Entity Beans com persistência explícita (BMP)

Helder da Rocha (helder@acm.org)

Introdução

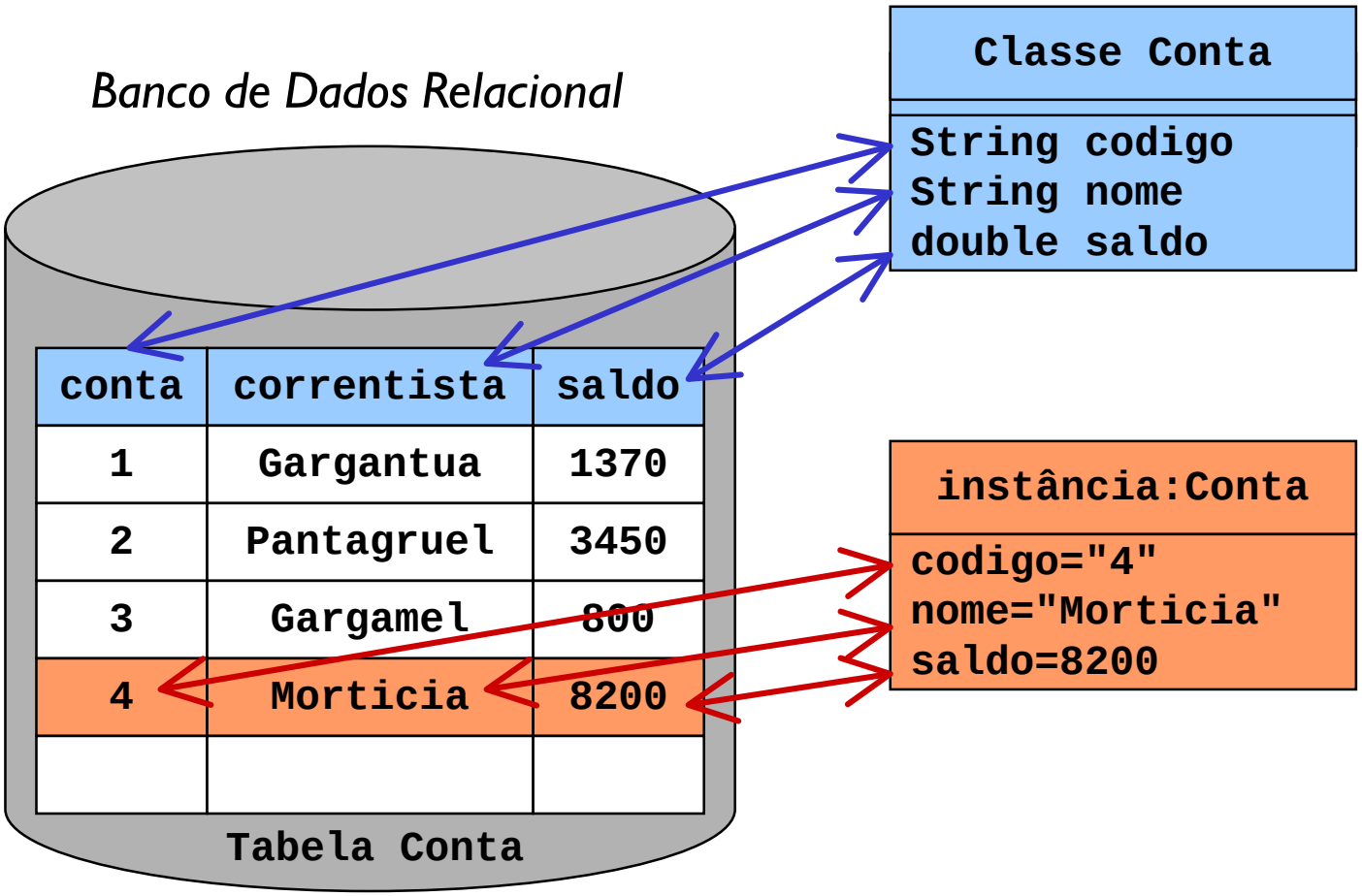
- *Entity beans (componentes de entidade) são objetos persistentes que podem ser armazenados em meio persistente de armazenamento*
  - *Pode-se fazer modelagem de dados com Entity Beans*
- *Tópicos deste módulo*
  - *Conceitos básicos de persistência*
  - *Definição de entity beans*
  - *Recursos*
  - *Conceitos de programação*
  - *Típos de entity beans*
  - *Exemplos de entity beans usando Bean-Managed Persistence*

Persistência

- *Uma maneira de armazenar o estado de objetos Java é através da serialização*
  - *Vantagem: facilidade de leitura e gravação: objeto pode ser enviado pela rede, armazenado em um banco ou árvore JNDI*
  - *Desvantagem: inviabilidade de realização de pesquisa sobre campos dos objetos: qualquer pesquisa necessitaria a desserialização de cada objeto para obter os campos pesquisados*
  - *Conclusão: para persistência de grande quantidade de dados que precisam de expor seus dados como atributos de pesquisa, serialização não é viável (solução seria serialização em XML!)*
- *Soluções alternativas*
  - *Bancos de dados relacionais: oferecem recursos poderosos e eficientes de busca, mas, não são orientados a objeto (SQLJ, JDBC)*
  - *Bancos de dados orientados a objeto*
  - *Mapeamento objeto-relacional (O/R Mapping)*

Object-Relational Mapping

- *Em vez de serializar os objetos, decompomos cada um em suas partes e guardamos suas partes no banco*



Fonte: [1]

O que é um Entity Bean?

- Session beans modelam a lógica relacionada a tarefas realizadas pela aplicação
- Entity beans modelam entidades
- São componentes que representam dados
  - Session beans podem guardar informações em bancos de dados mas não "representam" dados.
  - Entity beans são os dados
- Permitem que se manipule com objetos ignorando a forma de armazenamento
  - A especificação não define nenhum tipo obrigatório
- Duas partes
  - Entity bean instance: os dados na memória (representa uma instância da classe do Entity Bean)
  - Entity bean data: os dados fisicamente armazenados no banco

Partes de um entity bean

- Como qualquer EJB, um Entity Bean possui
  - Uma interface Home ou LocalHome
  - Uma interface Remote ou Local
  - Um deployment descriptor
  - Uma classe EJB
- As diferenças são
  - A classe do EntityBean está mapeada à uma definição de entidade de um esquema de banco de dados (uma tabela, por exemplo)
  - Há métodos especiais que lidam com a sincronização do EntityBean com o banco de dados
  - Um entity bean tem uma classe Primary Key que permite identificar univocamente sua instância

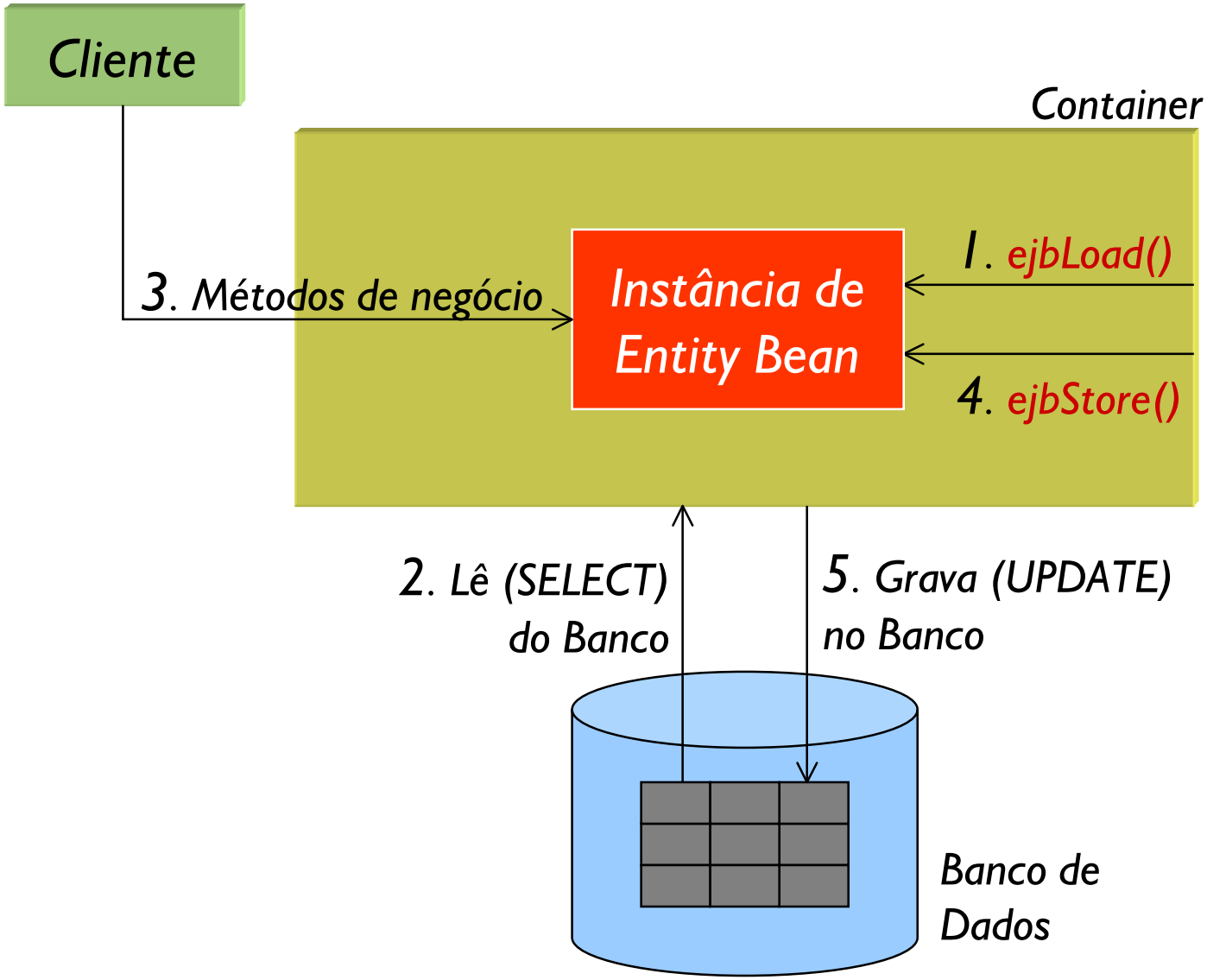
Entity Beans sobrevivem

- Entity Beans são objetos persistentes
  - Sobrevivem à quedas do servidor ou banco de dados
  - Por serem apenas representação de dados de um meio de armazenamento, o bean pode ser reconstruido lendo seus dados do banco.
- Ciclo de vida muito longo
  - Existem enquanto existir a tabela ou entidade que representam seus dados no banco
  - Muitas vezes já existem antes da construção da aplicação

Sincronismo

- Uma instância de um entity bean é uma visão do banco de dados
- A instância está sempre sincronizada com os dados do banco. Container possui mecanismo que faz a atualização após cada alteração
  - Todas as classes implementam métodos que realizam essa atualização
  - Container chama os métodos automaticamente
- void ejbLoad()
  - lê os dados do banco para dentro do entity bean
- void ejbStore()
  - grava os dados do entity bean no banco

ejbLoad() e ejbStore()



Fonte: [1]

Uso e reuso de Entity Beans

- *Instâncias diferentes de entity beans podem representar os mesmos dados*
  - *Transações garantem o isolamento necessário para evitar corrupção*
- *As mesmas instâncias podem ser reutilizadas para representar dados diferentes em momentos diferentes*
  - *Container gerencia ciclo de vida e ativação / passivação de Entity Beans, garantindo a sua disponibilidade*
  - *Programador pode controlar o que irá acontecer durante os métodos de sincronização ou pode deixar que o container se encarregue da tarefa.*

Duas formas de persistência

- Entity beans podem ser associados ao meio de armazenamento de duas formas
- **BMP** - Bean Managed Persistence
  - O bean deve usar uma API (JDBC, SQLJ) para fazer as chamadas de UPDATE no ejbStore(), INSERT no ejbCreate(), SELECT no ejbLoad(), etc. (caso use banco relacional)
- **CMP** - Container Managed Persistence
  - O container faz um mapeamento dos objetos a entidades em um banco
  - Métodos de ciclo de vida, campos de dados e métodos de negócio são gerados automaticamente pelo container

Criação e remoção de Entity Beans

- Como Entity Beans são os dados, criar um bean é inserir dados em um banco; remover um bean é apagar esses dados
- Entity Beans são criados através do EJBObject (proxy) via sua interface Home.
- Se você tiver o seguinte ejbCreate() no seu bean

```
public AccountPK ejbCreate(String id, String owner) {}
```

- Você deve ter o seguinte create() no seu Home

```
public Account create(String id, String owner)  {}
```

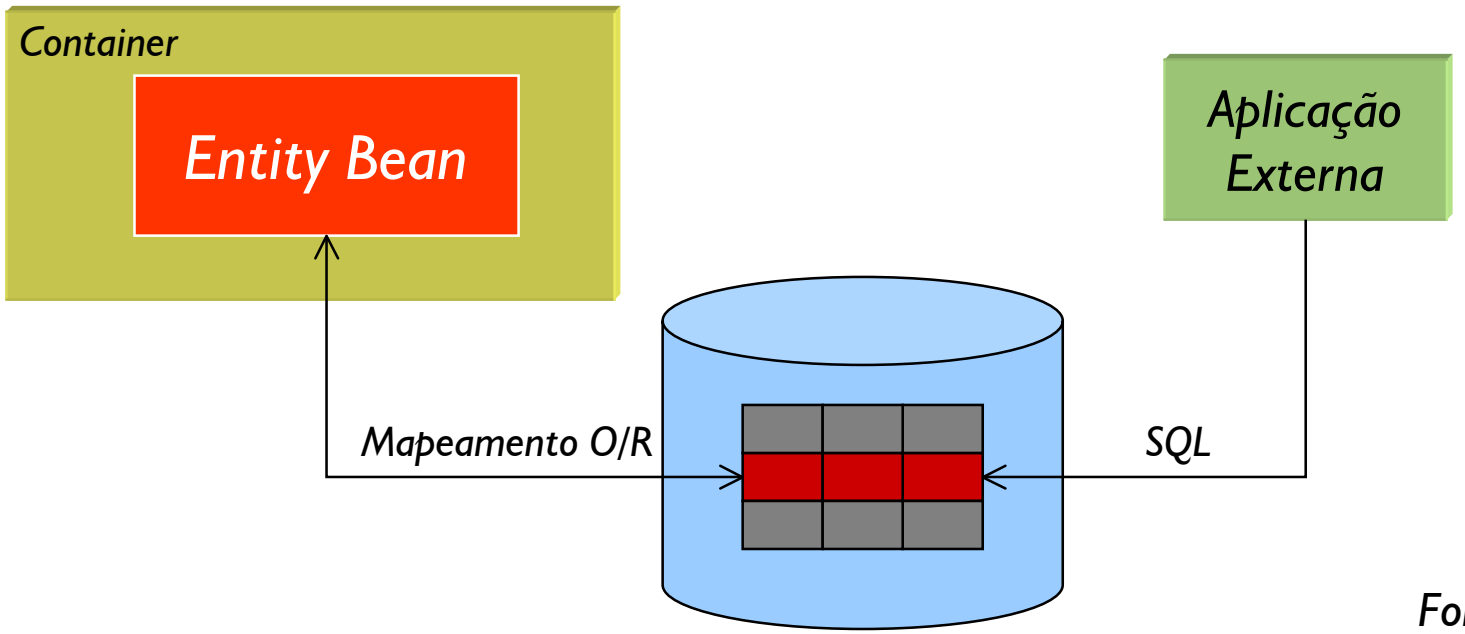
- Observe que os tipos retornados são diferentes
  - O bean retorna uma chave primária para o container (Home) que a utiliza para achar o bean para o cliente (que chamou create())
- Para destruir um bean, é preciso chamar remove() em Home
  - ejbRemove() deve localizar a chave primária do bean atual para removê-lo através o método getPrimaryKey() do contexto.

Entity Beans podem ser localizados

- Entity Beans nem sempre precisam ser criados. Podem já existir no banco
  - Neste caso, devem ser localizados
  - Todo bean pode ser localizado através de sua chave primária
  - Métodos find() adicionais podem ser criados para localizar beans de acordo com outros critérios
- Finder methods
  - No home: findXXX()
  - No EJB: **ejbF**indXXX()
  - Deve existir pelo menos o findByPrimaryKey()
  - Métodos create() são opcionais

Entity Beans podem ser modificados sem EJB

- Não é preciso usar EJB para modificar o conteúdo de um Entity Bean
  - Como Entity Beans representam dados externos, se esses dados puderem ser alterados externamente, sua mudança será refletida no Entity Bean
  - Riscos: cache de entity beans pode causar inconsistência (embora possível, a situação abaixo deve ser evitada)



Fonte: [1]

Entity Contexts

- Todo bean possui um objeto de contexto
- No caso do Entity Bean é o javax.ejb.EntityContext
- Através dele pode-se chamar os métodos
  - getEJBObject(): retorna uma referência para o EJBObject (interceptador, que implementa Remote)
  - getEJBLocalObject(): retorna uma referência local para o interceptador
  - getPrimaryKey(): retorna a chave primária associada com a entidade atual.
- getPrimaryKey() é essencial em ejbLoad() e ejbRemove() para descobrir qual a instância que será carregada ou removida, respectivamente

Métodos da interface Home em Entity Beans

- *Nos métodos abaixo, Componente pode ser a interface Remote (em EJBHome) ou Local (em EJBLocalHome) do bean*
  - *Nas interfaces Remote, todos provocam RemoteException também*
- **`Componente findByPrimaryKey(PK) throws FinderException`**
  - *Único método obrigatório de Home*
- **`Componente create(...) throws CreateException`**
  - *Retorna uma referência para o bean criado*
- **`Componente findXxx(...) throws FinderException`**
- **`Collection findXxx(...) throws FinderException`**
  - *Retorna referência para o componente ou Collection de referências caso o find retorne mais de um objeto*
- **`Tipo xxx(...)`**
  - *Método que não se refere a nenhuma instância particular*
  - *Deve ter um correspondente ejbHomeXxx() no bean com mesmos argumentos e tipo de retorno*

Métodos da especificação para Entity Beans

Métodos que implementam a interface Home

- **`PK ejbFindXxx(...) throws FinderException`**
  - *Deve haver pelo menos ejbFindByPrimaryKey(), que devolve a própria PrimaryKey (e faz um query de identidade para verificar se a PK existe no banco: SELECT pk where pk = pk)*
  - *Cada ejbFind() devolve ou a Primary Key ou uma Collection de PKs*
- **`PK ejbCreate(...) throws CreateException`**
  - *Pode haver zero ou mais (diferindo pelo número e tipo de args)*
  - *Deve criar o objeto (INSERT ou equivalente) e retornar seu PK*
- **`void ejbPostCreate(...)`**
  - *Deve haver um para cada ejbCreate() com mesmos argumentos*
  - *Chamado após o create e pode ser vazio*
- **`Tipo ejbHomeXxx(...)`**
  - *Métodos de Home que não se referem a nenhuma instância em particular. xxx() é o nome do método na interface Home.*

Relacionamento Home-Bean

- *Os métodos de EJBHome à esquerda delegam chamadas aos métodos de EntityBean à direita*
  - *Ou seja, se você tiver os métodos à esquerda no seu Home, deve ter os métodos à direita no seu Bean*

Interface Home	**EJB**
Objeto findByPrimaryKey(ObjetoPK)	ObjetoPK ejbFindByPrimaryKey(ObjetoPK)
Objeto findByParams(Par1, Par2)	ObjetoPK ejbFindByParams(Par1, Par2)
Collection findByParams(Par1, Par2)	Collection ejbFindByParams(Par1, Par2)
Objeto create(Par1, Par2)	ObjetoPK ejbCreate(Par1, Par2) void ejbPostCreate(Par1, Par2)
Tipo operacao(Par1, Par2)	Tipo ejbHomeOperacao(Par1, Par2)

- *Chave*
  - *Objeto: interface do objeto remoto*
  - *ObjetoPK: chave primária do objeto remoto*
  - *Par1, Par2 e Tipo: tipos primitivos, remotos ou serializáveis*

Métodos da interface EntityBean

Métodos de persistência e sincronização com o banco

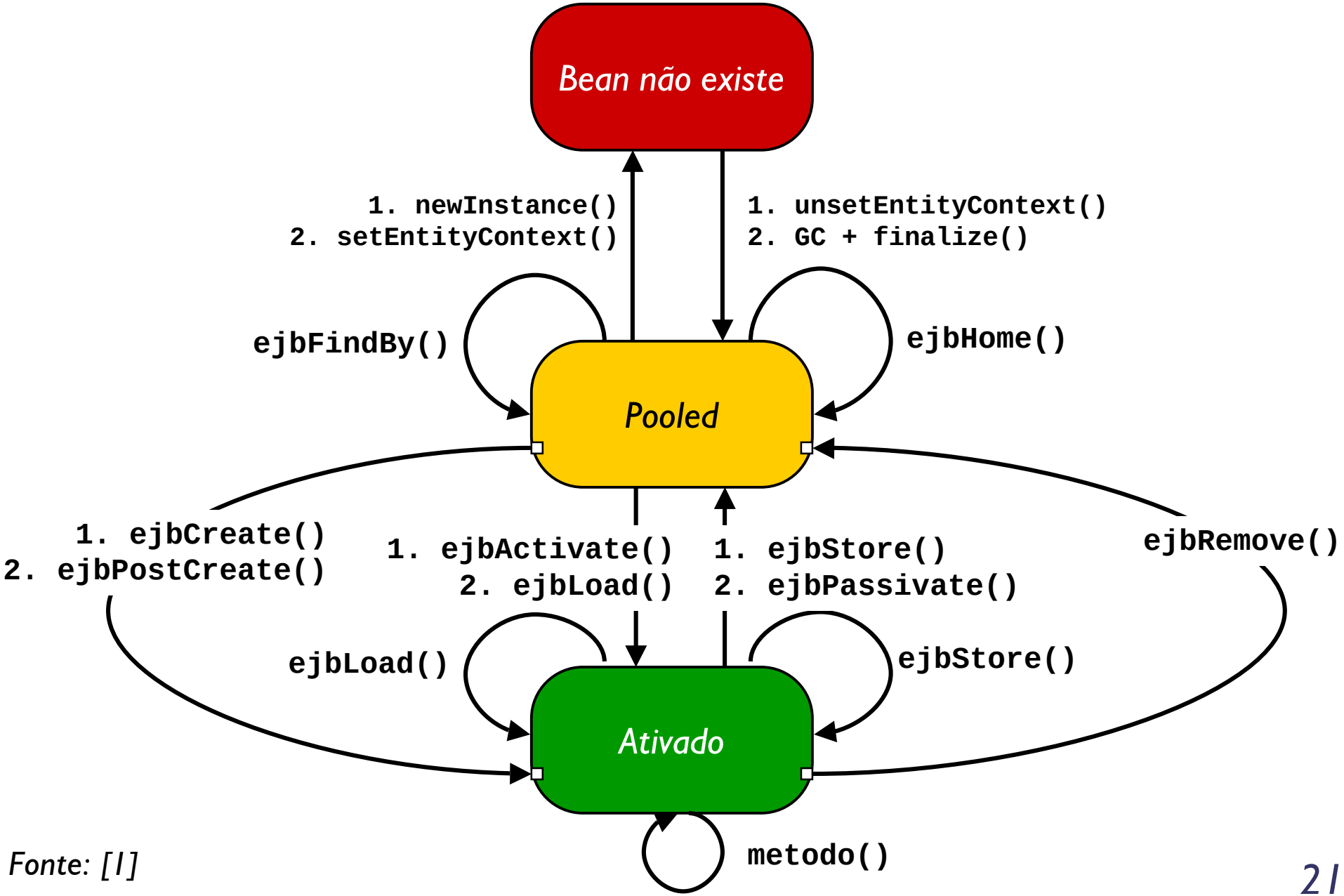
- **`void ejbLoad()`**
  - *Deve conter query SELECT ou equivalente e em seguida atualizar os atributos do bean com os dados recuperados*
  - *Use context.getPrimaryKey() para saber qual a chave primária do bean a ser lido*
- **`void ejbStore()`**
  - *Deve conter query UPDATE ou equivalente e gravar no banco o estado atual dos atributos do objeto*
- **`void ejbRemove()`**
  - *Chamado antes que os dados sejam removidos do banco*
  - *Deve conter query DELETE ou equivalente*
  - *Use context.getPrimaryKey() para saber qual a chave primária do bean a ser removido*

Métodos da interface EntityBean

Métodos do ciclo de vida das instâncias

- **`void ejbActivate()`**
  - *Chamado logo após a ativação do bean*
- **`void ejbPassivate()`**
  - *Chamado antes da passivação do bean*
- **`void setEntityContext(EntityContext ctx)`**
  - *Chamado após a criação da instância no pool. O contexto passado deve ser gravado em variável de instância pois pode ser usado para obter a chave primária da instância atual*
- **`void unsetEntityContext()`**
  - *Destrói a instância (será recolhida pelo GC). Isto destrói o objeto, mas não o entity bean (que são os dados)*

Ciclo de vida



Fonte: [1]

Exemplo: Interface do Componente Remote

```
public interface Account extends EJBObject {
  public void deposit(double amt)
              throws AccountException, RemoteException;
  public void withdraw(double amt)
              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: Interface Home

```
public interface AccountHome extends EJBHome {

   Account create(String accountID, String ownerName)
            throws CreateException, RemoteException;

   public Account findByPrimaryKey(AccountPK key)
            throws FinderException, RemoteException;

   public Collection findByOwnerName(String name)
            throws FinderException, RemoteException;

   public double getTotalBankValue()
            throws AccountException, RemoteException;

}
```

Exemplo: Primary Key

```
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) {
        return
          ( (AccountPK)account )
              .accountID.equals(accountID);
    }
}
```

Mesmo identificador usado no Bean (essencial, para CMP)

Exemplo: Bean

```
public class AccountBean implements EntityBean {
      protected EntityContext ctx;

      private String accountID;      // PK
      private String ownerName;
      private double balance;

      // Getters e Setters
      public String getAccountID() {...}  ...
      // Métodos de negócio
      public void deposit(double amount) {...}   ...

      // Métodos do ciclo de vida
      public void ejbActivate() {...}     ...
      // Métodos de Home
      public AccountPK findByPrimaryKey(AccountPK pk) {}
      // Métodos de persistência
      public void ejbLoad() {...} ...
}
```

Exemplo: Acesso ao banco via DataSource

```
private Connection getConnection() throws Exception {
  try {
    Context ctx = new InitialContext();
    javax.sql.DataSource ds =
      (javax.sql.DataSource)
          ctx.lookup("java:comp/env/jdbc/ejbPool");
    return ds.getConnection();
  } catch (Exception e) {
    System.err.println("Could not locate datasource:");
    e.printStackTrace();
    throw e;
  }
}
```

Veja o nome que foi usado no lookup e compare com o nome declarado como <resource-ref> no ejb.jar.xml e jboss.xml

Exemplo: Métodos do ciclo de vida

```
public void setEntityContext(EntityContext ctx) {
   System.out.println("setEntityContext called");
   this.ctx = ctx;
}

public void unsetEntityContext() {
   System.out.println("unsetEntityContext called");
   this.ctx = null;
}

public void ejbPassivate() {
   System.out.println("ejbPassivate () called.");
}

public void ejbActivate() {
   System.out.println("ejbActivate() called.");
}
```

Exemplo: Métodos de negócio e get/set

```
public void deposit(double amt) throws AccountException {
   balance += amt;
}

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

 // Getter/setter methods on Entity Bean fields
public double getBalance() {
   return balance;
}
public void setOwnerName(String name) {
   ownerName = name;
}
public String getOwnerName() {
   return ownerName;
}
...
```

Estes métodos são a razão de existir do bean! E são os mais simples!

Métodos de Home: findByPrimaryKey

```
public AccountPK ejbFindByPrimaryKey(AccountPK key)
                                    throws FinderException {
  PreparedStatement pstmt = null;
  Connection conn = null;
  try {
    conn = getConnection();
    pstmt = conn.prepareStatement
           ("select id from accounts where id = ?");
    pstmt.setString(1, key.toString());
    ResultSet rs = pstmt.executeQuery();
    rs.next();
    return key;
  } catch (Exception e) {
    throw new FinderException(e.toString());
  } finally { con.close(); ...   }
}
```

Query só para saber se PK existe no banco!

Métodos de Home: create()

```
public AccountPK ejbCreate(String accountID, String ownerName)
                                                  throws CreateException {
  PreparedStatement pstmt = null;
  Connection conn = null;
  try {
    this.accountID = accountID;
    this.ownerName = ownerName;
    this.balance = 0;
    conn = getConnection();
    pstmt = conn.prepareStatement("insert into accounts "+
                  "(id, ownerName, balance) values (?, ?, ?)");
    pstmt.setString(1, accountID);
    pstmt.setString(2, ownerName);
    pstmt.setDouble(3, balance);
    pstmt.executeUpdate();
    return new AccountPK(accountID);
  } catch (Exception e) {
     throw new CreateException(e.toString());
  } finally { con.close(); ...   }
}
public void ejbPostCreate(String accountID, String ownerName) {
}
```

Não esqueça de implementar ejbPostCreate()

Métodos de Home: findByOwnerName()

```
public Collection ejbFindByOwnerName(String name)
                                        throws FinderException {
  PreparedStatement pstmt = null;
  Connection conn = null;
  Vector v = new Vector();

  try {
     conn = getConnection();
     pstmt = conn.prepareStatement
             ("select id from accounts where ownerName = ?");
     pstmt.setString(1, name);
     ResultSet rs = pstmt.executeQuery();

     while (rs.next()) {
       String id = rs.getString("id");
       v.addElement(new AccountPK(id));
     }
     return v;
  } catch (Exception e) {
     throw new FinderException(e.toString());
  } finally { con.close(); ...   }
}
```

Métodos de Home: getTotalBankValue()

```
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 {...}
   throw new AccountException("Error!");
}
```

Exemplo: ejbRemove()

```
public void ejbRemove() throws RemoveException
   AccountPK pk = (AccountPK) ctx.getPrimaryKey();
   String id = pk.accountID;

   PreparedStatement pstmt = null;
   Connection conn = null;
   try {
     conn = getConnection();
     pstmt = conn.prepareStatement
                    ("delete from accounts where id = ?");
     pstmt.setString(1, id);
     if (pstmt.executeUpdate() == 0) {
       throw new RemoveException("...");
     }
   } catch (Exception ex) {
     throw new EJBException("...", ex);
   } finally {...conn.close() ... }
}
```

Exemplo: ejbStore()

```
public void ejbStore() {
   PreparedStatement pstmt = null;
   Connection conn = null;
   try {
      conn = getConnection();
      pstmt = conn.prepareStatement
        ("update accounts set ownerName = ?,
                              balance = ? where id = ?");
      pstmt.setString(1, ownerName);
      pstmt.setDouble(2, balance);
      pstmt.setString(3, accountID);
      pstmt.executeUpdate();

   } catch (Exception ex) {
      throw new EJBException("...", ex);
   } finally {...conn.close() ... }
}
```

Sempre provoque EJBException quando algo sair errado em ejbStore() e ejbLoad()

Exemplo: ejbLoad()

```
public void ejbLoad() {
   AccountPK pk = (AccountPK) ctx.getPrimaryKey();
   accountID = pk.accountID;
   PreparedStatement pstmt = null;
   Connection conn = null;
   try {
     conn = getConnection();
     pstmt = conn.prepareStatement
      ("select ownerName, balance from accounts where id = ?");
     pstmt.setString(1, accountID );
     ResultSet rs = pstmt.executeQuery();
     rs.next();
     ownerName = rs.getString("ownerName");
     balance = rs.getDouble("balance");
   } catch (Exception ex) {
     throw new EJBException("...", ex);
   } finally {...conn.close() ... }
}
```

Em BMP, use um DAO

- Nos exemplos anteriores, o código JDBC foi usado dentro dos métodos de persistência do bean
  - Esta prática é razoável como exemplo, mas não deve ser usada em produção
  - Dificulta a manutenção e mistura lógica do banco com a lógica de negócios do bean
- Use um DAO - Data Access Object
  - Com um DAO, cada chamada a método de persistência é delegada a um método de um DAO, que realiza as operações desejadas
  - O DAO pode encapsular toda a lógica de acesso e pode até usar outro mecanismo de persistência sem quebrar a aplicação.

Exemplo: Deployment Descriptor

```
<ejb-jar>
    <enterprise-beans>
        <entity>
            <ejb-name>Account</ejb-name>
            <home>examples.AccountHome</home>
            <remote>examples.Account</remote>
            <local-home>examples.AccountLocalHome</local-home>
            <local>examples.AccountLocal</local>
            <ejb-class>examples.AccountBean</ejb-class>
            <persistence-type>Bean</persistence-type>
            <prim-key-class>examples.AccountPK</prim-key-class>
            <reentrant>False</reentrant>
            <resource-ref>
                <res-ref-name>jdbc/ejbPool</res-ref-name>
                <res-type>javax.sql.DataSource</res-type>
                <res-auth>Container</res-auth>
            </resource-ref>
        </entity>
    </enterprise-beans>
    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>Account</ejb-name>
                <method-name>*</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>
    </assembly-descriptor>
</ejb-jar>
```

Este é o nome usado no ENC deste bean para a fonte de dados JDBC. O jboss.xml faz o mapeamento a uma fonte de dados local

Fonte de dados onde bean está armazenado

Declaração da política de transações é obrigatória em Entity Beans pois SEMPRE usam transações controladas pelo container

Exemplo: jboss.xml

```
<jboss>
  <enterprise-beans>
    <entity>
      <ejb-name>Account</ejb-name>
      <jndi-name>account/AccountHome</jndi-name>
    </entity>
  </enterprise-beans>
  <resource-managers>
    <resource-manager>
        <res-name>jdbc/ejbPool</res-name>
        <res-jndi-name>java:/DefaultDS</res-jndi-name>
    </resource-manager>
  </resource-managers>
</jboss>
```

Chamado por clientes dentro e fora do ENC

Veja chamada no bean

Execução

- Para executar a aplicação exemplo, use o Ant no diretório cap07/exemplos
  - Configure o build.properties se necessário
- Etapas
  - Povoar banco de dados: `ant create.table`
  - Deployment: `ant jboss.deploy`
  - Rodar o cliente: `ant run.jboss.client`
- Manhenha a janela do JBoss visível para ver simultaneamente as chamadas no cliente e servidor

Exercício

- 1. Implementar EntityBean que represente uma tabela com o seguinte esquema

```
create table produtos (
    id varchar(16) primary key,
    nome varchar(32),
    preco numeric(18),
    qte integer
);
```

  - a) Implemente, no pacote loja, as classes Produto, ProdutoPK, ProdutoHome, ProdutoBean (algumas classes já estão parcialmente prontas). Use o ProdutoDAO e ProdutoVO fornecidos para implementar métodos de ProdutoBean que acessam o banco
  - b) Implemente o método create() e os finders definidos na interface Home (use os métodos do DAO)
  - c) Configure a chamada JNDI no DAO para que localize o DataSource usando o nome jdbc/produtos no ENC
  - d) Preencha os tags em jboss.xml e ejb-jar.xml
  - e) Depois do deploy, termine o cliente e teste a aplicação

Fontes

[1] *Ed Roman, Mastering EJB 2, 2002. Fonte da maior parte dos diagramas usados e do exemplo (modificado).*

[2] *Dale Green. Bean-Managed Persistence Examples. Sun J2EE Tutorial, 2002. Fonte dos exemplos (modificados para rodar no JBoss)*

[3] *Linda de Michiel et al. Enterprise JavaBeans 2.1 Specification. Sun Microsystems, 2003.*

Curso J530: Enterprise JavaBeans

Revisão 2.0 - Junho de 2003

(helder@acm.org)

argonavis.com.br