Spring-WS

Craig Walls

About you...

- By show of hands...
  - Java 6? Java 5? Java 1.4? Java 1.3? Java 1.2-?
  - C#? Ruby? Groovy? Scala? Erlang? Python?
  - Spring 1.x? Spring 2.0.x? Spring 2.5.x?

E-mail: craig@habuma.com Blog: http://www.springloaded.info Source Code: svn://svn.geekisp.com/sia/trunk/Chapter09/Poker-ws

About me...

- Professionally developing software for almost 14 years
  - Java developer for most of that time
- Telecom, finance, retail, education, software
- Principle Consultant with Improving
- Author of Modular Java, Spring in Action and XDoclet in Action
- Spring fanatic

Agenda

- Spring remoting and web services
- Contract-first
- Designing the contract
- Introducing Spring-WS
- Building service endpoints
- Wiring it all together

Spring Remoting and Web Services

Building a service using XFire

Spring Remoting

- Client side:
  - Proxy factory bean : Produces wire-able proxy to a remote service
- Service side:
  - Remote exporter : Exports a Spring-configured POJO as a service
- Available in RMI, Hessian, Burlap, and HttpInvoker, and JAX-WS flavors

Spring-Loaded

XFire

- Full-featured web services stack
- Comes with full Spring remoting support
  - XFireClientFactoryBean
  - XFireExporter
- http://xfire.codehaus.org

SPRING-LOADED

Exporting an XFire Service

- Configure a POJO as a Spring bean
- Configure the XFireExporter to export the service
- Configure Spring's DispatcherServlet in web.xml
- Configure a handler-mapping to map URLs to the XFireExporter

SPRING-LOADED

E-MAIL: CRAIG@HABUMA.COM BLOG: HTTP://WWW.SPRINGLOADED.INFO SOURCE CODE: SVN://SVN.GEEKISP.COM/SIA/TRUNK/CHAPTER09/POKER-WS

JSR-181 and XFire

- Annotate bean class and methods
- Declare as <bean> in Spring
- Configure Spring's DispatcherServlet
- Configure a Jsr181HandlerMapping in Spring

SPRING-LOADED

What if the POJO changes?

- The focus is on the implementation
  - The service is method-centric, not message-oriented
- The contract is a by-product of the export
- The contract is volatile

SPRING-LOADED

E-mail: craig@habuma.com Blog: http://www.springloaded.info Source Code: svn://svn.geekisp.com/sia/trunk/Chapter09/Poker-ws

Contract last

- The most important piece of a service is the contract...not the implementation
- Nevertheless, many services are developed treating the contract as a second-class citizen
  - Some don't even think about it at all

E-mail: craig@habuma.com Blog: http://www.springloaded.info Source Code: svn://svn.geekisp.com/sia/trunk/Chapter09/Poker-WS

Contract-First Web Services

Contract First

- If the contract is so important, then why not elevate it to the position it deserves?
- Create the contract first
  - Then write code that satisfies (not implements) the contract

WSDL-first != Contract-First

- WSDL-First:
  - WSDL is used to generate service skeletons
  - Not unlike using IDL to generate CORBA skeletons
- Resulting skeletons are coupled to the contract
  - Changes to the contract change the skeletons

It's all in the message!

- Contract-last services are operation-centric
  - A service is defined by its operations (methods)
- Contract-first services are message-centric
  - A service is defined by the messages it processes and returns

The contract

- WSDL + XSD
- Operational contract: WSDL describes what the service can do
- Data contract: XSD describes the messages sent to the service

Contract-First: Basic Steps

- Define the messages (XML)
- Create the data contract (XSD)
- Develop an endpoint to process the messages (Spring-WS)
- Map messages to endpoints
- Deploy

E-mail: craig@habuma.com Blog: http://www.springloaded.info Source Code: svn://svn.geekisp.com/sia/trunk/Chapter09/Poker-WS

Defining the Contract

It's not as bad as you think

Create sample messages

- Write some XML that resembles what you want passed in and out of your service
- Believe it or not...this is the hardest part

PokerHandRequest

```
<EvaluateHandRequest
    xmlns="http://www.springinaction.com/poker/schemas">
  <card>
    <suit>HEARTS</suit>
    <face>TEN</face>
  </card>
  <card>
    <suit>SPADES</suit>
    <face>KING</face>
  </card>
  <card>
    <suit>HEARTS</suit>
    <face>KING</face>
  </card>
  <card>
    <suit>DIAMONDS</suit>
    <face>TEN</face>
  </card>
  <card>
    <suit>CLUBS</suit>
    <face>TEN</face>
  </card>
</EvaluateHandRequest>
```

PokerHandResponse

```
<EvaluateHandResponse xmlns=
    "http://www.springinaction.com/poker/schemas">
  <handName>Full House</handName>
</EvaluateHandResponse>
```


Create the data contract

- Create XML Schema that can validate the sample messages
- Options
  - Write it by hand (no fun)
  - Infer it
- Inferred XSD is never perfect
  - Will require some fine-tuning by hand

XSD Inference

- Microsoft's XSD inference tool
  - http://msdn2.microsoft.com/en-us/xml/Bb190622.aspx
- Trang
  - http://www.thaiopensource.com/relaxng/trang.html
- Nocternity (Perl)
  - http://projects.nocternity.net/index.py/en/xsd-inference

Introducing Spring-WS

What is Spring-WS?

- Web services framework that encourages contract-first development
- Web services are implemented as service endpoints
  - Endpoints process XML messages
  - Conceptually similar to Spring MVC
- Includes Object-to-XML mapping framework

Spring-WS vs. Spring MVC

- Spring MVC
  - DispatcherServlet
  - Handler mapping
  - Controller
  - SimpleMappingExceptionResolver
- Spring-WS
  - MessageDispatcher
  - Endpoint mapping
  - Endpoint
  - SoapFaultMappingExceptionResolver

Spring OXM

- Abstraction framework over several popular XML marshaling APIs
- Used by Spring-WS to marshal and unmarshal objects for endpoints
  - Can also be used for general purpose XML marshaling
  - To become part of Spring 3.0

Supported marshaling

- Castor XML
- JAXB (v1 and v2)
- JiBX
- XMLBeans
- XStream

Building service endpoints

Service endpoints

- Process XML messages
- Several Spring-WS base classes:
  - AbstractDom4jPayloadEndpoint
  - AbstractDomPayloadEndpoint
  - AbstractJDomPayloadEndpoint
  - AbstractMarshallingPayloadEndpoint
  - AbstractSaxPayloadEndpoint
  - AbstractStaxEventPayloadEndpoint
  - AbstractStaxStreamPayloadEndpoint
  - AbstractXomPayloadEndpoint
- Annotation-based endpoints
  - @Endpoint, @PayloadRoot, @XPathParam

AbstractJDomPayloadEndpoint

- Endpoint is given a JDom Element to process and must return a JDom Element as a result
- Can use XPath to pull info out of Element objects

```
public class EvaluateHandJDomEndpoint
    extends AbstractJDomPayloadEndpoint
    implements InitializingBean {

    protected Element invokeInternal(Element element)
            throws Exception {
        … // process message
    }
}
```

SPRING-LOADED

AbstractMarshallingPayloadEndpoint

- Given an Object to process; Must return an Object as a result
- Uses Spring OXM
  - XML message is unmarshaled into Object. Returned object is marshaled into XML.

```
public class EvaluateHandMarshallingEndpoint
   extends AbstractMarshallingPayloadEndpoint {

   protected Object invokeInternal(Object object)
       throws Exception {
     PurchaseOrder po = (PurchaseOrder) object;
  … // Process purchase order
     }
  }
```

SPRING-LOADED

Annotation-based endpoint

- Uses annotations to declare endpoints
  - Minimizes XML configuration
  - Allows for more POJO-ish endpoints
  - Helps with XML parsing
- Three annotations
  - @Endpoint - Declares a class as an endpoint
  - @PayloadRoot - Specifies a method as the destination for the payload message
  - @XPathParam - Defines how to parse message into method parameters

Spring-Loaded

Annotation-based endpoint

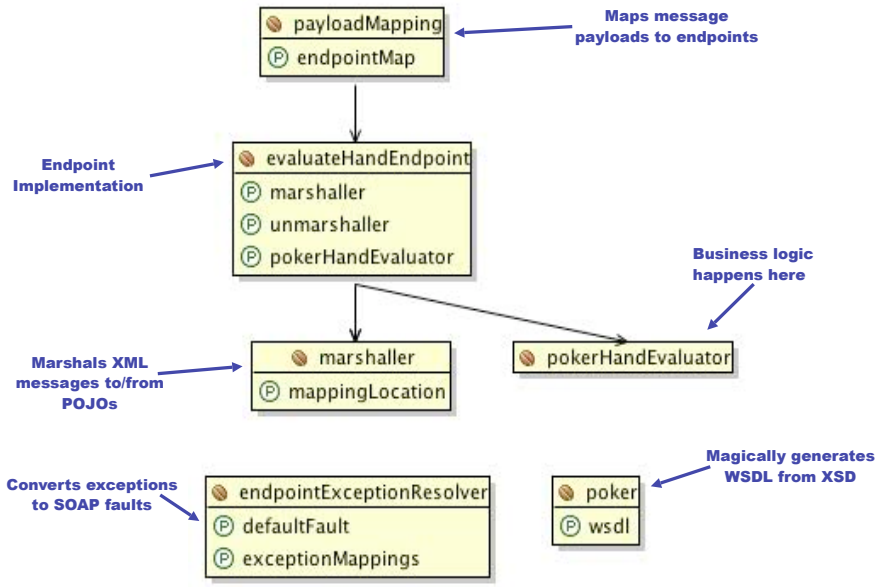
```
package com.springinaction.poker.webservice;
import org.springframework.ws.server.endpoint.annotation.Endpoint;
import org.springframework.ws.server.endpoint.annotation.PayloadRoot;
import com.springinaction.poker.PokerHand;
import com.springinaction.poker.PokerHandEvaluator;
import com.springinaction.poker.PokerHandType;

@Endpoint
public class EvaluateHandAnnotatedEndpoint {
   @PayloadRoot(namespace = "http://www.springinaction.com/poker/schemas",
                    localPart = "EvaluateHandRequest")
   public EvaluateHandResponse evaluateHand(EvaluateHandRequest request) {
      PokerHandType handType = pokerHandEvaluator.evaluateHand(new PokerHand(
                        request.getHand()));
      return new EvaluateHandResponse(handType);
   }

   // injected
   private PokerHandEvaluator pokerHandEvaluator;
   public void setPokerHandEvaluator(PokerHandEvaluator pokerHandEvaluator) {
      this.pokerHandEvaluator = pokerHandEvaluator;
   }
}
```

Spring-Loaded

E-mail: craig@habuma.com Blog: http://www.springloaded.info Source Code: svn://svn.geekisp.com/sia/trunk/Chapter09/Poker-WS

Wiring it all together

The big picture



MessageDispatcherServlet

- Spring-WS is based on Spring MVC
- Must configure a DispatcherServlet in web.xml
  - More specifically: MessageDispatcherServlet

```
<servlet>
  <servlet-name>poker</servlet-name>
  <servlet-class>
    org.springframework.ws.transport.http.MessageDispatcherServlet
  </servlet-class>
  <init-param>
    <param-name>transformWsdlLocations</param-name>
    <param-value>true</param-value>
  </init-param>
</servlet>

<servlet-mapping>
  <servlet-name>poker</servlet-name>
  <url-pattern>/services/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>poker</servlet-name>
  <url-pattern>*.wsdl</url-pattern>
</servlet-mapping>
```


Endpoint mapping

- Helps message dispatcher decide where messages should be sent
- Two kinds:
  - PayloadRootQNameEndpointMapping
  - SoapActionEndpointMapping

```
<bean id="payloadMapping"
    class="org.springframework.ws.server.endpoint.mapping.PayloadRootQNameEndpointMapping">
  <property name="endpointMap">
    <map>
      <entry key="{http://www.springinaction.com/poker/schemas}EvaluateHandRequest"
          value-ref="evaluateHandEndpoint" />
    </map>
  </property>
</bean>
```

Annotation endpoint mapping

- If using annotation-based endpoints...

```
<bean class="org.springframework.ws.server.endpoint.mapping.
PayloadRootAnnotationMethodEndpointMapping">
   <property name="order" value="1"/>
 </bean>
```

- If marshaling, you'll need a MarshallingMethodEndpointAdapter

```
<bean class="org.springframework.ws.server.endpoint.adapter.
            MarshallingMethodEndpointAdapter">
   <constructor-arg ref="marshaller"/>
 </bean>
```


Spring-Loaded

Wiring the endpoint

```
<bean id="evaluateHandEndpoint" class=
      "com.springinaction.poker.webservice.EvaluateHandJDomEndpoint">
  <property name="pokerHandEvaluator" ref="pokerHandEvaluator" />
</bean>

<bean id="pokerHandEvaluator"
    class="com.springinaction.poker.PokerHandEvaluatorImpl"/>
```


Spring-Loaded

A marshaling endpoint

```
<bean id="evaluateHandEndpoint"
   class="com.springinaction.poker.webservice.
                        EvaluateHandMarshallingEndpoint">
  <property name="marshaller"
      ref="marshaller" />
  <property name="unmarshaller"
      ref="marshaller" />
  <property name="pokerHandEvaluator"
      ref="pokerHandEvaluator" />
</bean>
```

SPRING-LOADED

An (un)marshaler: Castor

```
<bean id="marshaller" class=
    "org.springframework.oxm.castor.CastorMarshaller">
  <property name="mappingLocation"
      value="classpath:mapping.xml" />
</bean>
```

SPRING-LOADED

E-mail: craig@habuma.com Blog: http://www.springloaded.info Source Code: svn://svn.geekisp.com/sia/trunk/Chapter09/Poker-WS

What about exceptions?

- How are they mapped to SOAP faults?
- SoapFaultMappingExceptionResolver

```
<bean id="endpointExceptionResolver”  class=
  "org.springframework.ws.soap.server.endpoint.SoapFaultMappingExceptionResolver">
  <property name="defaultFault"
      value="RECEIVER,Server error" />
  <property name="exceptionMappings">
    <props>
      <prop key="org.springframework.oxm.UnmarshallingException">
           SENDER,Invalid request</prop>
      <prop key="org.springframework.oxm.ValidationFailureException">
           SENDER,Invalid request</prop>
    </props>
  </property>
</bean>
```

SPRING-LOADED

Loose-end: Service Locations

- The service location is hard-coded in WSDL
  - What if the service is deployed somewhere other than localhost?
  - SimpleWsdl11Definition serves WSDL, transforming locations to match request's server and context

```
<bean id="poker" class=
        "org.springframework.ws.wsdl.wsdl11.SimpleWsdl11Definition">
    <property name="wsdl" value="/PokerService.wsdl"/>
</bean>
```

SPRING-LOADED

Or...

- DynamicWsdl11Definition automatically generates WSDL from the message XSD

```
<bean id="poker" class=
     "org.springframework.ws.wsdl.wsdl11.DynamicWsdl11Definition">
  <property name="builder">
    <bean class="org.springframework.ws.wsdl.wsdl11.builder.
                    XsdBasedSoap11Wsdl4jDefinitionBuilder">
      <property name="schema" value="/PokerTypes.xsd"/>
      <property name="portTypeName" value="Poker"/>
      <property name="locationUri"
              value="http://localhost:8080/Poker-WS/services"/>
    </bean>
  </property>
</bean>
```


Consuming Spring-WS Services

Writing clients

- Proxy-based client APIs typically won't work with Spring-WS
  - Proxies are method-centric and very RPC-ish in nature
- What does work...
  - WSDL2Java-generated client stubs
  - Anything that can wrap supplied XML in a SOAP wrapper
  - Spring-WS client-side templates

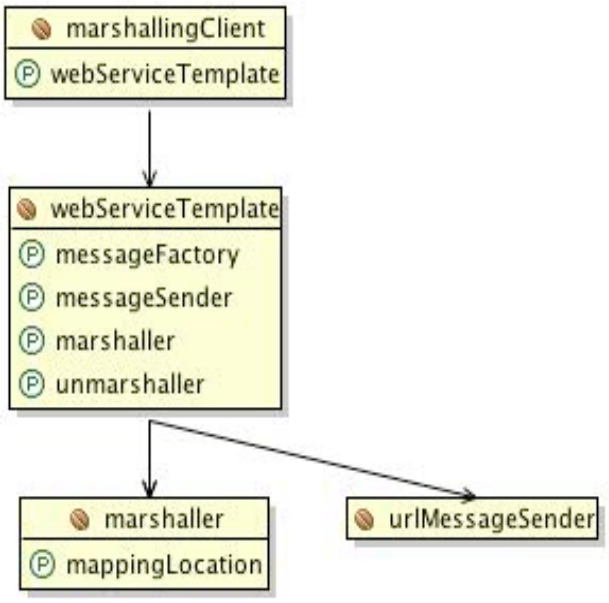
Using Spring-WS' client API

- Template-based
  - Like Spring JDBC, Spring JMS, etc
- Choices to make
  - To marshal or not to marshal?
  - Basic template or gateway support?

Spring-WS Client API

- WebServiceTemplate – Centerpiece of Spring-WS client API
  - Similar in concept to Spring's JDBC Template
- Message factory – Produces message
- Message sender – Sends message
- (Un)Marshaler (optional) – Converts XML to/from POJO

Spring-WS Client API



E-mail: craig@habuma.com Blog: http://www.springloaded.info Source Code: svn://svn.geekisp.com/sia/trunk/Chapter09/Poker-ws

Spring-Loaded

Template-based client

```
package com.springinaction.ws.client;
import java.io.IOException;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.Namespace;
import org.jdom.transform.JDOMResult;
import org.jdom.transform.JDOMSource;
import org.springframework.ws.client.core.WebServiceTemplate;
import com.springinaction.poker.Card;
import com.springinaction.poker.PokerHandType;

public class TemplateBasedPokerClient implements PokerClient {
   public PokerHandType evaluateHand(Card[] cards) throws IOException {
       // DETAILS OF THIS METHOD ARE ON THE NEXT SLIDE!!!
   }

   // INJECTED
   private WebServiceTemplate webServiceTemplate;
   public void setWebServiceTemplate(WebServiceTemplate webServiceTemplate) {
      this.webServiceTemplate = webServiceTemplate;
   }
}
```

SPRING-LOADED

Template-based client (2)

```
public PokerHandType evaluateHand(Card[] cards) throws IOException {
  Element requestElement = new Element("EvaluateHandRequest");
  Namespace ns = Namespace.getNamespace(
          "http://www.springinaction.com/poker/schemas");
  requestElement.setNamespace(ns);
  Document doc = new Document(requestElement);

  for (int i = 0; i < cards.length; i++) {
     Element cardElement = new Element("card");
     Element suitElement = new Element("suit");
     suitElement.setText(cards[i].getSuit().toString());
     Element faceElement = new Element("face");
     faceElement.setText(cards[i].getFace().toString());
     cardElement.addContent(suitElement);
     cardElement.addContent(faceElement);
     doc.getRootElement().addContent(cardElement);
  }

  JDOMResult result = new JDOMResult();
  webServiceTemplate.sendSourceAndReceiveToResult(new JDOMSource(doc), result);

  Document resultDocument = result.getDocument();
  Element responseElement = resultDocument.getRootElement();
  Element handNameElement = responseElement.getChild("handName", ns);
  return PokerHandType.valueOf(handNameElement.getText());
}
```

SPRING-LOADED

Template-based client (3)

```
<bean id="webServiceTemplate"
      class="org.springframework.ws.client.core.WebServiceTemplate">
  <property name="defaultUri"
          value="http://localhost:8080/Poker-WS/" />
</bean>

<bean id="templateBasedClient"
    class="com.springinaction.ws.client.TemplateBasedPokerClient">
  <property name="webServiceTemplate" ref="webServiceTemplate" />
</bean>
```


Marshaling client

```
package com.springinaction.ws.client;
import java.io.IOException;
import org.springframework.ws.client.core.WebServiceTemplate;
import com.springinaction.poker.Card;
import com.springinaction.poker.PokerHandType;
import com.springinaction.poker.webservice.EvaluateHandRequest;
import com.springinaction.poker.webservice.EvaluateHandResponse;

public class MarshallingPokerClient
    implements PokerClient {
  public PokerHandType evaluateHand(Card[] cards)
      throws IOException {
    EvaluateHandRequest request = new EvaluateHandRequest();

    request.setHand(cards);

    EvaluateHandResponse response = (EvaluateHandResponse)
        webServiceTemplate.marshalSendAndReceive(request);

    return response.getPokerHand();
  }

  // INJECTED
  private WebServiceTemplate webServiceTemplate;
  public void setWebServiceTemplate(WebServiceTemplate webServiceTemplate) {
    this.webServiceTemplate = webServiceTemplate;
  }
}
```

Marshaling client (2)

```
<bean id="webServiceTemplate"
     class="org.springframework.ws.client.core.WebServiceTemplate">
  <property name="marshaller" ref="marshaller" />
  <property name="unmarshaller" ref="marshaller" />
  <property name="defaultUri"
          value="http://localhost:8080/Poker-WS/" />
</bean>

<bean id="marshaller"
     class="org.springframework.oxm.castor.CastorMarshaller">
   <property name="mappingLocation" value="classpath:mapping.xml" />
</bean>

<bean id="marshallingClient"
     class="com.springinaction.ws.client.MarshallingPokerClient">
   <property name="webServiceTemplate" ref="webServiceTemplate" />
</bean>
```


Spring-Loaded

Gateway client

```
package com.springinaction.ws.client;
import java.io.IOException;
import org.springframework.ws.client.core.support.WebServiceGatewaySupport;
import com.springinaction.poker.Card;
import com.springinaction.poker.PokerHandType;
import com.springinaction.poker.webservice.EvaluateHandRequest;
import com.springinaction.poker.webservice.EvaluateHandResponse;

public class PokerClientGateway extends WebServiceGatewaySupport implements PokerClient {
  public PokerHandType evaluateHand(Card[] cards)
      throws IOException {
    EvaluateHandRequest request = new EvaluateHandRequest();

    request.setHand(cards);

    EvaluateHandResponse response = (EvaluateHandResponse)
        getWebServiceTemplate().marshalSendAndReceive(request);

    return response.getPokerHand();
  }
}
```


Spring-Loaded

Gateway Client (2)

```
<bean id="pokerClientGateway"
      class="com.springinaction.ws.client.PokerClientGateway">
  <property name="marshaller" ref="marshaller" />
  <property name="unmarshaller" ref="marshaller" />
  <property name="defaultUri"
          value="http://localhost:8080/Poker-WS/" />
</bean>

<bean id="marshaller"
      class="org.springframework.oxm.castor.CastorMarshaller">
  <property name="mappingLocation"
          value="classpath:mapping.xml" />
</bean>
```


Final thoughts

Creating a Spring-WS service

- Create sample messages
  - Just XML
- Create data contract
  - XSD inferred from XML
- Create endpoints
- Wire it up in Spring

Isn't that a lot of <bean>s?

- Mostly boilerplate
- Can be "pre-declared"
- Rolled into a Maven 2 archetype
- Spring namespace?

What I didn't talk about

- Spring-WS and WS-Security
- Sending POX messages
- RESTful Spring (coming in Spring 3)
- JMS transports
- Mail transports
- WS-Addressing

Recommending reading

- Spring-WS homepage
  - http://static.springsource.org/spring-ws/sites/1.5/
- Arjen Poutsma's web-log
  - http://blog.springframework.com/arjen
  - "Thoughts on Web Services" - 6/20/2005 - 6/28/2005
- Spring in Action, 2E; Chapter 9

Q & A

Thank you