Spring Web Services

Paul Bakker

Outline

- Introduction
- Designing the service
- Configuring Spring WS
- Handling messages
- Implementing a client using Spring WS
- WS-Addressing
- WS-Security

Introduction

- Contract-first web service stack
  - Makes Best Practice an Easy Practice
  - Both server and client
- A Spring sub project
  - Uses Spring configuration
  - Build on a Spring architecture

Comparing to JAX-WS

- Only Contract-First development
  - Harder to start
  - Easier to do it right
- A simple @Webservice annotation only works for helloWorld...
  - A well designed service is a lot harder

Designing a service

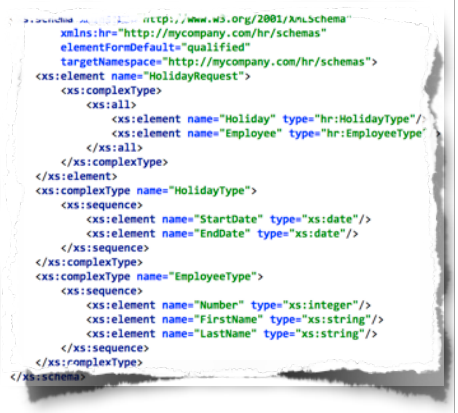
- A HR system to request holidays
- First: design the XSD
- Second: design the WSDL
- Third: Implement the service

A HolidayRequest message

```
<sch:HolidayRequest xmlns:sch="http://mycompany.com/hr/schemas">
    <!--You may enter the following 2 items in any order-->
    <sch:Holiday>
        <sch:StartDate>2009-09-01</sch:StartDate>
        <sch:EndDate>2009-09-15</sch:EndDate>
    </sch:Holiday>
    <sch:Employee>
        <sch:Number>10</sch:Number>
        <sch:FirstName>Paul</sch:FirstName>
        <sch:LastName>Bakker</sch:LastName>
    </sch:Employee>
</sch:HolidayRequest>
```

HolidayRequest XSD

- Three types:
  - HolidayRequest
  - HolidayType
  - EmployeeType
- HolidayRequest is the only message the service accepts




```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:hr="http://mycompany.com/hr/schemas"
        elementFormDefault="qualified"
        targetNamespace="http://mycompany.com/hr/schemas">
    <xs:element name="HolidayRequest">
        <xs:complexType>
            <xs:all>
                <xs:element name="Holiday" type="hr:HolidayType"/>
                <xs:element name="Employee" type="hr:EmployeeType"/>
            </xs:all>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="HolidayType">
        <xs:sequence>
            <xs:element name="StartDate" type="xs:date"/>
            <xs:element name="EndDate" type="xs:date"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="EmployeeType">
        <xs:sequence>
            <xs:element name="Number" type="xs:integer"/>
            <xs:element name="FirstName" type="xs:string"/>
            <xs:element name="LastName" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

HolidayRequest

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:hr="http://mycompany.com/hr/schemas"
        elementFormDefault="qualified"
        targetNamespace="http://mycompany.com/hr/schemas">
    <xs:element name="HolidayRequest">
        <xs:complexType>
            <xs:all>
                <xs:element name="Holiday" type="hr:HolidayType"/>
                <xs:element name="Employee" type="hr:EmployeeType"/>
            </xs:all>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="HolidayType">
        <xs:sequence>
            <xs:element name="StartDate" type="xs:date"/>
            <xs:element name="EndDate" type="xs:date"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="EmployeeType">
        <xs:sequence>
            <xs:element name="Number" type="xs:integer"/>
            <xs:element name="FirstName" type="xs:string"/>
            <xs:element name="LastName" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

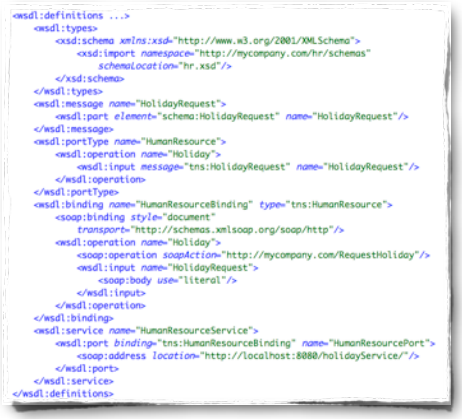


```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:hr="http://mycompany.com/hr/schemas"
        elementFormDefault="qualified"
        targetNamespace="http://mycompany.com/hr/schemas">
    <xs:element name="HolidayRequest">
        <xs:complexType>
            <xs:all>
                <xs:element name="Holiday" type="hr:HolidayType"/>
                <xs:element name="Employee" type="hr:EmployeeType"/>
            </xs:all>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="HolidayType">
        <xs:sequence>
            <xs:element name="StartDate" type="xs:date"/>
            <xs:element name="EndDate" type="xs:date"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="EmployeeType">
        <xs:sequence>
            <xs:element name="Number" type="xs:integer"/>
            <xs:element name="FirstName" type="xs:string"/>
            <xs:element name="LastName" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:hr="http://mycompany.com/hr/schemas"
        elementFormDefault="qualified"
        targetNamespace="http://mycompany.com/hr/schemas">
    <xs:element name="HolidayRequest">
        <xs:complexType>
            <xs:all>
                <xs:element name="Holiday" type="hr:HolidayType"/>
                <xs:element name="Employee" type="hr:EmployeeType"/>
            </xs:all>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="HolidayType">
        <xs:sequence>
            <xs:element name="StartDate" type="xs:date"/>
            <xs:element name="EndDate" type="xs:date"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="EmployeeType">
        <xs:sequence>
            <xs:element name="Number" type="xs:integer"/>
            <xs:element name="FirstName" type="xs:string"/>
            <xs:element name="LastName" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```


WSDL

- Defines a service based on the messages defined by the schema
- Everything is derived from the schema
- The WSDL can optionally be generated by Spring WS



```
<wsdl:definitions ...>
    <wsdl:types>
        <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:import namespace="http://mycompany.com/hr/schemas"
                schemaLocation="hr.xsd"/>
        </xsd:schema>
    </wsdl:types>
    <wsdl:message name="HolidayRequest">
        <wsdl:part element="schema:HolidayRequest" name="HolidayRequest"/>
    </wsdl:message>
    <wsdl:portType name="HumanResource">
        <wsdl:operation name="Holiday">
            <wsdl:input message="tns:HolidayRequest" name="HolidayRequest"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="HumanResourceBinding" type="tns:HumanResource">
        <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="Holiday">
            <soap:operation soapAction="http://mycompany.com/RequestHoliday"/>
            <wsdl:input name="HolidayRequest">
                <soap:body use="literal"/>
            </wsdl:input>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="HumanResourceService">
        <wsdl:port binding="tns:HumanResourceBinding" name="HumanResourcePort">
            <soap:address location="http://localhost:8080/holidayService/"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```


Configuring Spring WS

- Configure a MessageDispatcherServlet
  - web.xml
  - Similar to Spring MVC DispatcherServlet
- Configure the service
  - Specify the schema
  - Publish the WSDL
  - Configure an Endpoint

MessageDispatcherServlet

```
<servlet>
    <servlet-name>spring-ws</servlet-name>
    <servlet-class>
        org.springframework.ws.transport.http.MessageDispatcherServlet
    </servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>spring-ws</servlet-name>
    <url-pattern>/*</url-pattern>
</servlet-mapping>
```


Spring WS config file

- Uses Spring Scheme configuration
  - Everything is a Spring bean

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:oxm="http://www.springframework.org/schema/oxm"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http

    <bean id="schema" class="org.springframework.xml.xsd.SimpleXsdSchema">
        <property name="xsd" value="/WEB-INF/HolidayRequest.xsd"/>
    </bean>
```

Specify the XSD

Publish the WSDL

- Specify properties for the WSDL
- Published at:
  http://localhost:8080/spring-ws/

```
<bean id="holiday"
      class="org.springframework.ws.wsdl.wsdl11.DefaultWsdl11Definition">

    <property name="schema" ref="schema"/>

    <property name="portTypeName" value="HumanResource"/>

    <property name="locationUri"
              value="http://localhost:8080/spring-ws/holidayService/"/>

    <property name="targetNamespace"
              value="http://mycompany.com/hr/definitions"/>
</bean>
```


Endpoints

- Different types of Endpoints available
  - Message Endpoints (the full message)
  - Payload Endpoints (only the payload)
- DOM endpoints
- Marshalling endpoints
- XPath endpoints

Annotated XPath endpoint

```java
@Endpoint
public class HolidayRequestEndpoint {

    @PayloadRoot(localPart = "HolidayRequest",
            namespace = "http://mycompany.com/hr/schemas")
    public Source submitHolidayRequest(
            @XPathParam("//s:FirstName/text()") String firstname ) {
        System.out.println(firstname + " wants to have a holiday");
        return null;
    }
}
```

- Uses XPath to extract data from the incoming request

Configure the Endpoint

Declaring the Endpoint

```xml
<bean id="holidayEndpoint" class="example.holidays.HolidayRequestEndpoint"/>
```

Endpoint mapping (scans for @PayloadRoot)

```xml
<bean class="...PayloadRootAnnotationMethodEndpointMapping"/>
```

Declaring the namespace (for XPath)

```xml
<bean
  class=
  "org.springframework.ws.server.endpoint.adapter.XPathParamAnnotationMethodEndpointAdapter">
    <property name="namespaces">
        <props>
            <prop key="sch">http://mycompany.com/hr/schemas</prop>
        </props>
    </property>
</bean>
```

XML Marshalling

- Use objects instead of plain XML
- Different O/X Mapping frameworks supported
  - JAXB (1 & 2)
  - Castor
  - JiBX
  - XStream
  - XMLBeans

Mapping with JAXB 2

- JAXB classes can be generated from XSD

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "HolidayType", namespace = "http://mycompany.com/hr/schemas", propOrder = {
    "startDate",
    "endDate"
})
public class HolidayType {

    @XmlElement(name = "StartDate", namespace = "http://mycompany.com/hr/schemas", required = true)
    @XmlSchemaType(name = "date")
    protected XMLGregorianCalendar startDate;
    @XmlElement(name = "EndDate", namespace = "http://mycompany.com/hr/schemas", required = true)
    @XmlSchemaType(name = "date")
    protected XMLGregorianCalendar endDate;

     ...
```

Configure JAXB marshaller

```
<bean class="...GenericMarshallingMethodEndpointAdapter">
    <constructor-arg ref="marshaller"/>
</bean>

<oxm:jaxb2-marshaller id="marshaller" contextPath="example.holidays.jaxb"/>
```


Implementing the Endpoint

- Nothing about XML any more

```
@PayloadRoot(localPart = "HolidayRequest",
        namespace = "http://mycompany.com/hr/schemas")
public HolidayResponse submitHolidayRequest(
        HolidayRequest request) {
    System.out.println(
            request.getEmployee().getFirstName() + " wants to have a holiday");

    HolidayResponse response = new HolidayResponse();
    response.setApproved(true);
    return response;
}
```

Implementing a client

- Spring WS offers a WebserviceTemplate
  - Similar to JdbcTemplate and JMSTemplate

ApplicationContext

```
<bean id="messageFactory"
      class="org.springframework.ws.soap.saaj.SaajSoapMessageFactory"/>

<bean id="webServiceTemplate"
      class="org.springframework.ws.client.core.WebServiceTemplate">
    <constructor-arg ref="messageFactory"/>
    <property name="defaultUri"
              value="http://localhost:8080/spring-ws/holidayService"/>
    <property name="marshaller" ref="marshaller"/>
    <property name="unmarshaller" ref="marshaller"/>
</bean>

<oxm:jaxb2-marshaller id="marshaller" contextPath="example.holidays.jaxb"/>
```

Implementing the sender code

```
public class HolidayRequestSender {
    @Autowired
    private WebServiceTemplate template;

    public void send() {
        HolidayRequest request = new ObjectFactory().createHolidayRequest();
        //Construct JAXB objects

        template.marshalSendAndReceive(request);
    }
}
```


WS-Addressing

> “
>
> WS-Addressing provides transport-neutral mechanisms
>
> to address Web services and messages. Specifically, this specification defines XML [XML 1.0, XML Namespaces] elements to identify Web service endpoints and to secure end-to-end endpoint identification in messages.
>
> ”

- We will use the Action header
  - Finding an Endpoint now only needs the header

The SOAP message

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <SOAP-ENV:Header>
        <wsa:MessageID>urn:uuid:21363e0d-2645-4eb7-8afd-2f5ee1bb25cf</wsa:MessageID>
        <wsa:To>http://localhost:8080/spring-ws/holidayService</wsa:To>
Action → <wsa:Action>http://mycompany.com/hr/HolidayRequest</wsa:Action>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <ns2:HolidayRequest xmlns:ns2="http://mycompany.com/hr/schemas">
            <ns2:Holiday>
                <ns2:StartDate>2009-10-01</ns2:StartDate>
                <ns2:EndDate>2009-10-01</ns2:EndDate>
            </ns2:Holiday>
            <ns2:Employee>
                <ns2:Number>100</ns2:Number>
                <ns2:FirstName>Paul</ns2:FirstName>
                <ns2:LastName>Bakker</ns2:LastName>
            </ns2:Employee>
        </ns2:HolidayRequest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


Changing the server configuration

Don't use a Payload mapper anymore

```
<bean class="...PayloadRootAnnotationMethodEndpointMapping"/>
```

Use a Action mapper instead

```
<bean class="...PayloadRootAnnotationMethodEndpointMapping"/>
```

Endpoint implementation

- Now using the @Action annotation

```
@Action("http://mycompany.com/hr/HolidayRequest")
public HolidayResponse submitHolidayRequest(HolidayRequest request) {
    System.out.println(
            request.getEmployee().getFirstName() + " wants to have a holiday");
    HolidayResponse response = new HolidayResponse();
    response.setApproved(true);
    return response;
}
```


Client changes

- Just one line of code

```
template.marshalSendAndReceive(request);
```

becomes

```
template.marshalSendAndReceive(request,
        new ActionCallback("http://mycompany.com/hr/HolidayRequest"));
```

WS-Security

> “
>
> *Delivering a technical foundation for implementing security functions such as integrity and confidentiality in messages implementing higher-level Web services applications*
>
> ”

- Authentication
- Digital signing
- Message Timestamps
- Encryption

Authentication

- Add new header
- Password is encrypted

```
<wsse:Security SOAP-ENV:mustUnderstand="1">
    <wsse:UsernameToken>
        <wsse:Username>Bert</wsse:Username>
        <wsse:Password>jz6pdnIyg8qO+ye5Dl+hY0nJOFs=</wsse:Password>
        <wsse:Nonce >B7HAU5JFHiaJDBg6VXYkdA==</wsse:Nonce>
        <wsu:Created>2009-08-28T09:53:13.597Z</wsu:Created>
    </wsse:UsernameToken>
</wsse:Security>
```

Server configuration

- Add filter to EndpointMapping

```
<bean class="...AnnotationActionEndpointMapping">
    <property name="preInterceptors">
        <list>
            <ref bean="authenticationInterceptor"/>
        </list>
    </property>
</bean>
```


Filter configuration

Could use any kind of user store

```
<bean id="authenticationInterceptor" class="...Wss4jSecurityInterceptor">
    <property name="validationActions" value="UsernameToken"/>
    <property name="validationCallbackHandler">
        <bean class="...SimplePasswordValidationCallbackHandler">
            <property name="users">
                <props>
                    <prop key="Bert">Ernie</prop>
                </props>
            </property>
        </bean>
    </property>
</bean>
```

Client configuration

- Add interceptor

```
<bean id="webServiceTemplate"
      class="org.springframework.ws.client.core.WebServiceTemplate">
    ...
    <property name="interceptors">
        <list>
            <ref bean="usernameInterceptor"/>
        </list>
    </property>
</bean>
```

- Interceptor configuration

```
<bean id="usernameInterceptor" class="...Wss4jSecurityInterceptor">
        <property name="securementActions" value="UsernameToken"/>
        <property name="securementUsername" value="Bert"/>
        <property name="securementPassword" value="Ernie"/>
        <property name="securementMustUnderstand" value="true"/>
</bean>
```


Encryption

- A bit more complicated
  - We need a keystore, certificate and private key

```
keytool -genkey -alias myclientkey -keyalg RSA -keystore
 keystore.jks -storepass 123456 -validity 360 -keypass 123456
```

```
keytool -export -rfc -keystore keystore.jks -storepass clientpassword
 -alias myclientkey -file -MyClient.cer
```

```
keytool -import -trustcacerts -keystore client.jks
 -storepass clientpassword -alias myclientkey -file MyClient.cer
```

Server configuration

```
<bean id="encryptInterceptor" class="...Wss4jSecurityInterceptor">
    <property name="validationActions" value="UsernameToken Encrypt"/>
    <property name="validationDecryptionCrypto">
        <bean class="..CryptoFactoryBean">
            <property name="keyStorePassword" value="123456"/>
            <property name="keyStoreLocation" value="classpath:keystore.jks"/>
        </bean>
    </property>
    <property name="validationCallbackHandlers">
        <list>
        <!-- Password handler-->

        <bean class="..KeyStoreCallbackHandler">
            <property name="privateKeyPassword" value="123456"/>
        </bean>
        </list>
    </property>
</bean>
```


Client configuration

```
<bean id="usernameInterceptor" class="...Wss4jSecurityInterceptor">
    <property name="securementActions" value="UsernameToken Encrypt"/>
    <!-- authentication properties omitted-->

    <property name="securementEncryptionUser" value="myclientkey"/>
    <property name="securementEncryptionCrypto">
        <bean class="...CryptoFactoryBean">
            <property name="keyStorePassword" value="clientpassword"/>
            <property name="keyStoreLocation" value="classpath:client.jks"/>
        </bean>
    </property>

</bean>
```

The verdict

- Enforces best practices
- Supports different kinds of handling
- Adding WS-* is only configuration
- But only for Spring apps...