X36DSV cv. 6 – Axis2 v1.1

V tomto krátkém tutoriálu si ukážeme vytvoření jednoduché serverové a následně i klientské části echo služby.

Návrh aplikace – implementace pomocí API – 'serverová' část

Jedna z cest vytvoření aplikace pro Axis2. Má 4 fáze:

- Implementace tříd pomocí standardního API.
- Napsání services.xml pro popis nabízených služeb.
- Vytvoření .aar (Axis ARchive)
- 'Deploynutí' výsledného archivu.

Pro počáteční testovací **implmentaci** si vybereme velice jednoduchý echo server. Následuje kód serveru:

```
package x36dsv.cv6;
import javax.xml.stream.XMLStreamException;
import org.apache.axiom.om.OMElement;
public class TestServer1 {
      /* echo service */
      public OMElement echo(OMElement element) throws XMLStreamException {
        /* vytvoreni 'DOM' stromu z cele prichozi zpravy */
        element.build();
        /* 'osamostatneni' prijate struktury */
        element.detach();
        return element;
    }
}
```

Třída `OMElement` reprezentuje jak přijatou tak odeslanou xml zprávu resp. její tělo (`OMElement` je AXIOM verzí standardního XML elementu) . Dá se s ní pracovat pomocí podobných metod jaké jsou u vidění u klasického frameworku XML.

Tvorba souboru **services.xml** se dá udělat dvojím způsobem. Prvním z nich je ruční generace „kódu“ a tím druhým je automatická generace pomopcí pluginu do Eclipse (ten ale zatím není úplně bez chyb – viz http://ws.apache.org/axis2/tools/1_1/eclipse/wsdl2java-plugin.html).

```
<service name="TestServer">
    <description>
        Pokusny server c.1 pro potreby prezentace jednoduche aplikace.
    </description>
    <parameter name="ServiceClass"
locked="false">x36dsv.cv6.TestServer1</parameter>
    <operation name="echo">
        <messageReceiver
class="org.apache.axis2.receivers.RawXMLINOutMessageReceiver"/>
        <actionMapping>urn:echo</actionMapping>
    </operation>
</service>
```

Předchozí část kódu popisuje námi definovanou službu. Struktura tohoto XML souboru je následující:

`'service'` pojmenovává službu (mimochodem stejně by se měl jmenovat i výsledný .aar soubor), Axis použije tento název pro vytvoření endpointu () ve tvaru `http://<computer>:<port>/axis2/services/<nameofservice>`

`'description'` (volitelný tag) slouží ke slovnímu popisu služby

`'parameter'` specifikuje, která třída se implementuje dané operace

`'operation'` operace nabízené službou a jejich parametry a zpracovávající třídy (existují 2 předdefinované `RawXMLINOutMessageReceiver` (výchozí nastavení), `RawXMLINOnlyMessageReceiver`)

Dále je možno (a rozhodně je to doporučeno) vygenerovat si **WSDL** soubor příslušející dané třídě. Axis jej potom nabízí ke stáhnutí na adrese `<endpoint>?wsdl`. Zase jsou 2 možnosti – ruční psaní nebo automatická generace pomocí pluginu v Eclipse případně pomoci třídy Java2WSDL, která je obsažena přímo v distribuci Axisu. První cesta vyžaduje relativně podrobnou znalost struktury WSDL (http://www.w3.org/TR/wsdl). Druhá je o poznání snažší, postup pro použití pluginu lze nalézt zde http://ws.apache.org/axis2/tools/1_1/eclipse/wsdl2java-plugin.html (výsledný kód je v přiloženém archivu). V případě absence pluginu zbývá cesta příkazové řádky, která by v našem případě měla mít asi takovýto tvar (aktivní adresář `<axis_home>/bin/` a v něm nahraná třída, ze které generujeme WSDL soubor).

```
java2wsdl.bat -cn x36dsv.cv6.TestServer1 ^
              -cp . ^
              -tn "http://TestServer1.cv6.x36dsv" ^
              -tp "ns" ^
              -stn "http://TestServer1.cv6.x36dsv/types" ^
              -stp "types" ^
              -sn "TestServer1" ^
              -of testserver1.wsdl
```

V tuto chvíli již máme vše potřebné pro **vytvoření samotného archivu**. Zde by nám zase měl posloužit plugin do Eclipse. Dokumentaci pro jeho použití lze nalézt na adrese http://ws.apache.org/axis2/tools/1_1/eclipse/servicearchiver-plugin.html. Pokud se vyskytnou

problémy s pluginem, postačí výslednou strukturu 'zazipovat' a prejmenovat vysledny soubor na `<jmeno_sluzby>.aar`.

Samotné vystavení (**deployment**) služby je na celé operaci asi to nejlehčí. Postačí výsledný soubor nahrát do adresáře `<$TOMCAT_HOME>|<$JAVA_APP_SERVER_HOME>/webapps/`.

Návrh aplikace – implementace pomocí API – 'klientská' část

Klientská část je již podstatně lehčí. Zde následují zkrácené výpisy kódu.

```
public class TestClient1 {

    private static EndpointReference targetEPR = new
EndpointReference("http://<adresa_serveru>:8080/axis2/services/TestServer1");

    public static void main(String[] args) {
        try {
            OMElement payload = TestUtils.getEchoOMElement();
            Options options = new Options();
            options.setTo(targetEPR);

            System.out.println("Payload: "+payload);

            //Blocking invocation
            ServiceClient sender = new ServiceClient();
            sender.setOptions(options);
            OMElement result = sender.sendReceive(payload);

            System.out.println("Result: "+result);

        } catch (AxisFault axisFault) {
            axisFault.printStackTrace();
        }
    }
}

...

      public static OMElement getEchoOMElement() {
          OMFactory fac = OMAbstractFactory.getOMFactory();
          OMNamespace omNs = fac.createOMNamespace(
                  "http://example1.org/example1", "example1");
          OMElement method = fac.createOMElement("echo", omNs);
          OMElement value = fac.createOMElement("Text", omNs);
          value.addChild(fac.createOMText(value, "Axis2 Echo String "));
          method.addChild(value);

          return method;
      }
```

Metoda `getEchoOMElement()` vrací připravené tělo zprávy pro odeslání (jedná se o jen o tělo – tzn. o RAW zprávu – jelikož jsme použili `RawXMLINOutMessageReceiver`). Zbytek kodu je snad dostatečně pochopitelný.