**SPRING SOAP**

**What is SOAP?**

SOAP stands for Simple Object Access Protocol

SOAP is an application communication protocol

SOAP is a format for sending and receiving messages

SOAP is platform independent

SOAP is based on XML

SOAP is a W3C recommendation

**What is Contract First Approach for creating a SOAP Web Service?**

Spring web services promote Contract First Approach to develop a SOAP web service. That means we define the format of request and response before we start developing a web service.

Basically we are creating a SOAP web service provider here.

It involves 4 major steps below --

1) Service Definition - XML, XSD, WSDL

Typical SOAP request/response contains - Envelope, header and body. First step is to define how SOAP request/response should look like in XML which will be further encapsulated in SOAP body.

***Skeleton SOAP Message:***

<?xml version="1.0"?>

<soap:Envelope

xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"

soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Header>

...

</soap:Header>

<soap:Body>

...

<Actual request or response in XML format>

<!-- In case of an error/exception, soap:Fault is sent in soap:body -->

<soap:Fault>

...

</soap:Fault>

</soap:Body>

</soap:Envelope>

***XSD -*** XML Schema Definition defines and validates exact structure of SOAP request/response.

***What are valid values for each element in the SOAP request/response? --***

***WSDL -***

WSDL stands for Web Services Description Language. WSDL is used to describe web services at an endpoint level. WSDL is written in XML.

An WSDL document describes a web service. It specifies the location of the service, and the methods of the service.

The main structure of a WSDL document looks like this:

<definitions>

<types>

Defines the (XML Schema) data types used by the web service

</types>

<message>

Defines the data elements for each operation

</message>

<portType>

Describes the interface/operations that can be performed and the messages involved.

</portType>

<binding>

***How operations are exposed (SOAP over http or SOAP over MQ)?***

Defines the protocol and data format for each port type

</binding>

<service>

Endpoint - What is the URL which customers can use to call this web service

</service>

</definitions>

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint.

WSDL can be generated by using framework tools like - WSDL generator.

2) XML Java Binding - JAXB - Java API for XML Binding - JAXB is available as a plugin in maven/gradle.

XML ----> Java Object -- Unmarshalling

Java Object ----> XML -- Marshalling

JAXB uses XSDs for Marshalling or Unmarshalling.

3) Framework Configuration - Endpoint, WSDL-Generation

Endpoint configuration is required for SOAP web-service producer. Endpoint defines SOAP URL, Request payload and response payload with @Endpoint annotation at class level, @RequestPayload and @ResponsePayload annotations at method level.

Enable Spring web services using @EnableWs annotation in spring configuration class (@Configuration). Basically we are defining a ServletRegistrationBean with MessageDispatcherServlet and urlmapping to handle incoming SOAP requests.

WSDL is generated by Spring Framework. We need to configure Spring web services framework to generate wsdl from xsd file. PortType and NameSpace also needs to be configured.

4) Test service using SOAP Web Service Client - Wizdler Chrome Plugin

Here we are using Wizdler Chrome Plugin as a SOAP web-service consumer. We can also use SOAP-UI or develop our own SOAP client using Spring framework (WebServiceTemplate).

**What is a concept of namespace in XML?**

XML Namespaces provide a way to avoid element name conflicts. In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications. Conflict can be due to same element names in different XMLs. A user or an XML application will not know how to handle these differences.

Name conflicts in a XML can easily be avoided using a name prefix.

Below is a fragment of a single XML carries information about an HTML table(<h:table>), and a piece of furniture (<f:table>)

<root>

<h:table>

<h:tr>

<h:td>Apples</h:td>

<h:td>Bananas</h:td>

</h:tr>

</h:table>

<f:table>

<f:name>African Coffee Table</f:name>

<f:width>80</f:width>

<f:length>120</f:length>

</f:table>

</root>

***XML Namespaces - The xmlns Attribute --***

When using prefixes in XML, a namespace for the prefix must be defined. The namespace can be defined by an xmlns attribute in the start tag of an element. The namespace declaration has the following syntax. xmlns:prefix="URI".

<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">

<h:tr>

<h:td>Apples</h:td>

<h:td>Bananas</h:td>

</h:tr>

</h:table>

<f:table xmlns:f="https://www.w3schools.com/furniture">

<f:name>African Coffee Table</f:name>

<f:width>80</f:width>

<f:length>120</f:length>

</f:table>

</root>

When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.

Namespaces can also be declared in the XML root element:

<root xmlns:h="http://www.w3.org/TR/html4/"

xmlns:f="https://www.w3schools.com/furniture">

<h:table>

<h:tr>

<h:td>Apples</h:td>

<h:td>Bananas</h:td>

</h:tr>

</h:table>

<f:table>

<f:name>African Coffee Table</f:name>

<f:width>80</f:width>

<f:length>120</f:length>

</f:table>

</root>

Default Namespaces

Defining a default namespace for an element saves us from using prefixes in all the child elements. It has the following syntax:

xmlns="namespaceURI"

This XML carries HTML table information:

<table xmlns="http://www.w3.org/TR/html4/">

<tr>

<td>Apples</td>

<td>Bananas</td>

</tr>

</table>

**Explain XSD (XML Schema Definition) in detail?**

XSD is quite a complex and vast topic to deal with. Normally we do not create xsd files manually. Like WSDLs, we use generated XSDs in our code. However, we need to understand few basics of XSD creation.

An XML Schema describes the structure of an XML document. The XML Schema language is also referred to as XML Schema Definition (XSD). XML Schema is an XML-based (and more powerful) alternative to DTD.

The purpose of an XML Schema is to define the legal building blocks of an XML document:

* the elements and attributes that can appear in a document
* the number of (and order of) child elements
* data types for elements and attributes
* default and fixed values for elements and attributes

***Typical XSD looks like ---***

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" <!-- if ':xs' is absent, it becomes default namespace which is NOT a best practice. Always include namespace prefix. It indicates that the elements and data types used in the schema come from the "<http://www.w3.org/2001/XMLSchema>" namespace. It also specifies that the elements and data types that come from the "<http://www.w3.org/2001/XMLSchema>" namespace should be prefixed with xs: --> targetNamespace="http://in28minutes.com/courses" xmlns:tns="http://in28minutes.com/courses" <!-- ':tns' is a namespace prefix for target XML. It indicates that the elements (GetCourseDetailsRequest,id,GetCourseDetailsResponse,CourseDetails,status) and data types (tns:CourseDetails,tns:Status) defined by this schema come from the "<https://www.w3schools.com>" namespace. --> elementFormDefault="qualified"> <!-- indicates that any elements used by the XML document which were declared in this schema must be namespace qualified. -->

<!-- The GetCourseDetailsRequest, GetCourseDetailsResponse,DeleteCourseDetailsRequest and DeleteCourseDetailsResponse elements are complex types because it contains other elements.

The other elements (id, Status) are simple types because they do not contain other elements. -->

<xs:element name="GetCourseDetailsRequest">

<xs:complexType>

<xs:sequence>

<xs:element name="id" type="xs:int" />

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="GetCourseDetailsResponse">

<xs:complexType>

<xs:sequence>

<xs:element name="CourseDetails" type="tns:CourseDetails" /> <!-- complex type containing another complex type. As this is a custom complex type, we need to use targetNamespace to avoid naming conflicts when used with other XMLs -->

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="DeleteCourseDetailsRequest">

<xs:complexType>

<xs:sequence>

<xs:element name="id" type="xs:int" />

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="DeleteCourseDetailsResponse">

<xs:complexType>

<xs:sequence>

<xs:element name="status" type="tns:Status" /> <!-- complex type containing another simpleType -->

</xs:sequence>

</xs:complexType>

</xs:element>

<!-- Embedded simpleType Data type definition -->

<xs:simpleType name="Status">

<xs:restriction base="xs:string">

<xs:enumeration value="SUCCESS" />

<xs:enumeration value="FAILURE" />

</xs:restriction>

</xs:simpleType>

<!-- Embedded complexType Data type definition -->

<xs:complexType name="CourseDetails">

<xs:sequence>

<xs:element name="id" type="xs:int" />

<xs:element name="name" type="xs:string" />

<xs:element name="description" type="xs:string" />

</xs:sequence>

</xs:complexType>

</xs:schema>

Below XML documents have a reference to an XML Schema:

<?xml version="1.0"?>

<GetCourseDetailsRequest

xmlns="http://in28minutes.com/courses" <!-- default namespace -->

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://in28minutes.com/courses course-details.xsd"> <!--xsi:schemaLocation={namespace} {Location of XSD} -->

<id>234</id>

</GetCourseDetailsRequest>

<?xml version="1.0"?>

<GetCourseDetailsResponse

xmlns="http://in28minutes.com/courses" <!-- default namespace -->

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://in28minutes.com/courses course-details.xsd"> <!--xsi:schemaLocation={namespace} {Location of XSD} -->

<CourseDetails>

<id>234</id>

<name>My Spring</name>

<description>basics of Spring</description>

</CourseDetails>

</GetCourseDetailsResponse>

***Simple Element***

A simple element is an XML element that can contain only text. It cannot contain any other elements or attributes. However, the "only text" restriction is quite misleading. The text can be of many different types. It can be one of the types included in the XML Schema definition (boolean, string, date, etc.), or it can be a custom type that you can define yourself.

***Complex Element***

A complex element is an XML element that contains other elements and/or attributes.

There are four kinds of complex elements:

* empty elements
* elements that contain only other elements
* elements that contain only text
* elements that contain both other elements and text

***XSD Attributes***

Simple elements cannot have attributes. If an element has attributes, it is considered to be of a complex type. But the attribute itself is always declared as a simple type.

Here is an XML element with an attribute:

<lastname lang="EN">Smith</lastname>

And here is the corresponding attribute definition:

<xs:attribute name="lang" type="xs:string"/>

**What is JAXB and how to configure it?**

JAXB is Java API for XML Binding. JAXB configuration involves

1) Add jaxb2-maven-plugin in maven/gradle

2) Add xjc goal of jaxb2-maven-plugin

3) Define XSD source Folder

4) Define source folder for generated Java soruce files

5) Set clear java class folder to false to prevent clearing generated Java files every time.

The jaxb2-maven-plugin has four main goals, listed below --

1) jaxb2:schemagen Creates XML Schema Definition (XSD) file(s) from annotated Java sources.

2) jaxb2:testSchemagen Creates XML Schema Definition (XSD) file(s) from annotated Java test sources.

3) jaxb2:xjc Generates Java sources from XML Schema(s).

4) jaxb2:testXjc Generates Java test sources from XML Schema(s).

xjc is used more frequently in maven or gradle to generate Java sources from XSD.

**What is SOAP Fault?**

A SOAP fault is an error in a SOAP communication resulting from an exception during processing of SOAP request, incorrect message format, header-processing problems, or incompatibility between applications.

The optional SOAP Fault element hold errors and status information for a SOAP message. If a Fault element is present, it must appear as a child element of the Body element. A Fault element can only appear once in a SOAP message.

The SOAP Fault element has the following sub elements:

<faultcode> A code for identifying the fault

<faultstring> A human readable explanation of the fault

<faultactor> Information about who caused the fault to happen

<detail> Holds application specific error information related to the Body element

In case of an exception, we can define custom exception with @SoapFault annotation having attributes like faultCode and faultstring.

Custom Fault code can be provided using @SoapFault annotation.

**How to implement security for SOAP web services with ws-security?**

WS-security is a set of specifications which provide security around SOAP based web services.

WS-security is transport independent (HTTP/MQ doesn't matter).

WS-security provides security with Authentication based on passwords, Digital Signatures or certificates.

XWSS - XML and Web Services Security is one of the implementations of WS-security.

XwsSecurityInterceptor can be used to implement security policy around web services.

***Implementation --***

1) Add required maven/gradle dependencies

2) Define XwsSecurityInterceptor bean in Spring web services configuration

3) Define Call back handler (SimplePasswordValidationCallbackHandler) to intercept incoming requests and validate for security

4) Define Authentication credentials in SimplePasswordValidationCallbackHandler.

5) Define security policy in securityPolicy.xml

6) Add XwsSecurityInterceptor to list of existing interceptors.

**What is XSLT?**

XSLT is a language for transforming XML documents. XSL (eXtensible Stylesheet Language) is a styling language for XML.

XSLT stands for XSL Transformations. XSLT can be used to transform XML documents into other formats (like transforming XML into HTML).

XPath is a language for navigating in XML documents. XPath is a major element in the XSLT standard. XPath can be used to navigate through elements and attributes in an XML document.

XQuery is a language for querying XML documents. XQuery is to XML what SQL is to databases. XQuery is designed to query XML data.

XQuery is a language for finding and extracting elements and attributes from XML documents. XQuery is built on XPath expressions