Ei! Como é o seu DAO? Ele é tão abstraído quanto o meu?

Postado em 26. ago, 2006 por Paulo Silveira em Java

Essa é uma pergunta comum entre os desenvolvedores. Alguns acham que há uma fórmula única, que DAO é um pattern fechado e que possui seu diagrama de classes bem definido. Eu discordo.

Na minha humilde opinião, DAO é uma maneira de você encapsular o seu acesso a dados. Não importa se é através de uma factory de objetos que recebem uma Session do hibernate por injeção de dependências, ou se é um montão de métodos estáticos cheio de SQL dentro deles (argh!). Obviamente a última opção envolve diversos outros antipatterns (como singleton e métodos estáticos, que já criticamos anteriormente), porém ela consegue isolar seu acesso a dados, cumprindo o papel do DAO e de alguma maneira organizando sua camada.

Lendo a definição de Data Access Object no site da Sun, podemos chegar a um molde comum, mas nada impede que possamos incrementa-lo e modifica-lo de acordo com nossas necessidades. Por exemplo, com o Java 5, podemos tirar proveito dos generics. Na Caelum costumamos usar uma interface para os nossos DAOs parecida com a seguinte:

```
interface Dao<T> {
  T search(Serializable id);
  void persist(T t);
  void remove(T t);
  // ...
}
```

A nossa implementação de DAO genérico para hibernate fica parecida com:

```
class GenericHibernateDao<T> implements Dao<T> {
  private Session session;
  private Class persistentClass;

  public GenericHibernateDao(Session session, Class persistentClass) {
    this.session = session;
    this.persistentClass = persistentClass;
  }

  public T search(Serializable id) {
     return session.load(persistentClass, id);
  }

  // outros metodos da interface Dao
}
```

Repare que ele recebe `session` como argumento, que poderia ser injetada por um container. Outra opção seria buscar a session de um singleton (o clássico `HibernateUtil`), mas a primeira maneira é bem mais elegante e testável. Repare que ainda temos os problemas de controle de transações: onde abrimos e comitamos? dentro do Dao? de dentro do injetor de dependências? de dentro da sua lógica de negócios? Nós costumamos fazer esse controle em um interceptador ou mesmo em um `Filter` de servlets, dependendo do framework MVC usado. Nosso filtro é semelhante com o recomendado pela documentação do hibernate.

E então podemos utilizar nosso DAO:

```
Dao<Cliente> clienteDao = new GenericHibernateDao<Cliente>(session);
```

```
clienteDao.persist(cliente);
Cliente c5 = clienteDao.search(5);
```

Para as entidades que você precisa de mais que um simples cria/le/atualiza/deleta, criamos uma nova interface:

```
interface ClienteHibernateDao extends Dao<Cliente> {
   List<Cliente> buscaTodosClientesQueDevemDinheiro();
}
```

E teríamos uma classe `ClienteHibernateDao` que estende `GenericHibernateDao<Cliente>` e implementa `ClienteDao`.

Meus colegas da easytech preferem abstrair também o tipo da chave primária:

```
interface Dao<T, I> {
  T search(I id);
  // ...
}
```

Dessa maneira você realmente ganha mais tipagem, já que no DAO anterior poderíamos passar qualquer objeto serializável como chave. Podemos refinar muito mais nossas classes de acesso a dados, mas até que ponto chegar?

Ficar abstraindo demais a sua camada de persistência pode levar a proliferação de centenas de pequenas classes que simplesmente não fazem nada, só delegam e delegam. Claro que existem casos em que abstrair para apenas delegar é interessante, diminuindo o acoplamento entre suas classes, porém isso pode chegar a um ponto que vai atrapalhar o desenvolvimento.

Nesta entrada do blog dos desenvolvedores do hibernate, o uso de camadas para abstrair as ferramentas de mapeamento objeto relacional são duramente criticadas.

E você, qual é a receita de bolo para o seu DAO? Até onde você o refina?

Paulo Silveira (Google+)

Mais sobre o autor

Tags: design patterns, generics

31 Respostas para “Ei! Como é o seu DAO? Ele é tão abstraído quanto o meu?”

1. 

   Urubatan

   27. ago, 2006

   eu uso um bem parecido com isto 😀

   mas sigo mais a linha do pessoal a easytech como você comentou, ja com a classe da PK abstraida, e no meu DAO ja tenho metodos find by example prontos também …

   ```
   public interface TOBaseDao {
           public Class getObjectClass();
           public T save(T object);
           public T load(PK primaryKey);
           public T get(PK primaryKey);
           public List listAll();
           public List findByExample(final T example);
           public T findOneByExample(final T example);
           public List listAll(final int first,final int max);
           public int listAllPageCount();
           public List findByExample(final T example,final int first,final int max);
           public int findByExamplePageCount(final T example);
   ```

```
        public void update(T object);
        public void delete(T object);
}
```

e a implementação padrão dele fica parecido com isto:

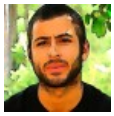
```
public class EnderecoEmailDaoImpl extends TOBaseHibernateDao implements EnderecoEmailDao {

    public EnderecoEmailDaoImpl() {
        super(EnderecoEmail.class);
    }

    @Override
    protected void addPropertiedToCriteria(final Criteria c, final EnderecoEmail example) {
        if (example.getCliente() != null) {
            c.add(Property.forName("cliente").eq(example.getCliente()));
        }
    }
}
```

O método `addPropertiedToCriteria` serve para contornar o problema do `findByExample` do Hibernate não considerar campos de relacionamento com outras entidades nos filtros … assim só com isto eu tenho um cadastro simples prontinho em 20 minutos 😀

Mas cada doido com as suas manias …

2. 

   Ederson de Lima

   28. ago, 2006

   Muito bom,
   e gostei da sua definição do DAO, utilizalo como encapsulamento o seu acesso aos dados.

   Muito bom..

3. 

   Michael Nascimento Santos

   28. ago, 2006

   Por que não declarar `persistentClass` como `Class<T>` ou `Class<? extends T>`?

4. 

   Paulo Silveira

   28. ago, 2006

   É verdade Michael, dessa maneira poderíamos usar o @SurpressWarnings(“unchecked”) nos métodos sem correr risco algum…

   Já usar com o `? extends T` não funciona, pois alguém pode criar um new `Dao(session, FilhaDeFuncionario.class)` e passar outras filhas de `Funcionario` como argumento, dando erro em alguns métodos que usem o `findByExample`, entre outros (já que o criteria seria criando com `FilhaDeFuncionario.class` e o Example seria criado em cima de uma instância de outra filha).

5. 

   Paulo Silveira

28. ago, 2006

Dando uma pesquisada, vi que na documentação do hibernate tem soluções muito parecidas com a de todos nós:

http://www.hibernate.org/328.html

Como o Michael indicou para o caso do Class<T>, devemos também restringir a chave primária para ser compatível com `Serializable`:

```
interface Dao<T, ID extends Serializable> {
}
```

6. 

Fabio Kung

29. ago, 2006

Ultimamente, andei pensando muito num assunto muito relacionado a tudo isso: a herança que a gente usa quando estamos escrevendo nossos daos…

```
public class UsuarioDAO extends GenericDAO implements UsuarioDAO { ... }
```

Se pararmos para pensar, esse é um exemplo de mau uso da herança. Estamos herdando sem usar o polimorfismo! Herdamos de GenericDAO só por conveniência, para ganhar alguns métodos que estão lá.
Várias vezes já tive vontade de mudar isso e usar um modelinho diferente para os meus DAOs, sem essa herança.
Na prática, como a herança não me trouxe nenhuma grande dor de cabeça (além de ter que lembrar de estender a classe GenericDAOWithGreatCRUDStuff e implementar a interface UselessWithAtMostThreeMethodsDAO), continuo usando a herança feia!

7. 

Fabio Kung

29. ago, 2006

Já quanto a abstrair o framework de mapeamento objeto relacional, eu sou totalmente a favor. Há alguns casos em que a performance do framework não é satisfatória, ou a ferramenta não suporta uma funcionalidade imprescindível do mecanismo final de persistência (banco de dados relacional, neste caso). Neste caso, que é só um exemplo, pode ser muito melhor sacrificar um pouco das vantagens da ferramenta de ORM e acessar o banco de dados direto.

O melhor de tudo é que o resto da aplicação nem vai saber disso, justamente porque a persistência está isolada. Não importa se é JDBC direto, se é Hibernate, se é Toplink, iBatis, ou se o banco de dados é legado e em um caso específico a persistência teve que ser feita em um arquivo xml.

8. 

Paulo Silveira

29. ago, 2006

Fábio, sobre o DAO você está coberto de razão. Estamos usando herança apenas pra nos livrar de escrever uma linhazinha a mais por método… porque o correto realmente seria a gente delegar as chamadas do `persist`, `search` e outros para o `GenericHibernateDao`. **ter** um `GenericHibernateDao` em vez de **ser** `GenericHibernateDao`.

9. 

Tiago Silveira

31. ago, 2006

Eu não faço isso com meus DAOs. Eu agrupo a persistência por módulos, sem herança feia, com uma única interface que tem todos os save(), load(), etc.

Essa abordagem também tem seus problemas, mas estou descobrindo que quanto menos camadas, menos código vc escreve pra atingir o mesmo resultado. Pelo menos no que toca a persistência de POJOs.

10. 

Bruno Hansen

17. out, 2006

Acho que nesse caso:

public class UsuarioDAO extends GenericDAO implements UsuarioDAO { … }

Não há muito problema se o UsuarioDao implementar metodos que ainda não existam no GenericDAO e não subscreva nada de GenericDAO!

Estava aqui pensando, se agente fosse delegar tudo para o GenericDAO tambem causaria a mau cheiro “Intermediário”

11. 

soro

20. out, 2006

quando vcs falam em delegar para o GenericDAO em vez d extendê-lo, seria algo assim:

```
public class GenericDAO implements DAO{
......blablabla
}

public class ClienteDAOImpl implements ClienteDAO{

GenericDAO genDao = new GenericDAO();

void persist(){
genDAO.persist(T t);
}
.....blablabla....
}
```

????

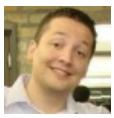
12. 

flaw

24. out, 2006

muito bom

http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

13. 

Márcio Barroso

22. nov, 2006

Olá, gostei mto da matéria.

Como sugerido, segue meus daos. Gostaria de que se por ventura alguém tenha alguma sugestão qto a minha implementação, que a comente por favor. Paulo, estou usando a sua sugestão qto ao uso do varargs. 

```
/**
* BaseDao definition
*/
public interface IBaseDao {

public void persist(T object);

public void delete(T object);

public void deleteByPrimaryKey(Serializable id);

public T findByPrimaryKey(Serializable id);

public Collection findByQuery(String hql);

public Collection findCriteriaByEntity(T object);

public Collection findCriteriaByEntityOrder(T object, String… fieldsOrdinance);

public Collection findCriteria(Criterion[] criterion, Order… order);

public Collection findAll();
}
```

Implementação :

```
public abstract class BaseDaoImpl implements IBaseDao {

private static Logger logger = Logger.getLogger(BaseDaoImpl.class);

private Class persistentClass;

protected SessionFactory sessionFactory;

protected HibernateTemplate hibernateTemplate;

public abstract void setSessionFactory(SessionFactory sessionFactory);

public BaseDaoImpl() {
logger.debug("Contrutor default");
this.persistentClass = (Class) ((ParameterizedType) getClass()
.getGenericSuperclass()).getActualTypeArguments()[0];
}

public void delete(T object) {
logger.debug("Deleting object " + object.getClass().getSimpleName());
this.hibernateTemplate.delete(object);
}

public void deleteByPrimaryKey(Serializable id) {
logger.debug("Deleting " + persistentClass.getSimpleName()
+ " by primary key " + id);
Object temp = this.hibernateTemplate.get(persistentClass, id);
this.hibernateTemplate.delete(temp);
}

public Collection findAll() {
logger.debug("Finding all " + persistentClass.getSimpleName());
DetachedCriteria criteria = DetachedCriteria
.forClass(persistentClass);
return this.hibernateTemplate.findByCriteria(criteria);
}

public T findByPrimaryKey(Serializable id) {
logger.debug("Finding " + persistentClass.getSimpleName()
```

```
+ ” by primary key ” + id);
return (T) this.hibernateTemplate.get(persistentClass, id);
}

public Collection findByQuery(String hql) {
logger.debug(“Find by query ” + hql);
return this.hibernateTemplate.find(hql);
}

public Collection findCriteria(Criterion[] criterion, Order… order) {
logger.debug(“Finding by criteria”);
DetachedCriteria criteria = DetachedCriteria
.forClass(persistentClass);
for (Criterion crit : criterion) {
criteria.add(crit);
}
for (Order ord : order) {
criteria.addOrder(ord);
}
return this.hibernateTemplate.findByCriteria(criteria);
}

public Collection findCriteriaByEntity(T object) {
logger.debug(“Finding by Entity”);
DetachedCriteria criteria = null;
try {
criteria = mountDynaCriteria(object);
} catch (Exception e) {
logger.error(e);
}
return this.hibernateTemplate.findByCriteria(criteria);
}

public void persist(T object) {
logger.debug(“Saving ” + persistentClass.getSimpleName());
this.hibernateTemplate.persist(object);
}

public Collection findCriteriaByEntityOrder(T object,
String… fieldsOrdinance) {
logger.debug(“Finding by criteria onder”);
DetachedCriteria criteria = null;
try {
criteria = mountDynaCriteria(object);
for (String ordinance : fieldsOrdinance) {
criteria.addOrder(Order.asc(ordinance));
}
} catch (Exception e) {
logger.error(e);
}
return this.hibernateTemplate.findByCriteria(criteria);
}

private DetachedCriteria mountDynaCriteria(Object object) throws Exception {
logger.debug(“Mounting dynamic criteria for ”
+ persistentClass.getSimpleName());
Class klass = object.getClass();
Method[] m = klass.getMethods();
DetachedCriteria criteria = DetachedCriteria.forClass(klass);
for (int i = 0; i {

public void setSessionFactory(SessionFactory sessionFactory) {
this.sessionFactory = sessionFactory;
this.hibernateTemplate = new HibernateTemplate(sessionFactory);
}
```

}

Eu tenho uma implementação genérica mais rica se falando em qtde de métodos, o q acaba economizando tempo na implementação dos meus daos.

Críticar e/ou sugestões são sempre bem aceitas.
[]'s

14. 

Edson

28. nov, 2006

Abstraído ou Abstrato?

15. 

Edson

30. nov, 2006

Insisto: Abstraído ou ABSTRATO?

16. 

Paulo Silveira

30. nov, 2006

ola Edson. Desculpe, mas nao entendi a insistencia. Voce esta querendo dizer de um possivel erro de portugues?

Olhando os dicionarios por ai voce pode ver que abstraido está corretissimo:
http://www.wordreference.com/ptes/abstra%EDdo

Ou foi algo semantico que nao consegui captar?

17. 

Thiago Oliveira

16. abr, 2007

Galera, sei que o post ficou em 2006, mas gostaria de expor algo (além de dizer que o post é muito bacana!).

A generalização do DAO é interessante para isolar a regra da persistência (evitar contato direto com o banco).
Mas existem situações em que a complexidade de uma rotina dentro de um método exige que se use alguma forma mais dinâmica de realizar uma consulta na base.
E nesses casos acabamos por usar HQL, ou JQPL (JPA), ou uma Criteria, ou até mesmo SQl Nativo, dentro da nossa regra de negócios.

Nesses casos, o que podemos fazer? Pois isso quebra o isolamento entre a regra de negócio e o mecanismo de persistência (imagine que preciso trocar o hibernate pela JPA.. precisaria rever todas essas consultas dinâmicas).. Existe saída??

Abraços!

18. 

Fabio Kung

17. abr, 2007

Como esses modos específicos de implementar a consulta devem continuar todos dentro do DAO (e não diretamente na sua lógica de negócios), a solução é reimplementar o DAO.

19. 

BARBARA

17. nov, 2007

Muito interessante,

Eu utilizo Top Link e gostaria muito de ver um exemplo DAO. Estou desenvolvendo um mas estou tendo alguns probleminhas pois só encontrei exemplos utilizando Hibernate.

20. 

Paulo

07. dez, 2008

Olá poderia me dar um exemplo para essa utilização do DAO ,pois sou iniciante e estou com varias duvidas …

21. 

Paulo

07. dez, 2008

Pois também não entendi onde vou acessar o arquivo persistence.xml com o jpa definido no arquivo ????alguém teria um exemplo pra isso ?????

22. 

Marcos

12. fev, 2009

A pergunta é, vale a pena?

* Normalmente uma classe possui muito mais metodos relacinados a dados que um CRUD.
* Agora seu implementador de código pode excluir qualquer coisa acesando diretamente a persistência, isso que dizer que seu código foi desenvolvido para 1 cabeça dar manutenção. Exemplo: excluir uma empresa da base (se não houver Constraint Restrict).
* Não economizou código, pelo contrario, escreveu mais. Praticamente terá que implementar muitos métodos do objeto session da API Hibernate.

*Nenhum isolamento de pacote…

Não existe uma DAO generico no proprio Hibernate? Porque? 😉

23. 

Eduardo

19. abr, 2010

Fala galera, alguem sabe me dizer como ficaria esse codigo todo sem usar o Hibernate?

Att.

24. 

lindemberg

11. nov, 2010

No meu DAO métodos tambem são objetos, logo existem as interfaces

```
interface Ipersist {
void persist(T t);
}

interface Iremove {
void remove(T t);
}

interface Isearch {
T search(Serializable id);
}
```

25. 

diego lovison

07. jun, 2011

Em vez de estender (DaoGenerico) ou implementar (IDaoGenerico) no DaoEspecifico, por que não injetamos o IDaoGenerico(persist, remove) e utilizamos para operações CRUD. E injetamos também o DaoEspecifico (clientesQueDevem, bonsClientes) e utilizamos para tratar questões especificas.

Porque vocês não fazem assim? O que tem de errado nesse conceito?

Obrigado

26. 

Paulo Silveira

08. jun, 2011

Ola Diego!

É uma opção válida. O que costuma acontecer é que o DAO genérico começa a não servir para muita coisa, pois, mesmo em situações simples como o persist, começamos a colocar outros detalhes de persistência dentro desses métodos, relativos a “lógica de persistência”. A partir desse momento o dao generico nao ajuda muito, nao importa se fizermos elegantemente via injecao de dependencias e sem heranca.

27. 

Diego Lovison

11. jun, 2011

Obrigado pela sua resposta Paulo.

Foi o que acabou acontecendo, eliminamos a implementação do DAO Genérico e, utilizamos somente a interface genérica que passou a ser

implementada por cada DAO especifico.

Um motivo disso foi que, em alguns momentos, as informações eram persistidas em arquivos. O fato de invocar o método persist do DAOGenérico sempre, havia momentos em que deveria ser invocado o metódo persist do DAOEspecifico. Isso iria acabar causando confusão para o desenvolvedor. Além de ficar uma arquitetura "confusa".

Eliminamos a palavra/conceito de DAO e adotamos a de Repository

Trackbacks/Pingbacks

1. Blog do Mister M » Seção de dados do blog » Vale a pena abstrair? - agosto 30, 2006

   [...] Um post do Paulo sobre abstração de DAOs me fez lembrar de escrever sobre este assunto. [...]

2. blog.caelum.com.br » Como não aprender orientação a objetos: Herança - outubro 14, 2006

   [...] Existem outros exemplos mais sutis de mau uso de herança. Um deles é quando criamos um DAO que é mãe de todos os DAOs. Nesse meu post sobre DAO genérico o Fábio Kung fez um bom comentário sobre isso, onde usamos herança apenas com o intuito de economizar meia dúzia de linhas, sendo que a relação "é um" não existe. [...]

3. blog.caelum.com.br » Brincando com Generics: o BizarreGenericDao - outubro 29, 2006

   [...] Em outras palavras, repare no código do Dao genérico que foi discutido aqui (estou pulando a interface por questão de simplicidade): [...]

4. rascunho » Blog Archive » links for 2009-01-20 - janeiro 20, 2009

   [...] Ei! Como é o seu DAO? Ele é tão abstraído quanto o meu? | blog.caelum.com.br Essa é uma pergunta comum entre os desenvolvedores. Alguns acham que há uma fórmula única, que DAO é um pattern fechado e que possui seu diagrama de classes bem definido. Eu discord (tags: blog.caelum.com.br 2009 mes0 dia19 DAO blog_post) [...]

Deixar uma Resposta

Nome (obrigatório) | E-mail (obrigatório) | Site (opcional)

Digite seu comentário aqui...

Deixe um comentário

ASSINE NOSSO RSS

Facebook

English | Portugese

Centro Paula Souza

Notificação de Acesso à Inte

CENTRO PAULA SOU

De acordo com a política da empresa, v acesso negado a URL:

Destaques

- Por uma Web mais rápida: 26 técnicas de otimização de Sites
- As Novidades do Eclipse Juno
- Use CDI no seu próximo projeto Java

- Flexibilidade em páginas para dispositivos móveis com media queries
- Como não aprender Java e Orientação a Objetos: getters e setters
- Usando o Google Maps e GPS no Android
- Pixels, pixels ou pixels? Dicas de Web Mobile com viewport
- Os 7 hábitos dos desenvolvedores Hibernate e JPA altamente eficazes
- Entenda os MVCs e os frameworks Action e Component Based
- As novidades do Hibernate 4

Siga-nos no Twitter