Esfinge Comparison

Um framework para Comparação de Objetos

Customize o algoritmo de comparação utilizando anotações e estendendo as funcionalidades desse framework.

A comparação entre objetos da mesma classe, recuperando as diferenças entre eles, pode ser bastante útil em diferentes cenários. Por exemplo, ao se criar uma trilha de auditoria, pode ser necessário comparar a versão atual de uma entidade com outro objeto representando sua versão anterior, para registrar quais foram as propriedades modificadas. Em um formulário com muitos campos, pode ser necessário ressaltar quais foram as informações modificadas a partir da versão anterior do mesmo. Esses são apenas alguns cenários onde pode ser necessária a comparação de objeto

A primeira alternativa de implementação que se imagina é realizar a codificação do algoritmo de comparação de forma manual. Isso traz diversas desvantagens, como o fato desse código ser altamente acoplado à estrutura da classe e consequentemente suscetível a qualquer modificação no mesmo. Outra desvantagem dessa abordagem é que pelo fato de ser um código repetitivo, é comum que se cometa erros por falta de atenção. Por exemplo, um campo pode deixar de ser incluído na comparação. Além disso, a criação de código de comparação para diversas classes pode ser uma tarefa trabalhosa, especialmente se o número de atributos não é pequeno.

Uma solução alternativa à criação manual do algoritmo de comparação seria a utilização de reflexão para a criação de um algoritmo de comparação genérico. Nessa abordagem, o algoritmo percorreria as propriedades da classe comparando cada uma e retornando sua lista ao final. A desvantagem nesse caso é que o algoritmo de comparação precisa ser um só para todas as classes, o que pode ser inadequado se existirem alguns casos especiais. Por exemplo, se um campo precisar ser excluído da comparação, isso pode inviabilizar o uso da reflexão. Uma solução nesse caso seria a introdução de metadados para serem considerados na comparação, porém isso muitas vezes acabaria extrapolando a complexidade que um requisito desse tipo deveria trazer à aplicação.

O Esfinge Comparison é um framework open-source baseado em metadados que realiza a comparação de duas instâncias da mesma classe e pode ser baixado no endereço http://esfinge.sf.net. Ele disponibiliza anotações que podem ser utilizadas para customizar o algoritmo de comparação. Ele dá suporte a questão como a comparação de propriedades de dependências, o tratamento de ciclos no grafo de objetos e a comparação de listas de objetos complexos. Além disso, o framework disponibiliza diversos pontos de extensão que podem ser utilizados para especializar seu comportamento, como a criação de novas anotações, classes para leitura de metadados e novas camadas de processamento de metadados. Mas não se assuste! É muito simples utilizar esse framework, e você ainda pode estender sua funcionalidade apenas quando for necessário.

Utilizando o Esfinge Comparison

A utilização do Esfinge Comparison é bem simples! O primeiro passo consiste em configurar a classe a ser comparada com anotações para que o algoritmo de comparação possa se ajustar as suas necessidades. As anotações devem ser colocadas nos métodos getters da classe desejada. É importante ressaltar que o framework busca todas as propriedades públicas da classe, aquelas que possuem um método get, não importando se a mesma possui algum atributo privado.

Vale ressaltar que o framework possibilita a criação de novas anotações de comparação, o que será mostrado nas próximas seções do artigo. Abaixo seguem as anotações que o framework disponibiliza nativamente:

» @IgnoreInComparison: ignora o atributo anotado, não incluindo-o na comparação.

Eduardo Guerra | guerra@mundoj.com.br | @emguerra

Desenvolvedor de frameworks, pesquisador em design de software, editor-chefe da revista MundoJ e professor do ITA, onde concluiu sua graduação, mestrado e doutorado. Possui diversas certificações da plataforma Java e experiência como arquiteto de software para softwares desktop, corporativos e móveis. Participou de projetos open-source, como SwingBean, ClassMock e Esfinge, e acredita que um bom software se faz mais com criatividade do que com código.

A implementação da comparação entre duas instâncias de uma classe normalmente é uma lógica que é implementada manualmente para cada classe. Nessa criação, justamente por ser um código repetitivo, é comum acontecer erros por faltas de atenção. Regras de comparação específicas de alguns objetos podem impedir que uma solução mais inteligente utilizando reflexão seja utilizada. O framework Esfinge Comparison permite que o algoritmo de comparação de cada classe seja customizado utilizando anotações. Esse artigo mostra como utilizar e estender esse framework, que pode ser uma importante ferramenta para se ter a mão.

Deve ser utilizado para campos que normalmente não são persistidos ou não importantes para o negócio da aplicação.

- @DeepComparison: executa o algoritmo de comparação na propriedade, realizando a comparação de suas propriedades internas. Deve ser utilizado para propriedades que representam o relacionamento entre classes de negócio que deve ser incluído na comparação.
- @Tolerance: configura a tolerância numérica permitida para o campo. Deve ser usado em campos do tipo ponto flutuante onde podem haver diferenças de arredondamento.
- @CompareSubstring(begin, end): compara apenas parte da string iniciada no atributo begin e terminada no atributo end da anotação. Deve ser usado em propriedades onde apenas parte do texto de uma string é relevante na comparação.

A Listagem 1 apresenta um exemplo de classe anotada com as anotações do Esfinge Comparison. A anotação @CompareSubstring foi utilizada na propriedade nomeTitulo, para que a parte inicial de strings como "Dr. Fulano" fosse ignorada na comparação. A propriedade cpf não recebeu nenhuma anotação, logo será comparada usando o método equals() do tipo da propriedade. A propriedade dataComparacao, por não apresentar uma propriedade propriamente dita do cliente, foi retirada da comparação utilizando a anotação @IgnoreInComparison. Finalmente, a propriedade altura, por usar um tipo baseado em ponto flutuante, está propensa a diferenças de arredondamento e é importante definirmos a tolerância numérica desejada com a anotação @Tolerance.

Como a classe Empresa, apresentada na Listagem 2, é um tipo complexo, que também possui propriedades, o método getEmpresa() na Listagem 1 foi anotado com @DeepComparison. Dessa forma, quando o framework for realizar uma comparação dessa propriedade, ele executará o algoritmo de comparação nas propriedades dela de forma recursiva, comparando uma a uma. Apesar de não ter sido utilizado no exemplo, todas as anotações podem ser utilizadas normalmente nas classes que compõe a classe principal da comparação.

Listagem 1. Classe anotada com as anotações de comparação.

```
public class Cliente {

    private String nomeTitulo;
    private String cpf;
    private Date dataModificacao;
    private Empresa empresa;
    private double altura;

    public Cliente(String nomeTitulo, String cpf, Date
dataModificacao, double altura) {
        this.nomeTitulo = nomeTitulo;
        this.cpf = cpf;
        this.dataModificacao = dataModificacao;
        this.altura = altura;
    }
    @CompareSubstring(begin=3)
    public String getNomeTitulo() {
        return nomeTitulo;
    }
```

```
    public void setNomeTitulo(String nomeTitulo) {
        this.nomeTitulo = nomeTitulo;
    }
    public String getCpf() {
        return cpf;
    }
    public void setCpf(String cpf) {
        this.cpf = cpf;
    }
    @IgnoreInComparison
    public Date getDataModificacao() {
        return dataModificacao;
    }
  public void setDataModificacao(Date
dataModificacao) {
        this.dataModificacao = dataModificacao;
    }
    @DeepComparison
    public Empresa getEmpresa() {
        return empresa;
    }
    public void setEmpresa(Empresa empresa) {
        this.empresa = empresa;
    }
    @Tolerance(0.1)
    public double getAltura() {
        return altura;
    }
    public void setAltura(double altura) {
        this.altura = altura;
    }
}
```

Listagem 2. Classe Empresa que compõe a classe principal.

```
public class Empresa {

    private String nomeFantasia;

    private String cnpj;
    private String endereco;

    public Empresa(String nomeFantasia, String cnpj,
    String endereco) {
        super();
        this.nomeFantasia = nomeFantasia;
        this.cnpj = cnpj;
        this.endereco = endereco;
    }

    //getters e setter omitidos
}
```

Para exemplificar a utilização do componente de comparação, a Listagem 3 apresenta um código que faz a comparação de duas instâncias da classe Cliente. A uma instância da classe ComparisonComponent deve ser utilizada para realizar a comparação. Ao invocar o método compare() passando duas instâncias da mesma classe, é retornada uma lista de diferenças da comparação, cada uma representada por uma instância da classe Difference. Caso instâncias de classes diferentes sejam passadas para esse método, será lançada uma CompareException.

Listagem 3. Utilizando o componente de comparação.

```
public class Principal {

    public static void main(String[] args) throws
CompareException {
        Cliente c1 = new Cliente(
          "Sr. Guerra","11111111111",new Date(),1.845);
        Empresa e1 = new Empresa("MundoJava",
            "99999/0001-99","Rua Detraz, 34");
        c1.setEmpresa(e1);

        Cliente c2 = new Cliente("Dr. Guerra",
            "11111111211",new Date(),1.842);
        Empresa e2 = new Empresa("MundoJ",
            "99999/0001-99","Rua Detraz, 34");
        c2.setEmpresa(e2);
              ComparisonComponent c =
            new ComparisonComponent();
        List<Difference> difs = c.compare(c2, c1);

        for(Difference d : difs){
            System.out.println(d);
        }
    }
}
```

Segundo os valores passados para o algoritmo de comparação na Listagem 3 e as configurações realizadas na Listagem 1, apenas duas diferenças devem ser retornadas. Da comparação da propriedade nomeTitulo, apesar das strings serem diferentes, não será acusada diferença, pois os metadados da classe configuram que a comparação seria apenas a partir do terceiro caractere. Similarmente, apesar das alturas também serem diferentes, a diferença entre elas é menor que o configurado na anotação @Tolerance e os valores serão considerados iguais. Vale também ressaltar a diferença que será encontrada na propriedade nomeFantasia dentro da classe Empresa, configurada com @DeepComparison. Segue a saída do console resultante da execução do código da Listagem 3:

```
empresa.nomeFantasia : MundoJ / MundoJava
cpf : 11111111211 / 11111111111
```

Criando as próprias anotações

Porém uma das funcionalidades mais interessantes do Esfinge Comparison não está nas anotações que ele possui, mas na possibilidade de criar novas anotações de comparação. Imagine que no exemplo apresentado fosse introduzida uma string na propriedade nomeTitulo com o seguinte valor: "Prof.

Guerra". Nessa propriedade, os três primeiros caracteres estão sendo ignorados na comparação devido ao fato do título não precisar ser incluído. Pelo exemplo apresentado é possível perceber que essa abordagem não é suficiente para cumprir esse requisito, pois os títulos podem possuir tamanhos diferentes e ter mais de três caracteres.

Como o Esfinge Comparison não possui uma anotação que configura a comparação de uma propriedade para ser dessa forma, vamos criar nossa própria anotação, conforme apresentado na Listagem 4. Para que ela seja entendida pelo framework como uma anotação de comparação, ela precisa possuir a anotação @DelegateReader, indicando a classe que será utilizada para a leitura dessa anotação, no caso a classe CompareAfterCharacterReader, apresentada na Listagem 5.

Listagem 4. Definição de nova anotação de comparação.

```
@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@DelegateReader(CompareAfterCharacterReader.
class)
public @interface CompareAfterCharacter {
    char value();
}
```

Para a leitura da anotação, a classe precisa implementar a interface AnnotationReader, que deve receber como tipo genérico o tipo da própria anotação. O método readAnnotation(), que será invocado pelo framework para esse propósito, recebe como parâmetro a anotação e uma instância da classe PropertyDescriptor. Essa classe possui as informações a respeito de como a comparação deve ser realizada. A propriedade processor de PropertyDescriptor armazena uma instância do tipo ComparisonProcessor para o qual o framework delega a comparação da propriedade. Dessa forma, é possível criar a nossa implementação e colocar no descritor para o uso do framework.

Listagem 5. Classe que lê a anotação.

```
public class CompareAfterCharacterReader
implements
AnnotationReader<CompareAfterCharacter>{

    @Override
    public void readAnnotation(CompareAfterCharacter
    annotation,PropertyDescriptor desc) {
        char c = annotation.value();
        AfterCharProcessor proc =
            new AfterCharProcessor(c);
        desc.setProcessor(proc);
    }

}
```

Referências circulares

É comum em modelos de domínio que existam relacionamentos bidirecionais ou referências circulares. Em um algoritmo de comparação em que se entra dentro das dependências e compara sua propriedades de forma recursiva, isso pode levar a um loop infinito. Para evitar que isso aconteça, o Esfinge Comparison armazena o caminho percorrido naquele ramo de comparação. Caso um objeto encontrado já exista naquele ramo, ele será ignorado e a comparação irá continuar nas outras propriedades. Caso o mesmo objeto seja encontrado em um ramo diferente, como no caso da mesma instância ser setada em propriedades diferentes, a comparação continuará normalmente.
Como cada componente armazena internamente o ramo onde a comparação se encontra, não é recomendável que a mesma instância seja utilizada para realizar comparações em paralelo. Se isso for necessário, uma instância deve ser criada para cada thread.

A Listagem 6 apresenta o código da classe que implementa ComparisonProcessor e que possui o algoritmo de comparação. O método compare() recebe como parâmetro o nome da propriedade e os valores a serem comparados. No exemplo, o algoritmo de comparação recupera a string depois do caractere configurado e só então realiza a comparação. Note que no caso de haver uma diferença, a classe precisa retornar uma das implementações de Difference. No caso foi retornado um PropertyDifference, que representa uma diferença simples entre propriedades.

Listagem 6. Classe que executa a comparação.

```
public class AfterCharProcessor implements
ComparisonProcessor {

    private char c;

    public AfterCharProcessor(char c) {
        this.c = c;
    }
    @Override
    public Difference compare(String prop, Object oldVal,
    Object newVal) {
        String oldStr = oldVal.toString().substring(oldVal.
            toString().indexOf(c)+1);
        String newStr = newVal.toString().substring(newVal.
            toString().indexOf(c)+1);
        if(!oldStr.equals(newStr))
```

```
            return new PropertyDifference(prop,
                newStr, oldStr);
        return null;
    }
}
```

Depois de criada a anotação e as duas classes, é preciso apenas utiizá-la na propriedade desejada. O trecho de código a seguir mostra como ela seria utilizada no método getter:

```
@CompareAfterCharacter('.')
public String getNomeTitulo() {
    return nomeTitulo;
}
```

Lendo metadados de outras fontes

O Esfinge Comparison possui como principal forma de definição de metadados suas anotações próprias, porém ele provê um ponto de extensão que permite que outras fontes sejam utilizadas. Por exemplo, uma definição de informações em um documento XML ou outra base de dados externa pode ser muito útil quando não se tem acesso ao código das classes que se deseja comparar. Outro exemplo em que isso pode ser utilizado, é para pegar informações a partir de anotações de outros frameworks e APIs.

A Listagem 7 apresenta a interface que precisa ser implementada para a criação de um novo leitor de metadados. O método populateContainer() recebe a classe para a qual se deseja ler os metadados e uma instância de ComparisonDescriptor, responsável por armazenar em tempo de execução os metadados lidos. Ao ser implementado, esse método deve ler os metadados de alguma fonte e inseri-los no ComaprisonDescriptor.

Listagem 7. Interface para implementação de novo leitor de metadados.

```
public interface ComparisonMetadataReader {
    public abstract void populateContainer(Class c,
        ComparisonDescriptor descriptor);
}
```

Um leitor de metadados opcional que já possui uma implementação é a classe JPAComparisonMetadataReader. Essa classe busca anotações da API JPA na classe visando a configuração do algoritmo de comparação. A Listagem 8 apresenta como esse leitor pode ser configurado para realizar essa leitura de forma adicional as anotações do Esfinge Comaprison. Caso um novo leitor seja criado, ele pode ser configurado de forma similar.

Listagem 8. Incluindo um novo leitor na cadeia de leitura de metadados.

```
ChainComparisonMetatataReader chainReader =
        new ChainComparisonMetatataReader(
        new AnnotationComparisonMetadataReader(),
        new JPAComparisonMetadataReader()
        );
        MetadataReaderProvider.set(chainReader);
```

A seguir estão descritas as anotações consideradas pelo framework e qual o seu efeito no algoritmo de comparação:

- @Transiente: como mudanças em campos transientes não são persistidas, elas não representam uma mudança de caráter permanente. Por esse motivo, as propriedades com essa anotação são ignoradas na comparação.
- @Entity: quando uma classe é uma entidade, o Esfinge Comaprison entende que se ela for uma propriedade de uma outra entidade, deve ser feita uma comparação profunda, comparando todos os seus campos.
- @Id e @EmbededId: são utilizados para a comparação de listas de objetos complexos, como será descrito na próxima seção do artigo.

Comparação de Listas

Quando uma propriedade que precisa ser comparada é uma lista, não basta dizer apenas se a lista é igual ou diferente. As diferenças devem refletir quais elementos foram adicionados e removidos. Adicionalmente, quando a lista for de objetos complexos, é importante saber se alguma propriedade de um elemento foi alterada. O framework Esfinge Comparison realiza a comparação de lista para todo atributo que for do tipo Collection, ou um de seus subtipos, como Set ou List.

O framework suporta a comparação de listas simples e de objetos complexos. Se nenhuma configuração for realizada, a comparação de listas simples será utilizada. Nesse caso, um novo item que não estiver na lista antiga será considerado uma adição e um item da lista antiga que não estiver mais na lista nova será considerado uma exclusão. É importante ressaltar que nesse caso, o método equals() será utilizado para as comparações, e ele precisa ser implementado de forma apropriada.

Para a comparação ser de listas de tipos complexos, a propriedade deve estar marcada com @DeepComparison ou o tipo genérico da coleção deve utilizar @Entity. Nesse caso, o framework busca na classe uma propriedade anotada com @Id (pode ser a anotação do JPA ou do próprio framework) ou @EmbededId. Essa propriedade será utilizada na identificação da entidade no lugar do método equals(). Entidade com valor de identificador nulo são sempre consideradas adições. Quando as propriedades pos-

Tipos de diferenças

O framework Esfinge Comparison possui dois tipos de diferenças, sendo que ambos são subtipos de Difference. A classe PropertyDifference representa uma diferença entre propriedades simples. Nesse tipo de diferença são aguardadas informações como o novo valor e o valor antigo. O outro tipo de diferença é ListChangeDifference, que representa uma adição ou exclusão na lista. Nesse caso, a diferença possui o item referenciado e a informação se foi adicionado ou removido. Fazendo o cast de Difference para a classe apropriada, é possível recuperar as informações específicas de cada diferença.

suírem o mesmo identificador, elas são submetidas ao algoritmo de comparação, possuindo suas propriedades comparadas uma a uma.

Exemplo com anotações de JPA e comparação de listas

Para fechar o artigo, será mostrado um exemplo de comparação envolvendo a comparação de listas e o uso de anotações de JPA. No cenário que será utilizado, imagine um sistema de agenda, onde o usuário pode sincronizar as informações com outras fontes. Nesse processo, devem ser detectadas as diferenças entre a versão local e a da outra fonte para atualizar as mudanças.

As Listagens 9 e 10 apresentam as entidades que serão utilizadas no exemplo utilizando as anotações do JPA. Observe que somente as anotações utilizadas pelo framework foram incluídas por uma questão de clareza. Observe que a classe Contato possui um atributo transiente referente ao número de ligações realizadas que deve ser ignorado na comparação. Adicionalmente ele possui uma lista do tipo Telefone, que por possuir a anotação @Entity configura a comparação da lista de objetos complexos.

Listagem 9. Entidade a ser comparada.

```
@Entity
public class Contato {

    @Id
    private int id;
    private String nome;
    private List<Telefone> telefones;
    @Transient
    private int ligacoesRealizadas;

    public Contato(int id, String nome, int
    ligacoesRealizadas) {
        this.id = id;
        this.nome = nome;
        this.telefones =  new ArrayList<>();
        this.ligacoesRealizadas = ligacoesRealizadas;
    }
    public void addTelefone(Telefone t){
        telefones.add(t);
    }
    //Outros métodos getters e setters omitidos
}
```

Listagem 10. Entidade que representa um item da lista da classe a ser comparada.

```
@Entity
public class Telefone {

    @Id
    private int idTel;
    private String ddd;
    private String numero;
    private String tipo;

    public Telefone(int idTel, String ddd, String numero,
    String tipo) {
        this.idTel = idTel;
        this.ddd = ddd;
        this.numero = numero;
        this.tipo = tipo;
    }

    @Override
    public String toString() {
        return "[" + idTel + ", " + ddd + ", "+ numero
+ ", " +
            tipo + "]";
    }
}
```

A Listagem 11 apresenta o código que faz a comparação entre duas instâncias da classe Contato. Um bloco estático configura o leitor de anotações do JPA no framework. Em seguida, o método main() realiza a comparação de duas instâncias da classe Contato. Observe que pelas propriedades configuradas, três diferenças devem ser encontradas: (a) o telefone com id 223 foi removido; (b) o telefone com id 224 foi adi-

cionado; e (c) a propriedade número do telefone com id 222 foi modificada. Observe que a propriedade ligacoesRealizadas, apesar de possuir valores diferentes, deve ser ignorada devido a possuir a anotação @Transient. Segue abaixo o resultado da execução desse código:

```
telefones[id=222].numero : 555-4321 / 555-1234
telefones[id=223] - ADDED - [223, 11, 9555-1234, CELULAR]
telefones[id=224] - REMOVED - [224, 11, 9555-1111, TRABALHO]
```

Listagem 11. Realização da comparação.

```
public class Exemplo {

    static {
        ChainComparisonMetatataReader chainReader =
            new ChainComparisonMetatataReader(
                new AnnotationComparisonMetadataReader(),
                new JPAComparisonMetadataReader());
        MetadataReaderProvider.set(chainReader);
    }

    public static void main(String[] args) throws
    CompareException {
        Contato c1 = new Contato(27, "João Pedro", 7);
        c1.addTelefone(new Telefone(222, "11",
            "555-1234",
        Contato c2 = new Contato(27, "João Pedro", 9);
        c2.addTelefone(new Telefone(222, "11",
            "555-4321",
            "RESIDENCIAL"));
        c2.addTelefone(new Telefone(224, "11",
             "9555-1111",
            "TRABALHO"));

        ComparisonComponent c =
            new ComparisonComponent();
        List<Difference> difs = c.compare(c2, c1);

        for (Difference d : difs) {
            System.out.println(d);
        }
    }
}
```

Considerações finais

Este artigo apresentou o Esfinge Comparison, um framework que tem o objetivo de simplificar a comparação de instâncias da mesma classe. Ele possui várias funcionalidades avançadas, como a customização do algoritmo de comparação utilizando metadados, tratamento de ciclos no grafo de objetos e comparação de listas complexas. Além dessas funcionalidades, o framework provê diversos pontos de extensão que permitem sua adaptação para necessidades mais específicas de cada aplicação.

Gostaria de fechar o artigo ressaltando que o Esfinge Comparison é um framework de código aberto, possui uma boa quantidade de testes automatizados, possui um documentação bem completa no site do projeto e já foi utilizado em projetos que se encontram hoje em produção. Apesar de ser de um domínio bem específico, é uma excelente ferramenta para se ter na manga quando esse tipo de funcionalidade for necessária. Deixo o convite para todos visitarem o site do projeto Esfinge e acessarem esse e os outros frameworks que estão disponíveis para download!

Agradecimentos

Aproveito a oportunidade para agradecer a empresa GSW Soluções Integradas pelo apoio que tem dado ao projeto Esfinge! Além de participar de seu desenvolvimento, a GSW ainda hospeda e mantém o portal do projeto. Espero que essa atitude sirva de exemplo para outras empresas brasileiras também investirem em projetos de código aberto.

/para saber mais

A revista MundoJ 52 apresentou outro framework que faz parte do projeto Esfinge, o Esfinge QueryBuilder. Esse framework gera consultas a partir da assinatura dos métodos de uma interface e possui a mesma filosofia do Esfinge Comparison: utilizar os metadados da classe para simplificar o desenvolvimento de aplicações.

A edição 34 da MundoJ trouxe o artigo "Padrões de Projeto para Componentes que Utilizam Metadados" que apresentou algumas das práticas de design que são utilizadas no Esfinge Comparison. O artigo utiliza como exemplo uma versão bem simplificada de um componente de comparação que acabou sendo a origem desse framework.

/referências

> Projeto Esfinge – http://esfinge.sf.net

> Guerra, E. M. ; Souza, J. T. ; Fernandes, C. T. . A Pattern Language for Metadata-based Frameworks. In: 16th Conference on Pattern Languages of Programs, 2009, Chicago.