**Apache CXF :**

[Apache CXF](http://cxf.apache.org/download.html) is a free and open source project, and a fully featured Webservice framework.

It helps you building webservices using different front-end API's, like as JAX-RS and JAX-WS.

Services will talk different protocols such as SOAP, RESTful HTTP, CORBA & XML/HTTP and work with different transports like JMS, HTTP or JBI.

Apache CXF Project was created by the merger of the Celtix and XFire projects. These two projects were merged by folks working together at the Apache Software Foundation.

**JAX-WS Support**

CXF implements the JAX-WS APIs which make building web services easy. JAX-WS encompasses many different areas:

* Generating WSDL from Java classes and generating Java classes from WSDL
* Provider API which allows you to create simple messaging receiving server endpoints
* Dispatch API which allows you to send raw XML messages to server endpoints
* Much more...

**Spring Integration**

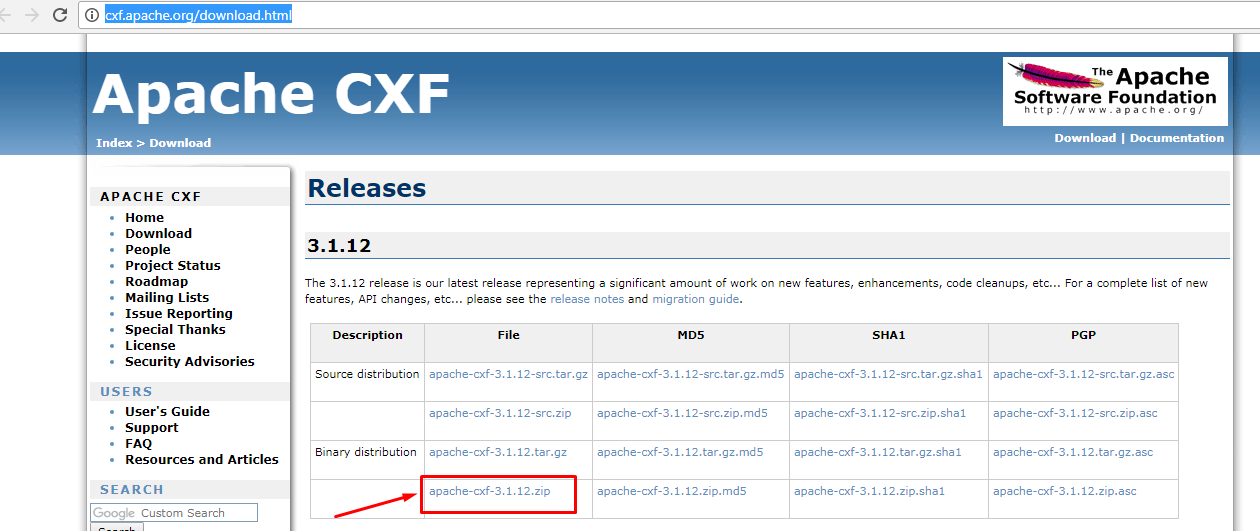
Spring is a first class citizen with Apache CXF. CXF supports the Spring 4.x XML syntax, making it trivial to declare endpoints which are backed by Spring and inject clients into your application.

**Note:**

The Current Version of CXF is 3.1.x .

To download CXF Binary Distribution use the following URL:

<http://cxf.apache.org/download.html>



After extracting the binary distribution set Path Environment variables to work with CXF tools.

NOTE: CXF provides the following tools for webservices development

1. Java2ws
2. Wsdl2java

As we know already webservices development is possible in two Approaches

1)Contract First Approach (OR) To down Approach

2)Contract Last Approach (OR) Bottom Up Approach

CXF supports Both the Approaches to develop webservices provider Applications.

CXF follows JAX-WS Guidelines.

**1)Contract Last Approach (OR) Bottom Up Approach:**

**Endpoint : Servlet**

**MEF : document/literal**

**API: JAXWS**

**Implementation :Apache CXF**

**Steps:-**

**Step1:** create Dynamic web project

Step2:- Write SEI and SEI implemented classes

Note :The SEI and SEI implemented classes must denote with

@WebService Annotation.

Step3: generate the WSDL document by running java2ws tool.

Step4: Write Spring cfg file ,to define webservices deployment description.

Step5: Configure CXFServlet and ContextLoaderListener in web.xml

Step6:- Deploy the Project into Server

**Include the following jar files in lib floder :**

Following jar must be in classpath

1. commons-logging-1.1.1.jar
2. aopalliance-1.0.jar
3. cxf-2.7.3.jar
4. httpasyncclient-4.0-beta3.jar
5. httpclient-4.2.1.jar
6. httpcore-4.2.2.jar
7. httpcore-nio-4.2.2.jar
8. neethi-3.0.2.jar
9. spring-aop-3.0.7.RELEASE.jar
10. spring-asm-3.0.7.RELEASE.jar
11. spring-beans-3.0.7.RELEASE.jar
12. spring-context-3.0.7.RELEASE.jar
13. spring-core-3.0.7.RELEASE.jar
14. spring-expression-3.0.7.RELEASE.jar
15. spring-web-3.0.7.RELEASE.jar
16. wsdl4j-1.6.2.jar
17. xmlschema-core-2.0.3.ja

**Note :-**

**If we are working with maven to get the CXF jar files include the following three dependencis in pom.xml file**

<dependency>

<groupId>org.apache.cxf</groupId>

<artifactId>cxf-rt-frontend-jaxws</artifactId>

<version>3.1.6</version>

</dependency>

<dependency>

<groupId>org.apache.cxf</groupId>

<artifactId>cxf-rt-transports-http</artifactId>

<version>3.1.6</version>

</dependency>

<dependency>

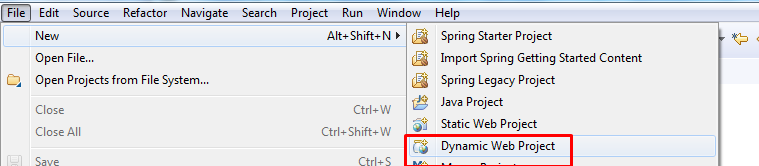
<groupId>org.springframework</groupId>

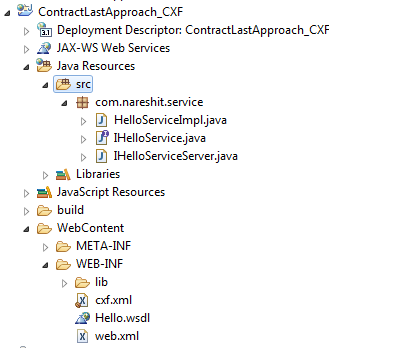
<artifactId>spring-web</artifactId>

<version>4.0.0.RELEASE</version>

</dependency>

1. create Dynamic web project





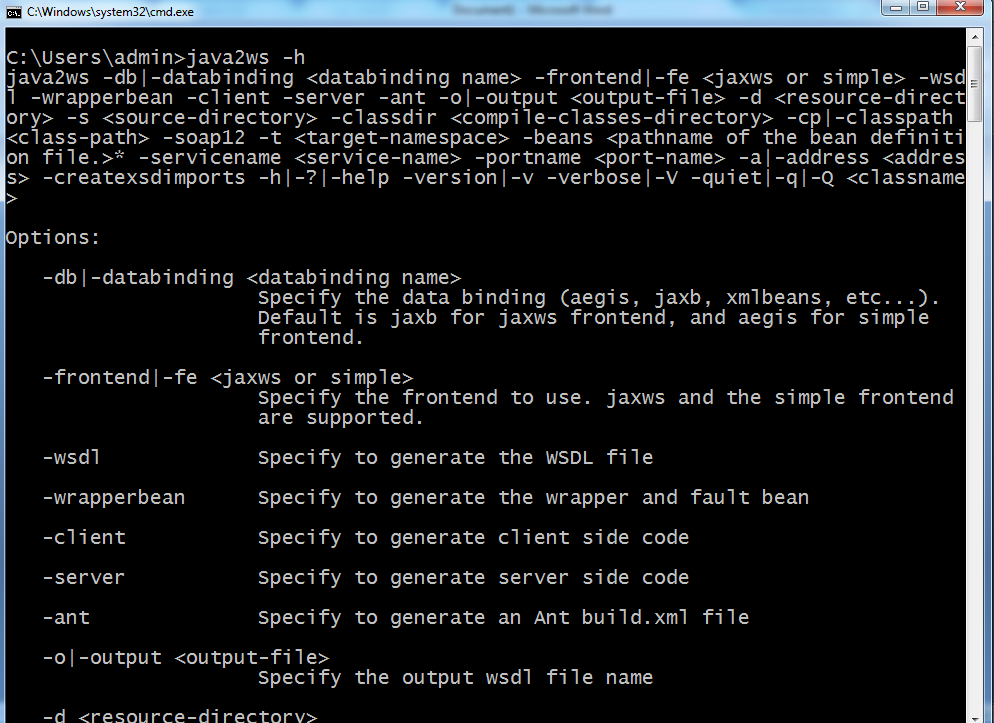
1. create SEI and SEI implemented classes as for your requirement

|  |
| --- |
| package com.nareshit.service;  import javax.jws.WebService;  @WebService  public interface IHelloService {  public String sayHello(String name);  } |

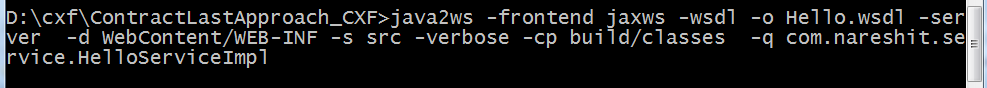
|  |
| --- |
| package com.nareshit.service;  import javax.jws.WebService;  @WebService(endpointInterface="com.nareshit.service.IHelloService")  public class HelloServiceImpl implements IHelloService{  @Override  public String sayHello(String name) {  String msg="Hello "+name+" welcome to Apache CXF";  return msg;  }  } |

Step3:- run java2ws tool to generate WSDL document

Java2ws tool options.



To generate wsdl document run as follows



Refresh the project indetify the generated classes and wsdl document.

**Hello.wsdl (generated wsdl document)**

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

<wsdl:definitions name=*"HelloServiceImplService"* targetNamespace=*"http://service.nareshit.com/"* xmlns:wsdl=*"http://schemas.xmlsoap.org/wsdl/"* xmlns:xsd=*"http://www.w3.org/2001/XMLSchema"* xmlns:tns=*"http://service.nareshit.com/"* xmlns:soap=*"http://schemas.xmlsoap.org/wsdl/soap/"*>

<wsdl:types>

<xs:schema xmlns:tns=*"http://service.nareshit.com/"* xmlns:xs=*"http://www.w3.org/2001/XMLSchema"* elementFormDefault=*"unqualified"* targetNamespace=*"http://service.nareshit.com/"* version=*"1.0"*>

<xs:element name=*"sayHello"* type=*"tns:sayHello"*/>

<xs:element name=*"sayHelloResponse"* type=*"tns:sayHelloResponse"*/>

<xs:complexType name=*"sayHello"*>

<xs:sequence>

<xs:element minOccurs=*"0"* name=*"arg0"* type=*"xs:string"*/>

</xs:sequence>

</xs:complexType>

<xs:complexType name=*"sayHelloResponse"*>

<xs:sequence>

<xs:element minOccurs=*"0"* name=*"return"* type=*"xs:string"*/>

</xs:sequence>

</xs:complexType>

</xs:schema>

</wsdl:types>

<wsdl:message name=*"sayHello"*>

<wsdl:part name=*"parameters"* element=*"tns:sayHello"*>

</wsdl:part>

</wsdl:message>

<wsdl:message name=*"sayHelloResponse"*>

<wsdl:part name=*"parameters"* element=*"tns:sayHelloResponse"*>

</wsdl:part>

</wsdl:message>

<wsdl:portType name=*"IHelloService"*>

<wsdl:operation name=*"sayHello"*>

<wsdl:input name=*"sayHello"* message=*"tns:sayHello"*>

</wsdl:input>

<wsdl:output name=*"sayHelloResponse"* message=*"tns:sayHelloResponse"*>

</wsdl:output>

</wsdl:operation>

</wsdl:portType>

<wsdl:binding name=*"HelloServiceImplServiceSoapBinding"* type=*"tns:IHelloService"*>

<soap:binding style=*"document"* transport=*"http://schemas.xmlsoap.org/soap/http"*/>

<wsdl:operation name=*"sayHello"*>

<soap:operation soapAction=*""* style=*"document"*/>

<wsdl:input name=*"sayHello"*>

<soap:body use=*"literal"*/>

</wsdl:input>

<wsdl:output name=*"sayHelloResponse"*>

<soap:body use=*"literal"*/>

</wsdl:output>

</wsdl:operation>

</wsdl:binding>

<wsdl:service name=*"HelloServiceImplService"*>

<wsdl:port name=*"HelloServiceImplPort"* binding=*"tns:HelloServiceImplServiceSoapBinding"*>

<soap:address location=*"http://localhost:9090/HelloServiceImplPort"*/>

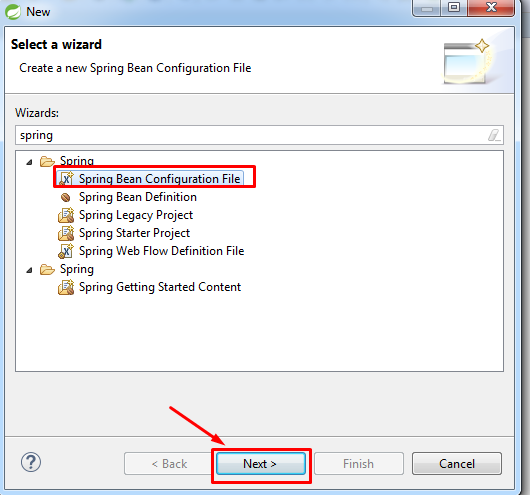
</wsdl:port>

</wsdl:service>

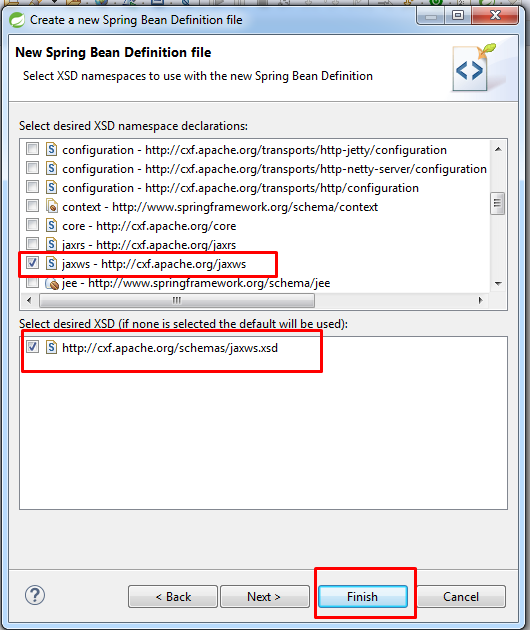
</wsdl:definitions>

**Step4:- write spring cfg file with webservices deployment description**

To create spring cfg file right click on project 🡪new 🡪other🡪search for spring bean cfg file🡪select the file 🡪click on Next button



Enter file name cxf.xml and click on next button->select xsd (jaxws xsd) 🡪click on Finish button



**cxf.xml file**

<beans xmlns=*"http://www.springframework.org/schema/beans"*

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

xmlns:jaxws=*"http://cxf.apache.org/jaxws"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd"*>

<jaxws:endpoint id=*"helloService"*  implementor=*"com.nareshit.service.HelloServiceImpl"* address=*"/hello"*/>

</beans>

**Step5:-Configure the CXFServlet and ContextLoaderListener**

The ContextLoaderListener will create Spring IOC container.

The ContextLoaderListener will load spring cfg file from WEB-INF folder.

By default it searches the spring cfg file name as applicationContext.xml . If the file name is different

name then configure the custom file name with Context Parameter . The Context Parameter name is contextConfigLocation .

**web.xml**

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

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"* xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"* id=*"WebApp\_ID"* version=*"3.1"*>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/cxf.xml</param-value>

</context-param>

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

<servlet>

<servlet-name>cxfServlet</servlet-name>

<servlet-class>

org.apache.cxf.transport.servlet.CXFServlet</servlet-class>

<load-on-startup>2</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>cxfServlet</servlet-name>

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

</servlet-mapping>

</web-app>

**CXF Project with Contract First Approach :-**

**Contract First Approach means first we can write wsdl document and last we will generate service classes.**

**Steps:-**

Step1: write the wsdl document

Step2: generate the SEI and server side artifacts by running wsdl2java tool

Step3: create SEI implemented class and write the Bussiness Logic in Sei implemented class

Step4: Write spring cfg file with webservices Deployment Description

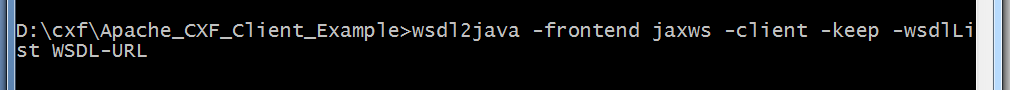
Step5:- Configure the CXFServlet and ContextLoaderListener in web.xml file

**Consumer Application by CXF:-**

**Steps:**

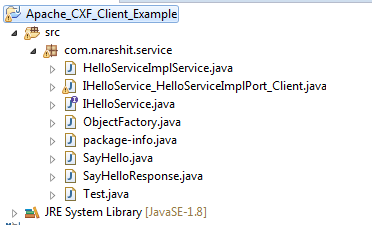
Step1:- create Standloane java project

Step2:- generate client side artifacts by running wsdl2java tool



Step3:- write the Logic in main method class to consume the service.

1. Create Service object
2. Get Port object (SEI object)
3. Call webservice methods



Step2:- generate client side artifacts by running wsdl2java tool

**Step3:-** write the Logic in main method class to consume the service.

**Test.java**

**package** com.nareshit.client;

**import** com.nareshit.service.HelloServiceImplService;

**import** com.nareshit.service.IHelloService;

**public** **class** Test {

**public** **static** **void** main(String[] args){

HelloServiceImplService service=**new** HelloServiceImplService();

IHelloService sei=service.getHelloServiceImplPort();

System.***out***.println(sei.sayHello("sathish"));

}

}