Dependency Injection

Átila Camurça

24 de abril de 2014

Sumary

Introdução

O conceito de Framework agrega além de códigos pré-prontos a utilização de Padrões de Projeto. Um desses padrões, que nasceu junto com as Frameworks é o Inversão de Controle, que mais tarde foi refinado por Martin Fowler e ganhou o nome de Injeção de Dependência ou Dependency Injection (DI).

`http://www.martinfowler.com/articles/injection.html`

Definição

Dependency Injection significa dizer que os componentes de uma classe não necessitam procurar por suas dependências num contexto global ou usando localizadores de serviços. Ao invés disso suas dependências são simplesmente injetadas na hora da criação e seu **Criador** é o responsável por provê-las.

Definição

Para isso são necessários três elementos:

- um **componente dependente**,
- uma **declaração das dependências** do componente e
- um **componente injetor** que provê o componente dependente com suas dependências.

Vantagens

Uma das razões que levaram a criação desse padrão é a necessidade de testes automatizados.

Usando **DI** você é capaz de criar ambientes para teste e produção devido ao **baixo acoplamento** proporcionado. Isso porque o padrão conta com um **Criador** o qual configura o ambiente adequado injetando as depedências necessárias.

Quem utiliza?

Figura: Zend Framework 2 (PHP)

Quem utiliza?

Figura: Spring (Java)

Quem utiliza?

Figura: AngularJS (Javascript)

Exemplo

Usando AngularJS vamos criar uma espécie de *playlist* com músicas de vários albúns no qual nós somos capazes de buscar por qualquer música.

Ver: github.com/atilacamurca/press-di/tree/master/playlist

Como isso funciona?

```
function Injector() {
  this.factories = {};
  this.services = {};
}

Injector.prototype.addService = function (key, factory) {
  this.factories[key] = factory;
};
```

Como isso funciona?

```
Injector.prototype.getService = function (key) {
  var service = this.services[key];
  if (!service) {
    var factory = this.factories[key];
    service = factory();
    this.services[key] = service;
  }
  return service;
};
```

Como isso funciona?

```
Injector.prototype.create = function (Constructor) {
  var Dependant = function () {};
  Dependant.prototype = Contructor.prototype;
  var instance = new Dependant();
  this.inject(Constructor, instance);
  return instance;
};

Injector.prototype.inject =
    function (Constructor, instance) {
  var keys = Constructor.prototype.$deps || [];
  var deps = keys.map(this.getService, this);
  Constructor.apply(instance, deps);
};
```

Como isso funciona?

```
function UserPageController(user_repo) {
  this.user_repo = user_repo;
}

UserPageController.prototype.$deps = [ 'user_repo' ];

function UserRepository() { /* ... */ }
```

Como isso funciona?

```
var injector = new Injector();
injector.addService('user_repo', function () {
  return new UserRepository();
});

var controller = injector.create(UserPageController);
```

Como isso funciona?

O método `map` da classe `Array` funciona assim:

```
var numbers = [1, 4, 9];
var roots = numbers.map(Math.sqrt);
/* roots is now [1, 2, 3], numbers is still [1, 4, 9] */
```

Links

- http://framework.zend.com/manual/2.0/en/modules/zend.di.introduction.html
- http://javafree.uol.com.br/artigo/871453/
- http://pt.wikipedia.org/wiki/Inje%C3%A7%C3%A3o_de_depend%C3%AAncia
- https://docs.angularjs.org/guide/di
- http://martinfowler.com/articles/injection.html
- http://googletesting.blogspot.com.br/2008/10/dependency-injection-myth-reference.html
- http://misko.hevery.com/2008/09/10/where-have-all-the-new-operators-gone/
- http://ionicframework.com/blog/angular-mobile-dev/
- https://github.com/btford/briantford.com/blob/master/jade/blog/understanding-di.md