

Sistemas Distribuídos na WEB

(Plataformas para Aplicações Distribuídas)

J2EE (Java 2 Enterprise Edition)

Fundamento de EJB

Enterprise Bean

- ✦ Tecnologia EJB é baseada em duas outras tecnologias: Java RMI-IIOP e JNDI
- ✦ Componente de software do lado do servidor que pode ser “entregue” (deployed) em um ambiente em múltiplas camadas.
- ✦ EB (Enterprise Bean) → pode ser composto de um ou mais objetos Java (filosofia de componente)

Tipos de Beans

✦ A EJB 2.0 define três tipos de EB:

- Session Bean
- Entity Bean
- Message-driven Bean

✦ Principais diferenças

✦ Tipos de Session Bean

✦ Dicas Tipos de Beans

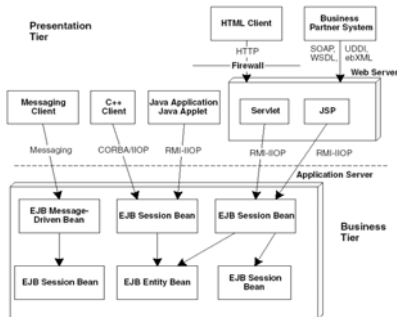


Tipos de Beans

✦ Exemplos: Session Bean chamando um Entity Bean

- Caixa de Banco → Conta Bancária
- “Autorizador” de Cartão de Crédito → Cartão de Crédito
- Sistema de entrada de Pedido → Pedido, Sequência de Pedidos
- Corretor de Leilão → Oferta, Produto
- Rotina de Aprovação de Pedido de Compra → Pedido de Compra

Cientes interagindo com um sistema EJB



5

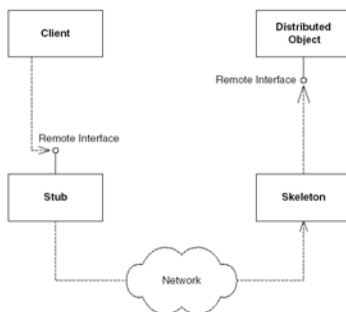
Objetos Distribuídos

- Componentes EJB são baseados em Objetos Distribuídos
- Objeto Distribuídos é um objeto que pode ser chamado de um sistema remoto
- Exemplos de Tecnologias: Java RMI-IIOP, OMG CORBA e Microsoft DCOM.

Sistemas Distribuídos 2007
Prof. Carlos Paes

6

Objetos Distribuídos



7

Objetos Distribuídos e Middleware

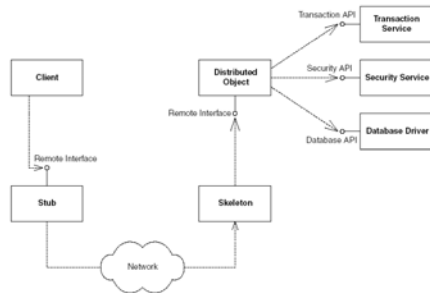
- Objetos distribuídos são importantes, pois permite que uma aplicação seja separada em uma rede
- São necessários alguns serviços providos por um Middleware → por exemplo transação e segurança
- Existem duas formas de Middleware's: Explícito e Implícito

Sistemas Distribuídos 2007
Prof. Carlos Paes

8

Objetos Distribuídos e Middleware

Middleware Explícito



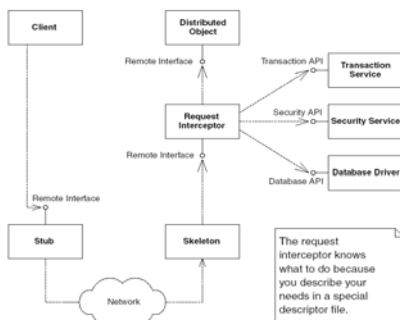
Objetos Distribuídos e Middleware

⚡ Lógica da aplicação misturada com a lógica de chamada das API's de um middleware

⚡ Problemas do Middleware Explícito:

- Dificuldade de implementação
- Dificuldade de Manutenção
- Dificuldade de suporte

OD e Middleware Implícito



Objetos Distribuídos e Middleware

⚡ Sistemas do passado:

- Monitores de Processamento Transacional (TUXEDO e CICS)

⚡ Tecnologias de Objetos Distribuídos Tradicionais:

- CORBA, DCOM e RMI

⚡ Novas Tecnologias de Sistemas Baseados em Componentes:

- EJB, CORBA Component Model e Microsoft .NET

⚡ Objeto Distribuídos → contém apenas lógica de negócio

O que constitui um Enterprise Bean?

✿ A classe Enterprise Bean

- Implementação da lógica de negócio
- Simples classe Java que adere uma interface bem definida e obedece certas regras (necessárias para o Bean executar em qualquer contêiner EJB)
- Contém detalhes de implementação do componente
- Session Beans X Entity Beans X Message-Driven Beans → são bem diferentes umas das outras

Classe Enterprise Bean

- ✿ Especificação EJB → define alguns padrões de interface que a classe Bean pode implementar
- ✿ EJB Componente Model → certos métodos que todos os Beans devem prover (usados pelo contêiner para gerenciar o Bean e notificar o Bean de eventos significantes)
- ✿ Interface básica para todos os Beans:

```
public interface javax.ejb.EnterpriseBean extends java.io.Serializable
{
}
```

Classe Enterprise Bean

- ✿ Os Beans Session Bean, Entity Bean e Message-Driven Bean têm mais interfaces específicas
- ✿ **Session Bean** → `javax.ejb.SessionBean`
- ✿ **Entity Bean** → `javax.ejb.EntityBean`
- ✿ **Message-Driven Bean** → `javax.ejb.MessageDrivenBean`

EJB Object

- ✿ Quando cliente deseja usar uma instância de uma classe EJB, ele nunca invoca diretamente o método de uma instância (objeto remoto)
- ✿ A invocação é interceptada pelo Contêiner EJB e então delegada para a instância do Bean
- ✿ Interceptação da Requisição → EJB contêiner pode realizar o middleware implícito automaticamente

EJB Object

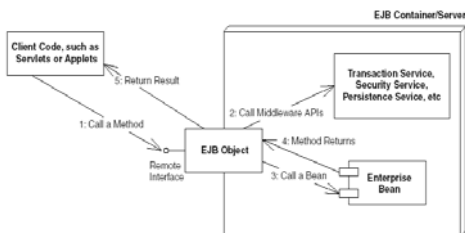
✳️ Alguns serviços realizados pela interceptação do contêiner EJB:

- Gerenciamento de transação distribuída
- Segurança
- Gerenciamento de recurso de ciclo de vida
- Persistência
- Suporte
- Transparência de localização do componente
- Monitoramento

EJB Object

- ✳️ EJB Contêiner → atua como uma camada indireta entre o código do cliente e o Bean
- ✳️ Esta camada indireta → representada por um único objeto na rede chamado de **EJB Object**
- ✳️ EJB Object → Interceptador de requisição
- ✳️ EJB Object → objeto inteligente que sabe como realizar a lógica intermediária que o contêiner necessita antes da chamada do método da instância de uma classe Bean

EJB Object



EJB Object

- ✳️ EJB Objects → partes físicas de um contêiner
- ✳️ Todos os EJB objects possuem um código específico do contêiner (cada contêiner manipula diferentemente o middleware e provém serviços qualitativos)
- ✳️ O EJB Object é gerado automaticamente pelas ferramentas do Container Provider a partir da interface especificada
- ✳️ Precisamos de uma ferramenta para acelerar o desenvolvimento J2EE e um contêiner J2EE:
 - Vamos usar IBM WebSphere Studio Application Developer e o WebSphere Application Server

Remote Interface

- ✳️ *Clientes → invocam métodos de Objetos EJB*
- ✳️ *Objetos EJBs precisam “conhecer” os métodos de negócio que Bean expõe (disponibiliza remotamente)*
- ✳️ *Interfaces remotas → devem aderir regras especiais que a especificação EJB define*
- ✳️ *Toda as interfaces remotas devem derivar de uma interface comum: javax.ejb.EJBObject*

Remote Interface

```
public interface javax.ejb.EJBObject extends java.rmi.Remote
{
    public javax.ejb.EJBHome getEJBHome()
        throws java.rmi.RemoteException;

    public java.lang.Object getPrimaryKey()
        throws java.rmi.RemoteException;

    public void remove()
        throws java.rmi.RemoteException,
            javax.ejb.RemoveException;

    public javax.ejb.Handle getHandle()
        throws java.rmi.RemoteException;

    public boolean isIdentical(javax.ejb.EJBObject)
        throws java.rmi.RemoteException;
}
```

Home Object

- ✳️ *Clientes interagem com EJB Objects e nunca com Beans diretamente*
- ✳️ *Pergunta: Como os clientes adquirem referências para EJB Objects?*
- ✳️ *Clientes não instanciam um EJB Object diretamente, pois eles podem existir em diferentes máquinas diferentes*
- ✳️ *EJB → transparência de localização (cliente não sabe onde reside o EJB Object)*

Home Object

- ✳️ *Para obter uma referência a um EJB Object, um cliente solicita para um EJB Object Factory*
- ✳️ *EJB Object Factory → responsável pela instânciação (e destruição) de um EJB Object*
- ✳️ *Especificação EJB → chama este Factory de Home Object*

Home Object

✿ Principais responsabilidades dos Home Objects:

- Criar EJB Objects
- Localizar EJB Objects existentes
- Remover EJB Objects

✿ Home Objects

- Proprietários e específicos para um determinado contêiner
- Contêiner pode disponibilizar: balanceamento de carga, console para administração, lógica para trilhar informações e etc..

Home Interface

- ✿ Home Object sabe como os EJB Object deve ser inicializado?
- ✿ Home Object pode expor diferentes métodos de inicialização (ex: parametro inteiro ou string)
- ✿ Container precisa saber dessas informações para gerar Home Objects
- ✿ Home Interface → provem estas informações
- ✿ Interface que o Home Interfaces tem que estender: `javax.ejb.EJBHome`

Home Interface

✿ Home Interfaces → simplesmente define métodos para criação, destruição e localização de EJB Objects.

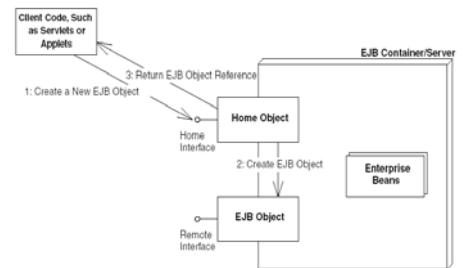
```
public interface javax.ejb.EJBHome extends java.rmi.Remote
{
    public EJBMetaData getEJBMetaData()
        throws java.rmi.RemoteException;

    public javax.ejb.HomeHandle getHomeHandle()
        throws java.rmi.RemoteException;

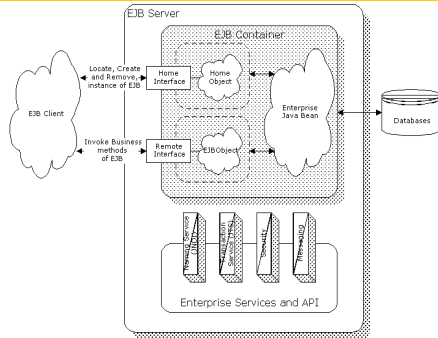
    public void remove(javax.ejb.Handle handle)
        throws java.rmi.RemoteException,
        javax.ejb.RemoveException;

    public void remove(Object primaryKey)
        throws java.rmi.RemoteException,
        javax.ejb.RemoveException;
}
```

Home Interface



Home Interface



Local Interfaces

🐝 *Home Interface e Remote Interface → criar Beans através desta interface é muito lento !*

🐝 *Passos da chamada de um EJB Object:*

1. Cliente → local stub
2. Stub empacota os parâmetros e envia através da rede p/ skeleton
3. Skeleton desempacota os parâmetros
4. Skeleton chama o EJB Object
5. EJB object interage com middleware (pooling de conexão, segurança, transação e serviços de ciclo de vida)
6. EJB Object chama a instancia EB, que realiza o seu trabalho e, em seguida, todo o processo deve ser repetido para o retorno

Local Interfaces

🐝 *EJB 2.0 → define uma forma chamada local através de Local Objects ao invés de EJB Objects*

🐝 *Local Objects → definem uma Local Interface ao invés de uma Remote Interface*

🐝 *Passos de chama de um Local Object:*

1. Cliente chama Local Object
2. Local Object interage com o Middleware (pooling de conexão, segurança, transação e serviços de ciclo de vida)
3. Uma vez que a instancia EB faz o trabalho, ela retorna o controle para o Local Object, na qual retorna o controle para o cliente

Local Interfaces

```
public interface javax.ejb.EJBHome extends java.rmi.Remote
{
    public EJBMetaData getEJBMetaData()
        throws java.rmi.RemoteException;

    public javax.ejb.HomeHandle getHomeHandle()
        throws java.rmi.RemoteException;

    public void remove(javax.ejb.Handle handle)
        throws java.rmi.RemoteException,
            javax.ejb.RemoveException;

    public void remove(Object primaryKey)
        throws java.rmi.RemoteException,
            javax.ejb.RemoveException;
}
```

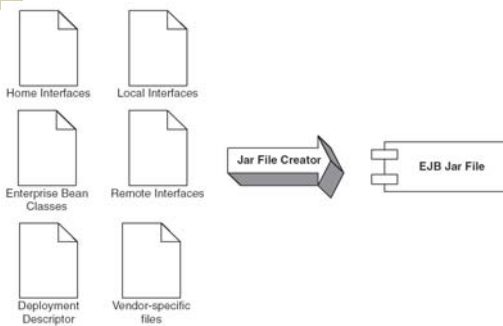
Deployment Descriptors

- ✳ Usado para declarar/informar o contêiner sobre os serviços que serão necessários dos componentes do Middleware
- ✳ Exemplo: como o contêiner deve gerenciar o ciclo de vida, a persistência, controle transacional e serviços de segurança
- ✳ Deployment Descriptor → é a chave para o middleware implícito

Deployment Descriptors

- ✳ EJB 2.0 → O deployment Descriptor é um arquivo no padrão XML
- ✳ Podemos escrever a mão (boa sorte !!) ou usar uma IDE (nosso caso)
- ✳ Deployment Descriptor → específico para cada EJB Server

J2EE Packaging



J2EE Packaging

