APIs Java para Web Services

Ivan Neto

Roteiro

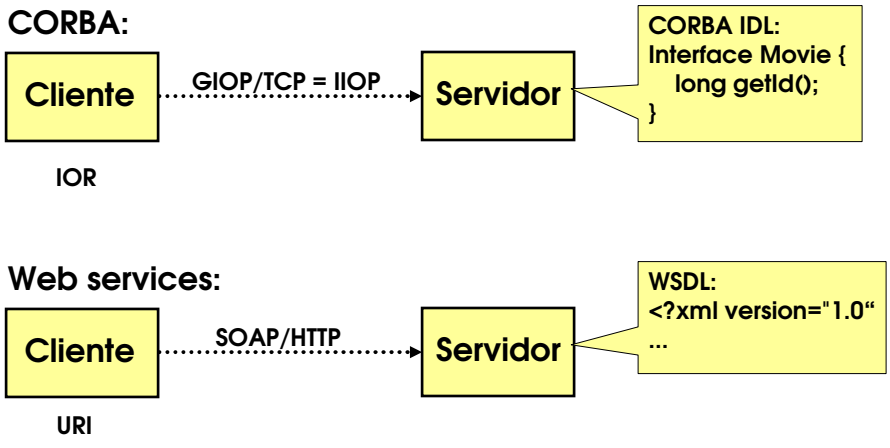
- Revisão
- Web services em Java
- Como implementar um web service
- Como acessar um web service
- Implementando um cliente J2EE
- Java EE 5

O que é um web service

- Uma aplicação
  - Identificada por uma URI
  - Interfaces descritas em XML
  - Interações via mensagens XML
    - Protocolos da Internet (ex. HTTP).

3

Comparação com CORBA



4

Implementando um web service

- Como implementar um web service?
- Depende!
  - Plataforma/infra-estrutura
  - Não há API padrão
    - Em CORBA a API é padronizada

```
module CORBA {
    interface ORB {
        string object_to_string(in Object obj);
        Object string_to_object(in string str);
        ...
    };
    ...
};
```

5

Web services em Java

- A Sun criou uma API Java para web services
  - Java API for XML-based Remote Procedure Call (JAX-RPC)
- A Sun também criou uma especificação web services para Java EE
  - JSR 109: Implementing Enterprise Web Services

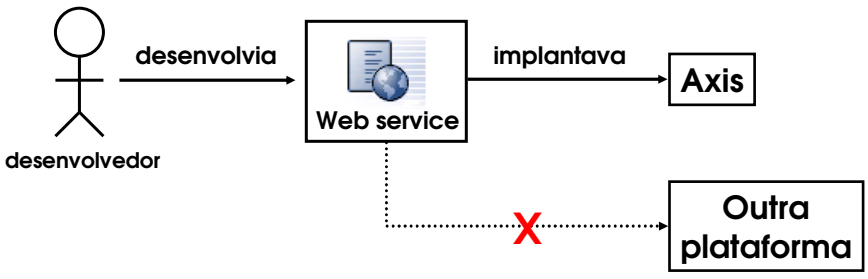
6

JSR-109?

- Mas para quê outra especificação?
  - Já não basta a API (JAX-RPC)?
- JSR-109
  - Portabilidade em ambientes J2EE
  - Define
    - Descritores de implantação
    - Empacotamento
    - Endpoints EJB

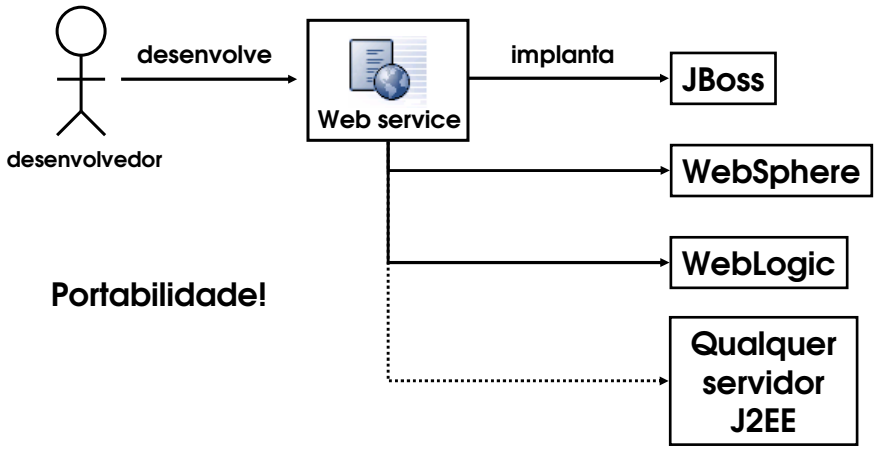
7

Antes da JSR-109



8

Com a JSR-109



9

JAX-RPC

- Define basicamente três coisas:
  - Mapeamento XML <-> Java
  - Mapeamento WSDL <-> Java
  - Modelo de programação/APIs
    - Cliente
    - Servidor

10

JAX-RPC: Mapeamento XML<->Java

```
<complexType name="MovieInfo">
  <sequence>
    <element name="Id" type="xsd:int" />
    <element name="Name" type="xsd:string" nillable="true" />
    <element name="Year" type="xsd:short" />
    ...
  </sequence>
</complexType>
```

XML

```
public class MovieInfo {
    protected int id;
    protected java.lang.String name;
    protected short year;
    ... // Getters & setters
}
```

Java

11

Comparando com CORBA

```
struct MovieInfo {
  long id;
  string name;
  short year;
  ...
};
```

IDL

```
public final class MovieInfo implements IDLEntity {
  public int id;
  public java.lang.String name = "";
  public short year;
  ...
}
```

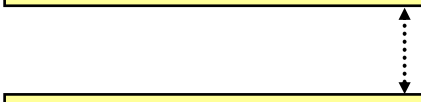
Java

12

JAX-RPC: Mapeamento WSDL<->Java

```
<portType name="RentalServiceRPC">
  <operation name="endRental" parameterOrder="movieCopyId">
    <input message="tns:RentalServiceRPC_endRental" />
    <output message="tns:RentalServiceRPC_endRentalResponse" />
  </operation>
  ...
</portType>
```

WSDL



```
public interface RentalServiceRPC extends java.rmi.Remote {
  public void endRental(int movieCopyId) throws RemoteException;
  ...
}
```

Java

13

Comparando com CORBA

```
interface RentalService {
  void endRental(in long movieCopyId);
  ...
};
```

IDL

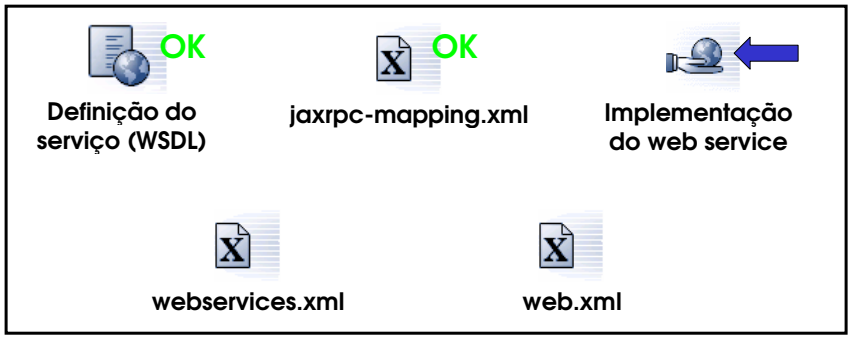
```
public interface RentalServiceOperations {
  void endRental(int movieCopyId);
  ...
}
```

Java

14

Criando um web service

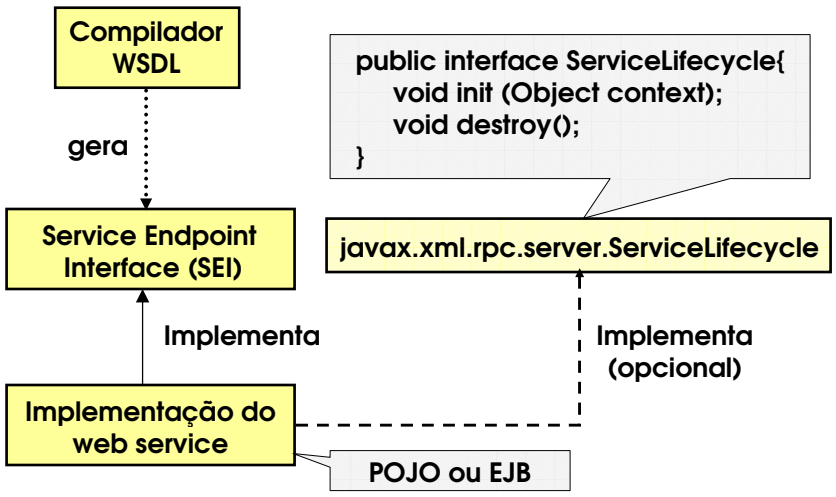
Modelo de programação do lado servidor:



Módulo J2EE

15

Implementação do web service



16

Exemplo: EP3 (1 de 2)

```
<portType name="RentalServiceRPC">
  <operation name="endRental" parameterOrder="movieCopyId">
    <input message="tns:RentalServiceRPC_endRental" />
    <output message="tns:RentalServiceRPC_endRentalResponse" />
  </operation>
  ...
</portType>
```

WSDL

Compilador WSDL

```
public interface RentalServiceRPC extends java.rmi.Remote {
  public void endRental(int movieCopyId) throws RemoteException;
  ...
}
```

Java SEI

17

Exemplo: EP3 (2 de 2)

Classe de implementação

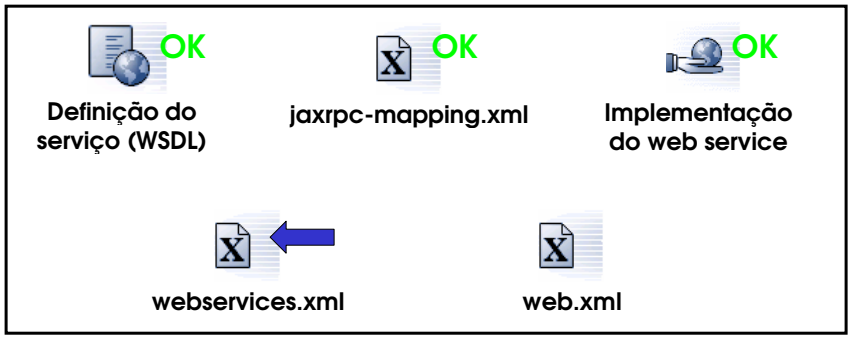
- Implementa Service Endpoint Interface (SEI)
- Poderia implementar ServiceLifeCycle

```
public class RentalServiceRPCImpl implements RentalServiceRPC {

   public void endRental(int movieCopyId) throws RemoteException {
      // Implementação do método
   }

   public MovieInfoArray findMovieByDirector(String name) throws
                                                    RemoteException  {
      // Implementação do método
   }
   ...
}
```

18

Criando um web service

Modelo de programação do lado servidor:



Módulo J2EE

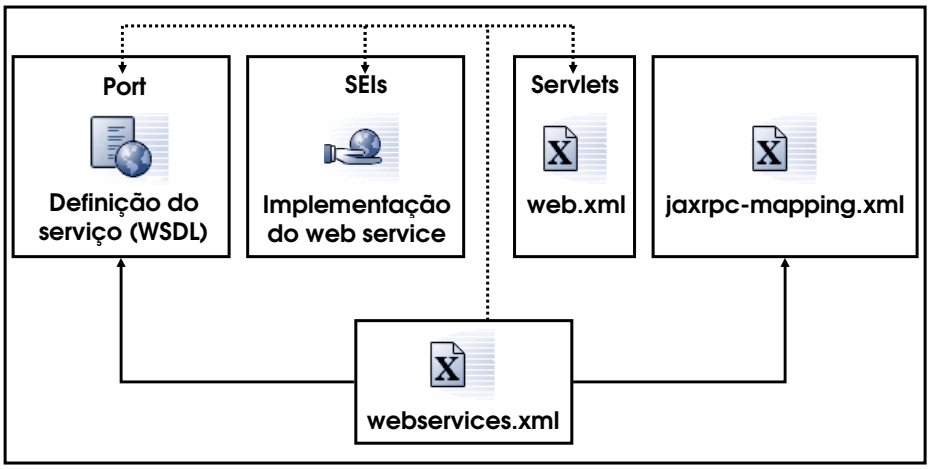
19

webservices.xml

- Fornece informações estruturais sobre o web service
- Funciona como uma espécie de "cola"
  - Relaciona os artefatos

20

webservices.xml



21

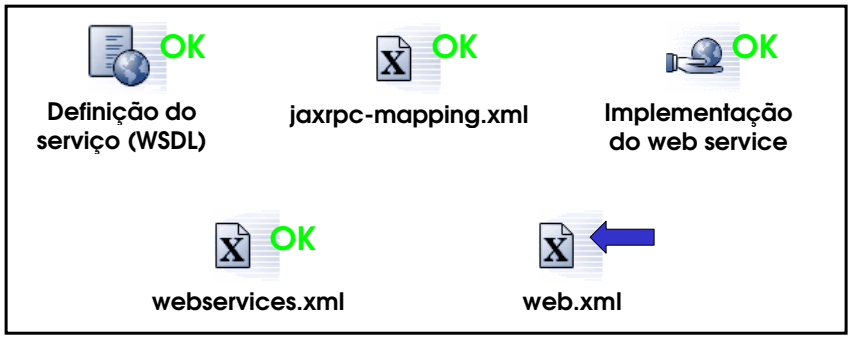
webservices.xml: Exemplo

```
<webservices>
 <webservice-description>
  <webservice-description-name>VideoRentalStore</webservice-description-name>
  <wsdl-file>WEB-INF/wsdl/VideoRental.wsdl</wsdl-file>
  <jaxrpc-mapping-file>WEB-INF/jaxrpc-mapping.xml</jaxrpc-mapping-file>
  <port-component>
   <port-component-name>RentalServiceRPCPort</port-component-name>
   <wsdl-port>sod:RentalServiceRPCPort</wsdl-port>
   <service-endpoint-interface>videorentalstore.webservice.RentalServiceRPC
   </service-endpoint-interface>
   <service-impl-bean>
    <servlet-link>RentalServiceRPCServlet</servlet-link>
   </service-impl-bean>
  </port-component>
  <port-component>
   ...
 </webservice-description>
</webservices>
```

22

Criando um web service

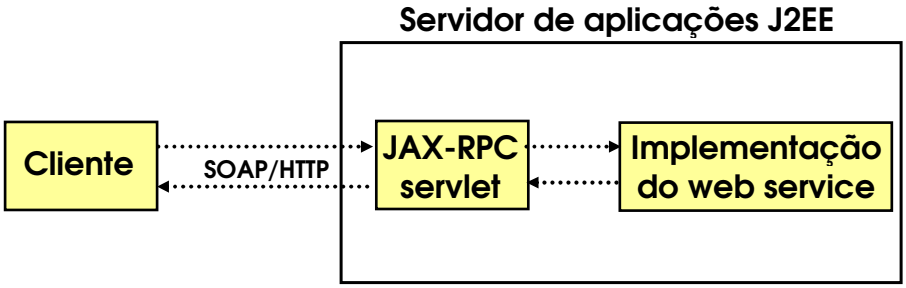
Modelo de programação do lado servidor:



Módulo J2EE

23

Por que um servlet?



Servlet

- Objeto que recebe uma requisição (ServletRequest) e gera uma resposta (ServletResponse)
- HttpServlet

24

web.xml

```
<web-app>
  <servlet>
    <servlet-name>RentalServiceRPCServlet</servlet-name>
    <servlet-class>videorentalstore.webservice.impl.RentalServiceRPCImpl
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>RentalServiceRPCServlet</servlet-name>
    <url-pattern>/rentalservicerpc</url-pattern>
  </servlet-mapping>
  ...
</web-app>
```

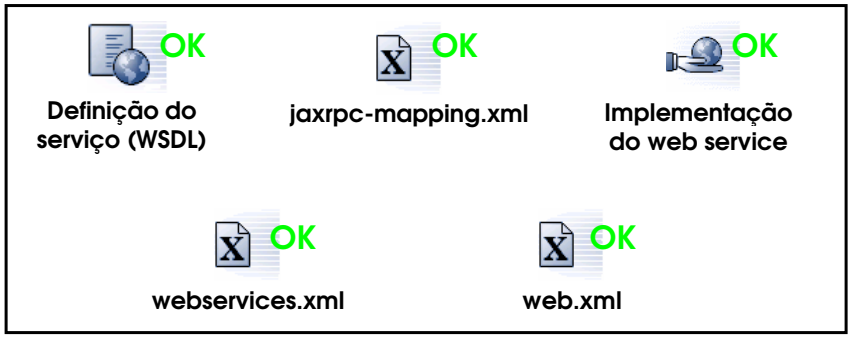
Não implementa javax.servlet.Servlet

Endereço do web service

- RentalServiceRPCImpl não é um servlet!
- Endereço: http://localhost:8080/${contextpath}/rentalservicerpc

Criando um web service

Modelo de programação do lado servidor:



Módulo J2EE

Empacotando o web service

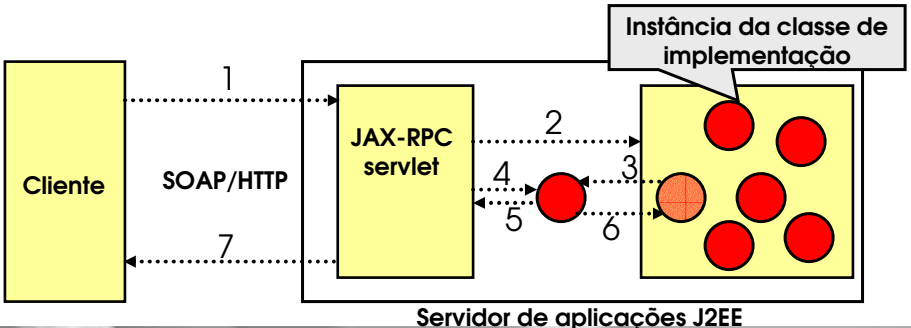
```
videorental.war
|-- WEB-INF
|    |-- classes
|    |    |-- (as classes Java compiladas vêm aqui)
|    |-- wsdl
|    |    |-- VideoRental.wsdl
|    |-- jaxrpc-mapping.xml
|    |-- web.xml
|    |-- webservices.xml
```

Basta implantar o web archive em qualquer servidor de aplicações J2EE e seu web service está acessível.

27

Quem instancia o web service?

- É o container
  - Containers têm liberdade para otimizações (ex. pool de instâncias)



28

Modelos de programação do cliente

- Há três modos de se acessar um web service:
  - Stubs estáticos (como em CORBA)
    - Não portáveis
  - Dynamic Invocation Interface (DII)
  - Stubs gerados em tempo de execução (proxies dinâmicos)

29

Dynamic Invocation Interface

- É o modo de invocação mais flexível
  - Pode chamar operações descobertas em tempo de execução
  - Não usa nenhum tipo de proxy, usa apenas uma API fixa
- É difiícil
  - De escrever e ler
  - Erros não são descobertos em tempo de compilação

30

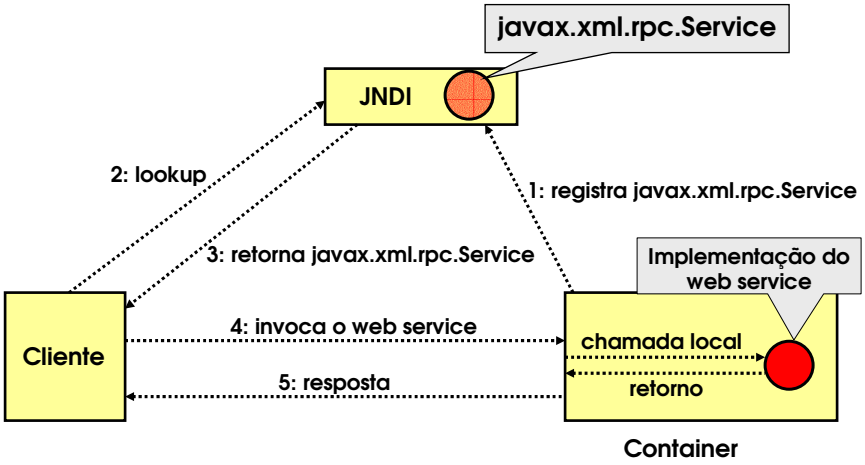
Dynamic Invocation Interface

```
Service service = ... //get Service
Call call = service.createCall();
QName operationName = new QName(TARGET_NAMESPACE, "echoString");
call.setOperationName(operationName);
String hello = "Hello";
String world = "world!";
Object retObj = call.invoke(new Object(){hello, world});
```

```
proxy.echoString(“Hello”, “world!”);
```

31

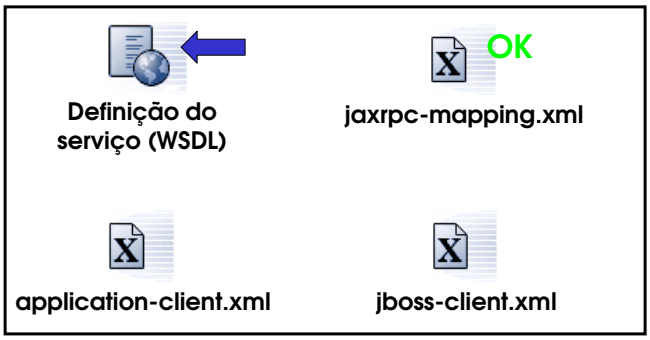
Um cliente J2EE



32

Criando um cliente J2EE

Modelo de programação do lado cliente:



Módulo J2EE

33

Um detalhe sobre o arquivo WSDL

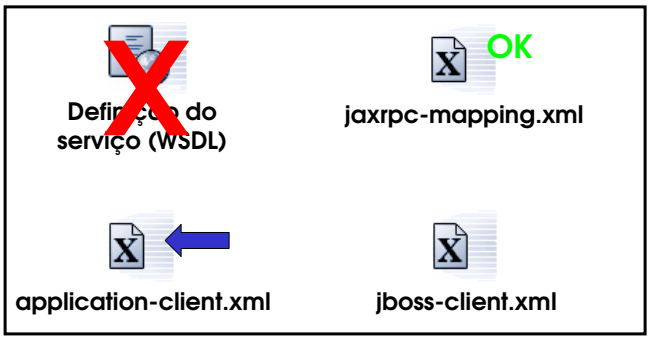
- O endereço dos ports não está presente neste arquivo
  - O JBoss preenche os endereços em tempo de implantação

```
<service name="VideoRentalStoreService">
  <port name="RentalServiceRPCPort" binding="tns:RentalServiceRPCBinding">
    <soap:address location="REPLACE_WITH_ACTUAL_URL" />
  </port>
  <port name="RentalServiceDocPort" binding="tns:RentalServiceDocBinding">
    <soap:address location="REPLACE_WITH_ACTUAL_URL" />
  </port>
</service>
```

34

Criando um cliente J2EE

Modelo de programação do lado cliente:



Módulo J2EE

35

application-client.xml

- Define referências para componentes J2EE
  - EJBs
  - Web services
- Seu cliente “enxerga” esses componentes

36

application-client.xml

```
<applic
 <displa                                    ent<
 <service-ref>
  <service-ref-name>service/videorental</service-ref-name>
  <service-interface>videorentalstore.webservice.VideoRentalStoreService
  </service-interface>
  <wsdl-file>USE_JBOSS_CLIENT_XML_OVERRIDE</wsdl-file>
  <jaxrpc-mapping-file>META-INF/jaxrpc-mapping.xml</jaxrpc-mapping-file>
  <port-component-ref>
   <service-endpoint-interface>videorentalstore.webservice.RentalServiceRPC
   </service-endpoint-interface>
  </port-component-ref>
  <port-component-ref>
   <service-endpoint-interface>videorentalstore.webservice.RentalServiceDoc
   </service-endpoint-interface>
  </port-component-ref>
 </service-ref>
</application-client>
```

Nome no JNDI (relativo a java:comp/env/)

Tem que ser um javax.xml.rpc.Service

Não é o WSDL

37

javax.xml.rpc.Service

```
public interface VideoRentalStoreService extends javax.xml.rpc.Service {
    public RentalServiceRPC getRentalServiceRPCPort() throws ServiceException;
    public RentalServiceDoc getRentalServiceDocPort() throws ServiceException;
}
```

Cliente quando service-interface é videorentalstore.webservice.VideoRentalStoreService:

```
VideoRentalStoreService service = // get VideoRentalStoreService inst. (JNDI lookup)
RentalServiceRPC port = service.getRentalServiceRPCPort();
```

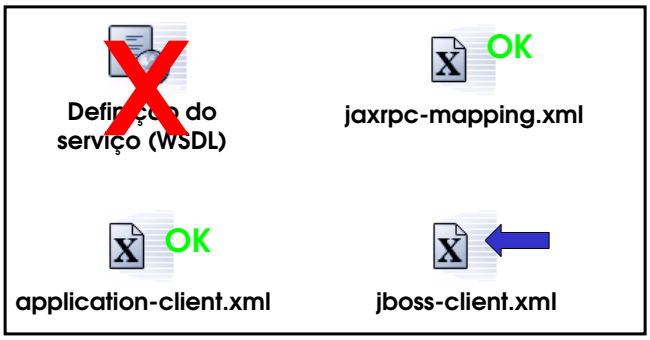
Cliente quando service-interface é javax.xml.rpc.Service:

```
Service service = // get Service instance (JNDI lookup)
RentalServiceRPC port = (RentalServiceRPC) service.getPort(RentalServiceRPC.class);
```

38

Criando um cliente J2EE

Modelo de programação do lado cliente:



Módulo J2EE

39

jboss-client.xml

Têm que ser igual a propriedade j2ee.clientName do jndi.properties

```
<jboss-client>
  <jndi-name>jbossws-client</jndi-name>
  <service-ref>
    <service-ref-name>service/videorental</service-ref-name>
    <wsdl-override>http://${jboss.bind.address}:8080/videorental/rentalservicerpc?wsdl
    </wsdl-override>
  </service-ref>
</jboss-client>
```

Nome do application-client.xml

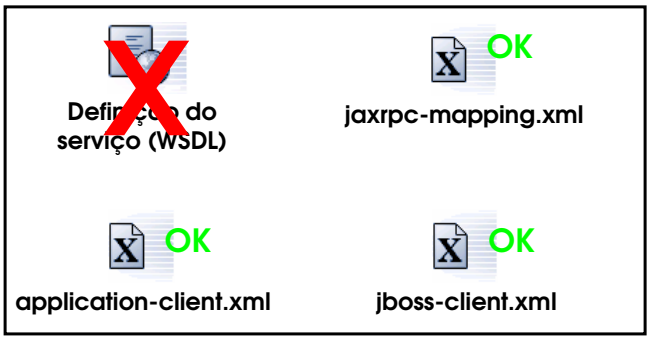
Localização do WSDL

Tudo isso é específico do JBoss!

40

Criando um cliente J2EE

Modelo de programação do lado cliente:



Módulo J2EE

41

Empacotando o cliente J2EE

```
videorental-client.jar
|-- META-INF
|    |-- jaxrpc-mapping.xml
|    |-- application-client.xml
|    |-- jboss-client.xml
```

Implantando o seu cliente J2EE fará com que o JBoss registre um proxy para o web service no JNDI.

42

Escrevendo o cliente

Como é que o cliente sabe onde está o serviço de nomes?

```
public static void main(String() args) {
   Context initialContext = new InitialContext();
   VideoRentalStoreService service = (VideoRentalStoreService) initialContext
         .lookup("java:comp/env/service/videorental");
   RentalServiceRPC rpcPort = (RentalServiceRPC) service.getRentalServiceRPCPort();
   MovieInfo info = rpcPort.findMovieById(1);
...
}
```

Nome definido no application-client.xml

jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming.client:org.jboss.naming
java.naming.provider.url=jnp://localhost:1099
j2ee.clientName=jbossws-client
```

Igual ao jndi-name do jboss-client.xml

43

Java EE 5

- A API para web services do Java EE 5 é a JAX-WS 2.0 (JAX-RPC 2.0)
  - JAX-RPC 1.1 continua sendo suportada
  - JAX-WS será parte do Java SE 1.6
- Uso de anotações do Java 5 (JSR-181)
- Mais fácil de programar
- Por que não usamos JAX-WS?
  - Recente (maio de 2006)
  - JBoss ainda não suporta a JAX-WS

44

Um web service em Java EE 5

```
@WebService(
    name = "EndpointInterface",
    targetNamespace = "http://org.jboss.ws/samples/jsr181pojo",
    serviceName = "TestService")
  @SOAPBinding(style = SOAPBinding.Style.RPC)
  public class JSEBean01
  {
    @WebMethod
    public String echo(String input)
    {
      return input;
    }
  }
```

Só essa classe + web.xml!

45

APIs para web services no Java EE 5

- Java API for XML Web Services (JAX-WS)
- Java API for XML-RPC (JAX-RPC)
- Java Architecture for XML Binding (JAXB)
- SOAP with Attachments for Java (SAAJ)

46

Especificações

- WSDL (1.2 e 2.0)
- SOAP (1.1 e 1.2)
- UDDI
- WS-I Basic Profile
- JSR-109 (Implementing Enterprise Web Services)
- JSR-181 (Web Services Metadata for the Java Platform)

47

WS-I Basic Profile

- As especificações nem sempre são claras
- O Basic Profile é um conjunto de recomendações que visam maximizar a interoperabilidade

R2007 A DESCRIPTION MUST specify a non-empty location attribute on the wsdl:import element.

R1025 A RECEIVER MUST handle messages in such a way that it appears that all checking of mandatory header blocks is performed before any actual processing.

48

Especificações WS-*

- WS-Addressing
  - Message IDs, ReplyTo, EndpointReference
- WS-Coordination
- WS-AtomicTransaction
- WS-BusinessActivity
- WS-Security
- + um monte de especificações

49

Plataformas para web services

- Apache Axis
  - Axis é compatível com a JAX-RPC 1.1
  - Axis2 não é nem com JAX-RPC nem com JAX-WS!
- XFire
- Servidores de aplicações J2EE (JSR-109)
  - JBoss, WebSphere (IBM), WebLogic (BEA), Geronimo (Apache), etc

50

Web services no JBoss

- Até a versão 4.0.3
  - Axis 1.x
- A partir da versão 4.0.4
  - JBossWS
    - Mais direcionado às especificações Java EE
    - Melhor integração com a arquitetura do JBoss
    - Implementação "do zero"!

51

Referências

- Monson-Haefel, R. J2EE Web services. Addison-Wesley, Boston, 2004.
- http://www-128.ibm.com/developerworks/webservices/library/ws-jsrart/

52