Padrões de Projeto

Repository

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande
Divisão de Computação

Agenda

1. Introdução
2. Problema
3. Solução
4. Funcionamento
   - Padrão Query Object
   - Resumo do Funcionamento
5. Quando usá-lo?
6. Considerações e Conclusão
   - Considerações
   - Conclusão
   - + Algumas Diferenças entre DAO e Repository
7. Exemplo

Introdução

Conceitualmente, um repositório encapsula o conjunto de objetos persistidos e as operações executadas sobre eles, fornecendo uma visão mais orientada a objetos da camada de persistência.

Um Repositório realiza a mediação entre as camadas mapeadores de dados e as camadas de domínio.

Os objetos clientes criam especificações de consultas declarativamente e as submetem ao Repositório para que sejam satisfeitas.

Introdução

Os Repositórios pertencem ao **modelo da aplicação**, **são parte de camada de negócios complementando o Model** e fornecendo estes objetos as outras camadas (como Controle e Apresentação, se aplicável) e recuperando estes do banco de dados.

Porém, sendo parte do modelo, os repositórios não conhecem detalhes de infraestrutura da aplicação (banco de dados, http, etc), e é na área de infraestrutura em que atuam os DAOs.

Introdução

Os DAO's tem o trabalho de traduzir as chamadas à persistência em chamadas de infraestrutura, sejam elas banco de dados, webservices, arquivos em disco ou outra abordagem qualquer.

A confusão acontece por que na maioria dos projetos toda persistência e acesso a dados é feita através de um, e apenas um, banco de dados.

- E, mais ainda, os frameworks atuais abstraem e traduzem os objetos em relacionamento de forma tão simples que o DAO simplesmente passa ao framework o que recebe e retorna o resultado.

Introdução

Exemplificando...

Dessa forma, em sistema onde temos um modelo *Usuário*, teríamos um *UsuárioRepository* atuando no modelo.

- Este repositório faria a interface de criar, recuperar e guardar os Usuários, e pode incluir métodos específicos, como filtros ou pré-preencher os usuários.

Mas o repositório não sabe da infraestrutura, então ele poderia ter que delegar a um DAO as chamadas.

- Desta forma os DAO apenas implementam as chamadas mais genéricas como: *select*, *insert*, *update* e *delete*.

Problema

As regras de pesquisa de uma instância ou conjunto de instâncias dependem fortemente das regras de domínio associadas às ditas instâncias.

- Por outro lado, as regras de pesquisa são traduzidas em comandos, como por exemplo, comandos SQL para um Banco de Dados Relacional.

Problema

- O padrão DAO foi usado para esconder qual o comando e qual o banco de dados em uso ao custo de deixar dentro do DAO a regra de domínio explicitada no SQL.
  - **Quando a regra mudava, o SQL tinha que ser alterado.**
- O padrão DAO prometia integrar sistemas tanto com banco de dados relacionais, tanto com LDAP ou XML, e mesmo com banco de dados diferentes simultaneamente.

Curiosidade

Algumas variantes existiam onde o SQL era deixado em um arquivo caso precisasse ser alterado depois.

Problema

Problema

Contudo, foi ficando claro que a cada nova tecnologia uma nova instrução de pesquisa tinha que ser escrita na linguagem ou idioma que essa tecnologia entendia.

Exemplo

Então uma simples regra como:

- Encontre os usuários ativos seria escrita de três formas diferentes usando XPath para XML, SQL para banco dados e Ldapquery para LDAP
  - **A mesma pesquisa era repetida em três linguagens diferentes.**

.

Solução

A solução que o padrão Repositório oferece é simples:

- **Separar a definição da instrução da sua execução.**

Então recorre-se ao padrão *Query Object* em que um objeto especialmente estruturado para representar a instrução de pesquisa e não uma simples string seria usado.

- O repositório tem portanto a responsabilidade de montar objetos no padrão *Query Object* e enviá-los a um outro objeto capaz de executar essa instrução de pesquisa.

Solução

Desta forma a verdadeira localização das instâncias e a tecnologia para as acessar é totalmente isolada, não apenas dos pesquisadores das instâncias, mas ainda, do construtor de tais pesquisas.

- Desta forma ao modificar a tecnologia de acesso e/ou a localização das instâncias apenas o interpretador precisa ser modificado.

Se a regra de domínio mudar, se mais ou menos constrangimentos precisarem ser feitos na instrução de pesquisa, apenas o repositório precisa ser modificado, comprovando que o padrão *Repository* realmente oferece uma boa **Separação de Responsabilidade**.

Funcionamento

Apesar de todo o mecanismo por trás dos panos, o Repositório apresenta uma interface simples:

1. Os clientes criam um objeto Critérios especificando as características dos objetos que eles querem que sejam retornados em uma pesquisa
   - Por exemplo: Encontrar objetos de pessoa pelo nome criamos dois critérios, configurando cada critério de forma individual apropriadamente:
     - criterios.ehIgual(Pessoa.SOBRENOME, "Fowler") e
     - critérios.como(Pessoa.PRENOME, "M")

Funcionamento

2. Então chamamos **repositório.atendendo(critérios)** para retornar uma lista de objetos do domínio representando pessoas com o sobrenome **Fowler** e um prenome começando com M.

Da perspectiva do código do cliente, não existe a noção de "execução" de uma consulta. Em vez disso, ocorre a seleção dos objetos apropriados por meio da "satisfação" da especificação da pesquisa.

Funcionamento

Ele recebe um objeto com a especificação da pesquisa e retorna uma coleção com os objetos de domínio que satisfazem essa pesquisa. A especificação é um objeto.

Qualquer objeto pode ser usado, mas o ideal é utilizar o padrão *Query Object*.

- Do ponto de vista do padrão *Query Object* o *Repository* atua como um interpretador da especificação.

Padrão Query Object

Query Object (Objeto de Pesquisa) é um padrão de projeto que visa libertar o programador de conhecer e/ou usar uma linguagem de pesquisa de dados como SQL.

Martin Flowler define Query Object como um construtor de frases SQL de pesquisa com base numa estrutura de objetos.

Exemplos

Exemplo de implementação deste conceito são os Criteria do Hibernate. Por este motivo este padrão é também chamado de Criteria

Funcionamento

Neste momento poderá estar pensando que essa estratégia é equivalente a uma implementação do padrão DAO. Para deixar mais claro imaginemos que o mapeamento é entre objetos Java e um banco de dados que suporta SQL.

Enquanto o DAO apresenta métodos simplificados para inserir, apagar e atualizar dados ele ainda receberia instruções em SQL para retornar os registros no banco de dados. O *Repository* tem então duas funções:

1. Traduzir a especificação de pesquisa enviada pela camada de domínio para uma frase SQL e Passar essa frase ao DAO
2. Converter os dados retornados pela pesquisa em objetos de domínio

Resumo do Funcionamento

1. Então, ao receber uma chamada o Repositório valida isto de acordo com as regras de negócio que o cabe (já que a maior parte pode estar no próprio modelo), e pede à camada de infraestrutura esses dados.
2. A camada de infraestrutura trata essa chamada no cabe à infraestrutura (valida, encode, traduz campos...) e acessa a persistência. Ela trata o retorna da persistência no que vale a infraestrutura (array para objetos, encoding, joins...) e devolve ao repositório.
3. O repositório então trata este retorno o que diz respeito ao negócio, criando objetos, cálculos e o que for aplicável.

Quando usá-lo?

Em um sistema grande, com muitos tipos de objetos de domínio e muitas pesquisas possíveis, o Repositório reduz a quantidade de código necessário para lidar com todas as consultas que ocorrem.

Entretanto, em situações com diversas fontes de dados, é onde realmente vemos o Repositório ter sua oportunidade de mostrar o quanto é útil.

Considerações

O papel principal de um *Repository* é ser um tradutor de pesquisas, mas podemos aumentar a sua responsabilidade um pouco mais.

Na presença de uma camada de mapeamento pesquisar não é a única tarefa delegada a ela pela camada de domínio. Tarefas como inserir e apagar também são importantes.

Considerações

Por outro lado, se o repositório é visto pela camada de domínio como uma coleção em memória como essa coleção é alterada?

Se a camada de domínio utiliza diretamente a camada de mapeamento para operações de edição dos dados e o repositório para operações de pesquisa existem dois pesos e duas medidas.

- Ou bem que tratamos todas as operações CRUD diretamente com a camada de mapeamento, ou bem que não.

Fazer isso algumas vezes e não outras introduz uma incoerência na dependência das camadas que rápidamente se torna o tendão de Aquiles quando o modelo de domínio aumentar, se tornar mais complexo, ou simplesmente mudar.

Considerações

Baseado nisto o *Repository* deve também concentrar a interface para edição da coleção de dados que ele representa.

Na realidade ele apenas delegará ao mecanismo na camada de mapeamento para os casos mais simples, mas promove um ponto de entrada para traduzir os dados do domínio para a camada de mapemaneto se for necessário.

Conclusão

O que temos afinal é:

O Repositório é uma camada de negócio da aplicação, responsável por manter e persistir os objetos de Modelo, enquanto isto não envolver infraestrutura.

- Quando tratar de infraestrutura, esse trabalho é delegado aos DAO ou diretamente ao framework, caso este seja bem abstraído.

+ Algumas Diferenças entre DAO e Repository

- O repositório é um padrão da camada de domínio mais relacionado às regras de negócio e do domínio enquanto o DAO é um padrão da camada de integração mais ligado a regras de tecnologia e a protocolos de dados.
- O DAO, quando bem desenhado é uma peça de infraestrutura capaz de ser utilizado em vários sistemas diferentes, enquanto o repositório é especifico a um sistema e a um "modo de pensar"o domínio.
- O DAO pode existir e ser usado mesmo em sistemas sem camada de domínio, o repositório não.

Exemplo

Opção1

```
public class Pessoa {

    public List dependentes ( ) {
        Repositório repositório = Registro.repositórioPessoa( );
        Critérios critérios = new Critérios( );
        critérios.igual(Pessoa.BENFEITOR, this);
        return repositório.satisfazendo(critérios);
    }
```

Exemplo

Opção2

```
public class RepositórioPessoa extends Repositório {
    public List dependentesDe (Pessoa umaPessoa) {
        Critérios critérios = new Critérios( );
        critérios.igual(Pessoa.BENFEITOR, umaPessoa);
        return satisfazendo(critérios);
    }
}
```

Padrões de Projeto

Repository

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande
Divisão de Computação