



Enterprise Java Beans

Java Enterprise Edition

Prof. Fellipe Aleixo (fellipe.aleixo@ifrn.edu.br)

Definição de Enterprise JavaBeans

- “A arquitetura *Enterprise JavaBeans* é uma **arquitetura de componentes** para o desenvolvimento e a implantação de aplicativos de negócio **distribuídos** baseados em componentes. Aplicativos escritos utilizando a arquitetura *Enterprise JavaBeans* são **escalonáveis, transacionais e seguros** com **multiusuários**. Esses aplicativos podem ser escritos uma vez e então implantados em qualquer plataforma de servidor que suporta a especificação *Enterprise JavaBeans*”
Sun Microsystems

Definição de Enterprise JavaBeans

- Definição simplificada: “Modelo padrão de componentes do lado do servidor para aplicativos de negócio distribuídos”
 - Componentes executando processos do negócio
- Arquitetura distribuída:
 - Distribuição de carga de processamento
 - Maximização de confiabilidade

Persistência e Beans de Entidade

- No EJB 3.0 a persistência deixou de fazer parte da plataforma, passando a ser especificada a parte – ***Java Persistence API*** – JPA
- JPA é uma abstração superior à API JDBC
 - Os objetos são mapeados em tabelas de banco dados. Estes podem ser consultados, carregados, atualizados ou removidos sem que necessário utilizar a API JDBC
 - Mapeia objetos Java simples e comuns (POJOs – *plain old Java objects*) – Beans de Entidade

Sistema de Mensagens Assíncronas

- Além do suporte à comunicação distribuída, baseada em RMI, o EJB suporta também um sistema de mensagens assíncronas
 - Uma mensagem é um pacote autocontido de dados do negócio e cabeçalhos de roteamento de rede
 - Mensagens assíncronas podem ser transmitidas entre um aplicativo e outro em uma rede, utilizando o Message-Oriented Middleware – MOM
 - O MOM assegura tolerância a falhas, escalabilidade, balanceamento de carga e suporte a transações

Sistema de Mensagens Assíncronas

- Embora cada fornecedor de um MOM utilize formatos e protocolos diferentes a semântica básica é a mesma e uma API Java é utilizada para criar, enviar e receber mensagens
- O EJB integra a funcionalidade do MOM ao seu modelo de componentes
- O EJB 3.0 suporta o sistema de mensagens assíncronas por meio do **Java Message Service – JMS** – *Message Driven Beans*

Serviços Web

- Tendência na computação distribuída
- “Aplicativos modulares autodescritos e autocontidos que podem ser publicados, localizados e invocados pela Web”
 - Independentes de plataforma
 - SOAP – gramática XML para protocolo de aplicativo
 - WSDL – gramática XML para definição de interface
- O EJB 3.0 permite o desenvolvimento de Serviços Web através da API JAX-WS

Arquitetura

Java Enterprise Edition

Arquiteturas JPA e EJB

- Para a construção de aplicações corporativas inicialmente a de se compreender as arquiteturas propostas
 - *Beans* de entidade
 - Componentes Enterprise *Beans*

Bean de Entidade

- Os *beans* de entidade espelham os objetos do mundo real
- Os *beans* de entidade na especificação **Java Persistence** estão disponíveis como POJOs → mapeados para tabelas de bancos de dados
- As entidades podem ser serializadas e enviadas pela rede

Bean de Entidade

- Implementando um *bean* de entidade:
 - Definir uma classe *bean*
 - Qual atributo funcionará como identificador
- Chave primária (identificador)
 - Fornece um identidade à classe *bean*
- Classe *bean*
 - Pode conter lógica do negócio, mas normalmente define as informações necessariamente persistentes

Bean de Entidade

- A classe *bean* é um POJO
 - Implementando **java.io.Serializable**
 - Anotada com **@javax.persistence.Entity**
 - Ter um campo designado como a chave primária, anotado com **@javax.persistence.Id**

Bean de Entidade

- Para interagir com *beans* de entidade, a **Java Persistence API** fornece o **EntityManager**
 - Todo acesso às entidades passa por este serviço
 - Este, fornece uma API de consulta e métodos de ciclo de vida para a entidade
- A JPA pode ser utilizada independentemente do servidor de aplicação *Enterprise Edition*
- Vejamos um exemplo de entidade...

```
import javax.persistence.*;
@Entity
@Table(name="ALUNO")
public class Aluno {
    private int id;
    private String nome;
    private String matricula;

    @Id
    @GeneratedValue
    @Column(name="ALUNO_ID")
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    @Column(name="ALUNO_NOME")
    public String getNome() { return nome; }
    public void setNome(String nome) { this.nome = nome; }

    @Column(name="ALUNO_MAT")
    public String getMatricula() { return matricula; }
    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }
}
```

Descritores de Implantação e Arquivos XML

- Os *beans* de entidade são agrupados em um conjunto chamado **unidade de persistência**
- Uma unidade de persistência deve estar associada a um banco de dados particular
- Cada **EntityManager** é responsável por gerenciar uma unidade de persistência
- Informações essas armazenadas em um descritor de implantação (**persistence.xml**)

Componente Enterprise Bean

- Dois tipos fundamentais:
 - Beans de sessão – *session beans*
 - Acessados utilizando vários protocolos de objetos distribuídos (RMI-IIOP)
 - Responsáveis por gerenciar processos ou tarefas
 - Beans orientados a mensagens – **MDBs**
 - Processam mensagens assincronamente a partir de sistemas como o JMS

Classes e Interfaces

- Interface Remota - **@javax.ejb.Remote**
 - Define os métodos do bean de sessão que são acessíveis por aplicativos de fora do contêiner
- Interface Local - **@javax.ejb.Local**
 - Define os métodos do bean de sessão acessíveis apenas por outros beans executando na mesma JVM
- Interface endpoint - **@javax.jws.WebService**
 - Define os métodos de negócio que são acessíveis por aplicativos fora do contêiner por meio de SOAP

Classes e Interfaces

- Interface MDB - **@javax.ejb.MessageDriven**
 - Define a forma como o sistema de mensagens assíncronas podem entregar mensagens
- Classe bean
 - Contém a lógica do negócio e precisa ter pelo menos uma interface remota, local ou *endpoint*
 - **@javax.ejb.Stateless** – sessão sem estado
 - **@javax.ejb.Stateful** – sessão com estado

A Interface Remota

- Exemplo - CalculadoraRemote:

```
import javax.ejb.Remote;
```

```
@Remote
```

```
public interface CalculadoraRemote {  
    public int soma(int x, int y);  
    public int subtrai(int x, int y);  
}
```

A Classe Bean

- Exemplo - CalculadoraBean:

```
import javax.ejb.*;

@Stateless
public class CalculadoraBean implements CalculadoraRemote {
    public int soma(int x, int y) {
        return x + y;
    }
    public int subtrai(int x, int y) {
        return x - y;
    }
}
```

Descritores de Implantação

- De que maneira o contêiner EJB sabe como tratar a segurança, transações e demais serviços?
 - (1) padrões (default) intuitivos
 - (2) anotações
 - (3) descritores de implantação em XML
- Exemplo de padrões:
 - O atributo padrão de transação é REQUIRED
 - A semântica de segurança é UNCHECKED

EJBs de Sessão

Session Beans

Beans de Sessão

- Atuam como **controlador** da interação entre outros *beans*
 - Representam um fluxo de tarefas
- Contêm a **lógica de negócio**
- Gerenciam as interações entre os *beans* de entidade
- Metáfora da peça de teatro
 - Os *beans* de sessão representam o roteiro
 - Os *beans* de entidade representam os atores em cena

Tipos de Beans de Sessão

- Dois tipos básicos
- (1º) *Beans* de sessão sem informação de estado
 - Coleção de serviços relacionados (métodos)
 - Não é mantido estado entre uma e outra invocação de qualquer método
- (2º) *Beans* de sessão com informação de estado
 - Trabalha em favor de um cliente, mantendo o estado
 - **estado conversacional** – que expira com o tempo
 - O estado é compartilhado pelos métodos do *bean*

Bean de Sessão sem Informação de Estado

- Não mantém nenhum estado conversacional
- Pode ser compartilhado entre vários clientes
- Todas as informações necessárias a um método são passadas por parâmetros
- Como não guarda informação de estado não requer **passivação** e **ativação**
- A exceção à regra, são as informações obtidas do: (1) SessionContext, (2) JNDI ENC, ou (3) referências ao ambiente injetadas no *bean*

Ciclo de Vida



Exemplo: EJB ProcessPayment #1

- Cenário:
 - Utilizado primariamente para o pagamento de passagens num sistema de agência de turismo
 - Compartilhado por várias aplicações que cobram serviços do cliente
 - Os pagamentos são registrados em uma tabela do banco de dados chamada de PAYMENT
 - Várias formas de pagamento podem ser utilizadas: cartão de crédito, cheque e dinheiro

Exemplo: EJB ProcessPayment #2

- Tabela do banco de dados PAYMENT:

```
CREATE TABLE PAYMENT
(
    customer_id      INTEGER,
    amount           DECIMAL(8,2),
    type             CHAR(10),
    check_bar_code   CHAR(50),
    check_number     INTEGER,
    credit_number     CHAR(20),
    credit_exp_date  DATE
)
```

Exemplo: EJB ProcessPayment #3

- Um bean de sessão sem informação de estado tem uma ou mais interfaces de negócio
- Uma interface de negócio pode ser **remota** ou **local**
 - Interfaces remotas são capazes de receber invocações de métodos de cliente da rede
 - Interfaces locais estão disponíveis dentro da mesma JVM
- Interface do negócio ProcessPayment:
 - Métodos: **byCredit()**, **byCash()** e **byCheck()**
 - Terá duas interfaces diferentes: uma local e uma remota

Exemplo: EJB ProcessPayment #4

```
package com.titan.processpayment;
import com.titan.domain.*;

public interface ProcessPayment {

    public boolean byCheck(Customer customer, CheckDO check,
        double amount) throws PaymentException;

    public boolean byCash(Customer customer, double amount)
        throws PaymentException;

    public boolean byCredit(Customer customer,
        CreditCardDO card, double amount)
        throws PaymentException;
}
```

Exemplo: EJB ProcessPayment #5

```
package com.titan.processpayment;  
import javax.ejb.Remote;  
  
@Remote  
public interface ProcessPaymentRemote  
    extends ProcessPayment {  
}
```

```
package com.titan.processpayment;  
import javax.ejb.Local;  
  
@Local  
public interface ProcessPaymentLocal  
    extends ProcessPayment {  
}
```

Exemplo: EJB ProcessPayment #6

- Cada método de negócio do ProcessPayment EJB recebe um *bean* de entidade como um parâmetro
- Como os *beans* de entidade são POJOs podem ser serializados pela rede – interface `java.io.Serializable` ou `Externalizable`

Exemplo: EJB ProcessPayment #7

- Classe de negócio CreditCardDO:

```
/* CreditCardDO.java */  
package com.titan.processpayment;  
import java.util.Date;  
  
public class CreditCardDO implements java.io.Serializable {  
    final static public String MASTER_CARD = "MASTER_CARD";  
    final static public String VISA = "VISA";  
    final static public String AMER_EXPRSS = "AMER_EXPRSS";  
    final static public String DISCOVER = "DISCOVER";  
    final static public String DINERS = "DINERS";  
}
```

Exemplo: EJB ProcessPayment #8

■ continuação...

```
public String number;  
public Date expiration;  
public String type;  
  
public CreditCardDO(String nmb, Date exp, String typ) {  
    number = nmb;  
    expiration = exp;  
    type = typ;  
}  
  
// Acessadores e modificadores ...  
}
```

Exemplo: EJB ProcessPayment #9

- Classe de negócio CheckDO:

```
/* CheckDO.java */
package com.titan.processpayment;

public class CheckDO implements java.io.Serializable {

    public String checkBarCode;
    public int checkNumber;

    public CheckDO(String barCode, int number) {
        checkBarCode = barCode;
        checkNumber = number;
    }
    // Acessadores e modificadores ...
}
```

Exemplo: EJB ProcessPayment #10

- A classe *bean* ProcessPaymentBean:

```
package com.titan.processpayment;  
import com.titan.domain.*;
```

```
import java.sql.*;  
import javax.ejb.*;  
import javax.annotation.Resource;
```

```
import javax.sql.DataSource;  
import javax.ejb.EJBException;
```

```
@Stateless
```

```
public class ProcessPaymentBean  
    implements ProcessPaymentRemote, ProcessPaymentLocal {
```

```
final public static String CASH = "CASH";  
final public static String CREDIT = "CREDIT";  
final public static String CHECK = "CHECK";
```

```
@Resource(mappedName = "titanDB") DataSource dataSource;  
@Resource(name = "min") int minCheckNumber;
```

```
public boolean byCash(Customer customer, double amount)  
    throws PaymentException {  
    return process(customer.getId(), amount, CASH,  
        null, -1, null, null);  
}
```

```
public boolean byCheck(Customer customer, CheckDO check,  
    double amount) throws PaymentException {  
    if (check.checkNumber > minCheckNumber) {  
        return process(customer.getId(), amount, CHECK,  
            check.checkBarCode,  
            check.checkNumber, null, null);  
    } else {  
        throw new PaymentException("Number too low");  
    }  
}
```

```
public boolean byCredit(Customer customer,
                        CreditCardDO card, double amount)
                        throws PaymentException {
    if (card.expiration.before(new java.util.Date())) {
        throw new PaymentException("Card expired!");
    } else {
        return process(customer.getId(), amount, CREDIT,
                        null, -1, card.number,
                        new Date(card.expiration.getTime()));
    }
}
```

```
private boolean process(int customerID, double amount,
                        String type, String checkBarCode,
                        int checkNumber, String creditNumber,
                        Date creditExpDate) throws PaymentException {
```

```
    Connection com = null;
    PreparedStatement ps = null;
```

```
try {  
    con = dataSource.getConnection();  
    ps = con.prepareStatement("INSERT INTO "+  
        "payment (customer_id, amount, type, "+  
        "check_bar_code, check_number, "+  
        "credit_number, credit_exp_date) "+  
        "VALUES(?,?,?,?,?,?,?)");  
    ps.setInt(1, customerID);  
    ps.setDouble(2, amount);  
    ps.setString(3, type);  
    ps.setString(4, checkBarCode);  
    ps.setInt(5, checkNumber);  
    ps.setString(6, creditNumber);  
    ps.setDate(7, creditExpDate);  
    int retVal = ps.executeUpdate();  
    if (retVal != 1) {  
        throw new EJBException  
            ("Payment insert failed");  
    }  
    return true;  
}
```

```
catch(SQLException sql) {  
    throw new EJBException(sql);  
}  
finally {  
    try {  
        if (ps != null) ps.close();  
        if (com != null) con.close();  
    } catch (SQLException se) {  
        se.printStackTrace();  
    }  
}  
}  
}
```

Exemplo: EJB ProcessPayment #11

- Acessando propriedades do ambiente (injeção):

```
<ejb-jar xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/ejb-jar\_3\_0.xsd" version="3.0">
  <enterprise-beans>
    <session>
      <ejb-name>ProcessPaymentBean</ejb-name>
      <env-entry>
        <env-entry-name>min</env-entry-name>
        <env-entry-type>java.lang.Integer</env-entry-type>
        <env-entry-value>250</env-entry-value>
      </env-entry>
    </session>
  </enterprise-beans>
</ejb-jar>
```

SessionContext

- Fornece uma visualização do ambiente do contêiner EJB – interface com o contêiner EJB
 - Obter informações sobre o contexto da chamada de invocação do método
 - Acesso à vários serviços EJB
- Um *bean* de sessão pode obter uma referência utilizando a anotação **@Resource**

```
@Sateteless
public class ProcessPaymentBean implements ProcessPaymentLocal {
    @Resource SessionContext ctx;
    ...
}
```