J2EE

Tecnologias de Middleware

2006/2007

Fernando Martins - fmp.martins@gmail.com

Tópicos

- J2EE
- Objectivo
- Paradigma e Conceitos
- Servidor J2EE
- Tecnologias
- Container
- Enterprise Beans
  - Session Bean
  - Entity Bean
  - Message-Driven Bean
- Connector
- Empacotamento e Instalação
- Arquitectura
- Exemplo
- Conclusão

J2EE

- J2EE:
  - Java 2 Platform, Enterprise Edition: J2EE (até à versão 1.4)
  - Java Platform, Enterprise Edition 5: J EE (a partir da versão 1.5/5)
- Sun Microsystems

Objectivo

- Modelo multi-camada de aplicações distribuídas.
- Reutilização de componentes.
- Modelo de segurança unificado.
- Controlo transaccional flexível.
- Suporte de *web services* através de troca de informação baseado em standards e protocolos abertos.
- Infraestrutura virada para o desenvolvimento web.
- Não dependente da plataforma.

Paradigma e Conceitos

- Expansão da plataforma Java para aplicações distribuídas.
- Baseado em componentes, *containers* e parametrizações.
- Convivência de várias tecnologias Java.
- Canal unificador para aplicações cliente distintas.
- Centrado na implementação da solução e não na gestão dos componentes.
- Permite o desenvolvimento de aplicações sem impor ou restringir aproximações e modelos.

Paradigma e Conceitos

- Multi-camada:
  - Cliente: aplicações que acedem ao servidor J2EE.
  - Negócio: componentes que disponibilizam toda lógica de negócio de uma aplicação.
  - Dados: qualquer fonte de dados do sistema de informação empresarial, bases de dados, sistemas de planeamento, sistemas *legacy*, i.e., qualquer recurso que faça parte da empresa.
  - Serviços: APIs e tecnologias

Servidor J2EE

- Um servidor J2EE acolhe vários tipos de componentes de aplicações que correspondem às camadas de uma aplicação multi-camada.
- O servidor J2EE disponibiliza serviços a esses componentes na forma de *containers.*
- A separação entre os componentes e os *containers* permite flexibilidade e simplicidade, pois a gestão do componente é feita por parametrização e não por programação.

Tecnologias

- **Enterprise JavaBeans (EJB):** define como os componentes são escritos e disponibilizados do lado do servidor.
- **Java Messaging System (JMS):** suporte para comunicação através de mensagens.
- **Java Naming and Directory Service (JNDI):** para aceder a sistemas de nomes e directórios.
- **Java IDL:** implementação de CORBA em Java.

Tecnologias

- **Java Remote Method Invocation (RMI):** forma nativa do Java comunicar entre objectos distribuidos. O RMI-IIOP é uma extensão do RMI para integração com CORBA.
- **Java API for XML RPC (JAX-RPC):** suporte para desenvolvimento de *web services*.
- **Java DataBase Connectivity (JDBC):** API de acesso a bases de dados relacionais.

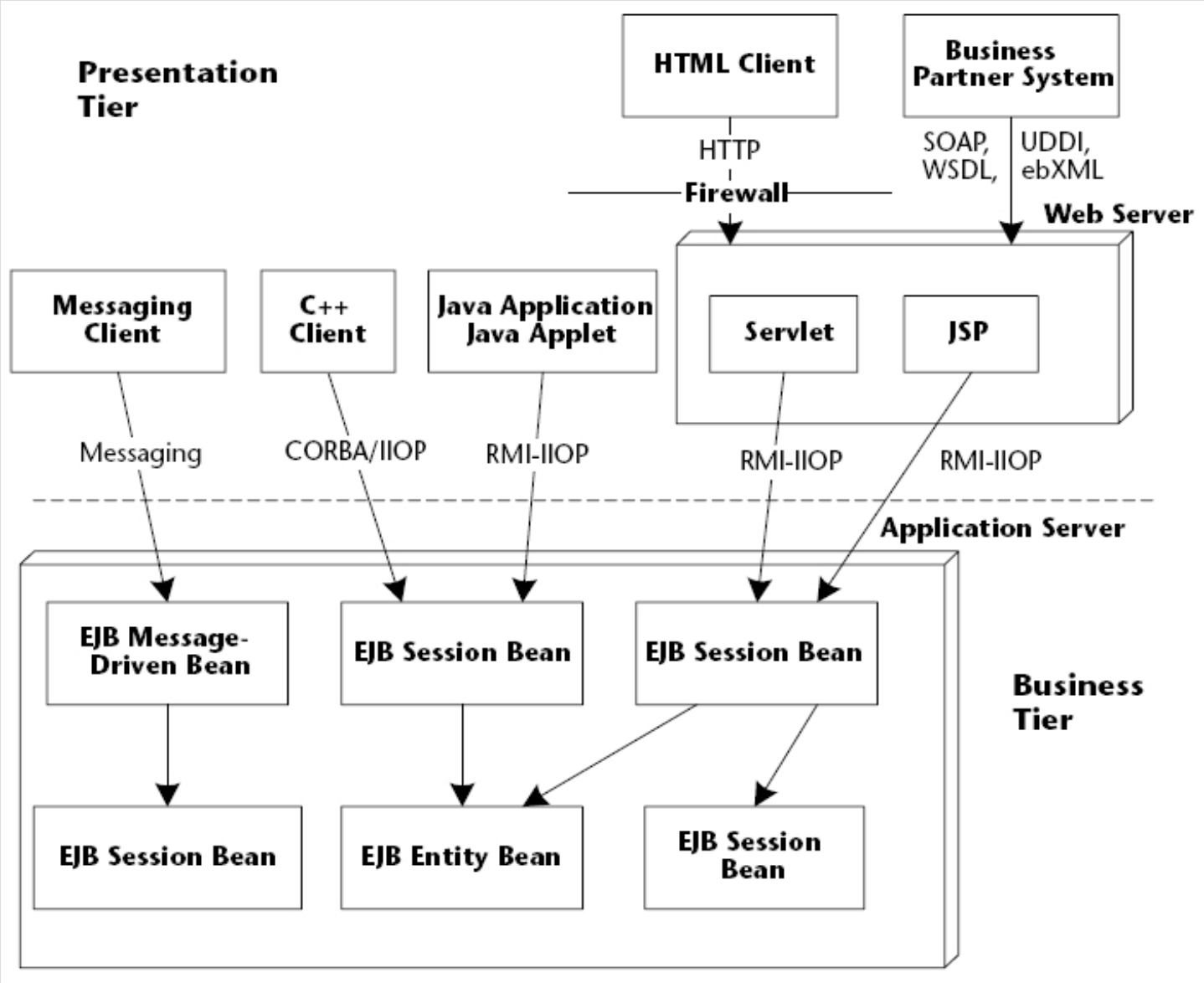
Tecnologias

- **Java Transaction API (JTA) e Java Transaction Service (JTS):** especificações que permitem transacções fiáveis.
- **Java Servlets:** componentes orientados a pedido/resposta para estender a funcionalidade de um servidor web.
- **JavaServer Pages (JSP):** scripts orientados à apresentação que são compilados em *servlets*.

Tecnologias

- **Java Mail:** serviço de envio de email.
- **J2EE Connector Arquitecture (JCA):** suporte para comunicação com qualquer outro sistema.
- **Java API for XML Parsing (JAXP):** API *de facto* para manipular documentos XML.
- **Java Authentication and Authorization Service (JAAS):** API de operações de segurança.

Tecnologias



Container

- *Container* é a interface entre um componente e a funcionalidade de baixo nível da plataforma que suporta o componente.

- Antes de serem usados, os componentes têm de ser empacotados num módulo J2EE e instalados nos seus *containers*.

Container

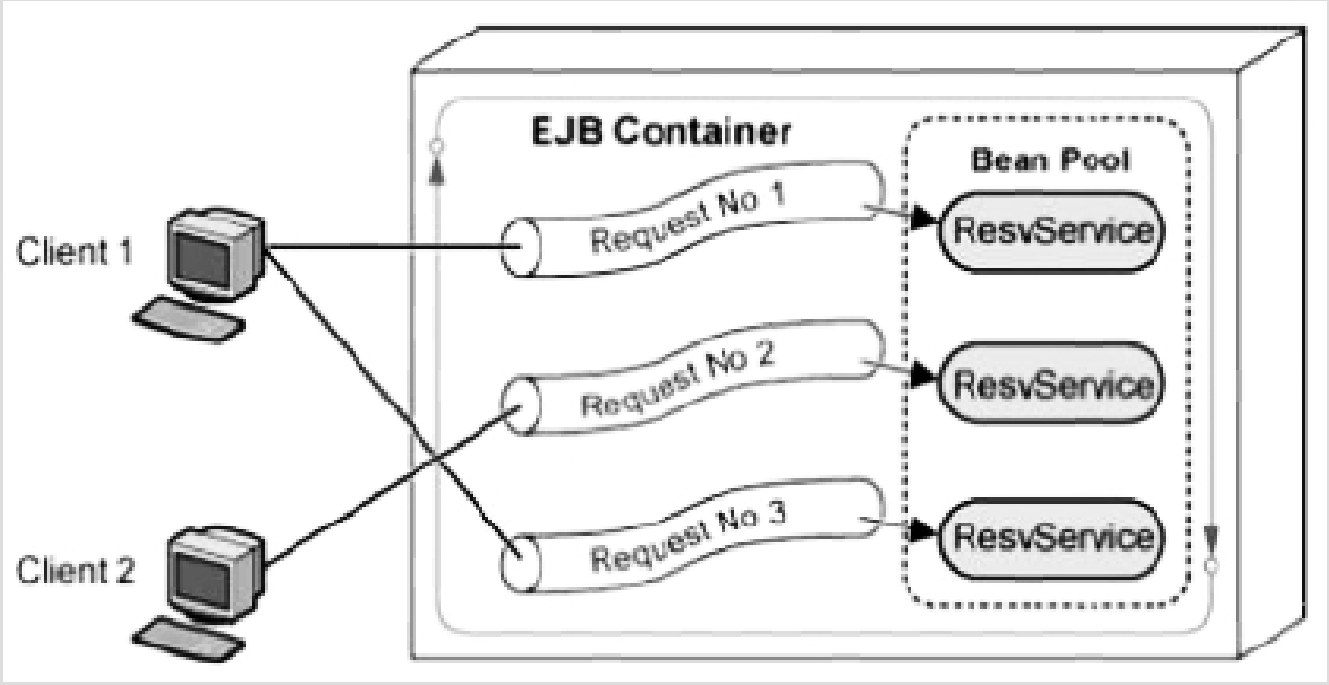
- ***Container* Web:** interface entre componentes web e servidor web, gere o ciclo de vida dos componentes, remete pedidos para os componentes da aplicação e disponibiliza interfaces para dados em contexto.
- ***Container* de Componentes da Aplicação:** interface entre as aplicações cliente J2EE e o servidor J2EE.
- ***Container* EJB:** interface entre os *enterprise beans* e o servidor J2EE.

Enterprise Beans

- Tecnologia *Enterprise JavaBean* (EJB) basea-se em RMI-IIOP e JNDI, é o cerne do J2EE.
- *Enterprise Beans* (EB) são componentes do lado do servidor que pretendem solucionar problemas comuns em sistemas distribuídos:
  - Persistência
  - Integridade Transaccional
  - Segurança
- Um *container* EJB suporta:
  - Sessions Beans
  - Entity Beans
  - Message-Driven Beans

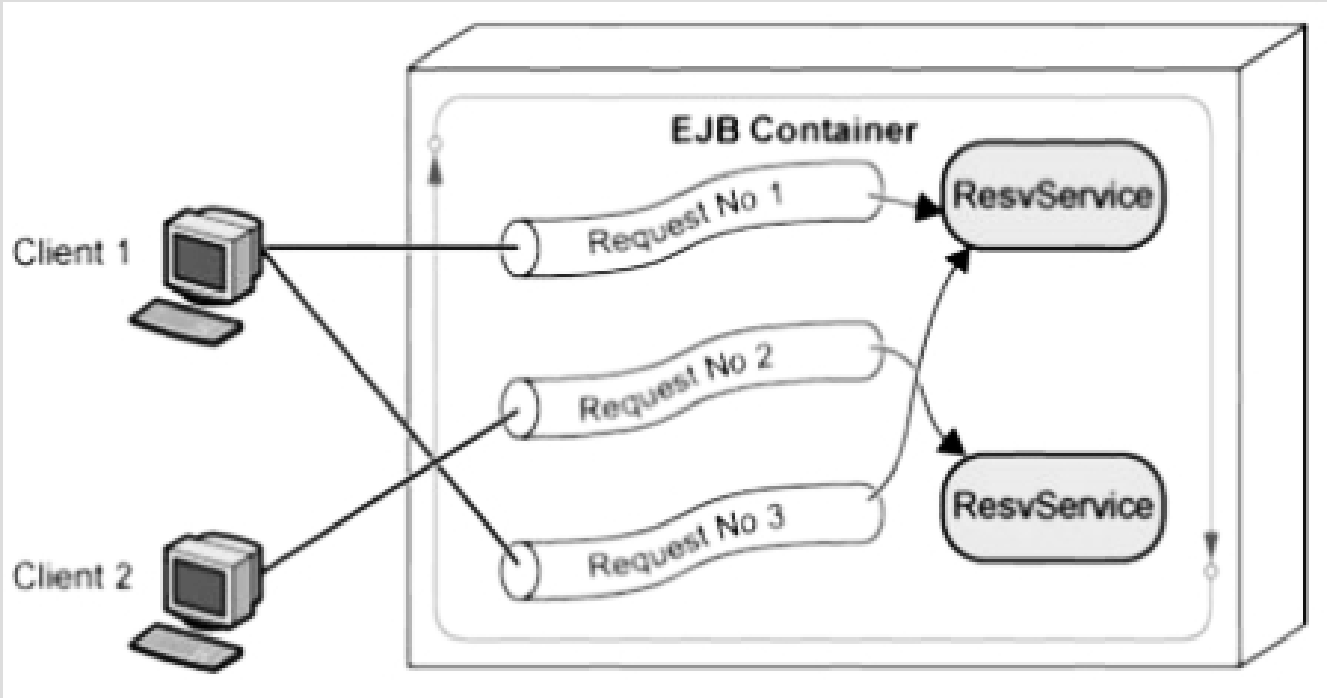
EB - Session Beans

- **Sem Estado:** objectos distribuídos sem qualquer estado associado, permitindo acesso concorrente. Não garante a preservação do conteúdo dos valores das instâncias entre chamadas.



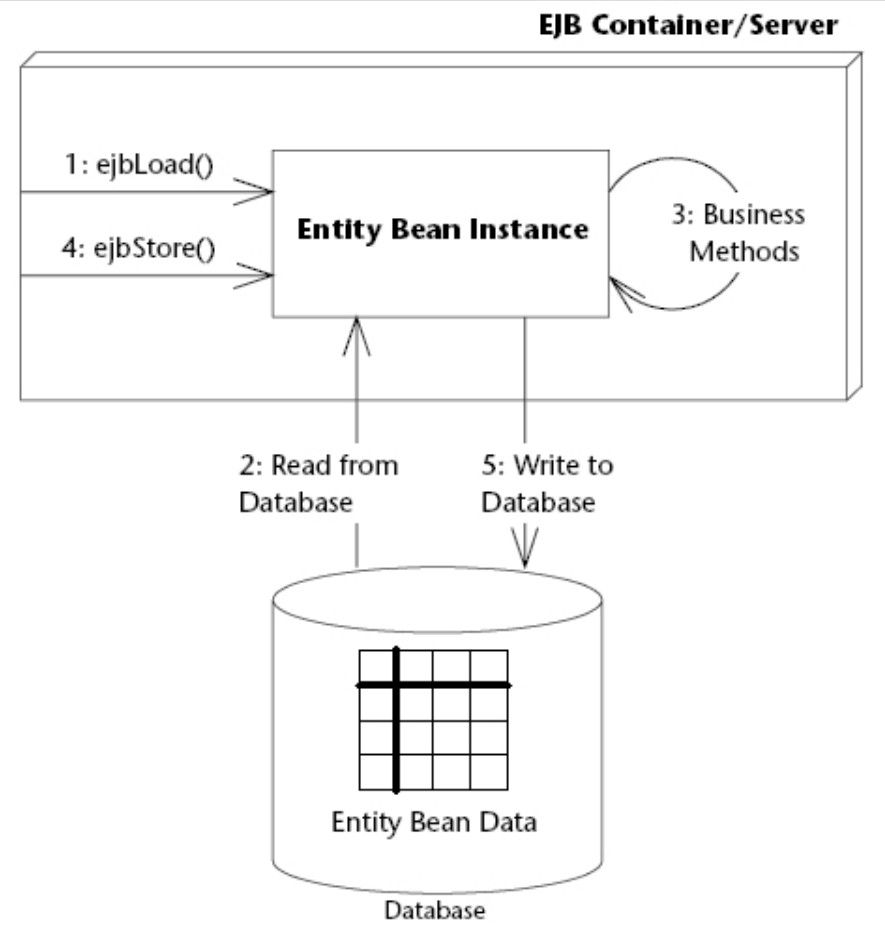
EB - Session Beans

- **Com Estado:** objectos distribuídos com estado preservado, i.e., uma instância responde apenas a um cliente.



EB - Entity Beans

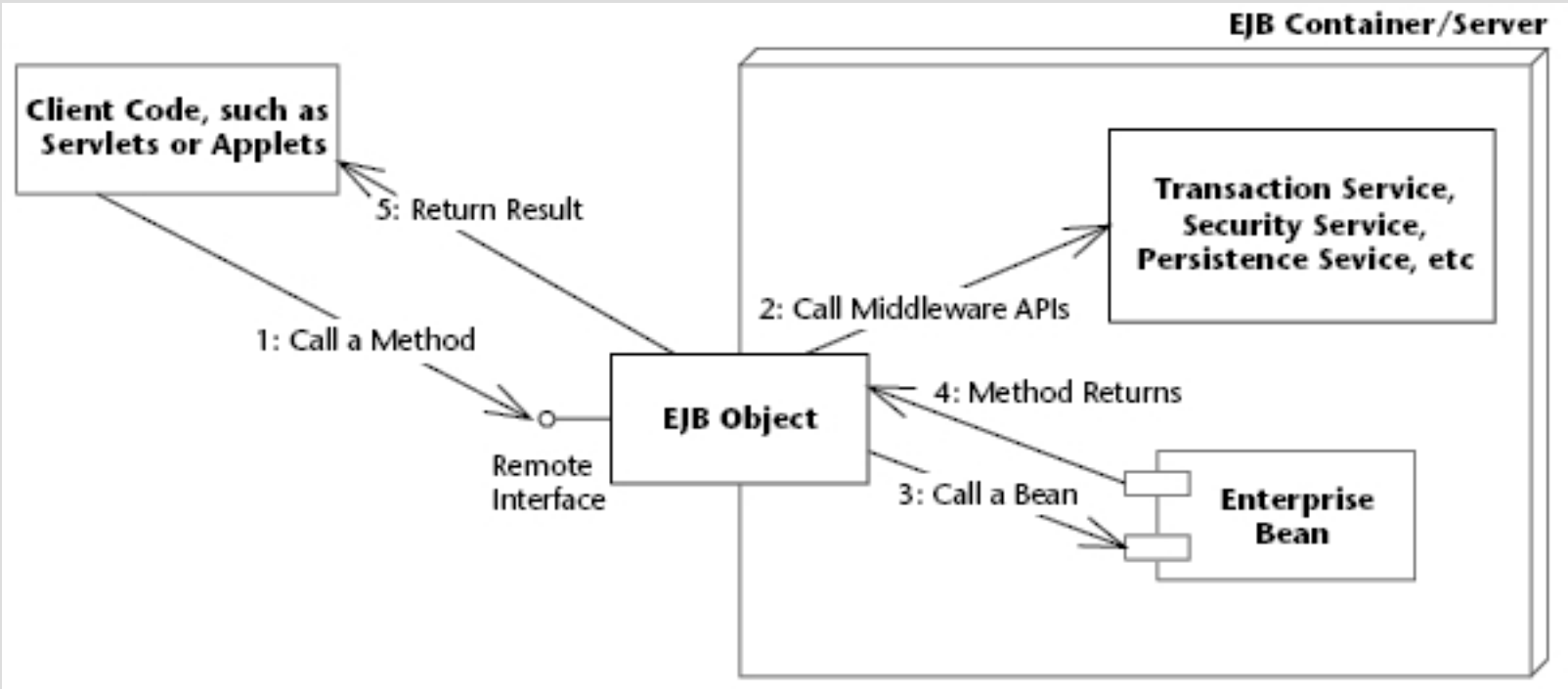
- Objectos distribuídos com persistência. A persistência pode, ou não, ser gerida pelo próprio *bean*.



EB - Message-Driven Beans

- Objectos distribuídos com comportamento assíncrono que gerem operações que não requerem resposta imediata.
- Semelhantes aos *Session Beans* mas difere na invocação, que acontece através do envio de menssagens.

EB – Arquitectura EJB



Connector

- Connector define uma API standard para integrar a tecnologia J2EE com sistemas de informação empresarial.
- Permite que um adaptador de um sistema de informação seja compatível com todos os *containers* J2EE, i.e., permite integrar outras fontes de dados numa aplicação J2EE.

Empacotamento e Instalação

- **Empacotamento:** processo de montagem componentes em módulos e módulos em aplicações empresariais.
- **Instalação:** instalação e costumização de uma aplicação num ambiente/sistema operacional.
- Módulos e aplicações são empacotadas e instaladas como *unidades de instalação*:
  - **EJB:** EJBs e classes relacionadas.
  - **Web:** componentes e recursos web.
  - **Aplicações Cliente:** classes de aplicações cliente.
  - **Adaptadores de Recursos:** conectores, adaptadores e bibliotecas.

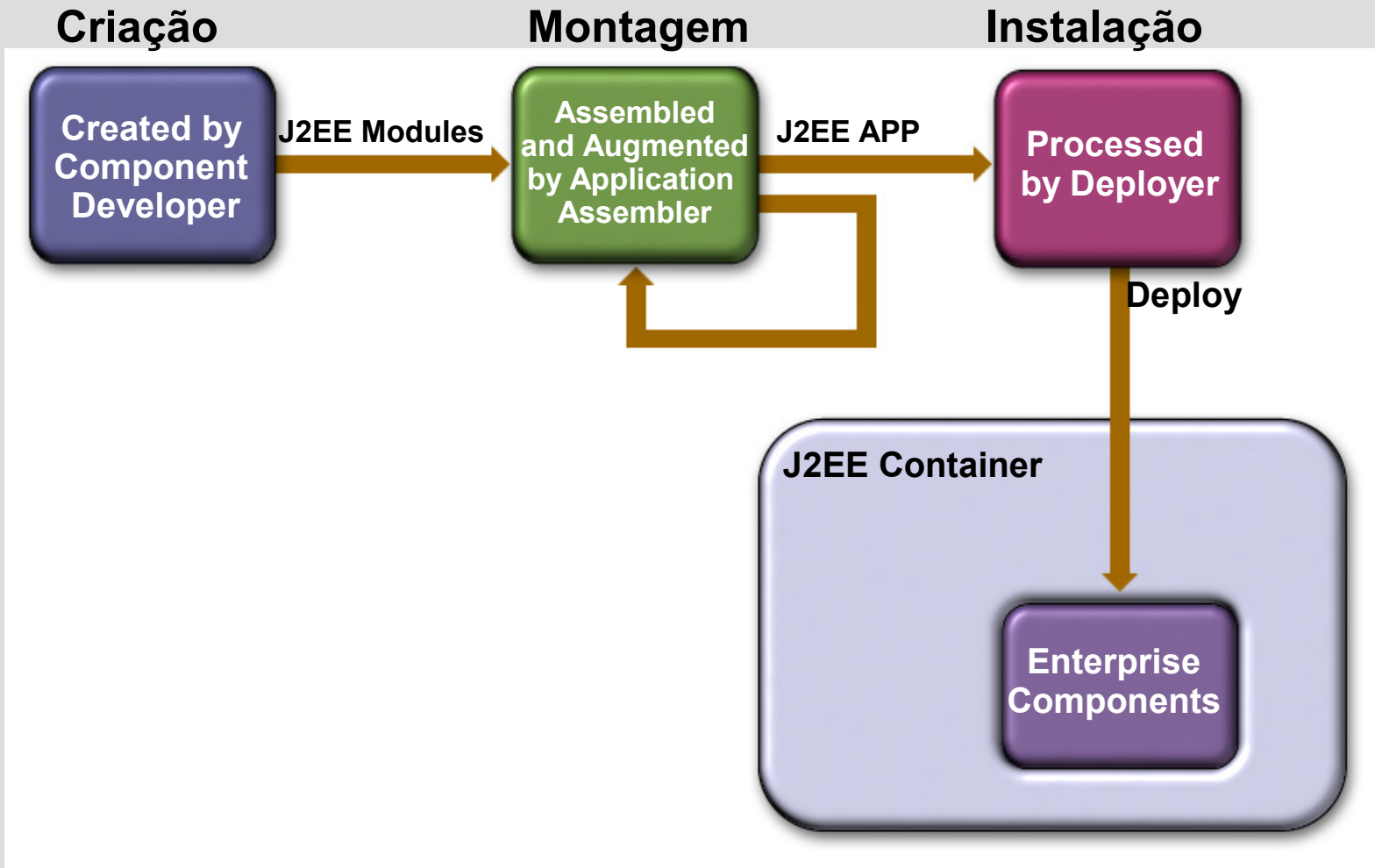
Empacotamento e Instalação

- Descritor de instalação é um ficheiro XML que especifica:
  - Informação Estrutural: meta-dados que descrevem os componentes, não configurável em tempo de instalação.
  - Informação de Montagem: informação opcional que descreve como os componentes de uma unidade de instalação se compõem com outras unidades de instalação para produzir um novo componente.
- Informação particular de soluções especificas, tipicamente definidas num segundo XML contendo extensões do servidor aplicacional.

Empacotamento e Instalação

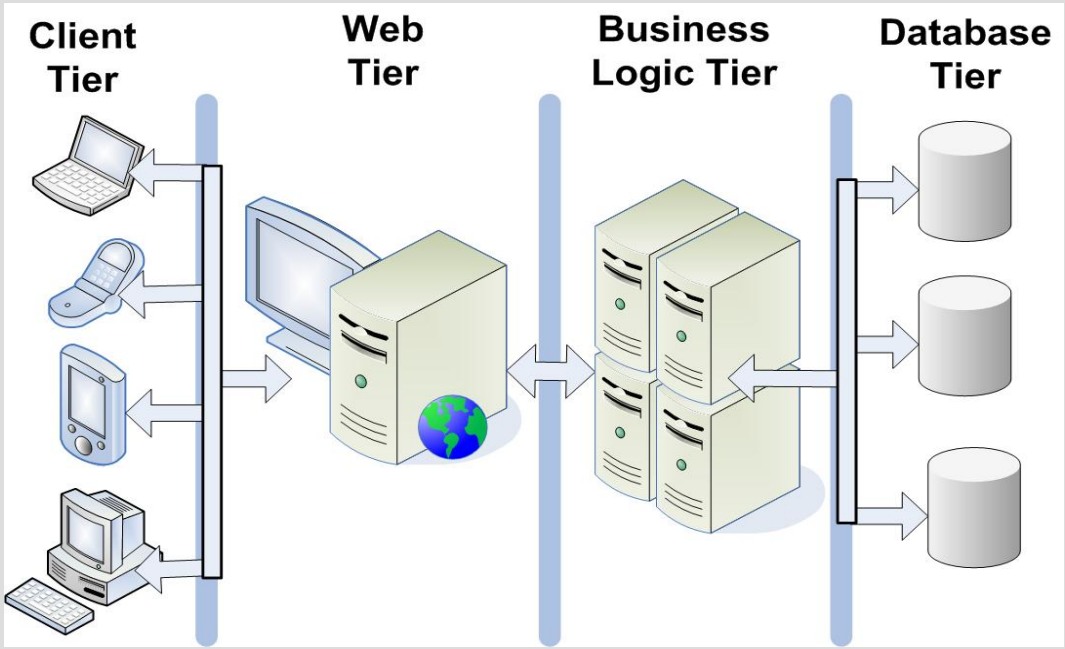
- Componentes são empacotados em módulos e, em conjunto com os seus descritores definem componentes de negócio.
- O módulo de instalação é sempre uma aplicação.
- Ferramentas de montagem resolvem dependências entre descritores de instalação formando *unidades de instalação* maiores.

Empacotamento e Instalação



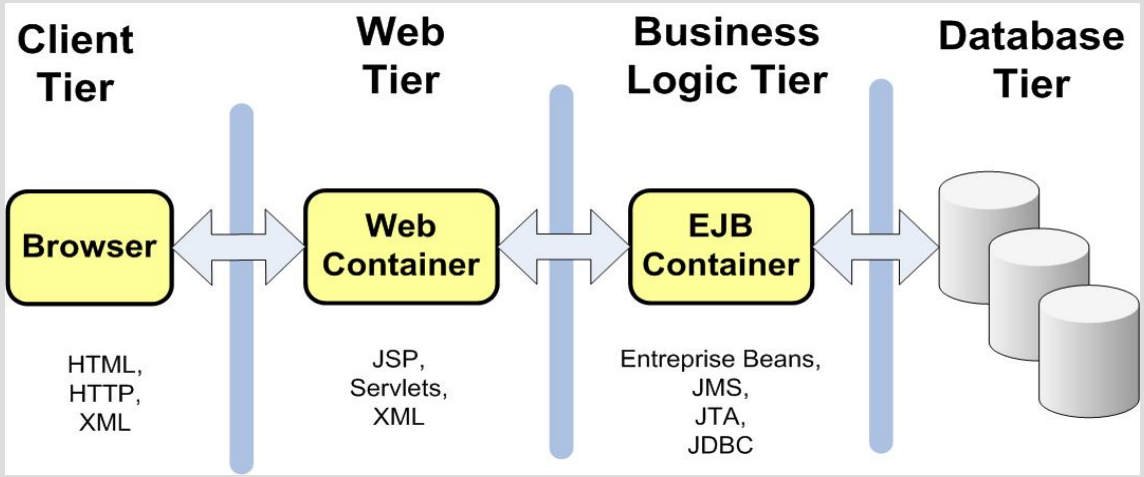
Arquitectura

- Suporte de várias tecnologias cliente e servidor.
- Separação dos vários componentes pelos respectivos *containers* e camadas.
- Possui soluções para lidar com as várias camadas envolvidas permitindo assim a sua interligação.

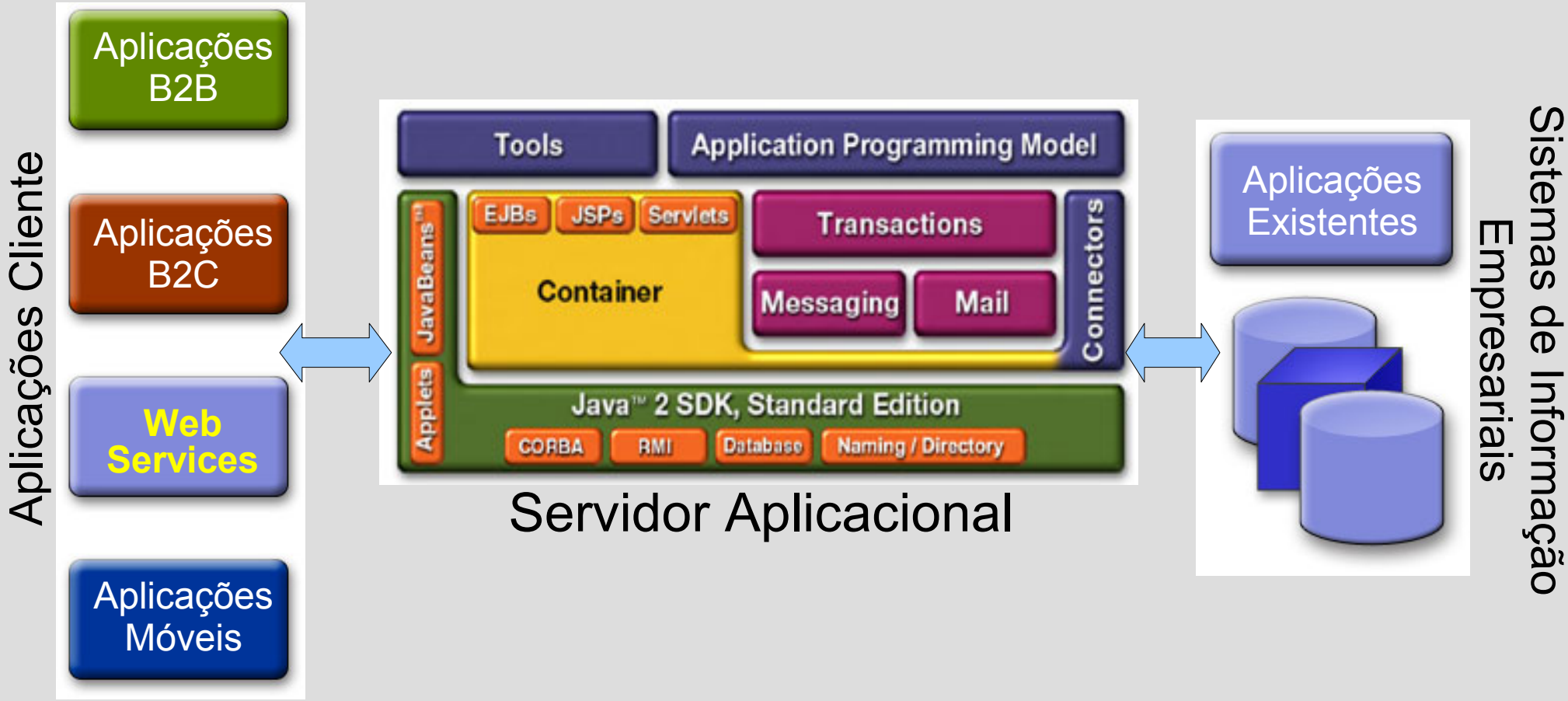


Arquitectura

- Aplicação cliente comunica com a camada de apresentação.
- Componentes de apresentação comunicam com componentes da camada lógica de negócio.
- Componentes de negócio comunicam com camada de dados.



Arquitectura



Exemplo - EJB

- **Remote Interface:** expõe os métodos do EJB para fora do *container*.
- **Home Interface:** especifica a gestão do ciclo de vida de um *bean*.
- **Bean Implementation:** disponibiliza a implementação do *bean*.
- **Deployment Descriptor:** *ejb-jar.xml* contém especificações da aplicação para servidor.

Exemplo - Anotações

- Necessárias quando implementam outras interfaces além de:
  - java.io.Serializable
  - java.io.Externalizable
  - Qualquer interface definida no pacote javax.ebj.
- A interface de negócio é considerada local a menos que seja especificada de outra forma.

Exemplo - Anotações

- @Local:
  - Tem de executar na mesma JVM do *Enterprise Bean* que acede.
  - Pode ser um componente web, uma aplicação cliente ou outro *Enterprise Bean*.
  - Para um cliente remoto, a localização do bean não é transparente.
- @Remote:
  - Pode executar numa máquina e numa Java Virtual Machine (JVM) diferente do *Enterprise Bean* que acede.
  - Pode ser um componente web, uma aplicação cliente ou outro *Enterprise Bean*.
  - Para um cliente remoto, a localização do bean é transparente.

Exemplo – Session Bean

- Criação do *Enterprise Bean*:
  - Interface remota
  - Interface de gestão
  - Implementação da lógica de negócio
- Descritor
- Criação da aplicação cliente

Nota: o exemplo de Session Bean seguinte está disponível na integra na documentação do servidor aplicacional *open source* JBoss 3.0, "JBoss 3.0 Documentation". Consulte a bibliografia para a referência completa sobre este manual.

Exemplo - Remote Interface

- Métodos para invocação remota.

```
package org.jboss.docs.interest;

import javax.ejb.EJBObject;
import java.rmi.RemoteException;

/**
This interface defines the 'Remote' interface for the 'Interest' EJB. Its
single method is the only method exposed to the outside world. The class
InterestBean implements the method.
*/
public interface Interest extends EJBObject
{
   /**
   Calulates the compound interest on the sum `principle', with interest rate per
   period `rate' over `periods' time periods. This method also prints a message to
   standard output; this is picked up by the EJB server and logged. In this way we
   can demonstrate that the method is actually being executed on the server,
   rather than the client.
   */
   public double calculateCompoundInterest(double principle,
                 double rate, double periods) throws RemoteException;
}
```

Exemplo - Home Interface

- Especificação da gestão do *bean*.

```
package org.jboss.docs.interest;

import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

/**
This interface defines the 'home' interface for the 'Interest' EJB.
*/
public interface InterestHome extends EJBHome
{
   /**
   Creates an instance of the 'InterestBean' class on the server, and returns a
   remote reference to an Interest interface on the client.
   */
   Interest create() throws RemoteException, CreateException;
}
```

Exemplo - Bean Implementation

- Implementação do *bean*, lógica de negócio:

```
package org.jboss.docs.interest;

import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

/**
 This class contains the implementation for the 'calculateCompoundInterest'
 method exposed by this Bean.
 */
public class InterestBean implements SessionBean
{
   /* Calulates the compound interest */
   public double calculateCompoundInterest(double principle,
      double rate, double periods)
   {
      System.out.println("Someone called 'calculateCompoundInterest!'");
      return principle * Math.pow(1+rate, periods) - principle;
   }

   [...]
}
```

Exemplo - Deployment Descriptor

- Descritor de instalação e configuração:

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

<ejb-jar>
     <description>JBoss Interest Sample Application</description>
     <display-name>Interest EJB</display-name>
     <enterprise-beans>
       <session>
         <ejb-name>Interest</ejb-name>
         <home>org.jboss.docs.interest.InterestHome</home>
         <remote>org.jboss.docs.interest.Interest</remote>
         <ejb-class>org.jboss.docs.interest.InterestBean</ejb-class>
         <session-type>Stateless</session-type>
         <transaction-type>Bean</transaction-type>
       </session>
     </enterprise-beans>
</ejb-jar>
```

Exemplo - Cliente

- Aplicação cliente

```
package test.client;

import org.jboss.docs.interest.Interest;
import javax.ejb.EJB;

public class TestClient {
    @EJB
    private static Interest interest;

    public TestClient(String[] args) { }

    public static void main(String[] args) {
        TestClient client = new TestClient(args);
        client.testInterest();
    }

    public void testInterest() {
      try {
        double result = interest.calculateCompoundInterest(12.2, 0.05, 3);
        System.out.println(" Result is " + result);
      } catch (Exception ex) {
        ex.printStackTrace();
      }
    }
}
```

Exemplo – Session Bean

- Aplicação cliente

```
package converter.client;
import converter.ejb.Converter;
import java.math.BigDecimal;
import javax.ejb.EJB;

public class ConverterClient {
    @EJB
    private static Converter converter;
    /** Creates a new instance of Client */
    public ConverterClient(String[] args) { }
    public static void main(String[] args) {
        ConverterClient client = new ConverterClient(args);
        client.doConversion();
    }
    public void doConversion() {
        try {
            BigDecimal param = new BigDecimal("100.00");
            BigDecimal yenAmount = converter.dollarToYen(param);
            System.out.println("$" + param + " is " + yenAmount + " Yen.");
            BigDecimal euroAmount = converter.yenToEuro(yenAmount);
            System.out.println(yenAmount + " Yen is " + euroAmount + " Euro.");
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Exemplo – Entity Bean

- Criação do *Enterprise Bean*:
  - Interface (*Account*).
  - Chave primária (*AccountPK*).
  - *Enterprise Bean* (*AccountBean*).

Nota: o exemplo de *Entity Bean* seguinte está disponível na integra no livro, "Mastering Enterprise JavaBeans". Consulte a bibliografia para a referência completa deste livro..

Exemplo – Entity Bean

- Interface:

```
package examples.bmp;

import javax.ejb.*;
import java.rmi.RemoteException;

public interface Account extends EJBObject { /* EJBLocalObject */
    public void deposit(double amt) throws AccountException, RemoteException;
    public void withdraw(double amount)throws AccountException, RemoteException;
    public double getBalance() throws RemoteException;
    public String getOwnerName() throws RemoteException;
    public void setOwnerName(String name) throws RemoteException;
    public String getAccountID() throws RemoteException;
    public void setAccountID(String id) throws RemoteException;
}
```

Exemplo – Entity Bean

- Chave primária:

```
package examples.bmp;

import javax.ejb.*;
import java.rmi.RemoteException;

public class AccountPK implements java.io.Serializable {
  public String accountID;

  public AccountPK(String id) {
   this.accountID = id;
  }

  public AccountPK() { }

  public String toString() { return accountID; }

  public int hashCode() { return accountID.hashCode(); }

  public boolean equals(Object account) {
   if (!(account instanceof AccountPK))
      return false;
   return ((AccountPK)account).accountID.equals(accountID);
  }
}
```

Exemplo – Entity Bean

- Implementação

```
package examples.bmp;

import java.sql.*; import javax.naming.*; import javax.ejb.*;import java.util.*;

public class AccountBean implements EntityBean {
  protected EntityContext ctx;
  private String accountID; // PK
  private String ownerName; private double balance;

  public AccountBean() {
    System.out.println("Bank Account Entity Bean created by EJB Container.");
  }

  public void deposit(double amt) throws AccountException {
    balance += amt;
  }
  public void withdraw(double amt) throws AccountException {
    if (amt > balance) {
     throw new AccountException("Balance is" +balance+ " Cannot withdraw " +amt+ "!");
    }
    balance -= amt;
  }

  [...]
```

Exemplo – Entity Bean

```
public double ejbHomeGetTotalBankValue() throws AccountException {
  PreparedStatement pstmt = null;
  Connection conn = null;
  try {
    conn = getConnection();
    pstmt = conn.prepareStatement("select sum(balance) as total from accounts");
    ResultSet rs = pstmt.executeQuery();
    if (rs.next()) {
      return rs.getDouble("total");
    }
  }
  catch (Exception e) {
    e.printStackTrace();
    throw new AccountException(e);
  }
  finally {
   try { if (pstmt != null) pstmt.close(); }
   catch (Exception e) { }
   try { if (conn != null) conn.close(); }
   catch (Exception e) { }
  }
  throw new AccountException("Error!");
}

[...]
```

Exemplo – Entity Bean

```
public Connection getConnection() throws Exception {
  try {
    Context ctx = new InitialContext();
    javax.sql.DataSource ds;
    ds = (javax.sql.DataSource)ctx.lookup("java:comp/env/jdbc/ejbPool");
    return ds.getConnection();
  }
  catch (Exception e) {
    System.err.println("Couldn't get datasource!");
    e.printStackTrace();
    throw e;
  }
}

[...]
```

Exemplo – Entity Bean

```
/* Called by the container. Updates in-memory object reflecting current values. */
public void ejbLoad() {
 /* Query the Entity Context to get current PK */
 AccountPK pk = (AccountPK) ctx.getPrimaryKey();
 String id = pk.accountID;
 PreparedStatement pstmt = null;  Connection conn = null;
 try {
   /* 1) Acquire a new DB Connection */
   conn = getConnection();
   /* 2) Get account from the DB, querying by account ID */
   pstmt=conn.prepareStatement("select ownerName, balance from accounts where id=?");
   pstmt.setString(1, id);
   ResultSet rs = pstmt.executeQuery();
   rs.next();
   ownerName = rs.getString("ownerName");
   balance = rs.getDouble("balance");
 } catch (Exception ex) {
   throw new EJBException("Account " + pk  + " failed to load from database", ex);
 } finally {
   /* 3) Release the DB Connection */
   try { if (pstmt != null) pstmt.close(); }
   catch (Exception e) { }
   try { if (conn != null) conn.close(); }
   catch (Exception e) { }
 }
}
[...]
}
```

Conclusão

- J2EE integra um conjunto de tecnologias formando assim uma plataforma de desenvolvimento de aplicações distribuídas.
- EJB, com os *containers* e os *beans,* é o cerne da plataforma.
- JNDI é a tecnologia que permite a descoberta dos serviços, como EJBs e ligações JDBC.
- EJB gerido pela *Home Interface*.
- *Remote Interface* define a interface dos métodos disponibilizados para comunicação
- Remete tópicos comuns das aplicações distribuídas para “simples” parametrizações.

Bibliografia

- Sun Microsystems. “Your First Cup:An Introduction to the Java EE Platform”, Sun Microsystems, 2006.
- Ball, Carson, Evans, Fordin, Haase, Jendrok. “The Java™ EE 5 Tutorial”, Sun Microsystems, 2006.
- Alonso, Casati, Kuno, Machiraju. “Web Services: Concepts, Architectures and Applications”, Springer, 2004.
- Roman, Sriganesh, Brose. “Mastering Enterprise JavaBeans”, 3rd Ed. Wiley Publishing Inc., 2005.
- Ashmore. “The J2EE Arquitect's Handbook”, DVT Press, 2004.
- Harrop, Machacek. “Pro Spring”, Apress.
- JBoss Organization. “JBoss 3.0 Documentation”, JBoss Organization, 2002.

Outras Fontes:

- J2EE: http://java.sun.com/javaee/
- The Java EE 5 Tutorial: http://java.sun.com/javaee/5/docs/tutorial/doc/
- The Server Side: http://www.theserverside.com/
- Ant: http://ant.apache.org/
- Spring: http://www.spring.org/
- JBoss: http://www.jboss.org/