

Unit-4

Web Services

Spring REST

Web services are reusable software components which are available on the network for consumption by other applications, irrespective of their platform and technology. Web Services can either be SOAP or REST based. And, REST has become more popular in recent days.

Spring framework provides wonderful support to develop RESTful web services with strict adherence to the REST principles.

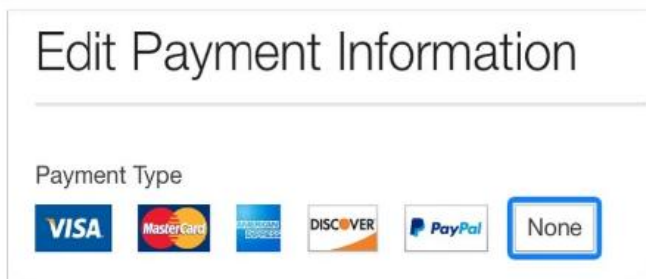
REST stands for **REpresentational State Transfer**. It is an architectural style that defines a set of rules and regulations to create web services. REST helps to build web services that are lightweight, maintainable and scalable. Following are the advantages of making the functionalities of an application as RESTful services.

- Interoperability
- Resource sharing
- Reusability
- Independent client-server architecture
- Stateless transactions

This course will give details about Spring's support for REST. And, this course uses code based demonstrations to explain how requests and data are handled in a Spring REST application. Additionally, this course gives a wide coverage on documenting and versioning the Spring REST endpoints.

Why Web Services?

Real Time Scenarios



A lot of companies which make money every second like Amazon communicates with various vendors to get the product details.

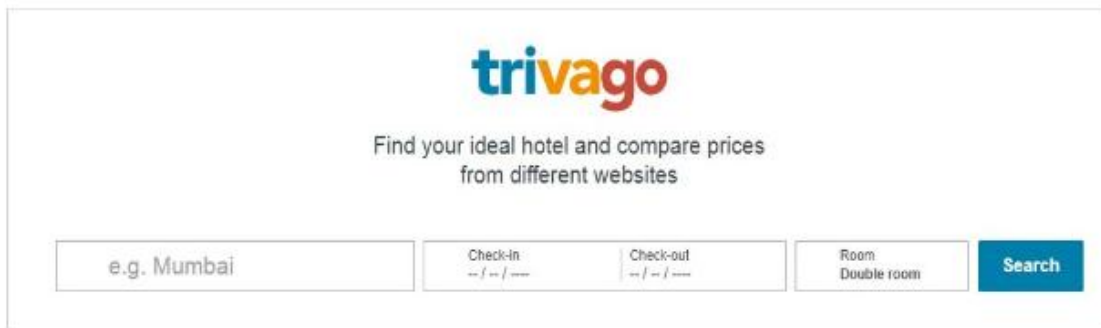
Also, Amazon leverages different payment gateway options to pay for the purchases.

It is not guaranteed that all the vendor-specific applications and the payment gateway applications are compatible with Amazon in terms of technology and platform where they get executed.

Even then, the communication between these applications and Amazon happens successfully.

Have you ever wondered how this becomes possible?

Note: All trademarks belong to their respective owners.

The image shows the Trivago website's search interface. At the top center is the Trivago logo in a colorful, lowercase font. Below the logo is the text "Find your ideal hotel and compare prices from different websites". Underneath this text are four input fields: the first is for the location, containing the text "e.g. Mumbai"; the second is for the check-in date, labeled "Check-in" with a calendar icon; the third is for the check-out date, labeled "Check-out" with a calendar icon; and the fourth is for the room type, labeled "Room" with the text "Double room" below it. To the right of these fields is a blue "Search" button.

Nowadays, our travel to different locations and stay are quite easy.

Travelling is made easier and comfortable through different forms of transportation.

Let us take the second aspect, stay, for our discussion. In order to make our stay comfortable and pleasant, we need to search for a good hotel that fits in our budget. We all know that different online hotel booking applications are there to make this search possible in a single click.

Now, let us talk about Trivago, a familiar online hotel booking site.

This application will give us the details of the hotels that are available in a particular locality based on our preference. As well, it will allow us to book rooms in a hotel for our stay.

Do you think, Trivago has access to the databases of the hotels to which it associates with?

The answer is a big NO. No one will allow an outsider to get access to their databases, simply because of the security reasons.

Or, do you think, these hotels push their room details to Trivago, periodically?

The answer is NO here as well, simply because when the updates happen periodically, there are situations where the end-customer gets misguided as Trivago and the partner hotels' databases are not in sync.

The correct mechanism behind this is, Trivago gives calls to the associated hotels' application services when the end customer requests.

And, calling those applications' services will be possible only when they are compatible with Trivago in terms of language and platform.

But, we cannot expect Trivago's compatibility with all its associated applications.

Then how does Trivago executes its calls successfully?

Note: All trademarks belong to their respective owners.



Quora, as we all know, is an ideal place for sharing and gaining knowledge.

Users of this application can post queries about any field and get quality replies. Also, people can step forward and share their knowledge in their fields of interest.

These activities need signing in with Quora.

Quora gives us many sign-in options like Google and Facebook apart from its standard authentication.

For example, if we have signed in to our Facebook already, it is not required for us to sign in to Quora individually. Rather, we can make use of an option called, “Continue with Facebook”. Here, it is implicitly understood that Quora avails the login service of Facebook. And, we see a lot of applications work this way in our day-to-day life.

So, it is quite natural to get a query here. Will Google and Facebook be compatible with Quora in terms of language and platform? Definitely not!!!!

But, how communication happens still?

The answer is Web Services.

Note: All trademarks belong to their respective owners.

Web Services and SOA

What is common that we see in all the three scenarios?

The answer is, there is a communication that happens between applications of different languages and platforms.

And, this is possible because the functionality that one application is trying to consume from the other is exposed as a Web Service.

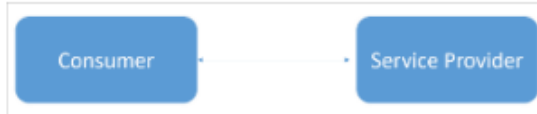
It is Web Services that makes the communication between incompatible applications, possible.

Before looking into Web Services in detail, let us take a quick look at SOA, the underlying principle behind Web Services.

SOA-Service Oriented Architecture

SOA expands to **Service Oriented Architecture**, is a software model designed for achieving communication among distributed application components, irrespective of the differences in terms of technology, platform, etc.,

The application component that requests for a service is the consumer and the one that renders the service is the producer.



SOA is an architecture that can help to build an enterprise application by availing the services/functionalities from different third-party entities without reinventing the existing functionalities.



SOA Principles

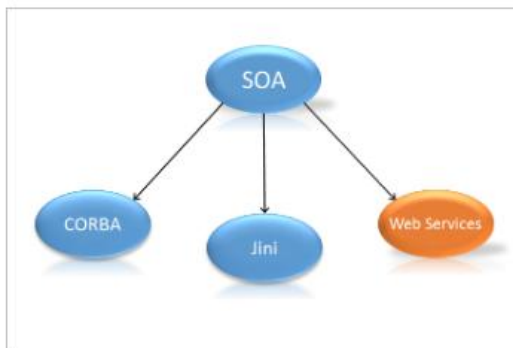
SOA that drives on high-value principles will take care of the following.

Principle	Explanation
Loose coupling	SOA aims at structuring procedures or software components as services. And, these services are designed in such a way that they are loosely coupled with the applications. So, these loosely-coupled services will come into picture only when needed.
Publishing the services	There should be a mechanism for publishing the services that include the functionality and input/output specifications of those services. In short, services are published in such a way that the developers can easily incorporate them into their applications.
Management	Finally, software components should be managed so that there will be a centralized control on the usage of the published services.

These principles are the driving factors behind SOA, ensuring a high level of abstraction, reliability, scalability, reusability, and maintainability.

SOA Implementation

Following is the pictorial representation that speaks about the different ways of realizing and implementing SOA.



CORBA: Common Object Request Broker Architecture is a standard that achieves interoperability among the distributed objects. Using CORBA, it is much possible to make the distributed application components communicate with each other irrespective of the location (the place where the components are available), language and platform.

Jini: Jini or Apache River is a way of designing distributed applications with the help of services that interact and cooperate.

Web Services: Web services are a way of making applications interact with each other on the web. Web services leverage the existing protocols and technologies but, uses certain standards.

Web Service is the most popular solution as it eliminates the issues related to interoperability with pre-defined standards and processes in place.

What are Web Services?

W3C defines Web Services as follows.

"A web service is a software system designed to support interoperable machine-to-machine interaction over a network".

- Web services are client-server applications that communicate (mostly) over Hyper Text Transfer Protocol (HTTP)
- Web services are a standard means of achieving interoperability between software applications running on diverse platforms and frameworks.
- Web Services are highly extensible. Services can be loosely coupled to solve complex operations.
- Web Service is a way of realizing Service Oriented Architecture (SOA). SOA is a style of software design where services are provided by the application components through a communication protocol. The basic principle of SOA makes it possible to be loosely-coupled with Vendor, product, and technology. As per SOA, a service is an individual unit of functionality that can be accessed remotely and updated independently, for example, retrieving a banking card statement online.

Comparitive study on Web Application and Web Services

So, how is a Web Application different from a Web Service?

Web Application	Web Services
A web application is meant for humans	Web Services are used to exchange data between two incompatible applications
The Web application is a complete application with GUI(Graphical User Interface)	Web Services do not necessarily have a user interface since it is used as a component in an end to end application
A typical web application, for example, a Java web application can be accessed by Java Clients (Web/Desktop) alone. And, a web application is generally accessed through browsers	Web Services can be accessed by applications of different languages and platforms
Reusing the functionalities of a web application is difficult	Reusing the functionalities that are exposed as web services is highly possible

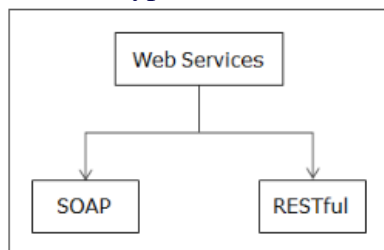
Benefits of using Web Service

Following are some of the common benefits of Web Services.

- Integrates with other systems easily irrespective of the difference in technology or platform - Interoperability
- Creates reusable components - Reusability
- Saves cost, effort and time - Ease of use

Types of Web Services

Web Services are of two types.

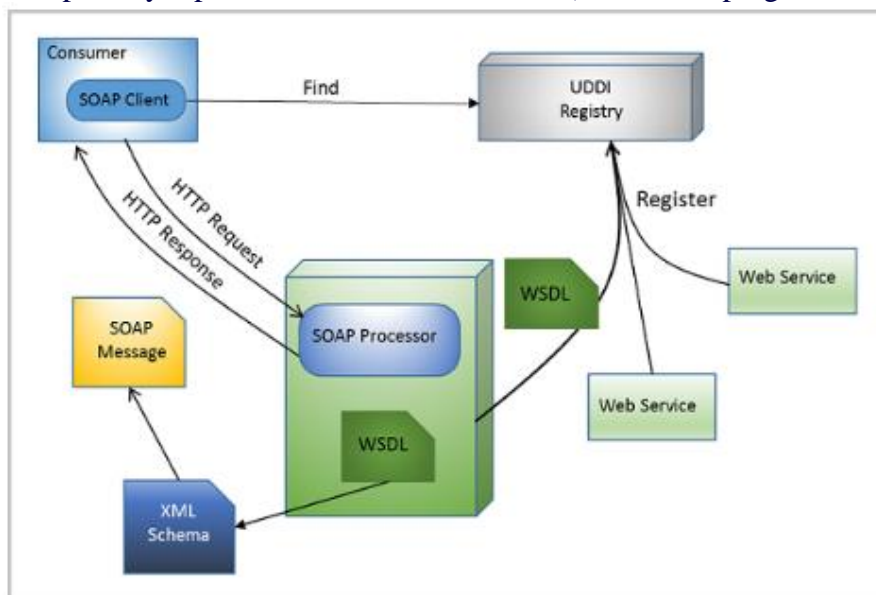


SOAP-based Web Services are **described, discovered and accessed** using the standards that are recommended by W3C.

RESTful Web Services are REST architecture based ones. As per REST architecture, everything is a resource. A resource (data/functionality/web page) is an object that has a specific type and associated data. Also, it has relationships to other resources and a well-defined set of methods that operate on it. RESTful web services are light-weight, scalable, maintainable and leverage the HTTP protocol to the maximum.

SOAP - Simple Object Access Protocol

SOAP (Simple Object Access Protocol) defines a very strongly typed messaging framework that relies heavily on XML and schemas. And, SOAP itself is a protocol (transported over HTTP, can also be carried over other transport layer protocol like SMTP/FTP, etc.) for developing SOAP-based APIs.



WSDL

Services are described using **WSDL (Web Services Description Language)**. And, this description is of the XML format.

UDDI

UDDI (Universal Description, Discovery, and Integration) is a registry service which is XML based, where Services can be discovered/registered.

SOAP

SOAP-based Web services are accessed using SOAP (Simple Object Access Protocol), an XML standard for defining the message format for information exchange.

SOAP based Web service - Sample

Here, we will

- Understand the basic concepts of SOAP
- Analyze the SOAP Requests and Responses

Let us look at a sample SOAP-based Web Service which when invoked by sending the name of the user will respond back with a string **"Hello <name>, Welcome to the world of Web Services"**.

This sample code is written in Java using **JAX-WS** API (API that helps to develop SOAP-based Web Services in Java)

GreetingService.java

```
1. /**
2.  A Sample SOAP based Web Service written in Java using JAX-WS API
3.  */
4.  package com.infy;
5.  import javax.jws.WebService;
6.  import javax.jws.WebMethod;
7.  import javax.jws.WebParam;
8.  //Below annotation is used to denote that this class is going to be exposed as a
9.  //Soap based Web Service and also names your Web Service while generation the WSDL
10. @WebService(serviceName = "GreetingService")
11. public class GreetingService {
12. /**
13. This is a sample web service operation called "hello"
14. */
15. @WebMethod(operationName = "hello")
16. public String myhello(@WebParam(name = "name") String uname) {
17. return "Hello " + uname + ",Welcome to the world of Web Services !";
18. }
19. }
```

Here is the WSDL file generated for GreetingService.java using WSDL tool.

```
1. <?xml version="1.0"?>
2. <definitions targetNamespace="http://infy.com/" name="GreetingService">
```



```
3. <types>
4. <schema>
5. <import namespace="http://infy.com/"
   schemaLocation="http://localhost:8080/GreetingService/GreetingService?xsd=1"/>
6. </schema>
7. </types>
8. <message name="hello">
9. <part name="parameters" element="tns:hello"/>
10. </message>
11. <message name="helloResponse">
12. <part name="parameters" element="tns:helloResponse"/>
13. </message>
14. <portType name="GreetingService">
15. <operation name="hello">
16. <input Action="http://infy.com/GreetingService/helloRequest" message="tns:hello"/>
17. <output Action="http://infy.com/GreetingService/helloResponse" message="tns:helloResponse"/>
18. </operation>
19. </portType>
20. <binding name="GreetingServicePortBinding" type="tns:GreetingService">
21. <binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
22. <operation name="hello">
23. <operation soapAction=""/>
24. <input>
25. <body use="literal"/>
26. </input>
27. <output>
28. <body use="literal"/>
29. </output>
30. </operation>
31. </binding>
32. <service name="GreetingService">
33. <port name="GreetingServicePort" binding="tns:GreetingServicePortBinding">
34. <address location="http://localhost:8080/GreetingService/GreetingService"/>
35. </port>
36. </service>
37. </definitions>
```

Sample SOAP Request for GreetingService:

```
1. <?xml version="1.0" encoding="UTF-8"?><S:Envelope
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
   ENV="http://schemas.xmlsoap.org/soap/envelope/">
2. <SOAP-ENV:Header/>
3. <S:Body>
```


4. <ns2:hello xmlns:ns2="http://infy.com/">
5. <name>Rekha</name>
6. </ns2:hello>
7. </S:Body>
8. </S:Envelope>

Sample SOAP Response for GreetingService:

1. <?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
2. <SOAP-ENV:Header/>
3. <S:Body>
4. <ns2:helloResponse xmlns:ns2="http://infy.com/">
5. <return>Hello Rekha,Welcome to the world of Web Services !</return>
6. </ns2:helloResponse>
7. </S:Body>
8. </S:Envelope>

RESTful Web Services

REST-REpresentational State Transfer is an architectural style for developing web services which uses HTTP for communication. The term, **REST** was coined by Roy Fielding in his doctoral dissertation in 2000.

In REST, everything is a **Resource** (data/functionality/web page) that is uniquely identified by URI (Uniform Resource Identifier).

Example:

Resource	URI
Data of an employee	http://www.infy.com/employees/john
Web Page	http://www.infy.com/about/infosys.html
Functionality that validates a credit card	http://www.infy.com/creditcards/1000001

The resources of REST can be **represented** in many ways that include JSON, XML, HTML, etc.,

For example, if the URI, http://www.infy.com/employees/john, is hit using HTTP GET method, the server may respond back with John's details in JSON format.

1. {
2. "name": "John",
3. "Level": 5,
4. "address": "Hyderabad",
5. "phoneno": 998765432
6. }

There are possibilities to expose John's details in XML format as well.

1. <employee>
2. <name>John</name>
3. <address>hyderabad</address>
4. <level>5</level>
5. <phoneno>998765432</phoneno>
6. </employee>

This means the server can represent the resource in many ways.

So now what does **State Transfer** mean?

When we make a request to a resource like <http://www.infy.com/employees/john> using HTTP, we may be asking for the current "state" of this resource or its desired "state".

We can use the same URI to indicate that, we want to:

- Retrieve the current value of an Employee John (current state)
- Perform Update operation on employee John's data (desired state) etc.

The server will understand which operation has to be invoked based on the HTTP method that was used to make a call.

In short it means that a client and server exchange, representation of a resource, which reflects its current state(GET) or its desired state(PUT).

We will understand more about REST and REST principles in the next set of sections.

Why RESTful Web Services?

We have two types of web services in general. But, why should we go for RESTful web services?

To understand this, let us look at the GreetingService example that we had seen while discussing SOAP-based Web Services. This is how the SOAP request looked like.

Sample SOAP Request to GreetingService:

1. Sample SOAP Request for GreetingService:
2. <?xml version="1.0" encoding="UTF-8"?>
3. <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
4. <SOAP-ENV:Header/>
5. <S:Body>
6. <ns2:hello xmlns:ns2="http://infy.com/">
7. <name>Rekha</name>
8. </ns2:hello>
9. </S:Body>
10. </S:Envelope>

This XML will be embedded as "payload" inside the **SOAP** request envelope which in turn will be wrapped using HTTP request and then, will be sent using HTTP POST.

The SOAP Response would be:

1. Sample SOAP Response for GreetingService:
2. `<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">`
3. `<SOAP-ENV:Header/>`
4. `<S:Body>`
5. `<ns2:helloResponse xmlns:ns2="http://infy.com/">`
6. `<return>Hello Rekha,Welcome to the world of Web Services !</return>`
7. `</ns2:helloResponse>`
8. `</S:Body>`
9. `</S:Envelope>`

If the same functionality is developed as a RESTful Web service, the request will be something like this,

1. GET `http://<server name>:<portno>/GreetingService/greetings?name=Rekha`

And, the response will be like the one that follows.

1. `{ "response": "Hello Rekha, Welcome to the world of Web Services" }`

SOAP vs REST

Both SOAP and REST provide support for building applications based on SOA.

Below is the table that summarizes the differences between SOAP and RESTful web services.

SOAP-based Web Service	RESTful Web Service
SOAP is a protocol (defines the way how messages have to be sent) which is transported over a transport layer Protocol, mostly HTTP.	REST is an architectural style.
SOAP messages can be transported using other transport layer protocols (SMTP, FTP) as well.	REST leverages HTTP Protocol.
Data (XML/JSON) wrapped in a SOAP message, comes with an additional overhead of XML processing at the client and server side as well.	REST is straight forward in handling requests and responses. Since REST supports multiple data formats other than XML, there is no overhead of wrapping the request and response data in XML. Also, REST uses the existing HTTP protocol for communication. All these factors together make REST, lightweight.
SOAP-based reads cannot be cached because SOAP messages are sent using the HTTP POST method.	REST-based reads can be cached.
A predefined formal contract is required between the consumer and the provider.	Formal contract is not mandatory. Service description can be done if needed, using a less verbose way with the help of WADL.

When REST?

To better understand REST and SOAP, let us take a simple analogy of mailing a letter.

- SOAP is like an envelope which is used to send/receive messages
- REST is like a postcard
 - Easier to handle (by the receiver)
 - Less usage of paper (i.e., consume less bandwidth)
 - Has a short content (REST requests are not really limited in length, especially, if POST is used instead of GET)

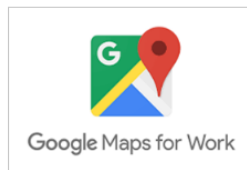
So eventually, if we want to develop an application which

- is light-weight
- can be accessed by any other application irrespective of the differences in terms of language and platform.
- leverages HTTP protocol
- is stateless
- does not require a specific message format to carry the request and response (like SOAP)
- represents data using various MIME types which all applications can consume

then, **RESTful** Web Service is the choice.

REST in Real World

The **Google Maps API** provides web services as an interface for requesting Maps API data from external services.



PayPal: Uses REST based payment services to accept online and mobile payments in an easier and secure way.



YouTube: In order to fetch the list of videos available on YouTube, A REST API service named "search" is provided by YouTube.

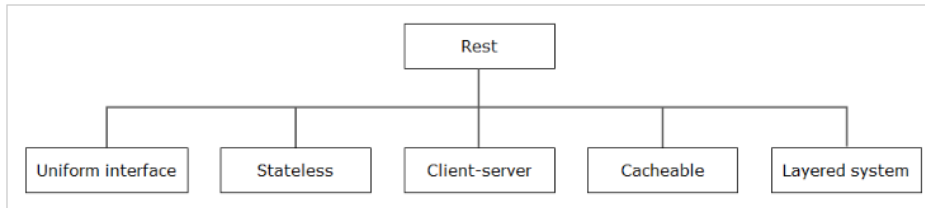


Note: All trademarks belong to the respective owners

REST Principles

RESTful web services are **fast**, **scalable** and **modifiable**.

RESTful web service or REpresentational State Transfer (REST) adheres to a set of architectural principles which makes RESTful web services fast, scalable and modifiable. Below is the list of principles that REST adheres to.



REST Principles

Let us understand REST principles in detail.

Uniform Interface

- In REST, data and functionality are considered as resources. Every resource has a representation (XML/JSON) and a URI to identify the same.
- Resources are manipulated using HTTP's GET, POST, PUT and DELETE methods.

We can use the HTTP Methods for the following operations on the resources.

HTTP Method	CRUD Operation	Description
GET	Read	Fetches a resource
POST	Create	Adds an existing resource to a server
PUT	Update	Transfers a resource to the server and overwrites the existing resource
DELETE	Delete	Discards resources

Example: If Customer is a resource, then Customer details can be added, deleted, updated or retrieved. If the blog is another resource, then the operations that we can perform on the blog will be read, create, update or delete.

Advantage: Single URI can perform multiple operations on the resource.

Stateless

- The interaction that takes place between the service provider and consumer is stateless.
- Being stateless, the server will never store the state information of its clients. Rather, every request from a service consumer should carry all the necessary information to make the server understand and process the request.

Advantage: Statelessness ensures that the requests from a consumer should get treated independently. This helps to scale the APIs (making them available in multiple servers) to support a large number of clients concurrently. Any server is free to serve any client since there is no session related dependency. Also, REST APIs stay less complex as server-side state synchronization logic is removed. On top, the performance of APIs is improvised because of less space occupation (no space is required for state-related info).

Client-Server

- Enforces the separation of concerns that helps to establish a distributed architecture. And, this is the most foundational constraint. Because of this distributed architecture with separation of concerns,

the change in one component will not affect other components. In simple words, the components stay loosely coupled making them more manageable.

Advantage: Supports the independent evolution of the client and server side logic.

Cacheable

- Service consumers can cache the response data and reuse the same. For example, HTTP GET and HEAD operations are cacheable.
- RESTful Web Services use the HTTP protocol as a carrier for the data. And, they can make use of the metadata that is available in the HTTP headers to enable caching. For example, Cache-Control headers are used to optimize caching to enhance performance.

Advantage: Enhances performance

Layered System

- REST based solution comprises of multiple architectural layers.
- The Consumer may interact directly with the actual provider or through the service that is residing in the middleware layer.

Advantage: Information hiding with a layered architecture helps to simplify the design of distributed architecture. Also, individual layers can be deployed and evolved independently.

Code-On-Demand

- This is an optional constraint which is primarily intended to allow client-side logic to be modified without doing changes in the server-side logic.
- For example, a service can be designed in such a way that it dynamically pushes part of the logic to the consumers.

Advantage: Logic that is present in the client end (such as Web browsers) can be modified/maintained independently without affecting the logic at the server side.

Ways to develop RESTful services

RESTful Web services can be developed and consumed in Java using **JAX-RS API** or **Spring REST**.



JAX-RS expands to Java API for RESTful Web services.

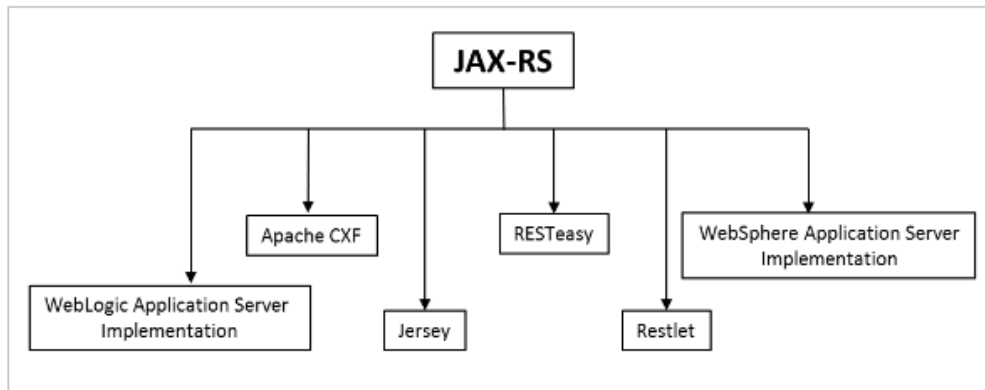
JAX-RS is a set of specifications to extend support for building and consuming RESTful Web services in Java.

JAX-RS 2.0 and above are annotation-driven which eases the development of Java-based RESTful services.

JAX-RS implementations

JAX-RS is simply a specification and we need actual implementations to write web services. There are many vendors in the market who adhere to the standards of JAX-RS such as, Jersey and RESTEasy.

Below are some of the JAX-RS implementations.



Spring REST

Spring is an end to end framework which has a lot of modules in a highly organized way.

One among such modules is, Spring MVC that provides the support for REST in addition to the standard MVC support.

The developers who use Spring MVC can go for constructing RESTful services in an effortless way. Also, it is guaranteed that the application stays lightweight as the build path is not polluted with many external dependencies, just to support REST.

Note: Spring REST is not an implementation of JAX-RS unlike Jersey and RESTEasy.