Serviços de Localização

Helder da Rocha
www.argonavis.com.br

Objetivos

- *Este módulo descreve os serviços básicos para localização de recursos em J2EE*
  - *Localização de componentes (servlets, EJBs)*
  - *Localização de fontes de dados (bancos, filas, pilhas)*
  - *Objetos compartilhados*
- *Recursos são localizados através da interface JNDI - Java Naming and Directory Interface*
  - *Este módulo descreve o uso básico de JNDI*
  - *Para maiores detalhes e sobre o uso de JNDI independente de um servidor de aplicações, consulte slides do minicurso J523, que explora outros detalhes de JNDI*
  - *Alguns exercícios foram adaptados do JNDI Tutorial da Sun*

Serviço de nomes

- *A principal função de um serviço de nomes é permitir a associação de um nome (ou uma outra representação alternativa mais simples) a recursos computacionais como*
  - *endereços de memória, de rede, de serviços*
  - *objetos e referências*
  - *códigos em geral*
- *Suas duas funções básicas são*
  - *Associar (mapear) um nome a um recurso*
  - *Localizar um recurso a partir de seu nome*
- *Exemplos*
  - *Sistema de arquivos: liga caminho a bloco(s) de memória:*

  - *Sistema DNS: liga nome de domínio a endereço IP:*

Contextos e Sistemas de Nomes

- *Um contexto é um conjunto de ligações nome-objeto*
  - *Em outras palavras, é um objeto que tem zero ou mais ligações*
- *Se o objeto (referência) contido no contexto for também um contexto ele é um subcontexto*
  - *O escopo do subcontexto é limitado pelo seu contexto pai*
- *Exemplos de contextos e subcontextos:*
  - **`/usr/bin/java/`**

    usr é o contexto; bin é subcontexto de usr, ...
  - **`www.abc.com.br`**

    br é o contexto, com é subcontexto de br, ...
- *Um sistema de nomes é um conjunto interligado de contextos que respeitam a mesma convenção e possuem um conjunto comum de operações*

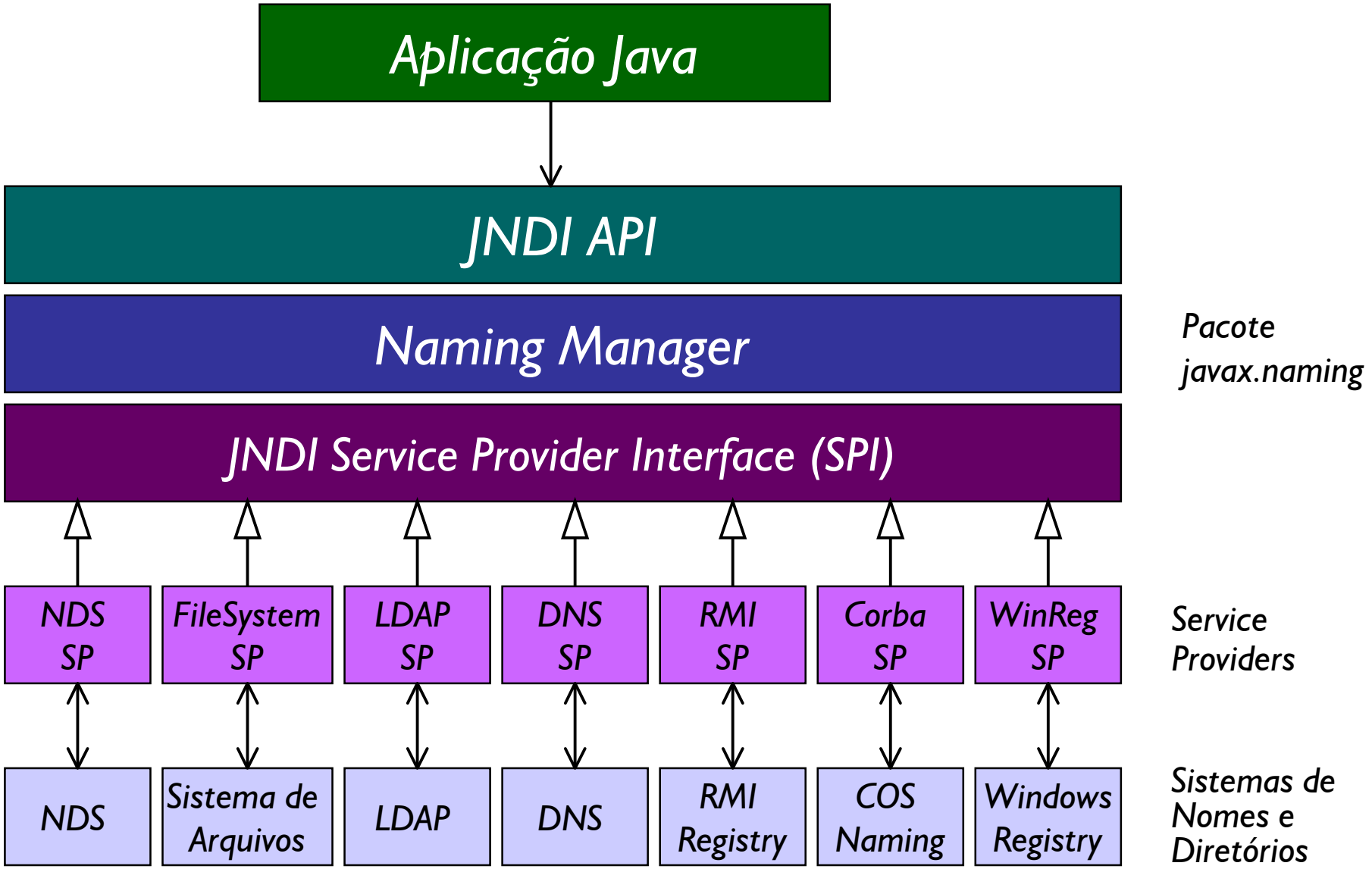
Diretórios e Serviços de Diretório

- *Diretório: conjunto interligado de objetos*
  - *Organização não precisa ser hierárquica (contextual)*
  - *Para cada item há um nome unívoco (chave)*
  - *Cada item possui um ou mais atributos*
- *Um serviço de diretório oferece operações para criar, remover, modificar e principalmente pesquisar atributos associados a objetos em um diretório*
- *Um atributo possui*
  - *Um identificador*
  - *Conjunto de valores*
  - *Um tipo: restringe os dados que um atributo pode receber*
- *Sistemas de nomes são frequentemente estendidos com serviços de diretório*

JNDI

- ***Java Naming and Directory Interface*** *é uma ponte sobre os diversos serviços de nomes e diretórios diferentes*
- *Vantagens*
  - *Só é preciso aprender uma única API para acessar vários tipos de informação de serviços de diretório*
  - *Isola a aplicação dos detalhes específicos do protocolo*
  - *Pode ser usada para ler objetos Java (serializados) que estejam armazenados em um diretório*
  - *Pode combinar diferentes tipos de diretório (federação) e tratá-los como um diretório único*
- *Componentes*
  - ***API*** *- Application Programming Interface*
  - ***SPI*** *- Service Provider Interface que permite que novos serviços sejam plugados transparentemente*

Arquitetura JNDI



Fonte: JNDI Tutorial

JNDI API

- A API JNDI está incluída no J2SDK 1.3 ou posterior nos pacotes e subpacotes descendentes de javax.naming.
- Para usar JNDI é preciso ter
  - As classes e interfaces do JNDI (pacotes javax.naming.*)
  - Pelo menos um provedor de serviços JNDI (driver)
- O Java 2 SDK inclui provedores de serviço (SPs) para
  - LDAP - Lightweight Directory Access Protocol
  - CORBA - Common ORB Architecture e COS name service
  - Java RMI Registry
- Outros provedores de serviços (para sistema de arquivos, para DNS, para JDBC, para Windows Registry, etc.) podem ser encontrados a partir do site

http://java.sun.com/products/jndi/serviceproviders.html

Principais classes

- A API JNDI consiste de cinco pacotes
- O principal pacote é **javax.naming** que contém as principais classes e interfaces
  - Context: interface onde se pode recuperar, ligar, desligar e renomear objetos, e criar e destruir contextos
  - InitialContext: ponto de partida (raiz) para todas as operações
  - Name: abstração de um nome. Contém geralmente um String de texto que corresponde ao nome do objeto ou contexto
  - NameClassPair: contém nome do objeto e de sua classe
  - Binding: contém nome do objeto ligado, nome da classe do objeto e o próprio objeto
  - Reference: abstração de uma referência para um objeto
  - NamingEnumeration: um tipo de java.util.Enumeration usado para colecionar componentes de um contexto
  - NamingException: principal exceção do JNDI

Contexto inicial

- *Precisa ser obtido antes de qualquer operação. Passos:*
- *1: selecionar o provedor de serviços*

```
Properties env = new Properties();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "classe.do.ProvedorDeServicos");
```

- *2: configurar o acesso ao serviço*

```
env.put(Context.PROVIDER_URL, "ldap://xyz.com:389");
env.put(Context.OUTRA_PROPRIEDADE, "valor"); (...)
```

- *3: criar um objeto para representar o contexto*

```
Context ctx = new InitialContext(env);
```

- *A configuração (1, 2) pode ser feita via propriedades do sistema*
  - *Passadas em linha de comando via argumento* `-Dprop=valor`
  - *Carregados via arquivos de propriedades*
- *Principais propriedades*

```
java.naming.factory.initial:  Context.INITIAL_CONTEXT_FACTORY
java.naming.provider.url:     Context.PROVIDER_URL
```

Arquivo jndi.properties

- Uma outra forma de definir propriedades usadas pelos clientes JNDI, é através de um arquivo *jndi.properties*
  - Um arquivo com este nome deve ser colocado no CLASSPATH da aplicação cliente
  - Ao inicializar o ambiente com *InitialContext()*, não passe nenhum parâmetro no construtor. O class loader irá procurar um arquivo jndi.properties no CLASSPATH e carregará as propriedades que estiverem definidas dentro dele
- Por exemplo, um arquivo pode conter

```
java.naming.factory.initial=\
    com.sun.jndi.fscontext.RefFSContextFactory
java.naming.provider.url=file:/cap02/lab/filesys
```

- Para inicializar o sistema com essas propriedades, use

```
Context ctx = new InitialContext();
```

Recuperação de objetos (lookup)

- Para obter a referência para um objeto de um contexto usa-se o método **lookup()**
  - Para usar o objeto retornado é preciso conhecer o seu tipo e fazer o cast (ou narrow, se objeto remoto) para promover a referência
  - Se o objeto for um contexto, lookup() age como um método para mudar de contexto (como o chdir, em Unix)
- Exemplo
  - O método **lookup()** usando com o provedor de serviço fscontext retorna um java.io.File pelo nome de arquivo

```
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.sun.jndi.fscontext.RefFSContextFactory");

env.put(Context.PROVIDER_URL,
        "file:/cap02/lab/filesys");

Context ctx = new InitialContext(env);

File f = (File)ctx.lookup("report.txt");
```

Serviço de nomes para sistema de arquivos

Diretório raiz do serviço

Arquivo localizado na raiz do serviço

Listagem do conteúdo de contextos

- *Em vez de obter um objeto de cada vez via lookup(), pode-se obter uma lista deles*
  - *Método **list()** retorna uma lista (NamingEnumeration) de pares nome / nome_da_classe (objetos do tipo NameClassPair)*

```
NamingEnumeration lista = ctx.list("awt");
while (lista.hasMore()) {
   NameClassPair nc = (NameClassPair)lista.next();
   System.out.println(nc);
}
```

Trecho de List.java

  - *Método **listBindings()** retorna uma lista de ligações nome / objeto (Binding)*

```
NamingEnumeration lista = ctx.listBindings("awt");
while (lista.hasMore()) {
   Binding bd = (Binding)lista.next();
   System.out.println(bd.getName() + ": " + bd.getObject());
}
```

Trecho de ListBindings.java

Modificação de ligações, objetos e contextos

- *Adicionando ligações*

```
Fruit fruit = new Fruit("orange");
ctx.bind("favorite", fruit);
```
Bind.java

- *Substituindo ligações*

```
Fruit fruit = new Fruit("lemon");
ctx.rebind("favorite", fruit);
```
Rebind.java

- *Removendo ligações*

```
ctx.unbind("favorite");
```
Unbind.java

- *Renomeando objetos*

```
ctx.rename("report.txt", "old_report.txt");
```
Rename.java

- *Criando novos contextos*

```
Context result = ctx.createSubcontext("new");
```
Create.java

- *Destruindo contextos*

```
ctx.destroySubcontext("new");
```
Destroy.java

Rode List ou ListBindings após cada operação para ver os resultados

Provedores de serviços para objetos

- Pode-se usar JNDI para mapear nomes a objetos
  - Objetos localizáveis por nome podem ser abstraídos do contexto ou até linguagem em que são usados
  - Aplicações diferentes podem compartilhar objetos
- Dois drivers JNDI estão disponíveis para acesso a objetos distribuídos no J2SDK
  - SPI CORBA (COS Naming): permite localização de objetos CORBA (serializados em um formato independende de linguagem) - usado em RMI-IIOP (EJB)

    `com.sun.jndi.cosnaming.CNCtxFactory`
  - SPI RMI: permite a localização de objetos Java serializados (objetos pode ser usados por outras aplicações Java)

    `com.sun.jndi.rmi.registry.RegistryContextFactory`

Objetos RMI

- *Proprieades a definir*

java.naming.factory.initial ou **Context.INITIAL_CONTEXT_FACTORY**

com.sun.jndi.rmi.registry.RegistryContextFactory

java.naming.provider.url ou **Context.PROVIDER_URL**

rmi://nome_do_host:1099 (endereço e porta do RMI Registry)

- *Mapeamento*

```
Context ctx = new InitialContext();
Fruit fruit = new Fruit("orange");
ctx.bind("favorite", fruit);
```

- *Localização*

```
Context ctx = new InitialContext();
Fruit fruit = (Fruit) ctx.lookup("favorite");
```

Objetos CORBA (RMI sobre IIOP)

- *Proprieades a definir*

java.naming.factory.initial ou **Context.INITIAL_CONTEXT_FACTORY**

com.sun.jndi.cosnaming.CNCtxFactory

java.naming.provider.url ou **Context.PROVIDER_URL**

iiop://nome_do_host:1900 (endereço e porta do ORB)

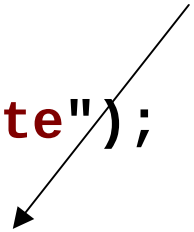
- *Mapeamento*

```
Context ctx = new InitialContext();
Fruit fruit = new Fruit("orange");
ctx.bind("favorite", fruit);
```

- *Localização*

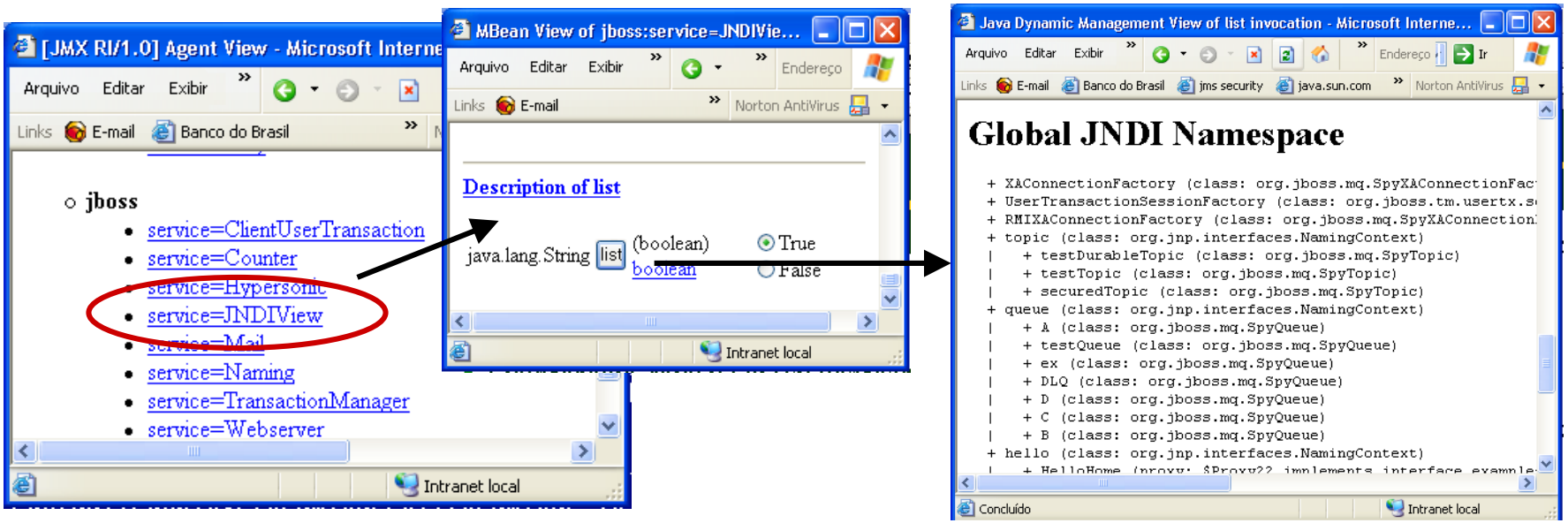
```
Context ctx = new InitialContext();
Object corbaFruit = ctx.lookup("favorite");
Object javaFruit =
  javax.rmi.PortableRemoteObject.narrow(corbaFruit,
                                        Fruit.class);

Fruit fruit = (Fruit)javaFruit;
```

Converte objeto CORBA em objeto Java



Serviço JNDI no JBoss

- Pode-se usar *bind()* e *lookup()* para armazenar e localizar objetos no servidor JNDI do JBoss
  - Componentes Web e EJB são mapeados (bound) automaticamente, durante a implantação (o servidor usa informações no *web.xml*, *ejb-jar.xml* e outros arquivos de configuração existentes)
- Para verificar os mapeamentos JNDI existentes, acesse o serviço *JNDIView* em *http://localhost:8080/jmx-console*



Fontes de dados

- *No JBoss, pools de conexões de bancos de dados são acessíveis no servidor através de objetos DataSource, publicadas no JNDI abaixo no namespace java:/*
  - **java:/CloudscapeDB**
  - **java:/OracleDB**
- *Os nomes são configurados nos arquivos *-ds.xml do JBoss (localizados em deploy)*
- *Para acessar um banco existente no servidor use JNDI e nome definido no *-service.xml correspondente*

```
Context ctx = new InitialContext();
javax.sql.DataSource ds = (javax.sql.DataSource)
    ctx.lookup("java:/DefaultDS");
java.sql.Connection = ds.getConnection();
```

Banco HSQLDB

- O JBoss traz embutido o banco de dados HSQLDB (ex-HyperSonic)
- Para acessar o banco através de JNDI, use
  - java:/DefaultDS
- DefaultDS é um nome genérico para designar a fonte de dados default do servidor
  - Pode ser configurada para apontar para outro banco
  - Pode haver outras fontes de dados criadas para o mesmo banco ou bancos diferentes no servidor
- Para consultar ou alterar usuario, senha, driver ou outras configurações de acesso ao HSQLDB, utilize o arquivo hsqldb-ds.xml, localizado em deploy.

Configuração Default do HSQLDB

- *JNDI Name:*
  `DefaultDS`
- *JDBC URL:*
  `jdbc:hsqldb:hsql://localhost:1701`
- *Driver:*
  `$JBOSS_HOME/server/default/lib/hsqldb.jar`
- *Driver class:*
  `org.hsqldb.jdbcDriver`
- *Arquivo de configuração JCA:*
  `$JBOSS_HOME/server/default/deploy/hsqldb-ds.xml`
- *Nível de isolamento suportado para transações:*
  `TRANSACTION_READ_UNCOMMITTED`
- *User-name:*
  `sa`
- *Password:*
  (nada)
- *Min-pool-size:*
  `5`
- *Security domain:*
  `HsqlDbRealm`

Enterprise Naming Context

- *Cada EJB e cada aplicação Web tem um namespace próprio para compartilhar objetos usando JNDI*

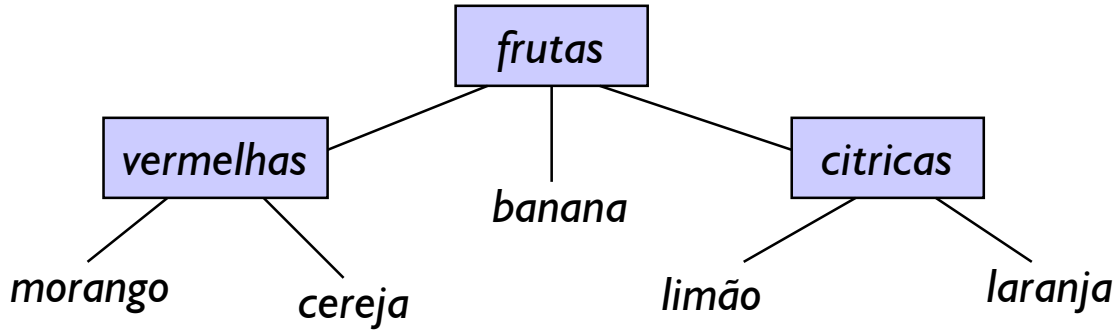
  `java:comp/env`

- *É chamado de Enterprise Naming Context (ENC)*
  - *É mapeado ao JNDI global durante a implantação: permite que componentes de uma mesma aplicação firmem ligações lógicas entre si sem depender de recursos externos*
  - *Nomes do ENC são mapeados a nomes JNDI reais pelo deployer*
  - *Acesso java:comp/env não é compartilhado: cada bean ou contexto Web tem o seu: é preciso declarar nomes usados em cada bean!*
- *Forma recomendada de comunicação em J2EE*
  - *Componentes Web e EJBs devem obter referências uns para os outros através de java:comp/env*

```
Objeto obj =
   (Objeto)ctx.lookup("java:comp/env/meuObjeto");
```

Exercícios

- *1. Publicação de objetos referenciáveis em JNDI*
  - *a) Crie a seguinte hierarquia de contextos e objetos (Fruit) usando as classes fornecidas e o driver JNDI do JBoss (para criar um objeto, use new normalmente)*



  - *b) Localize a estrutura criada através do serviço JNDIView*
- *2. Enterprise Naming Context (java:comp/env)*
  - *a) Analise o código do exemplo mostrado no primeiro capítulo e localize as chamadas java:comp/env em index.jsp, referências a EJBs no web.xml e mapeamentos com nomes globais em jboss.xml*
  - *b) Ache os nomes e mapeamentos no JNDIView*

Fontes para este capítulo

[1] *Rossana Lee. The JNDI Tutorial, Sun Microsystems, 2002*
http://java.sun.com/products/jndi/tutorial/
Parte deste capítulo é baseada nas primeiras duas seções (trilhas) do JNDI Tutorial.

[2] *Ed Roman et al. Mastering EJB 2.0, Wiley, 2001*
http://www.theserverside.com/books/masteringEJB/index.jsp
Apêndice A tem um breve e objetivo tutorial sobre JNDI

helder@argonavis.com.br

argonavis.com.br

J500 - Aplicações Distribuídas com J2EE e JBoss
Revisão 1.5 (junho de 2003)
J530 - Enterprise JavaBeans
Revisão 2.0 (junho de 2003)

Introdução a J2EE, 2000, 2001, 2002, 2003
Atualizado em Junho de 2003