Inversão de Controlo & Injeção de Dependências

José Morgado
pg27759@alunos.uminho.pt

Luís Miguel Pinto
pg27756@alunos.uminho.pt

Pedro Carneiro
pg25324@alunos.uminho.pt

Resumo

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

I. Introdução

Tem-se assistido a um crescimento da complexidade das arquiteturas dos sistemas de software, sendo cada vez mais comum a integração de múltiplos componentes. Hoje em dia, ninguém desenvolve aplicações web sem recorrer a frameworks e web services.

De modo a que isto seja possível, precisamos de técnicas que permitam garantir a modularidade e extensibilidade dos componentes de software. Foi neste contexto que surgiram os conceitos de Inversão de Controlo e Injeção de Dependências.

II. Inversão de controlo

A Inversão de Controlo é um princípio de concepção de software. Tipicamente, quando queremos construir um programa apenas codificamos parte do mesmo e recorremos a biblitecas para realizar parte das tarefas; neste caso, o nosso código (específico) depende de código externo (reutilizável). Usando Inversão de Controlo, a abordagem é a oposta: o código reutilizável das bibliotecas é que vai necessitar/depender do nosso código específico.

Um exemplo disso mesmo são as frameworks. Estas coordenam a execução das aplicações, o que nos permite ter que definir apenas parte do comportamento das mesmas. Ou seja, funcionam como esqueletos aos quais apenas temos de disponibilizar os plugins necessários.

Há várias técnicas que seguem este princípio, como a Injeção de Depedências, que será apresentada na secção seguinte.

III. Injeção de Dependências

O *pattern* de injeção de dependências é uma técnica que pretende diminuir as junções entre classes tornando assim a evolução do *software* mais suave e fácil. É uma das formas de se fazer Inversão de Controlo, descrita na secção anterior.

Existem três formas principais de se fazer injeção de dependências, através de *Constructor Injection*, *Setter Injection* e *Interface Injection* ou por um *Service Locator*.

Nas próximas subseções iremos abordar es-

tes métodos, apresentando algumas soluções.

I. Constructor Injection

Neste método, as dependências do objeto são injetadas diretamente no construtor. Depois da classe *Stand* ser instanciada é possível atribuir à variável de instância *veiculo* o objeto do qual ela depende.

Exemplo 1: *Injeção pelo Construtor*

```
public class Stand {
  private Veiculo veiculo;

  public Stand(Veiculo veiculo) {
    this.veiculo = veiculo;
  }
}
```

II. Setter Injection

Neste procedimento é utlizado o método *setter*, *setVeiculo*, que permite injectar através dos argumentos de entrada do método as dependências para a variável de instância do objeto.

Exemplo 2: *Injeção pelo Setter*

```
public class Stand {
  private Veiculo veiculo;

  @Inject
  @Required
  public void setVeiculo(Veiculo veiculo) {
    this.veiculo = veiculo;
  }
}
```

III. Interface Injection

Desenhou-se uma interface onde os métodos *setters* aceitam dependências. Então quando se implementa esta interface na classe *Stand* é necessário definir os métodos da interface e através do método *setVeiculo* é possível fazer injeção de dependências nos objetos da classe.

Exemplo 3: *Injeção pela Interface*

```
public interface injectVeiculo {
  public void setVeiculo(Veiculo veiculo);
}

public class Stand implements
    injectVeiculo {
  private Veiculo veiculo;

  public void setVeiculo(Veiculo veiculo) {
    this.veiculo = veiculo;
  }
}
```

IV. Service Locator

A principal idéia por trás de um *Service Locator* é ter um objeto que saiba adquirir todos os serviços da aplicação quando precisar. Assim, o *Service Locator* para o exemplo que se segue tem um método que retorna uma referência do *veiculo* quando é necessário.

Exemplo 4: *Injeção por Service Locator*

```
public class ServiceLocator {
  public static Veiculo getVeiculo() {
    return soleInstance.veiculo;
  }

  private static ServiceLocator
      soleInstance;
  private Veiculo veiculo;
}

public class Stand {
  private Veiculo veiculo;

  public Stand() {
    this.veiculo =
        ServiceLocator.getVeiculo();
  }
}
```

Referências

[1] http://en.wikipedia.org/wiki/Inversion_of_control

[2] http://martinfowler.com/bliki/InversionOfControl.html

[3] https://en.wikipedia.org/wiki/Dependency_injection

[4] http://martinfowler.com/articles/injection.html

[5] http://www.devmedia.com.br/padrao-de-injecao-de-dependencia/18506

[6] http://stackoverflow.com/questions/21218868/setter-injection-vs-constructor-injection