



COMDEX

SUCESU-SP 2002

Como implementar
Web Services
em **Java**

Helder da Rocha

www.argonavis.com.br

- Falar sobre Web Services...
 - Definir **Web Services**
 - Descrever as **tecnologias XML** padrão que oferecem suporte a Web Services
 - Descrever as **APIs Java** distribuídas com o Java Web Services Development Pack 1.0
- ... e mostrar como criar um
 - Utilizar a API **JAX-RPC** para desenvolver e implantar um Web Service simples baseado no protocolo **SOAP**
 - Gerar uma interface **WSDL** e utilizá-la para construir um cliente para o serviço
 - Registrar uma organização e a localização do arquivo WSDL em um servidor **UDDI** local

- **Helder da Rocha**
 - *Instrutor, consultor, desenvolvedor e autor de programas de treinamento em Java e XML*
 - *Usando Java desde 1995*
 - *Foco atual em tecnologias de computação distribuída, XML, metodologias ágeis e novos paradigmas*

- Ambiente de computação distribuída (DCE) que utiliza **XML** em todas as camadas
 - No formato de dados usado na **comunicação**
 - Na **interface** usada para descrever as operações suportadas
 - Na aplicação usada para **registrar** e **localizar serviços**
- Serviços são transportados principalmente via HTTP
 - Podem também utilizar outros protocolos populares
- Web Services visam comunicação entre **máquinas**
 - Serviços podem ser implementados usando CGI (com C, Perl, etc.), ASP, PHP, servlets, JSP, CFML, etc.
 - Acesso é feito via clientes HTTP (ou de outros protocolos)
- Tudo isto já existia! Qual a novidade?

A novidade é a padronização!

SUCESU-SP 2002

- **Todas as camadas em XML!**
 - Fácil de ler, transformar, converter
 - Existe ainda um esforço para padronizar os esquemas que definem a estrutura e vocabulário do XML usado
- **Web Services dá nova vida ao RPC**
 - Agora com formato universal para os dados!
 - ➔ **Marshalling**: converter dados em XML
 - ➔ **Unmarshalling**: extrair dados de XML
- **Principais características do RPC com Web Services**
 - Formato padrão de dados usados na comunicação é **XML**
 - **Interoperabilidade** em todos os níveis
 - Transporte é protocolo de larga aceitação: HTTP, SMTP, ...
 - Transparência de localidade e neutralidade de linguagem

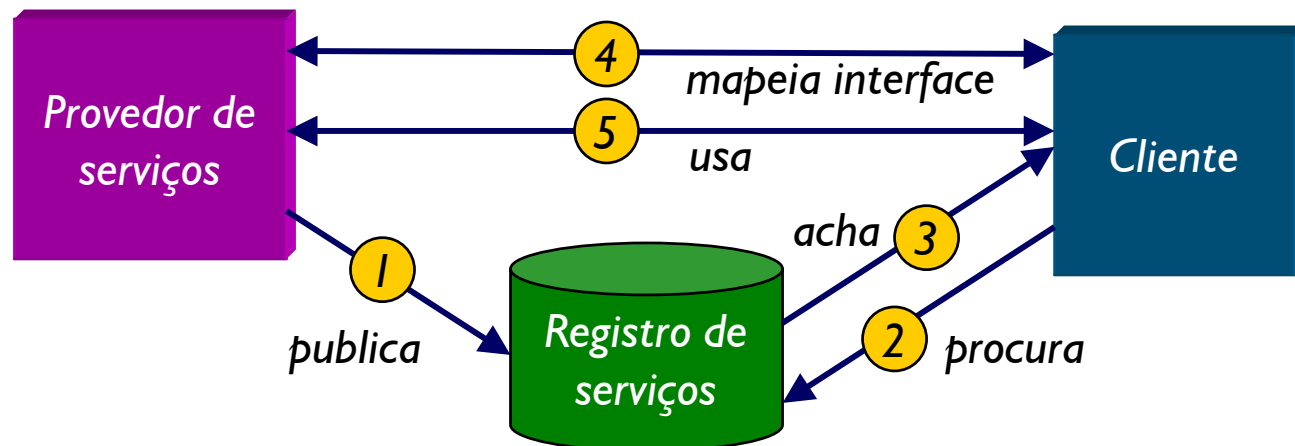
Arquitetura de Web Services: camadas

- **Camada de transporte**
 - Principais: HTTP (POST), FTP, SMTP
 - Emergentes: JRMP (Java RMI), IIOP (CORBA, EJB), JMS, IMAP, POP, BEEP, JXTA, ...
- **Camada de mensagens**
 - SOAP
- **Camada dados ou serviços**
 - XML (formato de mensagens)
 - XML-RPC
- **Camada de descrição de serviços**
 - WSDL
- **Camada de descoberta (registro)**
 - UDDI, ebXML



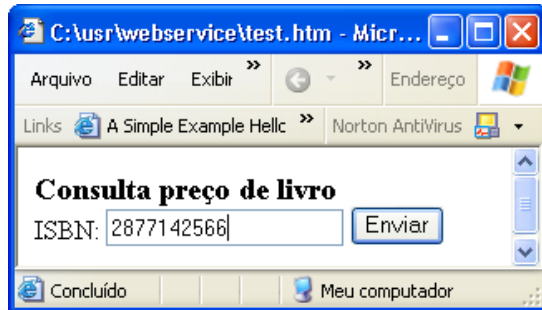
Arquitetura de Web Services: papéis

- **Provedor de serviços**
 - Oferece serviços, alguns dos quais podem ser Web Services
- **Registro de serviços**
 - Catálogo de endereços: repositório central que contém informações sobre web services
- **Cliente de serviços**
 - Aplicação que descobre um web service, implementa sua interface de comunicação e usa o serviço



Requisição e resposta HTTP POST

- Clientes HTTP usam o método POST para **enviar** dados
 - Tipicamente usado por browsers para enviar dados de formulários HTML e fazer upload de arquivos
- Exemplo
 - Formulário HTML



```
<FORM ACTION="/cgi-bin/catalogo.pl"
METHOD="POST">
```

```
<H3>Consulta preço de livro</H3>
```

```
<P>ISBN: <INPUT TYPE="text" NAME="isbn">
```

```
<INPUT TYPE="Submit" VALUE="Enviar">
```

```
</FORM>
```

- Requisição POST gerada pelo browser para o servidor

Cabeçalho HTTP

Linha em branco

Mensagem (corpo da requisição)

```
POST /cgi-bin/catalogo.pl HTTP/1.0
Content-type: text/x-www-form-urlencoded
Content-length: 15
```

```
isbn=2877142566
```

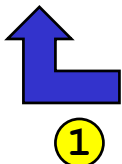
Enviando XML sobre POST

- Você pode criar um serviço RPC simples (um Web Service!) trocando mensagens XML via HTTP POST!
 - Defina esquemas para as mensagens de chamada e resposta
 - Escreva cliente que envie requisições POST para servidor Web
 - Escreva uma aplicação Web (JSP, ASP, PHP, servlet, CGI)

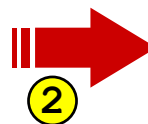
```
POST /ISBNService.jsp HTTP/1.0
Content-type: text/xml
Content-length: 90
```

```
<chamada>
  <funcao>
    <nome>getPrice</nome>
    <param>2877142566</param>
  </funcao>
</chamada>
```

gera
requisição



ISBNClient



ISBNService.jsp

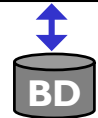
2877142566

ISBNQuery
getPrice()

19.50



gera
resposta



```
HTTP/1.1 200 OK
Content-type: text/xml
Content-length: 77
```

```
<resposta>
  <funcao>
    <param>19.50</param>
  </funcao>
</resposta>
```





- *Especificação para RPC em XML via HTTP POST*
 - *Projetada para ser a solução mais simples possível*
 - *Várias implementações: veja www.xml-rpc.com*
- *Exemplo anterior implementado com XML-RPC (cabeçalhos HTTP omitidos)*

```
<methodCall>
  <methodName>getPrice</methodName>
  <params>
    <param>
      <value><string>2877142566</string></value>
    </param>
  </params>
</methodCall>
```

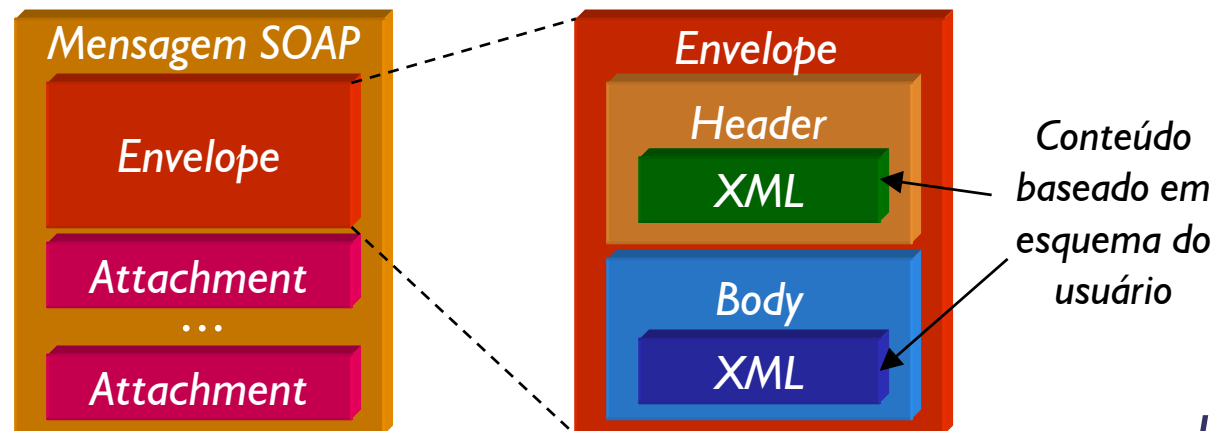
Requisição ←

```
<methodResponse>
  <params>
    <param>
      <value><double>19.5</double></value>
    </param>
  </params>
</methodResponse>
```

Resposta →

- **S**imple **O**bject **A**ccess **P**rotocol
- Protocolo padrão baseado em XML para trocar **mensagens** entre aplicações
 - SOAP não é um protocolo RPC, mas um par de mensagens SOAP pode ser usado para esse fim
 - Transporte pode ser HTTP, SMTP ou outro
 - Mensagens podem conter qualquer coisa (texto, bytes)
 - É extensível (mecanismo de RPC, por exemplo, é extensão)

Estrutura de uma mensagem SOAP




Simple requisição SOAP-RPC

- Principal aplicação do SOAP, hoje, é RPC sobre HTTP
 - Esquema do corpo da mensagem lida com RPC

```
POST /xmlrpc-bookstore/bookpoint/BookstoreIF HTTP/1.0
Content-Type: text/xml; charset="utf-8"
Content-Length: 585
SOAPAction: ""
```

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
  env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <env:Body>
    <ans1:getPrice xmlns:ans1="http://mybooks.org/wsdl">
      <String_1 xsi:type="xsd:string">2877142566</String_1>
    </ans1:getPrice>
  </env:Body>
</env:Envelope>
```


Parâmetro (ISBN)

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
SOAPAction: ""
Date: Thu, 08 Aug 2002 01:48:22 GMT
Server: Apache Coyote HTTP/1.1 Connector [1.0]
Connection: close
```

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<env:Envelope
```

```
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
```

```
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
```

```
  xmlns:ns0="http://mybooks.org/types"
```

```
  env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

```
<env:Body>
```

```
  <ans1:getPriceResponse xmlns:ans1="http://mybooks.org/wsdl">
```

```
    <result xsi:type="xsd:decimal">19.50</result>
```

```
  </ans1:getPriceResponse>
```

```
</env:Body>
```

```
</env:Envelope>
```



Resposta (Preço)

Descrição de um serviço RPC: WSDL

- *Para saber usar um Web Service, é preciso*
 - *Saber o que um serviço faz (quais as operações?)*
 - *Como chamar suas operações (parâmetros? tipos?)*
 - *Como encontrar o serviço (onde ele está?)*
- **Web Services Description Language**
 - *Documento XML de esquema padrão que contém todas as informações necessárias para que um cliente possa utilizar um Web Service*
 - *Define informações básicas (operações, mapeamentos, tipos, mensagens, serviço) e suporta extensões*
 - *Tem basicamente mesmo papel que linguagens IDL usadas em outros sistemas RPC*
 - *Pode ser usada na geração automática de código*

- WSDL serve apenas para **descrever interfaces**
 - Não serve para ser executada
 - Nenhuma aplicação **precisa** da WSDL (não faz parte da implementação - é só descrição de interface)
- WSDL pode ser mapeada a linguagens (binding)
 - **Mapeamento**: tipos de dados, estruturas, etc.
 - Pode-se **gerar código** de cliente e servidor a partir de WSDL (stubs & skeletons) em tempo de compilação ou execução
- WSDL facilita a interoperabilidade
 - Viabiliza RPC via SOAP
 - Pode-se gerar a **parte do cliente** em uma plataforma (ex: .NET) e a **parte do servidor** em outra (ex: J2EE), viabilizando a comunicação entre arquiteturas diferentes.

```

<?xml version="1.0" encoding="UTF-8"?>
<definitions name="BookstoreService"
  targetNamespace="http://mybooks.org/wsdl"
  xmlns:tns="http://mybooks.org/wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <types>...</types>
  <message name="BookstoreIF_getPrice">
    <part name="String_1" type="xsd:string"/>
  </message>
  <message name="BookstoreIF_getPriceResponse">
    <part name="result" type="xsd:decimal"/>
  </message>
  <portType name="BookstoreIF">
    <operation name="getPrice" parameterOrder="String_1">
      <input message="tns:BookstoreIF_getPrice"/>
      <output message="tns:BookstoreIF_getPriceResponse"/>
    </operation>
  </portType>
  <binding ... > ...</binding>
  <service ... > ... </service>
</definitions>

```

Compare com a
mensagem SOAP
mostrada
anteriormente

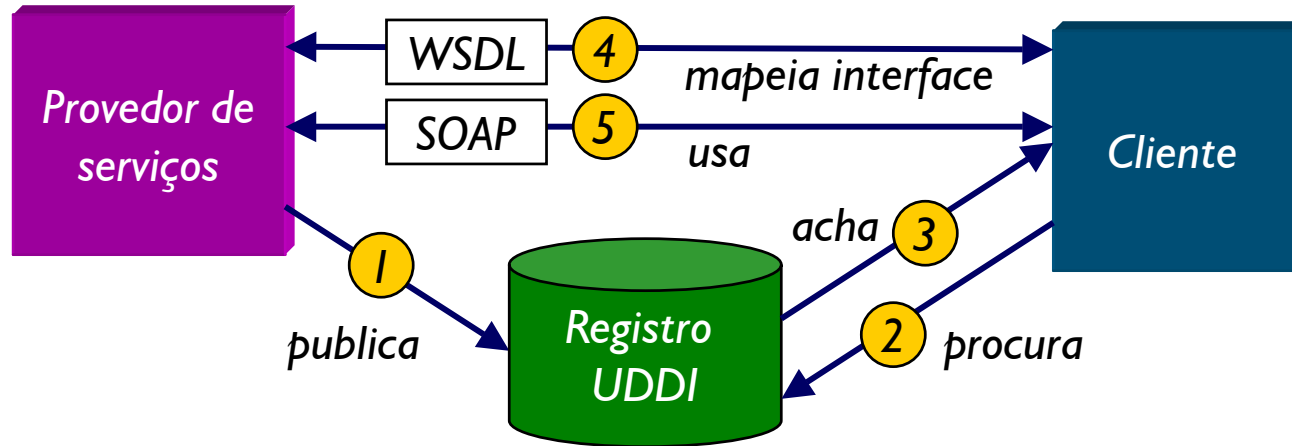


← Informa onde está o serviço (endpoint)

Registro e localização do serviço: UDDI

- **U**niversal **D**iscovery and **D**escription **I**ntegration
 - Registro global para Web Services: nuvem UDDI
 - Esquema padrão (XML) para representar firmas, serviços, pontos de acesso dos serviços, relacionamentos, etc.
 - Objetivo é permitir a maior automação no uso dos serviços
 - Registro UDDI acha e devolve URL do WSDL ou serviço
- Registro centralizado permite
 - Independência de localização
 - Facilidade para pesquisar e utilizar serviços existentes
- Tipos de informações armazenadas em UDDI
 - White pages: busca um serviço pelo nome
 - Yellow pages: busca um serviço por assunto
 - Green pages: busca com base em características técnicas

- Arquitetura de serviços usando SOAP, WSDL e UDDI



- Comparação com outras soluções de RPC

	Java RMI	CORBA	RMI / IIOP	Web Services
Registro	RMI Registry	COS Naming	JNDI	UDDI
Descrição de Serviços	Java	OMG IDL	Java	WSDL
Transporte	Java RMI	IIOP	IIOP	SOAP

Tecnologias Java para Web Services

- **Java 2 Enterprise Edition (J2EE)**
 - Versão 1.3 (atual): já possui todos os recursos necessários para **infraestrutura** de Web Services (servlets, JSP)
 - Versão 1.4 (2003): integração nativa com Web Services - será mais fácil transformar EJBs e componentes Web em clientes e provedores de Web Services
- **Para criar Web Services em Java hoje**
 - (1) Java Servlet API 2.3, JSP 1.2, JSTL 1.0
 - (2) Implementações Java de XML, SOAP, UDDI (há várias: IBM WSDL4J, UDDI4J, Apache SOAP, AXIS, Xerces, Xalan)
 - (3) Java XML Pack ("série JAX")

➔ **Java Web Services Development Pack = (1) + (3)**

Java Web Services Development Pack 1.0

SUCESU-SP 2002

- APIs
 - Processamento XML: **JAXP 1.1**
 - Web Services: **JAX-RPC 1.0, JAXM 1.1, SAAJ 1.1, JAXR 1.0**
 - Aplicações Web: **Servlet API 2.3, JSP 1.2, JSTL 1.0**
- Implementação de referência
 - **Ferramentas de desenvolvimento:** Web Deploytool, Compilador JAXRPC (xrpcc), Jakarta Ant, Jakarta Tomcat, Registry Browser e Apache Xindice (banco de dados XML)
 - Serviços de **registro UDDI, roteamento SOAP** e **JAXRPC** (implementados como servlets no Tomcat)

xindice
(Zeen-dee-chay)



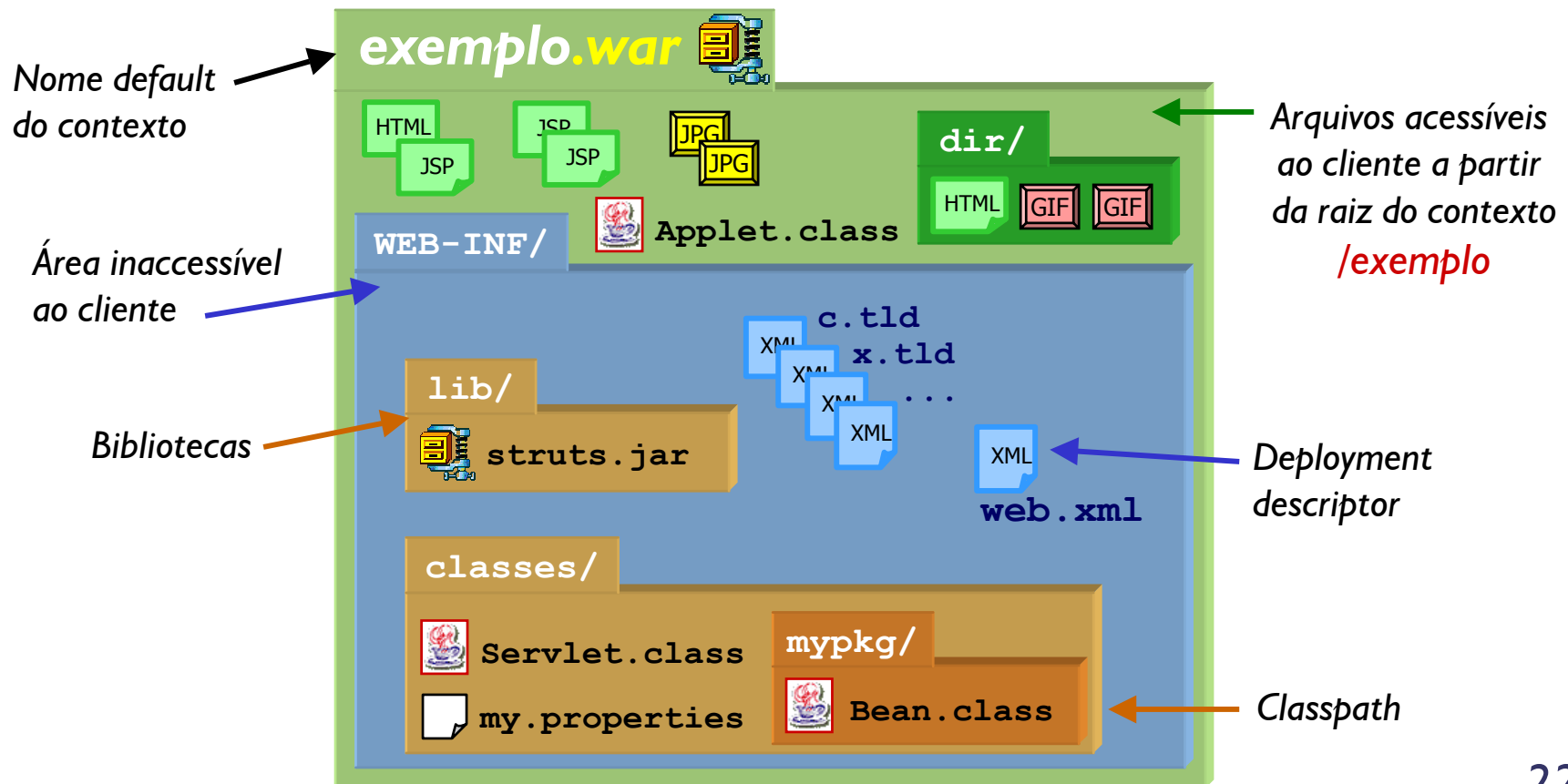
- Web Services podem ser desenvolvidos em Java usando os pacotes **javax.servlet.*** que permitem criar
 - **Servlets**: componentes capazes de processar requisições HTTP e gerar respostas HTTP
 - **Páginas JSP**: documentos de texto (HTML, XML) que são transformados em servlets na instalação ou execução
 - **Bibliotecas de tags**: implementações que permitem o uso de XML no lugar do código Java em páginas JSP
- **Deployment é muito simples**
 - Escreva os servlets ou JSPs que implementam Web Services
 - Escreva ou gere um deployment descriptor
 - Coloque tudo em um arquivo WAR
 - Instale o WAR no servidor (ex: copiar para pasta webapps/)

Estrutura de um arquivo WAR

SUCESU-SP 2002

- Aplicações Web são empacotadas em arquivos WAR para instalação automática em servidores J2EE

<http://servidor.com.br/exemplo>



- APIs padrão no J2SDK e J2EE
 - **JAXP**: suporte a APIs para processamento XML: DOM , SAX e XSLT
- APIs padrão no Java Web Services Development Pack
 - **JAXM, JAX-RPC** e **SAAJ**: suporte a protocolos de comunicação baseados em XML
 - **JAXR**: suporte a sistemas de registro baseados em XML
- Padrões propostos (em desenvolvimento)
 - **JAXB** (JSR-31: XML data binding): suporte à serialização de objetos em XML
 - **JDOM** (JSR-102): outro modelo para processamento XML (que não usa a interface W3C DOM)
 - JSR-181: linguagem de **metadados** para Web Services

- **Java API for XML Processing**

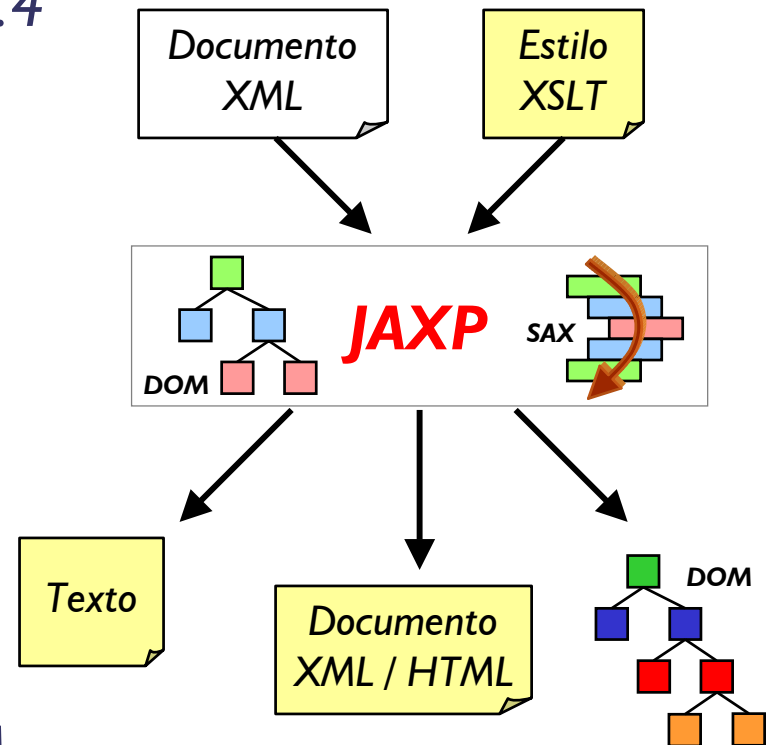
- Para leitura, criação, manipulação, transformação de XML
- Parte integrante do J2SDK 1.4

- **Pacotes**

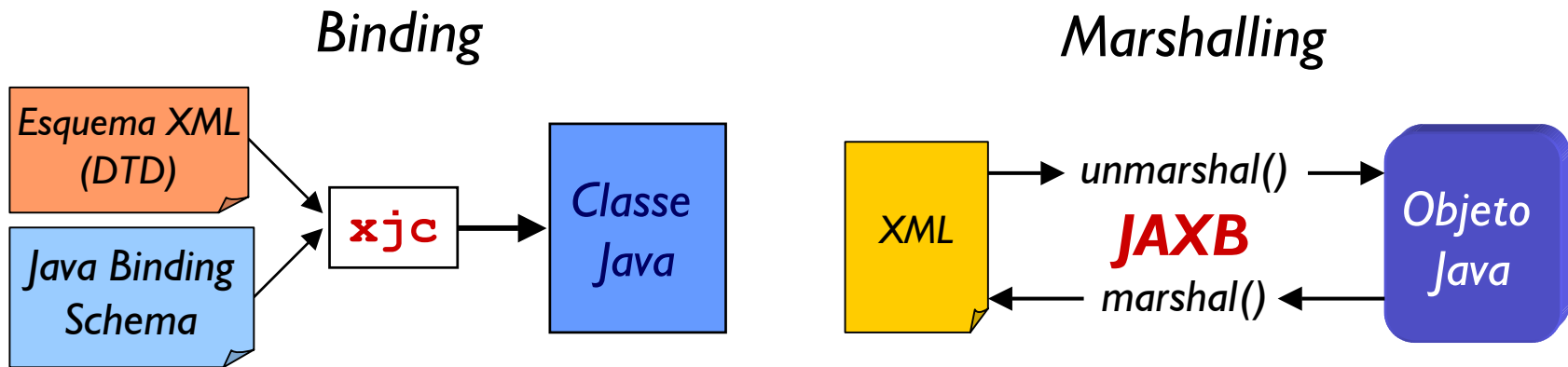
- `javax.xml.parsers`
- `javax.xml.transform.*`
- `org.w3c.dom`
- `org.w3c.sax.*`

- **Componentes**

- *Parsers para SAX e DOM*
- *Implementações em Java das APIs padrão SAX e DOM*
- *Implementações Java de API de transformação XSLT*

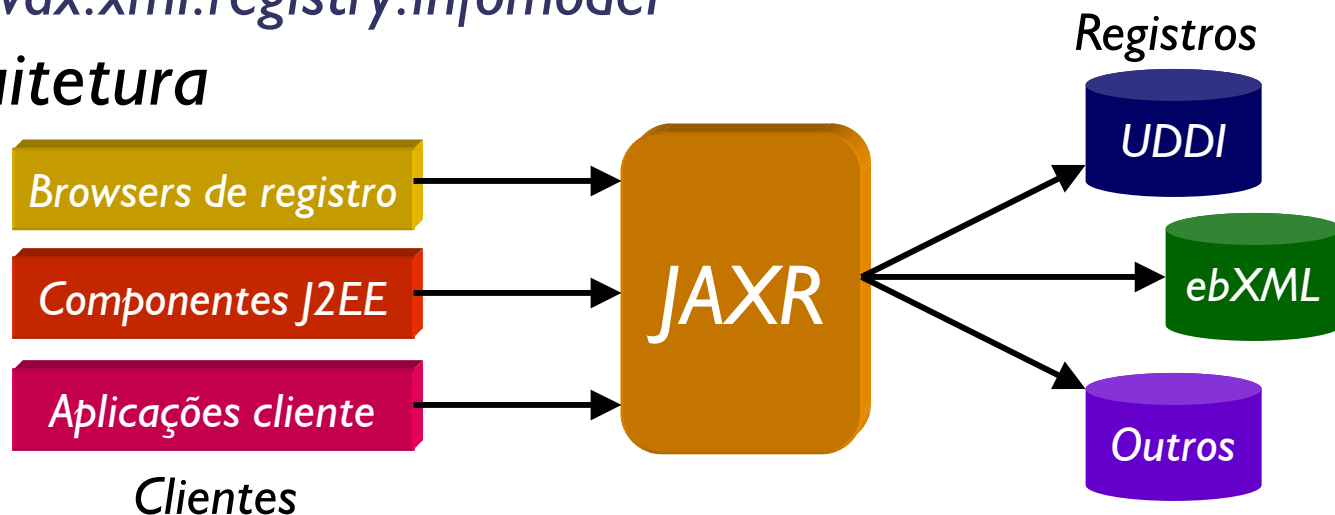


- **Java API for XML Binding (JSR-311)**
 - Mapeia classes Java a documentos XML
 - Permite gerar JavaBeans a partir de esquema XML
 - Permite serializar objetos para XML e vice-versa

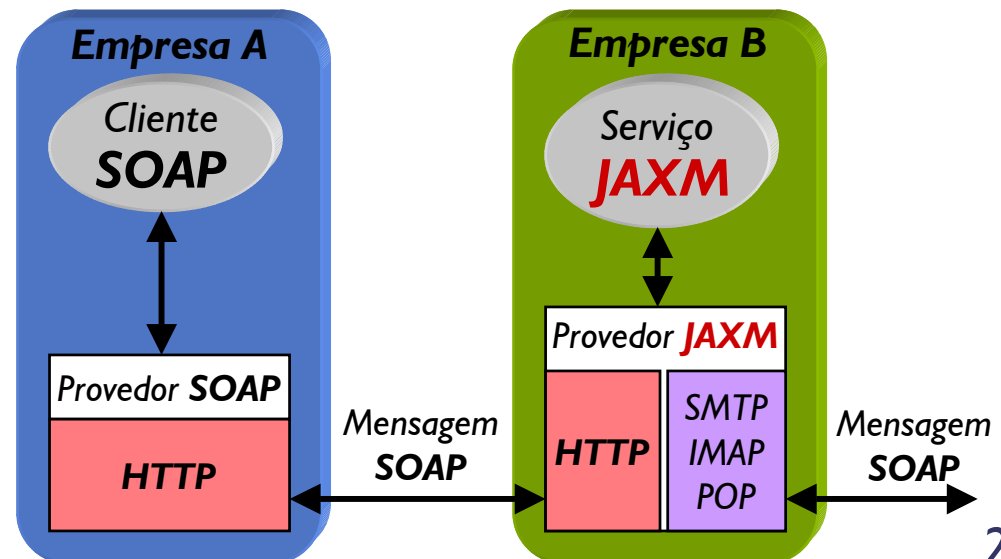


- **Pacotes (community review jul-2002)**
 - `javax.xml.bind`
 - `javax.xml.marshall`
- **Em desenvolvimento há 3 anos (29/ago/1999).**

- **Java API for XML Registries**
 - Oferece acesso uniforme a diferentes sistemas de registro de serviços baseados em XML
 - Possui mapeamentos para **UDDI** e **ebXML**
 - Permite a inclusão e pesquisa de organizações, serviços
- **Pacotes**
 - `javax.xml.registry`
 - `javax.xml.registry.infomodel`
- **Arquitetura**



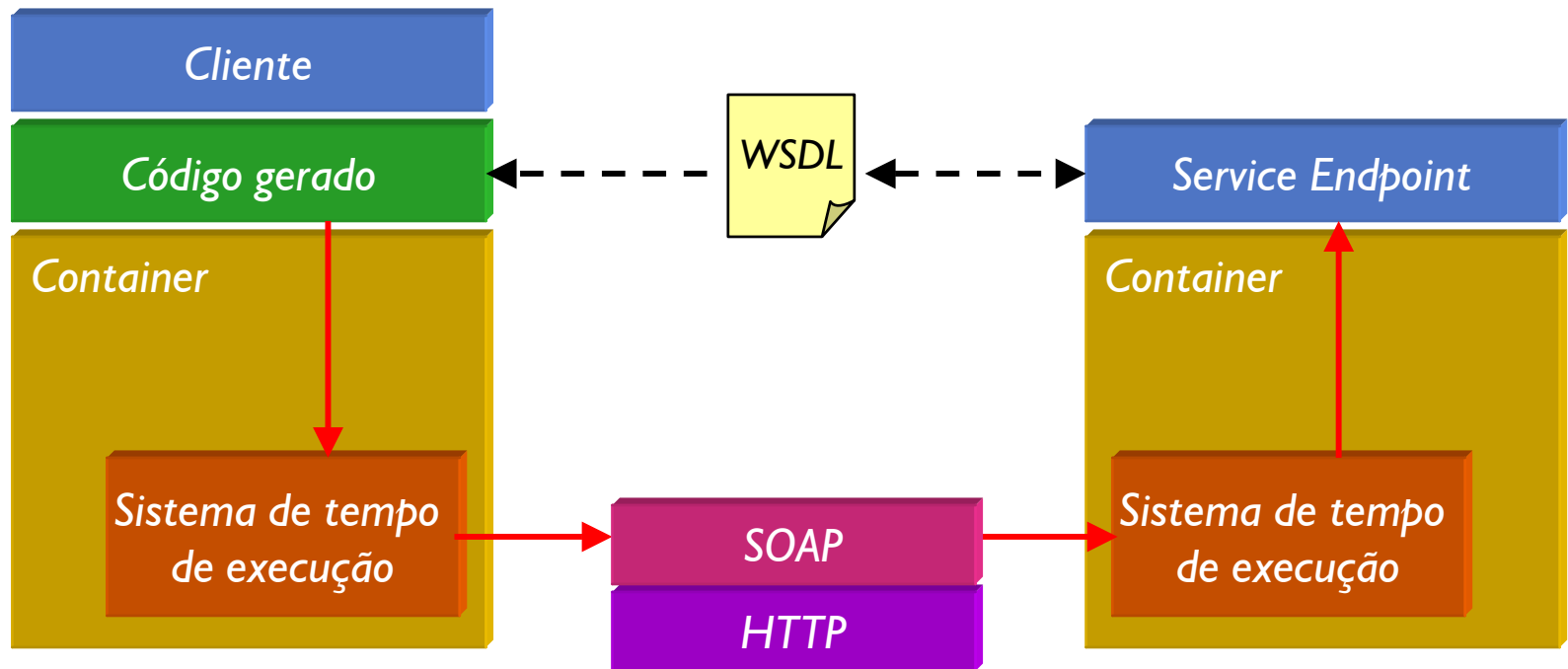
- **Java API for XML Messaging** (e **SOAP with Attachments API for Java**)
 - Conjunto de APIs para manipular envelopes SOAP e transportá-los sobre HTTP, SMTP ou outros protocolos
 - Suporta comunicação baseada em **eventos** (mensagens) e baseada em **RPC** (par de mensagens requisição/resposta)
 - Suporta especificações SOAP 1.1 e SOAP with Attachments
- **Pacotes:**
 - `javax.xml.soap`
 - `javax.xml.messaging`
 - `javax.xml.rpc.*`



Fonte da ilustração:
JAXM 1.0 specification

- **Java API for XML-Based Remote Procedure Calls**
 - Um tipo de **Java RMI** sobre SOAP/HTTP
 - Alto nível de abstração **permite ignorar envelope SOAP**
 - Utiliza **WSDL** para **gerar** classes de servidor e cliente
- **Pacotes**
 - `javax.xml.rpc.*`
- **Desenvolvimento semelhante a RMI (simples e baseado em geração de código e container)**
 - **Escreve-se RMI, obtém-se SOAP e WSDL**
 - **Cliente pode obter interface para comunicação com o serviço dinamicamente, em tempo de execução**
 - **Stubs também podem ser gerados em tempo de compilação para maior performance**

- São soluções diferentes para manipular o mesmo envelope SOAP
 - JAX-RPC implementa **WSDL**. JAXM **não usa WSDL**.
 - JAXM manipula **mensagens** sem ligar para seu conteúdo
 - JAX-RPC usa WSDL para formato de **requisições** e **respostas**
 - JAXM **expõe** todos os detalhes do envelope; JAX-RPC **oculta**
 - Tudo o que se faz em JAX-RPC, pode-se fazer com JAXM
 - **RPC** é mais fácil com JAX-RPC; JAXM é API de baixo nível e pode ser usada tanto para **messaging** ou RPC
 - Cliente e serviço JAX-RPC rodam em **container**
- **Conclusão**
 - Use **JAX-RPC** para criar aplicações SOAP-RPC com WSDL
 - Use **JAXM** para messaging ou quando precisar manipular o envelope SOAP diretamente



Criação de um Web Service com JAX-RPC (I)

1. Escrever uma interface RMI para o serviço

```
package example.service;  
  
public interface BookstoreIF extends java.rmi.Remote {  
    public BigDecimal getPrice (String isbn)  
        throws java.rmi.RemoteException;  
}
```

2. Implementar a interface

```
package example.service;  
  
public class BookstoreImpl implements BookstoreIF {  
    private BookstoreDB database = DB.getInstance();  
  
    public BigDecimal getPrice (String isbn) {  
        return database.selectPrice (isbn);  
    }  
}
```

Criação de um Web Service com JAX-RPC (2)

3. Escrever arquivo de configuração*

```
<configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <service name="BookstoreService" targetNamespace="http://mybooks.org/wsdl"
    typeNamespace="http://mybooks.org/types"
    packageName="example.service">
    <interface name="example.service.BookstoreIF"
      servantName="example.service.BookstoreImpl"/>
  </service>
</configuration>
```

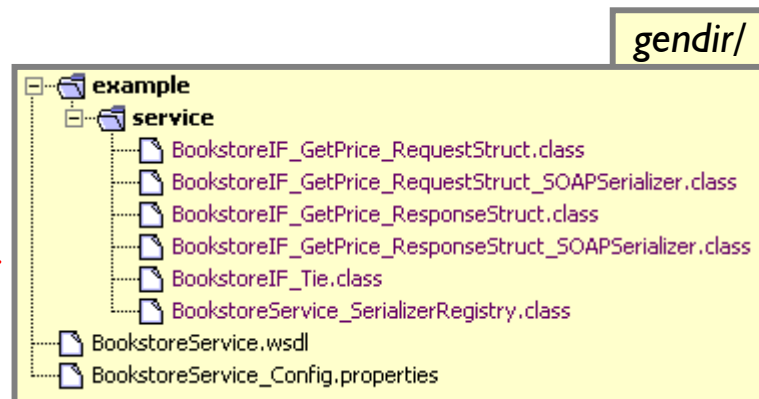
config_rmi.xml

4. Compilar classes e interfaces RMI

> javac -d mydir BookstoreIF.java BookstoreImpl.java

5. Gerar código do servidor

> xrpc -classpath mydir
-server -keep
-d gendir
config_rmi.xml



* Não faz parte da especificação - procedimento pode mudar no futuro

Criação de um Web Service com JAX-RPC (3)

SUCESU-SP 2002

- 6. Criar web deployment descriptor **web.xml**

```
<web-app>
  <servlet>
    <servlet-name>JAXRPCEndpoint</servlet-name>
    <servlet-class>
      com.sun.xml.rpc.server.http.JAXRPCServlet
    </servlet-class>
    <init-param>
      <param-name>configuration.file</param-name>
      <param-value>
        /WEB-INF/BookstoreService_Config.properties
      </param-value>
    </init-param>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>JAXRPCEndpoint</servlet-name>
    <url-pattern>/bookpoint/*</url-pattern>
  </servlet-mapping>
</web-app>
```

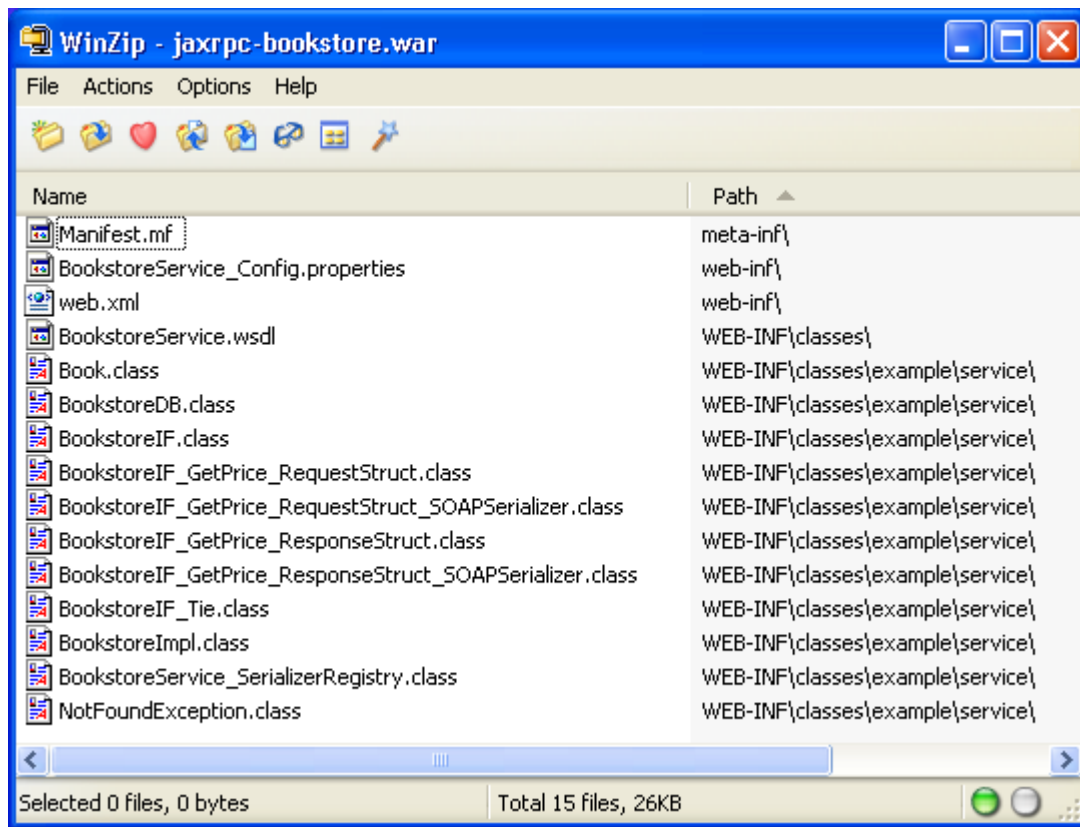
Nosso "container" →

Nome do arquivo gerado pelo xrpcc →

subcontexto que será o endpoint do serviço →

Criação de um Web Service com JAX-RPC (4)

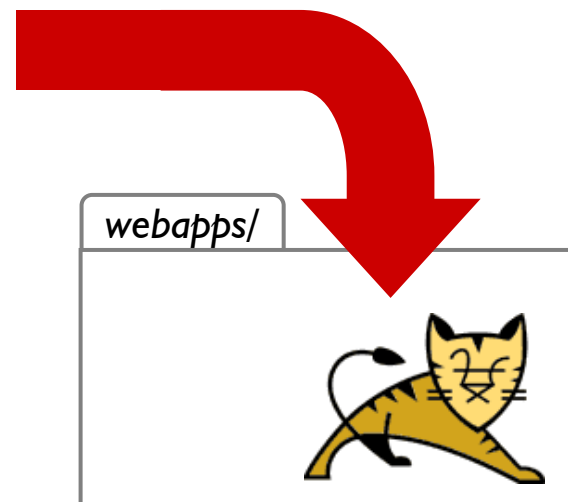
7. Colocar tudo em um WAR



jaxrpc-bookstore.war

8. Deployment no servidor

- Copiar arquivo para diretório webapps do Tomcat

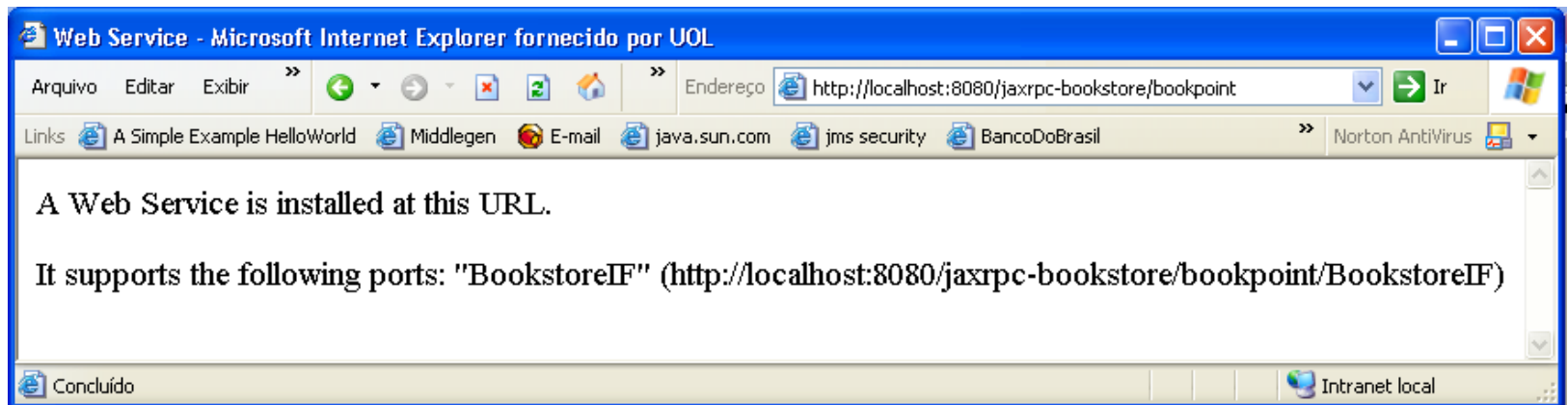


Construção e instalação do serviço com o Ant

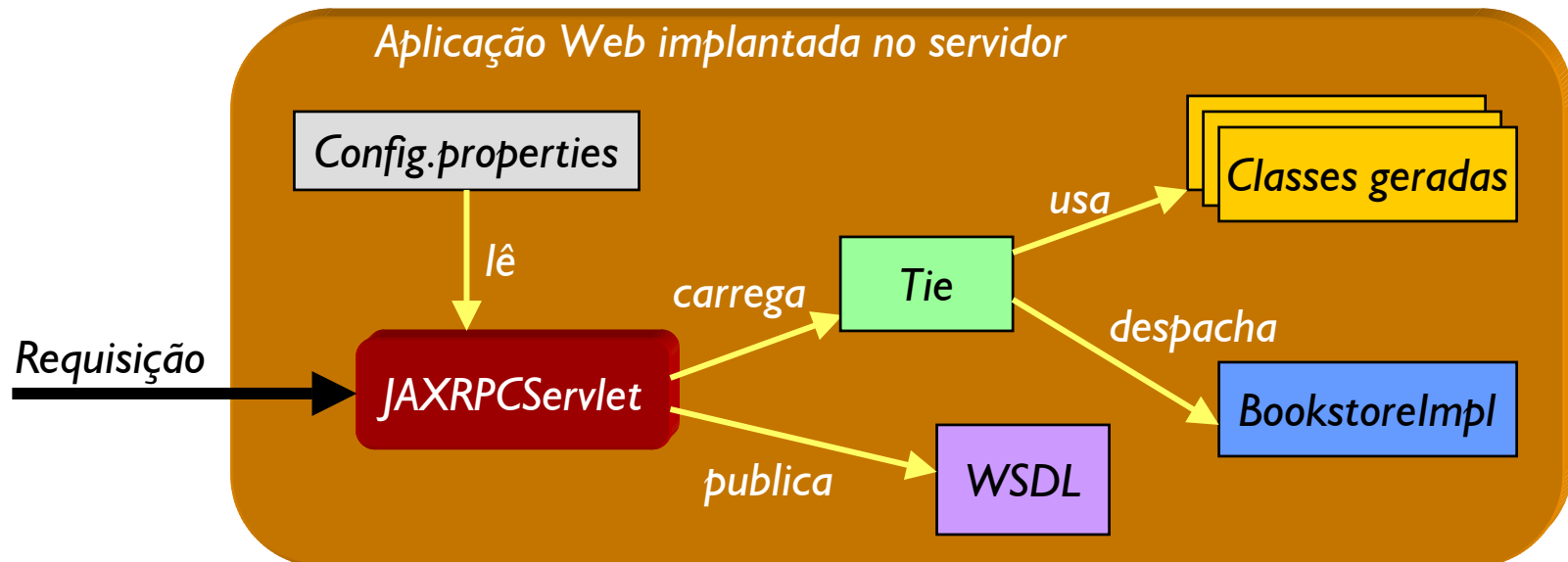
- Script do Ant para compilar as classes RMI, compilá-las com **xrjcc**, gerar o WSDL, empacotar no WAR e copiar para o diretório `webapps/` do Tomcat

> **ant BUILD.ALL.and.DEPLOY**

- Teste para saber se o serviço está no ar
 - Inicie o Tomcat do JWSDP
 - Acesse: **<http://localhost:8080/jaxrpc-bookstore/bookpoint>**



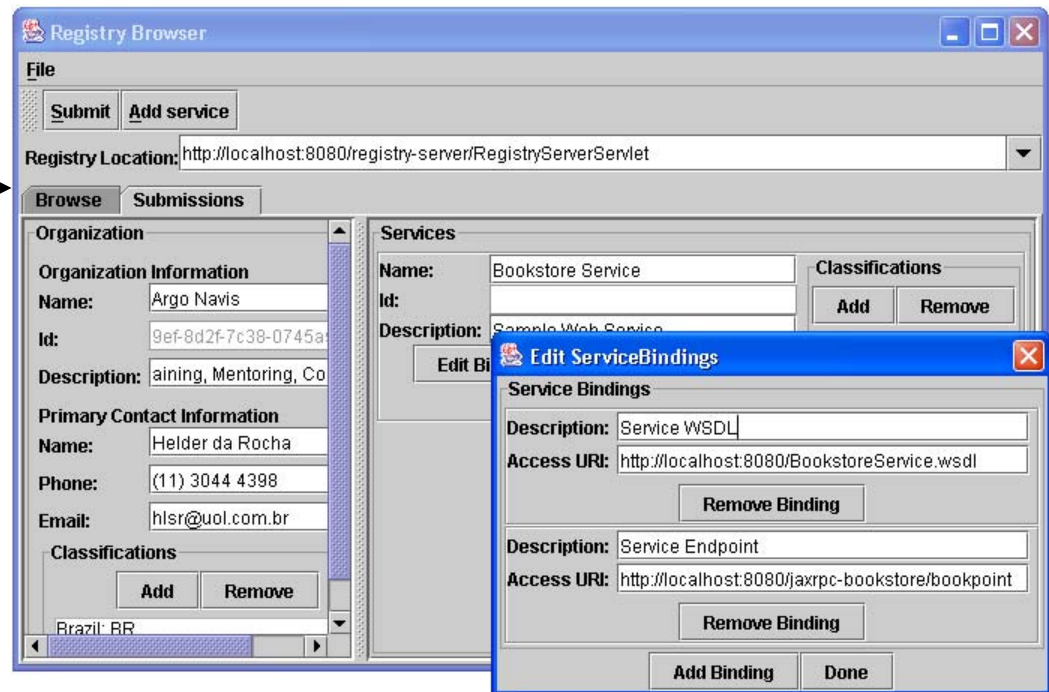
- O endpoint do serviço na implementação de referência JWSDP 1.0 é um **servlet** `com.sun.xml.rpc.server.http.JAXRPCServlet`
 - Próximas versões (e J2EE 1.4) devem oferecer implementação em **stateless session bean**
- Servlet é ponto de entrada para todas as requisições



- Podemos registrar o nosso Web Service
 - Automaticamente executando um cliente (ant REGISTER)
 - Interativamente usando o Registry Browser
- Para usar o servidor UDDI do JWSDP
 - 1. inicie o Xindice
 - 2. inicie o Tomcat

Registry Browser

1. Selecione a localização do servidor (*http://localhost/...*)
2. Crie uma nova organização
3. Crie novo serviço
4. Em "edit bindings" coloque URLs dos serviços
5. Aperte *submit*. Use "testuser" como nome e senha



- Há três tipos de cliente JAX-RPC:
 - 1. *Cliente estático tipo-RMI*: usa stubs gerados em **tempo de compilação** para se comunicar com o servidor e chama métodos do serviço remoto como se fossem locais
 - 2. *Cliente WSDL de interface dinâmica (DII)*: descobre a interface de comunicação em **tempo de execução** e chama métodos via mecanismo similar a Java reflection
 - 3. *Cliente WSDL de interface estática*: usa interface Java implementada por stubs gerados em **tempo de execução** e chama métodos remotos como se fossem locais
- *Clientes precisam aderir ao contrato com o Web Service (WSDL) mas podem ser implementados e usados com ou sem WSDL*

Cliente de implementação estática

- Manifest.mf
- BookstoreService.class
- BookstoreService_Impl.class
- BookstoreService_Registry.class
- BookstoreIF.class
- BookstoreIF_Impl.class
- BookstoreIF_getPrice_RequestStruct.class
- BookstoreIF_getPrice_ResponseStruct.class
- BookstoreIF_getPrice_RequestStruct_SOAPSerializer.class
- BookstoreIF_getPrice_ResponseStruct_SOAPSerializer.class
- BookstoreIF_Stub.class
- BookstoreClient.class**
- BookstoreIF.class
- Book.class
- BookstoreDB.class
- NotFoundException.class

stub (1)

+ performance, + acoplamento

2. Chama o serviço

Clientes de implementação dinâmica

- Manifest.mf
- BookstoreClient.class**
- BookstoreIF.class

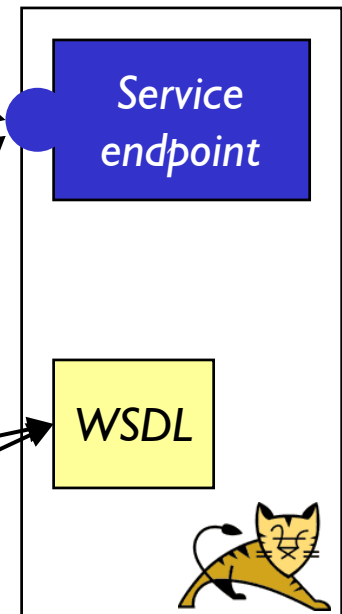
proxy (3)

- Manifest.mf
- BookstoreClient.class**

dynamic (2)

1. Obtém informações sobre o serviço

- performance, - acoplamento



Clientes JAX-RPC (detalhes)

- *1) Cliente com stub estático*

```
Stub stub = (Stub)(new BookstoreService_Impl().getBookstoreIFPort());
stub._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY, endpointURL);
BookstoreIF proxy = (BookstoreIF) stub;
System.out.println("Price R$ " + proxy.getPrice("2877142566"));
```

- *Cliente com interface dinâmica (DII)*

```
Service srv = factory.createService(new URL(wsdlURL),
                                   new QName(NS, "BookService"));
Call call = srv.createCall( new QName(NS, "BookstoreIFPort") );
call.setOperationName(new QName(NS, "getPrice"));
BigDecimal pr = (BigDecimal) call.invoke(new String[] {"2877142566"});
System.out.println("Price R$ " + pr);
```

- *Cliente com stub dinâmico (proxy)*

```
Service srv = factory.createService(new URL(wsdlURL),
                                   new QName(NS, "BookService"));
BookstoreIF proxy = (BookstoreIF)
    srv.getPort(new QName(NS, "BookstoreIFPort"), BookstoreIF.class);
System.out.println("Price R$ " + proxy.getPrice("2877142566"));
```

- *Para gerar os clientes*
 - *Cliente (1): gere stubs com `xrpsc -client` e arquivo WSDL (use `config_wsdl.xml`) e depois compile classe do cliente*
 - *Cientes (2) e (3): apenas compile a classe do cliente*
- *Script do Ant para compilar os três clientes e colocar as classes em um JAR*

```
> ant client.BUILD
```

- *Para rodar o cliente e executar o Web Service*

```
> ant dynamic-client.RUN
```

```
Buildfile: build.xml
```

```
dynamic-client.RUN:
```

```
[java] ISBN 2877142566. Price R$ 19.50
```

```
BUILD SUCCESSFUL
```

- Nesta palestra apresentamos a arquitetura de **Web Services**, suas tecnologias fundamentais SOAP, WSDL e UDDI e as APIs Java que as implementam.
- Java oferece **APIs** que permitem desde a **manipulação direta** de XML (DOM e SAX) até a criação de Web Services **sem contato** com XML (JAX-RPC)
- JAX-RPC é a forma mais **fácil** e **rápida** de criar Web Services em Java
- Serviços desenvolvidos em JAX-RPC poderão ser acessados de aplicações .NET e vice-versa.
 - Web Services viabilizam a integração de serviços entre plataformas diferentes: **interoperabilidade!**

- [1] JSR-101 Expert Group. *Java™ API for XML-based RPC: JAX-RPC 1.0 Specification*. Java Community Process: www.jcp.org.
- [2] Sun Microsystems. *Java™ Web Services Tutorial*. java.sun.com/webservices/.
Coleção de tutoriais sobre XML, JSP, servlets, Tomcat, SOAP, JAX-RPC, JAXM, etc.
- [3] JSR-109 Expert Group. *Web Services for J2EE 1.0 (Public Draft 15/04/2002)*. Java Community Process: www.jcp.org. *Descreve o suporte a Web Services em J2EE 1.3*
- [4] Nicholas Kasseem et al. (JSR-67). *Java™ API for XML Messaging (JAXM) e Soap with Attachments API for Java 1.1*. java.sun.com. *Modelo de programação de baixo nível (lida diretamente com SOAP enquanto JAX-RPC esconde) e mais abrangente.*
- [5] Roberto Chinnici. *Implementing Web Services with the Java™ Web Services Development Pack*. JavaONE Session 1777. java.sun.com/javaone. *Apresentação que oferece uma visão geral de JAX-RPC e o Web Services Development Pack da Sun.*
- [6] Brett McLaughlin. *Java & XML 2nd. Edition*. O'Reilly and Associates, 2001. *Explora as APIs Java para XML e oferece uma introdução à programação de WebServices em Java*
- [7] Ethan Cerami. *Web Services Essentials*. O'Reilly, Fev 2002. *XML-RPC, SOAP, UDDI e WSDL são explorados de forma didática e exemplos são implementados em Java usando ferramentas open-source.*
- [8] W3C Web Services Activity. <http://www.w3.org/2002/ws/>. *Página que dá acesso aos grupos de trabalho que desenvolvem especificações de SOAP (XMLP), WSDL e Arquitetura*

- [9] *Apache XML Project*. xml.apache.org. *Duas implementações de SOAP e uma implementação de XML-RPC em Java*.
- [10] *IBM Developerworks Open Source Projects*. <http://www-124.ibm.com/>. *Implementações UDDI4J e WSDL4J*.
- [11] Al Saganich. *Java and Web Services Primer*. O'Reilly Network 2001. <http://www.onjava.com/pub/a/onjava/2001/08/07/webservices.html>. *Ótimo tutorial sobre Web Services*.
- [12] Al Saganich. *Hangin' with the JAX Pack. Part 1: JAXP and JAXB, Part 2: JAXM, Part 3: Registries (JAXR), Part 4: JAX-RPC*. O'Reilly Network 2001-2002. <http://www.onjava.com/pub/a/onjava/2001/11/07/jax.html> *Esta série de quatro artigos publicados entre nov/2001 e abr/2002 é talvez o melhor ponto de partida para quem desejar aprender a usar as APIs Java para Web Services*.
- [13] David Chappell, Tyler Jewel. *Java Web Services*. O'Reilly and Associates, Mar 2002. *Explora implementações Java de Apache SOAP, WSDL e UDDI em Java. Tem um capítulo dedicado às APIs do JWSDP*.
- [14] Al Saganich. *JSR-109 Web Services inside of J2EE Apps*. O'Reilly Network, Aug 2002. <http://www.onjava.com/pub/a/onjava/2002/08/07/j2eewebsvs.html> *Mostra um resumo da proposta do JSR-109, que prevê a integração J2EE-Web Services*.

helder@argonavis.com.br

Selecione o link relativo a esta palestra no endereço

www.argonavis.com.br/comdex2002

Recursos disponíveis no site:

- *Palestra completa em PDF*
- *Todo o código-fonte usado nos exemplos e demonstrações*
- *Instruções detalhadas sobre como rodar e instalar os exemplos*
- *Links para software utilizado e documentação*