**RESTFUL**

**What is JAX-RS API?**

**Java API for RESTful Web Services (JAX-RS) is the Java API for creating REST web services. JAX-RS uses annotations to simplify the development and deployment of web services. JAX-RS is part of JDK, so you don’t need to include anything to use it’s annotations.**

**Name some implementations of JAX-RS API?**

**There are two major implementations of JAX-RS API.**

**Jersey: Jersey is the reference implementation provided by Sun. For using Jersey as our JAX-RS implementation, all we need to configure its servlet in web.xml and add required dependencies. Note that JAX-RS API is part of JDK not Jersey, so we have to add its dependency jars in our application.**

**RESTEasy:** RESTEasy **is the JBoss project that provides JAX-RS implementation.**

**Jersey**

**Jersey API for creating Restful web services and invoking the web service using Java client program and testing web service using tool.**

**Dependencies:**

**<dependencies>**

**<dependency>**

**<groupId>com.sun.jersey</groupId>**

**<artifactId>jersey-server</artifactId>**

**<version>1.14</version>**

**</dependency>**

**<dependency>**

**<groupId>com.sun.jersey</groupId>**

**<artifactId>jersey-servlet**</**artifactId>**

**<version>1.14</version>**

**</dependency>**

**<dependency>**

**<groupId>com.sun.jersey</groupId>**

**<artifactId>jersey-client</artifactId>**

**<version>1.14</version>**

**</dependency>**

**</dependencies>**

**web.xml:**

**Deployment descriptor for the web service. So any request with URI “http://:/jd/rest/\*” will be processed by Jersey ServletContainer servlet. Important thing to note is the init-param value passed for “com.sun.jersey.config.property.packages”. This defines in which package jersey will look for the web service classes. This property must point to your resources classes. It also looks for the resource classes into the sub-packages.**

<?xml version="1.0" encoding="UTF-8"?>

<web-app id="WebApp\_ID" version="2.5"

xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd">

<display-name>My Jersey Project</display-name>

<!-- Jersey Servlet configurations -->

<servlet>

<servlet-name>Jersey REST Service</servlet-name>

<servlet-class>**com.sun.jersey.spi.container.servlet.ServletContainer**</servlet-class>

<init-param>

<param-name>**com.sun.jersey.config.property.packages**</param-name>

<param-value>com.jd</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>Jersey REST Service</servlet-name>

<url-pattern>/rest/\*</url-pattern>

</servlet-mapping>

<!-- Jersey Servlet configurations -->

</web-app>

**RestTemplate**

**The** RestTemplate **is the core class for client-side access to RESTful services. It is conceptually similar to other template classes in Spring, such as JdbcTemplate and JmsTemplate and other template classes found in other Spring portfolio projects. RestTemplate’s behavior is customized by providing callback methods and configuring the `HttpMessageConverter used to marshal objects into the HTTP request body and to unmarshal any response back into an object. As it is common to use XML as a message format, Spring provides a** MarshallingHttpMessageConverter **that uses the Object-to-XML framework that is part of the org.springframework.**oxm **package. This gives you a wide range of choices of XML to Object mapping technologies to choose from.**

**Create AppConfig.java file under /src folder. Give appropriate package name to your file. We are using @EnableWebMvc, @ComponentScan and @Configuration annotations. These will bootstrap the spring mvc application and set package to scan controllers and resources.**

**@ComponentScan (basePackages = {"com.javarticles.spring.annotations.packageA",**

**"com.javarticles.spring.annotations.packageB" })**

**What is REST?**

**In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol. REST isn't protocol specific, but when people talk about REST they usually mean REST over HTTP.**

What are Restful web services?

In the web services terms, Representational State Transfer (REST) is a stateless client-server architecture in which the web services are viewed as resources and can be identified by their URIs. Web services client uses that URI to access the resource.  
It consists of two components REST server which provides access to the resources and a REST client which accesses and modify the REST resources.

SOAP vs REST Web Services

There are many differences between SOAP and REST web services. The important differences between SOAP and REST are given below:

|  |  |
| --- | --- |
| SOAP | REST |
| SOAP is a standard protocol for creating web services. | REST is an architectural style to create web services. |
| SOAP is acronym for Simple Object Access Protocol. | REST is acronym for REpresentational State Transfer. |
| SOAP uses WSDL to expose supported methods and technical details. | REST exposes methods through URIs, there are no technical details. |
| SOAP web services and client programs are bind with WSDL contract | REST doesn’t have any contract defined between server and client |
| SOAP web services and client are tightly coupled with contract. | REST web services are loosely coupled. |
| SOAP learning curve is hard, requires us to learn about WSDL generation, client stubs creation etc. | REST learning curve is simple, POJO classes can be generated easily and works on simple HTTP methods. |
| SOAP supports XML data format only | REST supports any data type such as XML, JSON, image etc. |
| SOAP web services are hard to maintain, any change in WSDL contract requires us to create client stubs again and then make changes to client code. | REST web services are easy to maintain when compared to SOAP, a new method can be added without any change at client side for existing resources. |
| SOAP web services can be tested through programs or software such as Soap UI. | REST can be easily tested through CURL command, Browsers and extensions such as Chrome Postman. |

|  |  |  |
| --- | --- | --- |
| No. | SOAP | REST |
| 1) | SOAP is a protocol. | REST is an architectural style. |
| 2) | SOAP stands for Simple Object Access Protocol. | REST stands for Representational State Transfer. |
| 3) | SOAP can't use REST because it is a protocol. | REST can use SOAP web services because it is a concept and can use any protocol like HTTP, SOAP. |
| 4) | SOAP uses services interfaces to expose the business logic. | REST uses URI to expose business logic. |
| 5) | JAX-WS is the java API for SOAP web services. | JAX-RS is the java API for RESTful web services. |
| 6) | SOAP defines standards to be strictly followed. | REST does not define too much standards like SOAP. |
| 7) | SOAP requires more bandwidth and resource than REST. | REST requires less bandwidth and resource than SOAP. |
| 8) | SOAP defines its own security. | Restful web services inherit security measures from the underlying transport. |
| 9) | SOAP permits XML data format only. | REST permits different data format such as Plain text, HTML, XML, JSON etc. |
| 10) | SOAP is less preferred than REST. | REST more preferred than SOAP. |
| 11. | JavaScript can call SOAP, but it is difficult to implement | Easy to call from JavaScript |
| 12 | Performance is not great compared to REST. | Performance is much better compared to SOAP less CPU intensive, leaner code etc. |
| 13. | Transfer is over HTTP, also uses other protocols such as SMTP, FTP etc. | Transfer is over HTTP only. |

**What Are Some of the Annotations Defined by JAX-RS?**

Here is a listing of some of the Java programming annotations that are defined by JAX-RS, with a brief description of how each is used.

Table 3–1 Summary of Jersey Annotations

| Annotation | Description |
| --- | --- |
| @Path | The @**Path** annotation's value is a relative URI path indicating where the Java class will be hosted, for example,/helloworld. You can also embed variables in the URIs to make a URI path template. For example, you could ask for the name of a user, and pass it to the application as a variable in the URI, like this, /helloworld/{username}. |
| @GET | The @**GET** annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP GET requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| @POST | The @**POST** annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP POST requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| @PUT | The @**PUT** annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP PUT requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| @DELETE | The @**DELETE** annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP DELETE requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| @HEAD | The @**HEAD** annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP HEAD requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| @PathParam | The @**PathParam** annotation is a type of parameter that you can extract for use in your resource class. URI path parameters are extracted from the request URI, and the parameter names correspond to the URI path template variable names specified in the @Path class-level annotation. |
| @QueryParam | The @QueryParam annotation is a type of parameter that you can extract for use in your resource class. Query parameters are extracted from the request URI query parameters. |
| @Consumes | The @Consumes annotation is used to specify the MIME media types of representations a resource can consume that were sent by the client. |
| @Produces | The @Produces annotation is used to specify the MIME media types of representations a resource can produce and send back to the client, for example, "text/plain". |
| @Provider | The @Provider annotation is used for anything that is of interest to the JAX-RS runtime, such as **MessageBodyReader** and **MessageBodyWriter**. For HTTP requests, the MessageBodyReader is used to map an HTTP request entity body to method parameters. On the response side, a return value is mapped to an HTTP response entity body using a **MessageBodyWriter**. If the application needs to supply additional metadata, such as HTTP headers or a different status code, a method can return a Response that wraps the entity, and which can be built using Response.**ResponseBuilder**. |
| @MatrixParam | Matrix parameters are a set of “**name=value**” in URI path. The matrix parameter is “**author=mkyong**“, separate by a semi colon “**;**“. |
| @HeaderParam | extracts information from the HTTP headers. |
| @CookieParam | extracts information from the cookies declared in cookie related HTTP headers. |
| @FormParam | to bind HTML form parameters value to a Java method |

**Requirements of a JAX-WS Endpoint?**

JAX-WS endpoints must follow these requirements:

The implementing class must be annotated with either the **javax.jws.WebService or javax.jws.WebServiceProvider** annotation.

The implementing class may explicitly reference an SEI through the endpoint Interface element of the @**WebService** annotation, but is not required to do so. If no endpoint Interface is not specified in @WebService, an SEI is implicitly defined for the implementing class.

The business methods of the implementing class must be public, and must not be declared static or final. Business methods that are exposed to web service clients must be annotated with **javax.jws.WebMethod**. Business methods that are exposed to web service clients must have JAX-B-compatible parameters and return types. See [Default Data Type Bindings](https://docs.oracle.com/cd/E17802_01/webservices/webservices/docs/2.0/tutorial/doc/JAXBWorks4.html#wp82947).

The implementing class must not be declared final and must not be abstract.

The implementing class must have a default public constructor.

The implementing class must not define the finalize method.

The implementing class may use the javax**.annotation.PostConstruct** or **javax.annotation.PreDestroy** annotations on its methods for lifecycle event callbacks.

The **@PostConstruct** method is called by the container before the implementing class begins responding to web service clients.

The **@PreDestroy** method is called by the container before the endpoint is removed from operation.

# RESTful Web Services Security Guide

Knowledge of securing RESTful Web Services is as much important as to write them. Mostly RESTful APIs are HTTP protocol based, you any user having internet connection can connect to it, and so hackers as well. It’s very important to write secure APIs to protect the business. But before starting to secure RESTful APIs, let’s understand what are our options as developers? What will be good fit for our usecase?

## 4 Ways to Secure RESTful Web Services

There are multiple ways to secure a RESTful API in Java. Let’s go through 4 most popular choices:

#### BASIC Authentication

It’s simplest of all techniques and probably most used as well. You use login/password forms – it’s basic authentication only. You input your username and password and submit the form to server, and application identify you as a user – you are allowed to use the system – else you get error.

The main problem with this security implementation is that credentials are propagated in a plain way from the client to the server. Credentials are merely encoded with Base64 in transit, but not encrypted or hashed in any way. This way, any sniffer could read the sent packages over the network.

HTTPS is, therefore, typically preferred over or used in conjunction with Basic Authentication which makes the conversation with the web server entirely encrypted. The best part is that nobody can even guess from the outside that Basic Auth is taking place.

<dependency>

<groupId>org.glassfish.jersey.core</groupId>

<artifactId>jersey-server</artifactId>

<version>2.13</version>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.containers</groupId>

<!-- if your container implements Servlet API older than 3.0, use "jersey-container-servlet-core" -->

<artifactId>jersey-container-servlet</artifactId>

<version>2.13</version>

</dependency>

<!-- https://mvnrepository.com/artifact/javax.ws.rs/javax.ws.rs-api -->

<dependency>

<groupId>javax.ws.rs</groupId>

<artifactId>javax.ws.rs-api</artifactId>

<version>2.0</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

</dependency>

**Web.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"* id=*"WebApp\_ID"* version=*"3.0"*>

<display-name>RESTful Service Authentication</display-name>

<welcome-file-list>

<welcome-file>index.htm</welcome-file>

</welcome-file-list>

<servlet>

<servlet-name>RESTful Jersey Hello World Service</servlet-name>

<servlet-class>**org.glassfish.jersey.servlet.ServletContainer**</servlet-class>

<init-param>

<param-name>**jersey.config.server.provider.packages**</param-name>

<param-value>com.javapapers.webservices.rest.jersey</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>RESTful Jersey Hello World Service</servlet-name>

<url-pattern>/rest/\*</url-pattern>

</servlet-mapping>

<filter>

<filter-name>AuthenticationFilter</filter-name>

<filter-class>**com.javapapers.webservices.rest.jersey.RestAuthenticationFilter**</filter-class>

</filter>

<filter-mapping>

<filter-name>AuthenticationFilter</filter-name>

<url-pattern>/rest/\*</url-pattern>

</filter-mapping>

</web-app>

#### DIGEST Authentication

This authentication method makes use of a hashing algorithms to encrypt the password (called **password hash**) entered by the user before sending it to the server. This, obviously, makes it much safer than the basic authentication method, in which the user’s password travels in plain text that can be easily read by whoever intercepts it.

**Read More:** [Generate Encrypted Passwords](http://howtodoinjava.com/security/how-to-generate-secure-password-hash-md5-sha-pbkdf2-bcrypt-examples/)

There are many such hashing algorithms in java also, which can prove really effective for password security such as MD5, SHA, BCrypt, SCrypt and PBKDF2WithHmacSHA1 algorithms.

Please remember that once this password hash is generated and stored in database, you cannot convert it back to original password. Each time user login into application, you have to regenerate password hash again, and match with hash stored in database. So, if user forgot his/her password, you will have to send him a temporary password and ask him to change it with his new password. Well, it’s common trend now-a-days.

#### CLIENT CERT Authentication

This is a mechanism in which a trust agreement is established between the server and the client through certificates. They must be signed by an agency established to ensure that the certificate presented for authentication is legitimate, which is known as CA.

Using this technique, when the client attempts to access a protected resource, instead of providing a username or password, it presents the certificate to the server. The certificate contains the user information for authentication including security credentials, besides a unique private-public key pair. The server then determines if the user is legitimate through the CA. Additionally, it must verify whether the user has access to the resource. This mechanism must use HTTPS as the communication protocol as we don’t have a secure channel to prevent anyone from stealing the client’s identity.

#### Using API Keys

If you have ever developed applications which interact other with other applications over cloud e.g. facebook integration or twitter authentication etc. then you have already used this. They require you to provide API key and API secret to rightly identify you. These API key and secret are some random encoded string which is impossible to guess.

To understand how it works, let’s assume you are using a Flickr (photo sharing application) and want to post some of your photos using it’s REST API. You build the request as documented in Flickr docs, then send it.

Then, when receiving the request, Flickr authenticates the user by reading the information from the API key with the secret key that belongs to the user. Once these validations are successful, the server delivers the response to the client. Thus, we obtain a response with all the photos that have been recently posted within Flickr.

As you’ll notice, this way, you can easily create applications using the provider’s API. Also, the provider will allow you to authenticate, access public information.

If someone starts disrespecting agreements e.g. sending junk traffic or any policy violation, the provider withdraws the API key and prevents the abusive use of its APIs.

**RESTful Web Services Security Implementations**

Apart from above concepts, you will usually need to secure your RESTful APIs in your company using below methods.

Using SecurityContext

The javax.ws.rs.core.SecurityContext interface provides access to security-related information for a request and is very similar to javax.servlet.http.HttpServletRequest.

You access the SecurityContext by injecting an instance into a class field, setter method, or method parameter using the javax.ws.rs.core.Context annotation e.g. in below code sc.isUserInRole() is used to check authorization for user.

|  |
| --- |
| @GET  @Produces("text/plain;charset=UTF-8")  @Path("/hello")  public String sayHello(@Context SecurityContext sc) {          if (sc.isUserInRole("admin"))              return "Hello World!";          throw new SecurityException("User is unauthorized.");  } |

Method level Authorization Using Annotations

This technique is widely used in enterprise application and used to verify roles and responsibities of an authenticated used – for any certain operation. JAX-RS provides below annotations for this purpose.

[@PermitAll](http://docs.oracle.com/javaee/6/api/javax/annotation/security/PermitAll.html)

[@DenyAll](http://docs.oracle.com/javaee/6/api/javax/annotation/security/DenyAll.html)

[@RolesAllowed](http://docs.oracle.com/javaee/6/api/javax/annotation/security/RolesAllowed.html)

An example use of annotation can be:

|  |
| --- |
| @RolesAllowed("ADMIN")  @PUT  @Path("/users/{id}")  public Response updateUserById(@PathParam("id") int id)  {      //Update the User resource      UserDatabase.updateUser(id);      return Response.status(200).build();  } |

Best Practices

Let’s note down some important points while designing security for your RESTful web services.

Use only HTTPS protocol so that your whole communication is always encrypted.

Never send auth credentials or API keys as query param. They appear in URL and can be logged or tracked easily.

Use hardest encryption level always. It will help in having more confidence.

For resources exposed by RESTful web services, it’s important to make sure any PUT, POST, and DELETE request is protected from Cross Site Request Forgery.

Always validate the input data asap it is received in server method. Use only primitive data as input parameter as much as possible.

Rely on framework provided validation features as they are tested by large community already.

Let me know your thoughts and experiences on securing RESTful web services in your organization.

**Q1  What are web services ?**  
  
According to [oracle docs](http://docs.oracle.com/javaee/6/tutorial/doc/gijvh.html), 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.

**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.

**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/

**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();

}

}

## Building Request AuthenticationFilter

you know that JAX-RS 2.0 has filters for pre and post request handling, so we will be using ContainerRequestFilterinterface. In this filter, we will get details of the method which request is trying to access. We will find-out all security related configuration on that method, and verify everything here in this filter e.g. annotation like @PermitAll, @DenyAll or @RolesAllowed.

According to annotation applied on methods, we will make the decision to pass or block the request.

|  |
| --- |
| package com.howtodoinjava.jersey.provider;    import java.lang.reflect.Method;  import java.util.Arrays;  import java.util.HashSet;  import java.util.List;  import java.util.Set;  import java.util.StringTokenizer;    import javax.annotation.security.DenyAll;  import javax.annotation.security.PermitAll;  import javax.annotation.security.RolesAllowed;  import javax.ws.rs.container.ContainerRequestContext;  import javax.ws.rs.container.ResourceInfo;  import javax.ws.rs.core.Context;  import javax.ws.rs.core.MultivaluedMap;  import javax.ws.rs.core.Response;  import javax.ws.rs.ext.Provider;    import org.glassfish.jersey.internal.util.Base64;    /\*\*   \* This filter verify the access permissions for a user   \* based on username and passowrd provided in request   \* \*/  @Provider  public class AuthenticationFilter implements **javax.ws.rs.container.ContainerRequestFilter**  {        @Context  **private ResourceInfo resourceInfo;**        private static final String AUTHORIZATION\_PROPERTY = "Authorization";      private static final String AUTHENTICATION\_SCHEME = "Basic";      private static final Response ACCESS\_DENIED = Response.status(Response.Status.UNAUTHORIZED)                                                          .entity("You cannot access this resource").build();      private static final Response ACCESS\_FORBIDDEN = Response.status(Response.Status.FORBIDDEN)                                                          .entity("Access blocked for all users !!").build();        @Override      public void filter(ContainerRequestContext requestContext)      {          Method method = resourceInfo.getResourceMethod();          //Access allowed for all          if( ! method.isAnnotationPresent(PermitAll.class))          {              //Access denied for all              if(method.isAnnotationPresent(DenyAll.class))              {                  requestContext.abortWith(ACCESS\_FORBIDDEN);                  return;              }                //Get request headers              final MultivaluedMap<String, String> headers = requestContext.getHeaders();                //Fetch authorization header              final List<String> authorization = headers.get(AUTHORIZATION\_PROPERTY);                //If no authorization information present; block access              if(authorization == null || authorization.isEmpty())              {                  requestContext.abortWith(ACCESS\_DENIED);                  return;              }                //Get encoded username and password              final String encodedUserPassword = authorization.get(0).replaceFirst(AUTHENTICATION\_SCHEME + " ", "");                //Decode username and password              String usernameAndPassword = new String(Base64.decode(encodedUserPassword.getBytes()));;                //Split username and password tokens              final StringTokenizer tokenizer = new StringTokenizer(usernameAndPassword, ":");              final String username = tokenizer.nextToken();              final String password = tokenizer.nextToken();                //Verifying Username and password              System.out.println(username);              System.out.println(password);                //Verify user access              if(method.isAnnotationPresent(RolesAllowed.class))              {                  RolesAllowed rolesAnnotation = method.getAnnotation(RolesAllowed.class);                  Set<String> rolesSet = new HashSet<String>(Arrays.asList(rolesAnnotation.value()));                    //Is user valid?                  if( ! isUserAllowed(username, password, rolesSet))                  {                      requestContext.abortWith(ACCESS\_DENIED);                      return;                  }              }          }      }      private boolean isUserAllowed(final String username, final String password, final Set<String> rolesSet)      {          boolean isAllowed = false;            //Step 1. Fetch password from database and match with password in argument          //If both match then get the defined role for user from database and continue; else return isAllowed [false]          //Access the database and do this part yourself          //String userRole = userMgr.getUserRole(username);            if(username.equals("howtodoinjava") && password.equals("password"))          {              String userRole = "ADMIN";                //Step 2. Verify user role              if(rolesSet.contains(userRole))              {                  isAllowed = true;              }          }          return isAllowed;      }  } |

## Register AuthenticationFilter with ResourceConfig

Now you will need to register above filter with ResourceConfig instance. So create an instance like below:

|  |
| --- |
| package com.howtodoinjava.jersey;    import org.glassfish.jersey.filter.LoggingFilter;  import org.glassfish.jersey.server.ResourceConfig;    import com.howtodoinjava.jersey.provider.AuthenticationFilter;  import com.howtodoinjava.jersey.provider.GsonMessageBodyHandler;    public class CustomApplication extends ResourceConfig  {      public CustomApplication()      {          packages("com.howtodoinjava.jersey");          register(LoggingFilter.class);          register(GsonMessageBodyHandler.class);            //Register Auth Filter here          register(AuthenticationFilter.class);      }  } |

And add this resource config in web.xml file.

|  |
| --- |
| <!DOCTYPE web-app PUBLIC   "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"   "<http://java.sun.com/dtd/web-app_2_3.dtd>" >    <web-app>        <display-name>Archetype Created Web Application</display-name>        <servlet>          <servlet-name>jersey-serlvet</servlet-name>          <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>          <init-param>              <param-name>javax.ws.rs.Application</param-name>              <param-value>com.howtodoinjava.jersey.CustomApplication</param-value>          </init-param>          <load-on-startup>1</load-on-startup>      </servlet>        <servlet-mapping>          <servlet-name>jersey-serlvet</servlet-name>          <url-pattern>/rest/\*</url-pattern>      </servlet-mapping>    </web-app> |

## Secure REST APIs

Now it’s time to secure the REST APIs. Use standard JAX-RS annotations for that like below.

|  |
| --- |
| @Path("/employees")  public class JerseyService  {      @RolesAllowed("ADMIN")      @GET      @Produces(MediaType.APPLICATION\_JSON)      @Consumes(MediaType.APPLICATION\_JSON)      public Employees getAllEmployees()      {          Employees list = new Employees();          list.setEmployeeList(new ArrayList<Employee>());            list.getEmployeeList().add(new Employee(1, "Lokesh Gupta"));          list.getEmployeeList().add(new Employee(2, "Alex Kolenchiskey"));          list.getEmployeeList().add(new Employee(3, "David Kameron"));            return list;      }  } |

**What is the use of @XmlRootElement annotation?**

XmlRootElement annotation is used by JAXB to transform java object to XML and vice versa. So we have to annotate model classes with this annotation.

**How to set different status code in HTTP response?**

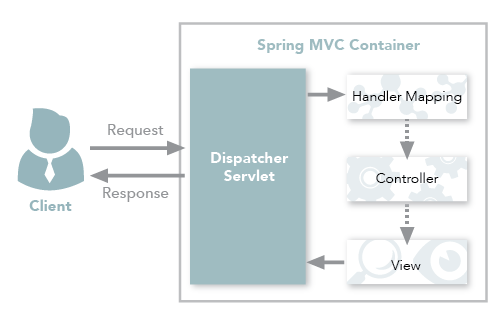
For setting HTTP status code other than 200, we have to use javax.ws.rs.core.Response class for response. Below are some of the sample return statements showing it’s usage.

return Response.status(422).entity(exception).build();

return Response.ok(response).build(); //200

## Spring MVC Framework and REST

Spring’s annotation based MVC framework simplifies the process of creating RESTful web services. The key difference between a traditional Spring MVC controller and the RESTful web service controller is the way the HTTP response body is created. While the traditional MVC controller relies on the View technology, the RESTful web service controller simply returns the object and the object data is written directly to the HTTP response as JSON/XML.  For a detailed description of creating RESTful web services using the Spring framework, click [here](http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/mvc.html).

  
Figure 1: Spring MVC traditional workflow

### Spring MVC REST Workflow

The following steps describe a typical Spring MVC REST workflow:

1. The client sends a request to a web service in URI form.
2. The request is intercepted by the DispatcherServlet which looks for Handler Mappings and its type.  
   • The Handler Mappings section defined in the application context file tells DispatcherServlet which strategy to use to find controllers based on the incoming request.  
   • Spring MVC supports three different types of mapping request URIs to controllers: annotation, name conventions and explicit mappings.
3. Requests are processed by the Controller and the response is returned to the DispatcherServlet which then dispatches to the view.

In Figure 1, notice that in the traditional workflow the ModelAndView object is forwarded from the controller to the client. Spring lets you return data directly from the controller, without looking for a view, using the @ResponseBody annotation on a method. Beginning with Version 4.0, this process is simplified even further with the introduction of the @RestController annotation. Each approach is explained below.

## Using the @ResponseBody Annotation

When you use the @ResponseBody annotation on a method, Spring converts the return value and writes it to the http response automatically. Each method in the Controller class must be annotated with @ResponseBody.

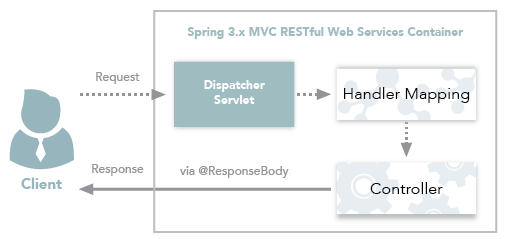


Figure 2: Spring 3.x MVC RESTful web services workflow

### Behind the Scenes

Spring has a list of HttpMessageConverters registered in the background. The responsibility of the HTTPMessageConverter is to convert the request body to a specific class and back to the response body again, depending on a predefined mime type. Every time an issued request hits @ResponseBody, Spring loops through all registered HTTPMessageConverters seeking the first that fits the given mime type and class, and then uses it for the actual conversion.

### Code Example

Let’s walk through @ResponseBody with a simple example.

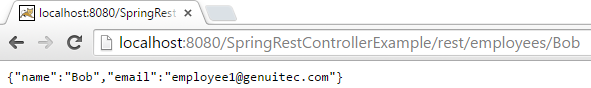
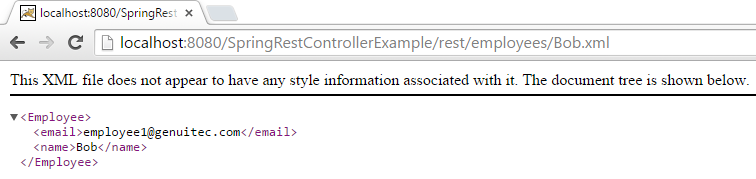
#### Project Creation and Setup

1. Create a Dynamic Web Project with Maven support in your Eclipse or MyEclipse IDE.
2. Configure Spring support for the project.  
   • If you are using Eclipse IDE, you need to download all Spring dependencies and configure your pom.xml to contain those dependencies.  
   • In MyEclipse, you only need to [install the Spring facet](https://www.genuitec.com/products/myeclipse/learning-center/spring/myeclipse-spring-development-overview/) and the rest of the configuration happens automatically.
3. Create the following Java class named Employee. This class is our POJO.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | package com.example.spring.model;    import javax.xml.bind.annotation.XmlRootElement;    @XmlRootElement(name = "Employee")  public class Employee {        String name;        String email;        public String getName() {     return name;      }        public void setName(String name) {     this.name = name;      }        public String getEmail() {     return email;      }        public void setEmail(String email) {     this.email = email;      }        public Employee() {      }    } |

1. Create the following @Controller class:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | package com.example.spring.rest;    import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.bind.annotation.ResponseBody;    import com.example.spring.model.Employee;    @Controller  @RequestMapping("employees")  public class EmployeeController {        Employee employee = new Employee();        @RequestMapping(value = "/{name}", method = RequestMethod.GET, produces = "application/json")      public @ResponseBody Employee getEmployeeInJSON(@PathVariable String name) {       employee.setName(name);     employee.setEmail("employee1@genuitec.com");       return employee;        }        @RequestMapping(value = "/{name}.xml", method = RequestMethod.GET, produces = "application/xml")      public @ResponseBody Employee getEmployeeInXML(@PathVariable String name) {       employee.setName(name);     employee.setEmail("employee1@genuitec.com");       return employee;        }    } |

1. Notice the @ResponseBody added to each of the @RequestMapping methods in the return value.
2. Add the <context:component-scan> and <mvc:annotation-driven /> tags to the Spring configuration file.  
   • <context:component-scan> activates the annotations and scans the packages to find and register beans within the application context.   
   • <mvc:annotation-driven/> adds support for reading and writing JSON/XML if the Jackson/JAXB libraries are on the classpath.   
   • For JSON format, include the jackson-databind jar and for XML include the jaxb-api-osgi jar to the project classpath.
3. Deploy and run the application on any server (e.g., Tomcat). If you are using MyEclipse, you can run the project on the [embedded Tomcat server](https://www.genuitec.com/products/myeclipse/learning-center/deploy/myeclipse-tomcat-server/).  
     
   **JSON**—Use the URL: http://localhost:8080/SpringRestControllerExample/rest/employees/Bob and the following output displays:  
   Figure 3: JSON output**XML**—Use the URL: http://localhost:8080/SpringRestControllerExample/rest/employees/Bob.xml and the following output displays:  
   Figure 4: XML output

## Using the @RestController Annotation

Spring 4.0 introduced @RestController, a specialized version of the controller which is a convenience annotation that does nothing more than add the @Controller and @ResponseBody annotations. By annotating the controller class with @RestController annotation, you no longer need to add @ResponseBody to all the request mapping methods. The @ResponseBody annotation is active by default. Click [here](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/RestController.html) to learn more.  
  
Figure 5: Spring 4.x MVC RESTful Web Services Workflow  
  
To use @RestController in our example, all we need to do is modify the @Controller to @RestController and remove the @ResponseBody from each method. The resultant class should look like the following:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | package com.example.spring.rest;    import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.bind.annotation.RestController;    import com.example.spring.model.Employee;    @RestController  @RequestMapping("employees")  public class EmployeeController {        Employee employee = new Employee();        @RequestMapping(value = "/{name}", method = RequestMethod.GET, produces = "application/json")      public Employee getEmployeeInJSON(@PathVariable String name) {       employee.setName(name);     employee.setEmail("employee1@genuitec.com");       return employee;        }        @RequestMapping(value = "/{name}.xml", method = RequestMethod.GET, produces = "application/xml")      public Employee getEmployeeInXML(@PathVariable String name) {       employee.setName(name);     employee.setEmail("employee1@genuitec.com");       return employee;        }    } |

Note that we no longer need to add the @ResponseBody to the request mapping methods. After making the changes, running the application on the server again results in same output as before.

## Conclusion

As you can see, using @RestController is quite simple and is the preferred method for creating MVC RESTful web services starting from Spring v4.0. I would like to extend a big thank you to my co-author, Swapna Sagi, for all of her help in bringing you this information!