
-to delete data --- delete request

As we writing HTTP methods in our resource , we no need to describe our resource.

If we go for SOAP, we need to describe our provider using WSDL.

In Restful services, as we are using HTTP methods description is not required.

If need to describe our Rest resources we can use WSDL.

WSDL - Webservices description language
WADL - Webapplication description language.

28-Feb-2019

Create a Resource class and bind resource methods to Http methods.

@Path annotation we will use at class level to make our class as Resource class. Then it will become distributed component.

@Path annotation we can use at method level also (Optional).

```
@Path("/products")
public class ProductResource{

   @POST
 @Path("/add")
 public boolean addProduct(Product p){
  //logic
  return true;
 }

 @GET
 @Path("/{pid}")
   public Product getProductById(int pid){
  //logic
  return product;
 }

 @GET
 public List<Product> getAllProducts(){
  return productList;
 }

   @PUT
   public Product updateProduct(Product p){
  //logic
  return product;
 }

 @DELETE
 public boolean deleteProduct(int pid){
  //logic
  return true;
 }
}
```
----

http://192.8.9.0.1:8080/App/products - GET
http://192.8.9.0.1:8080/App/products/101 - GET
http://192.8.9.0.1:8080/App/products/add - POST

3) Message Oriented Representation

In soap based webservices we will have consumer and and provider.

Consumer and provider will exchance the data.

The data has to be exchanged only in soap format.

When we go restful services we will have client and resource.

Client and resource can exchage the data in multiple formats.

Like xml, json, html and text.

Restful services offering real intereoperability for us.

Client can decide in which format he wants to send data as an input to resource.

Client can decide in which format he wants respons from resource.

At resource side we will use below two annotations to specify input and output formats.

- @Consumes - to specify input format
- @Produces - to specify outut format

```
--------------------------------------------------------
@Path("/paytm")
public class PaytmResource{

   @GET
   public String getBalance(long phno){
  return "100.45 rs";
   }

   @GET
   @Path("/getAccData/{accId}")
   @Produces({"application/xml","application/json"})
   public AccountDetails getDetails(Integer accId){
  //logic
  return accDetails;
   }

   @POST
   @Consumes({"application/xml","application/json"})
   public String createAcc(AccountDetails accDetails){
  return "100.45 rs";
   }

}

---------------------------------------------------------
@Path("/insurance")
public class InsuranceResource {

     @POST
     @Consumes({"application/xml","application/json"})
     @Produces({"application/xml","application/json"})
     public PlanInfo applyForPlan(PersonInfo personInfo){
```

```
  //logic

  PlanInfo planInfo = new PlanInfo();
  //set data to planInfo

  return planInfo;
     }
}
```

----

4) Communication statless

Statefull Protocol : A protocol that requires keeping of the internal state on the server is known as a stateful protocol.

A TCP connection-oriented session is a 'stateful' connection because both systems maintain information about the session itself during its life.

Stateless Protocol : A protocol that will not remember the data which is exchanged between client and server.

Http is a stateless protocol.

5) HATEOS

Hypermedia As The Engine Of Application State is a component of the REST application architecture that distinguishes it from other network application architectures. With HATEOAS, a client interacts with a network application whose application servers provide information dynamically through hypermedia.

GET Request URL : http://192.168.0.1:8080/EmpApp/101

Response :

```
{
 "id" : 101,
 "name" : "Raju",
 "email" : "raju@ibm.com",
 "address" "http://192.168.0.1:8080/EmpApp/address/101"
}
```
