

Stateful Session Beans

Stateful Session Beans

- Quando um cliente chama um método de um bean, ele está iniciando um diálogo
 - O estado do diálogo deve ser mantido para a próxima requisição
 - Container não pode fazer o mesmo tipo de pooling que faz com stateless session beans
- *Passivação e ativação*
 - Solução para o problema de pooling
 - Dados do bean são armazenados em meio persistente durante a passivação e recuperados na ativação
 - Permite manter poucas instâncias no ar e vários clientes
 - Estratégia comum: **LRU - Least Recently Used** - Se container precisar de recursos, beans menos usados serão passivados. Logo que receberem uma requisição, serão reativados
- Objetos são serializados, portanto, fazem parte do estado do diálogo apenas objetos e variáveis **não transientes**

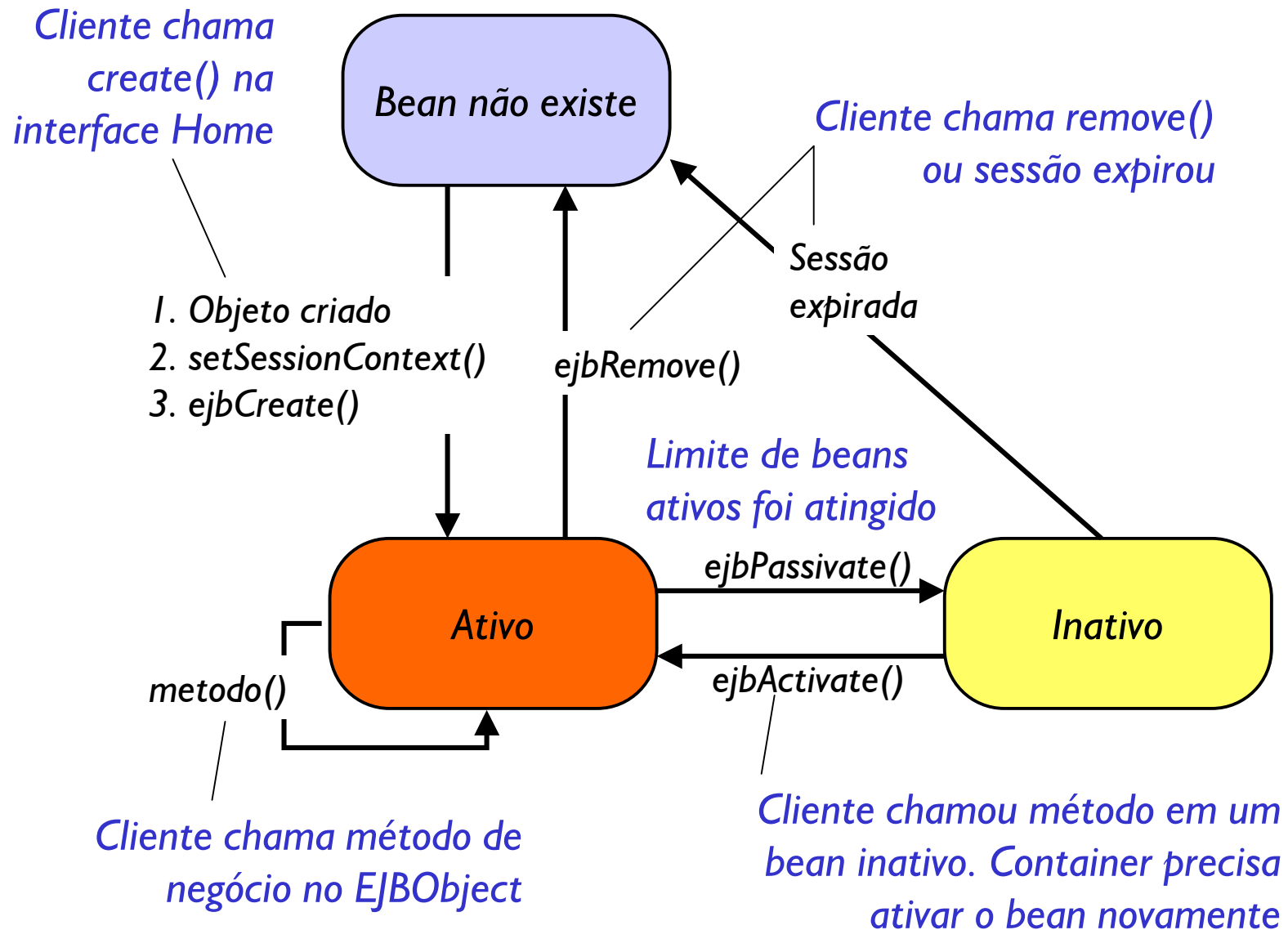
Métodos de um Stateful Session Bean

- *Stateful Session Beans precisam cumprir o mesmo contrato que Stateless beans, mas existem mais operações que podem ser implementadas*
- **void setSessionContext**(SessionContext ctx)
 - *Mesma função que stateless beans*
- **void ejbCreate**([Tipo argumentos])
 - *Deve haver pelo menos um. Argumentos são opcionais. Pode haver vários ejbCreate(), diferenciados pelo nome e tipo de argumentos*
- **void ejbRemove**()
 - *Chamado antes de liberar recursos e remover o bean da memória*
- **void ejbPassivate**()
 - *Chamado antes de fazer o swap do bean para o disco*
- **void ejbActivate**()
 - *chamado depois que estado do bean é recuperado do disc*

Ativação e Passivação

- Se há mais clientes **realizando operações contínuas** que beans no pool, container pode criar mais instâncias
- Se há mais clientes que beans no pool, mas uma boa parte permanece inativa por certos períodos (cenário realista), container gerencia recursos usando **ativação** e **passivação**
 - Administrador do sistema pode configurar o servidor para obter o melhor desempenho.
- Exemplo: Pool com 5 beans ativos (que participam de sessão) é chamado por sexto cliente.
 - 1. Estado do bean que foi usado há mais tempo é gravado em meio persistente. Antes, seu método **ejbPassivate()** é chamado
 - 2. Se cliente está continuando diálogo iniciado anteriormente, o estado anterior do bean é recuperado do meio persistente e usado para preencher o bean. Depois, **ejbActivate()** é chamado.

Ciclo de vida: Stateful Session Bean





Exemplo de Stateful Session Bean

- Veja exemplo no subdiretório *cap06*
 - *mejb2*: *CountBean* (comentado)
 - *sun*: *CheckerBean* e *CartBean*
- Para executar use o Ant (configure *build.properties*)
 - `ant jboss.deploy` (cria o JAR e copia para o deploy no JBoss)
 - `ant run.jboss.client` (executa um cliente de aplicação)
- Execução
 - Para demonstrar o efeito `ejbActivate()` e `ejbPassivate()`, o bean pool foi reduzido artificialmente (usando *jboss.xml*) para uma capacidade máxima de 2 beans
 - Como o cliente cria mais de dois beans, o container terá que passivar e depois ativar os beans (veja na saída da execução do JBoss quando cada método é chamado)
 - Observe a ordem utilizada pelo container para ativar e passivar (veja que ele passiva o bean usado menos recentemente)



Interfaces Remote e Home

```
package examples;  
  
public interface Count extends javax.ejb.EJBObject {  
    public int count() throws java.rmi.RemoteException;  
}
```

```
package examples;  
  
public interface CountHome extends javax.ejb.EJBHome {  
     Count create(int val)  
        throws java.rmi.RemoteException, CreateException;  
     Count create()  
        throws java.rmi.RemoteException, CreateException;  
}
```

Enterprise JavaBean

```
package examples;
public class CountBean implements javax.ejb.SessionBean {
    private SessionContext sessionContext;
    public int val;

    public int count() {
        System.out.println("count() chamado");
        return ++val;
    }
    public void ejbCreate(int val) throws CreateException {

        this.val = val;
        System.out.println("ejbCreate(val) chamado");
    }
    public void ejbCreate() throws CreateException {

        this.val = 0;
        System.out.println("ejbCreate() chamado");
    }
    public void ejbRemove() {
        System.out.println("ejbRemove() chamado");
    }
    public void ejbActivate() {
        System.out.println("ejbActivate() chamado");
    }
    public void ejbPassivate() {
        System.out.println("ejbPassivate() chamado");
    }
    public void setSessionContext(SessionContext ctx) {
        this.sessionContext = ctx;
    }
}
```


ejbCreate() e *create()*

- Para cada *create()* em *Home* deve existir um *ejbCreate()* no bean.
 - O número e tipo de argumentos deve combinar
 - Cada *create()* de *Home* retorna o tipo da interface do componente
 - Cada *ejbCreate()* do bean retorna void
 - Ambos provocam as mesmas exceções (exceto *RemoteException* que só é provocada nas interfaces *Home* remotas)

Deployment Descriptor

```
<!DOCTYPE ejb-jar PUBLIC
"-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
"http://java.sun.com/dtd/ejb-jar_2_0.dtd">

<ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>CountEJB</ejb-name>
      <home>examples.CountHome</home>
      <remote>examples.Count</remote>
      <ejb-class>examples.CountBean</ejb-class>
      <session-type>Stateful</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
  ...
</ejb-jar>
```

Exercício resolvido em sala (I)

I. Comparação Stateful-Stateless

- Remova o método *create(...)* com argumentos do exemplo do *CountBean* (deixando apenas o sem argumentos).
- Adapte o cliente para que ele use apenas *create()* e não *create(valor)*
- Execute a aplicação
- Mude, no deployment descriptor, o bean para *Stateless* e rode a aplicação novamente. Veja e discuta os resultados.

2. Stateful Session Bean que armazena um String

a) Implemente um session bean com a seguinte interface:

Carrinho
<code>+adicionarProduto(String)</code> <code>+listarProdutos():String</code>

- Bean mantém **um string** com um produto por linha.
- Listar produtos retorna o string
- Adicionar faz append no string.
- Preencha o `ejb-jar.xml` e `jboss.xml`. Faça o deploy.

b) Crie um cliente que

- Crie dois carrinhos
- Preencha ambos com 3 ou 4 produtos diferentes
- Liste o conteúdo de ambos.

Exercícios extras (opcionais)

3. Carrinho de compras usando ProdutoVO

- a) Implemente um Stateful Session Bean que mantenha **na memória** um HashMap de objetos ProdutoVO.
- b) O bean deve ter a seguinte interface e utilizar o objeto serializável ProdutoVO

```
public ProdutoVO[] listarConteudoCarrinho();  
public void adicionarProduto(ProdutoVO produto);  
public void removerProduto(String cod);  
public ProdutoVO detalharProduto(String cod);
```

- c) Comparação com Stateless: torne seu bean Stateless e veja o que acontece.

- [1] Ed Roman, *Mastering EJB 2*, 2002, Capítulo 4.
- [2] Dale Green. *Session Beans*. J2EE Tutorial, Sun
- [3] Linda de Michiel et al. *Enterprise JavaBeans 2.1 Specification*. Sun Microsystems, 2003.

Curso J530: Enterprise JavaBeans

Revisão 2.0 - Junho de 2003

© 2001-2003, Helder da Rocha
(helder@acm.org)

 argonavis.com.br