[HDDT-Intern] Tìm hiểu khái niệm DI, Service – Mai Trung Tiến

Dependency inversion principle

Service object design pattern

# Tìm hiểu khái niệm DI, Service

## Tìm hiểu khái niệm Dependency inversion principle

Là một phương pháp cụ thể cho các mo-dun phần mềm được kết nối lỏng lẻo. Khi tuân theo nguyên tắc này, các mối quan hệ phụ thuộc thông thường được thiết lập từ các mô-đun thiết lập chính sách cấp cao đến các mô-đun phụ thuộc cấp thấp sẽ bị đảo ngược, do đó hiển thị các mô-đun cấp cao độc lập với các chi tiết triển khai mô-đun cấp thấp.

Nguyên tắc nêu rõ:

* Các mô-đun cấp cao không được nhập bất cứ thứ gì từ các mô-đun cấp thấp. Cả hai nên phụ thuộc vào sự trừu tượng (ví dụ: giao diện).
* Sự trừu tượng không nên phụ thuộc vào chi tiết. Chi tiết (triển khai cụ thể) sẽ phụ thuộc vào sự trừu tượng.

Bằng cách quy định rằng cả đối tượng cấp cao và cấp thấp đều phải phụ thuộc vào cùng một sự trừu tượng.

Dependency inversion principle được coi là một khái niệm duy nhất cần được khái quát hóa, tức là áp dụng cho tất cả các giao diện giữa các mô-đun phần mềm.

* Sẽ đơn giản hơn khi xem một nguyên tắc tư duy tốt như một mẫu mã hóa. Khi một lớp trừu tượng hoặc một giao diện đã được mã hóa.
* Bởi vì nhiều công cụ kiểm tra đơn vị dựa vào tính kế thừa để thực hiện mô phỏng , nên việc sử dụng giao diện chung giữa các lớp (không chỉ giữa các mô-đun khi sử dụng tính tổng quát là hợp lý) đã trở thành quy tắc.

## Tìm hiểu khái niệm Dependency Injection (DI)

Dây là một cách để hiện thực Inversion of Control Pattern (Có thể coi nó là một design pattern riêng). Các module phụ thuộc (dependency) sẽ được inject vào module cấp cao.

Có thể hiểu Dependency Injection một cách đơn giản như sau:

* Các module không giao tiếp trực tiếp với nhau, mà thông qua interface. Module cấp thấp sẽ implement interface, module cấp cao sẽ gọi module cấp thấp thông qua interface.
* Việc khởi tạo các module cấp thấp sẽ do DI Container thực hiện.
* Việc Module nào gắn với interface nào sẽ được config trong code hoặc trong file XML.
* DI được dùng để làm giảm sự phụ thuộc giữa các module, dễ dàng hơn trong việc thay đổi module, bảo trì code và testing.

### Các dạng của DI

* Constructor Injection: Các dependency sẽ được container truyền vào (inject vào) 1 class thông qua constructor của class đó. Đây là cách thông dụng nhất.
* Setter Injection: Các dependency sẽ được truyền vào 1 class thông qua các hàm Setter.
* Interface Injection: Class cần inject sẽ implement 1 interface. Interface này chứa 1 hàm tên Inject. Container sẽ injection dependency vào 1 class thông qua việc gọi hàm Inject của interface đó. Đây là cách rườm rà và ít được sử dụng nhất.

### Ưu nhược điểm của DI

Ưu điểm

* Giảm sự kết dính giữa các module
* Code dễ bảo trì, dễ thay thế module
* Rất dễ test và viết [Unit Test](https://toidicodedao.com/2015/08/25/tutorial-viet-unit-test-trong-c-voi-nunit/)
* Dễ dàng thấy quan hệ giữa các module (Vì các dependecy đều được inject vào constructor)

Nhược điểm

* Khái niệm DI khá “khó tiêu”, các developer mới sẽ gặp khó khăn khi học
* Sử dụng interface nên đôi khi sẽ khó debug, do không biết chính xác module nào được gọi
* Các object được khởi tạo toàn bộ ngay từ đầu, có thể làm giảm performance
* Làm tăng độ phức tạp của code

## Khái niệm Service Objects Design Pattern

Link tham khảo.

<https://en.wikipedia.org/wiki/Dependency_inversion_principle>  
<https://en.wikipedia.org/wiki/Dependency_injection>  
<https://toidicodedao.com/2015/03/24/solid-la-gi-ap-dung-cac-nguyen-ly-solid-de-tro-thanh-lap-trinh-vien-code-cung/>

<https://toidicodedao.com/2015/11/03/dependency-injection-va-inversion-of-control-phan-1-dinh-nghia/>

<https://toidicodedao.com/2015/11/10/dependency-injection-va-inversion-of-control-phan-2-ap-dung-di-vao-code/>

<https://toidicodedao.com/2015/11/12/dependency-injection-va-inversion-of-control-phan-3-di-container-ap-dung-di-vao-asp-net-mvc/>