



# **Persistência Objeto-Relacional com Java**

**Fernando Lozano**

**<http://www.lozano.eti.br>**

**Consultor Independente**

Prof. Faculdades UniABEU

Prof. SENAC

Editor Adjunto da Revista Java Magazine

# Sobre o Autor

- Red Hat Certified Engineer  
LPI Certified Professional Level I  
Sair GNU/Linux Certified Professional
- IBM Certified Network Engineer  
IBM Certified DB2 Administrator & Developer
- Microsoft Certified Systems Engineer  
Microsoft Certified Solutions Developer
- Webmaster da Free Software Foundation
- Conselheiro do LPI Brasil
- Editor Adjunto da Revista Java Magazine
- Autor do Livro "*Java em GNU/Linux*"  
Ed. Alta Books [www.altabooks.com.br](http://www.altabooks.com.br)



# Agenda

- Porque persistir Objetos?
- Componentes de um mecanismo de persistência
  - Mapeamento OO/Relacional
  - Linguagem de Consulta
  - API de acesso
- Padrões de persistência do Java
  - CMP/CMR
  - JDO
  - Hibernate

# Porque Persistir Objetos

- Modelagem de negócios e processos baseados na UML
- Modelagem de sistemas (informações, arquitetura de código) Orientada a Objetos
- Adoção de design patterns OO
- Reduzir o gap entre o modelo conceitual e lógico dos sistemas (OO) e a tecnologia de armazenamento (Relacional)
- Maximizar o aproveitamento de conhecimento do profissional Java

# Por que não Bancos OO?

- Tipos de dados e modelo de objetos diferentes da linguagem de programação
- Estruturas de dados otimizadas para disco são diferentes as estruturas otimizadas para memória
- Bancos OO ainda não são tão maduros quanto os relacionais em respeito a recoverabilidade, performance e distribuição

# Persistência OO x Servidores de Aplicação

- Caches de objetivos agressivos e distribuídos
- Delimitação de transações
- Otimização de comandos SQL em função do cache e transações
- A lógica de negócios que costumava estar no banco (triggers, stored procedures) é migrada com vantagens para os objetos de negócios (EJBs)

# Componentes de um Mecanismo de Persistência

- Mapeamento OO/Relacional
- Linguagem de Consulta
- API de acesso

# Mapeamento OO/Relacional

- Um modelo OO é semanticamente mais rico do que um modelo Relacional
- Vários modelos Relacionais são possíveis para um mesmo modelo OO
- Objetivos de performance e confiabilidade podem levar a mapeamentos “degenerados”, semelhantes a bancos relacionais desnormalizados

# OO x Relacional

- Relacional
  - Entidades fortes e fracas
  - Relações 1/N, M/N e Chaves Estrangeiras
- Java
  - Composição e Agregação
  - Sets, Maps, Lists, Trees, ...

# Otimização de SQL

- Postergar gravações até o final das transações
  - Inserir/atualizar/deletar menos registros
  - Utilizar apenas as colunas afetadas
  - Fim dos problemas de nível de isolamento de transações
  - Lock otimista realizado pelo mecanismo de persistência
- Quando realizar leituras
  - No acesso a uma propriedade do objeto?
  - Carga antecipada de objetos relacionados?

# Linguagem de Consulta

- O grande avanço dos bancos relacionais em relação às tecnologias anteriores (rede, hierárquica, ISAM) foi a linguagem de consulta declarativa
- Utilizar apenas o grafo de objetos é reverter para consultas realizadas de forma procedural
- Mas um modelo OO não é um modelo Relacional – não são realizadas em termos de junções (joins), produtos cartesianos e projeções!

# OQL x SQL

- SQL necessita de muitos joins:

```
select nome from produto p, venda v
where p.id = v.produto
```

- OQL pode utilizar o grafo de objetos:

```
select v.produto.nome from venda v
```

- OQL pode utilizar operadores de conjunto:

```
select v.cliente from vendas v,
in v.produtos p
where sum(p.valor) > 10000
```

# API de Acesso

- Fornece os métodos para recuperação e atualização de objetos persistência
- APIs intrusivas:  
Exigem que suas classes estendam uma classe ou implementem uma interface
- APIs transparentes  
Utilizam aspectos, manipulação de byte-codes ou pré-processamento para modificar dinamicamente as classes e inserir chamadas ao mecanismo de persistência

# APIs de Acesso

- Basicamente, inserem um objeto persistente no contexto de uma transação e informam quando a transação é encerrada
- Fornecem meios de recuperar objetos persistentes, mas também é possível utilizar os métodos getXXX dos próprios objetos
- Em geral não há métodos explícitos para “gravar”, mas pode haver um factory para criar instâncias persistentes
- Na prática, o desenvolvedor usa mais a API das suas classes do que do mecanismo de persistência

# Padrões de Persistência do Java

- CMP/CMR
- JDO
- Hibernate

# CMP/CMR

- API intrusiva, considera que um objeto persistente é antes disso um componente distribuído (remoto)
- Exige o uso de um servidor de aplicações
- Bastante maduro, com recursos avançados de otimização e gerenciamento na maioria dos produtos disponíveis no mercado
- Curva de aprendizado bastante longa
- O padrão atual (2.1) peca por não especificar como é o mapeamento OO/Relacional, gerando dependência em relação ao servidor de aplicação (descritores proprietários)

# CMP/CMR

- Tem má fama no mercado por causa de limitações da versão 1.x, que não tinha recursos para relacionamentos entre Entity Beans
- A versão 3.0 é baseada no fato de que objetos persistentes não são expostos para a camada de apresentação (cliente) no modelo MVC
- Ficará semelhante ao Hibernate e ao JDO
- A nova versão também terá um padrão para o mapeamento OO/Relacional, compartilhado com o JDO 2.0

# JDO

- Criado como alternativa ao CMP para aplicações que não rodam no servidor de aplicações
- Mas também é bem-integrado em alguns servidores de aplicações
- API não intrusiva, diferentes implementações utilizam manipulação de bytecodes ou pré-processamento
- O padrão atual peca por não definir o mapeamento OO/Relacional, mas o problema será resolvido na versão 2.0 da especificação

# Hibernate

- Não é padrão do JCP, mas é quase um padrão de fato do mercado
- Ganhou muito espaço por causa do “preconceito” contra EJBs e da padronização incompleta do CMP e do JDO
- A versão 3.0 está melhorando muito a documentação e recursos de otimização
- Incorporado ao JBoss 4.0 como base do seu mecanismo CMP/CMR
- Famoso pela sua flexibilidade em termos de linguagem de consulta e API

# Conclusões

- Não há ainda um mecanismo estabelecido como o único padrão, e provavelmente os três indicados (além alguns outros proprietários, como o TopLink) permanecerão durante longo tempo com seus nichos
- Há uma convergência natural nestes mecanismos, que ficam cada vez mais parecidos
- Qualquer das três alternativas é uma boa escolha, mas avalie com cuidado sua capacidade de otimizar o ambiente de produção
- **Só haverá bom proveito se o desenvolvedor abandonar paradigmas do mundo relacional!**

# Referências

- **Mapeamento Objeto/Relacional**  
[www.chimu.com/publications/objectRelational](http://www.chimu.com/publications/objectRelational)  
[www.objectarchitects.de/ObjectArchitects/orpatterns](http://www.objectarchitects.de/ObjectArchitects/orpatterns)  
[www.objectmatter.com/vbsf/docs/maptool/ormapping.html](http://www.objectmatter.com/vbsf/docs/maptool/ormapping.html)
- **CMP/CMR**  
[java.sun.com/products/ejb/index.jsp](http://java.sun.com/products/ejb/index.jsp)  
[www.onjava.com/pub/a/onjava/2001/09/19/ejbql.html](http://www.onjava.com/pub/a/onjava/2001/09/19/ejbql.html)
- **JDO**  
[java.sun.com/products/jdo](http://java.sun.com/products/jdo)  
[db.apache.org/ojb](http://db.apache.org/ojb)  
[db.apache.org/ojb/docu/tutorials/jdo-tutorial.html](http://db.apache.org/ojb/docu/tutorials/jdo-tutorial.html)  
[www.jpox.org](http://www.jpox.org)   [tjdo.sourceforge.net](http://tjdo.sourceforge.net)
- **Hibernate**  
[www.hibernate.org](http://www.hibernate.org)  
[www.systemmobile.com/articles/IntroductionToHibernate.html](http://www.systemmobile.com/articles/IntroductionToHibernate.html)

# Perguntas

- Dúvidas:  
[fernando@lozano.eti.br](mailto:fernando@lozano.eti.br)
- Palestras, artigos e Apostilas:  
[www.lozano.eti.br](http://www.lozano.eti.br)
- Livro:  
Java em GNU/Linux  
[www.altabooks.com.br](http://www.altabooks.com.br)

