



# Fundamentos básicos de Transações

*Helder da Rocha*  
[www.argonavis.com.br](http://www.argonavis.com.br)

- *Apresentar conceitos essenciais sobre transações em aplicações J2EE*
  - *Este curso não trata de transações em profundidade. Há vários tópicos importantes não mencionados aqui.*
  - *Para uma abordagem mais profunda, consulte capítulo sobre transações no J2EE Tutorial e no livro Mastering EJB 2 (veja também slides do curso J550 - EJB)*
- *Roteiro*
  - *O que são transações*
  - *Transações ACID*
  - *Container-managed transactions*
  - *Bean-managed transactions*

# O que são transações

- Uma transação é uma seqüência de operações que devem ser executadas como se fosse uma só
  - Caso uma das operações falhe, todo o processo é abortado e revertido ao estado anterior ao início da transação
- Uso típico consiste do emprego de dois métodos
  - **commit()**: usado após a declaração das instruções a serem executadas
  - **rollback()**: chamado quando há uma falha. O método rollback() deve desfazer todas as operações iniciadas pelas instruções que foram executadas
- Há duas formas de controlar transações em J2EE
  - **CMT - Container Managed Transactions**: container administra todo o processo por método - aspect-oriented approach.
  - **BMT - Bean Managed Transactions**: bean chama as APIs diretamente e controla início e fim das transações

- *Características essenciais de uma transação:*  
**A**tomic, **C**onsistent, **I**solated, **D**urable
- **Atomic**
  - *Todas as tarefas de uma unidade transacional devem funcionar sem erros ou todo o processo é revertido*
- **Consistent**
  - *O estado do sistema após uma transação deve manter-se consistente (transações devem englobar processos de negócio completos)*
- **Isolated**
  - *Transação deve poder executar sem interferência de outros processos*
- **Durable**
  - *Dados alterados durante a transações devem ser guardados em meio persistente até que a transação complete com sucesso*

# CMT: JTS declarativo

- *Container-Managed Transactions*
  - *Controle de transações totalmente gerenciado pelo container*
  - *Não permite o uso de métodos commit() e rollback() dentro do código*
  - *Única forma de controlar transações em Entity Beans*
- *Deployment descriptor*
  - *Declaração do tipo de transação do bean: <ejb-jar>/<enterprise-beans> / <session>\*:  
<transaction-type>Container</transaction-type>*
  - *Descrição do atributo para cada método*

```
<assembly-descriptor>  
  <container-transaction>  
    <method> ... </method>  
    <trans-attribute>Required</trans-attribute>  
  </container-transaction>  
  (...)  
</assembly-descriptor>
```

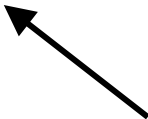
\* ou message-driven

# Atributos (políticas transacionais)

- **NotSupported**
  - Chamada de um método com este atributo suspende a transação até que o método termine
- **Supports**
  - Método será incluído no escopo de uma transação se for chamado dentro de uma transação
- **Required**
  - Método tem que ser chamado no escopo de uma transação
- **RequiresNew**
  - Nova transação sempre é iniciada quando este método for chamado
- **Mandatory**
  - Método sempre deve fazer parte do escopo do cliente
- **Never**
  - Método não deve ser chamado no escopo de uma transação

# JTS declarativo - exemplo

```
<ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>BankEJB</ejb-name>
      <home>j2eetut.bank.BankHome</home>
      <remote>j2eetut.bank.Bank</remote>
      <ejb-class>j2eetut.bank.BankBean</ejb-class>
      <session-type>Stateful</session-type>
      <transaction-type>Container</transaction-type>
      (...)
    </session>
  </enterprise-beans>
  <assembly-descriptor>
    <container-transaction>
      <method>
        <ejb-name>BankEJB</ejb-name>
        <method-name>getSavingBalance</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
    </container-transaction> (...)
  </assembly-descriptor>
</ejb-jar>
```



bank-ejb-jar.xml

# BMT - controle explícito de transações

- **Bean-Managed Transaction:** JTA pode ser usada por session beans e message-driven beans (BMT não pode ser usada por Entity Beans). Deployment descriptor:  
`<transaction-type>Bean</transaction-type>`
- **Transações JDBC**
  - commit/rollback em `java.sql.Connection`
- **JTS:** Java Transaction Service - implementação Java de OTS (Object Transaction Service)
  - Para programar diretamente em JTS é preciso usar sintaxe CORBA
- **JTA:** Java Transaction API
  - Mais simples que programar diretamente em JTS
  - Dois componentes: interface cliente de alto nível e interface XA (para transações distribuídas) de baixo nível
  - Principal interface do JTA:  
`javax.transaction.UserTransaction`



# Transações explícitas JTA - exemplo

```
(...)  
public void withdrawCash(double amount) {  
    UserTransaction ut = sessionCtx.getUserTransaction();  
    try {  
        ut.begin();  
        updateChecking(amount);  
        machineBalance -= amount;  
        insertMachine(machineBalance);  
        ut.commit();  
    } catch (Exception ex) {  
        try {  
            ut.rollback();  
        } catch (SystemException syex) {  
            throw new EJBException  
                ("Rollback failed: " + syex.getMessage());  
        }  
        throw new EJBException  
            ("Transaction failed: " + ex.getMessage());  
    }  
}
```

# Transações JDBC - exemplo

```
(...)  
public void ship (String productId,  
                  String orderId,  
                  int quantity) {  
    try {  
        con.setAutoCommit(false); // con é java.sql.Connection  
        updateOrderItem(productId, orderId);  
        updateInventory(productId, quantity);  
        con.commit();  
    } catch (Exception e) {  
        try {  
            con.rollback();  
            throw new EJBException("Transaction failed: "  
                                   + e.getMessage());  
        } catch (SQLException sqx) {  
            throw new EJBException("Rollback failed: "  
                                   + sqx.getMessage());  
        }  
    }  
}
```

WarehouseBean.java

# Transações: riscos

- *O uso de transações oferece alguns riscos*
  - *Impacto significativo na performance*
  - *Deadlock*
- *Em EJB o risco de deadlock é minimizado pela proibição de uso de threads, e uso correto das políticas transacionais (Required, RequiresNew, etc.)*
  - *Mantenha o sistema simples: use transações apenas nos métodos onde isto é realmente necessário*
  - *Use níveis de isolamento fracos quando possível*
- *A correta escolha entre BMT/CMT, políticas e níveis de isolamento de cada método permitem amplo controle sobre performance x segurança dos dados.*

- Os exemplos deste capítulo são do J2EE Tutorial
- Para executá-los, configure *build.properties* e rode o *ant* com os targets:
  - *jboss.deploy* implanta os beans no JBoss
  - *run.container.jts.client* roda aplicação CMT (Bank)
  - *run.bean.jdbc.client* roda aplicação BMT - JDBC (Warehouse)
  - *run.bean.jta.client* roda aplicação BMT - JTA (Teller)

- [1] *Richard Monson-Haefel. Enterprise JavaBeans, 3rd. Edition. O'Reilly and Associates, 2001*
- [2] *Dale Green. J2EE Tutorial - Transactions. Sun J2EE Tutorial, 2002*

*helder@ibpinet.net*

***www.argonavis.com.br***