Inyección de dependencias en **Plugcore**

Plug Soluciones TIC S.L.
info@plugcore.com


https://github.com/plugcore/plugcore


https://www.linkedin.com/company/plugcore


https://twitter.com/plugcoreit

Inyección de dependencias

1. **¿Qué es?**
2. **Ejemplo**
3. **Container**
4. **Decorators**
5. **Opciones**

¿Que és?

- Es un sistema que se encarga de gestionar el orden en el que se crean objetos y como se inyectan entre ellos.

- La idea es tener clases que se encarguen de sólo un concepto, por ejemplo, una clase que se encarga sólo de gestionar la conexión con la base de datos, y otra que utilizando la clase anterior, se encarga de hacer peticiones a la base de datos a una tabla determinada.

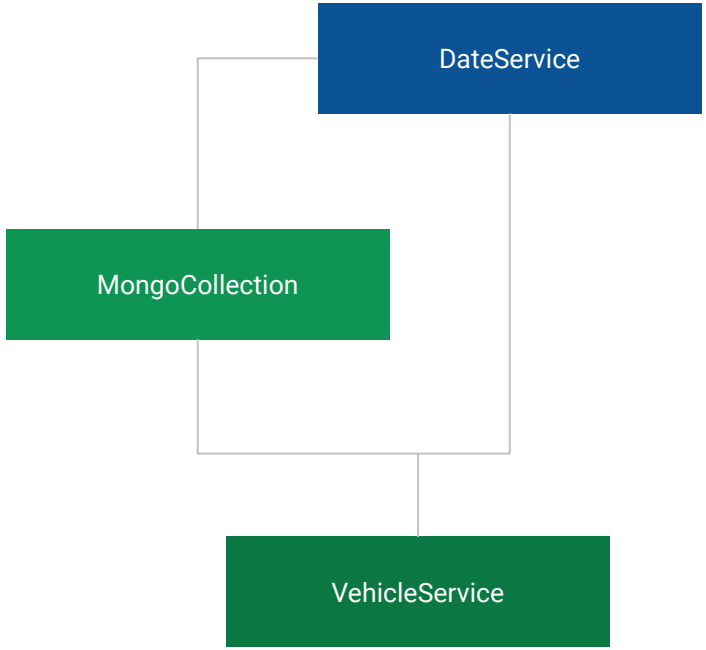
- Estas las llamamos “Service”, y estos servicios son creados y controlados por otra clase llamada “Container”.

- Lo que va a hacer el container es primero de todo leer todos los service, crear un árbol de dependencias, e ir creando los servicios en el orden correcto.

Ejemplo

- En este ejemplo vemos que “VehicleService” depende de “MongoCollection” y “DateService”
- Además “MongoCollection” depende de “DateService”
- Así que el orden de creación es:

1. DateService
2. MongoCollection <- DateService
3. VehicleService <- MongoCollection, DateService



Container

- Container es el encargado de manejar todas las dependencias, para ello primero registra todos los servicios, y después en el momento en el que alguien los pide, se encarga de crear todas las dependencias que necesite ese servicio.

```
class DateService {
    public sayHi() { return '1. This is DateService' }
}
class MongoCollection {
    constructor(dateService: DateService) { console.log(dateService.sayHi()); }
    public sayHi() { return '2. This is MongoCollection' }
}

Container.registrerService({ id: DateService, clazz: DateService });
Container.registrerService({ id: MongoCollection, clazz: MongoCollection, params: [DateService] });
Container.get(MongoCollection).then(mongoCollection => {
    console.log(mongoCollection.sayHi())
});
```

Decorators

- Los decorators de DI(Dependency injection) nos ayudan a la hora de hacer la configuración que necesita el container para gestionar las dependencias. Mismo ejemplo de antes pero con decorators:

```
@Service()
class DateService {
    public sayHi() { return '1. This is decorated DateService' }
}
@Service()
class MongoCollection {
    constructor(dateService: DateService) { console.log(dateService.sayHi()); }
    public sayHi() { return '2. This is decorated MongoCollection' }
}
Container.get(MongoCollection).then(mongoCollection => {
    console.log(mongoCollection.sayHi())
});
```

Opciones

- Además de encargarse del orden correcto de dependencias, existe:
    - **OnInit**: Nos permite ejecutar un método asíncrono justo en el momento en el que ya están todas las dependencias listas pero antes de ser inyectado en otro servicio
    - **Service id:** Además de las clases, podemos crear servicios con identificadores como strings.
    - **Context**: Por defecto todos los servicios están en un contexto global, pero se pueden tener grupos de servicios separados entre ellos.
    - **Variations**: Es un sistema que permite pasarle un objeto de configuración a la dependencia que estás usando.
- Más información en: https://github.com/plugcore/plugcore/wiki/Inyecci%C3%B3n-de-dependencias

GRACIAS POR SU TIEMPO

Para más información
info@plugcore.com
germanml@plugcore.com
sergiolc@plugcore.com

plugcore.com