J530 - Enterprise JavaBeans

Introdução a J2EE e servidores de aplicação EJB

Helder da Rocha (helder@acm.org)

argonavis.com.br

Objetivos

- Este módulo tem três objetivos
  - Configurar o ambiente de trabalho
  - Oferecer uma visão geral da tecnologia J2EE/EJB definindo os seus conceitos fundamentais e mostrando quais os problemas que a tecnologia pretende solucionar
  - Instalar uma aplicação EJB no JBoss e descrever seus principais componentes
  - Instalar aplicações Web e de Enterprise no JBoss
- Os exemplos utilizados neste módulo foram extraídos de livros e tutoriais
  - Exemplo do capítulo 3 do livro Mastering EJB2
  - Exemplos do capítulo 1 do livro J2EE Tutorial (alterados para implantação no JBoss)

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. E outras precisam ser baixadas separadamente.*

Banco de dados

- JDBC é a API padrão
  - Pacote *java.sql* (no J2SE)
  - 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* (no J2EE)
  - 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("Banco");
Connection con = ds.getConnection();
```

Data Source Name publicado em JNDI

Objetos distribuídos

- *Java RMI sobre JRMP** (*java.rmi no J2SE*)
  - *Solução 100% Java para comunicação em rede*
  - *Muito mais simples e fácil de manter e expandir que soluções baseadas em sockets*
- *Java IDL* (*org.omg.CORBA e org.omg.CosNaming no J2SE*)
  - *Implementação de CORBA em Java*
  - *Solução para integração com aplicações escritas em outras linguagens*
- *RMI sobre IIOP* (*javax.rmi no J2SE*)
  - *Modelo de programação RMI com execução CORBA*
- *EJB - Enterprise JavaBeans* (*javax.ejb no J2EE*)
  - *Arquitetura de componentes baseada em RMI sobre IIOP*

* *Java Remote Method Protocol*

Serviços distribuídos

- JTS (Java Transaction Service) (org.omg.CosTransactions)
  - Serviço de transações distribuído compatível com CORBA Transaction Service (OTS)
- JTA (Java Transaction API) (javax.transaction no J2EE)
  - API de programação Java que implementa o padrão XA do Open Group para transações distribuídas (two phase commit protocol) e oferece acesso mais simples a JTS
- DGC (Distributed Garbage Collection) (java.rmi.dgc)
  - Remove referências remotas que não podem mais ser usadas
  - Disponível apenas em aplicações Java RMI
- JNDI (Java Naming & Directory Interface) (javax.naming no J2SE)
  - Interface padrão genérica para sistemas de nomes
- JAAS (Java Authorization & Authentication Services) (J2SE)
  - API para implementação de sistemas de login com reconhecimento de identidade do usuário.

Eventos distribuídos

- JMS (Java Message Service) (javax.jms no J2EE)
  - "Message" == "Event" em ambientes distribuídos. Viabiliza baseada em notificação (padrão Observer)
  - API que permite interagir com diferentes sistemas de Message Oriented Middleware (MOM)
  - Permite implementar sistemas distribuídos com comunicação assíncrona
  - Alternativa aos protocolos síncronos (IIOP e RMI/JRMP)

Serviços Internet (Web e email)

- Componentes Web (javax.servlet.http e javax.servlet.jsp no J2EE)
  - 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 (javax.xml.* em J2SE, J2EE e pacotes a parte)
  - 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 (javax.mail em J2EE)
  - API para a programação de clientes SMTP e IMAP

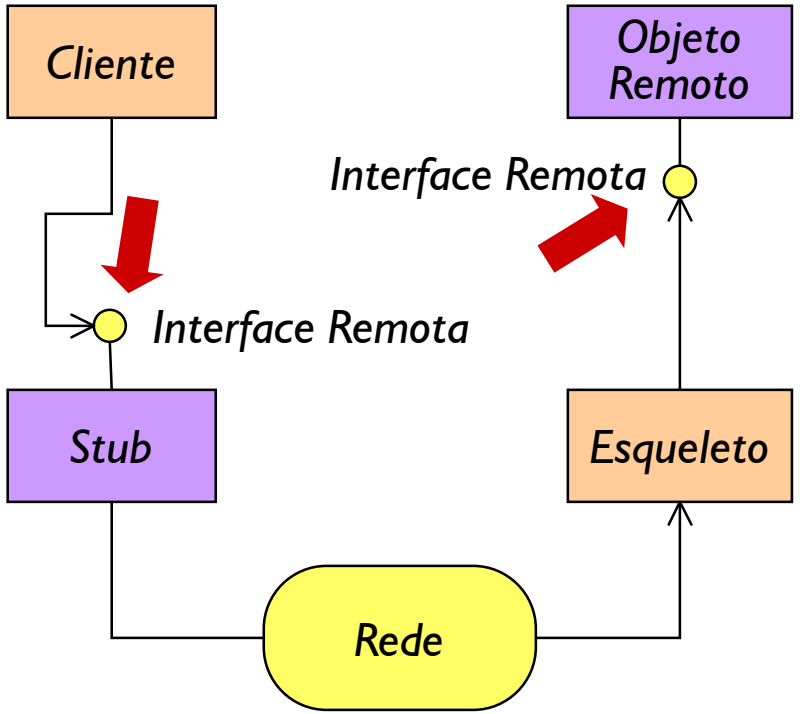
Servidores de aplicação

- *Servidores de aplicação OO, também chamados de Component Transaction Monitors [2], oferecem ambientes para operação de componentes (rodando em containers) e diversos serviços de middleware*
- *Servidores de aplicação que suportam EJB oferecem 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 (que podem ser gerados)*
- *Os serviços também podem ser usados de forma explícita*

** Cuja configuração não requer o uso de programação*

Por que EJB? (1)

- Sistemas de objetos distribuídos têm em comum uma arquitetura que consiste de
  - O objeto remoto
  - Um stub que representa o objeto remoto no cliente
  - Um esqueleto, que representa o cliente no servidor
- O cliente conversa com o stub pensando tratar-se do próprio objeto



- O esqueleto conversa com o objeto remoto que pensa que é o cliente quem fala com ele
- O stub e o objeto remoto implementam a **mesma interface**!

Fonte da ilustração: [4] Ed Roman, Mastering EJB 2

Por que EJB? (2)

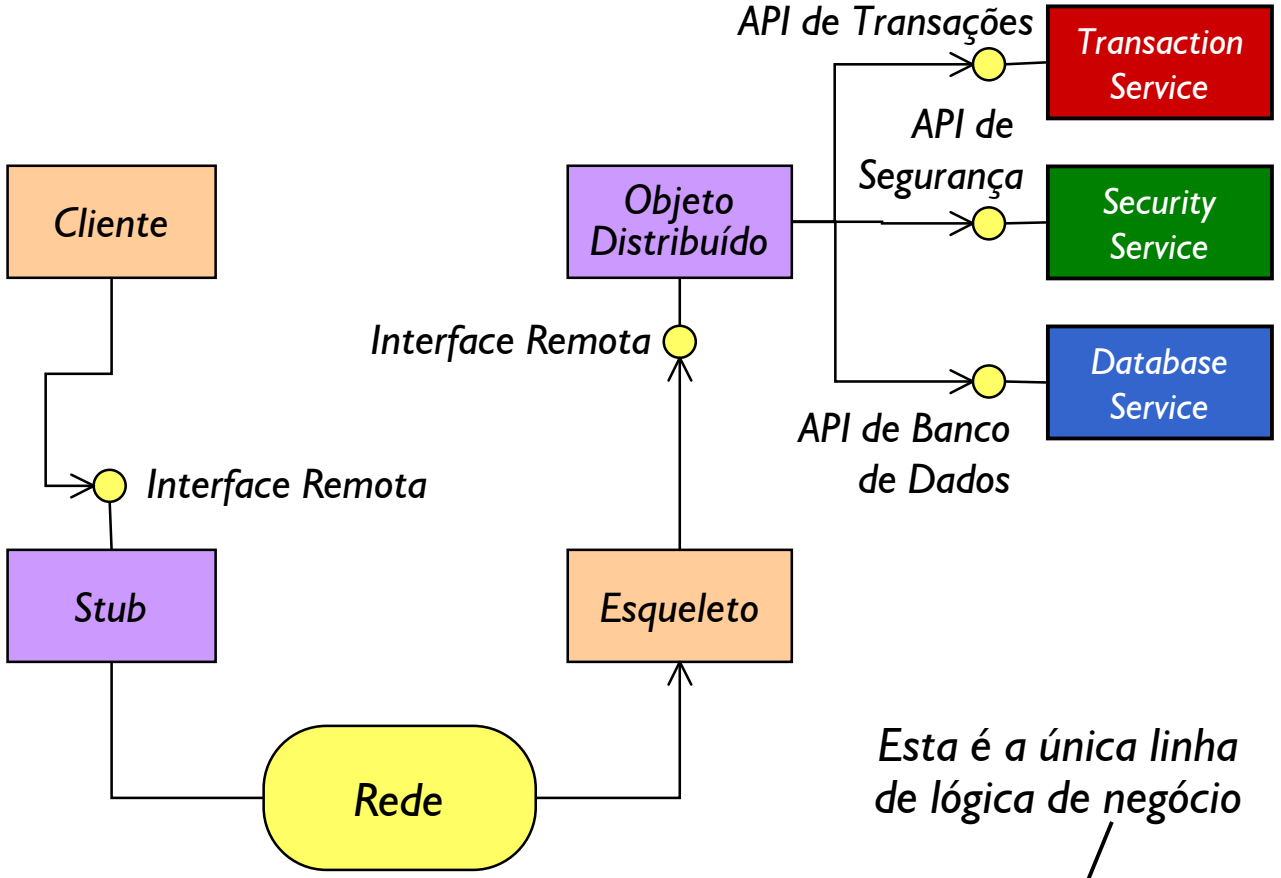
- Desenvolver um sistema distribuído não é simples.
  - Exige se concentrar em aspectos da aplicação que nada tem a ver com o problema de negócio a ser solucionado
  - Exige a observação de condições que geralmente são menos importantes ou dispensáveis em aplicações locais
- É preciso se preocupar com
  - A performance, que pode sofrer abalos em rede
  - Os custos de uma possível ampliação da capacidade
  - A segurança dos dados, controle de acesso, permissões
  - Transações: tarefas poderão falhar no meio do caminho
  - Integridade dos dados: clientes irão acessá-los simultaneamente
  - O que acontecerá se parte do sistema sair do ar?
- Mas esses são problemas de qualquer sistema distribuído
  - Será que precisamos implementar tudo isto?
  - Não! Use middleware!

Tipos de middleware

- Temos uma aplicação com vários objetos distribuídos e um servidor que oferece serviços de middleware. E agora? Como acessar os serviços?
- Serviços podem ser acessados através de uma API
  - Sua aplicação precisará escrever o código para usar a API, por exemplo, JDBC para controlar acesso a bancos de dados, ou JTA para controlar transações, ou APIs de segurança: middleware explícito
- Serviços podem ser configurados através de interfaces ou arquivos de configuração externos ao código
  - Sua aplicação não precisa conter código algum de acesso a recursos externos. Você declara os serviços que precisa externamente: middleware implícito

Middleware explícito

- *Tem que ser implementado em cada método de cada objeto que deseja utilizar o serviço*



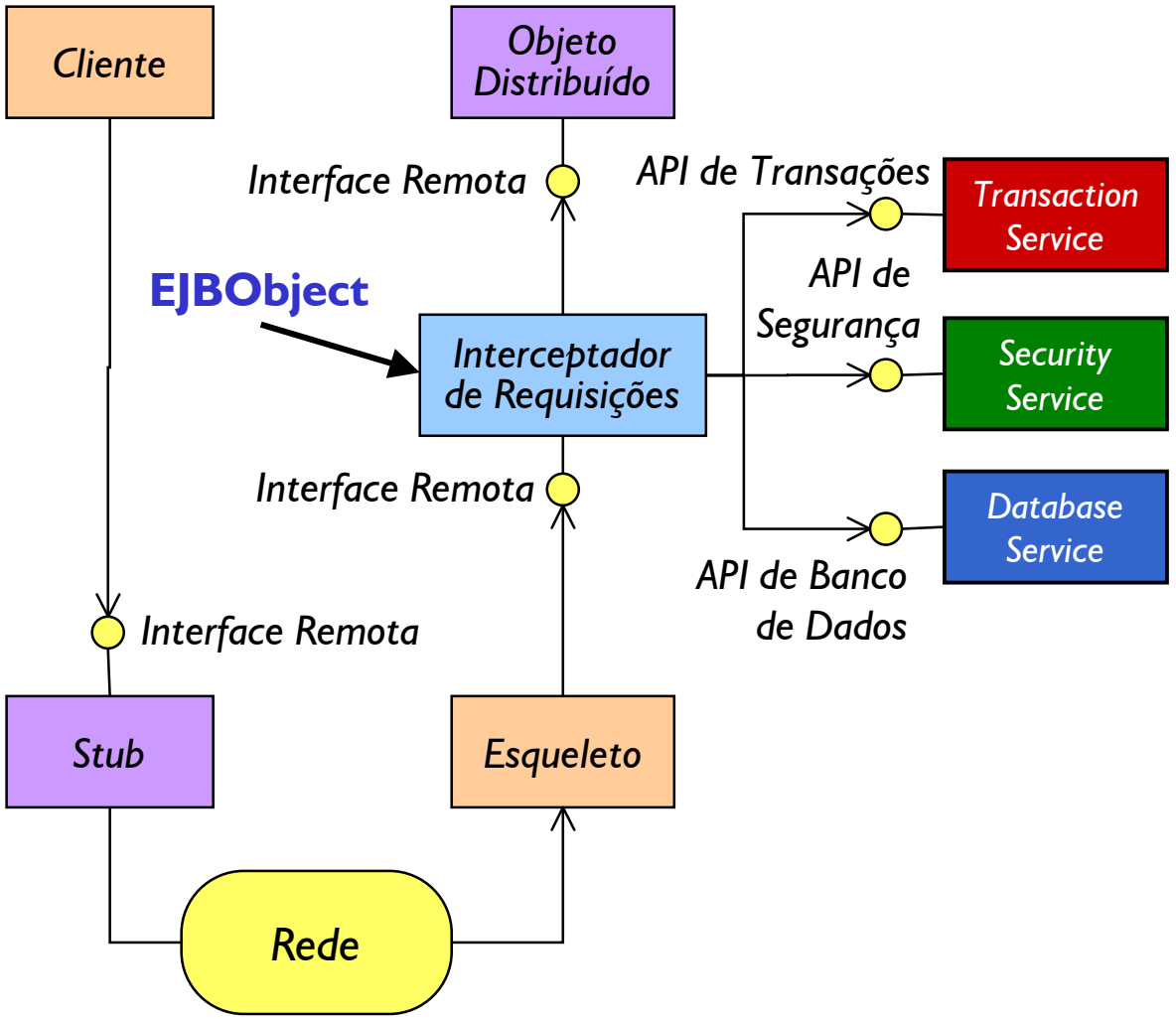
Esta é a única linha de lógica de negócio

```
public void transferir(Conta um, Conta dois, double valor) {
   // chama API para verificar segurança da requisição
   // chama API para iniciar a transação
   // chama API para carregar dados do banco
   // Subtraia o valor do saldo da conta um e some na conta dois
   // chama API para guardar dados no banco
   // chama API para fechar a transação
}
```

Fonte da ilustração: [4]

Middleware implícito

- Escreva seu objeto para conter apenas lógica de negócio
- Declare os serviços de middleware que seu objeto precisa em um arquivo separado
- Rode uma ferramenta que pega o descritor e gera um objeto interceptador de requisições




```
public void transferir(Conta um, Conta dois, double valor) {
   // Subtraia o valor do saldo da conta um e some na conta dois
}
```

Fonte da ilustração: [4]

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" ?

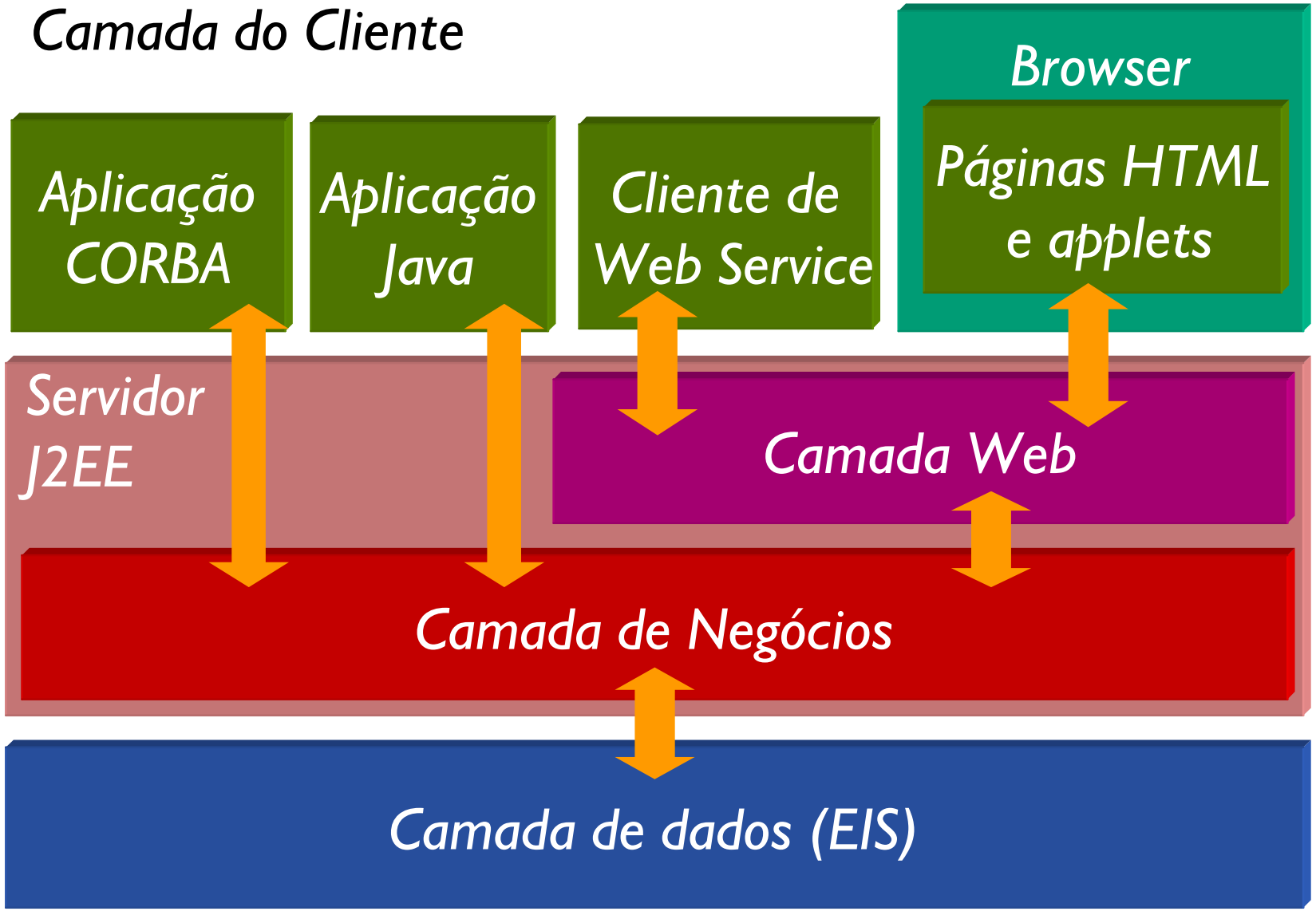
- *É um contrato entre o fornecedor e o usuário*
  - *Permitem que os fornecedores construam componentes compatíveis com uma classe de servidores: reuso!*
  - *Permite que os usuários substituam componentes existentes por outros similares: flexibilidade!*
- *Servidores de aplicações são usuários de componentes*
  - *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
  - É uma espécie de Sistema Operacional para objetos
  - Antes que um componente EJB ou Web possa ser executado em um container J2EE ele precisa ser implantado (deployed) no container
- O container é responsável por chamar os métodos (callback) que controlam o ciclo de vida dos componentes
- O container também é quem serve de interface para que o componente utilize serviços de middleware implícito declarados nos seus arquivos de configuração
- A plataforma J2EE define três tipos de containers
  - Container EJB
  - Container Web
  - Container Cliente*

* Definição é vaga e suporte opcional

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*

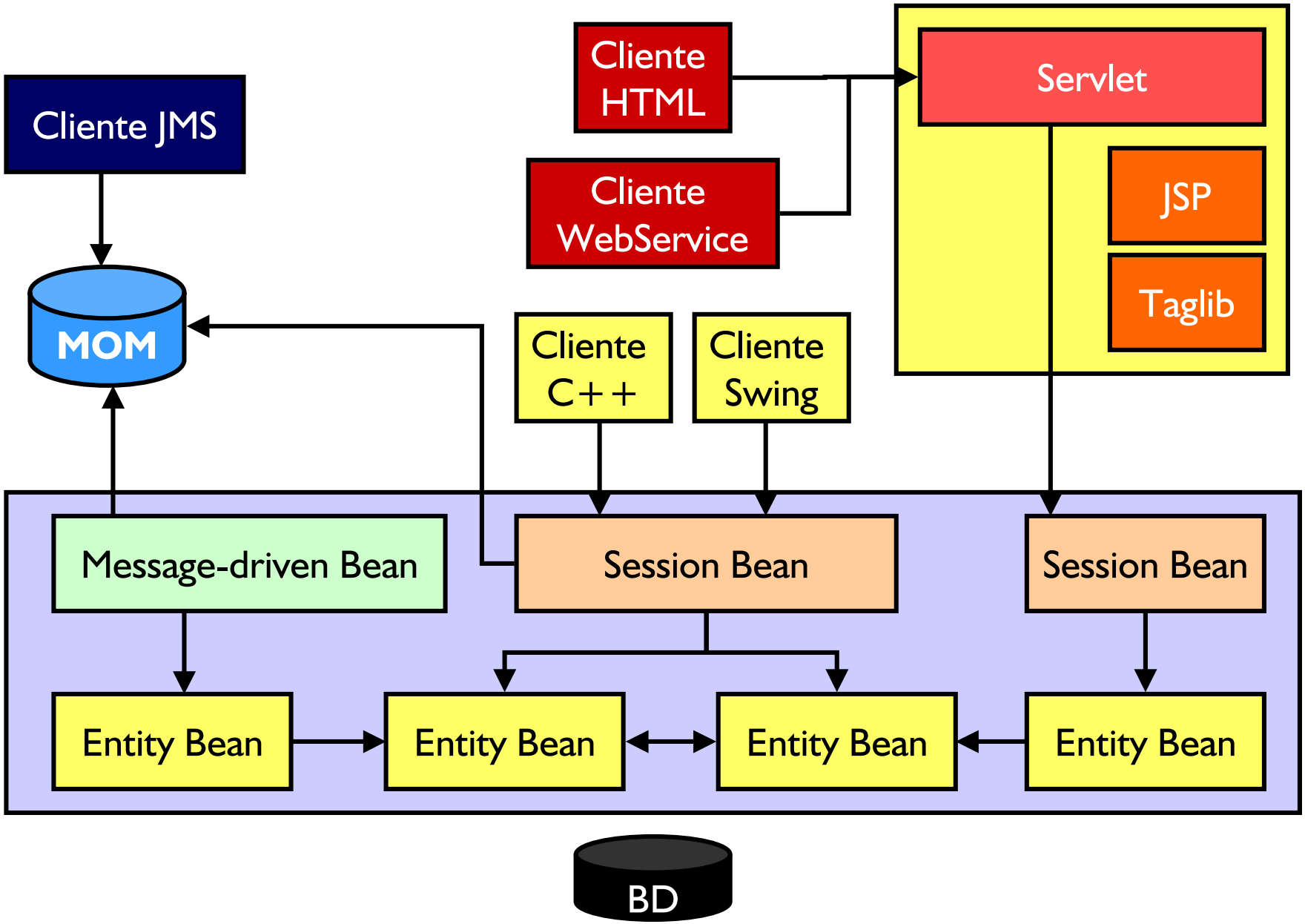
** Common Object Request Broker Architecture - padrão OMG para objetos distribuídos*

*** 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

Uso típico dos diferentes EJBs



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

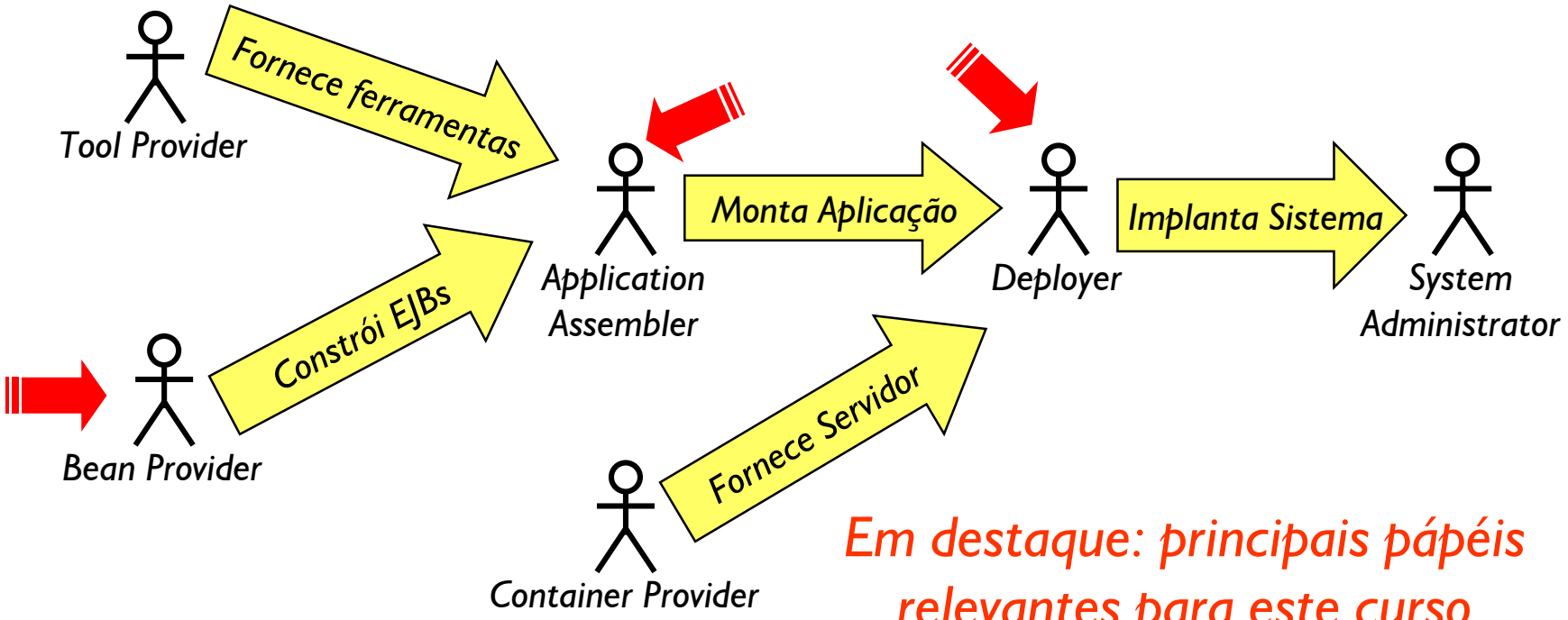
- Provedor de componentes (bean provider)
  - Desenvovedor que cria os componentes J2EE
- Provedor de ferramentas (tool provider)
  - Oferecem ferramentas para facilitar a montagem e manutenção das aplicações J2EE
  - Ferramentas de modelagem de dados, ferramentas de montagem de componentes, etc.
- Montador 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 pápéis relevantes para este curso

Fonte da ilustração: [4] Ed Roman et al. Mastering EJB 2, pag. 21

Componentes EJB e EJB-JAR

- Cada **componente EJB** contém
  - Uma Classe 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) e
    - 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

- **Componentes Web** guardados em arquivos WAR, contém
  - Páginas JSP, páginas HTML, imagens GIF e JPEG, arquivos Flash, Applets, JARs com 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

Enterprise Archive

- 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 utilitárias 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 jbosscmp-jdbc.xml configuram recursos adicionais (ambos são opcionais)
  - O arquivo jboss.xml deve ser incluido de preferência no diretório META-INF do EJB-JAR.

Como implantar (deploy) uma aplicação

- Este é um processo dependente do servidor
  - O servidor abre o EJB-JAR, WAR ou EAR, e extrai as informações que precisa do deployment descriptor para configurar o componente e serviços
  - Cada fabricante tem uma maneira diferente de fazer a implantação, que geralmente inclui a geração automática de classes e o registro dos componentes no serviço de nomes (JNDI).
- Hot deployment no JBoss
  - Simplesmente joque o JAR, WAR ou EAR no diretório "deploy" do servidor e a aplicação se instala sozinha.

J2EE Reference Implementation

- *É um servidor completo que implementa a especificação J2EE*
  - *Você 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*

API J2EE (pacotes top-level)

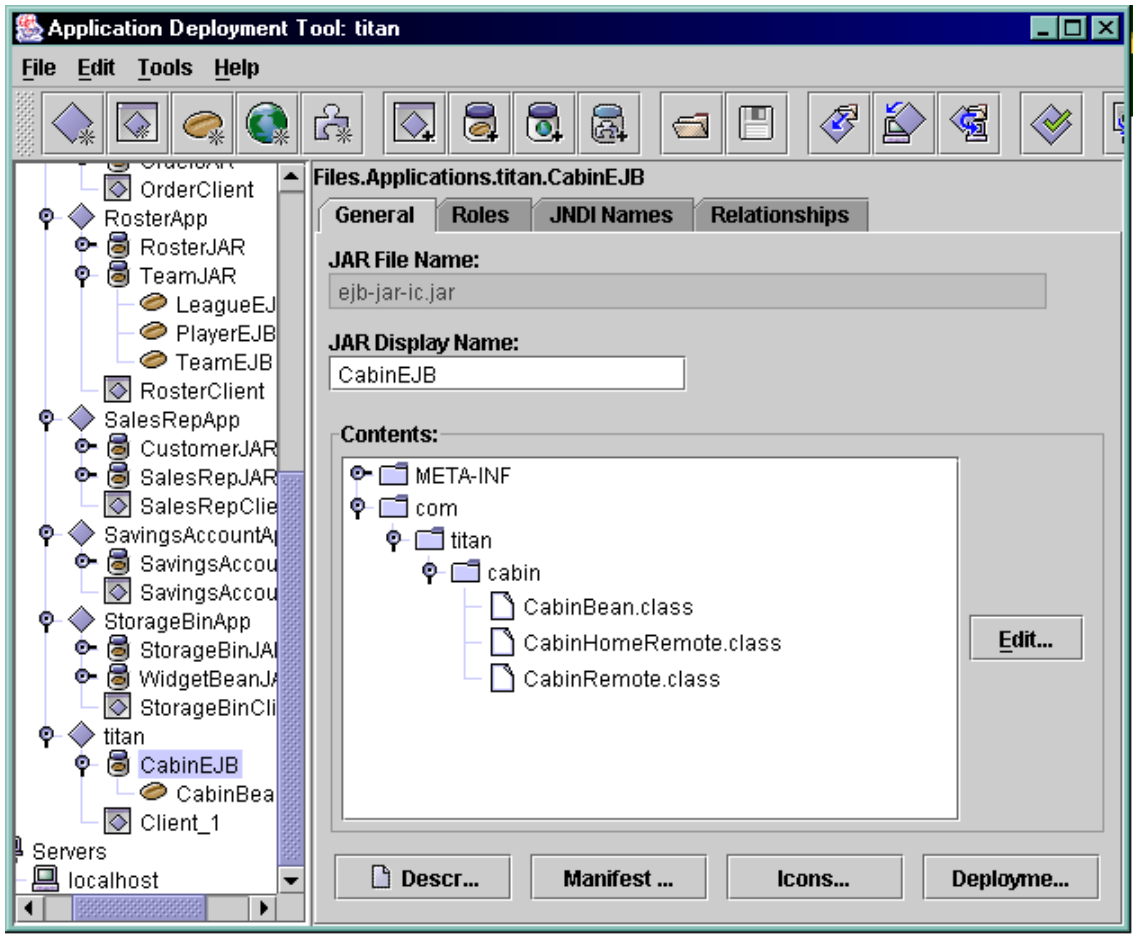
Disponíveis em j2ee.jar (implementação de referência)

- *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

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 portável) de criar componentes JAR, WAR, etc. é criar alvos para o Ant em arquivos build.xml:*
  - *Permitem a criação de "scripts" para montagem de componentes*

```
<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>
```

- *Informações sobre como usar essas tarefas no manual do Ant em $ANT_HOME/docs/manual/index.html*
  - *O Ant também será abordado no módulo 2*

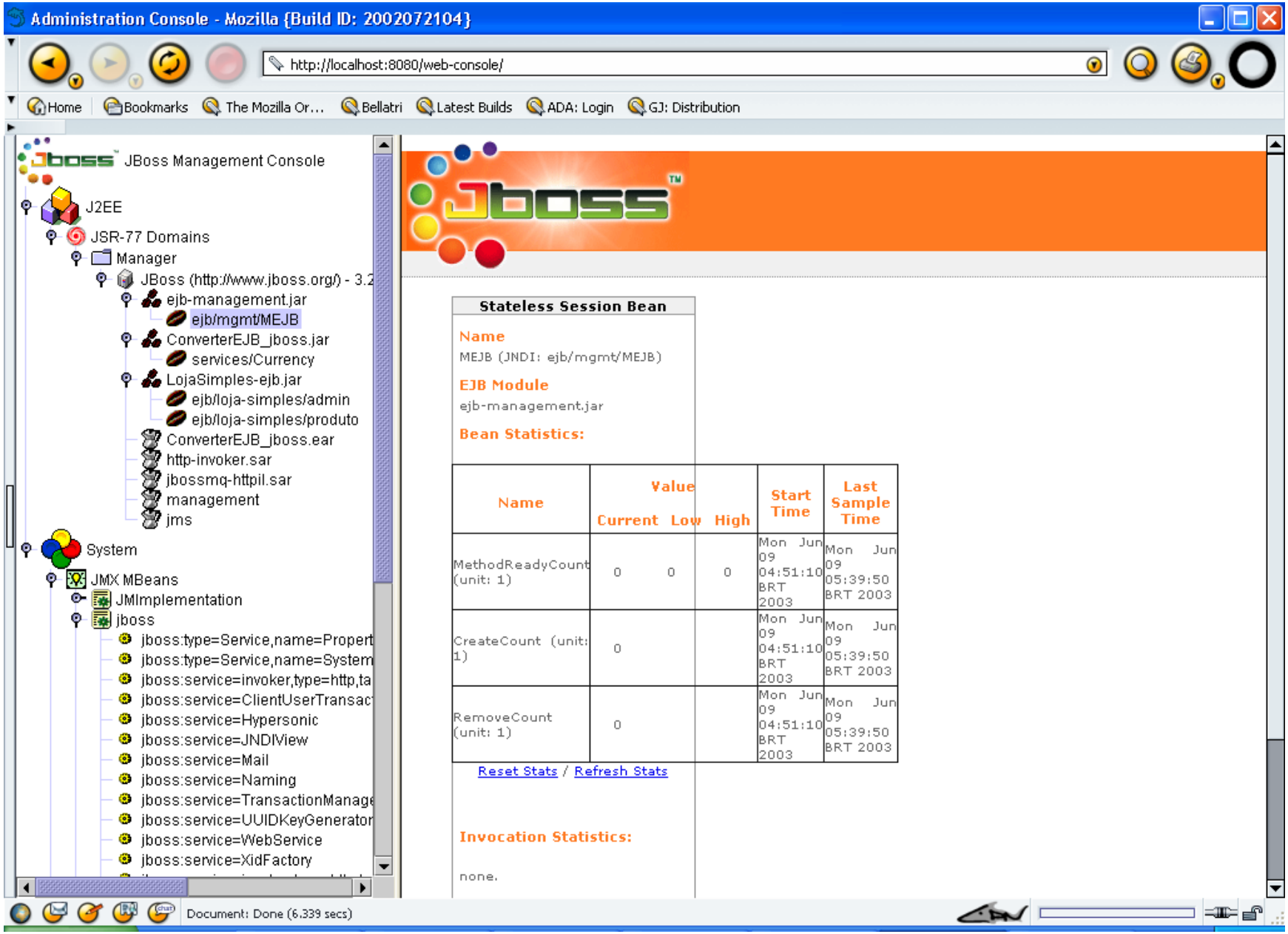
JBoss 3.x

- Servidor J2EE Open Source
  - Líder de mercado
  - Não é certificado oficialmente pela Sun
  - Vencedor do prêmio JavaWorld (concorrendo com BEA, IBM, Sun e outros)
  - Além de J2EE, suporta clustering e outros recursos extras
- Onde conseguir
  - **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

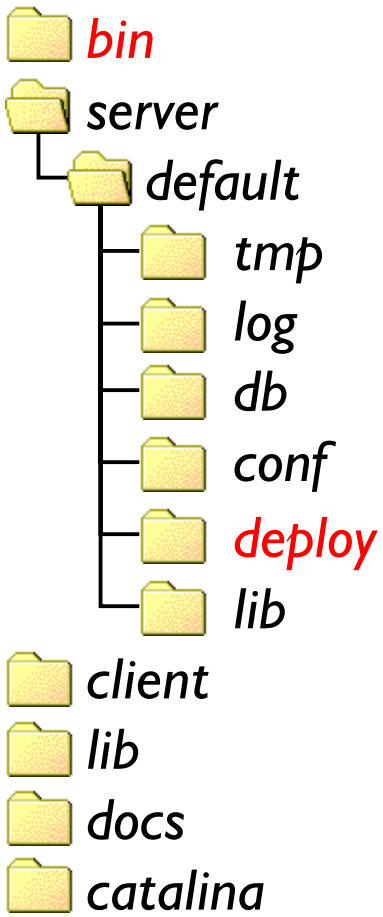
JBoss 3.x (2)

- Documentação oficial
  - A documentação do JBoss é usada para financiar o projeto Open Source. São vários livros de US$10.00 cada um. Para a maior parte dos ambientes, o primeiro livro é suficiente (os outros lidam com clustering, implementação de persistência e assuntos secundários)
- Documentação básica ($ 0.00)
  - Veja no CD: JBoss3.0GettingStarted.pdf
- Administração e configuração
  - Configuração pode ser feita nos arquivos .xml no diretório conf ou via Web (contextos jmx-console e web-console)
  - A interface de administração do JBoss é baseada em JMX (Java Management Extensions): MBeans

JBoss JMX Web Console



Estrutura de diretórios JBoss 3.x



bin	Executáveis. Para iniciar o JBoss, rode 'run' neste diretório
server	Contém pastas de servidores
default	Servidor default (se você usa clustering tem outros)
tmp	Cache do servidor. Contém todas as aplicações instaladas
log	Contém os logs de acesso ao container EJB
db	Repositório para informações persistentes
conf	Arquivos de configuração do servidor (XML/MBean)
deploy	Diretório de "hot deployment": Joque seu JAR ou EAR aqui!
lib	JARs usados nas aplicações J2EE que rodam no JBoss
client	JARs usados em aplicações-cliente
lib	JARs usados pelo servidor JBoss
docs	Contém os DTDs e exemplos de conectores para serviços
catalina	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

Exemplos

- 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

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 Java*
  `web/` *Contém as fontes Web (HTML, JSP, GIFs, JPG*
  `etc/` *ou* `dd/` *Contém arquivos de configuração (XML)*
  `lib/` *Contém bibliotecas requeridas e config dos clientes*
  `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/` *ou* `classes/` *contém código Java compilado*
  `release/` *contém componentes empacotados em JARs, WARs e EARs*

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/exemplos/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

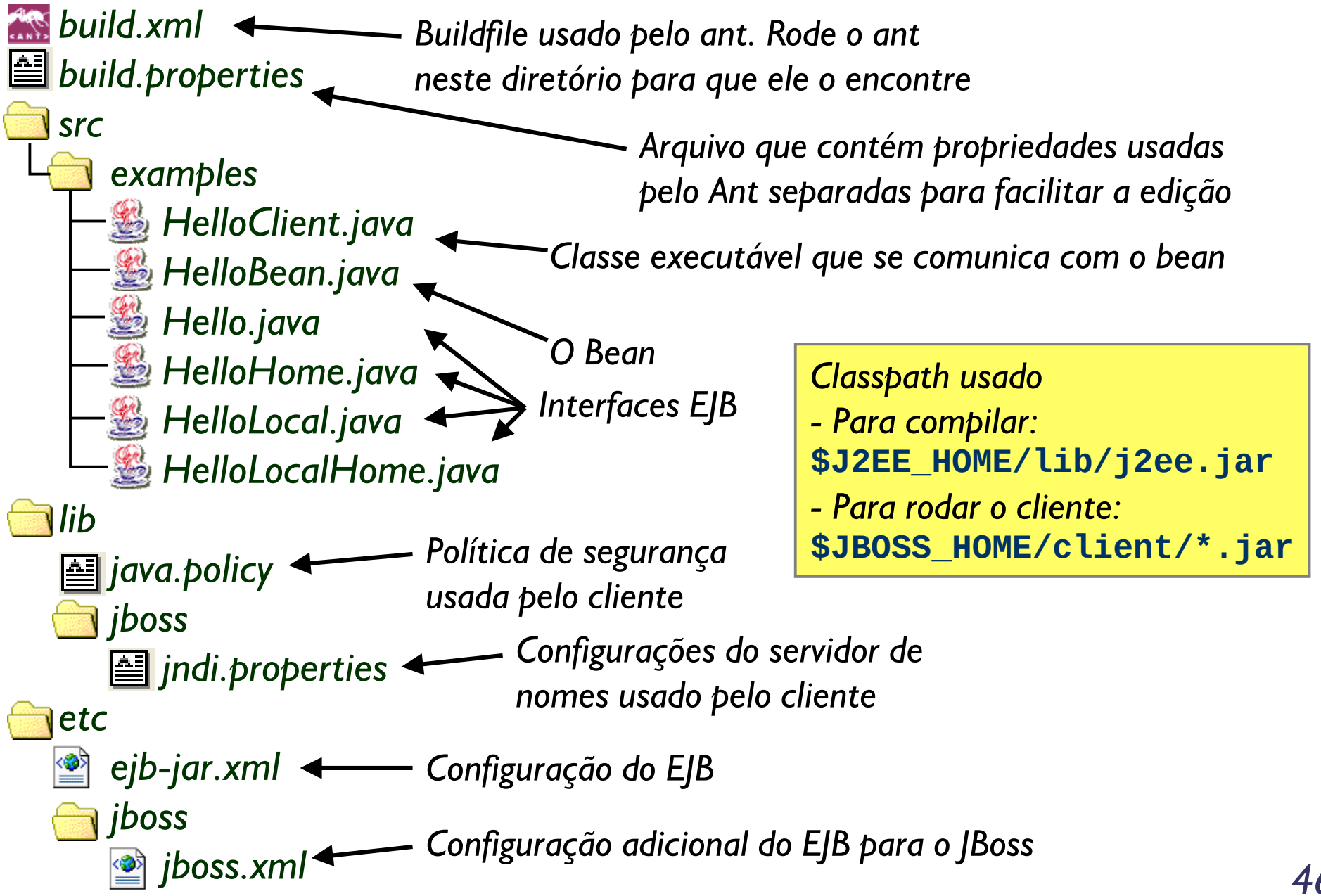
```
> ant jbossdeploy
```

- *Para rodar a aplicação cliente (e ver se o EJB funciona) use, mais uma vez, o ant:*

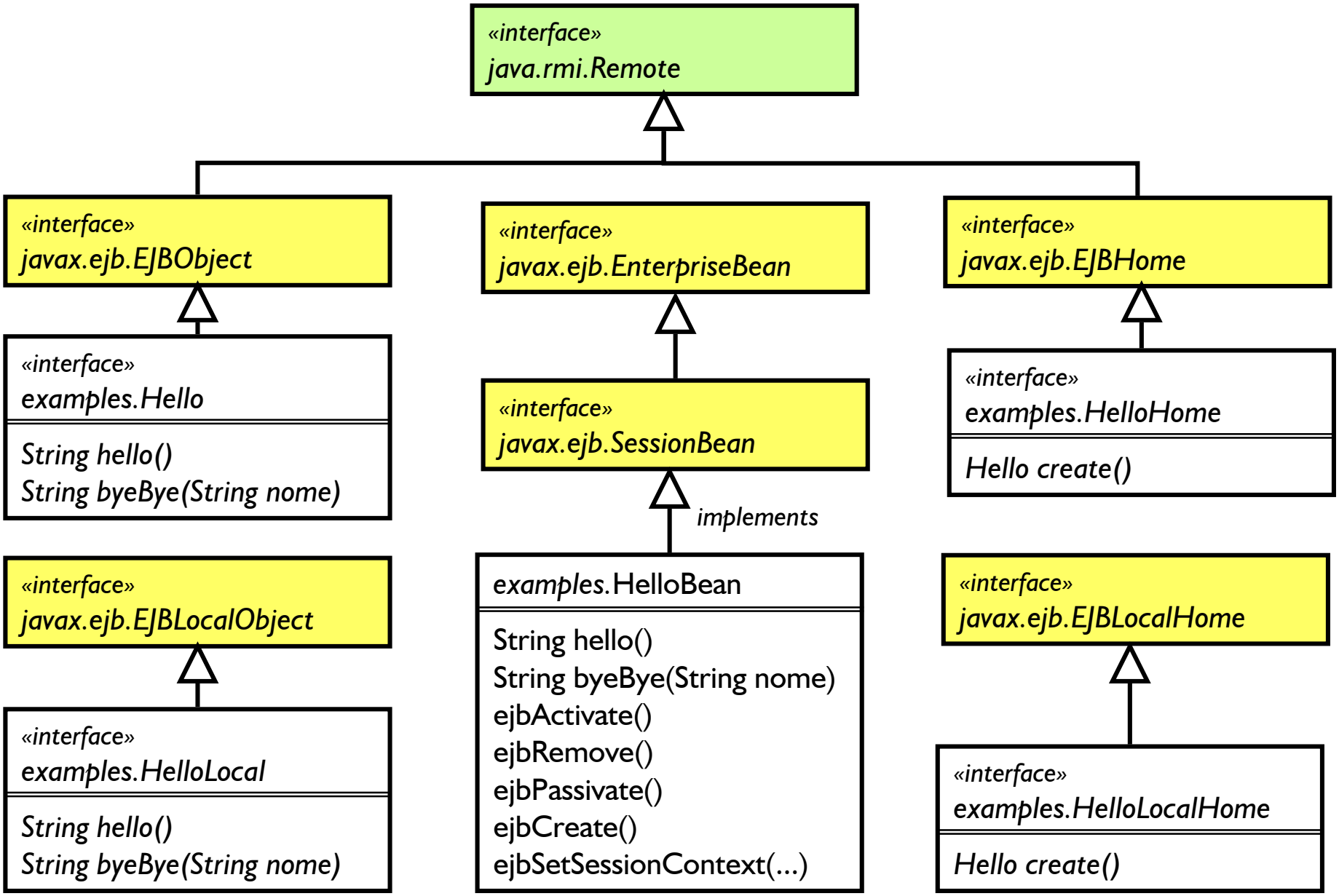
```
> ant runjbossclient
```

** 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

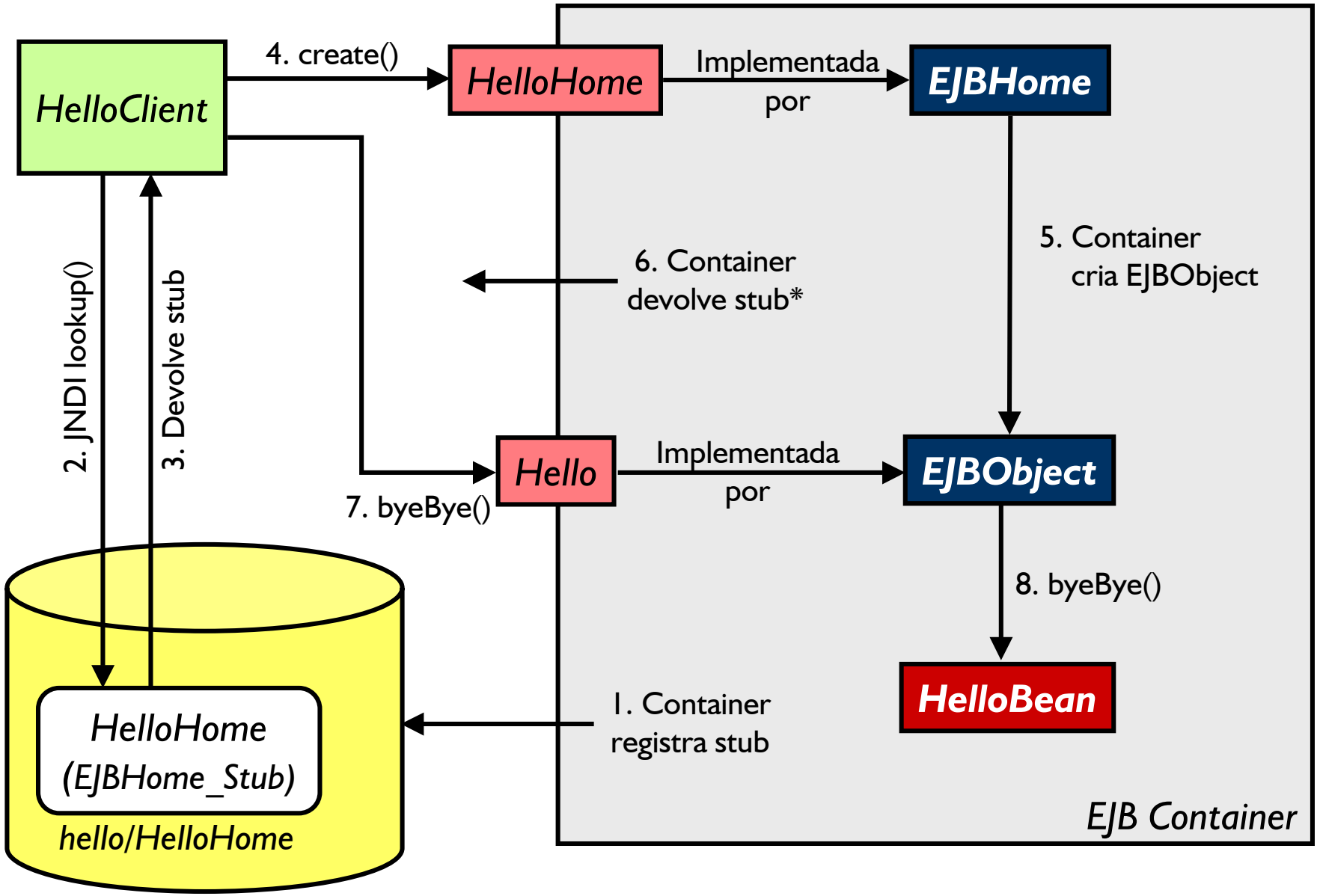
```
<jboss>
  <enterprise-beans>
    <session>
      <ejb-name>Hello</ejb-name>
      <jndi-name>hello/HelloHome</jndi-name>
    </session>
  </enterprise-beans>
</jboss>
```

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 e outros serviços 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)

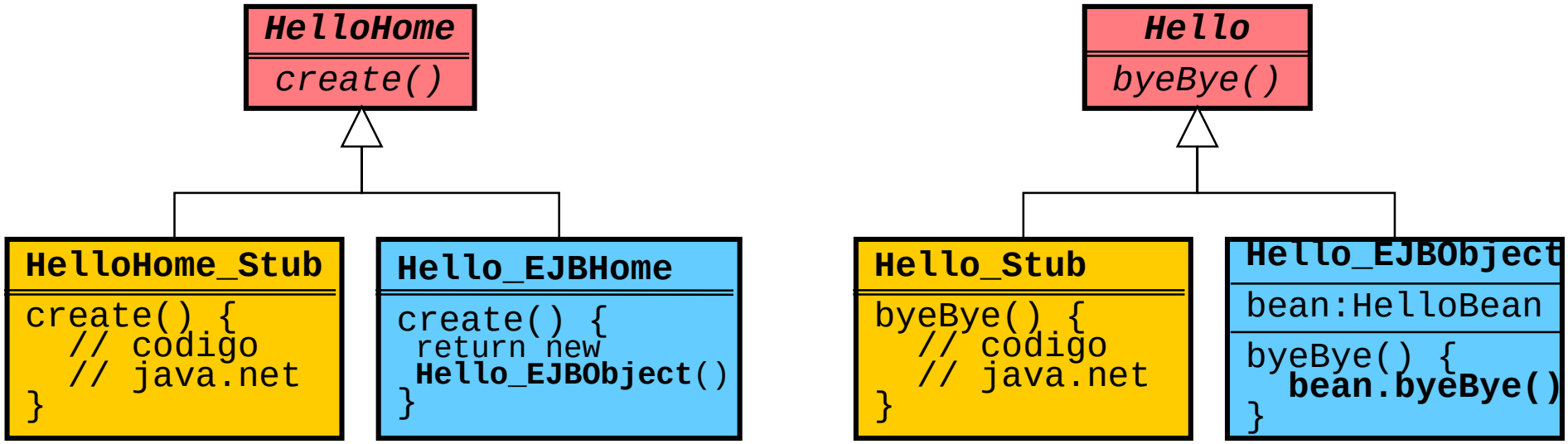
Funcionamento



* *detalhes como comunicacão entre stubs e skeletons foram omitidos*

Detalhes

- *Durante o deployment*
  - *Objetos EJBHome e EJBObject são gerados*
  - *Stubs são gerados. Stub da interface Home é mapeado a nome no servidor JNDI*
- *Hierarquias dos objetos gerados pelo container**



- *EJBObject delega requisições para o Bean*

** Apenas para efeito de ilustração. Implementação exata depende, na verdade, do fabricante*

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

```
<!DOCTYPE ...> <web-app></web-app>
```
web.xml

- 3. Na pasta web 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 web em um JAR (com extensão WAR)

```
> jar cf hello.war -C web .
```

- 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 J2EE 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/exemplos/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

Observações sobre Deployment

- Quando implantar uma aplicação no JBoss, fique atento ao que é mostrado em sua janela
  - Veja se o JBoss "reage". Se qualquer arquivo for copiado ao seu diretório deploy, ele deve imprimir alguma coisa. Se nada acontecer, veja se o arquivo realmente foi copiado ou se o servidor/sistema não está travado.
  - Veja se o JBoss informa que o deployment ocorreu. Se houver qualquer erro na implantação, o serviço não será instalado. Mensagens de erro são mostradas na tela e impressas nos logs do servidor

Conclusão

- *Apesar de serem simples os exemplos vistos neste módulo, eles ilustram todo o processo de desenvolvimento J2EE/EJB*
  1. *Codificação das interfaces, enterprise bean e componentes Web*
  2. *Empacotamento em EJB-JARs, WARs e EARs*
  3. *Configuração dos componentes através de deployment descriptors*
  4. *Implantação (deployment) em um servidor de aplicações*
- *Já temos, portanto, bons fundamentos teóricos e alguma experiência prática para começar a desenvolver e montar aplicações EJB*

Exercícios

1. *Implante uma aplicação Web no JBoss (copie o WAR para o diretório deploy) e teste-a.*
2. *Implante os exemplos demonstrados em sala no JBoss instalado na sua máquina. Teste-os.*

Exercícios extras. Use os arquivos em cap01/exercicios/*

3. *a) Altere o session bean (similar ao do primeiro exemplo) e acrescente novos métodos*
   - *Um método que retorne a data e a hora*
   - *Um método que receba dois números e retorne a soma*
3. *b) Empacote o bean e faça o deployment no JBoss*
3. *c) Complete o cliente fornecido para que chame os métodos*
4. *Escreva uma aplicação Web (preencha o JSP) para que conecte ao bean e chame seus métodos*

** Opcionais. Podem ser feitos fora do horário de aula ou resolvidos em sala pelo instrutor como demonstração*

Fontes

[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.

Curso J530: Enterprise JavaBeans

Revisão 2.0 - Junho de 2003

(helder@acm.org)

argonavis.com.br