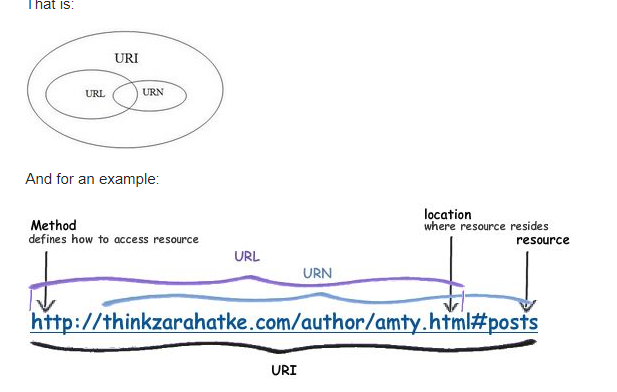
Top 40 Java Web Services Interview Questions And Answers

### Difference between URL and URI



|  |
| --- |
| **URL**  1. Abbreviation: URL stands for Uniform Resource Locator 2. Meaning: URL is a subset of URI that specifies where an identified resource is available and  the mechanism for retrieving it.Meachanism actually means one of the potocol schemes (e.g.,  http, ftp, file, mailto) provided by URI. 3. Examples: • http:/www.dotnetfunda.com/users/login.aspx Here, http is the protocol, dotnetfunda.com is the domain name, users is the folder name,  login.aspx is the filename • [http://dotnetfunda.com/articles/default.aspx](http://www.dotnetobject.com/expage.php?exurl=http://dotnetfunda.com/articles/default.aspx) Here http is the protocol, dotnetfunda.com is the domain name, articles is the folder name  and default.aspx is the file name   **URI**  1. Abbreviation: URI stands for Uniform Resource Identifier 2. Meaning: A URI is a superset of URL that identifies a resource either by location (URL),  or a name(URN), or both (URL and URN).  3.Examples: • [http://www.dotnetfunda.com](http://www.dotnetobject.com/expage.php?exurl=http://www.dotnetfunda.com) • /some/page.html  **Summary:**  1.A URI is either a URL or a URN. 2.Every URL is a URI. 3.Every URN is a URI. 4.A URN is never a URL. 5.A URL is never a URN. 6.Every URI is not a URL 7.If the URL describes both the location and name of a resource, the term to use is URI.  Since this is generally the case most of us encounter everyday, URI is the correct term.  Here, *URI* --> A Uniform Resource Identifier(URI) is used to identify something on the World Wide Web. *URN* --> Uniform Resource Name (URN), a type of URI, basically states what something is,  but do not have information on how to access it. *URL* --> Uniform Resource Locator (URL), a type of URI, contains the location of something  and tell the client program (usually a browser) how to access it. |

**Q1  What are web services ?**  
  
According to [oracle docs](http://docs.oracle.com/javaee/6/tutorial/doc/gijvh.html" \t "_blank), web services can be defined as

*Web services are client and server applications that communicate over the World Wide Web’s (WWW) HyperText Transfer Protocol (HTTP). Web services provide a standard means of inter operating between software applications running on a variety of platforms and frameworks.*

Main characteristics of the Web Services  are :  
  
1. Interoperability   
2. Extensibility  
3. Machine processable descriptions.  
  
for example in simple words , when we call somebody so the person dialing and calling is the client  application , while person receiving the call is server applicationand "hello" word is the protocol as similar to HTTP request .  
  
  
**Q2** **What is the difference between SOA and a web service?**

SOA (Service-Oriented Architecture) is an architectural pattern that makes possible for

services to interact with one another independently.

Web Services is a realization of SOA concept, that leverages XML, JSON, etc. and common Internet protocols such as HTTP(S), SMTP, etc.

SOA is a system-level architectural style that tries to expose business. WOA is an interface-level architectural style that focuses on the means by which these service capabilities are exposed to consumers.  
 **Q3 What is SOAP?**  
*SOAP* *(*Simple Object Access Protocol*)*is a transport protocol for sending and receiving requests and responses on XML format, which can be used on top of transport protocols such as HTTP, SMTP, UDP, etc.

**Q4** **What is REST?**

REST (REpresentational State Transfer) is an architectural style by which data can be transmitted over transport protocol such as HTTP(S).  
 **Q5  What is the difference between a REST web service and a SOAP web service?**

Below are the main differences between REST and SOAP web service

* REST supports different formats like text, JSON and XML; SOAP only supports XML;
* REST works only over HTTP(S) on a transport layer; SOAP can be used different protocols on a transport layer;
* REST works with resources, each unique URL is some representation of a resource; SOAP works with operations, which implement some business logic through different interfaces;
* SOAP based reads can’t be cached, for SOAP need to provide caching; REST based reads can be cached;
* SOAP supports SSL security and WS-security(Web Service-security); REST only supports SSL security;
* SOAP supports ACID (Atomicity, Consistency, Isolation, Durability); REST supports transactions, but it is neither ACID compliant nor can provide two phase commit.

**Q6 How  to decide which one of web service to use REST or SOAP?**  
  
“REST vs SOAP” we can rephrased to "Simplicity vs Standard". Of course, "Simplicity" with REST at most cases wins, it wins in performance, scalability and support for multiple data formats, but SOAP is favored where service requires comprehensive support for security (WS-security) and transactional safety (ACID).

**“SOAP”**  
 **Q7  What is WSDL?**

WSDL (Web Services Description Language) is an XML format for describing web services and how to access them.

**Q8  What is JAX-WS?**

JAX-WS (Java API for XML Web Services) is a set of APIs for creating web services in XML format.

**Q9 What is JAXB?**

JAXB (Java Architecture for XML Binding) is a Java standard that defines how Java objects are converted from and to XML. It makes reading and writing of XML via Java relatively easy.

**Q10 Can we send soap messages with attachments?**

Yes, we can send different formats such as PDF document, image or other binary file with soap messages as an attachment. Messages send using the binary data. SOAP messages is attached with MIME extensions that come in multipart/related.

An example:

MIME-Version: 1.0

Content-Type: Multipart/Related; boundary=MIME\_boundary; type=text/xml;

        start="<claim061400a.xml@ javahungry.com>"

Content-Description: This is the optional message description.

<?xml version='1.0' ?>

<SOAP-ENV:Envelope

xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Body>

..

<theSignedForm href="cid:claim061400a.tiff@javahungry.com"/>

..

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

--MIME\_boundary

Content-Type: image/tiff

Content-Transfer-Encoding: binary

Content-ID: <claim061400a.tiff@javahungry.com>

...binary TIFF image...

--MIME\_boundary—  
 **Q11 What is MTOM?**

MTOM (Message Transmission Optimization Mechanism) is a mechanism for transmitting large binary attachments with SOAP messages as raw bytes, allowing for smaller messages.

**Q12 What is XOP?**

XOP (XML-binary Optimized Packaging) is a mechanism defined for the serializationof XML Information Sets that contain binary data, as well as deserialization back into the XML Information Set.

**Q13 What is a SOAP envelope element?**

SOAP envelop element is the root element of a SOAP message which defines the XML document as a SOAP message.

An example:

<?xml version="1.0"?>  
<soap:Envelope  
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"  
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">  
  ...  
  Message information  
  ...  
</soap:Envelope>  
 **Q14 What does a SOAP namespace defines?**

SOAP *namespace*defines the Envelope as a *SOAP* Envelope.

An example:

xmlns:soap=http://www.w3.org/2001/12/soap-envelope

**Q15 What is the SOAP encoding?**

SOAP encoding is a method for structuring the request which is suggested within the SOAP specification, known as the SOAP serialization.

**Q16 What does SOAP encodingStyle attribute defines?**

SOAP encodingStyle defines the serialization rules used in a SOAP message. This attribute may appear on any element, and is scoped to that element's contents and all child elements not themselves containing such an attribute. There is no default encoding defined for a SOAP message.

An example:

SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding"

**Q17 What are 2 styles web service’s endpoint by using JAX-WS?**

* RPC (remote procedure call) style web service in JAX-WS;
* document style web service in JAX-WS.

**Q18 What is encoding rules for header entries?**

* a header entry is identified by its fully qualified element name, which consists of the namespace URI and the local name. All immediate child elements of the SOAP Header element must be namespace-qualified.
* the SOAP encodingStyle attribute may be used to indicate the encoding style used for the header entries.
* the SOAP mustUnderstand attribute and SOAP actor attribute may be used to indicate how to process the entry and by whom.

**Q19 What is the wsimport tool?** 

The wsimport tool is used to parse an existing Web Services Description Language (WSDL) file and generate required files (JAX-WS portable artifacts) for web service client to access the published web services: https://docs.oracle.com/javase/6/docs/technotes/tools/share/wsimport.html

**Q20 What is the wsgen tool?**

The wsgen tool is used to parse an existing web service implementation class and generates required files (JAX-WS portable artifacts) for web service deployment: http://docs.oracle.com/javase/6/docs/technotes/tools/share/wsgen.html

* What the tool are required to test SOAP services?

      SOAPUI tool for SOAP WS: http://www.soapui.org/

**Q21 What is the difference between SOAP and other remote access techniques?**

* SOAP is simple to use and it is non - symmetrical unlike DCOM or CORBA is highly popular and usually have complexity in them.
* SOAP provides greater platform independent with the language independence unlike DCOM or CORBA doesn't provide any of these.
* SOAP uses HTTP as its transport protocol and the data are being saved in XML format that can be ready by human, whereas DCOM or CORBA have their own binary formats that are used to transport the data in complicated manner.   
  SOAP identify the object other than URL endpoint. SOAP objects are stateless and it is hard to maintain that. Whereas, it is not hard to maintain in case of other remote access techniques.

**“REST”**

**Q22 What is a resource in a REST?**

      A resource is a unique URL with representation of an object which we can get contents via GET and modify via PUT, POST, DELETE.

**Q23 What are HTTP methods supported by REST?**

* GET;
* POST;
* PUT;
* DELETE;
* OPTIONS;
* HEAD.

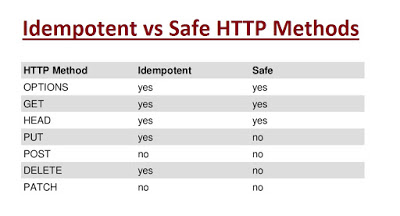
For example, to create a new Order you should use:  
  
POST /orders  
  
and to update an existing order, you should use  
  
PUT /orders/13892  
  
which means modify the order with OrderId 13892  
  
If you execute POST request multiple times, it will end up create that many orders, but when you execute PUT it will always produce the same result because of its [idempotent](http://javarevisited.blogspot.com/2016/05/what-are-idempotent-and-safe-methods-of-HTTP-and-REST.html). You should also remember that both PUT and POST are **unsafe methods**. Safe methods in HTTP do not modify the resource in the server e..g GET or HEAD, while Idempotent HTTP methods return same result irrespective of how many times you call them.

### Summary

Here is nice overview of which HTTP methods are safe and Idempotent:

* GET is both Safe and Idempotent.
* HEAD is also both safe and idempotent.
* OPTIONS is also safe and idempotent.
* PUT is not safe but idempotent.
* DELETE is not safe but idempotent.
* POST is neither safe nor idempotent.
* PATCH is also neither safe nor idempotent.

Here is a slide which explains which methods of HTTP protocol are safe and which are Idempotent

[](https://2.bp.blogspot.com/-6JFwUOCTM7w/V02b8UY9y_I/AAAAAAAAGE0/IP-CEIcWZ34deazdz7StE71gBHXkv9dEwCLcB/s1600/Safe+and+Idempotent+methods+of+HTTP+and+REST.jpg)

|  |
| --- |
| Why POST is not idempotent.  In the real world its quite likely that your POST request may timeout, what will happen to the resource that. Is the resource actually updated? Does the timeout happen during sending the request to the server, or the response to the client? Can we safely retry again, or do we need to figure out first what has happened with the resource? By using idempotent methods like PUT, you don't have to answer this question, but we can safely resend the request until we actually get a response back from the server. |

**Q24 Whether can use GET request instead of POST to create a resource?**

It is not possibly, because **GET can’t change a resource.**

**Q25 What is the difference between PUT and POST?**

Need to use PUT when can update a resource completely through a specific resource. For example, if know that an article resides at http://javahungry.blogspot.com/article/123, can PUT a new resource representation of this article through a PUT on this URL. If do not know the actual resource location for instance, when add a new article, can use POST.

PUT is idempotent, while POST is not. It means if use PUT an object twice, it has no effect.

**Q26 What is WADL?**

WADL (Web Application Description Language) is a XML description of a deployed RESTful web application.

|  |
| --- |
| Example: The Storage-Service WADL This is the Web Application Description Language file (the WADL) that is generated for the storage-service sample application:  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>  <application xmlns="http://research.sun.com/wadl/2006/10">  <doc xmlns:jersey="http://jersey.dev.java.net/"  jersey:generatedBy="Jersey: 1.0-ea-SNAPSHOT 10/02/2008 12:17 PM"/>  <resources base="http://localhost:9998/storage/">  <resource path="/containers">  <method name="GET" id="getContainers">  <response>  <representation mediaType="application/xml"/>  </response>  </method>  <resource path="{container}">  <param xmlns:xs="http://www.w3.org/2001/XMLSchema"  type="xs:string" style="template" name="container"/>  <method name="PUT" id="putContainer">  <response>  <representation mediaType="application/xml"/>  </response>  </method>  <method name="DELETE" id="deleteContainer"/>  <method name="GET" id="getContainer">  <request>  <param xmlns:xs="http://www.w3.org/2001/XMLSchema"  type="xs:string" style="query" name="search"/>  </request>  <response>  <representation mediaType="application/xml"/>  </response>  </method>  <resource path="{item: .+}">  <param xmlns:xs="http://www.w3.org/2001/XMLSchema"  type="xs:string" style="template" name="item"/>  <method name="PUT" id="putItem">  <request>  <representation mediaType="\*/\*"/>  </request>  <response>  <representation mediaType="\*/\*"/>  </response>  </method>  <method name="DELETE" id="deleteItem"/>  <method name="GET" id="getItem">  <response>  <representation mediaType="\*/\*"/>  </response>  </method>  </resource>  </resource>  </resource>  </resources>  </application> |

**Q27 What are frameworks available to implement REST web services?**

Jersey, Restlet, EasyRest, etc.

**Q28 What is the Restlet framework?**

Restlet is a lightweight, comprehensive, open source RESTful web API framework for the Java platform.

It has advantages such as

* websocket and server-sent events support;
* HTTP/2 support;
* transparent HTTP PATCH support;
* client cache service;
* fluent APIs.

http://restlet.com/

**Q29 What is the Jersey framework?**

Jersey is open source framework for developing RESTful Web Services in Java that provides support for JAX-RS APIs and serves as a JAX-RS (JSR 311 & JSR 339) Reference Implementation. It has advantages such as

* contains support for Web Application Description Language (WADL);
* contains Jersey Test Framework which lets run and test Jersey REST services inside JUnit;
* supports for the REST MVC pattern, which would allow to return a View from Jersey services rather than just data.

https://jersey.java.net/

**Q30 What is the RESTeasy framework?**

RESTeasy is a JBoss project, which implements of the JAX-RS specification. It has benefits such as

* fully certified JAX-RS implementation; supports HTTP 1.1 caching semantics including cache revalidation;
* JAXB marshalling into XML, JSON, Jackson, Fastinfoset, and Atom as well as wrappers for maps, arrays, lists, and sets of JAXB Objects;
* OAuth2 and Distributed SSO with JBoss AS7;
* rich set of providers for: XML, JSON, YAML, Fastinfoset, Multipart, XOP, Atom, etc.

http://resteasy.jboss.org/

**Q31 What is the difference between AJAX and REST?**

* in Ajax, the request are sent to the server by using XMLHttpRequest objects; REST have a URL structure and a request/response pattern the revolve around the use of resources;
* Ajax eliminates the interaction between the customer and server asynchronously; REST requires the interaction between the customer and server;
* Ajax is a set of technology; REST is a type of software architecture and a method for users to request data or information from servers.

**Q32 What tool are required to test REST services?**

Firefox “poster” plugin for RESTFUL services. https://addons.mozilla.org/en-us/firefox/addon/poster/

[](http://1.bp.blogspot.com/-l_4Sb1tEk_c/VbSfpcirsbI/AAAAAAAAAo4/zqLjasVLwWw/s1600/java+web+services+interview+questions+and+answers.jpg)

**Q33 What does a @Path annotation do?**

     @Path annotation binds URI pattern to a Java method.

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

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

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

**@Path**("/persons")

public **class** **PersonRestService** {

**@GET**

public Response getPerson() {

**return** Response.status(**200**).entity("getPerson is called").build();

}

**@GET**

**@Path**("/vip")

public Response getPersonVIP() {

**return** Response.status(**200**).entity("getPersonVIP is called").build();

}

}

On calling URI: “/persons” result:  getPerson is called

On calling URI: “/persons/vip” result:  getPersonVIP is called

**Q34  What does a @PathParam do?**

     @PathParam annotation injects the value of URI parameter that defined in @Path expression.

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

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

**import** **javax.ws.rs.PathParam**;

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

**@Path**("/persons")

public **class** **PersonRestService** {

**@GET**

**@Path**("{id}")

public Response getPersonById(**@PathParam**("id") String id) {

**return** Response.status(**200**).entity("getPersonById is called, id : " + id).build();

}

}

On calling URI: “/persons/1” result:  getPersonById is called, id : 1

**Q35  What does a @QueryParam do?**

    @QueryParam annotation injects URI query parameter into Java method.

**import** **java.util.List**;

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

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

**import** **javax.ws.rs.QueryParam**;

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

**@Path**("/persons")

public **class** **PersonService** {

**@GET**

**@Path**("/query")

public Response getPersons(

**@QueryParam**("from") int from,

**@QueryParam**("to") int to,

**@QueryParam**("orderBy") List&lt;String&gt; orderBy) {

**return** Response

.status(**200**)

.entity("getPersons is called, from : " + **from** + ", to : " + to

+ ", orderBy" + orderBy.toString()).build();

}

}

On calling URI: “/persons/query?from=10&to=20&orderBy=age&orderBy=name” result: getPersons is called, from : 10, to : 20, orderBy**[**age, name**]**

**Q36  What does a @MatrixParam do?**

@MatrixParam are a set of **“name=value”** in URI path.

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

**import** **javax.ws.rs.MatrixParam**;

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

**import** **javax.ws.rs.PathParam**;

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

**@Path**("/books")

public **class** **BookService** {

**@GET**

**@Path**("{year}")

public Response getBooks(**@PathParam**("year") String year,

**@MatrixParam**("author") String author,

**@MatrixParam**("country") String country) {

**return** Response

.status(**200**)

.entity("getBooks is called, year : " + year

+ ", author : " + author + ", country : " + country)

.build();

}

}

On calling URI: “/books/2015” result: getBooks is called, year : 2015, author : null, country : null

On calling URI: “/books/2015;author= doyle;country=scotland” result: getBooks is called, year : 2015, author : doyle, country : scotland

**Q37  What does a @FormParam do?**

@FormParam bind HTML form parameters value to a Java method.

**import** **javax.ws.rs.FormParam**;

**import** **javax.ws.rs.POST**;

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

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

**@Path**("/persons")

public **class** **PersonService** {

**@POST**

**@Path**("/add")

public Response addPerson(

**@FormParam**("name") String name,

**@FormParam**("age") int age) {

**return** Response.status(**200**)

.entity("addPerson is called, name : " + name + ", age : " + age)

.build();

}

}

HTML form:

<html>

<body>

<form action="/persons/add" method="post">

<p>

Name : <input type="text" name="name" />

</p>

<p>

Age : <input type="text" name="age" />

</p>

<input type="submit" value="Add Person" />

</form>

</body>

</html>

**Q39  How to get HTTP request header in JAX-RS (2 ways)?**

* inject directly with @HeaderParam;

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

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

**import** **javax.ws.rs.HeaderParam**;

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

**@Path**("/persons")

public **class** **PersonService** {

**@GET**

**@Path**("/get")

public Response getPerson(

**@HeaderParam**("person-agent") String personAgent) {

**return** Response.status(**200**)

.entity("getPerson is called, personAgent : " + personAgent)

.build();

}

}

On calling URI: “/persons/get” result: getPerson is called, personAgent : Mozilla**/**5.0 **(**Windows NT 6.1; rv:5.0**)** Gecko**/**20100101 Firefox**/**5.0

* pragmatically via @Context.

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

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

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

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

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

**@Path**("/persons")

public **class** **PersonService** {

**@GET**

**@Path**("/get")

public Response getPerson(**@Context** HttpHeaders headers) {

String personAgent = headers.getRequestHeader("person-agent").get(**0**);

**return** Response.status(**200**)

.entity("getPerson is called, personAgent : " + personAgent)

.build();

}

}

On calling URI: “/persons/get” result: getPerson is called, personAgent : Mozilla**/**5.0 **(**Windows NT 6.1; rv:5.0**)** Gecko**/**20100101 Firefox**/**5.0

**Q40  How to download file in JAX-RS?**

* put @Produces(“?”) on service method, with a Response return type. Instead “?” write a type text/plain, image/png, etc.
* set “Content-Disposition” in Response header to tell browser pop up a download box for user to download.

**import** **java.io.File**;

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

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

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

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

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

**@Path**("/image")

public **class** **ImageService** {

private static final String FILE\_PATH = "c:**\\**my.png";

**@GET**

**@Path**("/get")

**@Produces**("image/png")

public Response getFile() {

File file = new File(FILE\_PATH);

ResponseBuilder response = Response.ok((Object) file);

response.header("Content-Disposition",

"attachment; filename=image\_from\_server.png");

**return** response.build();

}

}

Some questions may not be applicable to eveyone. For example if you have used anything else than CXF or RESTLET for building REST web services then the question number 15 may not hold good for you

If you are unable to get answer to any of the question, do drop a comment, I will be more than happy to get back with the answer...

1) What is a web service?

2) Can I use GET request instead of PUT to create resources?

3) What is the difference between a RESTful web service (on HTTP) and a HTTP Servlet both of which perform the same function?

4) How will you migrate from SOAP web service to RESTfull web service?

5) What is the difference between HTTP POST and PUT requests?

6) What all kind of output formats can one generate using RESTful web service?

7) What all tools have you used to write RESTful web service?

8) Can my web browser be a client to RESTful web service? If yes then how will you generate DELETE request from web browser?

9) What is meant by JAX-WS and JAX-RS?

10) How is JAXB related to RESTful web services?

11) How will you handle synchronization issues when multiple clients try to consume web service simultaneously?

12) Can you describe any one RESTful web service you have written?

13) What will you do when an error code has to be returned to the client

OR

How will you handle application error scenarios in RESTful web service

14) Is it mandatory to use HTTP protocol for performing CRUD operations by a REST based web service?

15) What are the pros and cons of using the RESTLET framework than CXF for writing RESTful web service?

16) Can one RESTful web service be a client to another RESTful web service for performing business operations? Describe with example?

17) Can you generate HTML output from a RESTful web service? How?

18) Can a RESTful web service generate output in various formats based on some parameter received from the client.

19) With a RESTful web service, whose state is getting transferred and how?

20) Who owns the specification for REST web services and how does it get updated to newer versions?

==================================================================

**Use HTTP status codes**

The HTTP standard provides over 70 status codes to describe the return values. We don’t need them all, but  there should be used at least a mount of 10.

200 – OK – Eyerything is working  
201 – OK – New resource has been created  
204 – OK – The resource was successfully deleted

304 – Not Modified – The client can use cached data

400 – Bad Request – The request was invalid or cannot be served. The exact error should be explained in the error payload. E.g. „The JSON is not valid“  
401 – Unauthorized – The request requires an user authentication  
403 – Forbidden – The server understood the request, but is refusing it or the access is not allowed.  
404 – Not found – There is no resource behind the URI.  
422 – Unprocessable Entity – Should be used if the server cannot process the enitity, e.g. if an image cannot be formatted or mandatory fields are missing in the payload.

500 – Internal Server Error – API developers should avoid this error. If an error occurs in the global catch blog, the stracktrace should be logged and not returned as response.

# Top 20 RESTful Web Services Interview Question and Answers

**The most frequently asked RESTful Web services interview questions:**

Web services, a very well known term when we talk about exchanging some sort of data between multiple applications or say software. Based on the client-server model, these services can be used by multiple software application written in various languages and also it has an advantage of running on various platforms.

Likewise REST, Representational State Transfer is also based on stateless client-server style architecture, which can be easily accessed over the network and is identified by URIs i.e. Uniform Resource Identifier.

The main aim of describing the definition of Web services as well as REST above is to help you relate to the term ‘RESTful web services’ because RESTful web services are defined as web services which use HTTP method and is based on the architecture of REST. It has useful features like high scalability and maintainability, the creation of APIs, etc.



In this article, you will find the collection of question and answer which will clear your basics and help develop the better understanding of the subject.

**Also read:**

* **[Top Web Services Interview Questions](http://www.softwaretestinghelp.com/web-services-interview-questions/)**
* **[Top SoapUI interview questions](http://www.softwaretestinghelp.com/soapui-interview-questions-and-answers/)**

## RESTful web services Question and Answers

Let’s start.

**Q #1) What is your understanding of what are RESTful web services?**

Just like SOAP (Simple Object Access Protocol), which is used to develop web services by XML method, RESTful web services use web protocol i.e. HTTP protocol method. They have the feature like scalability, maintainability, help multiple application communication built on various programming languages etc.

RESTful web service implementation defines the method of accessing various resources which are required by the client and he has sent the request to the server through the web browser. The important aspects of this implementation include:

* Resources
* Request Headers
* Request Body
* Response Body
* Status codes

**Q #2) Name the protocol which is used by RESTful web services.**

RESTful web services use a famous web protocol i.e. HTTP protocol. This serves as a medium of data communication between client and server. HTTP standard methods are used to access resources in RESTful web service architecture.

**Q #3) Explain the term ‘Addressing’ with respect to RESTful WEB service.**

Just like we require address with postal code to reach any person, in the same way, ‘Addressing’ locates resources that are present on the server for the purpose of hosting web services. This is usually done with URI i.e. Unified Resource Identifier.

**Q #4) Enlist features of RESTful web services.**

Every RESTful web services should have following features and characteristics that are enlisted below:

* Based on the Client Server representation.
* Use of HTTP protocol for performing functions like fetching data from the web service, retrieving resources, execution of any query, etc.
* The communication between the server and client is performed through the medium known as ‘messaging’.
* Addressing of resources available on the server through URIs.
* Based on the concept of statelessness where every client request and the response is independent of the other with complete assurance of providing required information.
* Uses the concept of caching.
* Works on Uniform interface.

**Q #5) Explain messaging technique.**

Messages are the mode of exchanging data for any type of communication to take place. In the same way, HTTP protocol plays the role of message communication between the client and server through HTTP Request and Response methods. HTTP request is sent by the client who contains information about the data and in turn, receives HTTP Response from the server.

Messages are the collection of information about the data i.e. Metadata.

**Q #6) What are the core components of HTTP request and HTTP response?**

The core components that come under HTTP Request are:

* **Verb:** Includes methods like GET, PUT, POST, etc.
* Uniform Resource Identifier for identifying the resources available on the server.
* HTTP Version for specifying the HTTP version.
* HTTP Request header for containing the information about the data.
* HTTP Request body that contains the representation of the resources in use.

**The core components that come under HTTP Response are:**

* **Request Code:** This contains various codes which determine the status of the server response.
* HTTP Version for specifying the HTTP version.
* HTTP Response header for containing the information about the data.
* HTTP Response body that contains the representation of the resources in use.

**Q #7) Explain the term ‘Statelessness’ with respect to RESTful WEB service.**

In REST, ST itself defines State Transfer and Statelessness means complete isolation. This means, the state of the client’s application is never stored on the server and is passed on. In this process, the clients send all the information that is required for the server to fulfill the HTTP request that has been sent. Thus every client request and the response is independent of the other with complete assurance of providing required information.

Every client passes a ‘session identifier’ which also acts as an identifier for each session.

**Q #8) Enlist advantages and disadvantages of ‘Statelessness’.**

In the above question, we have understood the meaning of statelessness with respect to the client-server communication. Now, let us see some of its advantages and disadvantages.

**Advantages:**

* Every method required for communication is identified as an independent method i.e. there are no dependencies to other methods.
* Any previous communication with the client and server is not maintained and thus the whole process is very much simplified.
* If any information or metadata used earlier in required in another method, then the client sends again that information with HTTP request.
* HTTP protocol and REST web service, both shares the feature of statelessness.

**Disadvantages:**

* In every HTTP request from the client, the availability of some information regarding the client state is required by the web service.

**Q #9) Enlist some important constraints for RESTful web services.**

Every constraint has positive as well as negative impacts and to produce an overall architecture, there should be the balance between both of them. Below mentioned are some important constraints for RESTful web service:

* There should be separate concerns for each server and client which will help to maintain the modularity within the application. This will also reduce the complexity and increase the scalability.
* The client-server communication should be stateless, which means no previous information is used and the complete execution is done in isolation. In cases of failure, it also helps the client to recover.
* In client-server communication, the HTTP response should be cacheable so that when required cached copy can be used which in turn enhances the scalability and performance of the server.
* The fourth constraint is the uniform interface which allows client-server interaction to be easily understood. This constraint is further divided into four sub-constraints as:
  + Resource Identification
  + Resource manipulation
  + Each message is easily understood and is self-descriptive.
  + Hypermedia, which is defined as the text with hyperlinks and when clicked it moves to another application state.
* Client-server communication should be done on a layered system and thus the client should only have knowledge about the intermediate level with which communication is being done,

**Q #10) What is a ‘Resource’?**

Just like the ‘Object’ instance, we have learned in object orient programming Language, in the same way, ‘Resource’ is defined as an object of a type which can be an image, HTML file, text data, and any type of dynamic data. There are varieties of representation formats available in order to represent a resource.

**Some most common are enlisted below:**

* JSON
* YAML
* XML
* HTML

**Q #11) Why proper representation of Resource is required?**

Representation is very important because it determines the easy identification of resources. With proper representations of resource in the proper format, allows the client to easily understand the format.

**Q #12) Enlist some important points that should be kept in mind while designing Resources representation for RESTful web services.**

As there are no restrictions on the format in which the resource representation is done but just that the main requirement is the format of the representation should be as per the client requirement. A good resource representation is designed by considering the following main points:

* The resource representation format should be easily understood by the client and server.
* The representation should be complete regardless of its format structure, which may be complex or simple.
* In the case of the link of the resources to other resources, such cases should also be considered and handled.

**Q #13) What is Caching?**

Caching is the process in which server response is stored so that a cached copy can be used when required and there is no need of generating the same response again. This process not only reduces the server load but in turn increase the scalability and performance of the server. Only the client is able to cache the response and that too for a limited period of time.

Mentioned below are the header of the resources and their brief description so that they can be identified for the caching process:

* Time and Date of resource creation
* Time and date of resource modification that usually stores the last detail.
* Cache control header
* Time and date at which the cached resource will expire.
* The age which determines the time from when the resource has been fetched.

**Q #14) Explain Cache-control header.**

A standard Cache control header can help in attaining cache ability. Enlisted below is the brief description of various cache control header:

* **Public:** Resources that are marked as the public can be cached by any intermediate components between the client and server.
* **Private:** Resources that are marked as private can only be cached by the client.
* No cache means that particular resource cannot be cached and thus the whole process is stopped.

**Q #16) What is Payload?**

The request data which is present in the body part of every HTTP message is referred as ‘Payload’.  In Restful web service, the payload can only be passed to the recipient through POST method.

There is no limit of sending data as payload through POST method but the only concern is that more data with consuming more time and bandwidth. This may consume much of user’s time also.

**Q #17) Enlist some of the HTTP methods with description.**

Mentioned below is the list of HTTP methods with their descriptions:

* **GET:** This is a read only operation which fetches the list of users on the server.
* **PUT:** This operation is used for the creation of any new resource on the server.
* **POST:** This operation is used for updating an old resource or for creating a new resource.
* **DELETE:** As the name suggests, this operation is used for deleting any resource on the server.
* **OPTIONS:** This operation fetches the list of any supported options of resources that are available on the server.

**Q #18) What is the difference between PUT method and POST method?**

The major difference between the PUT and POST method is that the result generated with PUT method is always same no matter how many times the operation is performed. On the other hand, the result generated by POST operation is always different every time.

**Q #19) What is your understanding about JAX-RS?**

JAX-RS is defined as the Java API for RESTful web service. Among multiple libraries and framework, this is considered as the most suitable Java programming language based API which supports RESTful web service.

**Some of the implementations of JAX-RS are:**

* Jersey
* RESTEasy
* Apache CFX
* Play

Among these, Jersey is the most popular framework.

[**Jersey**](https://jersey.java.net/)  
Jersey RESTful Web Services framework is open source, production quality, a framework for developing RESTful Web Services in Java that provides support for JAX-RS APIs and serves as a JAX-RS (JSR 311 & JSR 339) reference Implementation and initially provided by Sun Microsystem.  
  
  
Jersey framework is more than the JAX-RS Reference Implementation. Jersey provides its own API that extends the JAX-RS toolkit with additional features and utilities to further simplify RESTful service and client development. Jersey also exposes numerous extension SPIs so that developers may extend Jersey to best suit their needs.  
  
Jersey also has some of the best tooling (IDE) support especially if you are using Netbeans. So you can achieve better productivity from the tooling perspective. There are some challenges with Jersey-Spring integration, especially with AOP.  
  
The latest release of Jersey is 2.22.2  
  
[**Apache CXF**](http://cxf.apache.org/)

The CXF is another free and open source web service framework and a JAX-RS implementation from Apache. CXF helps you to build and develop services using frontend programming APIs like JAX-RS and JAX-WS. These services can speak a variety of protocols such as SOAP, XML/HTTP, RESTful HTTP, or CORBA and work over a variety of transports such as HTTP, JMS, and JBI.  
  
One of the key difference between Apache CXF and Jersy's JAX-RS implementation is that it is implemented as a CXF filter sitting behind the servlets, while Jersey and RestEasy are, servlet filters.  
  
One more advantage of using ApacheCXF is that it makes it easy to produce both a JAX-RS and JAX-WS (SOAP) endpoint from the exact same data model and service interface at the same time. So if that is something which matters to you, this may be the way to go. CXF had issues with handling SSL and HTTP proxies which seem to have been addressed in recent releases.  You can also read [Developing Web Services with Apache CXF and Axis2](http://aax-us-east.amazon-adsystem.com/x/c/Qo7Qefc73O0K720heHPn9kIAAAFfc3g_HwEAAAFKAavjfC0/https:/assoc-redirect.amazon.com/g/r/https:/www.amazon.com/Developing-Web-Services-Apache-Axis2/dp/0557254329/ref=as_at?creativeASIN=0557254329&linkCode=w61&imprToken=sFv0W.SryfYKmYTj3VGcEg&slotNum=2&tag=javamysqlanta-20) to learn more about developing RESTful web services using Apache CXF framework in Java.  
  
Read more: <http://javarevisited.blogspot.com/2017/02/difference-between-jax-rs-restlet-jersey-apache-cfx-RESTEasy.html#ixzz4x6kdq9tS>

**Q #20) What are HTTP status codes? Enlist few with meaning.**

HTTP status codes basically are the representation of the status of the task that has been performed on the server, with the mode of some codes. Every code has their own meaning.

**Some of the HTTP status codes with their meaning are as follows:**

* **Code 200:** This indicates success.
* **Code 201:** This indicates resource has been successfully created.
* **Code 204:** This indicates that there is no content in the response body.
* **Code 404:** This indicates that there is no method available.

**Q #15) What are the best practices that are to be followed while designing RESTful web services?**

To design a secure RESTful web service, there are some best practices or say points that should be considered. These are explained as follows:

* Every input on the server should be validated.
* Input should be well formed.
* Never pass any sensitive data through URL.
* For any session, the user should be authenticated.
* Only HTTP error messages should be used for indicating any fault.
* Use message format that is easily understood and is required by the client.
* Unified Resource Identifier should be descriptive and easily understood.

## Why is Scalability Important?

Finally, it’s important to contextualize exactly why scalability important. It sounds good in analogy form, but is this actually a significant enough issue to demand best practices? Well, consider this data supplied by HighScalability.com:

* Netflix has 100 million subscribers;
* 25% of US citizens won’t subscribe to traditional cable, favoring streaming solutions;
* The largest Google Computer Engine job utilized 220,000 high-throughput cores;
* Sling TV has 1.3 million subscribers; and
* There are 105 data centers worldwide needed for cloud computing.

Why is this data important? All of these data points are from services that started out small, and grew **exponentially larger**. Netflix started as a relatively small service, and grew extremely fast. Google started essentially as a small experiment, and became one of the largest organizations in the world. Sling TV had a small subscriber base, and has considerably and consistently grown.

That the smallest, single-purpose solution is not going to stay that way forever. Designing for the traffic and function that you currently handle is fine, but intrinsically limits the future growth of the platform.

How to scalable Service.

There are two fundamental ways we can scale software: 'Vertically' or 'horizontally'.

**Vertical Scaling** addresses the scalability of a single instance of the service. A simple way to scale most software is simply to run it on a more powerful machine; one with a faster processor or more memory.

**Horizontal Scaling.** Here we run multiple instances of the application rather than focussing on the performance of a single instance. This has the advantage of being linearly scalable; rather than buying a bigger, more expensive box, we just buy more copies of the same cheap box. With the right architectural design, this approach can scale massively. Indeed it's the approach taken by almost all of largest internet scale companies: Facebook, Google, Twitter etc.. Horizontal Scaling also introduces redundancy; the loss of a single node need not impact the system as a whole. For these reasons, horizontal scaling is the preferred approach to building scalable, redundant systems.

So, the fundamental approach to building scalable systems is to compose them of horizontally scaled services. In order to do this we need to follow a few basic principles:

* **Stateless.** Any services that stores state across an interaction with another service is hard to scale. For example, a web service that stores in-memory session state between requests requires a sophisticated session-aware load balancer. A stateless service, by contrast, only requires simple round-robin load balancing. For a web application (or service) you should avoid using session state or any static or application level variables.
* **Coarse Grained API.** To be stateless, a service should expose an API that exposes operations as a single interaction. A chatty API, where one sets up some data, asks for some transition, and then reads off some results, implies statefulness by its design. The service would need to identify a session and then maintain information about that session between successive calls. Instead a single call, or message, to the service should encapsulate all the information that the service requires to complete the operation.
* **Idempotent.** Much scalable infrastructure is a trade-off between competing constraints. Delivery guarantees are one of these. For various reasons it's is far simpler to guarantee 'at least once' delivery than 'exactly once'. If you can make your software tolerant of multiple deliveries of the same message it will be easier to scale.
* **Embrace Failure.** Arrays of services are redundant if the system as a whole can survive the loss of a single node. You should design your services and infrastructure to expect and survive failure. Consider implementing a [Chaos Monkey](http://www.codinghorror.com/blog/2011/04/working-with-the-chaos-monkey.html) that randomly kills processes. If you start by expecting your services to fail, you'll be prepared when they inevitably do.
* **Avoid instance specific configuration.** A scalable service should be designed in such a way that it doesn't need to know about other instances of itself, or have to identify itself as a specific instance. I shouldn't need to have to configure one instance any differently than another. This would include communication mechanisms that require messages to be addressed to a specific instance of the service, or some non-convention based way that the service was required to identify itself. Instead we should rely on infrastructure (load-balancers, pub-sub messaging etc.) to manage the communication between arrays of services.
* **Simple automated deployment.** Have a service that can scale is no advantage if we can't deploy it when we are close to capacity. A scalable system must have automated processes to deploy new instances of services as the need arises.
* **Monitoring.** We need to know when services are close to capacity so that we can add additional service instances. Monitoring is usually an infrastructure concern; we should be monitoring CPU, network, and memory usage and have alerts in place to warn us when these pass certain trigger points. Sometimes it's worth introducing application specific alerts when some internal trigger is reached, such as the number of items in an in-memory queue, for example.
* **KISS - Keep It Small and Simple.** This is good advice for any software project, but is especially pertinent to building scalable resilient systems. Large monolithic codebases are hard to reason about, hard to monitor, and hard to scale. Building your system out of many small pieces makes it easy to address those pieces independently. Design your system so that each service has only one purpose and is decoupled from the operations of other services. Have your services communicate using non-proprietary open standards to avoid vendor lock-in and allow for a heterogeneous platform. JSON over HTTP, for example, is an excellent choice for intra-service communication. Every platform has HTTP and JSON libraries and there is abundant off-the-shelf infrastructure (proxies, load-balancers, caches) that can be used to help your system scale.

## Why JSON is Better Than XML

XML is much more difficult to parse than JSON.  
JSON is parsed into a ready-to-use JavaScript object.

For AJAX applications, JSON is faster and easier than XML:

Using XML

* Fetch an XML document
* Use the XML DOM to loop through the document
* Extract values and store in variables

Using JSON

* Fetch a JSON string
* JSON.Parse the JSON string

## JSON is Unlike XML Because

* JSON doesn't use end tag
* JSON is shorter
* JSON is quicker to read and write
* JSON can use arrays

The biggest difference is:

 XML has to be parsed with an XML parser. JSON can be parsed by a standard JavaScript function.

|  |
| --- |
| REST follows one-to-one mapping between create, read, update, and delete (CRUD) operations and HTTP methods.   * To create a resource on the server, use POST. * To retrieve a resource, use GET. * To change the state of a resource or to update it, use PUT. * To remove or delete a resource, use DELETE. |

### @GET

Annotate your Get request methods with @GET.

|  |  |
| --- | --- |
| 1  2  3  4 | @GET  public String getHTML() {    ...  } |

### @Produces

@Produces annotation specifies the type of output this method (or web service) will produce.

|  |  |
| --- | --- |
| 1  2  3  4  5 | @GET  @Produces("application/xml")  public Contact getXML() {    ...  } |
| 1  2  3  4  5 | @GET  @Produces("application/json")  public Contact getJSON() {    ...  } |

### @Path

@Path annotation specify the URL path on which this method will be invoked.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @GET  @Produces("application/xml")  @Path("xml/{firstName}")  public Contact getXML() {    ...  } |

### @PathParam

We can bind REST-style URL parameters to method arguments using @PathParam annotation as shown below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | @GET  @Produces("application/xml")  @Path("xml/{firstName}")  public Contact getXML(@PathParam("firstName") String firstName) {    Contact contact = contactService.findByFirstName(firstName);    return contact;  } |
| 1  2  3  4  5  6  7 | @GET  @Produces("application/json")  @Path("json/{firstName}")  public Contact getJSON(@PathParam("firstName") String firstName) {    Contact contact = contactService.findByFirstName(firstName);    return contact;  } |

### @QueryParam

Request parameters in query string can be accessed using @QueryParam annotation as shown below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | @GET  @Produces("application/json")  @Path("json/companyList")  public CompanyList getJSON(@QueryParam("start") int start, @QueryParam("limit") int limit) {    CompanyList list = new CompanyList(companyService.listCompanies(start, limit));    return list;  } |

The example above returns a list of companies (with server side pagination) which can be displayed with rich clients implemented using Ext-js or jQuery. You can read more more about setting up [ExtJS grid panel with remote sorting and pagination using Hibernate](http://blog.techferry.com/2012/01/25/extjs-grid-panel-with-remote-sorting-and-pagination-using-hibernate/).

### @POST

Annotate POST request methods with @POST.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @POST  @Consumes("application/json")  @Produces("application/json")  public RestResponse<Contact> create(Contact contact) {  ...  } |

### @Consumes

The @Consumes annotation is used to specify the MIME media types a REST resource can consume.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | @PUT  @Consumes("application/json")  @Produces("application/json")  @Path("{contactId}")  public RestResponse<Contact> update(Contact contact) {  ...  } |

### @FormParam

The REST resources will usually consume XML/JSON for the complete Entity Bean. Sometimes, you may want to read parameters sent in POST requests directly and you can do that using @FormParam annotation. GET Request query parameters can be accessed using [@QueryParam](http://www.techferry.com/articles/RESTful-web-services-JAX-RS-annotations.html#QueryParam) annotation.

|  |  |
| --- | --- |
| 1  2  3  4  5 | @POST  public String save(@FormParam("firstName") String firstName,      @FormParam("lastName") String lastName) {        ...    } |

### @PUT

Annotate PUT request methods with @PUT.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | @PUT  @Consumes("application/json")  @Produces("application/json")  @Path("{contactId}")  public RestResponse<Contact> update(Contact contact) {  ...  } |

### @DELETE

Annotate DELETE request methods with @DELETE.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @DELETE  @Produces("application/json")  @Path("{contactId}")  public RestResponse<Contact> delete(@PathParam("contactId") int contactId) {  ...  } |