JavaOne℠

Coding SOAP and REST Together

Jakub Podlesak,
Martin Grebac
Sun Microsystems
Czech Republic

Goal

We will show you how to combine the good from REST and SOAP to build better web services

Agenda

- **JAX-WS (SOAP) Basics**
- JAX-RS (REST) Basics
- Code both together (demo)
- Bad Practices (demo)
- Good Practices
- Music Store demo

Simple Web Service - POJO

Write code:

```
import javax.jws.WebService;
@WebService
public class TechSessionWS {
    public TechSession getSession(String s) {
        return getTS(s);
    }
}
```

Compile & Deploy

Simple Web Service – The Result

WSDL Definition (The Contract)

```
- <definitions targetNamespace="http://greeter/" name="GreeterService">
  - <types>
    - <xsd:schema>
        <xsd:import namespace="http://greeter/" schemaLocation="http://localhost:8080/GreeterProject/GreeterService?xsd=1"/>
      </xsd:schema>
    </types>
  - <message name="sayHello">
      <part name="parameters" element="tns:sayHello"/>
    </message>
  - <message name="sayHelloResponse">
      <part name="parameters" element="tns:sayHelloResponse"/>
    </message>
  - <portType name="Greeter">
    - <operation name="sayHello">
        <input message="tns:sayHello"/>
        <output message="tns:sayHelloResponse"/>
      </operation>
    </portType>
  - <binding name="GreeterPortBinding" type="tns:Greeter">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    - <operation name="sayHello">
        <soap:operation soapAction=""/>
      - <input>
          <soap:body use="literal"/>
        </input>
      - <output>
          <soap:body use="literal"/>
        </output>
      </operation>
    </binding>
  - <service name="GreeterService">
    - <port name="GreeterPort" binding="tns:GreeterPortBinding">
        <soap:address location="http://localhost:8080/GreeterProject/GreeterService"/>
      </port>
    </service>
</definitions>
```

Request

```
- <soapenv:Envelope>
  - <soapenv:Body>
    - <q0:sayHello>
        <s>JavaOne</s>
      </q0:sayHello>
    </soapenv:Body>
</soapenv:Envelope>
```

Response

```
- <S:Envelope>
  - <S:Body>
    - <sayHelloResponse>
        <return>Hello JavaOne</return>
      </sayHelloResponse>
    </S:Body>
</S:Envelope>
```

Java EE Client

```
@Stateless
public class MyBean {
    @WebServiceRef(TechSessionWS.class)
    TechSessionWS proxy;

    public TechSession useIt() {
      return proxy.getSession("4883");
    }
}
```

Advantages

- No deployment descriptors needed
  - Use annotations
  - Descriptors still available if required
- Part of Java EE 5 and 6
- Protocol and Transport independence
  - JMS, SMTP, Fast Infoset, ...

The Runtime

- METRO = JAXB + JAX-WS + WSIT
  - Web Service Stack
  - http://metro.dev.java.net
- Running on GlassFish v2 / v3 / Tomcat / ...

Why SOAP?

- You contract/part of it is already defined
- Well established infrastructure and know-how
- Advanced security scenarios
- Other non-functional requirements (QoS)

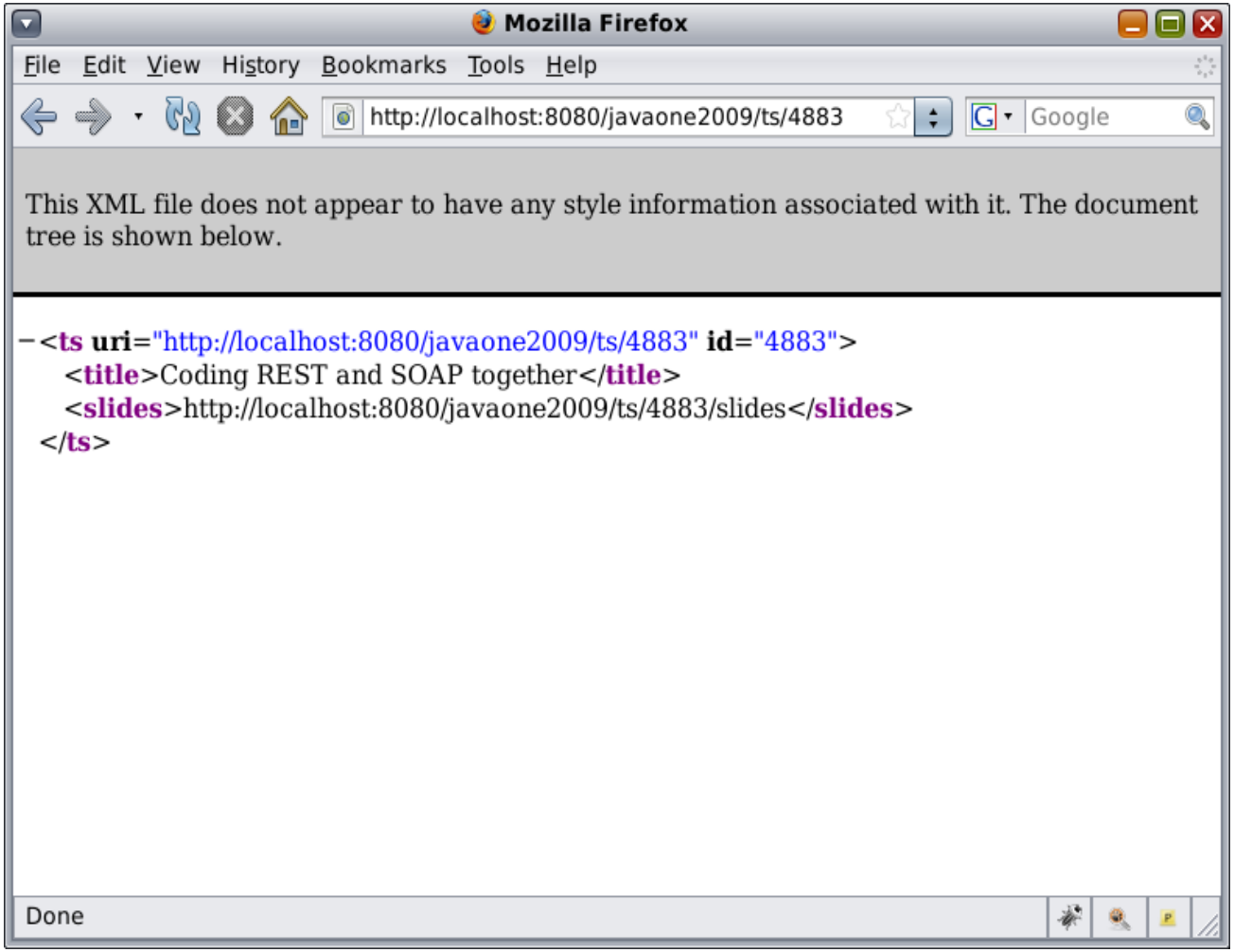
Agenda

- JAX-WS (SOAP) Basics
- **JAX-RS (REST) Basics**
- Code both together (demo)
- Bad Practices (demo)
- Good Practices
- Music Store demo

JAX-RS Web Resource

```
@Path("ts/{id}")
public class TechSessionResource {
  @GET
  @Produces("application/xml","application/json")
  public TechSession getSession(
      @PathParam("id") String id) {
    return getTS(id);
  }
}
```

Client side


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/javaone2009/ts/4883
Google
This XML file does not appear to have any style information associated with it. The document tree is shown below.
−<ts uri="http://localhost:8080/javaone2009/ts/4883" id="4883">
<title>Coding REST and SOAP together</title>
<slides>http://localhost:8080/javaone2009/ts/4883/slides</slides>
</ts>
Done

Another Web Resource

```
public class SlidesResource {
  @PUT @Consumes(
     "application/vnd.oasis.opendocument.presentation")
  public void updateSlides(Slides s) { slides = s;}
  @GET @Produces({"application/pdf",
      "application/vnd.oasis.opendocument.presentation"})
  public Slides getSlides() { return slides;}
  @DELETE
  public void deleteSlides() { slides = null;}
}
```

Why REST ?

- Simplicity (remember the browser as a client?)
- Takes advantage of the existing caches/proxies
- Serendipity
- Scalability (HTTP load balancers)

Common JAX-WS/RS concepts

- POJO based
- Annotations driven
- Web container at runtime

Agenda

- JAX-WS (SOAP) Basics
- JAX-RS (REST) Basics
- **Code both together (demo)**
- Bad Practices (demo)
- Good Practices
- Music Store demo

Let's combine them together!

```
@Path("ts/{id}")
  @WebService
public class TechSessionResource {
  @GET
  @Produces("application/xml","application/json")
  @WebMethod
  public TechSession getSession(
          @PathParam("id") String id) {
     return getTS(id);
  }
}
```

Let's combine them together!

- Demo – TechSession SOAP & REST live

Agenda

- JAX-WS (SOAP) Basics
- JAX-RS (REST) Basics
- Code both together (demo)
- **Bad Practices (demo)**
- Good Practices
- Music Store demo

What might get wrong?

- Do not fall into RPC over HTTP
  - REST-RPC Hybrid
- Use compatible security
  - Securing SOAP doesn't automatically secure the REST
- Annotation hell

What might get wrong

- Demo – how to break REST

What went wrong

- Misuse of HTTP GET method
  - coding method names into URIs
  - GET used for data manipulation
  - danger of unintended data corruption
- Another widely misused method is POST
- Introduced a security hole
  - Only SOAP is secured
  - REST interface remained unsecured !

Agenda

- JAX-WS (SOAP) Basics
- JAX-RS (REST) Basics
- Code both together (demo)
- Bad Practices (demo)
- **Good Practices**
- Music Store demo

Good practices

- Think about the consumers of your service
- Do you really need both SOAP and REST?
  - Consider using REST only in part of your application
- Use `@WebMethod` annotations
  - Enables control over exposed SOAP methods
- To get the widest possible exposure, simply consider providing client code with your service

Agenda

- JAX-WS (SOAP) Basics
- JAX-RS (REST) Basics
- Code both together (demo)
- Bad Practices (demo)
- Good Practices
- **Music Store demo**

Summary

1. You can easily develop SOAP & REST in Java
2. You can easily combine them together
3. You know how to get the best of both worlds

References

- http://jsr311.dev.java.net
- http://jersey.dev.java.net
- http://jax-ws.dev.java.net
- http://metro.dev.java.net

See also

- Thursday, June 04
  - BOF-4878 - Developing RESTful Web Services with Jersey and Java™ API for RESTful Web Services (JAX-RS)
    - 7:30 PM - 8:20 PM
  - TS-4402 - Metro Web Services Security Usage Scenarios
    - 9:30 AM - 10:30 AM
  - BOF-5305 - Java™ API for XML Web Services (JAX-WS) 2.2
    - 7:45 PM - 8:35 PM

Thank You

Jakub Podlesak,
Martin Grebac
http://blogs.sun.com/japod
http://blogs.sun.com/mgrebac