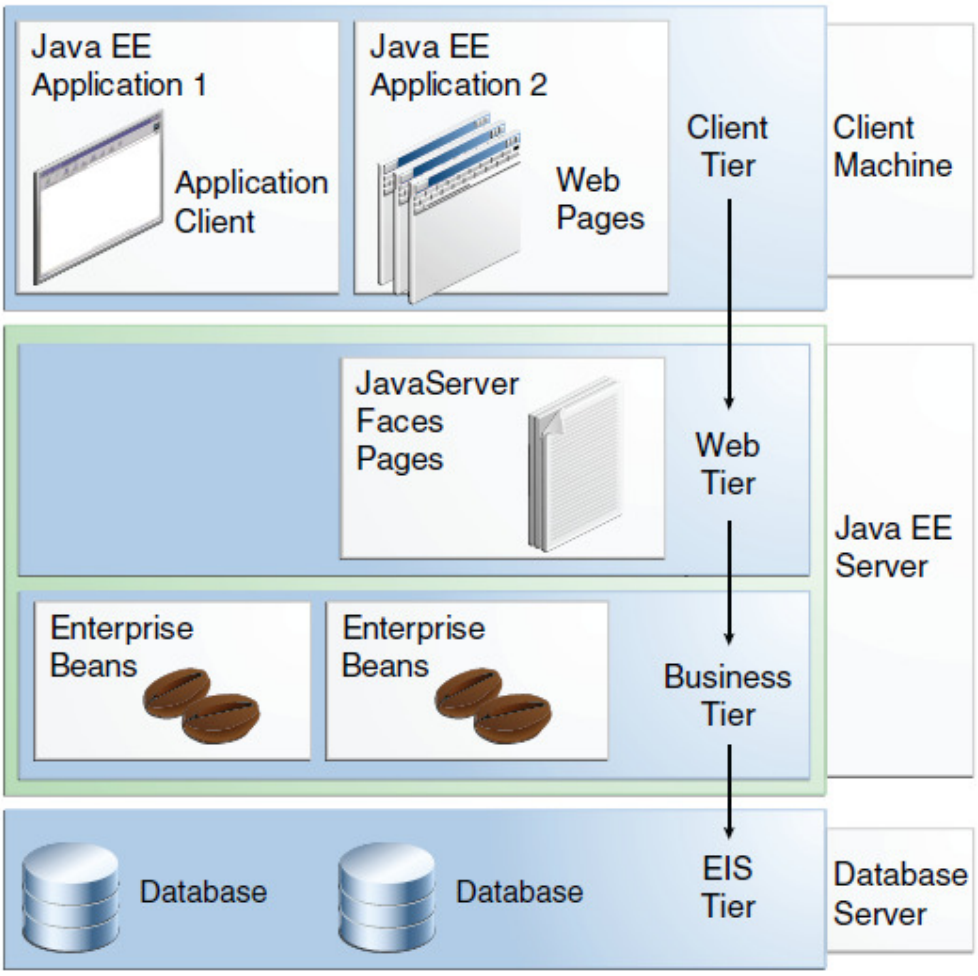
Enterprise Java Beans

Prof. Pasteur Ottoni de Miranda Junior DCC PUC Minas
Disponível em www.pasteurjr.blogspot.com

1-O que é um Enterprise Java Bean?

O *Entertprise Java Bean* (EJB) é um componente *server-side* que encapsula a lógica de negócio de uma aplicação. A lógica de negócio é o código que preenche os objetivos da aplicação. Ocupa a camada web em em um container EJB.



2-Benefícios do uso de EJBs

EJBs simplificam o desenvolvimento de aplicações grandes e distribuídas. Primeiro, porque o container EJB fornece serviços de nível de sistema a elas. Assim sendo o desenvolvedor pode se concentrar em resolver problemas do negócio. O container EJB é responsável por serviços como gestão de transações e autorizações de segurança. Segundo, porque são os EJBs que contêm a lógica de negócios, não os clientes. Assim sendo, o desenvolvedor da aplicação cliente pode se concentrar na apresentação, não tendo que implementar regras de negócio ou bancos de dados de acesso. Como resultado, clientes tornam-se mais leves, executáveis em máquinas menos poderosas.

Terceiro, porque EJBs são componentes portáteis, podendo ser reutilizados em outros aplicativos.

3-Quando usar EJBs

- Quando a aplicação for escalável. Para suportar um crescente número de usuários, pode-se desejar distribuir aplicações por múltipas máquinas.
- Quando transações devam garantir integridade dos dados. EJBs suportam transações.
- Quando a aplicação contém um grande número de clientes, leves e variados.

4-Tipos de EJBs

4.1-*Session Beans*

Session beans encapsulam lógica de negócio que pode ser invocada programaticamente por um cliente de maneira local, remota ou via *web service.* Para acessar uma aplicação armazenada em um servidor, o cliente invoca os métodos do *session bean.*

Tipos de session beans

Stateful: Em um EJB *stateful,* suas variáveis de instância representam o estado de uma sessão única aberta entre o cliente e o EJB. Devido ao fato do cliente interagir com o o EJB, esse estado é frequentemente denominado estado conversacional. O estado é mantido enquanto durar a sessão cliente/EJB. São apropriados para as seguintes situações:

- O estado do EJB representa a interação entre o EJB e um cliente específico;
- O EJB precisa manter informação do cliente entre invocação de métodos;

Stateless. Esses EJBs não mantêm um estado conversacional com o cliente. Quando o cliente invoca métodos de um EJB *stateless,* as variáveis de instância mantêm um estado específico apenas enquanto durar a execução do método. Quando esta termina, o estado não é mantido. Exceto durante a invocação de métodos, todas as instâncias de EJBs *stateless* são equivalentes, permitindo alocar uma instância para qualquer cliente.
Provêem melhor escalabilidade para aplicações que requerem um número maior de clientes, pois aplicações requerem menos EJBs *stateless* do que *stateful* para atender ao mesmo número de clientes.
Para incrementar a performance, deve-se escolher um EJB *stateless* se:

- Em uma única invocação de método, o EJB realiza uma tarefa genérica para todos os clientes;
- O EJB implementa um *web service.*

Singleton - São instanciados apenas uma vez por aplicação e existem durante o ciclo de vida da mesma. São projetados para circunstâncias nas quais uma única instância do EJB é compartilhada e concorrentemente acessada por clientes. Oferecem a mesma funcionalidade dos EJBs *stateful,* a menos da instância única. O estado é mantido entre invocações de clientes, mas não quando ocorrem quedas do servidor.
São apropriados nas seguintes circunstâncias:

- O estado precisa ser mantido durante a execução da aplicação;
- Um único EJB precisa ser acessado por múltiplos *threads* concorrentemente;

- A aplicação precisa necessita que o EJB realize tarefas entre seu *startup* e *shutdown.*
- O EJB implementa um *web service.*

4.2-*Message-driven Beans*

São EJBs que permitem a aplicações JEE processar mensagens assincronamente. Agem de maneira semelhante a um *listener* de eventos, mas ao invés de eventos, recebe mensagens provenientes de aplicações, outro EJB ou componentes web.
Clientes não localizam message-driven beans e invocam métodos diretamente neles. Eles são acessados através de um serviço de mensagens (JMS), enviando mensagens ao destinatário (*MessageListener).* São executados mediante a recepção de uma mensagem vinda do cliente, assincronamente e *stateless.*
Quando a mensagem chega, o container chama o método *onmessage* do message-driven bean , que por sua vez chama métodos auxiliares (*helper methods)* para processá-la. Tudo isso ocorre em um contexto transacional.
Eles não são acessados via interfaces, como session beans, ou seja, possuem apenas uma classe bean.

5-Acesso local ou remoto

Acesso local: Cliente deve estar na mesma aplicação do EJB. Pode ser um componente web ou outro EJB. *@ Local* deve ser colocado no cabeçalho da interface do EJB local

Acesso remoto: Pode rodar em uma máquina e JVM diferentes. Pode ser um componente web, uma aplicação cliente ou outro EJB. *@ Remote* deve ser colocado no cabeçalho da interface do EJB remote.

6-Interface do EJB

Os métodos que darão acesso às regras de negócio do EJB devem estar contidos dentro de uma interface Java, como no exemplo abaixo:

```
package com.webage.ejb;

import javax.ejb.*;

@Remote
public interface SimpleBean {
      public String sayHello(String name);
}
```

7-A classe que implementa a interface

A classe que implementa a interface acima deve estar contida no mesmo pacote da mesma. Repare, abaixo, como declaramos o tipo do EJB (@Stateless, no caso).

```
package com.webage.ejb;

import javax.ejb.*;

@Stateless(name="Example", mappedName="ejb/SimpleBeanJNDI")
public class SimpleBeanImpl implements SimpleBean {
    public String sayHello(String name) {
        return "Hello " + name + "!";
    }
}
```

8)O cliente EJB

A classe exibida abaixo faz o acesso ao Session Bean criado. O objeto ctx da classe InitialContext é instanciado e é um contexto criado para acesso ao Session Bean através do chamado JNDI (Java Name and Directory Interface) Lookup, que é uma interface para localização da instância do session bean. Em seguida, o SessionBean é instanciado, por meio de uma chamada ao método *lookup* de InitialContexto, passando-se com o parâmetro o caminho JNDI. Na linha seguinte, o método correspondente do bean (sayHello) é invocado explicitamente.

```
package com.webage.client;

import javax.naming.*;

import com.webage.ejb.SimpleBean;

public class TestClient {

      public void runTest() throws Exception {
            InitialContext ctx = new InitialContext();
            SimpleBean bean = (SimpleBean)
ctx.lookup("ejb/SimpleBeanJNDI");
            String result = bean.sayHello("Billy Bob");
            System.out.println(result);
      }

      public static void main(String[] args) {
            try {
                  TestClient cli = new TestClient();
                  cli.runTest();

            } catch (Exception e) {
                  e.printStackTrace();
            }
      }
}
```

REFERÊNCIAS

SUN Microsystems/Oracle – *Tutorial JEE6*

Material disponível em
http://www.webagesolutions.com/knowledgebase/javakb/jkb005/index.html