SOUTH EASTERN EUROPE

JavaCro '13

JAX-WS trikovi

MILJEN MIKIĆ

5.6.2013.

JavaCro@Tuhelj

Danas ćemo pričati o..

- JAX-WS-u (logično ☺)
- Metro frameworku
- nešto malo JAXB-a
- agenda: 10 trikova i savjeta iz „stvarnog svijeta”

Za početak – malo terminologije..

JAX-WS

- Java **API** for XML Web services
- služi stvaranju i korištenju Web servisa na Javi
- v2.0 uključen u JDK 6 od početka, JDK 6u4 donosi v2.1, a v2.2 nalazi se u JDK 7

JAX-WS RI

- referentna implementacija JAX-WS-a
- također uključena u JDK6+, aktualna verzija je v2.2.7

Metro

- JAX-WS RI + WSIT
- Web service framework, aktualna verzija v2.2.1

..i malo *toolkita*

`wsimport`

- alat koji iz WSDL dokumenta generira klijentske *stub*-ove
- korisne opcije: `-b, -p, -catalog`..

`wsgen`

- alat koji na temelju SEI-a generira artifakte (klase) potrebne za postavljanje Web servisa na poslužitelj
- uz opciju `-wsdl` generira i WSDL servisa

Trik #1 – endorsaj me nježno

- problem: imamo JDK sa starijom verzijom JAX-WS-a, a u aplikaciji želimo koristiti novu verziju JAX-WS-a (bez `LinkageError-a` i sličnih manifestacija)
- rješenje: "endorse" – mehanizam u Javi za osvježavanje verzija tehnologija uključenih u JDK
- u `<JDK_HOME>/jre/lib/endorse` staviti nove verzije datoteka: `jax-ws.jar` i `jaxb.jar`
  - alternativno, ako ne želimo dirati JDK, iskopirati navedene datoteke u proizvoljni direktorij i koristiti parametar `-Djava.endorsed.dirs`

Trik #2 – JAX-WS na Tomcatu

- problem: ne želimo koristiti `Endpoint.publish(..)`, već želimo postaviti naš Web servis na aplikacijski poslužitelj
- rješenje: postavljanje Web servisa na dva moguća načina:
  - JSR-109 (enterprise Web services)
  - JSR-224 (JAX-WS)
- Tomcat nije JSR-109-*compliant*, tako da preostaje samo druga opcija
- dva koraka: ispravna konfiguracija Web aplikacije i instaliranje JAX-WS/Metro datoteka na Tomcat

Trik #2 – JAX-WS na Tomcatu

- HelloJavaCro.java sučelje:

```
package javacro.tuhelj;
..
@WebService
@SOAPBinding(style = Style.DOCUMENT)
public interface HelloJavaCro {
        @WebMethod
        public String hello();
}
```

- HelloJavaCroImpl implementacija:

```
package javacro.tuhelj;
..
@WebService(endpointInterface="javacro.tuhelj.HelloJavaCro")
public class HelloJavaCroImpl implements HelloJavaCro {
        @Override
        public String hello() {
                return „Hello Java people!”;
        }
}
```

Trik #2 – JAX-WS na Tomcatu

- `web.xml` datoteka:

..

```
<listener>
          <listener-class>
                    com.sun.xml.ws.transport.http.servlet.WSServletContextListener
          </listener-class>
</listener>
<servlet>
          <servlet-name>HelloJavaPeople</servlet-name>
          <servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>
          <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
          <servlet-name>HelloJavaPeople</servlet-name>
          <url-pattern>/JavaCro</url-pattern>
</servlet-mapping>
```

Trik #2 – JAX-WS na Tomcatu

- `sun-jaxws.xml` datoteka:

```
..

<endpoints xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime" version="2.0">
          <endpoint name="JavaCroService"
                    implementation="javacro.tuhelj.HelloJavaCroImpl"
                    url-pattern="/JavaCro"/>
</endpoints>
```

- instaliranje potrebnih datoteka na Tomcat (v6+):
  - Metro: u `<CATALINA_HOME>/shared/lib`
  - `webservices-rt.jar`
  - `webservices-tools.jar`
  - `webservices-extra.jar`
  - `webservices-extra-api.jar`;
  - `webservices-api.jar` u `<CATALINA_HOME>/endorsed` (opet endorsanje!)
  - JAX-WS RI: cijeli `/lib` direktorij u `<CATALINA_HOME>/lib`

Trik #2 – JAX-WS na Tomcatu

- malo analize: zašto je uopće potrebno instalirati dodatne datoteke? (JAX-WS RI već dolazi s JDK6+, zar ne?)
- odgovor: `WSServlet` i `WSServletContextListener` naslijeđuju/implementiraju klase i sučelja iz Servlet specifikacije
  - Servlet je dio Java EE!
  - Sun je svojedobno odlučio da ne želi miješati Javu SE i EE, pa su neke klase doslovno izvađene iz JAX-WS RI implementacije koja dolazi s JDK-om!
- bonus trik: od uvođenja Servlet 3.0 specifikacije, `web.xml` više uopće nije potreban! („*dynamic registration feature*”) – Tomcat7+

Trik #3 – nomen est omen

- *code-first* pristup:

```
@WebMethod
public void createPerson(String name, String surname, String address) { .. }
```

- izgenerirani *WSDL* (točnije, *XSD*):

```
<xs:complexType name="createPerson">
<xs:sequence>
<xs:element name="arg1" type="xs:string" minOccurs="0"/>
<xs:element name="arg2" type="xs:string" minOccurs="0"/>
<xs:element name="arg3" type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
```

Trik #3 – nomen est omen

- klijent naravno nema pojma što treba slati u kojem polju
- rješenje: anotacija `@WebParam`!

```
@WebMethod
public void createPerson(
        @WebParam(name="ime") String name,
        @WebParam(name="prezime") String surname,
        @WebParam(name="adresa") String address) {
        ..
}
```

Trik #4 – obavezni parametar

- *code-first* pristup, očekujemo od pozivatelja da popuni sve parametre
- izgenerirani *WSDL* (točnije, *XSD*):

```
<xs:complexType name="createPerson">
<xs:sequence>
<xs:element name="ime" type="xs:string" minOccurs="0"/>
<xs:element name="prezime" type="xs:string" minOccurs="0"/>
<xs:element name="adresa" type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
```

Trik #4 – obavezni parametar

- rješenje: anotacija `@XmlElement(required=true)` nad parametrom, ali u `wsgen`-generiranoj klasi:

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "createPerson")
public class CreatePerson {
        @XmlElement(name = "ime", required=true)
        protected String name;
        @XmlElement(name = "prezime", required=true)
        protected String surname;
        @XmlElement(name = "adresa", required=true)
        protected String address;
        ..
}
```

  - rezultat: izostavi se `minOccurs="0"` iz XSD-a (a po *defaultu* je 1)
  - samo za Metro v2.0+, JAX-WS 2.2+

Trik #5 – korištenje lokalnog WSDL-a

- JAX-WS klijent prije poziva servisa povuče WSDL s originalne lokacije kako bi izvukao dodatne (*metadata*) podatke o servisu, napravio usporedbu s klijentskim klasama itd.

- mogući problemi:
  - nepotrebno trošenje mrežnog prometa
  - nedostupnost originalne lokacije na kojoj je bio WSDL
  - originalni WSDL se promijenio; iako su promjene čuvale kompatibilnost unatrag, do poziva uopće neće doći zbog validacije koja neće proći

- rješenje: spremiti WSDL lokalno, 4 moguća načina dohvata

Trik #5 – korištenje lokalnog WSDL-a

- prvi način - dohvat iz koda:

```
URL url = getClass().getResource("./wsdl/JavaCro.wsdl");
HelloJavaCroImplService serv = new HelloJavaCroImplService(url, new
        QName("http://tuhelj.javacro/", "HelloJavaCroImplService"));
HelloJavaCro stub = serv.getHelloJavaCroImplPort();
System.out.println(stub.hello());
```

- drugi način – dohvat preko `jax-ws-catalog.xml` datoteke:

```
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog" prefer="system">
    <system systemId="http://localhost:8080/JavaCroWS/JavaCro?wsdl" uri="/wsdl/JavaCro.wsdl"/>
    <system systemId="http://localhost:8080/JavaCroWS/JavaCro?xsd=1" uri="/wsdl/JavaCro.xsd"/>
</catalog>
```

  - korištenje: `wsimport -catalog <ime_datoteke>`, to radi NetBeans od verzije 6.7
  - prednost: nikakvi posebni zahvati u kodu nisu potrebni

Trik #5 – korištenje lokalnog WSDL-a

- treći način – korištenje `wsimport`-a uz opciju `-wsdllocation`:

```
wsimport -keep -d build -p javacro.client
http://localhost:8080/JavaCroWS/JavaCro?wsdl
-wsdllocation=../JavaCro.wsdl
```

  - izgenerira se ovakav klijent (kojeg onda treba instancirati i koristiti):

```
@WebServiceClient(name = "HelloJavaCroImplService", targetNamespace =
"http://tuhelj.javacro/", wsdlLocation = "../JavaCro.wsdl")
public class HelloJavaCroImplService extends Service
{..}
```

  - `JavaCro.wsdl` mora biti u direktoriju `build` (relativna putanja!)

Trik #5 – korištenje lokalnog WSDL-a

- četvrti način – korištenje `wsimport`-a uz opciju `-clientjar`:

```
wsimport -clientjar javacroclient.jar
http://localhost:8080/JavaCroWS/JavaCro?wsdl
```

  - svi potrebni artifakti zapakirani u `javacroclient.jar`
  - od JAX-WS v2.2 nadalje

- bonus trik: ako se koristi Websphere, kompatibilnost unatrag može se postići i bez lokalnog spremanja WSDL-a; dovoljno je koristiti JVM parametar `jaxws.ignore.extraWSDLOps=true`

Trik #6 – promjena *endpointa*

- sad kad smo neovisni o lokaciji WSDL-a, željeli bismo napraviti generičkog klijenta koji može ići na različite instance Web servisa
- te instance mogu biti po različitim poslužiteljima = različiti *endpointi*
- opet ćemo vidjeti dva načina kako to napraviti ☺
- prvi način – promjena lokalno spremljenog WSDL-a i *rebuild*:

```
<service name="HelloJavaCroImplService">
  <port name="HelloJavaCroImplPort" binding="tns:HelloJavaCroImplPortBinding">
        <soap:address location="http://drugi-server.com/JavaCroWS/JavaCro"/>
  </port>
</service>
```

Trik #6 – promjena *endpointa*

- ali željeli bismo raditi promjene u *runtime*-u!
- može i to - drugi način:

```
URL url = getClass().getResource("./wsdl/JavaCro.wsdl");
HelloJavaCroImplService serv = new HelloJavaCroImplService(url, new
        QName("http://tuhelj.javacro/", "HelloJavaCroImplService"));
HelloJavaCro stub = serv.getHelloJavaCroImplPort();
BindingProvider bp = (BindingProvider)stub;
bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
        "http://drugi-server.com/JavaCroWS/JavaCro");
System.out.println(stub.hello());
```

Trik #7 – JPA Entity kao povratni tip

- želimo vratiti JPA/Hibernate Entity klasu direktno kao povratnu vrijednost Web metode
- može, ali…. `LazyInitializationException` ☹
- uzrok: vezane kolekcije se dohvaćaju tek po pozivu, a kad do njega dođe, JPA/Hibernate *session* je već zatvoren!
- rješenje: na svim mjestima gdje koristimo *lazy loading* (`@OneToMany` i `@ManyToMany`) staviti `fetch=FetchType.EAGER`

Trik #8 – malo JAXB-a

- *contract-first* pristup
- problem: XSD parametri su tipa `String`, a umjesto toga izgenerirani servis pun je `JAXBElement<String>` parametara:

```
public void createPerson(
        JAXBElement<String> name,
        JAXBElement<String> surname,
        JAXBElement<String> address) {..}
```

- uzrok: u XSD-u su ti parametri istovremeno i opcionalni (`minOccurs="0"`) i *nillable* (`nillable="true"`), čest slučaj kod WCF-generiranih XSD-ova i WSDL-ova

Trik #8 – malo JAXB-a

- `String` nije dovoljan za razlučivanje između "nema ga" i "ima ga, ali *null* je"
- rješenje: posebna *bindings* datoteka:

```
<jaxb:bindings version="2.0"
        xmlns:jxb="http://java.sun.com/xml/ns/jaxb"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <jaxb:globalBindings generateElementProperty="false"/>
</jaxb:bindings>
```

- korištenje: `wsimport -b <ime_datoteke>...`

Trik #9 – još malo JAXB-a

- želimo izgenerirati JAX-WS klijenta na temelju WSDL/XSD-a u kojem se `complexType` i `element` jednako zovu (često kod PHP-generiranih):

```
<xsd:complexType name="createAccountResponse">
     <xsd:all>
         <xsd:element name="status" type="xsd:string" nillable="true"/>
         <xsd:element name="message" type="xsd:string" nillable="true"/>
         <xsd:element name="accountNumber" type="xsd:string" nillable="true"/>
      </xsd:all>
</xsd:complexType>
<xsd:element name="createAccountResponse">
..
</xsd:element>
```

- ovo neće proći („`class with the same name is already in use`”)

Trik #9 – još malo JAXB-a

- prvo „rješenje” – ručno preimenovati sve `complexType` strukture
- drugo rješenje – opet *bindings* datoteka:

```
<jxb:bindings version="2.0"
    xmlns:xjc="http://java.sun.com/xml/ns/jaxb/xjc"
    xmlns:jxb="http://java.sun.com/xml/ns/jaxb"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <jxb:bindings schemaLocation="schema.xsd">
        <jxb:bindings node="//xsd:element[@name='createAccountResponse']">
            <jxb:class name="CreateAccountResponseElement" />
    </jxb:bindings>
</jxb:bindings>
```

Trik #10 – presretači

Trik #10 – presretači

- cilj: presretanje dolazne/odlazne SOAP poruke prije nego stigne na odredište radi dodatne obrade
- rješenje: `SOAPHandler` sučelje!

```
package javacro.tuhelj;
public class Presretac1 implements SOAPHandler<SOAPMessageContext> {
        @Override
        public boolean handleMessage(SOAPMessageContext context) {
                SOAPMessage msg = context.getMessage();
                SOAPEnvelope env = msg.getSOAPPart().getEnvelope();
                SOAPBody body = env.getBody();
                //sad kad imamo body mijenjamo poruku po želji..
        .. }
}
```

- slično definiramo i drugog presretača, `Presretac2`

Trik #10 – presretači

- `handlers.xml` služi za deklariranje presretača i poretka presretanja:

```
..
<handler-chain>
        <handler>
                <handler-class>javacro.tuhelj.Presretac1</handler-class>
        </handler>
        <handler>
                <handler-class>javacro.tuhelj.Presretac2</handler-class>
        </handler>
</handler-chain>
```

- konačno, moramo obavijestiti i servis o postojanju presretača:

```
@WebService(..)
@HandlerChain(file = "handlers.xml")
public class HelloJavaCroImpl implements HelloJavaCro { ..
```

SOUTH EASTERN EUROPE

JavaCro '13

Hvala na pažnji!

MILJEN MIKIĆ
5.6.2013.
JavaCro@Tuhelj