Webszolgáltatások

Standard webszolgáltatások, WSDL, SOAP, JAX-WS

Simon Károly
simon.karoly@codespring.ro

Webszolgáltatások

- Megoldás osztott rendszerek esetében a platform- és programozási nyelv-független kommunikációra.
- Alapja: XML, SOAP, WSDL
- Java EE integráció: Java API for XML Web Services (JAX-WS) – a referencia implementáció a Glassfish projekt része.
- Java EE "definíció" a standard webszolgáltatásokra: WSDL által leírt távoli alkalmazások, amelyekkel SOAP protokollon keresztül kommunikálhatunk.
- További API-k:
  - Java API for XML based RPC (JAX-RPC) – ezt váltotta a JAX-WS, a váltás és átnevezés motivációja egyszerűsíteni a modellt és hangsúlyozni, hogy a WS nem csak RPC-ről szól.
  - SOAP with Attachements API for Java (SAAJ) – SOAP üzenetek manipulációja (alacsonyabb szinten).
  - Java API for XML Registries (JAXR) – hozzáférés WS regiszterkehez (általában UDDI-hoz)
    - UDDI – Universal Description, Discovery and Integration: specifikáció, amely leírja, hogy hogyan lehet WS-eket publikussá tenni és keresni az Interneten. Repository, amely vállalatokat és általuk biztosított szolgáltatásokat ír le. Kereshetünk vállalatok azonosítói (White pages), üzleti doménium (Yellow pages) vagy technológiai információk (Green Pages alapján). Maga az UDDI repo is egy WS, amely SOAP üzenetek segítségével manipulálható.
    - Vállalatok saját UDDI implementációi + UBR (Universal Business Registry), amely bárki számára elérhető (Microsoft, IBM, SAP)

XML - ismétlés

- XML, DTD, XSD, XML Namespaces
- Példa: Titan Cruise: Reservation, Customer, Address

```
<?xml version='1.0' encoding='UTF-8' ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.titan.com/Address">
  <complexType name="AddressType">
    <sequence>
      <element name="street" type="string"/>
      <element name="city" type="string"/>
      <element name="state" type="string"/>
      <element name="zip" type="string"/>
    </sequence>
  </complexType>
</schema>
```

XML - ismétlés

```
<?xml version='1.0' encoding='UTF-8' ?>
<schema
    xmlns=http://www.w3.org/2001/XMLSchema
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-Instance
    xmlns:addr=http://www.titan.com/Address
    xmlns:res=http://www.titan.com/Reservation
    targetNamespace="http://www.titan.com/Reservation">
  <import namespace="http://www.titan.com/Address"
                xsi:schemaLocation="http://www.titan.com/Address.xsd" />
  <element name="reservation" type="res:ReservationType"/>
  <complexType name="ReservationType">
    <sequence>
              <element name="customer" type="res:CustomerType "/>
              <element name="cruise-id" type="int"/>
              <element name="cabin-id" type="int"/>
              <element name="price-paid" type="double"/>
    </sequence>
  </complexType>
  <complexType name="CustomerType">
    <sequence>
              <element name="last-name" type="string"/>
              <element name="first-name" type="string"/>
              <element name="address" type="addr:AddressType"/>
              <element name="credit-card" type="res:CreditCardType"/>
    </sequence>
  </complexType>
  <complexType name="CreditCardType">
    <sequence>
              <element name="exp-date" type="dateTime"/>
              <element name="number" type="string"/>
              <element name="name" type="string"/>
              <element name="organization" type="string"/>
    </sequence>
  </complexType>
</schema>
```

XML - ismétlés

```
<?xml version='1.0' encoding='UTF-8' ?>
<reservation xmlns="http://www.titan.com/Reservation"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance"
             xsi:schemaLocation="http://www.titan.com/Reservation
                                 http://www.titan.com/schemas/reservation.xsd">
  <customer>
    <last-name>Jones</last-name>
    <first-name>Sara</first-name>
    <address xmlns="http://www.titan.com/Address"
                 xsi:schemaLocation="http://www.titan.com/Address
                                     http://www.titan.com/schemas/address.xsd">
      <street>3243 West 1st Ave.</street>
      <city>Madison</city>
      <state>WI</state>
      <zip>53591</zip>
    </address>
    <credit-card>
      <exp-date>09-2007</exp-date>
      <number>03940293028940289 30</number>
      <name>Sara Jones</name>
      <organization>VISA</organization>
    </credit-card>
  </customer>
  <cruise-id>123</cruise-id>
  <cabin-id>333</cabin-id>
  <price-paid>6234.55</price-paid>
</reservation>
```

SOAP

- A CORBA IIOP-hez hasonló protokoll osztott rendszereken belüli kommunikációhoz.
- XML alapú, a saját XML sémája alapján definiált.
- Üzenet példa:

```
<?xml version='1.0' encoding='UTF-8' ?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Header />
   <env:Body>
       <reservation xmlns="http://www.titan.com/Reservation">
           <customer>
                   <!-- customer info goes here -->
           </customer>
           <cruise-id>123</cruise-id>
           <cabin-id>333</cabin-id>
           <price-paid>6234.55</price-paid>
       </reservation>
   </env:Body>
</env:Envelope>
```

- Envelope, header (biztonsággal, tranzakcióval kapcsolatos illetve routing információk stb.) - opcionális, body (az alkalmazással kapcsolatos adatok).
- Hálózati protokolltól független, de általában HTTP-vel használják
- Megjegyzés: az üzenetek általában szoftver által generáltak és feldolgozottak, ritkán kerülünk "direkt" kapcsolatba a SOAP-al.

SOAP & WS-I BP

- WS-Interoperability Basic Profile: WS-ek közötti kommunikáció specifikálása, standardizálása.
- WS stílusok/kommunikációs lehetőségek:
  - Document/Literal message: a SOAP üzenet body része egyetlen XML séma, így egyszerűen validálható és következményként ez a preferált stílus (előző példa).
  - RPC/Literal: a SOAP üzenetek RP hívások, paraméterekkel és visszatérített értékkel, mindegyikhez saját XML séma tartozik.

- ```
  public interface TravelAgent {
     public void makeReservation(int cruiseID, int cabinID,
                                 int customerId, double price);
     }
  ```
- ```
  <env:Envelope
     xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:titan="http://www.titan.com/TravelAgent"/>
       <env:Body>
         <titan:makeReservation>
             <cruiseId>23</cruiseId>
             <cabinId>144</cabinId>
             <customerId>9393</customerId>
             <price>5677.88</price>
         </titan:makeReservation>
       </env:Body>
     </env:Envelope
  ```

WSDL

- Web Service Description Language: WS-eket leíró XML dokumentum, programozási nyelvtől, platformtól és protokolltól független.
- Példa: makeReservation metódust tartalmazó TravelAgent interfész implementációja WS-ként. Megfelelő WSDL:

```
<?xml version="1.0"?>
<definitions name="TravelAgent"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:titan="http://www.titan.com/TravelAgent"
   targetNamespace="http://www.titan.com/TravelAgent">
<!-- message elements describe the parameters and return values -->
<message name="RequestMessage">
   <part name="cruiseId"   type="xsd:int" />
   <part name="cabinId"    type="xsd:int" />
   <part name="customerId" type="xsd:int" />
   <part name="price"      type="xsd:double" />
</message>
<message name="ResponseMessage">
   <part name="reservationId" type="xsd:string" />
</message>
```

WSDL

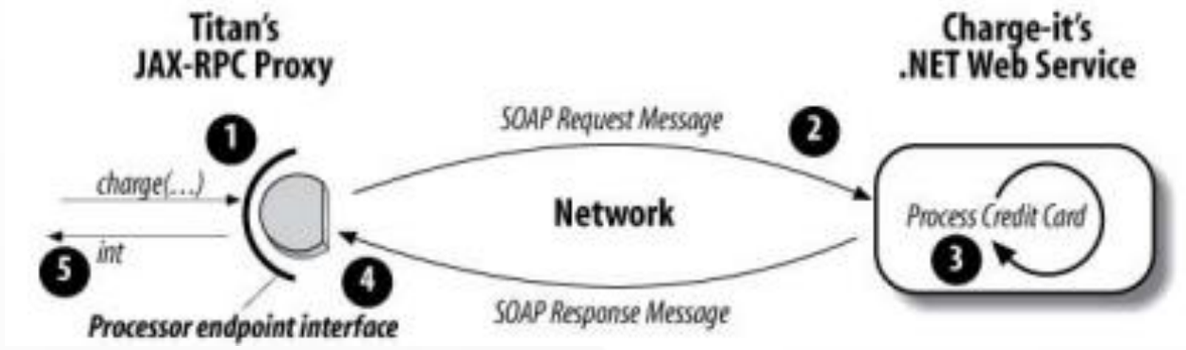
```
<!-- portType element describes the abstract
 interface of a web service -->
<portType name="TravelAgent">
  <operation name="makeReservation">
     <input message="titan:RequestMessage"/>
     <output message="titan:ResponseMessage"/>
  </operation>
</portType>

<!-- binding element tells us which protocols and encoding styles are used -->
<binding name="TravelAgentBinding" type="titan:TravelAgent">
   <soap:binding style="rpc"
                 transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="makeReservation">
        <soap:operation soapAction="" />
        <input>
          <soap:body use="literal"
                namespace="http://www.titan.com/TravelAgent"/>
        </input>
        <output>
          <soap:body use="literal"
                namespace="http://www.titan.com/TravelAgent"/>
        </output>
   </operation>
</binding>
<!-- service element tells us the Internet address of a web service -->
<service name="TravelAgentService">
  <port name="TravelAgentPort" binding="titan:TravelAgentBinding">
     <soap:address location="http://www.titan.com/webservices/TravelAgent" />
  </port>
</service>
</definitions>
```

WSDL

- Definitions: felhasznált XML névterek deklarációja.
- Message: a paraméterek és visszatérített értékek típusainak meghatározása.
- Port type: a rendelkezésre álló WS műveletek (esetünkben Java metódusok).
  - A műveletekhez hozzárendelhetőek input, output és fault (SOAP) üzenetek (messages).
  - Két alapvető WS típus: request-response, one-way (egyetlen input elem, aszinkron kommunikációnál alkalmazható). Továbbá: notifikáció (egyetlen output elem), kérés (egyetlen input elem) (a WSDL biztosítja, de a WS-I BP által nem támogatottak).
- Types: ha szolgáltatásoknak specifikus típusokra van szüksége, ezek ilyen elemeken belül definiálhatóak.
- Binding: a stílus, encoding és protokoll meghatározása, protokoll-specifikus elemekkel kombinált.
- Service: a WS elérhetősége (Internetes címe).

JAX-RPC

- A CORBA/RMI modellhez hasonló, némileg elavultnak számít: a JAX-WS váltotta. A váltás motivációja: a JAX-RPC kommunikációs modell bonyolultsága + a WS nem csak RPC-ről szól.
- Példa:



- JAX-RPC alapú WS létrehozásának lépései:
  - WSDL létrehozása
  - JAX-RPC specifikus részek (artifacts) generálása a WSDL alapján: **Java (endpoint) interfész** (a message és portType elemek alapján generált), az interfészt implementáló **proxy** (a binding és port elemek alapján generált) – feladata a metódushívások SOAP üzenetekben alakítása, illetve a válaszként érkező SOAP üzenetek visszatérési értékbe vagy kivételbe alakítása, **service interfész** – futási idejű hozzáférés a proxy példányhoz (a service elem alapján generált).

JAX-RPC WS meghívása EJB-ből

- DI alkalmazható, így szerezhető referencia a proxy-ra.
- A WS definíció megadásához az EJB telepítés leíróba szükséges egy <service-ref> elem (hogy megtörténhessen a DI). A service-ref-en belüli adatok: service-ref-name, service-interface, wsdl-file, jaxrpc-mapping-file, service-qname, mapped-name + injection-target elem.
- JAX-RPC mapping file: a szolgáltatásnak, illetve kliensnek szüksége van egy ilyen állományra, amely a Java-WSDL és WSDL-JAVA megfeleltetés leírására szolgál (a több lehetőség miatti JAX-RPC implementációk közötti inkompatibilitás kiszűrésére). Automatikusan generálható és a JAX-WS által már nem alkalmazott.
- Ha hálózati probléma lép fel, vagy SOAP üzenet feldolgozásával kapcsolatos probléma, a proxy RemoteException-t dob, ami a konténer által EJBException-ként lesz továbbdobva.
- Ha a WS hívás befejezése után, de az EJB metódus visszatérése előtt lép fel hiba, az esetleges rollback művelet csak részleges (a WS oldalán – pl. távoli rendszeren – nincs rollback). A WS hívás nem része a hívó tranzakció kontextusának.

JAX-RPC WS implementációja

- Servlet vagy EJB segítségével
- EJB: a központi komponens az EJB endpoint, amely egy állapot nélküli session bean WS-ként publikálva. (A további remote/local interfészek mellett) implementál egy service endpoint interfészt, amelyben a WS kliens által hívható metódusokat deklarálja. Az interfész a Remote interfész kiterjesztése.
- Mindegyik EJB endpoint-hoz tartozik egy WSDL, amely leírja a WS-t. Fontos a WSDL portType és endpoint interfész közötti helyes megfeleltetés. Általában a jar állomány META-INF könyvtárában helyezzük el.
- A WSDL és JAX-RPC mapping állományokon kívül szükséges egy webservices.xml állomány, amely összekapcsolja a leíró állományokat és bean-eket (egy vagy több webservice-description elemet tartalmaz).
- Megjegyzés: a JAX-WS egyszerűsíti ezt a mechanizmust és szükségtelenné tesz bizonyos konfigurációs/telepítés-leíró állományokat.

JAX-WS

- JSR-181 (Web Services Metadata for the Java Platform) annotációk.
- Példa:

```
package com.titan.webservice;
import javax.ejb.Stateless;
import javax.jws.WebService;
import javax.jws.WebMethod;
@Stateless
@WebService
public class TravelAgentBean
{
   @WebMethod
   public String makeReservation(int cruiseId, int cabinId,
                                 int customerId, double price) {
   ...
   }
}
```

JAX-WS

- **@WebService:** állapot nélküli session bean publikálása WS-ként:
  - Name - a WS neve, a portType nevének megadásakor felhasználva a WSDL-en belül, alapértelmezetten a Java osztály neve.
  - targetNamespace – az XML névtér, alapértelmezetten a csomagnév alapján.
  - wsdlLocation – a WSDL dokumentum URL-je, akkor szükséges, ha létező wsdl-nek felel meg a szolgáltatás.
  - endpointInterface – Java interfész formájában különválasztani a szolgáltatás deklarációját (a "szerződést").
  - portName – a WSDL port.
- **@WebMethod**
  - Ha nincs ilyen, akkor minden metódus hozzáférhetővé lesz téve a szolgáltatáson keresztül.
  - operationName – a metódusnak megfelelő WSDL operation meghatározása, alapértelmezetten a metódus neve.
- További annotációk: @SOAPBinding (stílus meghatározása, alapértelmezetten Document/Literal), @WebParam (WSDL paraméter típusok meghatározása), @WebResult (WSDL válasz típusának meghatározása), @OneWay (jelzi, hogy nincs visszatérített érték, pl. jelezve, hogy lehetséges az aszinkron kommunikáció – bár a konkrét végrehajtás Java EE implementációtól függ).

JAX-WS

- WS "szerződés" különválasztása: külön endpoint interfész és a @WebService endpointInterface attribútumának használata. Az interfészt szintén a @WebService annotációval látjuk el, a többi annotáció opcionális.

```
package com.charge_it;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
@WebService
@SOAPBinding(style = SOAPBinding.Style.RPC)
public interface Processor{
    public int charge(String name, String number, java.util.Calendar expDate,
                      String cardType, float amount);
}
```

JAX-WS

- A JAX-WS biztosít egy Service osztályt, amelyet a kliens használ a WS-el történő kommunikációhoz. Példa:

```
package com.charge_it;
import javax.xml.ws.WebServiceClient;
import javax.xml.ws.WebEndpoint;
@WebServiceClient(name="ProcessorService",
                  targetNamespace="http://charge-it.com/Processor"
                  wsdlLocation="http://charge-it.com/Processor?wsdl")
public class ProcessorService extends javax.xml.ws.Service {
   public ProcessorService( ) {
      super(new URL("http://charge-it.com/Processor?wsdl"),
            new QName("http://charge-it.com/Processor", "ProcessorService"));
   }
   public ProcessorService(String wsdlLocation, QName serviceName) {
      super(wsdlLocation, serviceName);
   }
   @WebEndpoint(name = "ProcessorPort")
   public Processor getProcessorPort( ) {
      return (Processor)
         super.getPort(
            new QName("http://charge-it.com/Processor", "ProcessorPort"),
            Processor.class);
   }
}
```

JAX-WS

- @WebServiceRef annotáció is alkalmazható a JAX-WS kliensek esetében (az ejb-jar.xml állományon belüli service-ref elem mellett). Attribútumai:
  - Name: JNDI azonosító
  - wsdlLocation: WSDL URL
  - mappedName: szolgáltató-specifikus globális azonosító
  - Type és value: az injection végrehajtásához lehet szükséges
- Példa: implementáció injektálása, a típus alapján:

```
@WebServiceRef(ProcessorService.class)
private Processor endpoint;
```

- Példa: szolgáltatás interfész injektálása, típus alapján:

```
@WebServiceRef
private ProcessorService service;
```

JAX-WS – Hello World

```
package helloservice.endpoint;
import javax.jws.WebService;
import javax.jws.WebMethod;

@WebService
public class Hello {
        private String message = new String("Hello, ");

        public void Hello() { }

        @WebMethod
        public String sayHello(String name) {
                return message + name + ".";
        }
}
```

JAX-WS – Hello World

```
package appclient;
import helloservice.endpoint.HelloService;
import javax.xml.ws.WebServiceRef;

public class HelloAppClient {

	@WebServiceRef(wsdlLocation =
		"META-INF/wsdl/localhost_8080/helloservice/HelloService.wsdl")
	private static HelloService service;

	public static void main(String[] args) {
		System.out.println(sayHello("world"));
	}

	private static String sayHello(java.lang.String arg0) {
		helloservice.endpoint.Hello port = service.getHelloPort();
		return port.sayHello(arg0);
	}
}
```

JAX-B

- Java Architecture for XML Binding
- XML tartalom Java objektumokba alakítása, illetve Java objektumok XML dokumentumba alakítása
- XML mellett JSON formátum is alkalmazható
- Annotációk alkalmazása
- ```
  <xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://titan.com/custom">
     <xs:complexType name="address">
       <xs:sequence>
         <xs:element name="street" type="xs:string" minOccurs="0"/>
         <xs:element name="city" type="xs:string" minOccurs="0"/>
         <xs:element name="zip" type="xs:string" minOccurs="0"/>
       </xs:sequence>
     </xs:complexType>
  </xs:schema>
  ```

- ```
  @XmlAccessorType(XmlAccessType.FIELD)
  @XmlType(name = "address", propOrder = {"street","city","zip"})
  public class Address {
      protected String street;
      protected String city;
      protected String zip;

          …  // getters and setters
  }
  ```

RESTful webszolgáltatások

- REST (Representational State Transfer): architektúra/programozási modell
  - Állapot nélküli kommunikációra tervezett kliens-szerver architektúra, tipikusan HTTP protokollon keresztüli kommunikáció.
  - Az adatok és műveletek erőforrásokként (resources) vannak kezelve, egységesen, URI-k által beazonosítottak és hozzáférhetőek.
  - Az erőforrások manipulációja négy jól meghatározott egyszerű művelettel lehetséges, HTTP analógia: PUT, GET, POST, DELETE
    - PUT: erőforrás módosítása
    - DELETE: erőforrás törlése
    - GET: erőforrás aktuális állapotának lekérdezése (valamilyen reprezentációnak megfelelően)
    - POST: erőforrás létrehozása
  - Különböző formátumok: HTML, XML, JSON, plain text, PDF, JPEG stb.
  - Erőforrásokkal kapcsolatos metaadatok
  - Alapvetően állapot nélküli, de bizonyos módszerekkel lehetőség van állapotinformációk megőrzésére/továbbítására is (explicit állapot-továbbítás: URI átírás, cookies stb.)

JAX-RS

- Java API for RESTful Web Services – JSR 311.
- Referencia implementáció: Jersey.
- Annotációkon alapszik: runtime annotációk, reflection alkalmazása.
- Erőforrás osztályok: Root Resource classes – @Path annotációval ellátott POJOk, vagy legalább egy @GET, @PUT, @POST, @DELETE vagy @Path annotációval ellátott metódussal rendelkező osztályok.
  - @Path – a Java osztály helyét meghatározó relatív URI, pl. /helloworld. Lehetőség van változók beágyazására, pl. felhasználónév lekérdezése: /helloworld/{username}.
  - @GET – HTTP GET kérés, tipikusan erőforrások lekérdezésére
  - @POST – HTTP POST kérés, tipikusan erőforrások létrehozására
  - @PUT – HTTP PUT kérés, tipikusan erőforrások módosítására
  - @DELETE – HTTP DELETE kérés, tipikusan erőforrások törlésére

JAX-RS – további annotációk

- @HEAD – HTTP head kérés (GET kéréshez hasonló, de csak a header-t adja vissza).
- @PathParam – paraméter típus, ami az erőforrás osztályon belül használható. A paraméterek a kérés URI-ból lesznek lekérve, neveik megfelelnek az URI template változóneveivel, amelyeket a @Path annotációban adunk meg.
- @QueryParam – paraméter típus, amelyet az erőforrás osztályon belül használhatunk, az URI query paramétereiből lesz kinyerve
- @Consumes – a kliens által küldött erőforrás reprezentációjának MIME médiatípusa
- @Produces – a kliensnek küldött erőforrás reprezentációjának MIME médiatípusa
- @Provider – a JAX-RS runtime által használt elemek jelölése, pl. MessageBodyReader és MessageBodyWriter, paraméterek HTTP kérésekből és válaszokból történő kinyerésére, metaadatok (pl. HTTP header, státusz/hibakódok) küldése (ResponseBuilder segítségével felépített Response objektumok).