Top-down web services development :

Top-down Web services development involves creating a WSDL and then creating the Web service using the WSDL file.

In top-down approach, first you design the implementation of the Web service by creating a WSDL file. You can then create the Web service skeleton Java classes from the wsdl, and add the required code.

JAX-WS tool wsimport can be used for creating a top down web service.

The "wsimport" can create stubs from a wsdl.

jaxws:wsimport - Maven plugin used to generate the mesasge class from XSD/WSDL files.

Bottom-up web services development :

When creating a Web service using a bottom-up approach, first you write the java classes for the web service and then create the WSDL file and publish the web service.

Although bottom-up Web service development may be faster and easier, the top-down approach is the recommended way of creating a Web service.

JAX-WS tool wsgen can be used for creating a bottoms up web service.

The wsgen tool reads an existing web service implementation class (SIB) and generates the required JAX–WS portable artifacts for web service development and deployment.​

"wsgen" tool to generate the WSDL & XSD files from EIB.

SOAP VS REST :

SOAP stands for Simple Object Access Protocol

SOAP is a protocol. SOAP was designed with a specification. It includes a WSDL file which has the required information on what the web service does in addition to the location of the web service.

Message, Binding Section, EIB, Port Types, Exception Message

SOAP requires more bandwidth for its usage.

SOAP can only work with XML format.

SOAP follows strict guidelines - W3C Standards.

Offers strict Security implementation. ( Secure Mark Up Assertion language - SAML)

( SOAP Header, Protocal level Security )

REST stands for Representational State Transfer

REST is an Architectural style

REST does not need much bandwidth when requests are sent to the server. REST messages mostly just consist of JSON messages.

REST permits different data format such as Plain text, HTML, XML, JSON, etc.

Explain SOAP Message Tags :

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

<SOAP-ENV:Header>

<!-- Optiona -->

</SOAP-ENV:Header>

<SOAP-ENV:Body>

<ns2:getCountryResponse xmlns:ns2 = "http://tutorialspoint/schemas">

<ns2:country>

<ns2:name>United States</ns2:name>

<ns2:population>46704314</ns2:population>

<ns2:capital>Washington</ns2:capital>

<ns2:currency>USD</ns2:currency>

</ns2:country>

</ns2:getCountryResponse>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

Explain WSDL sections/different Tags

Explain Marshalling/UnMarshalling (What are the frameworks used for Marshalling)

Spring WS Annoation & Sample Provider Example :

Step1 : Configure MessageDispatcherServlet in web.xml

<servlet>

<servlet-name>spring-ws</servlet-name>

<servlet-class>org.springframework.ws.transport.http.MessageDispatcherServlet

</servlet-class>

<init-param>

<param-name>transformWsdlLocations</param-name>

<param-value>true</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>spring-ws</servlet-name>

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

</servlet-mapping>

Step 2 : Configure spring-ws-servlet.xml ( by configuring portTypeName, targetNameSpace & WSDL location)

<sws:annotation-driven/>

<sws:dynamic-wsdl id="countries"

portTypeName = "CountriesPort"

locationUri = "/countryService/"

targetNamespace = "http://tutorialspoint.com/definitions">

<sws:xsd location = "/WEB-INF/countries.xsd"/>

</sws:dynamic-wsdl>

or <sws:static-wsdl id = "leave" location = "/WEB-INF/wsdl/leave.wsdl"/>

Step 3 : Create Endpoint Class annotate with @Endpoint, and annotate method with PayloadRoot & @ResponsePayload

@Endpoint

public class CountryEndPoint {

private static final String NAMESPACE\_URI = "http://tutorialspoint/schemas";

private CountryRepository countryRepository;

@Autowired

public CountryEndPoint(CountryRepository countryRepository) throws JDOMException {

this.countryRepository = countryRepository;

}

@PayloadRoot(namespace = NAMESPACE\_URI, localPart = "getCountryRequest")

@ResponsePayload

public GetCountryResponse getCountry(@RequestPayload GetCountryRequest request)

throws JDOMException {

Country country = countryRepository.findCountry(request.getName());

GetCountryResponse response = new GetCountryResponse();

response.setCountry(country);

return response;

}

}

SOAP Web Service Consumer :

Step 1 : Create Web Service Template.

Step 2 : Set Marshaller & UnMarshaller

Step 3 : Set Client Interceptor ( Log request & Response)

Step 4 : Set the Message Sender for timeout, If any. (WebServiceMessageSender - max connection, connection time out)

@Bean

public WebServiceTemplate webServiceTemplate() {

WebServiceTemplate webServiceTemplate = new WebServiceTemplate();

webServiceTemplate.setMarshaller(jaxb2Marshaller());

webServiceTemplate.setUnmarshaller(jaxb2Marshaller());

webServiceTemplate.setDefaultUri(defaultUri);

// register the LogHttpHeaderClientInterceptor

ClientInterceptor[] interceptors = new ClientInterceptor[] {new LogHttpHeaderClientInterceptor()};

webServiceTemplate.setInterceptors(interceptors);

return webServiceTemplate;

}

public class LogHttpHeaderClientInterceptor implements ClientInterceptor {

@Override

public void afterCompletion(MessageContext arg0, Exception arg1)

}

@Override

public boolean handleFault(MessageContext messageContext) throws WebServiceClientException {

return true;

}

@Override

public boolean handleRequest(MessageContext messageContext) throws WebServiceClientException {

return true;

}

@Override

public boolean handleResponse(MessageContext messageContext) throws WebServiceClientException {

return true;

}

}

Consuming Rest Service:

Step 1 :create instance of RestTemplate

Step : add Message Converter. (By default, Spring will register JacksonMessage converter for JSON)

@Bean

RestTemplate restTemplate() {

var factory = new SimpleClientHttpRequestFactory();

factory.setConnectTimeout(3000);

factory.setReadTimeout(3000);

RestTemplate restTemplate = new RestTemplate();

MappingJacksonHttpMessageConverter converter = new MappingJacksonHttpMessageConverter();

converter.setObjectMapper(new ObjectMapper());

restTemplate.getMessageConverters().add(converter);

return restTemplate;

}

Step : 4

Use the correct Method for post & get.

Get : getForEntity(URI url, Class<T> responseType)

Post: postForEntity(URI url, Object request, Class<T> responseType)

Swagger

1. Framwork for generating API Doc & Swagger WEB UI by adding annotation

2. Offers CLI jar for generating the Client Stubs

3. GUI tool for editing the API Doc

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

What are all the Swagger Annotation

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

@Api Marks a class as a Swagger resource.

@ApiImplicitParam Represents a single parameter in an API Operation.

@ApiImplicitParams A wrapper to allow a list of multiple ApiImplicitParam objects.

@ApiModel Provides additional information about Swagger models.

@ApiModelProperty Adds and manipulates data of a model property.

@ApiOperation Describes an operation or typically a HTTP method against a specific path.

@ApiParam Adds additional meta-data for operation parameters.

@ApiResponse Describes a possible response of an operation.

@ApiResponses A wrapper to allow a list of multiple ApiResponse objects.

@Authorization Declares an authorization scheme to be used on a resource or an operation.

@AuthorizationScope Describes an OAuth2 authorization scope.

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