



Fundamentos da Plataforma J2EE

Helder da Rocha
www.argonavis.com.br

- *Este módulo tem três objetivos*
 - *Configurar o ambiente de trabalho*
 - *Oferecer uma visão geral da tecnologia J2EE definindo os seus conceitos fundamentais e mostrando quais os problemas que a tecnologia pretende solucionar*
 - *Apresentar uma introdução prática à tecnologia J2EE através do desenvolvimento, montagem, instalação e configuração de componentes simples*
- *Para a parte prática usaremos*
 - *Exemplo do capítulo 3 do livro Mastering EJB2**
 - *Roteiro e exemplos do capítulo 1 do livro J2EE Tutorial* (alterados para implantação no JBoss)*
 - *Veja diretório cap01 no CD deste curso*

* Veja referências no final

- *Introdução: ambiente e arquivos*
- *Conceitos fundamentais*
 - *Servidores de aplicação*
 - *Arquitetura de componentes*
 - *Plataforma J2EE: arquitetura, APIs, serviços e papéis*
 - *Componentes J2EE*
- *Infraestrutura*
 - *Ferramentas do J2EE SDK: deploytool, servidor e verifier*
 - *Servidor JBoss*
- *Introdução prática*
 - *Tecnologia EJB: como criar e instalar um Session Bean*
 - *Tecnologia Web: como criar e instalar um JSP (WAR)*
 - *Implantação de um Enterprise Archive (EAR)*

Estrutura dos exemplos no CD

- Cada módulo possui um diretório no CD que contém arquivos com exemplos. Para a maioria, a compilação, montagem e execução pode ser feita através do **Ant**
 - O Ant executa um roteiro escrito em XML: o buildfile (`build.xml`)
 - Entender a linguagem do Ant ajuda a entender como as aplicações são montadas e executadas
- Estrutura típica da maior parte dos diretórios de exemplos
 - `src/` Contém os fontes
 - `websrc/` Contém as fontes Web (HTML, JSP, GIFs, JPG)
 - `deployment/` ou `dd/` contém deployment descriptors (XML)
 - `lib/` Contém bibliotecas requeridas
 - `build.xml` Buildfile usado pelo Ant (alguns requerem modificação)
 - `build.properties` **Configure com os dados do seu ambiente!**
- Principais diretórios gerados pelo Ant (pode haver outros)
 - `build` contém código compilado
 - `jars` contém componentes empacotados em JARs, WARs e EARs

Tecnologias "corporativas"

- *"Enterprise" é o nome usado para identificar as tecnologias Java que são usadas em sistemas distribuídos*
 - *"Enterprise" == "Distributed"*
- *São várias as tecnologias Java para sistemas distribuídos*
 - *Banco de dados: JDBC, SQLJ, JDO*
 - *Objetos distribuídos: Java RMI, RMI-IIOP, Java IDL (CORBA), EJB*
 - *Serviços distribuídos: JTA, JTS, JAAS, DGC, JNDI*
 - *Eventos distribuídos: JMS*
 - *Web: Servlets, JSP, JSTL*
 - *XML e Web services: JAXP, JAXB, JAXM, JAXR, JAX-RPC*
 - *E-mail: JavaMail*
 - *...*
- *Algumas tecnologias podem ser usadas através de APIs do J2SE. Outras são distribuídas apenas no pacote J2EE.*

- *JDBC é a API padrão*
 - *Pacote java.sql*
 - *Conecta programas em Java, de forma transparente, a bancos de dados relacionais*
 - *Maior parte dos bancos do mercado possuem drivers*
- *Extensões opcionais*
 - *Pacote javax.sql*
 - *Fazem parte de J2EE*
 - *Suporte a RowSets e DataSources*
 - *javax.sql.DataSource permite a obtenção de uma conexão de um pool de conexões via JNDI*

```
DataSource ds = (DataSource) ctx.lookup("jdbc/Banco");  
Connection con = ds.getConnection();
```

Objetos distribuídos

- *Java RMI (sobre JRMP)*
 - *Solução 100% Java para comunicação em rede*
 - *Muito mais simples e fácil de manter e expandir que soluções baseadas em sockets*
- *Java IDL*
 - *Implementação de CORBA em Java*
 - *Solução para integração com aplicações escritas em outras linguagens*
- *RMI sobre IIOP*
 - *Modelo de programação RMI com execução CORBA*
- *EJB*
 - *Arquitetura de componentes baseada em RMI sobre IIOP*

Serviços distribuídos

- *JTS (Java Transaction Service)*
 - *Serviço de transações distribuído compatível com CORBA Transaction Service (OTS)*
- *JTA (Java Transaction API)*
 - *API de programação Java que implementa o padrão XA do Open Group para transações distribuídas (two phase commit protocol)*
- *DGC (Distributed Garbage Collection)*
 - *Remove referências remotas que não podem mais ser usadas*
 - *Disponível apenas em aplicações Java RMI*
- *JNDI (Java Naming and Directory Interface)*
 - *Interface padrão genérica para sistemas de nomes*
- *JAAS (Java Authorization and Authentication Services)*
 - *API para implementação de sistemas de login com reconhecimento de identidade do usuário.*

- *JMS (Java Message Service)*
 - *"Message" == "Event" em ambientes distribuídos*
 - *API que permite interagir com diferentes sistemas de Message Oriented Middleware (MOM)*
 - *Permite criar sistemas distribuídos com comunicação assíncrona*
 - *Alternativa aos protocolos síncronos (IIOP e RMI/JRMP)*

Serviços Internet (Web e email)

- **Componentes Web**

- *Servlets: substitui CGI, ISAPI, e similares estendendo serviço HTTP*
- *JSP: modelo de programação orientado à interface para servlets; substitui ASP, PHP, CFML e similares*

- **Web Services**

- *JAXP: coleção de APIs para leitura, geração e transformação de XML. Suporte a DOM, SAX e XSLT*
- *JAXB: API para mapeamento Java-XML. Usada para gerar código Java a partir de documentos XML*
- *JAXM: API para troca de mensagens XML com SOAP*
- *JAX-RPC: API para implementar aplicações RPC com SOAP*
- *JAXR: API para acesso a registros UDDI, ebXML*

- **JavaMail**

- *API para a programação de clientes SMTP e IMAP*

Servidores de aplicação

- Servidores de aplicação OO, também chamados de **Component Transaction Monitors**, oferecem ambientes para operação de **componentes** (rodando em **containers**) e diversos serviços de middleware
- Servidores que suportam EJB permitem oferecer vários **serviços de middleware implícito***, entre eles
 - Controle de transações
 - Autenticação e autorização
 - Gerenciamento de recursos
 - Persistência
- Nos servidores EJB esses serviços são configurados através de arquivos XML
- Os serviços também podem ser usados de forma explícita*

* mais detalhes no capítulo 9

Java 2 Enterprise Edition

- J2EE é
 - Uma **especificação** para servidores de aplicação que define padrão de suporte a componentes e serviços
 - Um pacote de **APIs** e **ferramentas** para desenvolver componentes que rodam nesses servidores
- É objetivo da **plataforma J2EE** reduzir o custo e a complexidade de desenvolver serviços multi-camada
- Servidores de aplicação compatíveis com a especificação J2EE oferecem
 - Suporte à arquitetura de componentes EJB
 - Suporte a serviços Web, servlets e JSP
 - Suporte a serviços de middleware explícito e implícito

Componentes J2EE

- Aplicações J2EE são aplicações em n-camadas feitas de **componentes**
- A especificação J2EE define os seguintes componentes
 - **Componentes cliente** - rodam na máquina do usuário
 - **Componentes Web** (JSP e servlets) - rodam em servidor
 - **Componentes EJB** - rodam em servidor
- Componentes J2EE são escritos em Java
 - Aderem a um determinado **formato padrão** de construção definida na especificação J2EE que inclui regras para a construção de métodos, classes, arquivos XML de configuração e empacotamento
 - Implantáveis (**deployable**) para produção em servidores de aplicação compatíveis com a plataforma J2EE

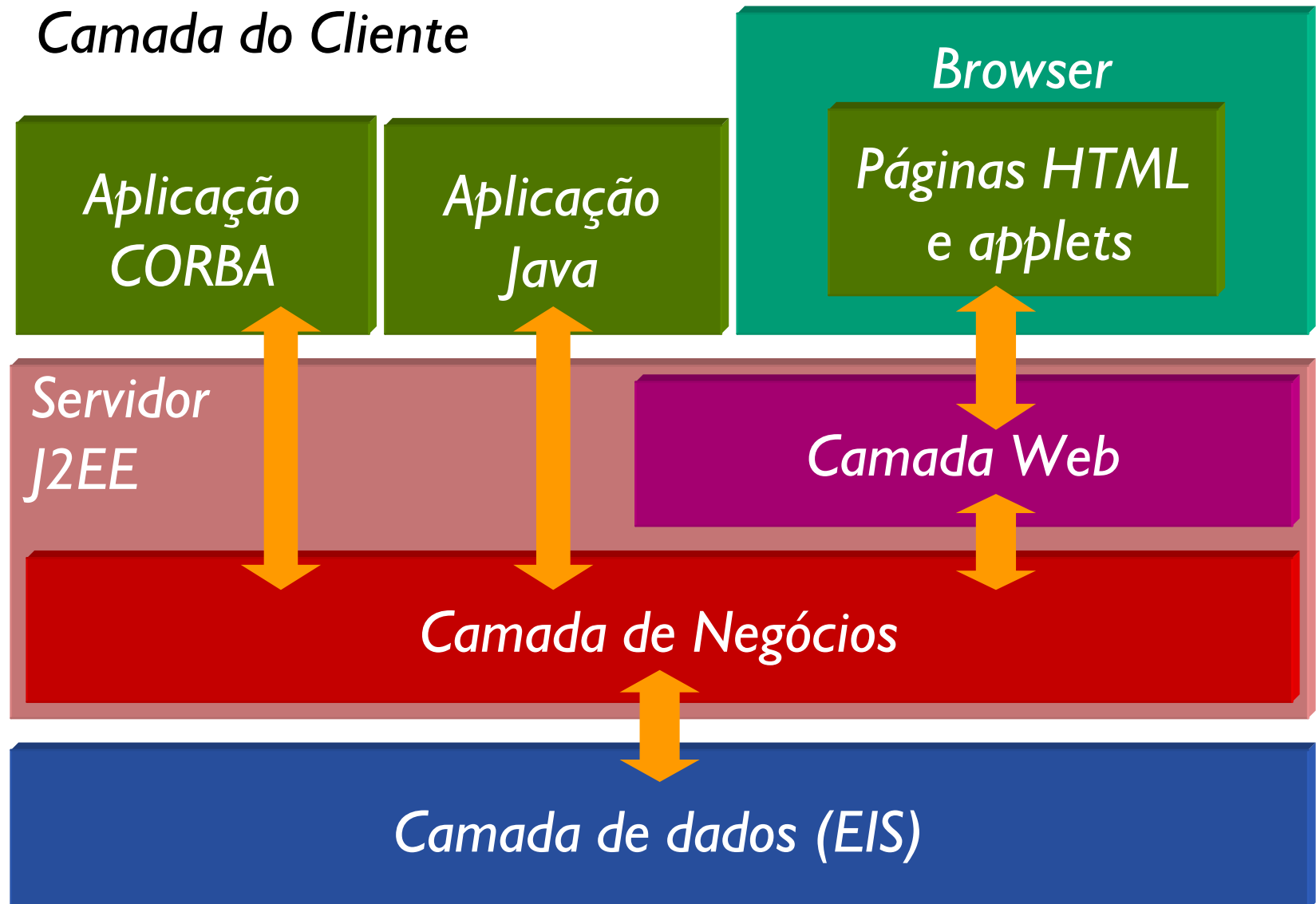
Arquitetura de componentes

- *É um contrato entre o fornecedor e o usuário*
 - *Permitem que os fornecedores construam componentes compatíveis com uma classe de servidores: **reuso!***
 - *Permite que os usuários substituam componentes existentes por outros similares: **flexibilidade!***
- *Servidores de aplicações são usuários de componentes*
 - *Cada servidor utiliza componentes que obedecem à determinada arquitetura de componentes*
- *Plug & Play*
 - *Um componente pode ser "plugado" no servidor de aplicações e passar a funcionar imediatamente.*
- *Analogia*
 - *Se CD-ROM é servidor de aplicações, o CD é componente*

Containers J2EE

- Um **container** é a interface entre o componente e as funções de baixo nível da plataforma onde roda
 - Antes que um componente EJB ou Web possa ser executado em um container J2EE ele precisa ser montado em uma aplicação J2EE e implantado no seu container
- O container é responsável por chamar os métodos que controlam o ciclo de vida dos componentes
- O container também é quem serve de interface para que o componente utiliza serviços de middleware implícito declarados nos seus arquivos de configuração
- A plataforma J2EE oferece três tipos de containers
 - Container EJB
 - Container Web
 - Container Cliente (para executar aplicações cliente)

Arquitetura em Camadas



Componentes da camada do cliente

- *Clientes Web (browsers ou web services)*
 - *Acesso indireto a EJBs via camada Web*
 - *Mídia Web estática (HTML, XML, GIF, etc.) geradas por componentes Web no servidor*
 - *Clientes sempre magros (não contém lógica de negócio)*
- *Clientes de aplicação*
 - *Podem ter acesso direto a EJBs na camada de negócios*
 - *Consistem de aplicações de linha de comando, aplicações gráficas AWT ou Swing e permitem uma interface do usuário mais sofisticada*
- *Outros clientes*
 - *Applets - podem ser clientes Web ou de aplicação*
 - *Clientes CORBA - acesso direto a EJBs via IIOP*

Componentes da camada Web

- **Servlets**

- *Classes pré-compiladas* que processam requisições HTTP e devolvem respostas HTTP de qualquer tipo
- *Ideais para a geração de conteúdo dinâmico que não é enviado para o browser como texto (imagens, vídeos, arquivos ZIP, Flash, etc.)*
- *Usados como controladores em aplicações JSP*

- **JavaServer Pages (JSP)**

- *Páginas de texto* contendo Java embutido que operam como servlets
- *Compiladas após a instalação ou durante a execução*
- *Ideais para gerar páginas de texto, HTML e XML (porta de comunicação para Web Services)*




Componentes da camada de negócio

- **Enterprise JavaBeans (EJB)**
 - Formam o núcleo de uma aplicação distribuída
 - Recebem e processam dados de clientes e enviam (transparentemente) à camada de armazenamento
 - Recuperam dados da camada de dados, processam e enviam para clientes
- **Enterprise JavaBeans são objetos CORBA***
 - Acessíveis via IIOP**, podem ser chamados por clientes CORBA (mesmo clientes escritos em outras linguagens)
- **EJBs sempre são escritos em Java**
 - São desenvolvidos usando RMI sobre IIOP: modelo de programação Java que gera objetos CORBA

* Common Object Request Broker Architecture - padrão OMG para objetos distribuídos

** Internet Inter-ORB Protocol (mais sobre isto no módulo 3)

Três tipos de enterprise beans

- Session Beans 
 - Modelam **processos** de negócio. São **ações**, verbos.
 - **Fazem** coisas: acessam um banco, fazem contas,
 - Podem manter ou não estado **não-persistente**
 - Processar informação, comprar produto, validar cartão
- Entity Beans 
 - Modelam **dados** de negócio. São **coisas**, substantivos.
 - **Representam** informações em bancos de dados
 - Mantêm estado **persistente**
 - Um produto, um empregado, um pedido
- Message-driven beans 
 - Modelam processos **assíncronos**. Respondem a **eventos**.
 - Agem somente quando recebem uma **mensagem**
 - Não mantêm estado

Enterprise JavaBeans vs. JavaBeans

- Um **Enterprise JavaBean** não é um tipo de **JavaBean**
- Ambos fazem parte de uma arquitetura de componentes
 - Implementam um contrato que permite o seu reuso por alguma outra aplicação padrão ou framework
- A arquitetura de componentes Enterprise JavaBeans define
 - Regras para construir **componentes** contendo classes, interfaces e arquivos XML de configuração visando a implantação automática em servidores EJB
 - Um EJB consiste de no mínimo três classes e um XML em um JAR
- A arquitetura de componentes JavaBeans define
 - Regras para construir **classes**, para permitir um tratamento especial por parte de ferramentas e frameworks
 - Um JavaBean consiste de no mínimo uma classe contendo um método get() e um construtor sem argumentos
- JavaBeans não são parte da arquitetura J2EE

Papéis definidos na especificação J2EE

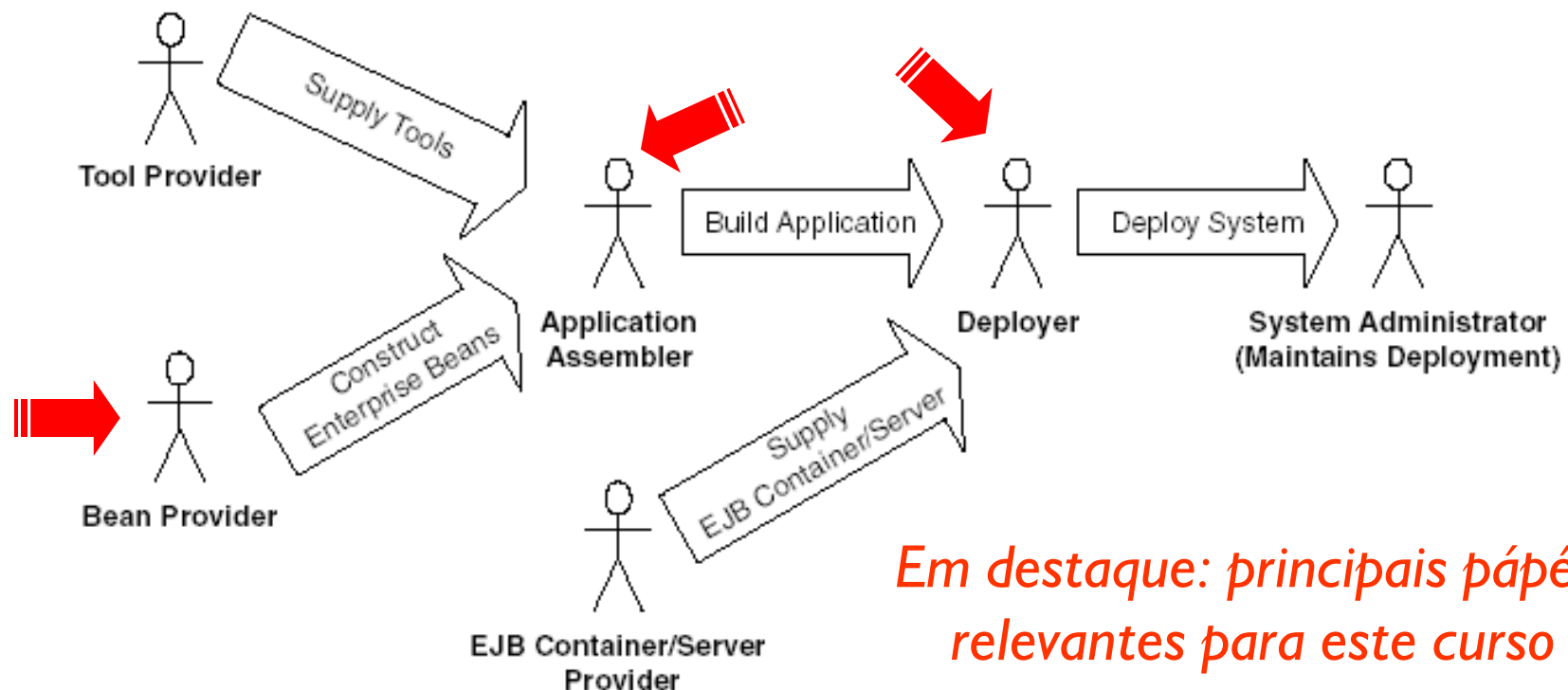
- *Provedor de componentes (**bean provider**)*
 - *Desenvolvedor que cria os componentes J2EE*
- *Provedor de ferramentas (**tool provider**)*
 - *Oferecem ferramentas para facilitar a montagem e manutenção das aplicações J2EE*
 - *Ferramentas de modelagem de dados, ferramentas de montagem de componentes, etc.*
- *Montador de aplicações (**application assembler**)*
 - *Arquiteto de aplicações que monta aplicações usando componentes desenvolvidos pelo provedor de componentes com as ferramentas do provedor de ferramentas*
 - *Escreve código de integração entre componentes, lógica de relacionamentos, etc.*

Papéis J2EE (2)

- **Provedor dos containers (*container/server provider*)**
 - *Fabricantes que oferecem containers EJB e Web*
 - *Garantem o ambiente de execução onde vivem os beans e os serviços de middleware que eles têm acesso*
 - *JBoss, WebLogic, WebSphere, Tomcat (Web), IPlanet, etc.*
- **Implantador de aplicações (*deployer*)**
 - *Instala e coloca para funcionar a aplicação no software fornecido pelo provedor de containers*
 - *Garante a integração com sistemas e serviços, escolhe o hardware, ajusta a segurança, performance, acesso a serviços e recursos externos*
- **Administrador do sistema (*system administrator*)**
 - *Garante a estabilidade da solução operacional*

Por que separação de papéis?

- Camadas distintas (devido à arquitetura J2EE) permitem que companhias ou indivíduos se especializem em certos papéis
- Alguns papéis podem ser combinados
 - Em pequena empresa, bean provider, application assembler e deployer podem ser mesma pessoa



Em destaque: principais papéis relevantes para este curso

API J2EE (pacotes top-level)

Disponíveis em *j2ee.jar*

- *javax.activation* *JavaBeans Activation Framework usado pelo JavaMail*
- *javax.ejb* *Classes e interfaces para construir EJBs*
- *javax.jms* *Classes e interfaces para construir aplicações JMS*
- *javax.mail* *Classes que modelam um cliente de e-mail*
- *javax.resource* *JCA: API para desenvolvimento de conectores (RARs)*
- *javax.security.auth* *JAAS: API de autenticação e autorização*
- *javax.servlet* *Classes e interfaces para construir servlets e páginas JSP*
- *javax.sql* *Pacote JDBC estendido*
- *javax.transaction* *JTA: API para controle de transações*
- *javax.xml.parsers* *JAXP: Classes para processamento XML*
- *javax.xml.transform* *Classes para processamento de transformações XSLT*
- *org.w3c.dom* *Implementação de DOM, componente do JAXP.*
- *org.xml.sax* *Implementação de SAX, componente do JAXP*

Componentes EJB e EJB-JAR

➡ Cada **componente EJB** contém

- **Um Enterprise Bean** (`javax.ejb.EnterpriseBean`)
 - Implementa a lógica da aplicação em uma das três sub-interfaces de `EnterpriseBean`: `EntityBean`, `SessionBean` ou `MessageDrivenBean`
- **Uma Interface Home** (`javax.ejb.EJBHome`)
 - Fábrica usada pelos clientes para obter instâncias do EJB
- **Uma Interface Remote** (`javax.ejb.EJBObject`) e/ou
 - Declara os métodos da interface de negócio de cada bean
- **Interfaces locais** (`javax.ejb.EJBLocalObject` e `EJBLocalHome`)
 - Alternativas às interface Home e Remote para acesso local eficiente
- Uma entrada no **Deployment Descriptor** (`ejb-jar.xml`)
 - Arquivo XML que associa cada uma das classes acima com seu enterprise bean e permite definir e configurar o uso de serviços como transações, persistência, permissões de acesso, referências, etc.

➡ As classes, com o `ejb-jar.xml` são guardadas em um JAR

Web Application Archive e Client-JAR

- ➡ **Componentes Web** armazenados em arquivos **WAR**, contém
 - Páginas JSP, páginas HTML, imagens, arquivos Flash, Applets
 - Servlets, JavaBeans e outras classes
 - JARs contendo classes usadas pelos componentes
 - Bibliotecas de tags para JSP (Taglibs)
 - Tag library descriptors (*.tld) para cada biblioteca incluída
 - Web Deployment Descriptor (web.xml), que descreve os mapeamentos entre servlets e nomes de contexto, filtros, parâmetros iniciais, referências com EJBs e outras informações
- ➡ **Componentes Cliente*** são armazenados em arquivos JAR
 - Application Client Descriptor (application-client.xml), armazenado junto com as classes cliente descrevem as referências para EJBs chamados, associando-os a nomes JNDI no domínio java:comp/env

* clientes EJB e clientes Web não precisam ser "componentes J2EE" (que rodam em container)
Se um servidor oferece um container para aplicações cliente, o desenvolvimento do cliente como componente será facilitado pois o mesmo poderá usar serviços do seu container

- *EJB-JARs, JARs de componente-cliente e WARs podem ser empacotados em JARs chamados de arquivos **EAR***
- *Um arquivo EAR deve conter*
 - *Os JARs, WARs e EJB-JARs da aplicação*
 - *Bibliotecas adicionais opcionais*
 - *Um Application Deployment Descriptor (**application.xml**), que descreve cada módulo (contém o nome do JAR, WAR ou EJB-JAR dentro de um tag que descreve seu tipo)*
- *Além desses arquivos EJB-JARs, WARs e EARs podem conter arquivos dependentes do fabricante*



















Arquivos dependentes do fabricante

- O fabricante de um servidor de aplicações pode definir outros arquivos necessários para a sua implantação
 - Geralmente consistem de um ou mais arquivos XML que devem ser embutidos nos EJB-JARs, WARs ou EARs
 - Nos servidores comerciais (e no J2EE Reference Implementation) os arquivos podem ser gerados e automaticamente incluídos nos JARs
- Os arquivos servem para configurar os componentes para usarem serviços proprietários ou para redefinir valores default (como nomes de contextos JNDI)
- No JBoss, o arquivo **jboss.xml** é usado para este fim
 - Outros dois arquivos: jboss-web.xml e jboss-cmp-jdbc.xml configuram recursos adicionais (ambos são opcionais)
 - O arquivo jboss.xml deve ser incluído no META-INF do EJB-JAR.

Como implantar (deploy) uma aplicação

- Este é um processo **dependente do servidor**
 - Basicamente, o servidor registra o EJB-JAR, WAR ou EAR, que contém todas as informações necessárias para a implantação do componente guardadas nos seus deployment descriptors
 - Cada fabricante tem uma maneira diferente de fazer esse registro
- No ambiente de desenvolvimento do J2EE (Reference Implementation)
 - Use o deployment wizard na ferramenta deploytool
- No JBoss
 - Simplesmente jogue o JAR, WAR ou EAR no diretório "deploy"

Estrutura do pacote Sun J2EE SDK

 bin	Executáveis
 j2ee	Servidor Sun J2EE Reference Implementation
 cloudscape	Servidor de banco de dados Informix Cloudscape
 runclient	Ferramenta para execução de clientes com autenticação
 cleanup	Esvazia o cache
 verifier	Verificador de JARs (diagnostica erros em JARs, EARs)
 deploytool	Ferramenta gráfica para deployment e montagem
 j2eeadmin	Ferramenta de administração (datasources, destinos JMS)
 packager	Empacotador de JARs, EARs, WARs, RARs
 realmtool	Gerencia usuários, grupos, etc.
 keytool	Gerencia chaves de criptografia
 cloudscape	Diretório raiz do servidor de banco de dados Cloudscape
 doc	Documentação do J2EE SDK
 api	Documentação da API
 config	Arquivos .properties com configuração para aplicações
 lib	JARs usados em aplicações J2EE (contém o j2ee.jar)
 public_html	Raiz do servidor Web
 repository	Repositório de dados e cache do servidor

Configuração: variáveis de ambiente

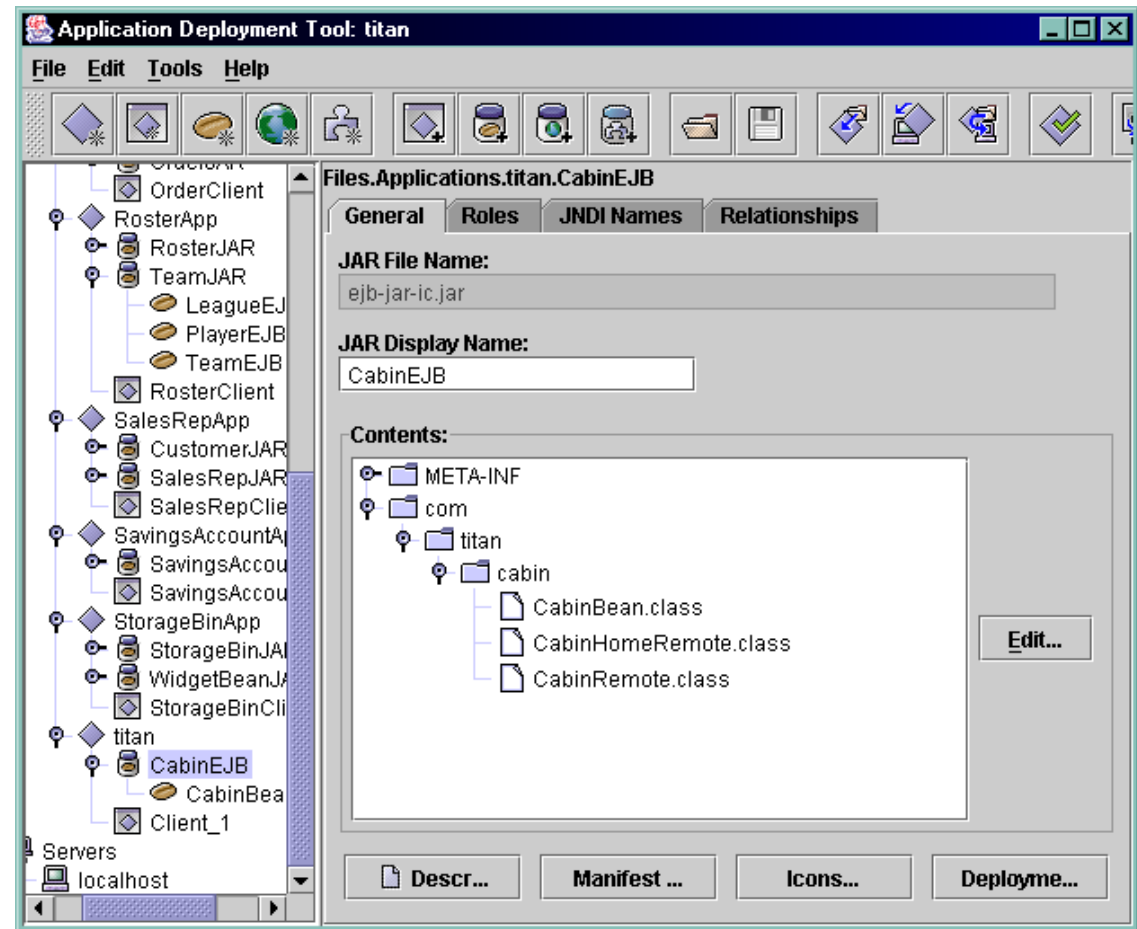
- *Para todas as aplicações*
 - **ANT_HOME**: deve apontar para o diretório de instalação do ANT (ex: c:\jakarta-ant1.5Beta)
 - **JAVA_HOME**: deve apontar para o diretório de instalação do JSDK (ex: c:\j2sdk1.4.0)
 - **PATH** contendo ANT_HOME/bin e JAVA_HOME/bin
- *Para o J2EE Reference Implementation*
 - **J2EE_HOME**: diretório de instalação do J2SDK EE
 - **PATH** contendo J2EE_HOME/bin
- *Para compilar, use o CLASSPATH*
 - \$J2EE_HOME/lib/j2ee.jar
- *Para executar aplicações, use os JARs indicados pelo seu servidor de aplicações*
 - JBoss: JARs específicos localizados em \$JBOSS_HOME/lib/ e client/
 - J2EE: \$J2EE_HOME/lib/j2ee.jar e \$J2EE_HOME/lib/locale

J2EE Reference Implementation

- *É um servidor completo que implementa a especificação J2EE*
 - *Você geralmente não precisa do J2EE SDK se já tem um servidor de aplicação compatível com a especificação J2EE*
- *Contém*
 - *Ferramentas de montagem e instalação para o servidor J2EE RI*
 - *EJB container, App Client container e Servlet container (Tomcat)*
 - *Servidor de banco de dados (Cloudscape)*
 - *Serviços de transações (JTS), persistência (CMP), autenticação e autorização (JAAS) e outros serviços básicos*
- *Para iniciar/parar o servidor*
 - > **j2ee** -verbose *Exibe mensagens durante a execução*
 - > **j2ee** -stop *Interrompe o servidor*
- *Web container:*
 - *Porta 8000: servidor Web*
 - *Porta 7000: servidor Web seguro*

J2EE: deployment tool (deploytool)

- Objetivo: facilitar a criação, montagem, implantação e manutenção de aplicações J2EE **no servidor J2EE RI**
 - Pode também ser usado para montar componentes (WAR, EJB-JAR, EAR, etc.) para outros servidores também (os que não oferecem interface equivalente como o JBoss*)
- Para executar:
 - > **deploytool** &



* neste caso pode ser preciso substituir os arquivos ***-ri.xml** (do servidor da Sun) gerados por arquivos equivalentes **jboss*.xml** (não precisa removê-los, basta acrescentar os outros)

Packager e Administration Tool

- **packager***: ferramenta de linha de comando para gerar EJB-JARs, WARs, EARs e RARs portáteis (ou não)
 - > **packager** -<opção> <parâmetros>
 - Use <opção> -ejbJar para gerar um EJB-JAR
 - Use <opção> -webArchive para gerar um WAR
 - Use <opção> -enterpriseArchive para gerar um EAR
- **j2eeadmin**: ferramenta que adiciona, remove e lista recursos (acessíveis via JNDI) no sistema de nomes servidor J2EE RI
 - > **j2eeadmin** -add|-list|-remove<recurso>
- **verifier**: verifica se há erros em um JAR, WAR ou EAR
- Veja exemplos de sintaxe do packager, j2eeadmin e outras ferramentas no Apêndice A do Java Tutorial: J2EE SDK Tools

***Usuários Windows:** Pode haver conflito com o packager.exe e verifier.exe que fazem parte da instalação do Windows. Sugestão: mude os nomes para **pack.bat** e **verify.bat**

ANT: tarefas relacionadas com J2EE

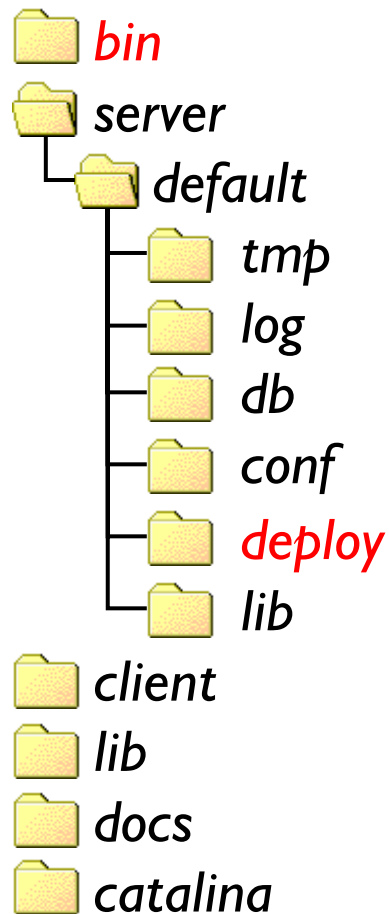
- Uma forma mais simples (e portátil) de criar componentes JAR, WAR, etc. é criar alvos para o **Ant** em arquivos **build.xml**:

```
<ear destfile="app.ear" appxml="application.xml">  
    <fileset dir="${build.dir}" includes="*.jar,*.war"/>  
</ear>
```
- ```
<jar destfile="${dist}/lib/app.jar">
 <fileset dir="${build}/classes"/>
</jar>
```
- ```
<war destfile="myapp.war" webxml="meta/web.xml">  
    <fileset dir="src/jsp/myapp"/>  
    <lib dir="jars" /> <classes dir="build/main"/>  
    <zipfileset dir="images/gifs" prefix="images"/>  
</war>
```
- ```
<ejbjar srcdir="${build}" descriptor="xml.dir">
 <jboss destdir="${deployjars.dir}" />
</ejbjar>
```
- Mais informações sobre como usar essas tarefas no manual do Ant em [\\$ANT\\_HOME/docs/manual/index.html](http://$ANT_HOME/docs/manual/index.html)

- *Servidor J2EE Open Source*
  - *Líder de mercado*
  - *Não certificado oficialmente pela Sun*
  - *Vencedor do prêmio JavaWorld (concorrendo com BEA, IBM, Sun e outros)*
- *Onde conseguir*
  - **[www.jboss.org](http://www.jboss.org)**
- *Instalação e administração*
  - *Abra o ZIP em algum lugar de sua máquina, mude para o diretório bin da instalação e rode 'run'*
  - *O JBoss geralmente não precisa de pré-configuração para funcionar*

- **Documentação oficial**
  - A documentação do JBoss é usada para financiar o projeto Open Source. Na versão atual são três livros de US\$ 10.00 cada um. Para a maior parte dos ambientes, o primeiro livro é suficiente (os outros dois lidam com clustering e implementação de persistência)
- **Documentação on-line (não suportada oficialmente)**
  - Manual on-line: [www.jboss.org/online-manual/HTML/](http://www.jboss.org/online-manual/HTML/)
- **Administração e configuração**
  - Configuração pode ser feita nos arquivos .xml no diretório **conf** ou via interface Web (na porta 8082)
  - A interface de administração do JBoss é baseada em JMX (Java Management Extensions): MBeans

# Estrutura de diretórios JBoss 3.0.0



Executáveis. **Para iniciar o JBoss, rode 'run' neste diretório**

Contém pastas de servidores

Servidor default (se você usa clustering tem outros)

Cache do servidor. Contém todas as aplicações instaladas

Contém os logs de acesso ao container EJB

Repositório para informações persistentes

Arquivos de configuração do servidor (XML/MBean)

Diretório de "hot deployment": **Joque seu JAR ou EAR aqui!**

JARs usados nas aplicações J2EE que rodam no JBoss

JARs usados em aplicações-cliente

JARs usados pelo servidor JBoss

Contém os DTDs e exemplos de conectores para serviços

Diretório raiz do Jakarta-Tomcat (opcional)

- O diretório **catalina** só existe nas versões com Tomcat
- A estrutura acima é diferente para os servidores JBoss 2.4.x

- Nesta seção iremos demonstrar a construção, montagem e implementação de aplicações J2EE
- Três exemplos
  - Componente **EJB**: Session bean
  - Componente **Web**: Página JSP
  - Componente J2EE: Enterprise Archive (**EAR**)
- O código-fonte de todos os exemplos está em **cap01/**
- O roteiro que explica os detalhes do código e da implementação está nos livros-texto
  - Veja cópias em PDF e ZIP no subdiretório docs/ebooks e docs/tutorials do CD

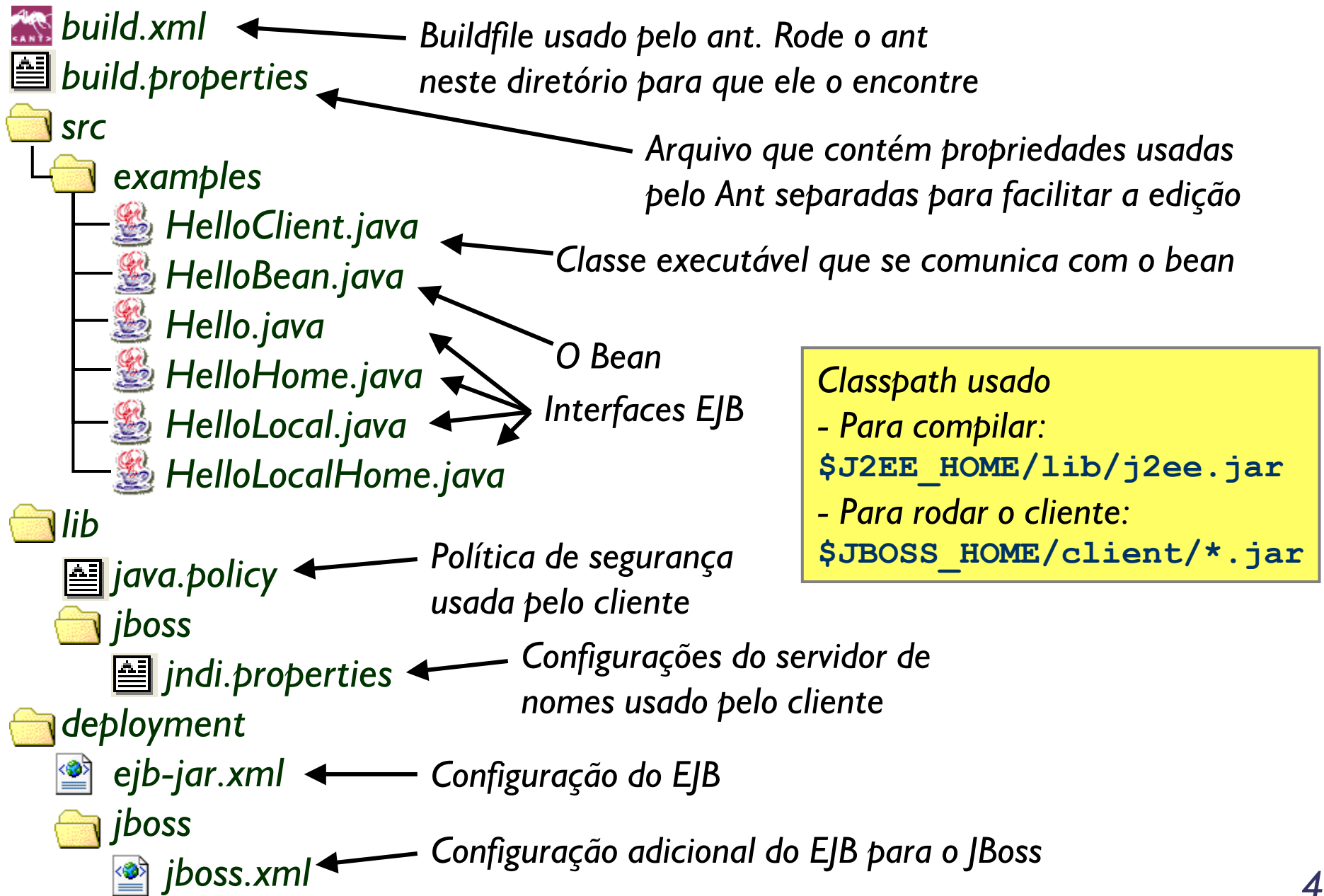


# Exemplo 1: Componente EJB

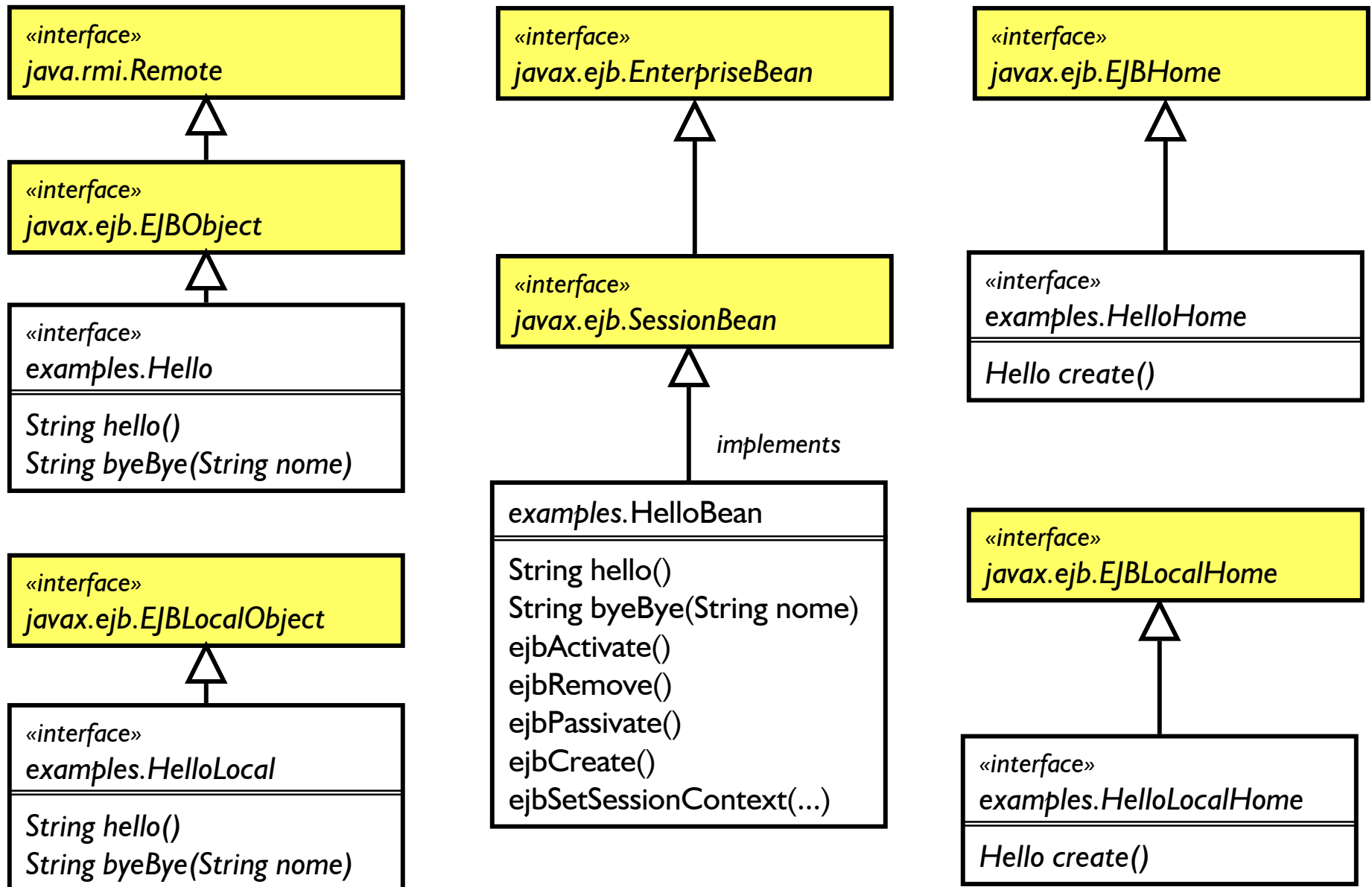
- Para este exemplo utilizaremos o roteiro no capítulo 3 do livro-texto "**Mastering EJB 2**" (consulte) para criar, empacotar e implantar um Session Bean no JBoss
- Os arquivos estão em
  - `cap01/mejb2/`
- Para montar a aplicação usamos o Ant\*
  - > `ant buildjboss`que monta o `ejb-jar`. Depois é só copiar para o diretório de deployment do JBoss. O ant faz isto também
  - > `ant jbossdeploy`
- Para rodar a aplicação cliente (e ver se o EJB funciona) use, mais uma vez, o ant:
  - > `ant runjbossclient`

\* Veja que é preciso configurar o arquivo `build.properties` com informações do seu sistema

# Estrutura da aplicação



# Classes e interfaces



# Deployment descriptors

```
<!DOCTYPE ejb-jar PUBLIC
"-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
"http://java.sun.com/dtd/ejb-jar_2_0.dtd">

<ejb-jar>
 <enterprise-beans>
 <session>
 <ejb-name>Hello</ejb-name>
 <home>examples.HelloHome</home>
 <remote>examples.Hello</remote>
 <local-home>examples.HelloLocalHome</local-home>
 <local>examples.HelloLocal</local>
 <ejb-class>examples.HelloBean</ejb-class>
 <session-type>Stateless</session-type>
 <transaction-type>Container</transaction-type>
 </session>
 </enterprise-beans>
</ejb-jar>
```

ejb-jar.xml

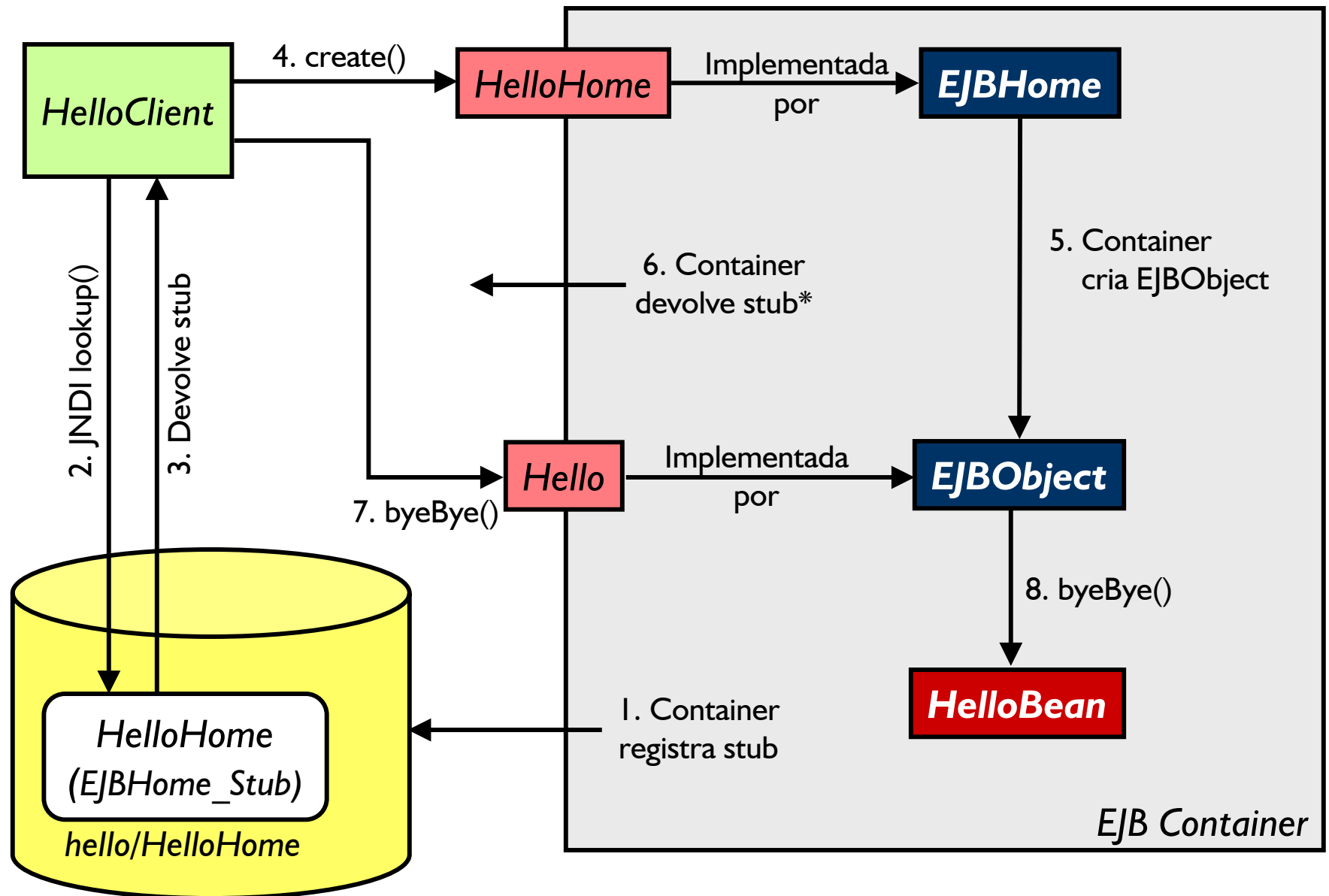
```
<jboss>
 <enterprise-beans>
 <session>
 <ejb-name>Hello</ejb-name>
 <jndi-name>hello/HelloHome</jndi-name>
 </session>
 </enterprise-beans>
</jboss>
```

jboss.xml

# Configuração do cliente

- O JBoss não oferece um container para a execução do cliente (como faz o servidor da Sun)
- É preciso que a aplicação cliente
  - saiba onde está o servidor de nomes
  - implemente autenticação se necessário
- Se o arquivo *jndi.properties* estiver no classpath ele será usado pelo cliente. O arquivo contém
  - URL contendo nome do servidor onde roda o serviço de nomes
  - Driver JNDI (nome da classe a ser usada) e pacotes adicionais (é preciso que o JAR onde essas classes estão esteja no CLASSPATH)
- O CLASSPATH do cliente deve enxergar
  - As interfaces Remote e Home do bean
  - JARs que contém drivers de serviços usados (JNDI, JAAS, etc.)
  - Possíveis arquivos de configuração (ex: domínio JAAS)

# Funcionamento



# Exemplo 2: Componente Web

- Neste exemplo criaremos um **componente JSP** muito simples e empacotaremos em um arquivo WAR

- 1. Criamos primeiro a seguinte estrutura de diretórios



➡ Use o DTD: web-app\_2\_3.dtd

- 2. No diretório WEB-INF colocamos o seguinte arquivo **web.xml**

```
<web-app></web-app>
```

web.xml

- 3. Na pasta websrc colocamos o seguinte arquivo **index.jsp**

```
<% String msg = "World!";
String parameter = null;
if ((parameter = request.getParameter("nome")) != null) {
 msg = parameter;
}
%>
<h1>Hello, <%=msg %></h1>
```

index.jsp

- 4. Jogamos todo o conteúdo de websrc em um JAR (com extensão WAR)  
> jar cf **hello.war** -C websrc .
- 5. Copiamos o arquivo WAR para o diretório **deploy** do Tomcat ou JBoss
- 6. Abrimos o browser apontando na URL

<http://localhost:8080/hello/index.jsp?nome=Seu+Nome>

## Exemplo 3: Componente Enterprise

- Combinando os dois exemplos anteriores, montaremos um componente EAR fazendo com que o JSP chame o Session Bean criado no primeiro exemplo.
- Este exemplo é o mesmo mostrado na *primeira parte do J2EE Tutorial\** (Getting Started) só que vamos implantar o componente no JBoss
- Os arquivos estão em
  - *cap01/sun/*
- Mais uma vez, automatizamos todo o processo no Ant
  - > `ant jboss.deploy`
- compila tudo, monta o EJB-JAR, o WAR, o EAR e joga no diretório deploy. Para rodar o cliente
  - > `ant run.jboss.client`

\* consulte o tutorial para explicações sobre o código-fonte



- Apesar de serem simples os exemplos vistos neste módulo, eles ilustram todo o processo de desenvolvimento J2EE
  - Codificação das interfaces, enterprise bean e componentes Web
  - Empacotamento em EJB-JARs, WARs e EARs
  - Configuração dos componentes através de deployment descriptors
  - Implantação (deployment) em um servidor de aplicações
- Já temos, portanto, bons fundamentos teóricos e alguma experiência prática para começar a desenvolver e montar aplicações J2EE
- Nos próximos três módulos veremos tecnologias que dão suporte aos componentes J2EE:
  - **JNDI**, que oferece acesso ao serviço de nomes
  - **RMI-IIOP**, que permite a comunicação entre Enterprise Beans, e
  - **JMS**, que permite a comunicação assíncrona através de Enterprise Beans especiais chamados de Message Driven Beans

- 1. Crie um session bean que calcule os juros incidentes sobre um valor principal dada a taxa de juros/período e quantidade de períodos. A fórmula é  
$$\text{principal} * ((\text{taxa} + 1)^{\text{períodos}} - 1)$$
- 2. Empacote o bean e faça o deployment no JBoss
- 3. Escreva uma aplicação cliente (linha de comando) que receba os três valores, conecte-se ao servidor e imprima o resultado
- 4. Escreva uma aplicação cliente Web (JSP) que utilize uma interface formada por um formulário com três campos e um botão para interagir com o bean (empacote em um WAR).
- 5. Empacote tudo em um EAR e faça o deployment no JBoss

- [1] Bill Shannon. *J2EE Specification*. Sun Microsystems. <http://java.sun.com/j2ee/>
- [2] Richard Monson-Haefel. *Enterprise JavaBeans, 3rd. Edition*. O'Reilly, 2001. *Uma das mais importantes referências sobre EJB*
- [3] Sun Microsystems. *Simplified Guide to the J2EE*. <http://java.sun.com/j2ee/>. *White paper com introdução a J2EE.*
- [4] Ed Roman et al. *Mastering EJB 2, Chaps. 1 to 3*  
<http://www.theserverside.com/books/masteringEJB/index.jsp>  
*Contém ótima introdução a middleware, motivação e fundamentos de EJB*
- [5] Rossana Lee. *The J2EE Tutorial*, Sun Microsystems.  
<http://java.sun.com/j2ee/tutorial/>. *Roteiro principal dos exemplos e exercícios.*
- [6] Kevin Boone, et al. *JBoss User's Manual: Chapter 1: First Steps*.  
<http://www.jboss.org/online-manual/HTML/ch01.html>. *Passo-a-passo para montar e instalar um EJB no JBoss*
- [7] Bruce Eckel. *Thinking in Java 2*. <http://www.bruceeckel.com>. *Capítulo final dá uma visão geral da tecnologia J2EE (versão 1.2)*
- [8] Duane Fields e Mark Kolb. *Web Development with JavaServer Pages*, Manning, 2000. *Referência sobre JSP.*

*helder@ibpinet.net*

***www.argonavis.com.br***

*Introdução a J2EE, 2000, 2001, 2002*  
*Atualizado em Junho de 2002*