Objetos Distribuídos

EJB

EJB

Enterprise JavaBeans (EJB): uma arquitetura gerenciada de componente do lado do servidor utilizada para encapsular a lógica de negócios de uma aplicação. A tecnologia EJB permite o desenvolvimento rápido e simplificado de aplicações distribuídas, transacionais, seguras e portáteis baseadas na tecnologia Java;

EJB

A segurança, a grande quantidade de acessos simultâneos, a integridade das informações, a estabilidade, entre outros aspectos podem ser gerenciados com EJBs e container

EJB

- EJB oferece serviços de persistência de dados, **controle de concorrência, envio de mensagens, serviço de agendamento, chamadas a métodos remotos e a web services**

Container EJB

- Aplicações Java: é necessário o uso de uma Máquina Virtual Java. A JVM fornece uma infraestrutura que permite a execução de aplicações, garante o controle do uso de memória e assegura que a aplicação execute em diversos sistemas operacionais

Container EJB

- WebLogic Server Application da Oracle e o WebSphere da IBM.
- servidores de software livre como JBoss e o GlassFish

OBS: Tomcat é um servidor web e não é possível utilizá-lo para executar aplicações EJB

Container EJB

- container EJB é responsável por prover serviços e recursos especiais para seus componentes. A ação de colocar a aplicação EJB dentro de um container é chamada de implantação, ou **deploy**

Componente EJB

- Classes javas, podem conter lógica de negócio ou de persistência de dados;
- Dois tipos: **Beans de Sessão** e **MDB** (Message-Driven Bean), além das entidades **JPA**, que não são EJBs e substituem os antigos entity beans.
- Os Beans de sessão dividem-se: **stateless, stateful e singleton**.

Componente EJB

- beans de sessão e os MDBs são responsáveis por conter as **lógicas** de negócio corporativas.

OBS: O ESTUDO PRINCIPAL são os **beans de sessão**, por serem os componentes mais utilizados e de menor complexidade

Anotação

- grande melhoria que o EJB sofreu a partir de sua versão 3.0, foi a possibilidade de usar **anotações** ao invés de fazer toda a configuração em arquivos XML
- **A anotação no EJB é uma forma de dizer para o container para fazer alguma coisa**

Anotações

- o container irá executar uma série de instruções de forma transparente para atender o seu propósito, lembrando que toda anotação é precedida de **‘@´**
- Para injetar um componente EJB, utiliza-se a anotação **@EJB** sobre o componente em questão

Componente EJB

- Quando o container ler o @EJB,ele irá até o pool de beans, obter um bean que esteja disponível e retorná-lo para a aplicação.

Beans de Sessão

- Contém as regras de negócio
- Os beans de sessão podem ser acessados de duas formas: local ou remota; utilizando as anotações **@Local** ou **@Remote**, respectivamente.

Beans de Sessão

- **@Local**, significa que ele só poderá ser acessado por requisições originárias do próprio container em que o bean é executado.
- **@Remote** diz que o bean pode ser acessado de forma remota, ou seja, pode ser acessado por qualquer cliente, em qualquer container de qualquer parte do mundo.

Beans de Sessão

Injetando Componentes

Desenvolvimento tradicional:

```
Calculadora calculadora = new Calculadora()
```

Desenvolvimento tradicional:

```
@EJB
 Calculadora calculadora;
```

Beans de Sessão

Stateless (sem estado): são responsáveis por conter operações que não necessitam durar mais do que uma chamada. São responsáveis por conter operações que não necessitam durar mais do que uma chamada

Beans de Sessão

- O container EJB controla a quantidade de instâncias do componente do tipo **stateless** automaticamente conforme a necessidade.
- as instâncias ficam em um pool de beans e são invocadas sempre que há uma solicitação
- Após sua utilização ela volta para o pool e fica disponível para uma próxima chamada.
- instância do bean stateless esteja em uso, ela não pode ser utilizada ao mesmo tempo por outro usuário

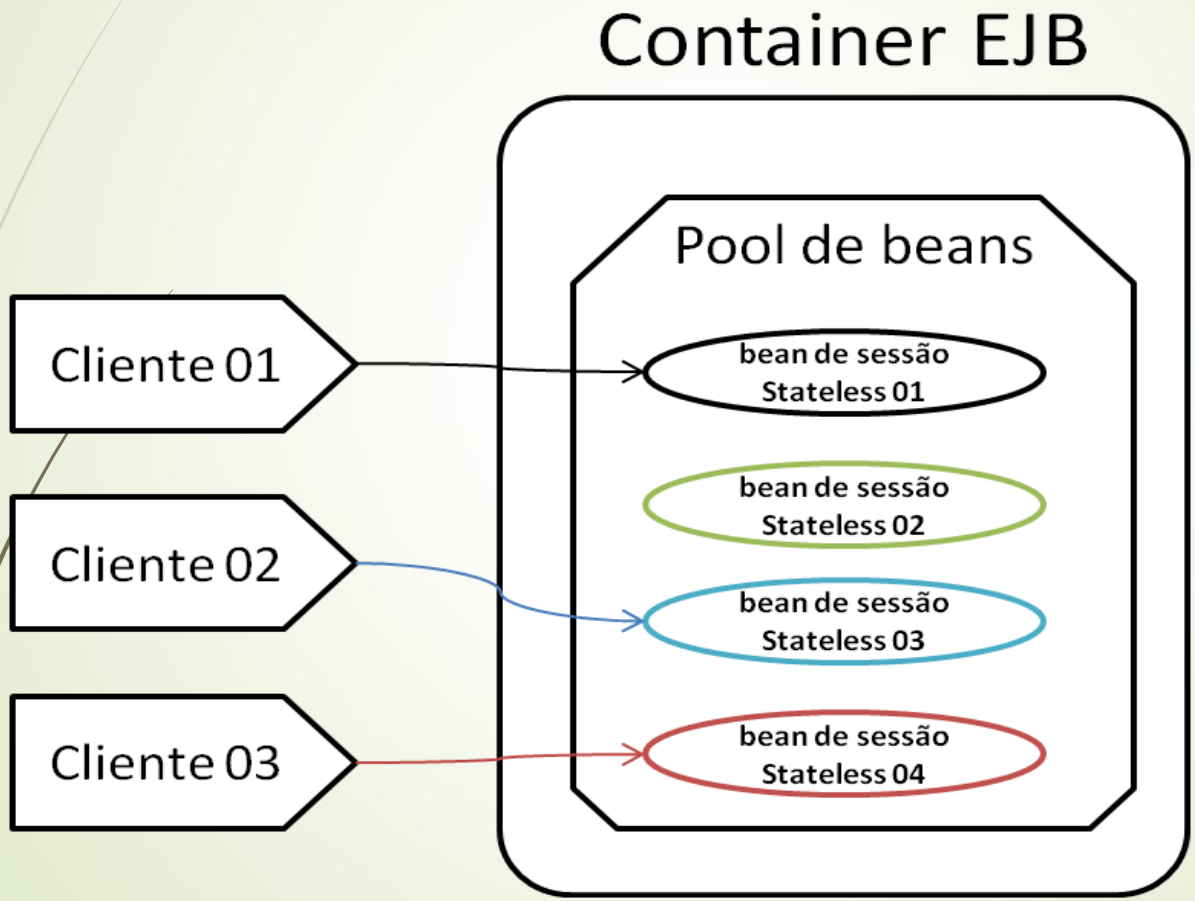
Beans de Sessão stateless

- **Quando usar:** conversão de dados, componentes que possuem métodos para manipulação de Strings, e componentes que executem chamadas para inserir ou realizar uma busca no banco de dados, etc.
- **Eles não precisam durar mais do que uma chamada.**

Exemplo stateless

```
import java.text.SimpleDateFormat;
import java.util.Date;

import javax.ejb.LocalBean;
import javax.ejb.Stateless;

@Stateless
@Local
public class ConversorBean {

public String formatarData(Date data){

SimpleDateFormat formata = new SimpleDateFormat("MM/dd/yyyy");
return formata.format(data);
}
}
```

Exemplo stateless



Beans de Sessão stateful

- Os beans de sessão do tipo **stateful** são responsáveis por conter operações que necessitam durar **mais do que uma chamada**, ou seja, que **após a execução do componente o estado dos objetos(os valores) modificados seja mantido.**

Beans de Sessão stateful

- uma vez criada uma instância para um usuário, ela não é em nenhum momento utilizada por outras requisições de usuários diferentes.
- **Aumenta o consumo de memória**
- o EJB dispõe de um recurso que permite que o desenvolvedor **libere a instância** do componente ligado ao usuário, quando esta não for mais utilizada

Beans de Sessão stateful

- Para fazer isso (liberar a instância), basta colocar a anotação **@Remove** em um método. Assim, quando este método for invocado e concluído, a instância em questão será excluída pelo container EJB.

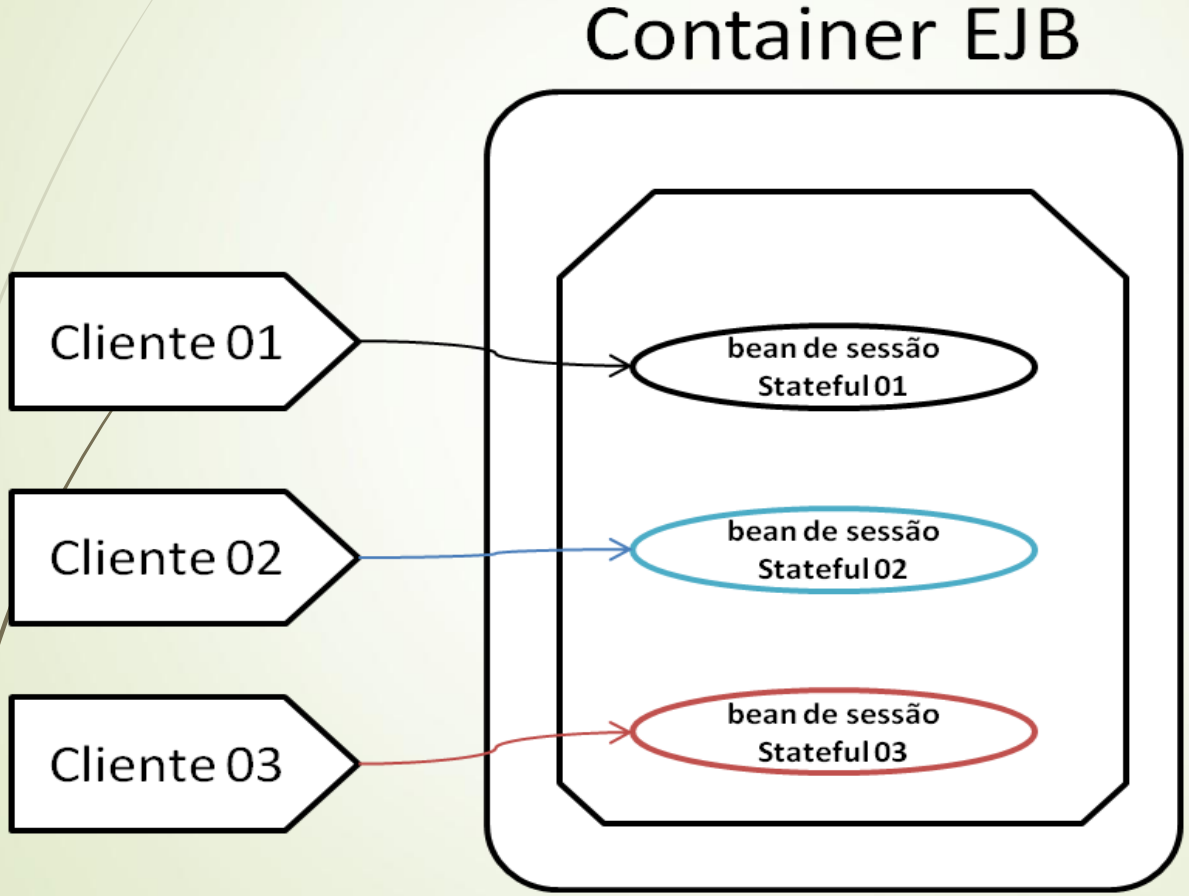
Beans de Sessão stateful - exemplo

- Em situações reais para o uso do componente stateful, pode-se listar um **site de compras**, onde o usuário pode adicionar itens ao seu carrinho de compras e depois voltar a navegar pelo site e efetuar várias requisições que os itens escolhidos permanecem salvos no carrinho do usuário. **Sem o uso do bean Stateful este recurso de salvar o item na memória não seria simples de ser desenvolvido.**

Beans de Sessão stateful - exemplo

```
import javax.ejb.Local;
import javax.ejb.Stateful;
import javax.ejb.Remove;

@Stateful
@Local
public class ItensBean {

  public void addItensNoCarrinho(String item) {
    //Lógica para adicionar itens no carrinho
  }

   @Remove
   public void finalizarCompra(){
     //Após a execução deste método, a instancia será des
truída pelo container
  }
}
```

Beans de Sessão stateful - exemplo



Beans de Sessão Singleton

- bean de sessão do tipo Singleton foi uma das novidades da versão 3.1
- Como já citado, os beans do tipo **Stateless** duram apenas uma requisição e os **Stateful** duram várias requisições de um mesmo usuário. Já o tipo **Singleton** é criado **apenas uma vez e disponível para todos os usuários do container EJB**.

Beans de Sessão Singleton

- Deste modo, qualquer alteração feita em um componente Singleton estará visível para todos os usuários da aplicação
- Outra funcionalidade disponível nos beans de sessão Singleton são as anotações **@ReadOnly e @ReadWrite**, que dizem ao container se o bean é imutável ou se ele pode sofrer alterações

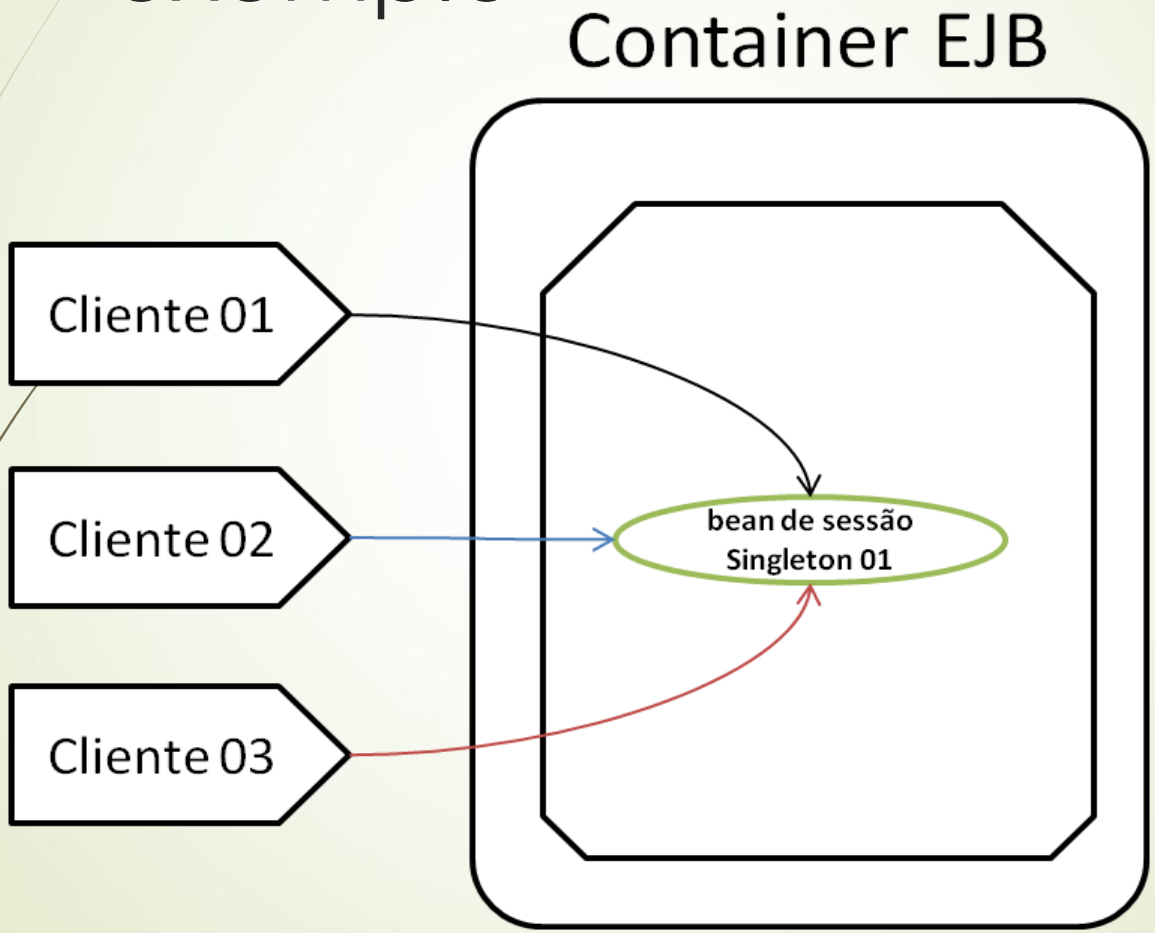
Beans de Sessão Singleton - exemplo

```
import javax.ejb.Local;
import javax.ejb.Singleton;

@Singleton
@Local
public class MensagemBean {

public String exibirMensagem(){
//Lógica para recuperar a mensagem
}
}
```

Beans de Sessão Singleton - exemplo

- Imagine um sistema online, onde o administrador em determinadas ocasiões tem a necessidade de exibir mensagens para todos os usuários logados no sistema. Ao utilizar um componente singleton que contém uma lista de Strings, por exemplo, o administrador pode inserir as mensagens e todos os usuários terão acesso, pois haverá sempre uma única instância presente no container EJB.

Beans de Sessão Singleton - exemplo



Métodos *callback*

- Os métodos callback do EJB são um grande aliado na construção de componentes distribuídos, pois permite ao desenvolvedor tomar ações em determinados momentos de seu ciclo de vida. **Eles são invocados pelo container em situações específicas durante o ciclo de vida de um componente.**

Métodos *callback*

- Com eles é possível **implementar lógicas de negócio em determinadas fases do componente**, como por exemplo, **abrir uma conexão** com o banco de dados sempre que um componente **for criado ou liberar** um recurso sempre que uma instância for destruída pelo container

Métodos *callback*

- Para declarar um método callback a única coisa necessária é colocar uma **anotação no método desejado**. Os métodos callback são analisados a seguir:

@PostConstruct

O método que possuir a anotação @PostConstruct será invocado sempre que o container criar uma nova instância do componente EJB.

Métodos *callback*

Este método pode ser usado para abrir uma conexão com o banco ou inicializar determinado processo

@PreDestroy

O método anotado com @PreDestroy será invocado pelo container EJB momentos antes da instância do componente ser destruída. Assim, é possível criar lógicas para liberação de recursos ou mesmo fechar uma conexão com o banco de dados.

Os callbacks @PostConstruct e @PreDestroy podem ser usados para os três tipos de componentes do EJB: stateless, stateful e singleton.

Métodos *callback*

- componente **Stateful** atende somente a um usuário e armazena todas as informações de sua sessão. Com isso, é acumulada uma **grande quantidade de instâncias no servidor**, o que sobrecarregará a memória. Para amenizar este problema, o container EJB, de forma transparente e automática, aplica um recurso chamado de **passivação**, que consiste em retirar um componente **que não esteja em uso da memória e guardá-lo em disco, liberando memória no servidor**. Se o usuário invocar qualquer método deste componente que se encontra passivado, o container imediatamente recupera a instância e a coloca de volta na memória.

Métodos *callback*

- ***@PrePassivate***
  O método anotado com @PrePassivate será invocado antes do componente EJB sofrer o processo de passivação. Assim é possível implementar lógicas para liberar recursos ou executar alguma funcionalidade propícia.
- ***@PostActivate***
  O método que possuir a anotação @PostActivate será invocado após um componente EJB ser resgatado do disco e voltar para a memória. Este processo também é chamado de ativação.
- ***Os callbacks @PrePassivate e @PostActivate podem ser usados somente para componentes do tipo stateful.***