Spring 4 + SOAP Web Service Producer with Tomcat

This page will provide spring 4 and SOAP web service integration to produce and consume soap request and response using annotation. Spring web service uses contract-first SOAP service which produces flexible web services that can use different manipulated XML payloads. Find some basic feature of spring web service.

1. Spring web service provides loose coupling between contract and implementation.
2. It provides powerful mapping between incoming XML request and any object.
3. Spring web service uses JAXP APIs to handle incoming XML messages.
4. It provides flexible XML marshaling using JAXB 1 and 2, Castor, XMLBeans, JiBX, and XStream.
5. Spring Web service security allows signing SOAP messages, encrypting and decrypting them.

Software Used to Run Example

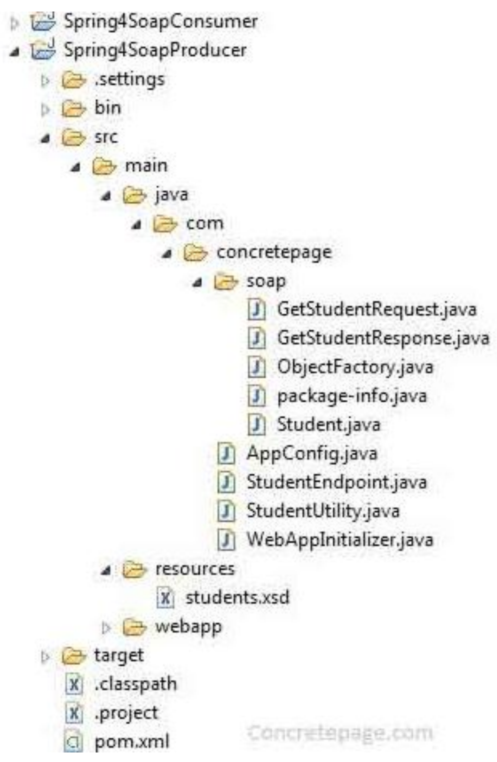
To run the example we are using below software.
1. Java 7
2. Tomcat 8
3. Eclipse
4. Maven

Produce SOAP Web Service

We will create an application that will act as web service for student data. For the given student id, our web service will respond the complete profile of student.

Project Configuration in Eclipse for SOAP Web Service Producer

Find the project configuration in eclipse.



```
Spring4SoapConsumer
Spring4SoapProducer
  .settings
  bin
  src
    main
      java
        com
          concretepage
            soap
              GetStudentRequest.java
              GetStudentResponse.java
              ObjectFactory.java
              package-info.java
              Student.java
            AppConfig.java
            StudentEndpoint.java
            StudentUtility.java
            WebAppInitializer.java
      resources
        students.xsd
      webapp
  target
  .classpath
  .project
  pom.xml
```

Sample XSD file used in Example

Find the sample XSD file being used in the example. On the basis of this file, maven will generate java classes.

students.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://yorkchen.com/soap"
           targetNamespace="http://yorkchen.com/soap" elementFormDefault="qualified">
    <xs:element name="getStudentRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="studentId" type="xs:int"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="getStudentResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="student" type="tns:student"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="student">
        <xs:sequence>
            <xs:element name="studentId" type="xs:int"/>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="age" type="xs:int"/>
            <xs:element name="class" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Run the Maven to Resolve JAR Dependency and Generated Classes for XSD file

Maven will resolve the JAR dependency and will produce java classes for the XSD file by its plugin.

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
 <modelVersion>4.0.0</modelVersion>
  <groupId>com.yorkchen.app</groupId>
  <artifactId>spring4soap</artifactId>
  <version>1</version>
  <packaging>war</packaging>
  <name>Spring 4 Soap</name>
  <dependencies>
         <dependency>
             <groupId>org.springframework.boot</groupId>
             <artifactId>spring-boot-starter-ws</artifactId>
             <version>1.2.0.RELEASE</version>
         </dependency>
         <dependency>
             <groupId>wsdl4j</groupId>
```

```
            <artifactId>wsdl4j</artifactId>
            <version>1.6.1</version>
        </dependency>
        <dependency>
            <groupId>javax.xml.bind</groupId>
            <artifactId>jaxb-api</artifactId>
            <version>2.2.12</version>
        </dependency>
  </dependencies>
  <build>
          <plugins>
                   <plugin>
                     <groupId>org.codehaus.mojo</groupId>
                     <artifactId>jaxb2-maven-plugin</artifactId>
                     <version>1.6</version>
                     <executions>
                         <execution>
                             <id>xjc</id>
                             <goals>
                                 <goal>xjc</goal>
                             </goals>
                         </execution>
                     </executions>
                     <configuration>
                         <schemaDirectory>${project.basedir}/src/main/resources/</schemaDirectory>
                         <outputDirectory>${project.basedir}/src/main/java</outputDirectory>
                         <clearOutputDir>false</clearOutputDir>
                     </configuration>
                    </plugin>
           </plugins>
   </build>
</project>
```

The plugin will generate java classes based on students.xsd located at classpath. The generated classes will be as below.
1. GetStudentRequest.java
2. GetStudentResponse.java
3. ObjectFactory.java
4. package-info.java
5. Student.java

Sample Data for Output

For the demo, we are populating student profile in a map and providing a method to return student profile for the given student id.

StudentUtility.java

```
package com.yorkchen;
import java.util.HashMap;
import java.util.Map;
import org.springframework.stereotype.Component;
import com.yorkchen.soap.Student;
```

```
@Component
public class StudentUtility {
         private Map<Integer,Student> studentMap = new HashMap<Integer,Student>();
         public StudentUtility(){
                  Student s1 = new Student();
                  s1.setStudentId(1);
                  s1.setName("Ram");
                  s1.setAge(20);
                  s1.setClazz("ABC");
                  studentMap.put(1, s1);
                  Student s2 = new Student();
                  s2.setStudentId(2);
                  s2.setName("Shyam");
                  s2.setAge(22);
                  s2.setClazz("EFG");
                  studentMap.put(2, s2);
         }
         public Student getStudent(int studentId){
                  return studentMap.get(studentId);
         }
}
```

Configuration Class: Use WsConfigurerAdapter , @EnableWs, DefaultWsdl11Definition, XsdSchema

The configuration class should be annotated with @EnableWs that will provide web service configuration. We need to override WsConfigurerAdapter to get required WsConfigurer methods. *The bean name for DefaultWsdl11Definition is must because this bean name will be the part of WSDL URL as students.wsdl*.

@EnableWs : Provides spring web service configuration.
WsConfigurerAdapter : This class is an adapter class that contains only required methods of WsConfigurer.
DefaultWsdl11Definition : Creates SOAP for the given XSD schema.
XsdSchema : Abstraction for XSD schema.

Now find the configuration class.
AppConfig.java

```
package com.yorkchen;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.ClassPathResource;
import org.springframework.ws.config.annotation.EnableWs;
import org.springframework.ws.config.annotation.WsConfigurerAdapter;
import org.springframework.ws.wsdl.wsdl11.DefaultWsdl11Definition;
import org.springframework.xml.xsd.SimpleXsdSchema;
import org.springframework.xml.xsd.XsdSchema;
@Configuration
@EnableWs
@ComponentScan("com.yorkchen")
```

```
public class AppConfig extends WsConfigurerAdapter {
        @Bean(name = "students")
        public DefaultWsdl11Definition defaultWsdl11Definition(XsdSchema studentsSchema) {
                DefaultWsdl11Definition wsdl11Definition = new DefaultWsdl11Definition();
                wsdl11Definition.setPortTypeName("StudentsPort");
                wsdl11Definition.setLocationUri("/soapws");
                wsdl11Definition.setTargetNamespace("http://yorkchen.com/soap");
                wsdl11Definition.setSchema(studentsSchema);
                return wsdl11Definition;
        }
        @Bean
        public XsdSchema studentsSchema() {
                return new SimpleXsdSchema(new ClassPathResource("students.xsd"));
        }
}
```

WebApplicationInitializer using MessageDispatcherServlet

MessageDispatcherServlet is a servlet that is specific to dispatch web service message. We need to use this class in place of DispatcherServlet for web service. The method *setTransformSchemaLocations* can be true or false. If true, MessageDispatcherServlet transforms relative address location in XSD to incoming request.
WebAppInitializer.java

```
package com.yorkchen;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRegistration.Dynamic;
import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.ws.transport.http.MessageDispatcherServlet;
public class WebAppInitializer implements WebApplicationInitializer {
         public void onStartup(ServletContext servletContext) throws ServletException {
        AnnotationConfigWebApplicationContext ctx = new AnnotationConfigWebApplicationContext();
        ctx.register(AppConfig.class);
        ctx.setServletContext(servletContext);
        MessageDispatcherServlet servlet = new MessageDispatcherServlet();
         servlet.setApplicationContext(ctx);
         servlet.setTransformWsdlLocations(true);
        Dynamic dynamic = servletContext.addServlet("dispatcher",servlet);
        dynamic.addMapping("/soapws/*");
        dynamic.setLoadOnStartup(1);
   }
}
```

Endpoint for SOAP Web Service using Annotation

Find the endpoint class for the web service. This classe uses different annotations as below.

@Endpoint: Indicates an annotated class for web service end point.
@PayloadRoot: Marks the method to be handler for incoming request.

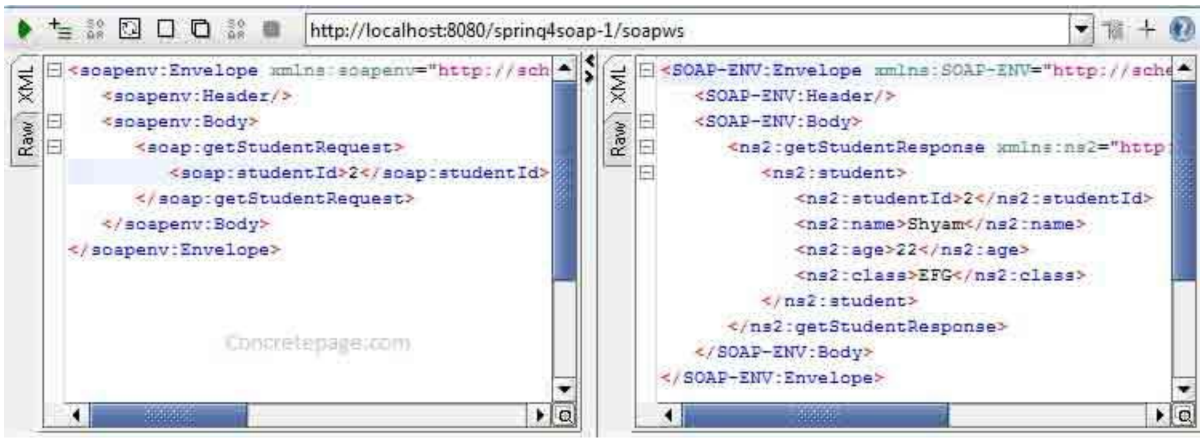
@ResponsePayload: Bounds the method return type to the response payload.
@RequestPayload : Bounds the method parameter to the request payload.

Find the web service endpoint class.
StudentEndpoint.java

```
package com.yorkchen;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.ws.server.endpoint.annotation.Endpoint;
import org.springframework.ws.server.endpoint.annotation.PayloadRoot;
import org.springframework.ws.server.endpoint.annotation.RequestPayload;
import org.springframework.ws.server.endpoint.annotation.ResponsePayload;
import com.yorkchen.soap.GetStudentRequest;
import com.yorkchen.soap.GetStudentResponse;
@Endpoint
public class StudentEndpoint {
         private static final String NAMESPACE_URI = "http://yorkchen.com/soap";
         @Autowired
         private StudentUtility studentUtility;
         @PayloadRoot(namespace = NAMESPACE_URI, localPart = "getStudentRequest")
         @ResponsePayload
         public GetStudentResponse getCountry(@RequestPayload GetStudentRequest request) {
                  GetStudentResponse response = new GetStudentResponse();
                  response.setStudent(studentUtility.getStudent(request.getStudentId()));
                  return response;
         }
}
```

Test SOAP Web Service using SoapUI

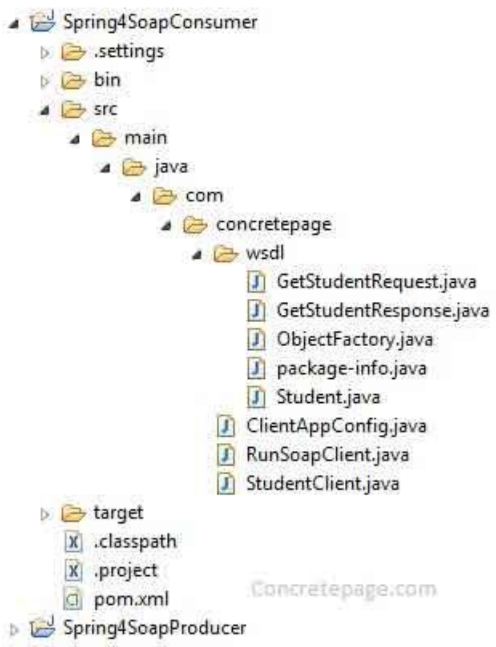
Deploy the WAR file in tomcat 8 generated by Maven. The WSDL URL will be
http://localhost:8080/spring4soap-1/soapws/students.wsdl
We can access it in browser directly to get WSDL response. To test in SOAP UI, use a request soap in which we need to provide student id and in response, we will get student profile.



```
<soapenv:Envelope xmlns:soapenv="http://sch
   <soapenv:Header/>
   <soapenv:Body>
      <soap:getStudentRequest>
         <soap:studentId>2</soap:studentId>
      </soap:getStudentRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://sche
   <SOAP-ENV:Header/>
   <SOAP-ENV:Body>
      <ns2:getStudentResponse xmlns:ns2="http
         <ns2:student>
            <ns2:studentId>2</ns2:studentId>
            <ns2:name>Shyam</ns2:name>
            <ns2:age>22</ns2:age>
            <ns2:class>EFG</ns2:class>
         </ns2:student>
      </ns2:getStudentResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Project Structure in Eclipse for SOAP Web Service Consumer
Find the project structure in eclipse for SOAP Web Service Consumer.



```
Spring4SoapConsumer
  .settings
  bin
  src
    main
      java
        com
          concretepage
            wsdl
              GetStudentRequest.java
              GetStudentResponse.java
              ObjectFactory.java
              package-info.java
              Student.java
            ClientAppConfig.java
            RunSoapClient.java
            StudentClient.java
  target
  .classpath
  .project
  pom.xml
Spring4SoapProducer
```

Maven to Resolve JAR Dependency and Generate Classes for WSDL URL
Find pom.xml that will resolve the JAR dependency and will generate java classes for the given WSDL URL.
pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
 <modelVersion>4.0.0</modelVersion>
  <groupId>com.yorkchen.app</groupId>
  <artifactId>spring4soap</artifactId>
  <version>1</version>
  <packaging>jar</packaging>
  <name>Spring 4 Soap Client</name>
  <dependencies>
         <dependency>
                  <groupId>org.springframework.boot</groupId>
                  <artifactId>spring-boot-starter-ws</artifactId>
                  <version>1.2.0.RELEASE</version>
         </dependency>
         <dependency>
                  <groupId>javax.xml.bind</groupId>
                  <artifactId>jaxb-api</artifactId>
                  <version>2.2.12</version>
         </dependency>
  </dependencies>
  <build>
      <plugins>
```

```
                  <plugin>
                      <groupId>org.jvnet.jaxb2.maven2</groupId>
                      <artifactId>maven-jaxb2-plugin</artifactId>
                      <executions>
                          <execution>
                              <goals>
                                  <goal>generate</goal>
                              </goals>
                          </execution>
                      </executions>
                      <configuration>
                          <schemaLanguage>WSDL</schemaLanguage>
                          <generatePackage>com.yorkchen.wsdl</generatePackage>
                          <forceRegenerate>true</forceRegenerate>
                          <schemas>
                              <schema>
                                  <url>http://localhost:8080/spring4soap-1/soapws/students.wsdl</url>
                              </schema>
                          </schemas>
                      </configuration>
                  </plugin>
           </plugins>
   </build>
</project>
```

The plugin will generate java classes for the given WSDL URL. The classes will be generated as below.
1. GetStudentRequest.java
2. GetStudentResponse.java
3. ObjectFactory.java
4. package-info.java
5. Student.java

Create Client Class Using WebServiceGatewaySupport

WebServiceGatewaySupport is a super class that facilitates the sub class to access web service methods. This class has its own WebServiceTemplate that can be fetched by the method *getWebServiceTemplate()*
StudentClient.java

```
package com.yorkchen;
import org.springframework.ws.client.core.support.WebServiceGatewaySupport;
import org.springframework.ws.soap.client.core.SoapActionCallback;
import com.yorkchen.wsdl.GetStudentRequest;
import com.yorkchen.wsdl.GetStudentResponse;
public class StudentClient extends WebServiceGatewaySupport  {
         public GetStudentResponse getStudentById(int studentId) {
                  GetStudentRequest request = new GetStudentRequest();
                  request.setStudentId(studentId);
                  GetStudentResponse response = (GetStudentResponse)
getWebServiceTemplate().marshalSendAndReceive(
                                    request, new SoapActionCallback("http://localhost:8080/spring4soap-
1/soapws/getStudentResponse"));
```

```
                  return response;
         }
}
```

Configuration Class: Use Jaxb2Marshaller

Jaxb2Marshaller is the implementation of GenericMarshaller interface to support JAXB. It is being used here to set context path using *setContextPath* .

ClientAppConfig.java

```
package com.yorkchen;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.oxm.jaxb.Jaxb2Marshaller;
@Configuration
public class ClientAppConfig {
         @Bean
         public Jaxb2Marshaller marshaller() {
                  Jaxb2Marshaller marshaller = new Jaxb2Marshaller();
                  marshaller.setContextPath("com.yorkchen.wsdl");
                  return marshaller;
         }
         @Bean
         public StudentClient studentClient(Jaxb2Marshaller marshaller) {
                  StudentClient client = new StudentClient();
                  client.setDefaultUri("http://localhost:8080/spring4soap-1/soapws/students.wsdl");
                  client.setMarshaller(marshaller);
                  client.setUnmarshaller(marshaller);
                  return client;
         }
}
```

Main Method to Run the Demo

Now test the consuming web service. Here we are passing student id and web service will return student profile.

RunSoapClient.java

```
package com.yorkchen;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import com.yorkchen.wsdl.GetStudentResponse;
public class RunSoapClient {
         public static void main(String[] args) {
                  AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();
                 ctx.register(ClientAppConfig.class);
                 ctx.refresh();
                  StudentClient studentClient = ctx.getBean(StudentClient.class);
                  System.out.println("For Student Id: 1");
                  GetStudentResponse response = studentClient.getStudentById(1);
                  System.out.println("Name:"+response.getStudent().getName());
                  System.out.println("Age:"+response.getStudent().getAge());
```

```
                    System.out.println("Class:"+response.getStudent().getClazz());
        }
}
```

ind the output.

```
For Student Id: 1
Name:Ram
Age:20
Class:ABC
```