Develop webservice with springboot

SOAP, Simple Object Access Protocol, is a communication protocol based on XML to realize data exchange in the network.
A SOAP message is an ordinary XML document that contains the following elements:
1. Required Envelope element to identify this XML document as a SOAP message
2. Required Body element, containing all call and response information
3. An optional Header element containing header information
4. An optional Fault element that provides information about the error that occurred while processing this message
Grammar rules:
1. SOAP messages must be encoded in XML
2. SOAP messages must use the SOAP Envelope namespace
3. SOAP messages must use the SOAP Encoding namespace
4. SOAP messages cannot contain DTD references
5. SOAP messages cannot contain XML processing instructions

1. Environment

Apache maven

IntelliJ idea

2. Add dependency in pom.xml

```xml
<!--web service-->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web-services</artifactId>
      <version>2.6.7</version>
    </dependency>

    <!--wsdl, used for webservice-->
    <dependency>
      <groupId>wsdl4j</groupId>
      <artifactId>wsdl4j</artifactId>
    </dependency>
```

3.Add plugin in pom.xml

```xml
<!--Convert between Java classes and XML Schema
      sources: xsd directory
      outputDirectory: java class directory
      packageName: package directory
      clearOutputDir: if clear current directory when re-produce
      -->
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>jaxb2-maven-plugin</artifactId>
        <version>2.5.0</version>
        <executions>
          <execution>
            <id>xjc</id>
            <goals>
              <goal>xjc</goal>
            </goals>
          </execution>
        </executions>
        <configuration>
          <sources>
            <source>src/main/resources/xsd</source>
          </sources>
          <outputDirectory>src/main/java</outputDirectory>
          <packageName>com.felix.shoppingcentre.soap.producer.generated</packageName>
          <clearOutputDir>false</clearOutputDir>
        </configuration>
      </plugin>
```

4.Create an XML Schema to define the domain model and add the user.xsd file to the src/main/resources/xsd directory

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:tns="http://tutorial.spring.boot/soap/produce/user"
           targetNamespace="http://tutorial.spring.boot/soap/produce/user"
           elementFormDefault="qualified">
    <xs:complexType name="User">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="birth" type="xs:date"/>
            <xs:element name="gender" type="tns:Gender"/>
        </xs:sequence>
    </xs:complexType>

    <xs:simpleType name="Gender">
        <xs:restriction base="xs:string">
            <xs:enumeration value="Male"/>
            <xs:enumeration value="Female"/>
            <xs:enumeration value="Unknown"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:element name="UserRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="UserResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="user" type="tns:User"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

5. Execute mvn compile in the root directory of the project

```
PS F:\springworkplace\shoppingcentre> mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< com.felix:shoppingcentre >----------------------
[INFO] Building shoppingcentre 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ shoppingcentre ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 25 source files to F:\springworkplace\shoppingcentre\target\classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.602 s
[INFO] Finished at: 2022-05-09T18:09:50-04:00
[INFO] ------------------------------------------------------------------------
```

6.Create webservice configuration class

```
package com.felix.shoppingcentre.soap.producer.config;

import org.springframework.boot.web.servlet.ServletRegistrationBean;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.ClassPathResource;
import org.springframework.ws.config.annotation.EnableWs;
import org.springframework.ws.transport.http.MessageDispatcherServlet;
import org.springframework.ws.wsdl.wsdl11.DefaultWsdl11Definition;
import org.springframework.ws.wsdl.wsdl11.Wsdl11Definition;
```

```
import org.springframework.xml.xsd.SimpleXsdSchema;
import org.springframework.xml.xsd.XsdSchema;

/**
 * Spring WS uses MessageDispatcherServlet to process SOAP messages,
 * so creating a Web Service configuration requires creating a
 * new instance of MessageDispatcherServlet and injecting the application context
 * ApplicationContext into the instance.
 * The MessageDispatcherServlet instance named messageDispatcherServlet
 * does not replace the Spring Boot default DispatcherServlet bean.
 * DefaultWsdl11Definition exposes standard WSDL 1.1 using XSD Schema.
 * Note: The MessageDispatcherServlet and DefaultWsdl11Definition instances
 *  must be given names,
 * which determine the WSDL URL. In this example, the MessageDispatcherServlet
 * instance name is ws,
 * and the DefaultWsdl11Definition instance name is user,
 * so the WSDL URL is http://<host>:<port>/ws/user.wsdl.
 */
@EnableWs
@Configuration
public class WebServiceConfig {

    @Bean
    public ServletRegistrationBean messageDispatcherServlet(ApplicationContext applicationContext) {
        MessageDispatcherServlet servlet = new MessageDispatcherServlet();
        servlet.setApplicationContext(applicationContext);
        servlet.setTransformWsdlLocations(true);
        return new ServletRegistrationBean(servlet, "/ws/*");
    }

    @Bean(name = "user")
    public Wsdl11Definition defaultWsdl11Definition(XsdSchema schema) {
        DefaultWsdl11Definition wsdl11Definition = new DefaultWsdl11Definition();
        wsdl11Definition.setPortTypeName("UserPort");
        wsdl11Definition.setLocationUri("/ws");
        wsdl11Definition.setTargetNamespace("http://tutorial.spring.boot/soap/produce/user");
        wsdl11Definition.setSchema(schema);
        return wsdl11Definition;
    }

    @Bean
    public XsdSchema userSchema() {
        return new SimpleXsdSchema(new ClassPathResource("xsd/user.xsd"));
    }
}
```

7. Create a service endpoint: define a POJO with @Endpoint annotation class to handle incoming SOAP requests

```
package com.felix.shoppingcentre.soap.producer.controller;

import com.felix.shoppingcentre.soap.producer.generated.Gender;
import com.felix.shoppingcentre.soap.producer.generated.User;
import com.felix.shoppingcentre.soap.producer.generated.UserRequest;
import com.felix.shoppingcentre.soap.producer.generated.UserResponse;
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 javax.xml.datatype.DatatypeConfigurationException;
import javax.xml.datatype.DatatypeFactory;
import java.time.LocalDate;

/**
 * @Endpoint
 * Annotated classes will be registered with Spring WS for processing incoming SOAP messages.
 * @PayloadRoot
 * Spring WS uses this annotation to find processing methods that match the namespace and localPart of the message.
 * @RequestPayload
 * Identifies which parameter of the method the incoming message will be mapped to.
 * @ResponsePayload
 * This annotation identifies how Spring WS maps method return values to response payloads.
 */
@Endpoint
public class UserWebServiceController {
    private static final String NAMESPACE_URI = "http://tutorial.spring.boot/soap/produce/user";
```

```java
        @PayloadRoot(namespace = NAMESPACE_URI, localPart = "UserRequest")
        @ResponsePayload
        public UserResponse getUser(@RequestPayload UserRequest request) throws DatatypeConfigurationException {
            UserResponse response = new UserResponse();
            User user = new User();
            String name = request.getName();
            user.setName(name);
            switch (name) {
                case "Mike":
                    user.setBirth(
                            DatatypeFactory.newInstance().newXMLGregorianCalendar(
                                    LocalDate.of(2000, 1, 1).toString()
                            )
                    );
                    user.setGender(Gender.MALE);
                    break;
                case "Ketty":
                    user.setBirth(
                            DatatypeFactory.newInstance().newXMLGregorianCalendar(
                                    LocalDate.of(2010, 12, 31).toString()
                            )
                    );
                    user.setGender(Gender.FEMALE);
                    break;
                default:
                    user.setGender(Gender.UNKNOWN);
                    break;
            }
            response.setUser(user);
            return response;
        }
}
```

9. After launching the application, use a browser to access http://127.0.0.1:8080/ws/user.wsdl

```xml
This XML file does not appear to have any style information associated with it. The document tree is shown below.
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:sch="http://tutorial.spring.boot/soap/produce/user" xmlns:soap="http:
<wsdl:types>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" targetNamespace="http://tutorial.spring.boot/soap/pro
<xs:complexType name="User">
<xs:sequence>
<xs:element name="name" type="xs:string"/>
<xs:element name="birth" type="xs:date"/>
<xs:element name="gender" type="tns:Gender"/>
</xs:sequence>
</xs:complexType>
<xs:simpleType name="Gender">
<xs:restriction base="xs:string">
<xs:enumeration value="Male"/>
<xs:enumeration value="Female"/>
<xs:enumeration value="Unknown"/>
</xs:restriction>
</xs:simpleType>
<xs:element name="UserRequest">
<xs:complexType>
<xs:sequence>
<xs:element name="name" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="UserResponse">
<xs:complexType>
<xs:sequence>
<xs:element name="user" type="tns:User"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
</wsdl:types>
<wsdl:message name="UserRequest">
<wsdl:part element="tns:UserRequest" name="UserRequest"> </wsdl:part>
</wsdl:message>
<wsdl:message name="UserResponse">
<wsdl:part element="tns:UserResponse" name="UserResponse"> </wsdl:part>
</wsdl:message>
<wsdl:portType name="UserPort">
```

```
<wsdl:operation name="User">
<wsdl:input message="tns:UserRequest" name="UserRequest"> </wsdl:input>
<wsdl:output message="tns:UserResponse" name="UserResponse"> </wsdl:output>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="UserPortSoap11" type="tns:UserPort">
<soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="User">
<soap:operation soapAction=""/>
<wsdl:input name="UserRequest">
<soap:body use="literal"/>
</wsdl:input>
<wsdl:output name="UserResponse">
<soap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="UserPortService">
<wsdl:port binding="tns:UserPortSoap11" name="UserPortSoap11">
<soap:address location="http://127.0.0.1:8080/shoppingcenter/ws"/>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

10. Sum up

```
Summarize
Steps to develop a SOAP service with Spring Boot:

Create a Spring Boot application, add spring-boot-starter-web-services and wsdl4j dependencies and jaxb2-maven-plugin;
create xsd file;
Execute mvn compile to generate Java classes from xsd files;
Create a Web Service configuration class;
Create a business service class;
Start the application, view the wsdl description file through the browser, and execute the test.
```

11.Test webservice

- 11.1 Create a request.xml (I create the file in F disk) `UserRequest` and name are defined in user.xsd

```
xmlns:gs="http://tutorial.spring.boot/soap/produce/user">
    <soapenv:Header/>
    <soapenv:Body>
       <gs:UserRequest>
          <gs:name>Ketty</gs:name>
       </gs:UserRequest>
    </soapenv:Body>
</soapenv:Envelope>
```

- 11.2 Use ms-cmd execute:

```
F:\>curl --header "content-type: text/xml" -d @request.xml http://127.0.0.1:8080/shoppingcenter/ws/user.wsdl
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"><SOAP-ENV:Header/><SOAP-ENV:Body><ns2:UserResponse xmlns:ns2=
```
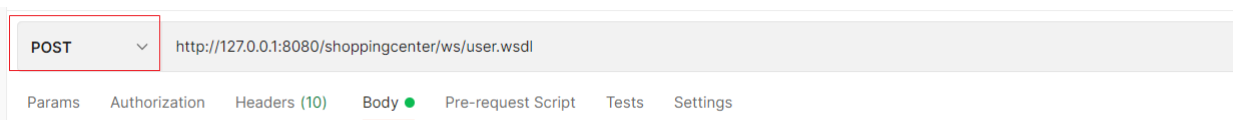
11.3 if use postman test webservice

11.3.1 Choose Post method in dropdown list

POST ∨	http://127.0.0.1:8080/shoppingcenter/ws/user.wsdl

Params   Authorization   Headers (10)   Body ●   Pre-request Script   Tests   Settings

11.3.2 Add Content-Type, cancel default Content-Tpye

```
Open the request Headers. If the auto-generated headers are hidden, select the notice to display them.
Deselect the Content-Type header Postman added automatically.
Add a new row with Content-Type in the Key field and text/xml in the Value field.
```


11.3.3

1. In the **Body** tab, select **raw** and choose **XML** from the dropdown list.

2. Enter your XML in the text entry area. Attention：xmlns:soap is a fixed value, don't change it to another value just in case