BenchResources.Net

Java, Collection, JDBC, Spring, Web Services, Maven, Android, Oracle SOA-OSB & Open Source

Apache CXF JAX-WS: SOAP based Web Service using Top-Down approach + Integrating with Spring & Hibernate ORM framework

December 6, 2014 SJ Apache CXF (SOAP) 5

In previous article, we have integrated spring framework with SOAP web service. We will extend the same article to integrate with hibernate ORM framework for database operation

We will use MySql database for this demo example

Technology Used

- Java 1.7
- Eclipse Luna IDE
- Apache CXF-3.0.2
- Spring-4.1.0.RELEASE
- Hibernate-4.2.15.Final
- MySql-connector-java-5.1.32
- Apache Maven 3.2.1
- Apache Tomcat-8.0.15
- Oracle Weblogic server 12c

Mavenize or download required jars

SEARCH TUTORIALS

SEARCH …

SUBSCRIBE VIA EMAIL

Join 184 other subscribers

Email Address

SUBSCRIBE

POPULAR ARTICLES

Spring JDBC: An example on JdbcTemplate using Annotation
Java JDBC: An example to connect MS Access database

Add *apache-cxf-3.0.2*, *spring-4.1.0.RELEASE*, *hibernate-4.2.15.Final* & *MySql-5.1.32* dependencies to pom.xml

```
<!-- properties -->
<properties>
    <cxf.version>3.0.2</cxf.version>
    <spring.version>4.1.0.RELEASE</spring.ver
    <hibernate.version>4.2.15.Final</hibernat
    <mysql.version>5.1.32</mysql.version>
    <cxf.scope>compile</cxf.scope>
    <jaxws.scope>compile</jaxws.scope>
    <spring.scope>compile</spring.scope>
    <hibernate.scope>compile</hibernate.scope
    <spring.scope>compile</spring.scope>
    <compileSource>1.7</compileSource>
    <maven.compiler.target>1.7</maven.compile
    <maven.compiler.source>1.7</maven.compile
    <project.build.sourceEncoding>UTF-8</proj
</properties>

<dependencies>
    <!-- apache cxf jax-ws-3.0.2 -->
    <dependency>
        <groupId>org.apache.cxf</groupId>
        <artifactId>cxf-rt-frontend-jaxws</ar
        <version>${cxf.version}</version>
        <scope>${cxf.scope}</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.cxf</groupId>
        <artifactId>cxf-rt-transports-http</a
        <version>${cxf.version}</version>
        <scope>${cxf.scope}</scope>
    </dependency>

    <!-- Spring Framework-4.x -->
    <dependency>
        <groupId>org.springframework</groupIc
        <artifactId>spring-webmvc</artifactIc
        <version>${spring.version}</version>
        <scope>${spring.scope}</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupIc
        <artifactId>spring-orm</artifactId>
        <version>${spring.version}</version>
        <scope>${spring.scope}</scope>
    </dependency>

    <!-- Hibernate Core-4.2.x -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactI
        <version>${hibernate.version}</versic
        <scope>${hibernate.scope}</scope>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-ehcache</artifa
        <version>${hibernate.version}</versic
        <scope>${hibernate.scope}</scope>
    </dependency>

    <!-- MySql-Connector-5.1.32 -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</art
        <version>${mysql.version}</version>
```

JDBC: An example to connect MS Access database in Java 8

Oracle OSB 12c: Service Callout and Routing Table example

Oracle OSB 12c: Hello World service with both Business and Proxy Service

Benchl

G+

85 follow

BenchResou

Like Page

Be the first of your friends tc

```
            <scope>compile</scope>
        </dependency>
    </dependencies>
```

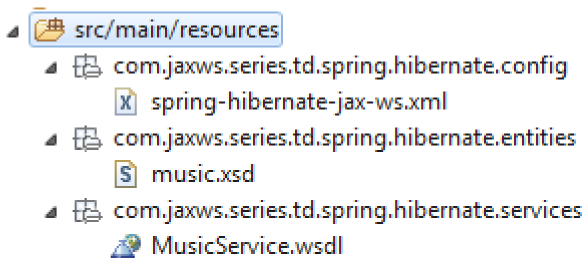
Folks who aren't familiar with Maven concepts or don't require maven for their project, can download the below jars individually from the central repository or maven repository or maven2 and include them in the classpath

- cxf-core-3.0.2
- cxf-rt-bindings-soap-3.0.2
- cxf-rt-bindings-xml-3.0.2
- cxf-rt-databinding-jaxb-3.0.2
- cxf-rt-frontend-jaxws-3.0.2
- cxf-rt-frontend-simple-3.0.2
- cxf-rt-transports-http-3.0.2
- cxf-rt-ws-addr-3.0.2
- cxf-rt-ws-policy-3.0.2
- cxf-rt-wsdl-3.0.2
- jaxb-core-2.2.10
- jaxb-impl-2.2.10
- neethi-3.0.3
- stax2-api-3.1.4
- woodstox-core-asl-4.4.1
- wsdl4j-1.6.3
- xml-resolver-1.2
- xmlschema-core-2.1.0
- aopalliance-1.0
- asm-3.3.1
- spring-aop-4.1.0.RELEASE
- spring-beans-4.1.0.RELEASE
- spring-context-4.1.0.RELEASE
- spring-core-4.1.0.RELEASE
- spring-expression-4.1.0.RELEASE
- spring-web-4.1.0.RELEASE
- hibernate-core-4.2.15.Final
- hibernate-jpa-2.0-api-1.0.1.Final
- hibernate-commons-annotations-4.0.2.Final
- hibernate-ehcache-4.2.15.Final
- ehcache-core-2.4.3
- slf4j-api-1.6.1
- mysql-connector-java-5.1.32

Steps to generate Java artifacts from WSDL/XSD

- write/design XML Schema (XSD)

- similarly, write/design WSDL document including above XSD for Type attributes
- configure maven plugins (wsimport/wsdl2java goal) in pom.xml with correct and complete path of the wsdl file under wsdlOptions/wsdlOption
- Run maven command "*mvn generate-sources*" from project's context-root
- java artifacts will be generated under "*generated*" folder within specified targetNamespace



```
src/main/resources
    com.jaxws.series.td.spring.hibernate.config
        spring-hibernate-jax-ws.xml
    com.jaxws.series.td.spring.hibernate.entities
        music.xsd
    com.jaxws.series.td.spring.hibernate.services
        MusicService.wsdl
```

Let us understand above steps in more detail

Write/design well-formed XML Schema

music.xsd
(src/main/resources/com/jaxws/series/td/spring/hibernate/entities)

Below XSD contains two elements with name "*MusicListRequestType*" and "*MusicListResponseType*" with a "*BusinessFaultType*" element in case of any exception

- MusicListRequestType contains single string called *composerName*
- MusicListResponseType contains simple type string called *composer* and complex type which references to *MovieListType* (which again references to *MovieType*)
- BusinessFaultType for exception wraps three sub-elements namely *errorCode*, *errorMessage* and *errorDescription*

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001
    targetNamespace="http://benchresources.ir
    elementFormDefault="qualified">

    <!-- Music List Request Type -->
    <xsd:element name="MusicListRequestType">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="composerNa
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <!-- Music List Response Type -->
    <xsd:element name="MusicListResponseType"
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="composer"
                <xsd:element ref="tns:MovieLi
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <!-- List of Movies -->
    <xsd:element name="MovieListType">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="tns:MovieTy
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <!-- Movie Type -->
    <xsd:element name="MovieType">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="movieName"
                <xsd:element name="year" type
                <xsd:element name="director"
                <xsd:element name="comments"
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <!-- Business Exception Type -->
    <xsd:element name="BusinessFaultType">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="errorCode"
                <xsd:element name="errorMessa
                <xsd:element name="errorDescr
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

</xsd:schema>
```

Write/design well-formed WSDL

MusicService.wsdl
(src/main/resources/com/jaxws/series/td/spring/hibernate/services)

This is the contract document for Music Service exposing one operation called "*getMovieDetailByComposer*" whose input argument is "*MusicListRequestType*" and return type is "*MusicListResponseType*" and fault is "*BusinessFaultType*"

Note: In case of any exception while invoking this exposed service, business exception will be returned stating the reason instead of actual response type

```
<?xml version="1.0" encoding="UTF-8" standa
```

```
<wsdl:definitions xmlns:wsdl="http://schemas.
    xmlns:xsd="http://www.w3.org/2001/XMLSche
    targetNamespace="http://benchresources.ir
    xmlns:tns="http://benchresources.in/servi
    xmlns:muzix="http://benchresources.in/ent

    <wsdl:types>
        <xsd:schema targetNamespace="http://b
            <xsd:import namespace="http://ber
                schemaLocation="../entities/m
        </xsd:schema>
    </wsdl:types>

    <wsdl:message name="MusicListRequest">
        <wsdl:part element="muzix:MusicListRe
    </wsdl:message>
    <wsdl:message name="MusicListResponse">
        <wsdl:part element="muzix:MusicListRe
    </wsdl:message>
    <wsdl:message name="BusinessException">
        <wsdl:part element="muzix:BusinessFau
    </wsdl:message>

    <wsdl:portType name="IMusicService">
        <wsdl:operation name="getAllMovieDeta
            <wsdl:input message="tns:MusicLis
            <wsdl:output message="tns:MusicLi
            <wsdl:fault name="businessExcepti
        </wsdl:operation>
    </wsdl:portType>

    <wsdl:binding name="MusicServiceSOAPBindi
        <soap:binding style="document"
            transport="http://schemas.xmlsoap
        <wsdl:operation name="getAllMovieDeta
            <soap:operation
                soapAction="" />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
            <wsdl:fault name="businessExcepti
                <soap:fault name="businessExc
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>

    <wsdl:service name="MusicService">
        <wsdl:port name="MusicServicePort" bi
            <soap:address
                location="http://localhost:80
        </wsdl:port>
    </wsdl:service>

</wsdl:definitions>
```

Configure maven plugin in pom.xml (wsdl2java goal)

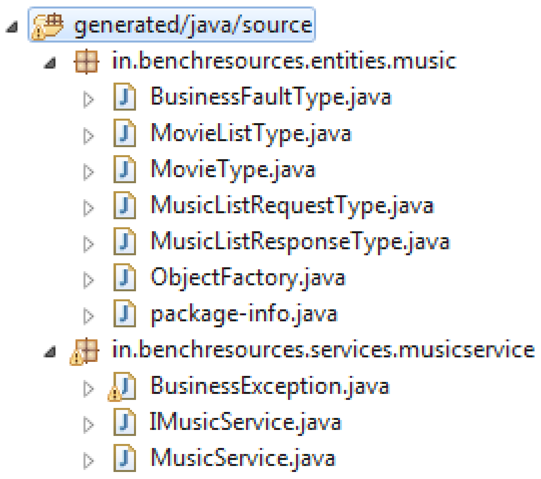
This plugin which defines *wsdl2java* goal from *cxf-codegen-plugin* generates java artifacts from the supplied WSDL file under resources folder

```
<!-- plugin 4- apache cxf codegen wsdl2java ?:
<plugin>
    <groupId>org.apache.cxf</groupId>
    <artifactId>cxf-codegen-plugin</artifactI
    <version>3.0.2</version>
    <executions>
        <execution>
            <configuration>
                <sourceRoot>${basedir}/genera
                <wsdlOptions>
                    <wsdlOption>
                        <wsdl>${basedir}/src/
                    </wsdlOption>
                </wsdlOptions>
            </configuration>
            <goals>
                <goal>wsdl2java</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Run "mvn generate-sources"

Look at the generated java source files in the generated folder

After running above maven command, you will get to see below generated java files



```
generated/java/source
    in.benchresources.entities.music
        BusinessFaultType.java
        MovieListType.java
        MovieType.java
        MusicListRequestType.java
        MusicListResponseType.java
        ObjectFactory.java
        package-info.java
    in.benchresources.services.musicservice
        BusinessException.java
        IMusicService.java
        MusicService.java
```

- IMusicService.java
- MusicRequestType.java
- MusicResponseType.java
- BusinessFaultType.java
- BusinessException.java
- MusicService.java
- ObjectFactory.java
- package-info.java

We will look at one file IMusicService.java, for other files you can download this eclipse project provided in the last section "Download Project"

This interface which is implemented by our endpoint business implementation class

IMusicService.java

```
package in.benchresources.services.musicser?i

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.xml.bind.annotation.XmlSeeAlso;

/**
 * This class was generated by Apache CXF 3.0
 * 2014-11-21T01:27:54.560+05:30
 * Generated source version: 3.0.2
 *
 */
@WebService(targetNamespace = "http://benchre
@XmlSeeAlso({in.benchresources.entities.music
@SOAPBinding(parameterStyle = SOAPBinding.Par
public interface IMusicService {

    @WebResult(name = "MusicListResponseType"
    @WebMethod(action = "http://benchresource
    public in.benchresources.entities.music.M
        @WebParam(partName = "parameters", na
        in.benchresources.entities.music.Musi
    ) throws BusinessException;
}
```

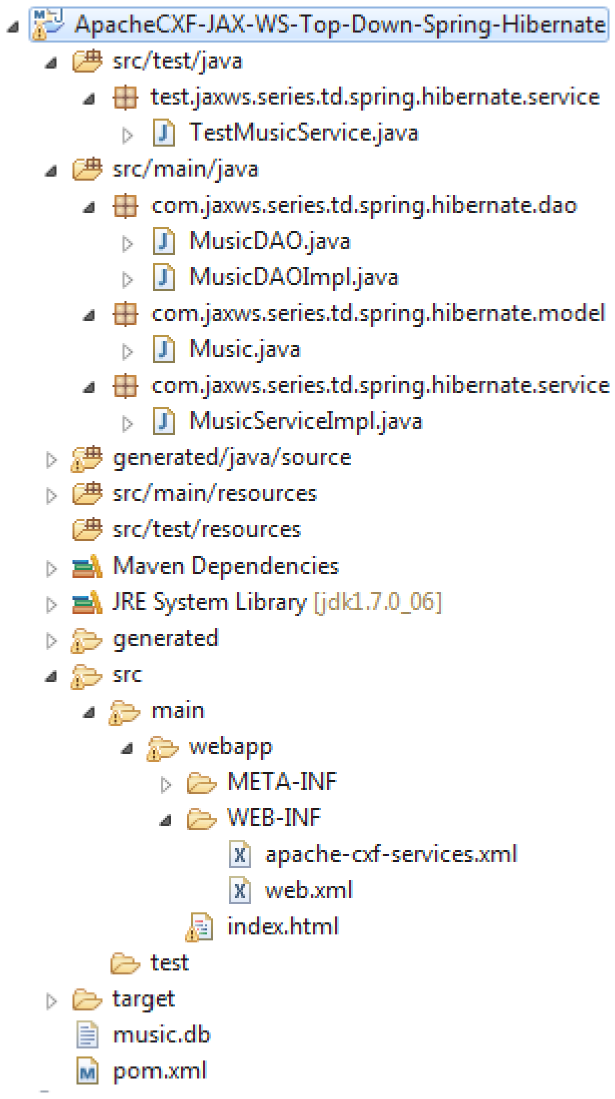
Directory Structure

Before moving on, let us understand the directory/package structure once you create project and/plus after generating java artifacts in Eclipse IDE

Maven has to follow certain directory structure

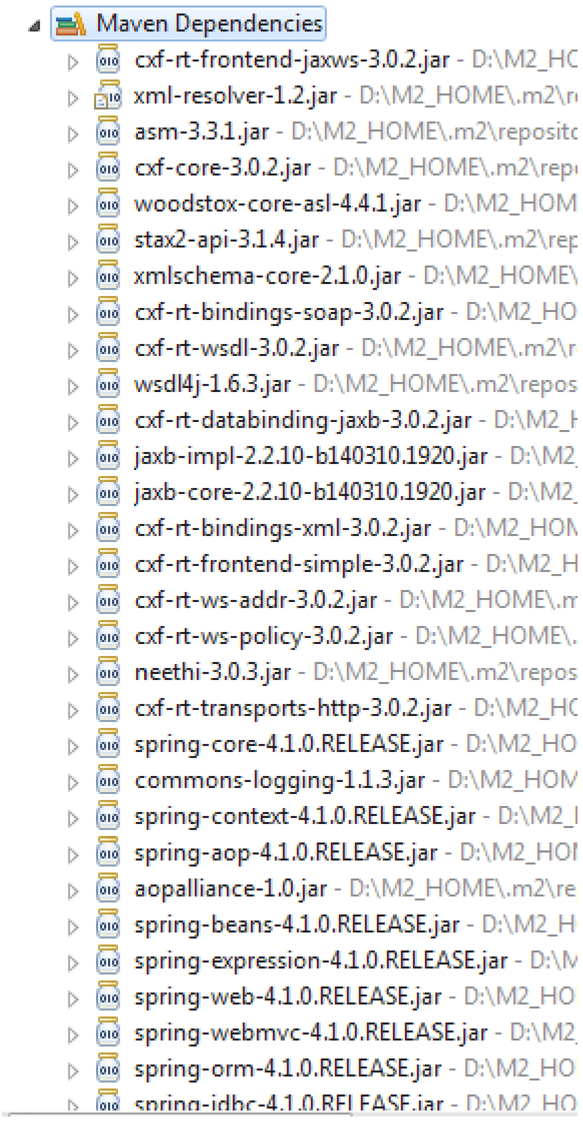
- src/test/java –> test related files, mostly JUnit test cases
- src/main/java –> create java source files under this folder
- generated/java/source –> generated java source files are placed here
- src/main/resources –> all configuration files placed here
- src/test/resources –> all test related configuration files placed here
- Maven Dependencies or Referenced Libraries –> includes jars in the classpath
- WEB-INF under webapp –> stores web.xml & other configuration files related to web application

Project Structure (Package Explorer view in Eclipse)



```
ApacheCXF-JAX-WS-Top-Down-Spring-Hibernate
  src/test/java
    test.jaxws.series.td.spring.hibernate.service
      TestMusicService.java
  src/main/java
    com.jaxws.series.td.spring.hibernate.dao
      MusicDAO.java
      MusicDAOImpl.java
    com.jaxws.series.td.spring.hibernate.model
      Music.java
    com.jaxws.series.td.spring.hibernate.service
      MusicServiceImpl.java
  generated/java/source
  src/main/resources
  src/test/resources
  Maven Dependencies
  JRE System Library [jdk1.7.0_06]
  generated
  src
    main
      webapp
        META-INF
        WEB-INF
          apache-cxf-services.xml
          web.xml
        index.html
    test
  target
  music.db
  pom.xml
```

Jar Libraries Used in the Project (Maven Dependencies)



Maven Dependencies
cxf-rt-frontend-jaxws-3.0.2.jar - D:\M2_HC
xml-resolver-1.2.jar - D:\M2_HOME\.m2\r
asm-3.3.1.jar - D:\M2_HOME\.m2\reposit
cxf-core-3.0.2.jar - D:\M2_HOME\.m2\rep
woodstox-core-asl-4.4.1.jar - D:\M2_HOM
stax2-api-3.1.4.jar - D:\M2_HOME\.m2\rep
xmlschema-core-2.1.0.jar - D:\M2_HOME\
cxf-rt-bindings-soap-3.0.2.jar - D:\M2_HO
cxf-rt-wsdl-3.0.2.jar - D:\M2_HOME\.m2\r
wsdl4j-1.6.3.jar - D:\M2_HOME\.m2\repos
cxf-rt-databinding-jaxb-3.0.2.jar - D:\M2_H
jaxb-impl-2.2.10-b140310.1920.jar - D:\M2
jaxb-core-2.2.10-b140310.1920.jar - D:\M2
cxf-rt-bindings-xml-3.0.2.jar - D:\M2_HON
cxf-rt-frontend-simple-3.0.2.jar - D:\M2_H
cxf-rt-ws-addr-3.0.2.jar - D:\M2_HOME\.m
cxf-rt-ws-policy-3.0.2.jar - D:\M2_HOME\.
neethi-3.0.3.jar - D:\M2_HOME\.m2\repos
cxf-rt-transports-http-3.0.2.jar - D:\M2_HC
spring-core-4.1.0.RELEASE.jar - D:\M2_HO
commons-logging-1.1.3.jar - D:\M2_HOM
spring-context-4.1.0.RELEASE.jar - D:\M2_I
spring-aop-4.1.0.RELEASE.jar - D:\M2_HOI
aopalliance-1.0.jar - D:\M2_HOME\.m2\re
spring-beans-4.1.0.RELEASE.jar - D:\M2_H
spring-expression-4.1.0.RELEASE.jar - D:\M
spring-web-4.1.0.RELEASE.jar - D:\M2_HO
spring-webmvc-4.1.0.RELEASE.jar - D:\M2
spring-orm-4.1.0.RELEASE.jar - D:\M2_HO
spring-jdbc-4.1.0.RELEASE.jar - D:\M2_HO

Database scripts

Database: Creating table and inserting few records for this example

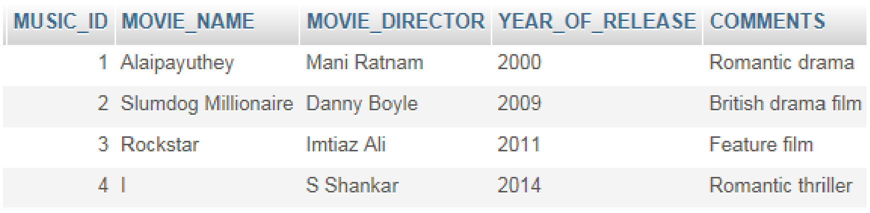
Create Table command

```
CREATE TABLE `MUSIC` (
  `MUSIC_ID` INT(6) NOT NULL AUTO_INCREMENT,
  `MOVIE_NAME` VARCHAR(50) NOT NULL,
  `MOVIE_DIRECTOR` VARCHAR(50) NOT NULL,
  `YEAR_OF_RELEASE` VARCHAR(50) NOT NULL,
  `COMMENTS` VARCHAR(50),
  PRIMARY KEY (`MUSIC_ID`)
);
```

Insert command (examples)

```
INSERT INTO `music`(`MOVIE_NAME`,
`MOVIE_DIRECTOR`, `YEAR_OF_RELEASE`,
`COMMENTS`) VALUES ('Alaipayuthey','Mani
Ratnam','2000','Romantic drama');
INSERT INTO `music`(`MOVIE_NAME`,
`MOVIE_DIRECTOR`, `YEAR_OF_RELEASE`,
`COMMENTS`) VALUES ('Slumdog
Millionaire','Danny Boyle','2009','British
drama film');
INSERT INTO `music`(`MOVIE_NAME`,
`MOVIE_DIRECTOR`, `YEAR_OF_RELEASE`,
`COMMENTS`) VALUES ('Rockstar','Imtiaz
Ali','2011','Feature film');
INSERT INTO `music`(`MOVIE_NAME`,
`MOVIE_DIRECTOR`, `YEAR_OF_RELEASE`,
`COMMENTS`) VALUES ('I','S
Shankar','2014','Romantic thriller');
```

Select * from music;



MUSIC_ID	MOVIE_NAME	MOVIE_DIRECTOR	YEAR_OF_RELEASE	COMMENTS
1	Alaipayuthey	Mani Ratnam	2000	Romantic drama
2	Slumdog Millionaire	Danny Boyle	2009	British drama film
3	Rockstar	Imtiaz Ali	2011	Feature film
4	I	S Shankar	2014	Romantic thriller

Web application

For any web application, entry point is ***web.xml*** which describes how the incoming http requests are served / processed. Further, it describes about the global-context and local-context param (i.e.; *<context-param>* & *<init-param>*) for loading files particular to project requirements & contains respective listener

With this introduction, we will understand how we configured ***web.xml*** for Apache CXF JAX-WS SOAP based Web Service

web.xml (the entry point –> under WEB-INF)

This ***web.xml*** file describes,

- Like any JEE web framework register "*org.apache.cxf.transport.servlet.CXFServlet*" with servlet container
- http requests with URL pattern ***"/services/*"*** will be sent to the registered servlet called "*CXFServlet*" i.e.; *(org.apache.cxf.transport.servlet.CXFServlet)*
- configure spring context loader listener for loading spring context files "*org.springframework.web.context.ContextLoaderListener*"
- <*context-param*> with its attributes describes the location of the ***"apache-cxf-service.xml"*** & ***"spring-hibernate-jax-ws"*** files from where it has to be loaded. We will discuss briefly about these files
- configure session timeout in secs using <*session-config*> tag
- <*welcome-file-list*> files under this tag is the start-up page

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sur
    xmlns:xsi="http://www.w3.org/2001/XMLSche
    xsi:schemaLocation="http://java.sun.com/x

    <display-name>ApacheCXF-JAX-WS-Top-Down-S

    <!-- listener to startup (spring) -->
    <listener>
        <listener-class>org.springframework.w
    </listener>

    <!-- loading spring context file from cla
    <context-param>
        <param-name>contextConfigLocation</pa
        <param-value>
            \WEB-INF\apache-cxf-services.xml,
            classpath:com\jaxws\series\td\spr
        </param-value>
    </context-param>

    <!-- Apache CXF servlet -->
    <servlet>
        <servlet-name>CXFServlet</servlet-nam
        <servlet-class>org.apache.cxf.transpo
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>CXFServlet</servlet-nam
        <url-pattern>/services/*</url-patterr
    </servlet-mapping>

    <!-- session timeout -->
    <session-config>
        <session-timeout>60</session-timeout>
    </session-config>

    <!-- welcome file list -->
    <welcome-file-list>
        <welcome-file>index.html</welcome-fil
    </welcome-file-list>

</web-app>
```

Apache CXF services

Apache CXF comes with spring based configuration, so it is easy to register beans in the spring container much like we do any bean in spring application in addition to configuring JAX-WS endpoints and interceptors, etc

In CXF endpoint, we can define implementor i.e.; Java endpoint implementation class and address. So, incoming requests from "*CXFServlet*" servlet invokes corresponding implementation class with configured address-pattern

For more JAX-WS element details see here

This ***apache-cxf-services.xml*** describes,

- < *jaxws:endpoint />* defines which service implementation class to be invoked for the incoming http/https SOAP requests with address-pattern configured

NOTE: For two different beans we can have two different url-pattern (address) like

```
<jaxws:endpoint id="bookservice"
        implementor="com.jaxws.series.top.down
.approach.service.BookServiceImpl"
        address="/book">
</jaxws:endpoint>

<jaxws:endpoint id="otherservice"

implementor="other.qualified.package.name"
        address="/other">
</jaxws:endpoint>
```

apache-cxf-services.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/
    xmlns:xsi="http://www.w3.org/2001/XMLSche
    xmlns:jaxrs="http://cxf.apache.org/jaxrs"
    xmlns:util="http://www.springframework.or
    xsi:schemaLocation="http://www.springfram
    http://cxf.apache.org/jaxrs http://cxf.ap
    http://www.springframework.org/schema/cor
    http://www.springframework.org/schema/uti

    <jaxws:endpoint id="musicservice"
        implementor="com.jaxws.series.td.spri
        address="/music">
    </jaxws:endpoint>

</beans>
```

Spring Application Context file

This Spring Application Context file describes,

- *<context:annotation-config />* to activate annotation on the registered beans with application context
- *<context:component-scan base-package="" />* tag scans all classes & sub-classes under the value of base-package attribute and register them with the Spring container
- bean with id="*transactionManager*" to inform spring to take care of the database transaction. All methods annotated with @Transactional
- *<tx:annotation-driven />* to turn ON transaction annotation on all DAO methods
- bean with id="*sessionFactory*" defines hibernate properties to let it take care of database operations using hibernate's rich API
- bean with id="*dataSource*" defines values for *driverClassName*, *url*, *username* and *password* for MySql database
- **Note:** injection series between *transactionManager*, *sessionFactory* and *dataSource*

spring-hibernate-jax-ws.xml (src/main/resources/com/jaxws/series/td/spring/hibernate/config)

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/
    xmlns:xsi="http://www.w3.org/2001/XMLSche
    xmlns:tx="http://www.springframework.org/
    xsi:schemaLocation="http://www.springfram
    http://www.springframework.org/schema/cor
    http://www.springframework.org/schema/tx

    <!-- to activate annotations in beans alr
    <context:annotation-config />

    <!-- scans packages to find and register
    <context:component-scan base-package="com

    <!-- turn on spring transaction annotatic
    <tx:annotation-driven transaction-manager

    <!-- Transaction Manager -->
    <bean id="transactionManager"
        class="org.springframework.orm.hiberr
        <property name="sessionFactory" ref="
    </bean>

    <!-- Session Factory -->
    <bean id="sessionFactory"
        class="org.springframework.orm.hiberr
        <property name="dataSource" ref="data
        <property name="annotatedClasses">
            <list>
                <value>com.jaxws.series.td.sp
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect"
                <prop key="hibernate.hbm2ddl.
                <prop key="hibernate.show_sql
            </props>
        </property>
    </bean>

    <!-- dataSource configuration -->
    <bean id="dataSource"
        class="org.springframework.jdbc.datas
        <property name="driverClassName" valu
        <property name="url" value="jdbc:mysc
        <property name="username" value="root
        <property name="password" value="" />
    </bean>

</beans>
```

Let's see coding in action

URL Pattern

Http url for any common web application is http://<server>:<port>/<root-context>/<from_here_application_specific_path>

In our example, we are going to deploy the war into Tomcat 8.0 server, so our server and port are *localhost* and *8080* respectively. Context root is the project name i.e.; ApacheCXF-JAX-WS-Top-Down-Spring-Hibernate. Initial path for this application is http://localhost:8080/ApacheCXF-JAX-WS-Top-Down-Spring-Hibernate

We have configured ***"/services/*"*** as url-pattern for the "*CXFServlet*" servlet in web.xml and our business implementation class implements the portType interface generated from WSDL file which is annotated with *@WebService* annotation at class-level

Model Class (POJO)

Model class Music with five primitive attributes with their getter/setter

Also Hibernate POJO class is annotated describing the mapping between java property and database columns

@**Entity**: represents an object that can be persisted in the database and for this class should have **no-arg** constructor
@**Table:** describes which table in the database to map with this class properties
@**Id:** defines this is unique which means it represents primary key in the database table
@**GeneratedValue:** this will be taken care by hibernate to define generator sequence
@**Column:** tells to map this particular property to table column in the database
For example, "musicId" property used to map "MUSIC_ID" column in the "MUSIC" table in the database
@Column(name= "MUSIC_ID")
private int musicId;

Note: we can add attributes to the @Column annotation like name, length, nullable and unique

Music.java

```
package com.jaxws.series.td.spring.hibernat

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "MUSIC")
public class Music {

    // member variables
    @Id
    @GeneratedValue
    @Column(name = "MUSIC_ID")
    private int musicId;

    @Column(name= "MOVIE_NAME")
    private String movieName;

    @Column(name= "MOVIE_DIRECTOR")
    private String director;

    @Column(name= "YEAR_OF_RELEASE")
    private String yearOfRelease;

    @Column(name= "COMMENTS")
    private String comments;

    // getters & setters
    public int getMusicId() {
        return musicId;
    }

    public void setMusicId(int musicId) {
        this.musicId = musicId;
    }

    public String getMovieName() {
        return movieName;
    }

    public void setMovieName(String movieName
        this.movieName = movieName;
    }

    public String getDirector() {
        return director;
    }

    public void setDirector(String director)
        this.director = director;
    }

    public String getYearOfRelease() {
        return yearOfRelease;
    }

    public void setYearOfRelease(String yearO
        this.yearOfRelease = yearOfRelease;
    }

    public String getComments() {
        return comments;
    }

    public void setComments(String comments)
        this.comments = comments;
    }
}
```

Music Service Implementation (business logic)

This service provider class implements portType interface generated from WSDL file. Also, class annotated with *@WebService* annotation at class-level and this is very important

Note: This class extends SpringBeanAutowiringSupport class to support annotation

MusicServiceImpl.java

```
package com.jaxws.series.td.spring.hibernat?.

import in.benchresources.entities.music.Busir
import in.benchresources.entities.music.Movie
import in.benchresources.entities.music.Movie
import in.benchresources.entities.music.Music
import in.benchresources.entities.music.Music
import in.benchresources.services.musicservic
import in.benchresources.services.musicservic

import java.util.List;

import javax.jws.WebService;

import org.springframework.beans.factory.annc
import org.springframework.stereotype.Service
import org.springframework.web.context.suppor

import com.jaxws.series.td.spring.hibernate.c
import com.jaxws.series.td.spring.hibernate.m

@WebService(serviceName="MusicService", endpc
targetNamespace="http://benchresources.in/ser
public class MusicServiceImpl extends SpringE

    @Autowired
    private MusicDAO musicDAO;

    @Override
    public MusicListResponseType getAllMovieD

        // local variables
        List<Music> lstMusic = null;
        MovieType movieType = null;
        MovieListType movieListType = null;
        MusicListResponseType musicListRespor
        BusinessFaultType businessFaultType =

        try{
            if(null != parameters && !paramet

                // invoke dao to get values
                lstMusic = musicDAO.getAllMov

                // create musicListType to se
                musicListResponseType = new M
                musicListResponseType.setComp

                // create movieListType and a
                movieListType = new MovieList

                // iterate through lstMusic a
                for(Music music : lstMusic){

                    // set values retrieved f
                    movieType = new MovieType
                    movieType.setMovieName(mu
                    movieType.setDirector(mus
                    movieType.setYear(music.g
                    movieType.setComments(mus
                    movieListType.getMovieTyp
                }

                // finally set movieListType
                musicListResponseType.setMovi
            }
        }
        catch(Exception ex){
            // dummy setting for business exc
            businessFaultType = new BusinessF
            businessFaultType.setErrorCode(16
            businessFaultType.setErrorMessage
            businessFaultType.setErrorDescrip
        }
        finally{
            // close resources, if any
        }
        return musicListResponseType;
    }
}
```

DAO layer

This DAO layer takes care of the database interaction i.e.; uses Hibernate's rich API to interact with MySql database using MySqlDialect

MusicDAO.java

```
package com.jaxws.series.td.spring.hibernat

import java.util.List;

import com.jaxws.series.td.spring.hibernate.m

public interface MusicDAO {

    public List<Music> getAllMoviesByComposer
}
```

MusicDAOImpl.java

```
package com.jaxws.series.td.spring.hibernat

import java.util.List;

import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annc
import org.springframework.stereotype.Reposit
import org.springframework.transaction.annota

import com.jaxws.series.td.spring.hibernate.m

@Repository("musicDAO")
public class MusicDAOImpl implements MusicDAC

    public static final String MUSIC_COMPOSEF

    @Autowired
    private SessionFactory sessionFactory;

    @SuppressWarnings("unchecked")
    @Override
    @Transactional(value="transactionManager"
    public List<Music> getAllMoviesByComposer

        // local variables
        List<Music> lstMusic = null;

        if(null != composerName && composerNa

            // get all books info from databa
            lstMusic = sessionFactory.getCurr
        }
        return lstMusic;
    }
}
```

That's all with coding part, now let us move on to deployment and testing

Apache Tomcat-8.0.15 Deployment

- Run maven command to build the war: ***mvn clean install*** (use command prompt or integrated maven in eclipse IDE)
- Copy(ctrl+c) the war file from the target folder
- Paste(ctrl+v) it into apache tomcat (webapps folder)
- Start the tomcat server (Tomcat_Home\bin\startup.bat)

Oracle Weblogic server 12.1.1.0 Deployment

Apache CXF based JAX-WS web service can't be deployed directly into Oracle WebLogic server as WAR file –> it's a two step process. First build a WAR file and then package this WAR file into an EAR file –> Deploy EAR file into weblogic server

See this article for explanation: Packaging WAR as EAR

Steps to be followed

- Run maven command to build the war: ***mvn clean install*** (use command prompt or integrated maven in eclipse IDE)
- Once you see "BUILD SUCCESS" after running maven command, it means your war file is successfully built and installed in the local maven repository
- Package this WAR file into an EAR file as explained in this article
- Start weblogic 12c application server and hit the URL http://localhost:7001/console in any of the latest web

browser and enter username/password you configured while setting up weblogic 12c server
- Go to Deployments –> click install button –> browse through EAR file location –> say Next –> say Next –> Finish

For Oracle WebLogic 12c server Installation steps see here

Test the service !!

Testing

There are many ways to do testing

- SOAP UI Client
- Java Client using JDK's in-built classes like *HttpURLConnection*
- Java Client using SOAP API
- Eclipse's Web Services Explorer
- Write your own client for example, Java client using *httpcomponents* from Apache

We will cover first 2 ways of testing above JAX-WS deployed service

1. SOAP UI Client

Load the endpoint URL in SOAP UI Client, which will pre-populate the request XML based on the operation deployed/exposed using WSDL

For example, http://localhost:8080/ApacheCXF-JAX-WS-Top-Down-Spring-Hibernate/services/music/MusicService?wsdl

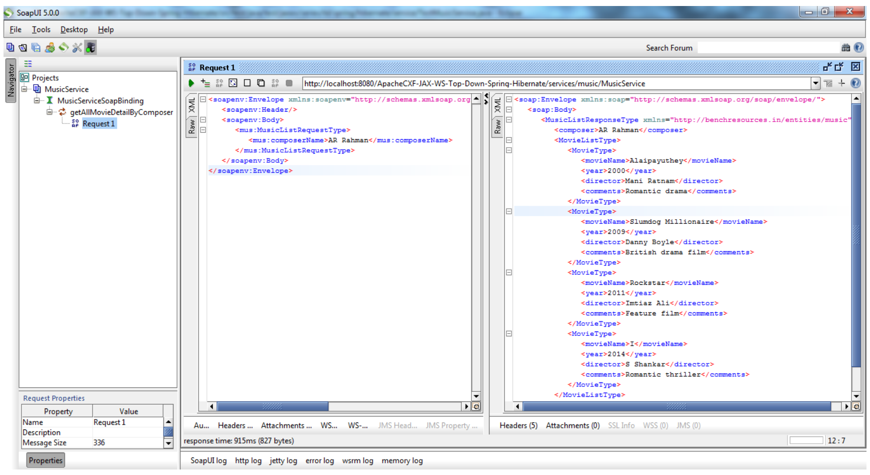
Request XML

```
<soapenv:Envelope xmlns:soapenv="http://sche
    xmlns:mus="http://benchresources.in/entiti
    <soapenv:Header />
    <soapenv:Body>
        <mus:MusicListRequestType>
            <mus:composerName>AR Rahman</mus:c
        </mus:MusicListRequestType>
    </soapenv:Body>
</soapenv:Envelope>
```

Response XML

```
<soap:Envelope xmlns:soap="http://schemas.x
    <soap:Body>
```

```
        <MusicListResponseType xmlns="http://
            <composer>AR Rahman</composer>
            <MovieListType>
                <MovieType>
                    <movieName>Alaipayuthey</
                    <year>2000</year>
                    <director>Mani Ratnam</di
                    <comments>Romantic drama<
                </MovieType>
                <MovieType>
                    <movieName>Slumdog Millic
                    <year>2009</year>
                    <director>Danny Boyle</di
                    <comments>British drama f
                </MovieType>
                <MovieType>
                    <movieName>Rockstar</movi
                    <year>2011</year>
                    <director>Imtiaz Ali</dir
                    <comments>Feature film</c
                </MovieType>
                <MovieType>
                    <movieName>I</movieName>
                    <year>2014</year>
                    <director>S Shankar</dire
                    <comments>Romantic thrill
                </MovieType>
            </MovieListType>
        </MusicListResponseType>
    </soap:Body>
</soap:Envelope>
```


SoapUI 5.0.0
File Tools Desktop Help
Search Forum
Navigator
Projects
MusicService
MusicServiceSoapBinding
getAllMovieDetailByComposer
Request 1
XML
Raw
Au... Headers ... Attachments ... WS... WS-... JMS Head... JMS Property ...
Headers (5) Attachments (0) SSL Info WSS (0) JMS (0)
12 : 7
Properties
SoapUI log http log jetty log error log wsrm log memory log

http://localhost:8080/ApacheCXF-JAX-WS-Top-Down-Spring-Hibernate/services/music/MusicService

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org
   <soapenv:Header/>
   <soapenv:Body>
      <mus:MusicListRequestType>
         <mus:composerName>AR Rahman</mus:composerName>
      </mus:MusicListRequestType>
   </soapenv:Body>
</soapenv:Envelope>
```

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <MusicListResponseType xmlns="http://benchresources.in/entities/music"
         <composer>AR Rahman</composer>
         <MovieListType>
            <MovieType>
               <movieName>Alaipayuthey</movieName>
               <year>2000</year>
               <director>Mani Ratnam</director>
               <comments>Romantic drama</comments>
            </MovieType>
            <MovieType>
               <movieName>Slumdog Millionaire</movieName>
               <year>2009</year>
               <director>Danny Boyle</director>
               <comments>British drama film</comments>
            </MovieType>
            <MovieType>
               <movieName>Rockstar</movieName>
               <year>2011</year>
               <director>Imtiaz Ali</director>
               <comments>Feature film</comments>
            </MovieType>
            <MovieType>
               <movieName>I</movieName>
               <year>2014</year>
               <director>S Shankar</director>
               <comments>Romantic thriller</comments>
            </MovieType>
         </MovieListType>
```

Property	Value
Name	Request 1
Description	
Message Size	336

response time: 915ms (827 bytes)

2. Java client

For this java client to work/execute, we don't need to add any extra jars or any new dependency in pom.xml as these classes comes shipped along with JDK. Observe, import statements closely for this client

Note: Request XML pattern formed taking help from pre-populated request XML from SOAP UI client as explained above

TestMusicService.java

```
package test.jaxws.series.td.spring.hibern

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.URL;

public class TestMusicService {

    /**
     * JAX-WS top-down web service approach
     * main() method to test/start soap web
     * @param args
     * @throws IOException
     */
    public static void main(String[] args) t

        String httpRequestURL = "http://loca
        String soapRequestParam =   "<soaper
                +                   "<sc
                +                   "<sc
                +
                +
                +
                +                   "</s
                +               "</soape
        String responseString = testBookServ
        System.out.println("Response String
    }

    /**
     * This method uses HttpURLConnection tc
     *
     * @param httpRequestURL
     * @param requestXmlParam
     * @return responseXML
     * @throws IOException
     */
    public static String testBookService(Str

        // local variables
        URL url = null;
        HttpURLConnection httpURLConnection
        OutputStreamWriter outputStreamWrite
        String responseMessageFromServer = r
        String responseXML = null;

        try {
            // set basic request parameters
            url = new URL(httpRequestURL);
            httpURLConnection = (HttpURLConr
            httpURLConnection.setDoOutput(tr
            httpURLConnection.setRequestMeth
            //          httpURLConnection.se
            httpURLConnection.setRequestProp
            httpURLConnection.setRequestProp

            // write request XML to the HTTF
            outputStreamWriter = new OutputS
            outputStreamWriter.write(request
            outputStreamWriter.flush();

            System.out.println("Response coc
            if (httpURLConnection.getRespons

                responseMessageFromServer = 
                System.out.println("Response
                responseXML = getResponseXML
            }
        }
        catch (IOException ioex) {
            ioex.printStackTrace();
            throw ioex;
        }
        finally{
            // finally close all operations
            outputStreamWriter.close();
            httpURLConnection.disconnect();
        }
        return responseXML;
    }

    /**
     * This method is used to get response X
     *
     * @param httpURLConnection
     * @return stringBuffer.toString()
     * @throws IOException
     */
    private static String getResponseXML(Htt

        // local variables
        StringBuffer stringBuffer = new Stri
        BufferedReader bufferedReader = null
        InputStreamReader inputStreamReader 
        String readSingleLine = null;

        try{
            // read the response stream AND 
            inputStreamReader = new InputStr
            bufferedReader = new BufferedRea

            // reading the XML response cont
            while ((readSingleLine = buffere
                stringBuffer.append(readSing
            }
        }
        catch (IOException ioex) {
            ioex.printStackTrace();
            throw ioex;
        }
        finally{
            // finally close all operations
            bufferedReader.close();
            httpURLConnection.disconnect();
        }
        return stringBuffer.toString();
    }
}
```

Output in console

```
Response code: 200
ResponseMessageFromServer: OK
Response String :
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/en
```

```
velope/">
        <soap:Body>
                <MusicListResponseType
xmlns="http://benchresources.in/entities/music
">
                        <composer>AR
Rahman</composer>
                        <MovieListType>
                                <MovieType>

<movieName>Alaipayuthey</movieName>

<year>2000</year>

<director>Mani Ratnam</director>

<comments>Romantic drama</comments>
                                </MovieType>
                                <MovieType>

<movieName>Slumdog Millionaire</movieName>

<year>2009</year>

<director>Danny Boyle</director>

<comments>British drama film</comments>
                                </MovieType>
                                <MovieType>

<movieName>Rockstar</movieName>

<year>2011</year>

<director>Imtiaz Ali</director>

<comments>Feature film</comments>
                                </MovieType>
                                <MovieType>

<movieName>I</movieName>

<year>2014</year>

<director>S Shankar</director>

<comments>Romantic thriller</comments>
```

```
                                        </MovieType>
                                </MovieListType>
                        </MusicListResponseType>
                </soap:Body>
        </soap:Envelope>
```

Conclusion: Thus, we have implemented & understood SOAP based Web Service implementation using top-down approach integrating with Spring and Hibernate ORM framework

Download project

```
ApacheCXF-JAX-WS-Top-Down-Spring-Hibernate
(19kB)
```

Happy Coding !!
Happy Learning !!

« Oracle WebLogic server 12c + Apache-CXF JAX-WS + Packaging WAR as EAR

Apache CXF JAX-WS: SOAP based Web Service using Top-Down approach + Integrating with Spring framework »

Related Posts:

1. Apache CXF JAX-WS: SOAP based Web Service using Top-Down approach + Integrating with Spring framework
2. Apache CXF JAX-WS: SOAP based Web Service using Top-Down approach
3. Apache CXF JAX-WS: SOAP based Web Service using Bottom-Up approach
4. Apache CXF JAX-WS: Web Service using Top-Down approach + Adding WS-Security policy using UsernameToken profile

APACHE CXF HIBERNATE JAVA

JAVA WEB SERVICES JAX-WS JAXB SOAP

SOAP WEB SERVICES SPRING WEB SERVICES

5 Comments **BenchResources.Net** 1 **Login**

Recommend   Share   Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

nikki • 5 months ago
can you please give the example of creating soap client using cxf spring in which i am getting input as json formate from frontend side and take this input in restcontroller and pass to the soap client and there i call provider from those input.
• Reply • Share ›

Rajnish • 2 years ago
best ever Till now
• Reply • Share ›

SJ → Rajnish • 2 years ago
Thanks Rajnish
• Reply • Share ›

Manoj Dhanji • 2 years ago
By far the most detailed exmaple. Thanks!
• Reply • Share ›

SJ → Manoj Dhanji • 2 years ago
Thanks Manoj
• Reply • Share ›

ALSO ON **BENCHRESOURCES.NET**

JDBC: An example to connect MS Access database in Java 8
17 comments • a year ago
Avatar Daroga Jee — then tell us.. Where to go and connect to MS-Excel..

Exception Hierarchy in Java
1 comment • a year ago
Avatar Md.Ruhul Amin (Ruhul) — Thank you sir. Very good resource and explanation

Java features version-wise
2 comments • 8 months ago
Avatar Waseem Siddiqi — Thanks

Apache CXF JAX-WS: Web Service using Top-Down
1 comment • a year ago
basanta hota — Any one can