Vežbe 10 JAX-WS

Web servisi predstavljaju programe koji su dostupni putem javno objavljenih interfejsa i putem standardnih komunikacionih protokola. Pod web servisima najčešće podrazumevamo programe:

- dostupne putem SOAP (Simple Object Access Protocol) protokola
- sa interfejsom za pristup opisanim pomodu WSDL (Web Service Description Language) jezika
- (potencijalno) registrovani u UDDI (Universal Description, Discovery and Integration) servisu

WSDL je XML gramatika za opisivanje web servisa kao skupa krajnjih pristupnih tačaka. WSDL opisuje: šta servis radi, kako pozvati njegove operacije i gde ga pronadi.

Kod web servisa su svi elementi arhitekture, uključujudi i komunikacioni protokol zasnovani na XML-u. Ovo obezbeđuje vedu interoperabilnost među različitim implementacijama, međutim zbog stalne konverzije podataka iz/u XML format nisu pogodni u slučaju kada su performanse komunikacije od značaja.

JAX-WS predstavlja tehnologiju za izgradnju web servisa i klijenata koji komuniciraju putem XML-a. Operacije web servisa se pozivaju pomodu XML-baziranih protokola kao što je SOAP. Sa JAX-WS ne moraju se direktno generisati i parsirati SOAP poruke, time se bavi JAX-WS runtime engine (automatska konverzija API poziva/odgovara iz/u SOAP poruke). U prikazanim primerima se koristi Apache CXF koji implementira JAX-WS API.

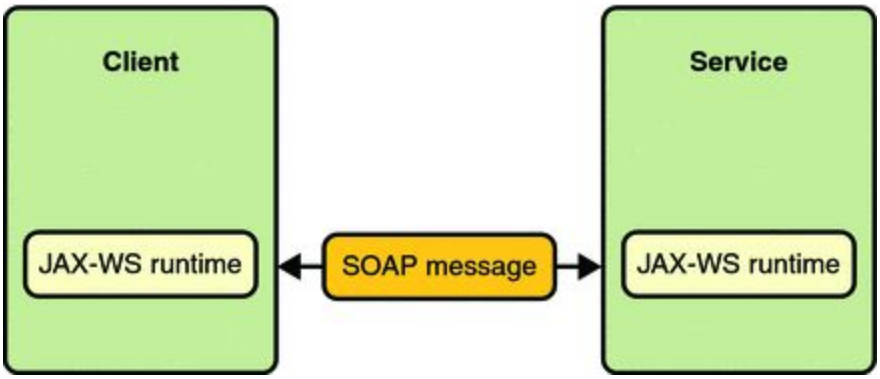


Figura 1. Komunikacija između JAX-WS Service i Client-a

1. Primer - SOAP JAX-WS (Example 1)

U ovom primeru počinjemo od pisanja Java klasa koje implementiraju web servis. Na osnovu napisanih klasa se generiše odgovarajući WSDL fajl. U primeru su dva servisa napravljeni - servis Hello i AddressBook.

Postupak:

1. Piše se endpoint interfejs (SEI) koji definiše metode koje klijent može pozvati na web servisu i JAX-WS endpoint java klasa koja ga implementira

U datom primeru, interfejsi su **Hello** i **AddressBook**, a njihove implementacije **HelloImpl** i **AddressBookImpl**.

Parametri i rezultati web metoda implementiranih web servisa označavaju se JAXB anotacijama kako bi se omogudilo mapiranje podataka izmedju objektnog i XML modela, bududi da se pozivi metoda web servisa i njihovi odgovori prenose kao SOAP poruke (XML fajlovi).

2. Postavlja se adresa u **application.properties** na **cxf.path=/ws/**
3. U pom.xml je podešen **cxf-java2ws-plugin** za maven generisanje tokom "mvn package" ciklusa, wsdl fajlovi su napravljeni unutar **target/generated/wsdl/**

   Rezultati ovog koraka predstavljaju izgenerisani WSDL fajlovi AddressBook.wsdl i Hello.wsdl koji se nalaze u okviru target/generated/wsdl direktorijuma. Takođe se može dodati generisanje standalone client klase. Default-na konfiguracija, pored putanje, je predstavljena ispod:

```
<configuration>
   <className>...</className>
   <classpath>...</classpath>
   <outputFile>...</outputFile>
   <genWsdl>true</genWsdl>
   <genServer>false</genServer>
   <genClient>false</genClient>
   <genWrapperbean>false</genWrapperbean>
   <frontend>jaxws</frontend>
   <databinding>jaxb</databinding>
   <serviceName>...</serviceName>
   <soap12>false</soap12>
   <targetNameSpace>...</targetNameSpace>
   <verbose>false</verbose>
   <quiet>false</quiet>
   <attachWsdl>true</attachWsdl>
   <address>...</address>
</configuration>
```

4. Za svaki WebService koji će postati WSDL fajl se definiše poseban <execution> sa istom maven phase-om, kao i odgovarajuće konfiguracije iz 3. tačke.

5. Servis je potrebno registrovati kao Endpoint unutar **EndpointConfig** koja indikuje više @Bean klasa pomoću @Configuration iz Spring Framework-a. Svaki Endpoint iz JAX-WS se vraća nakon publikovanja instance implementacije.

```
@Configuration
public class EndpointConfig {

    @Autowired
    private Bus bus;

    @Bean
    public Endpoint helloEndpoint() {
        EndpointImpl endpoint = new EndpointImpl(bus, new HelloPortImpl());
        endpoint.publish("/helloMessage");
        return endpoint;
    }
}
```

6. Pokrenuti Spring boot aplikaciju SoapApplication > run as java application.

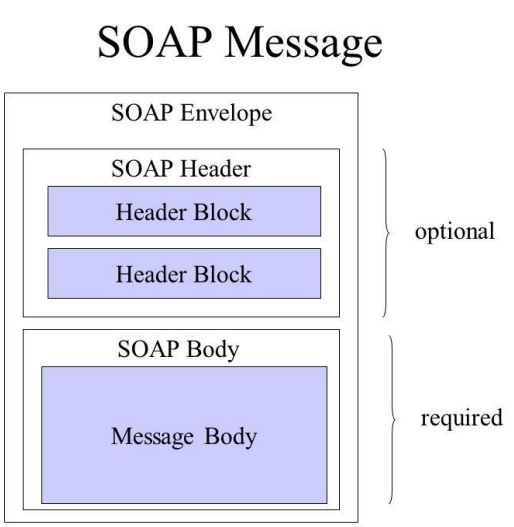
   Otići na http://localhost:8080/ws/, da se vidi sve servisi koji su dostupni na adresi.

   Sam WSDL fajl se nalazi na http://localhost:8080/ws/helloMessage?wsdl

7. Testirati SOAP komunikaciju sa primerima client aplikacije u okviru com.spring.soap.ws.client paketa ili pomoću postman aplikacije.

2. Primer - SOAP JAX-WS (Example 2)

SOAP predstavlja protokol za komunikaciju sa web servisima - definiše format poruka koje razmenjuju učesnici u komunikaciji. Oslanja se na neki transportni mehanizam za prenos SOAP poruka - najdešde je to HTTP POST metoda.



Struktura SOAP poruke (XML dokument): Envelope - korenski element, Header - opcioni podelement (podaci za autentifikaciju, pradenje poruke, ...), Body - sadrži konkretan SOAP zahtev/odgovor, fault - opcioni podelement sa podacima o nastaloj grešci

@SOAPBinding anotacija definiše kako je web servis vezan za SOAP messaging protocol. Binding može biti:

- RPC (Remote Procedure Calls) - podržava se sintaksa i semantika pozivanje funkcija/metoda.
- DOCUMENT (message passing, document-style) - - komunikacija između klijenta i servera se odvija pomodu slanja (strukturiranih) poruka.

Use - specificira kako se podaci SOAP poruke stream-uju. Može biti:

- Literal - delovi poruke se enkodiraju pomoću XML šeme koju referencijraju
- Encoded - delovi poruke se enkodiraju prema pravilima navedenim u encodingStyle atributu (da li se tip parametara enkodira ili ne)

U primeru su prikazane varijante u dizajnu web servisa:

1. **RPC/literal** (paket com.spring.soap.ws.style.rpc)

```
@WebMethod(operationName="sayHello")
public String sayHello(@WebParam(name="person") Person person);
```

Request:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <ns1:sayHello xmlns:ns1="http://soap.spring.com/ws/style/rpc">
            <person>
                <firstName>Mitar</firstName>
                <lastName>Miric</lastName>
            </person>
```

```
        </ns1:sayHello>
    </soap:Body>
</soap:Envelope>
```

Response:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header />
    <soap:Body>
        <ns1:sayHelloResponse xmlns:ns1="http://soap.spring.com/ws/style/rpc">
            <return>Hello world Mitar Miric</return>
        </ns1:sayHelloResponse>
    </soap:Body>
</soap:Envelope>
```

```
@WebMethod(operationName="sayHelloSeparated")
public String sayHello(@WebParam(name="firstName") String firstName,
@WebParam(name="lastName") String lastName);
```

Request:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <ns1:sayHelloSeparated xmlns:ns1="http://soap.spring.com/ws/style/rpc">
            <firstName>Mitar</firstName>
            <lastName>Miric</lastName>
        </ns1:sayHelloSeparated>
    </soap:Body>
</soap:Envelope>
```

Redosled podelemenata je isti kao redosled parametara metode

Response:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header />
    <soap:Body>
        <ns1:sayHelloSeparatedResponse
xmlns:ns1="http://soap.spring.com/ws/style/rpc">
            <return>Hello world Mitar Miric</return>
        </ns1:sayHelloSeparatedResponse>
    </soap:Body>
</soap:Envelope>
```

2. RPC/encoded

Ne postoji primer u projektu, ali primer poruke bi bio:

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x xsi:type="xsd:int">5</x>
            <y xsi:type="xsd:float">5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

3. **Document/literal/bare** (paket com.spring.soap.ws.style.doc.bare)

Ako se ništa ne navede u @SOAPBinding anotaciji, stil će po defaultu biti document/literal. parameterStyle specificira kako se parametri metoda postavljaju u telu SOAP poruke:

- **bare** - parametri se postavlja u telo poruke kao child elementi korena poruke

- **Wrapped -** svi parametri su povezani u jedisven element u request poruci, a to važi i za izlazne parametre u response parametre. Kod RPC stila parameterStyle mora da bude wrapped (svi parametri su vezani u element koji se zove po imenu metode)

```
public String sayHello(@WebParam(name="person") Person person);
```

Request:

```
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <ns2:person
            xmlns:ns2="http://soap.spring.com/ws/style/doc/bare">
            <firstName>Mitar</firstName>
            <lastName>Miric</lastName>
        </ns2:person>
    </soap:Body>
</soap:Envelope>
```

Response:

```
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header/>
    <soap:Body>
        <ns2:sayHelloResponse
            xmlns:ns2="http://soap.spring.com/ws/style/doc/bare">
Hello world Mitar Miric
        </ns2:sayHelloResponse>
    </soap:Body>
</soap:Envelope>
```

4. **Document/literal/wrapped** (paket com.spring.soap.ws.style.doc.wrap)

Bududi da wrapped stil ne dozvoljava dva razdvojena parametra u metodama, CXF generiše i klase SayHello i SayHelloResponse → ove klase su definisane pomoću **@RequestWrapper** anotacije

```
@RequestWrapper(localName = "sayHello", targetNamespace =
"http://soap.spring.com/ws/style/doc/wrap", className =
"com.spring.soap.ws.style.doc.wrap.SayHello")
@ResponseWrapper(localName = "sayHelloResponse", targetNamespace =
"http://soap.spring.com/ws/style/doc/wrap", className =
"com.spring.soap.ws.style.doc.wrap.SayHelloResponse")
public String sayHello(@WebParam(name = "firstName")String firstName,
                       @WebParam(name = "lastName")String lastName);
```

Request:

```
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <ns2:sayHello
            xmlns:ns2="http://soap.spring.com/ws/style/doc/wrap">
            <firstName>Mitar</firstName>
            <lastName>Miric</lastName>
        </ns2:sayHello>
    </soap:Body>
</soap:Envelope>
```

Response:

```
<soap:Envelope
```

```
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header/>
    <soap:Body>
        <ns2:sayHelloResponse
            xmlns:ns2="http://soap.spring.com/ws/style/doc/wrap">
            <return>Hello world Mitar Miric</return>
        </ns2:sayHelloResponse>
    </soap:Body>
</soap:Envelope>
```

Razlike u WSDLu:

RPC:

```
<message name="myMethodRequest">
<part name="x" type="xsd:int"/>
<part name="y" type ="xsd:float"/>
</message>
<message name="empty"/>
```

Part je tip

part je element

Document/literal/bare:

```
<types>
<schema>
<element name="xElement"
type="xsd:int"/>
<element name="yElement"
type="xsd:float"/>
</schema>
</types>
<message name="myMethodRequest">
<part name="x" element="xElement"/>
<part name="y" element="yElement"/>
</message>
```

Moguće više part elementa u message elementu

Samo jedan part element u message elementu Part element ima ime operacije i kompleksnog je tipa - podelementi predstavljaju parametre operacije

Document/literal/wrapped:

```
<types>
<schema>
<element name="myMethod">
<complexType>
<sequence>
<element name="x"
type="xsd:int"/>
<element name="y"
type="xsd:float"/>
</sequence>
</complexType>
</element>
<element
name="myMethodResponse">
<complexType/>
</element>
</schema>
</types>
<message
name="myMethodRequest">
<part name="parameters"
element="myMethod"/>
</message>
<message name="empty">
<part name="parameters"
element="myMethodResponse"/>
</message>
<portType name="PT">
<operation name="myMethod">
<input
message="myMethodRequest"/>
<output message="empty"/>
</operation>
</portType>
```

5. **Message stil**

Kod message stila se direktno prima i šalje XML, tj. radi se direktno sa SOAP porukama (nema prebacivanja u Java objekte niti JAXB mapiranja). U primeru se koristi DOM za generisanje request i response poruke.

Umesto **@WebService** anotacije stavlja se **@WebServiceProvider** anotacija.

Endpoint klasa implementira Provider<T> interfejs koji ima samo jednu invoke() metodu (kod @WebService anotacije imali smo više različitih metoda za svaku vrstu poruke koju je mogude primiti).

invoke() metoda prima zahteve zapakovane u tip objekta definisan tipom Provider interfejsa i vrada odgovore zapakovane u taj isti tip objekta.

3. Primer - SOAP JAX-WS (Example 3)

U ovom primeru počinjemo od pisanja WSDL fajla, na osnovu koga CXF generiše Java klase koje implementiraju servis opisan u WSDL-u. Ovaj način pristupa je **Contact First Web Services**!

1. Piše se WSDL fajl koji opisuje web servis. U datom primeru, data su dva WSDLa: **HelloDocument** & **AddressBook**.
2. Postavlja se adresa u application.properties na cxf.path=/ws/
3. U **pom.xml** se podešava **wsdl2java** tako da prilikom pokretanja "mvn generate-sources -P generate" (moguće je aktivirati profil pomoću maven > activate profile unutar eclipse-a) generišu java klase iz wsdl fajlova**.** Generisane klase se nalaze u **target/generated/cxf/com...**

   Generisanje na ovan način podrazumeva i generisanje WebService-a, WebService implementacione klase, po potrebi i client zavisno od podešavanja maven plugin-a.
4. Svaki WSDL fajl će predstavljati jedan **wsdlOption** unutar **wsdlOptions** tag-a.
5. Svaka <Ime>**PortImpl** klasa je samo **template** i nije dovršena implementacija. Logiku vaših web servisa ćete implementirati u okviru generisanih metoda.
6. Pokrenuti Spring boot aplikaciju SoapApplication > run as java application.

   Otići na http://localhost:8080/ws/, da se vidi sve servisi koji su dostupni na adresi.

   Sam WSDL fajl se nalazi na http://localhost:8080/ws/hello?wsdl

7. Testirati SOAP komunikaciju sa primerima client aplikacije u okviru com.spring.soap.ws.client paketa ili pomoću postman aplikacije.
   U klasi **HelloClient** su prikazana **2 načina poziva web servisa**:
   a. Kao i u prethodnim projektima (kroz port - lokalni objekat koji se ponaša kao proxy koji reprezentuje udaljeni servis) - metoda **testIt1**()
   b. Kroz klasu koju je generisao **CXF** (HelloDocumenService.java) - metoda **testIt2**(). U principu, oba načina su identična, samo što u ovom slučaju **automatski izgenerisana Service klasa predstavlja wrapper** za kod iz slučaja navedenog pod a).

Zadatak

Za XML šemu Racuni.xsd sa prethodnih termina vežbi pomocu JAX-WS tehnologije napraviti traženo. Napraviti web servis za vođenje evidencije o računu koji ima sledeće metode:

- Dodavanje nove stavke u račun
- Metoda koja vraca ceo račun (sve unete stavke)
- Metoda koja prima id stavke računa i iz računa briše stavku sa datim id.

Za poruke koje se razmenjuju sa web servisom koristiti elemente i tipove definisane u shemi Racuni.xsd. Web servis treba da koristi SOAP format poruka i to document/literal/wrapped stila. Za pomenuti servis konstruisati adekvatnog klijenta koji testira (poziva) sve metode servisa.