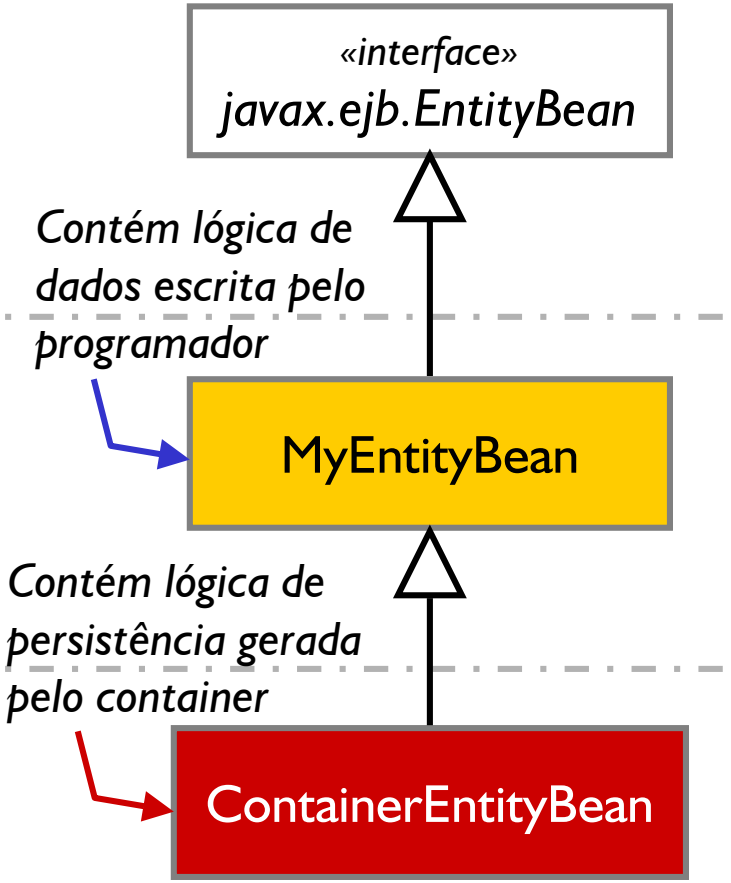
J530 - Enterprise JavaBeans

Entity Beans & com persistência implícita (CMP)

Helder da Rocha (helder@acm.org)

Características de EJB CMP

- Com container-managed persistence o programador do bean não implementa lógica de persistência
  - O bean não precisa usar APIs como JDBC
  - O container se encarrega de criar as tabelas (no deployment), criar os beans e os dados, removê-los e alterá-los
- O container gera automaticamente o JDBC ao estender a classe do bean criada pelo programador
  - O entity bean verdadeiro é o que foi gerado: a combinação da classe implementada pelo programador e a gerada pelo container



Entity Beans CMP não têm ...

- ... campos de dados declarados
  - São implementados na subclasse* (criada pelo container)
  - Programador declara seus campos no deployment descriptor (abstract persistence schema) usando <cmp-field>
- ... implementação de métodos de acesso
  - Programador declara todos os seus métodos de acesso (get/set) como abstract
  - Os nomes dos métodos devem ser compatíveis com os nomes dos campos declarados:

    Se há `getCoisa()` no bean, e no deployment descriptor deve haver:
    `<cmp-field><field-name>coisa</field-name></cmp-field>`

  - Subclasse gerada pelo container implementa os métodos (o container pode usar BMP, por exemplo)

EJB-QL

- Como não há como fazer queries JDBC com entity beans CMP, há uma linguagem para fazer queries nos próprios beans: Enterprise JavaBean Query Language (EJB-QL)
- EJB-QL é uma linguagem com sintaxe semelhante a SQL para pesquisar entity beans. Possui
  - cláusula SELECT
  - cláusula FROM
  - cláusula WHERE, opcional
- EJB-QL é colocado nos deployment descriptors no contexto da declaração do método
  - Todos os finders são implementados com EJB-QL
- O servidor transforma expressões EJB-QL em SQL e realiza as chamadas no banco de dados
  - Isto pode ser configurado no servidor usando recursos proprietários

Métodos de Entity Beans que usam CMP

- **`setEntityContext()` / `unsetEntityContext()`**
  - *Mesmo comportamento de entity beans com BMP*
- **`ejbFindXXX(...)`**
  - *Não são implementados no bean em CMP*
  - *Declarados no deployment descriptor (DD) e têm lógica em EJB-QL*
- **`ejbSelectXXX(...)`**
  - *Usados para fazer queries genéricos para o bean*
  - *É método de negócio mas não pode ser chamado pelo cliente*
  - *Declarados como abstract com lógica descrita no DD em EJB-QL*
- **`ejbHomeXXX(...)`**
  - *Usado para definir métodos que não são específicos a uma instância*
  - *Deve chamar um ou mais ejbSelect() para realizar as operações*
- **`ejbCreate(...)` / `ejbPostCreate(...)`**
  - *Chama os métodos de acesso abstratos inicializando-os com os dados que serão usados pelo container para criar novo registro*
  - *Deve retornar null*

Métodos (2)

- **`ejbActivate()` / `ejbPassivate()`**
  - *Mesmo comportamento que BMP*
- **`ejbLoad()`**
  - *Chamado ao carregar dados do banco para o bean*
  - *Pode ser vazio ou conter rotinas para transformar dados recebidos*
- **`ejbStore()`**
  - *Chamado ao atualizar o banco com alterações no bean*
  - *Pode ser vazio ou conter rotinas p/ transformar dados antes do envio*
- **`ejbRemove()`**
  - *Chamado por home.remove(), remove os dados do banco (não remove a instância - que pode ser reutilizada)*
  - *Pode ser vazio ou conter código para ser executado antes da remoção dos dados*

Diferenças BMP-CMP

Diferença	Container-Managed Persistence	Bean-Managed Persistence
Definição da classe	*Abstrata*	*Concreta*
Chamadas de acesso ao banco de dados	*Gerada pelas ferramentas no deployment*	*Codificada pelo programador*
Estado persistente	*Representadas como campos persistentes virtuais*	*Codificadas como variáveis de instância*
Métodos de acesso a campos persistentes e relacionamentos	*Obrigatórios (abstract)*	*Não há*
Método findByPrimaryKey	*Gerado pelo container*	*Codificado pelo programador*
Métodos finder customizados	*Gerados pelo container mas programador escreve EJB-QL*	*Codificado pelo programador*
Métodos select	*Gerados pelo container*	*Não há*
Valor de retorno de ejbCreate()	*Deve ser null*	*Deve ser a chave primária*

Fonte: J2EE Tutorial

Exemplo: Como executar

- Diretório *cap08/exemplos/mejb/*
- Configuração
  - *Configure o arquivo build.properties com seu ambiente*
  - *Inicie o JBoss*
- Inicialização

```
> ant drop.table
> ant clean
```

- Deployment

```
> ant jboss.deploy
```
(observe as mensagens de criação das tabelas no JBoss)

- Execução do cliente

```
> ant run jboss.client
> ant select.all
```
(faz SELECT na tabela)

Componentes de um EJB-JAR que usa CMP

- Interfaces Home e Component e classe Primary Key
  - Idênticas às interfaces e classe Primary Key de um entity bean equivalente que usa BMP
  - PK não pode conter campos de relacionamento (CMR)
- Classe Enterprise bean (muito menor que classe BMP)
  - Métodos ejbLoad(), ejbStore() não possuem código de persistência
  - Classe é abstrata; métodos de acesso são abstratos; métodos ejbSelectXXX() são abstratos
  - Métodos de negócio contém apenas lógica de negócio
  - Métodos finder não são declarados
- Deployment descriptor (maior que versão BMP)
  - Definição da lógica de métodos e relacionamentos
- Arquivos de configuração do fabricante (JBoss)
  - Mapeamento de tipos, mapeamento de esquema

Enterprise Bean

```
public abstract class ProdutoBean implements EntityBean {
  protected EntityContext ctx;

  public abstract String getName();
  public abstract void setName(String name);
  public abstract String getCodigo();
  public abstract void setCodigo(String codigo);
    (...)
  public void ejbStore() {}
  public void ejbLoad() {}
  public void ejbPassivate() {}
  public void ejbRemove() {}
  public void ejbActivate() {}
  public void setEntityContext(EntityContext c) {ctx = c;}
  public void unsetEntityContext() {ctx = null;}
  public void ejbPostCreate(String codigo, String name,...) {}
  public void ejbCreate(String codigo, String name, ...) {
    setCodigo(codigo);
    setName(name);
    (...)
  }
}
```

Métodos de negócio abstratos (lógica no deployment descriptor)

Não expor na interface

ejbLoad *e* ***ejbStore*** *são implementados pelo container*

ejbFindBy *não são declarados (lógica está no deployment descriptor)*

ejbCreate *chama métodos abstratos para passar dados recebidos*

Deployment Descriptor (I)

```
<ejb-jar>
  <enterprise-beans>
    <entity>
      <ejb-name>ProdutoEJB</ejb-name>
      <home>loja.ProdutoHome</home>
      <remote>loja.Produto</remote>
      <ejb-class>loja.ProdutoBean</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>loja.ProdutoPK</prim-key-class>

      <reentrant>False</reentrant>

      <cmp-version>2.x</cmp-version>
      <abstract-schema-name>
         Produto
      </abstract-schema-name>

      ...
```

Deployment Descriptor (2)

```
...
<cmp-field>
  <field-name>codigo</field-name>
</cmp-field>
<cmp-field><field-name>name</field-name></cmp-field>
<cmp-field>
  <field-name>preco</field-name>
</cmp-field>
<cmp-field>
  <field-name>quantidade</field-name>
</cmp-field>
<!--
<primkey-field>codigo</primkey-field>
-->
<query>
  ... declações EJB-QL
<query>
</entity>
</enterprise-beans>
...
```

Se PK for um campo unidimensional em vez de uma classe (String, por exemplo) use este elemento e indique o seu tipo no <prim-key-class> (java.lang.String, por exemplo)

Primary Key

- A Primary Key de um bean CMP, se implementada com uma classe, deve ter campos de dados públicos com os mesmos identificadores que os campos persistentes que a compõem
  - Por exemplo, se uma PK simples é representada pelo campo codigo, deve ter uma variável de instância codigo e ela deve ser pública
  - Se for uma PK composta, cada um dos campos que a representa deve ter o mesmo identificador que seus atributos públicos
  - Se esta regra não for observada, o container não terá como descobrir quem é a Primary Key do objeto
- Os métodos que alteram atributos pertencentes a uma PK não devem ser expostos através das interfaces e só devem ser chamados uma vez dentro do bean
  - Para os atributos do PK, bean deve ter par get/set para cada atributo, mas interface Local ou Remote só deve ter o método get().

EJB-QL

- Toda pesquisa no banco é feita através de Enterprise JavaBeans Query Language (EJB-QL)
- O servidor mapeia expressões EJB-QL a expressões SQL para realizar as pesquisas no banco
- EJB-QL é usado no deployment descriptor
  - Não há SQL ou EJB-QL na classe do bean. O bean lida apenas com lógica de negócio
  - O bean pode usar métodos que retornam queries customizados, previamente definidos no deployment descriptor em EJB-QL

Exemplos de EJB-QL

O que vai ser selecionado

Abstract schema name

Declara variável p como sendo um Produto

```
SELECT p.codigo
  FROM Produto AS p
 WHERE p.codigo = '123'
```

Parâmetro

Objeto (bean) sendo selecionado

AS *é opcional*

Parâmetro do método

```
SELECT OBJECT(p)
  FROM Produto p
 WHERE p.nome = ?1
```

Palavras-chave EJB-QL

- Usadas na cláusula SELECT
  - DISTINCT, OBJECT
- Usadas na cláusula FROM
  - AS, IN
- Usadas na cláusula WHERE
  - AND, OR, NOT
  - BETWEEN
  - LIKE, IS, MEMBER, OF
  - TRUE, FALSE
  - EMPTY, NULL
  - UNKNOWN

Cláusula FROM

- A cláusula **FROM** é quem informa o esquema abstrato que está sendo pesquisado, e declara variáveis usados no resto do query
  - Cada variável tem um identificador e um tipo
  - O identificador é qualquer palavra não-reservada
  - O tipo faz parte do esquema abstrato do bean (campos de dados e abstract schema name)
  - A palavra-chave AS conecta o tipo ao identificador, mas é opcional

Cláusula SELECT

- A cláusula **SELECT** informa o que se deseja obter com a pesquisa. Pode retornar
  - Objetos (interfaces locais ou remotas)
  - Atributos dos objetos (tipos dos campos <cmp-field>)
- A palavra SELECT pode ser seguida de DISTINCT para eliminar valores duplicados nos resultados
- SELECT utiliza uma variável declarada na cláusula FROM
- Se retorna um objeto, é preciso usar OBJECT(variavel)
- Exemplos

```
SELECT OBJECT(p) FROM Produto p
SELECT DISTINCT OBJECT(p) FROM Produto p
SELECT p.codigo FROM Produto p
SELECT DISTINCT p.codigo FROM Produto p
```

Cláusula WHERE

- A cláusula **WHERE** é opcional e restringe os resultados da pesquisa com base em uma ou mais expressões condicionais concatenadas
- As expressões podem usar
  - Literais (strings, booleanos ou números)
  - Identificadores (declarados no FROM)
  - Operadores
  - Funções
  - Parâmetros do método que utiliza a pesquisa (**?1**, **?2**,...)
- Literais
  - Strings são representados entre apóstrofes: 'nome'
  - Números têm mesmas regras de literais Java long e double
  - Booleanos são TRUE e FALSE (case-insensitive)

Operadores EJB-QL

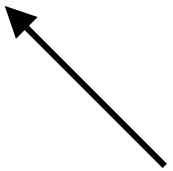
- *Expressões matemáticas*
  - `+, -, *, /`
- *Expressões de comparação*
  - `=, >, >=, <, <=, <>`
- *Operadores lógicos*
  - `NOT, AND, OR`
- *Outros operadores*
  - `BETWEEN, NOT BETWEEN`
  - `IN, NOT IN`
  - `LIKE, NOT LIKE`
  - `NULL, NOT NULL`
  - `IS EMPTY, IS NOT EMPTY`
  - `MEMBER, NOT MEMBER`
- *Operadores do LIKE*
  - `_` *representa um único caractere*
  - `%` *representa uma seqüência de zero ou mais caracteres*
  - `\` *caractere de escape (necessário para usar _ ou %) literalmente*

Funções EJB-QL

- **CONCAT (String, String)**
  - *Retorna String. Concatena dois strings*
- **SUBSTRING(String, int start, int length)**
  - *Retorna String. Parte um string em um string menor*
- **LOCATE(String, String, [start])**
  - *Retorna um inteiro mostrando onde um string está localizado dentro de outro String*
- **LENGTH(String)**
  - *Retorna um inteiro com o comprimento do String*
- **ABS(numero)**
  - *Retorna o valor absoluto de um número (int, double)*
- **SQRT(double numero)**
  - *Retorna a raiz quadrada de um número (double)*
- **MOD(int, int)**
  - *Retorna inteiro com o resto da divisão*

Não existem em EJB-QL 2.0

- Várias funções e tipos comuns em SQL não existem em EJB-QL 2.0. Para obter o mesmo efeito, é preciso às vezes recorrer à combinação de métodos ejbSelect() ou mesmo a BMP
  - Em último caso, pode-se recorrer a extensões proprietárias sacrificando-se a portabilidade
- Funções ausentes
  - ORDER BY
  - MIN, MAX
  - COUNT
  - SUM
- Tipos ausentes
  - DATE (use long para guardar datas)



Se puder, use um servidor que suporte EJB 2.1 para não comprometer a independência de fabricante

Exemplos de EJB-QL (de [1])

- *Encontre todos os produtos que são chips e cuja margem de lucro é positiva*
  - ```
    SELECT OBJECT(p)
    FROM Produto p
    WHERE (p.descricao = 'chip'
    AND (p.preco - p.custo > 0)
    ```
- *Encontre todos os produtos cujo preço é pelo menos 1000 e no máximo 2000*
  - ```
    SELECT OBJECT(p)
    FROM Produto p
    WHERE p.preco BETWEEN 1000 AND 2000
    ```
- *Encontre todos os produtos cujo fabricante é Sun ou Intel*
  - ```
    SELECT OBJECT(p)
    FROM Produto p
    WHERE p.fabricante IN ('Intel', 'Sun')
    ```
    ```

Exemplos de EJB-QL (2)

- *Encontre todos os produtos com IDs que começam com 12 e terminam em 3*
  - ```
    SELECT OBJECT(p)
    FROM Produto p
    WHERE p.id LIKE '12%3'
    ```
- *Encontre todos os produtos que têm descrições null*
  - ```
    SELECT OBJECT(p)
    FROM Produto p
    WHERE p.descricao IS NULL
    ```
- *Encontre todos os pedidos que não têm itens (coleção)*
  - ```
    SELECT OBJECT(pedido) FROM Pedido pedido
    WHERE pedido.itens IS EMPTY
    ```
- *Encontre todos os itens ligados a pedidos em coleção*
  - ```
    SELECT OBJECT(item)
    FROM Pedido pedido, Item item
    WHERE item IS MEMBER pedido.itens
    ```

<query> EJB-QL

```
<query>
  <query-method>
    <method-name>findByName</method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
    </method-params>
  </query-method>
  <ejb-ql><![CDATA[ SELECT OBJECT(obj)
  FROM ProdutoBean obj WHERE obj.name = ?1 ]]></ejb-ql>
</query>
      ...
<query>
  <query-method>
    <method-name>findAllProducts</method-name>
  </query-method>
  <ejb-ql><![CDATA[
   SELECT OBJECT(obj) FROM ProdutoBean AS obj
          WHERE obj.codigo IS NOT NULL
  ]]></ejb-ql>
</query>
```

Nome de método de Home ou Remote

Parâmetro recebido

abstract schema name

Nome de um cmp-field

AS é opcional

findByPrimaryKey não deve ser declarado!

JBoss deployment descriptor

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS 3.0//EN"
                       "http://localhost/dtd/jboss_3_0.dtd">

<jboss>
  <enterprise-beans>
    <entity>
      <ejb-name>ProdutoEJB</ejb-name>
      <jndi-name>loja/produtos</jndi-name>
    </entity>
  </enterprise-beans>
  <resource-managers>
    <resource-manager>
        <res-name>jdbc/LojaDB</res-name>
        <res-jndi-name>java:/DefaultDS</res-jndi-name>
    </resource-manager>
  </resource-managers>
</jboss>
```

Nome JNDI Global

Nome do ENC java:comp/env

Nome JNDI global domínio java: local

JBoss CMP: jbosscmp-jdbc.xml

- *Este arquivo é necessário se a aplicação irá se comunicar com fonte de dados que tem esquema próprio*

```
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <ejb-name>ProdutoEJB</ejb-name>
         <table-name>produtos</table-name>
         <cmp-field>
            <field-name>name</field-name>
            <column-name>nome</column-name>
         </cmp-field>
         <cmp-field>
            <field-name>codigo</field-name>
            <column-name>id</column-name>
         </cmp-field>
         <cmp-field>
            <field-name>quantidade</field-name>
            <column-name>qte</column-name>
         </cmp-field>
            ...
      </entity>
   </enterprise-beans>
</jbosscmp-jdbc>
```

Tabela que será usada (ou criada) no banco

Campos CMP

Campos do banco de dados

Pode-se ainda redefinir o mapeamento de tipos para todos os beans no arquivo de configuração standardjbosscmp-jdbc.xml

ejbSelect()

- Método de pesquisa, como um finder, mas que não é exposto ao cliente através da interface Home
  - É privativo ao bean: usado internamente como um método auxiliar na realização de pesquisas genéricas
  - É declarado no bean como abstract (para que possa ser usado pelos outros métodos) e provoca FinderException
- Com ejbSelect() é possível realizar pesquisas e devolver objetos e também valores não relacionados com objetos
  - Finders são limitados à seleção de objetos (beans) e coleções de objetos
  - Seletores podem retornar qualquer coisa

ejbHome()

- *Para implementar métodos ejbHome() em CMP é preciso recorrer a métodos ejbSelect()*
  - *Use métodos ejbSelect() dentro de ejbHome() para selecionar dados e devolver os resultados desejados*
- *Exemplo: getProximoCodigo() em Home, é implementado no bean como*

```
public String ejbHomeGetProximoCodigo() {
    Collection codigos = this.ejbSelectCodigos();
    Iterator it = codigos.iterator();
    int max = 0;
    while(it.hasNext()) {
        int cod = Integer.parseInt((String)it.next());
        if (cod > max) max = cod;
    }
    return "" + (max+1);
}
```

Exemplo de ejbSelect()

- *No bean, ejbSelect() são declarados para que métodos ejbHome() possam usá-los*

```
public abstract Collection ejbSelectCodigos();
```

- *No deployment descriptor, são definidos em EJB-QL*

```
<query>
  <query-method>
    <method-name>ejbSelectCodigos</method-name>
    <method-params/>
  </query-method>
  <ejb-ql><![CDATA[
   SELECT obj.codigo FROM ProdutoBean obj
  ]]></ejb-ql>
</query>
```

Resumo

- *Beans CMP são classes abstratas*
- *Atributos de dados não são declarados na classe do bean mas no DD como parte do esquema de dados do bean*
- *Métodos get/set têm que combinar com atributos declarados e devem ser abstract no bean*
- *Nenhum finder é implementado no bean. Todos são declarados no DD com parâmetros e lógica descritos em EJB-QL que usa os componentes do esquema de dados*
- *findByPrimaryKey() só é declarado na interface Home*
- *Métodos ejbHome() não são declarados no DD, mas usam métodos privativos ejbSelect(), abstratos no bean, que não aparecem no Home mas são especificados em EJB-QL para realizar suas tarefas.*

Conclusões

- *CMP é uma solução que permite a criação de componentes que contém 100% de lógica de negócio*
  - *Implementação de persistência é resolvida pelo container*
  - *Desenvolvedor se concentra na modelagem e lógica de negócios e não se preocupa com os detalhes da persistência*
- *Ainda não é possível, sem recorrer a extensões proprietárias, implementar qualquer modelo entidade-relacionamento usando CMP*
  - *Ideal é quando o modelo de objetos determina a estrutura e relacionamentos entre as tabelas. É mais difícil implementar CMP em sistemas legados, com tabelas gigantes, mal-estruturadas, etc.*
  - *Pode-se usar BMP em parte da aplicação já que cada bean pode decidir se terá a persistência controlada pelo container ou não*
- *CMP facilita e flexibiliza relacionamentos entre objetos!*
  - *Isto é um tema que será explorado mais adiante*

Exercícios Teóricos

- *1. Escreva queries CMP para as seguintes situações (invente referências e nomes significativos)*
  - *a) Localizar todas as contas com saldo maior que zero*
  - *b) Localizar todas as contas cujos identificadores começam com '0000-'*
  - *c) Localizar as contas cujos saldos sejam menores que 100 ou maiores que 10000*
  - *d) Localizar produtos que custam menos que 1000 e que tenham pelo menos duas unidades em estoque*
- *Queries de relacionamento (extras)*
  - *e) Localize todos os clientes que têm pelo menos um pedido na sua coleção de pedidos (não está vazio)*
  - *f) Localize todos os pedidos que façam parte da coleção de pedidos do cliente cujo userid é 'genghis'*

Exercícios práticos

- 2. Implemente o exemplo mostrado no capítulo anterior (Account) em CMP
  - Faça o deployment
  - Execute usando o mesmo cliente
- 3. Adapte o exercício do capítulo anterior (Produto) para CMP
  - Faça o deployment
  - Execute usando o mesmo cliente
  - Implemente queries sofisticados para pesquisar produtos (por preço, por nome, por quantidade, etc.)

Fontes

[1] *Ed Roman. Mastering EJB 2. 2002*

[2] *Richard Monson-Haefel. Enterprise JavaBeans, 3rd. Edition. O'Reilly, 2002*

[3] *Dale Green, J2EE Tutorial - Container Managed Persistence Examples. Sun Microsystems, 2002*

Curso J530: Enterprise JavaBeans

Revisão 2.0 - Junho de 2003

(helder@acm.org)

argonavis.com.br