عنوان: **الگوی معکوس سازی کنترل چیست؟**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۲۵ ۲۲:۴۸
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns, Dependency Injection, IoC

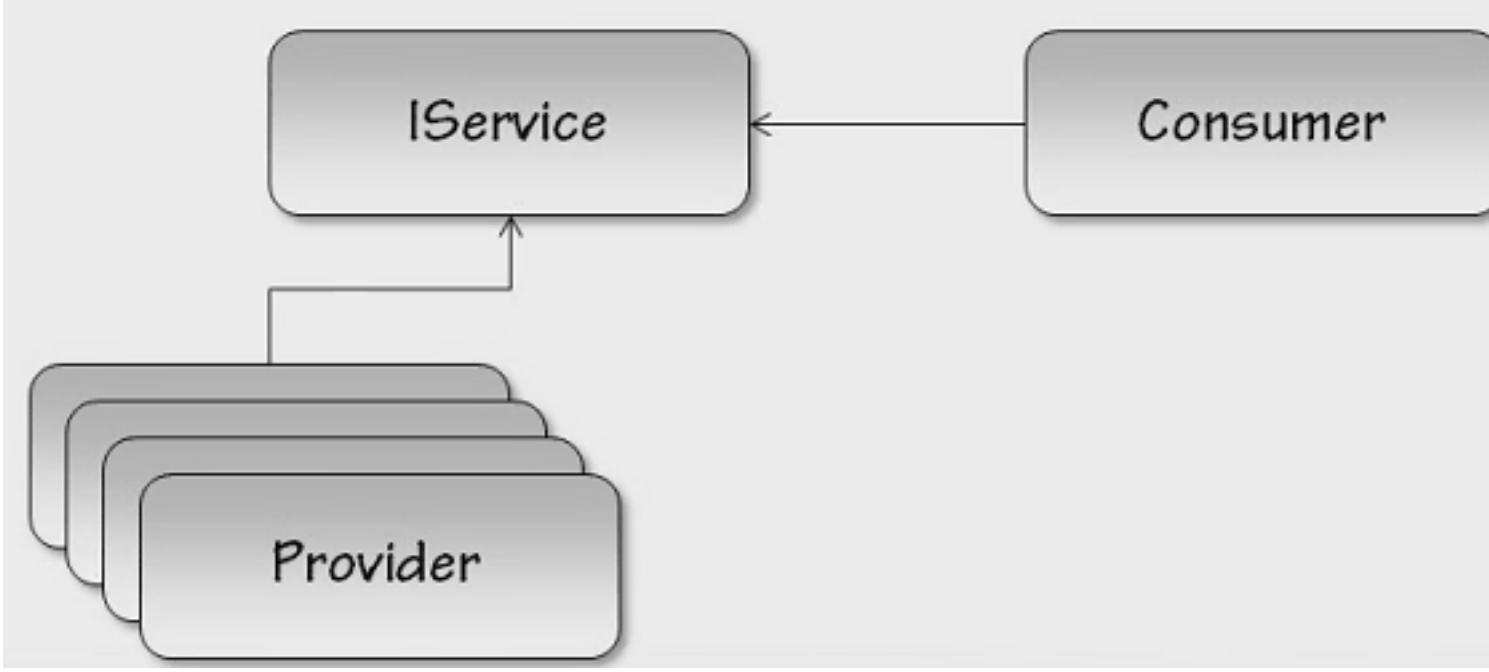
معکوس سازی کنترل (Inversion of Control) الگویی است که نحوه پیاده سازی اصل معکوس سازی وابستگی‌ها (Dependency inversion principle) را بیان می‌کند. با معکوس سازی کنترل، کنترل چیزی را با تغییر کنترل کننده، معکوس می‌کنیم. برای نمونه کلاسی را داریم که ایجاد اشیاء را کنترل می‌کند؛ با معکوس سازی آن به کلاسی یا قسمتی دیگر از سیستم، این مسئولیت را واگذار خواهیم کرد.
IoC یک الگوی سطح بالا است و به روش‌های مختلفی به مسایل متفاوتی جهت معکوس سازی کنترل، قابل اعمال می‌باشد؛ مانند:
- کنترل اینترفیس‌های بین دو سیستم
- کنترل جریان کاری برنامه
- کنترل بر روی ایجاد وابستگی‌ها (جایی که تزریق وابستگی‌ها و DI ظاهر می‌شوند)

سؤال: بین IoC و DIP چه تفاوتی وجود دارد؟

در DIP (قسمت قبل) به این نتیجه رسیدیم که یک ماژول سطح بالاتر نباید به جزئیات پیاده سازی‌های ماژولی سطح پایین‌تر وابسته باشد. هر دوی این‌ها باید بر اساس Abstraction با یکدیگر ارتباط برقرار کنند. IoC روشی است که این Abstraction را فراهم می‌کند. در DIP فقط نگران این هستیم که ماژول‌های موجود در لایه‌های مختلف برنامه به یکدیگر وابسته نباشند اما بیان نکردیم که چگونه.

معکوس سازی اینترفیس‌ها

هدف از معکوس سازی اینترفیس‌ها، استفاده صحیح و معنا دار از اینترفیس‌ها می‌باشد. به این معنا که صرفا تعریف اینترفیس‌ها به این معنا نیست که طراحی صحیحی در برنامه بکار گرفته شده است و در حالت کلی هیچ معنای خاصی ندارد و ارزشی را به برنامه و سیستم شما اضافه نخواهد کرد.
برای مثال یک مسابقه بوکس را درنظر بگیرید. در اینجا Ali یک بوکسور است. مطابق عادت معمول، یک اینترفیس را مخصوص این کلاس ایجاد کرده، به نام IAli و مسابقه بوکس از آن استفاده خواهد کرد. در اینجا تعریف یک اینترفیس برای Ali، هیچ ارزش افزوده‌ای را به همراه ندارد و متاسفانه عادتی است که در بین بسیاری از برنامه نویس‌ها متداول شده است؛ بدون اینکه علت واقعی آن‌را بدانند و تسلطی به الگوهای طراحی برنامه نویسی شیءگرا داشته باشند. صرف اینکه به آن‌ها گفته شده است تعریف اینترفیس خوب است، سعی می‌کنند برای همه چیز اینترفیس تعریف کنند!
تعریف یک اینترفیس تنها زمانی ارزش خواهد داشت که چندین پیاده سازی از آن ارائه شود. در مثال ما پیاده سازی‌های مختلفی از اینترفیس IAli بی‌مفهوم است. همچنین در دنیای واقعی، در یک مسابقه بوکس، چندین و چند شرکت کننده وجود خواهند داشت. آیا باید به ازای هر کدام یک اینترفیس جداگانه تعریف کرد؟ ضمنا ممکن است اینترفیس IAli متدی داشته باشد به نام ضربه، اینترفیس IVahid متد دیگری داشته باشد به نام دفاع.
کاری که در اینجا جهت طراحی صحیح باید صورت گیرد، معکوس سازی اینترفیس‌ها است. به این ترتیب که مسابقه بوکس است که باید اینترفیس مورد نیاز خود را تعریف کند و آن هم تنها یک اینترفیس است به نام IBoxer. اکنون Ali، Vahid و سایرین باید این اینترفیس را جهت شرکت در مسابقه بوکس پیاده سازی کنند. بنابراین دیگر صرف وجود یک کلاس، اینترفیس مجزایی برای آن تعریف نشده و بر اساس معکوس سازی کنترل است که تعریف اینترفیس IBoxer معنا پیدا کرده است. اکنون IBoxer دارای چندین و چند پیاده سازی خواهد بود. به این ترتیب، تعریف اینترفیس، ارزشی را به سیستم افزوده است.
به این نوع معکوس سازی اینترفیس‌ها، الگوی provider model نیز گفته می‌شود. برای مثال کلاسی که از چندین سرویس استفاده می‌کند، بهتر است یک IService را ایجاد کرده و تامین کننده‌هایی، این IService را پیاده سازی کنند. نمونه‌ای از آن در دنیای دات نت، Membership Provider موجود در ASP.NET است که پیاده سازی‌های بسیاری از آن تاکنون تهیه و ارائه شده‌اند.



معکوس سازی جریان کاری برنامه

جریان کاری معمول یک برنامه یا Noraml flow، عموما رویه‌ای یا Procedural است؛ به این معنا که از یک مرحله به مرحله‌ای بعد هدایت خواهد شد. برای مثال یک برنامه خط فرمان را درنظر بگیرید که ابتدا می‌پرسد نام شما چیست؟ در مرحله بعد مثلا رنگ مورد علاقه شما را خواهد پرسید.
برای معکوس سازی این جریان کاری، از یک رابط کاربری گرافیکی یا GUI استفاده می‌شود. مثلا یک فرم را درنظر بگیرید که در آن دو جعبه متنی، کار دریافت نام و رنگ را به عهده دارند؛ به همراه یک دکمه ثبت اطلاعات. به این ترتیب بجای اینکه برنامه، مرحله به مرحله کاربر را جهت ثبت اطلاعات هدایت کند، کنترل به کاربر منتقل و معکوس شده است.

معکوس سازی تولید اشیاء

معکوس سازی تولید اشیاء، اصل بحث دوره و سری جاری را تشکیل می‌دهد و در ادامه مباحث، بیشتر و عمیق‌تر بررسی خواهد گردید.
روش متداول تعریف و استفاده از اشیاء دیگر درون یک کلاس، وهله سازی آن‌ها توسط کلمه کلیدی new است. به این ترتیب از یک وابستگی به صورت مستقیم درون کدهای کلاس استفاده خواهد شد. بنابراین در این حالت کلاس‌های سطح بالاتر به ماژول‌های سطح پایین، به صورت مستقیم وابسته می‌گردند.
برای اینکه این کنترل را معکوس کنیم، نیاز است ایجاد و وهله سازی این اشیاء وابستگی را در خارج از کلاس جاری انجام دهیم.
شاید در اینجا بپرسید که چرا؟
اگر با الگوی طراحی شیءگرای Factory آشنا باشید، همان ایده در اینجا مدنظر است:

```
Button button;
switch (UserSettings.UserSkinType)
{
   case UserSkinTypes.Normal:
      button = new Button();
      break;
   case UserSkinTypes.Fancy:
      button = new FancyButton();
      break;
}
```

در این مثال بر اساس تنظیمات کاربر، قرار است شکل دکمه‌های نمایش داده شده در صفحه تغییر کنند.
حال در این برنامه اگر قرار باشد کار و کنترل محل وهله سازی این دکمه‌ها معکوس نشود، در هر قسمتی از برنامه نیاز است این سوئیچ تکرار گردد (برای مثال در چند ده فرم مختلف برنامه). بنابراین بهتر است محل ایجاد این دکمه‌ها به کلاس دیگری منتقل شود مانند ButtonFactory و سپس از این کلاس در مکان‌های مختلف برنامه استفاده گردد:

```
Button button = ButtonFactory.CreateButton();
```

در این حالت علاوه بر کاهش کدهای تکراری، اگر حالت دکمه جدیدی نیز طراحی گردید، نیاز نخواهد بود تا بسیاری از نقاط برنامه بازنویسی شوند.
بنابراین در مثال فوق، کنترل ایجاد دکمه‌ها به یک کلاس پایه قرار گرفته در خارج از کلاس جاری، معکوس شده است.

انواع معکوس سازی تولید اشیاء

بسیاری شاید تصور کنند که تنها راه معکوس سازی تولید اشیاء، تزریق وابستگی‌ها است؛ اما روش‌های چندی برای انجام اینکار وجود دارد:
الف) استفاده از الگوی طراحی Factory (که نمونه‌ای از آن‌را در قسمت قبل مشاهده کردید)
ب) استفاده از الگوی Service Locator

```
Button button = ServiceLocator.Create(IButton.Class)
```

در این الگو بر اساس یک سری تنظیمات اولیه، نوع خاصی از یک اینترفیس درخواست شده و نهایتا وهله‌ای که آن‌را پیاده سازی می‌کند، به برنامه بازگشت داده می‌شود.
ج) تزریق وابستگی‌ها

```
Button button = GetTheButton();
Form1 frm = new Form1(button);
```

در اینجا نوع وابستگی مورد نیاز کلاس Form1 در سازنده آن تعریف شده و کار تهیه وهله‌ای از آن وابستگی در خارج از کلاس صورت می‌گیرد.

به صورت خلاصه هر زمانیکه تولید و وهله سازی وابستگی‌های یک کلاس را به خارج از آن منتقل کردید، کار معکوس سازی تولید وابستگی‌ها انجام شده است.

نظرات خوانندگان

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۰۵/۲۱ ۱۲:۱۱

ممکن است مثالهای دیگری(غیر از Command line, GUI) در مورد معکوس سازی جریان کاری برنامه بیان نمایید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۱ ۱۳:۱۵

مثلا می‌توانید دسترسی به داده‌ها رو مستقیما در کدهای یک کنترلر یا فرم قرار دهید و یا با استفاده از معکوس سازی وابستگی‌ها، این دسترسی را به یک لایه سرویس منتقل و معکوس کرده و استفاده کنید (در کنترلر یا فرم برنامه). استفاده نهایی از امکانات اینترفیس‌های لایه سرویس خواهد بود، بدون اطلاع از نحوه پیاده سازی خاص لایه سرویس برنامه. به این ترتیب علاوه بر جداسازی لایه‌های مختلف برنامه، امکان ارائه نگارش‌های مختلفی از پیاده سازی‌های اینترفیس‌ها نیز وجود خواهد داشت. در برنامه واقعی، پیاده سازی کننده‌ها، از دیتابیس واقعی استفاده می‌کنند و در مثلا در حین آزمایش واحد، از پیاده سازی خاص استفاده کننده از یک بانک اطلاعاتی ساده تشکیل شده در حافظه برای بالا رفتن سرعت آزمون‌ها.

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۰۵/۲۱ ۱۳:۲۸

ممکنه بیشتر توضیح بدین؟ متوجه نشدم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۱ ۱۳:۳۳

در قسمت‌های بعدی با مثال انجام شده. مثلا در قسمت معرفی IoC Container ، در مورد نحوه معکوس سازی ارسال ایمیل به لایه سرویس با مثال بحث شده. یا در قسمت نحوه یافتن وابستگی‌ها در یک پروژه موجود ، نحوه معکوس سازی دریافت اطلاعات از وب به لایه‌ای دیگر بحث شده.

نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۲/۰۶/۰۴ ۱۸:۴۶

تعریف ساده دیگری از معکوس سازی کنترل:

فرض کنید دو کلاس **مشتری** و **آدرس** دارید، در حالت عادی کلاس **مشتری** ایجاد و استفاده از کلاس **آدرس** را کنترل می‌کند ولی با استفاده از اصل معکوس سازی کنترل، کلاس **آدرس** به کلاس **مشتری** می‌گوید که تو من را ایجاد نکن و من خودم را در جایی دیگر ایجاد می‌کنم.