

عنوان: اصل معکوس سازی وابستگی‌ها

نویسنده: وحید نصیری

تاریخ: ۹:۵۱۳۹۲/۰۱/۲۵

آدرس: www.dotnettips.info

برچسب‌ها: Design patterns, Dependency Injection, IoC

پیش از شروع این سری نیاز است با تعدادی از واژه‌های بکار رفته در آن به اختصار آشنا شویم؛ از این واژه‌ها به کرات استفاده شده و در طول دوره به بررسی جزئیات آن‌ها خواهیم پرداخت:

1) Dependency inversion principle یا DIP (اصل معکوس سازی وابستگی‌ها)
DIP یکی از اصول طراحی نرم افزار است و D آن همان D معروف **SOLID** است (اصول پذیرفته شده شیء‌گرایی).

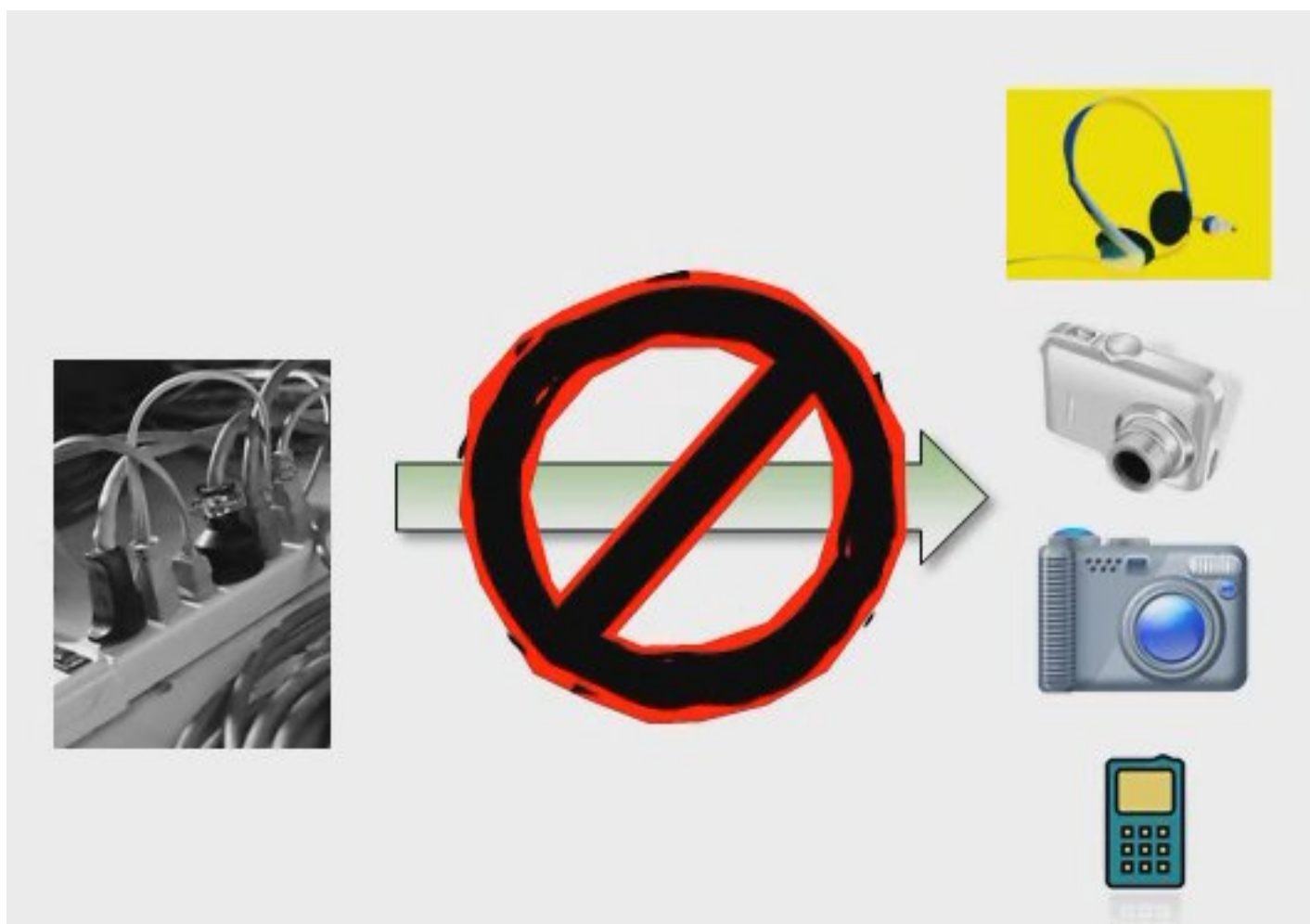
2) Inversion of Control یا IOC (معکوس سازی کنترل)
الگویی است که نحوه پیاده سازی DIP را بیان می‌کند.

3) Dependency injection یا DI (تزریق وابستگی‌ها)
یکی از روش‌های پیاده سازی IOC است.

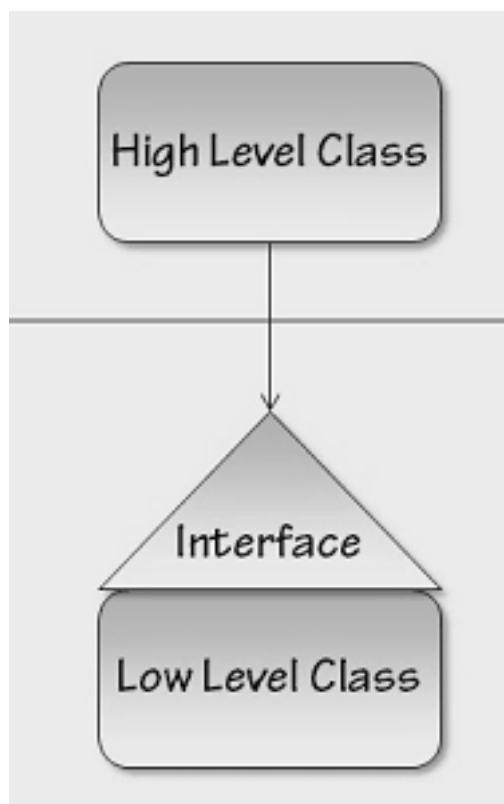
4) IOC container
به فریم ورک‌هایی که کار DI را انجام می‌دهند گفته می‌شود.

چيست Dependency inversion principle؟

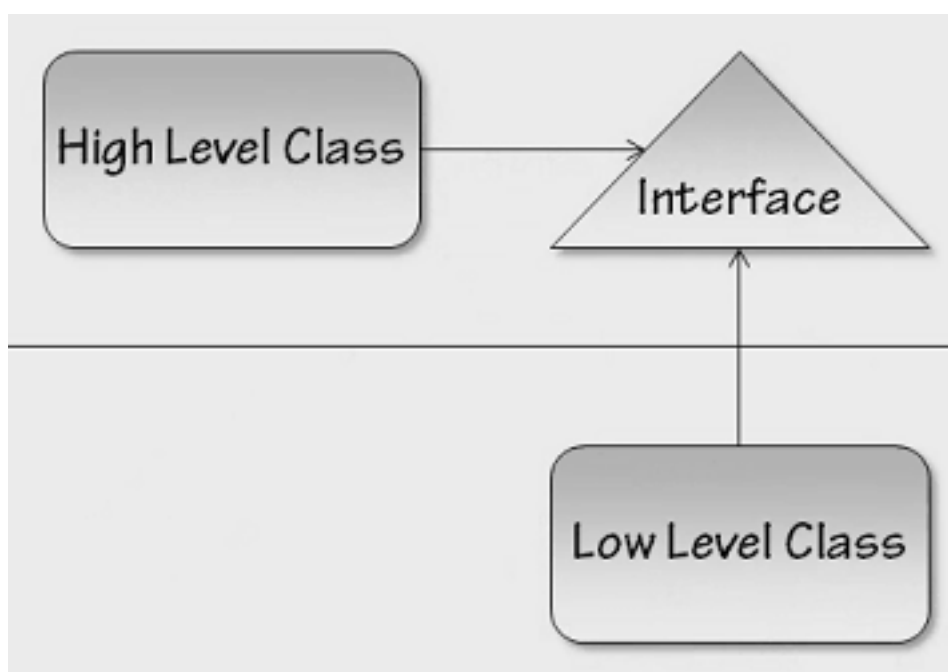
اصل معکوس سازی وابستگی‌ها به این معنا است که بجای اینکه ماژول‌های سطح پایین سیستم، رابط‌های قابل استفاده‌ای از خود را در اختیار سطوح بالاتر سیستم قرار دهند، ماژول‌های قرار گرفته در سطوح بالاتر، اینترفیس‌هایی را تعریف می‌کنند که توسط ماژول‌های سطح پایین پیاده سازی خواهند شد.
همانطور که ملاحظه می‌کنید به این ترتیب وابستگی‌های سیستم معکوس خواهند شد. نمونه‌ای از عدم استفاده از این طراحی را در دنیای واقعی به صورت رومزه با آن‌ها سر و کار داریم؛ مانند وسایل الکترونیکی قابل حملی که نیاز به شارژ مجدد دارند. برای مثال تلفن‌های همراه، دوربین‌های عکاسی دیجیتال و امثال آن.



هر کدام از این‌ها، رابط‌های اتصال متفاوتی دارند. یکی USB2، یکی USB3 دیگری Mini USB و بعضی‌ها هم از پورت‌های دیگری استفاده می‌کنند. چون هر کدام از لایه‌های زیرین سیستم (در اینجا وسایل قابل شارژ) رابط‌های اتصال مختلفی را ارائه داده‌اند، برای اتصال آن‌ها به منبع قدرت که در سطحی بالاتر قرار دارد، نیاز به تبدیلگرها و درگاه‌های مختلفی خواهد بود. اگر در این نوع طراحی‌ها، اصل معکوس سازی وابستگی‌ها رعایت می‌شد، درگاه و رابط اتصال به منبع قدرت باید تعیین کننده نحوه طراحی اینترفیس‌های لایه‌های زیرین می‌بود تا با این آشفتگی نیاز به انواع و اقسام تبدیلگرها، روبرو نمی‌شدیم.

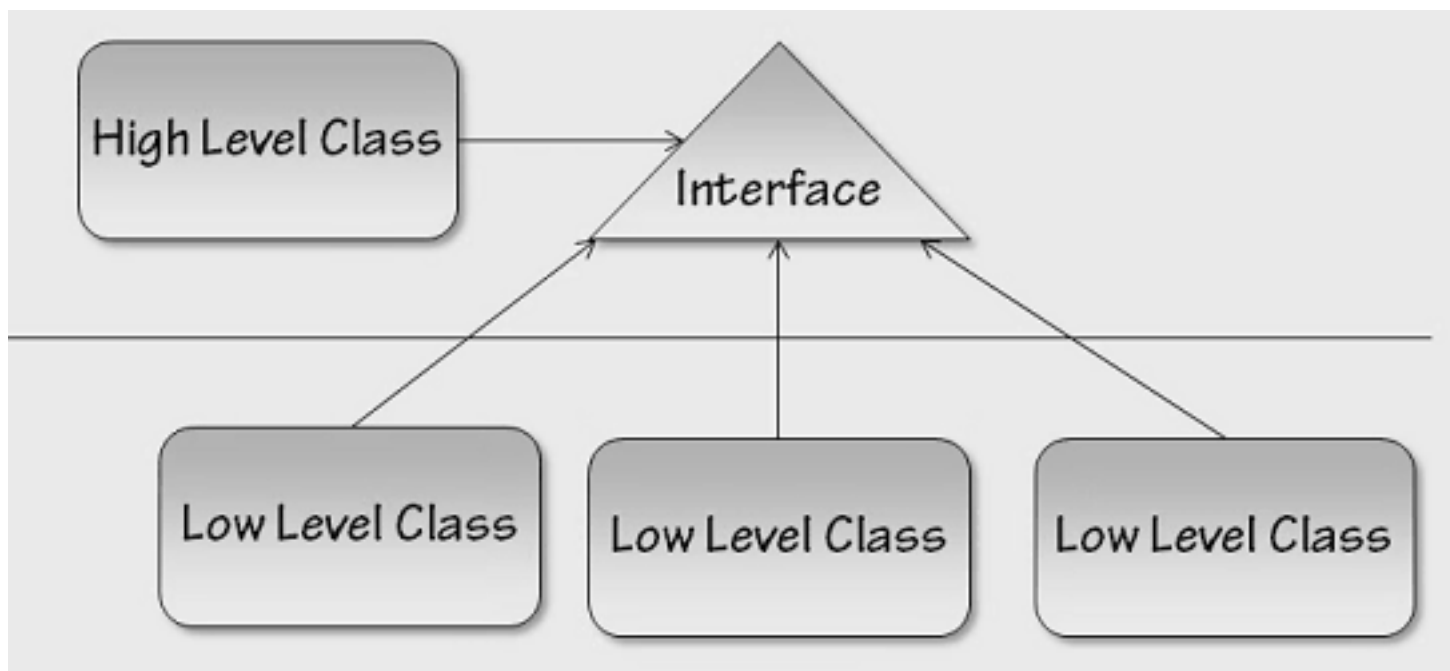


اگر وابستگی‌ها معکوس نشوند مطابق تصویر فوق، کلاس سطح بالایی را خواهیم داشت که به اینترفیس کلاس‌های سطح پایین وابسته است. البته در اینجا اینترفیس یک کلمه عمومی است و بیشتر نحوه در معرض دید و استفاده قرار دادن اعضای یک کلاس مد نظر بوده است تا اینکه مثلاً الزاماً اینترفیس‌های زبان خاصی مدنظر باشند. مشکلی که در این حالت به زودی بروز خواهد کرد، افزایش کلاس‌های سطح پایین و بیشتر شدن وابستگی کلاس‌های سطح بالا به آنها است. به این ترتیب قابلیت استفاده مجدد خود را از دست خواهند داد.

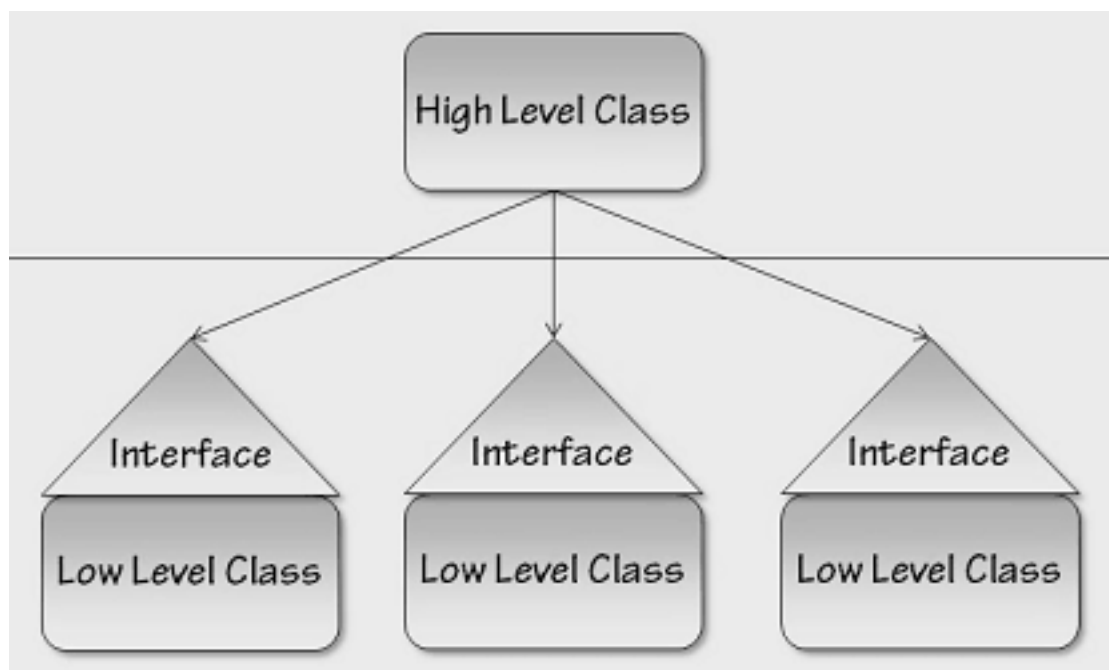


در تصویر فوق حالتی را مشاهده می‌کنید که وابستگی‌ها معکوس شده‌اند. تغییر مهمی که در اینجا نسبت به حالت قبل رخ داده است، بالا بردن اینترفیس، به بالای خط میانی است که در تصویر مشخص گردیده است. این خط، معرف تعریف لایه‌های مختلف سیستم است. به عبارتی کلاس‌های سطح بالا در لایه دیگری نسبت به کلاس‌های سطح پایین قرار دارند. در اینجا اجازه داده‌ایم تا کلاس لایه بالایی اینترفیس مورد نیاز خود را تعریف کند. این نوع اینترفیس‌ها در زبان سی شارپ می‌توانند یک کلاس Abstract و یا حتی یک Interface متداول باشند.

با معکوس شدن وابستگی‌ها، لایه سطح بالا است که به لایه زیرین عنوان می‌کند: تو باید این امکانات را در اختیار من قرار دهی تا بتوانم کارم را انجام دهم.



اکنون اگر در یک سیستم واقعی تعداد کلاس‌های سطح پایین افزایش پیدا کنند، نیازی نیست تا کلاس سطح بالا تغییری کند. کلاس‌های سطح پایین تنها باید عملکردهای تعیین شده در اینترفیس را پیاده سازی کنند. و این برخلاف حالتی است که وابستگی‌ها معکوس نشده‌اند:



تاریخچه اصل معکوس سازی وابستگی‌ها

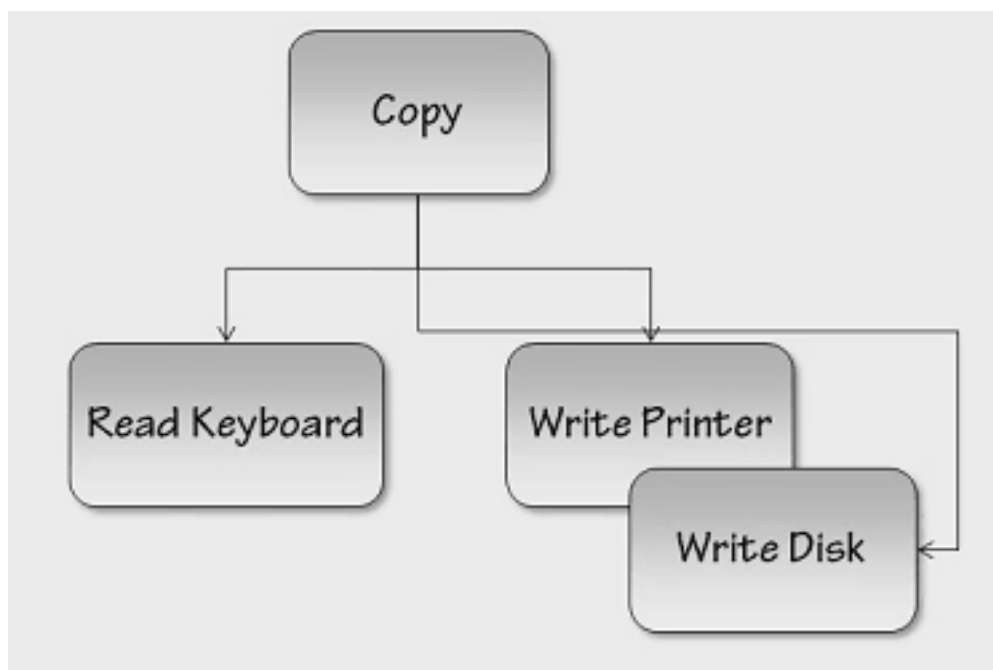
اصل معکوس سازی وابستگی‌ها در نشریه C++ Report سال 1996 توسط شخصی به نام [Bob Martin](#) (معروف به Uncle Bob!) برای اولین بار مطرح گردید. ایشان همچنین یکی از آغاز کنندگان گروهی بود که مباحث Agile را ارائه کردند. به علاوه ایشان برای اولین بار مباحث SOLID را در دنیای شیءگرایی معرفی کردند (همان مباحث معروف هر کلاس باید تک مسئولیتی باشد، باز باشد برای توسعه، بسته برای تغییر و امثال آن که ما در این سری مباحث قسمت D آنرا در حالت بررسی هستیم).

مطابق تعاریف Uncle Bob:

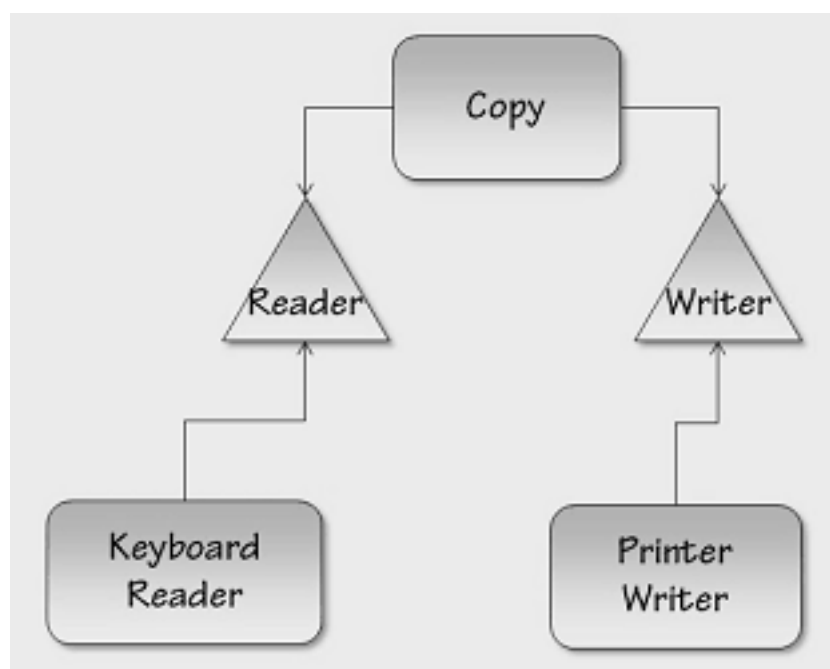
الف) ماژول‌های سطح بالا نباید به ماژول‌های سطح پایین وابسته باشند. هر دوی این‌ها باید به Abstraction وابسته باشند.
ب) Abstraction نباید وابسته به جزئیات باشد. جزئیات (پایه سازی‌ها) باید وابسته به Abstraction باشند.

مثال برنامه کپی

اگر به مقاله Uncle Bob مراجعه کنید، یکی از مواردی را که عنوان کرده‌اند، یک برنامه کپی است که می‌تواند اطلاعات را از صفحه کلید دریافت و در یک چاپگر، چاپ کند.



حال اگر به این مجموعه، ذخیره سازی اطلاعات بر روی دیسک سخت را اضافه کنیم چطور؟ به این ترتیب سیستم با افزایش وابستگی‌ها، پیچیدگی و if و else‌های بیشتری را خواهد یافت؛ از این جهت که سطح بالایی سیستم به صورت مستقیم وابسته خواهد بود به ماژول‌های سطح پایین آن. روشی را که ایشان برای حل این مشکل ارائه داده‌اند، معکوس کردن وابستگی‌ها است:

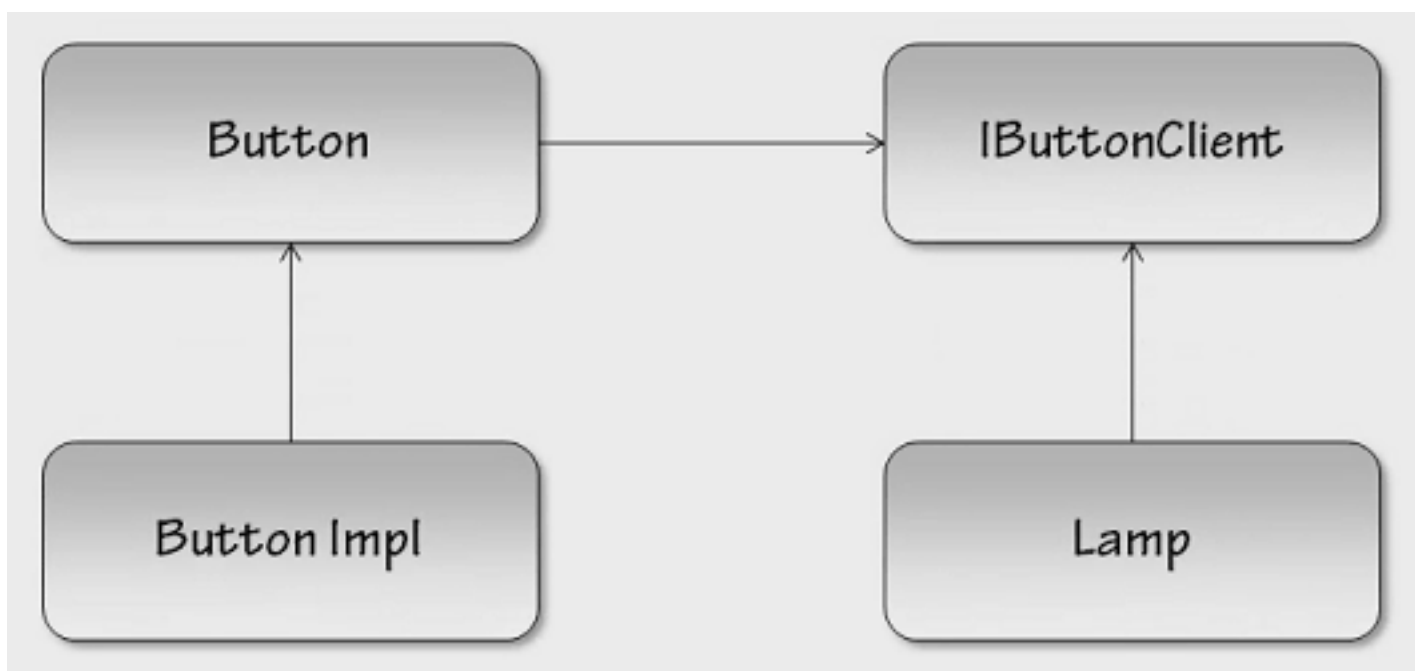


در اینجا سطح بالایی سیستم وابسته است به یک سری تعاریف Abstract خواندن و یا نوشتن؛ بجای وابستگی مستقیم به پیاده سازی‌های سطح پایین آن‌ها. در این حالت اگر تعداد Readers و یا Writers افزایش یابند، باز هم سطح بالایی سیستم نیازی نیست تغییر کند زیرا وابسته است

به یک اینترفیس و نه پیاده سازی آن که محول شده است به لایه‌های زیرین سیستم. این مساله بر روی لایه بندی سیستم نیز تاثیرگذار است. در روش متداول برنامه نویسی، لایه بالایی به صورت مستقیم متدهای لایه‌های زیرین را صدا زده و مورد استفاده قرار می‌دهد. به این ترتیب هر تغییری در لایه‌های مختلف، بر روی سایر لایه‌ها به شدت تاثیرگذار خواهد بود. اما در حالت معکوس سازی وابستگی‌ها، هر کدام از لایه‌های بالاتر، از طریق اینترفیس از لایه زیرین خود استفاده خواهد کرد. در این حالت هرگونه تغییری در لایه‌های زیرین برنامه تا زمانیکه اینترفیس تعریف شده را پیاده سازی کنند، اهمیتی نخواهد داشت.

مثال برنامه دکمه و لامپ

مثال دیگری که در مقاله Uncle Bob ارائه شده، مثال برنامه دکمه و لامپ است. در حالت متداول، یک دکمه داریم که وابسته است به لامپ. برای مثال وهله‌ای از لامپ به دکمه ارسال شده و سپس دکمه آنرا کنترل خواهد کرد (خاموش یا روشن). مشکلی که در اینجا وجود دارد وابستگی دکمه به نوعی خاص از لامپ است و تعویض یا استفاده مجدد از آن به سادگی میسر نیست. راه حلی که برای این مساله ارائه شده، ارائه یک اینترفیس بین دکمه و لامپ است که خاموش و روشن کردن در آن تعریف شده‌اند. اکنون هر لامپی (یا هر وسیله الکتریکی دیگری) که بتواند این متدها را ارائه دهد، در سیستم قابل استفاده خواهد بود.



نظرات خوانندگان

نویسنده: سیروس
تاریخ: ۱۳۹۲/۰۱/۲۵ ۹:۴۹

مثل همیشه عالی بود، واقعا جامعه برنامه نویسان به این مطالب زیاد نیاز دارند. خیلی اوقات فکر می‌کنیم همین که از اینترفیس استفاده کنیم کار تمومه؛ غافل از اینکه این اینترفیس‌ها پایین‌تر از اون خطه!

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۱/۲۵ ۱۳:۱۵

این بحث با کیفیت مطلوب و دقت نظر خاصی مطرح شده است که البته از آقای نصیری هم جز این انتظار نمی‌ره. ابتدا شرایط موجود بیان و بررسی می‌شود و برای بهبود آن راهکاری را مفید می‌بینیم و می‌پذیریم به نام اصل وارونگی وابستگی یا همان معکوس‌سازی وابستگی که در این بخش به آن پرداخته شده است. موارد ۲ و ۳ و ۴ که به آن اشاره شده است در حقیقت روند طبیعی محقق شدن این اصل است که در بخش‌های بعدی به آن‌ها پرداخته خواهد شد. توجه به این روند سبب می‌شود این مفاهیم به جای هم به کار برده نشوند. پس از قبول اصل یاد شده باید آن‌را پیاده سازی نمود. الگوی وارونگی کنترل برای پیاده سازی آن تدوین می‌گردد. برای اجرای این الگو نیاز به روشی پیدا می‌کنیم که وابستگی را به طور کامل بیرون شیء وابسته نگه داریم. پس چاره ای جز تزریق آن در زمانی که لازم است نداریم. بسیار خوب نام آن‌را روش تزریق وابستگی می‌گذاریم. در نهایت تزریق‌هایی که ممکن است پی در پی لازم شود را گردن کس دیگری می‌اندازیم که همان برنامه‌جانبی یا کتابخانه یا فریم‌ورک‌هایی هستند که به آن‌ها می‌گوییم چه وقت چی تزریق کنند. (آن‌ها در بردارنده اطلاعاتی هستند که مثلاً شیء ۱ برای انجام کار شیء ۲ باید به آن داده شود) حال اگر تازه با این مفاهیم آشنا شده اید توصیه می‌کنم یک بار دیگر این بخش را مطالعه کنید و منتظر روند پیاده سازی این اصل در بخش‌های بعد باشید.

تشکر از مهندس نصیری برای اجرای این دوره آموزشی.

نویسنده: مسعود 2
تاریخ: ۱۳۹۲/۰۵/۲۰ ۱۲:۲۲

تعریف دقیق ماژولهای سطح بالا و سطح پایین چیست و چطوری میشه اونها رو در نرم افزار پیدا کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۰ ۱۲:۳۰

مراجعه کنید به مطلب «[مراحل Refactoring یک قطعه کد برای اعمال تزریق وابستگی‌ها](#)» جهت نحوه یافتن وابستگی‌ها و معکوس کردن آن‌ها.

نویسنده: مسعود 2
تاریخ: ۱۳۹۲/۰۵/۲۱ ۱۴:۳۶

فرمودید: "پس چاره ای جز تزریق آن در زمانی که لازم است نداریم". آیا نمیتوان به جای تزریق وابستگی از الگوهای مثل Service Locator و Factory استفاده نمود؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۱ ۱۴:۵۵

اول یکبار دوره را کامل مطالعه کنید. در طی این سری مباحث به «[بایدها و نبایدهای تزریق وابستگی‌ها](#)» مفصل پرداخته شده.

نویسنده: Programmer

تاریخ: ۱۴:۴۷ ۱۳۹۲/۰۶/۰۵

با سلام
اینکه با مثال مفهوم رو توضیح دادید خیلی خوبه!
با توجه به انتزاعی بودن برنامه نویسی، ارائه یک مثال عینی کار رو خیلی راحت‌تر می‌کنه و خواننده راحت‌تر تصویر سازی می‌کنه و متوجه امر میشه.
بازم ممنون
راستی این وهله سازی یعنی چی؟

نویسنده: وحید نصیری
تاریخ: ۱۴:۵۵ ۱۳۹۲/۰۶/۰۵

وهله سازی معادل instantiation است. یک instance یا یک وهله.

نویسنده: وحید م
تاریخ: ۱۳:۵۷ ۱۳۹۲/۰۷/۱۳

با تشکر از مطالب مفیدتون .
ببخشید منظور از لایه بالاتر همان ui و پایین همان service است . ممنون از راهنمایی شما

نویسنده: وحید نصیری
تاریخ: ۱۴:۳۰ ۱۳۹۲/۰۷/۱۳

[یک مثالش](#) می‌تونه لایه UI و لایه سرویس باشه.

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

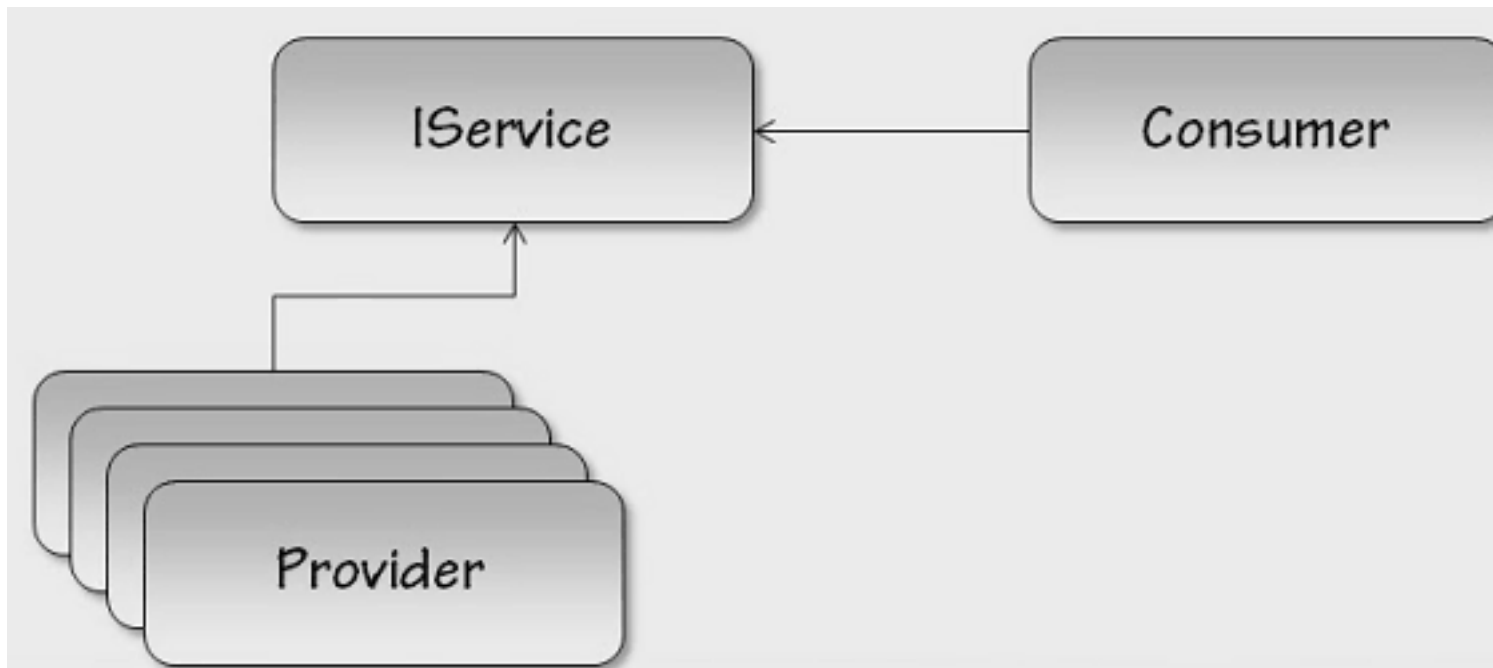
هدف از معکوس سازی اینترفیس‌ها، استفاده صحیح و معنا دار از اینترفیس‌ها می‌باشد. به این معنا که صرفاً تعریف اینترفیس‌ها به این معنا نیست که طراحی صحیحی در برنامه بکار گرفته شده است و در حالت کلی هیچ معنای خاصی ندارد و ارزشی را به برنامه و سیستم شما اضافه نخواهد کرد.

برای مثال یک مسابقه بوکس را در نظر 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 متأسفانه عادت است که در بین بسیاری از برنامه نویسی‌ها 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؟ ضمناً ممکن است اینترفیس IAli متدی داشته باشد به نام ضربه، اینترفیس IVahid متدی 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 Normal flow، عموماً رویه‌ای یا Procedural است؛ به این معنا که از یک مرحله به مرحله‌ای بعد هدایت خواهد شد. برای مثال یک برنامه خط فرمان را در نظر بگیرید که ابتدا می‌پرسد نام شما چیست؟ در مرحله بعد مثلاً رنگ مورد علاقه شما را خواهد پرسید. برای معکوس سازی این جریان کاری، از یک رابط کاربری گرافیکی یا GUI استفاده می‌شود. مثلاً یک فرم را در نظر 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 و هله 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 نهایتاً هله‌ای 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 هله‌ای از آن وابستگی در خارج از کلاس صورت می‌گیرد.

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

نظرات خوانندگان

نویسنده: مسعود 2
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: مسعود 2
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:

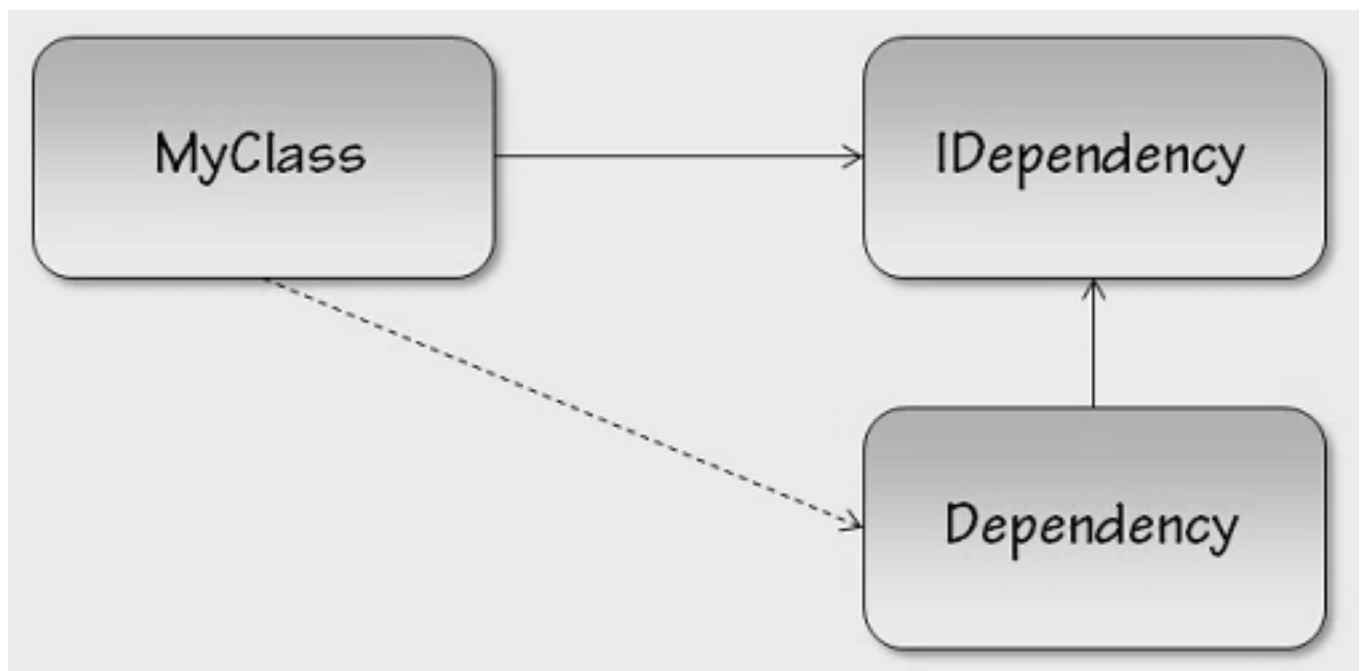
فرض کنید دو کلاس **مشتری** و **آدرس** دارید، در حالت عادی کلاس **مشتری** ایجاد و استفاده از کلاس **آدرس** را کنترل می‌کند ولی با استفاده از اصل معکوس سازی کنترل، کلاس **آدرس** به کلاس **مشتری** می‌گوید که تو من را ایجاد نکن و من خودم را در جایی دیگر ایجاد می‌کنم.

خوب! بالاخره بعد از دو قسمت قبل، به بحث اصلی تزریق وابستگی‌ها رسیدیم. بحثی که بسیاری از برنامه نویسی‌های تصور می‌کنند همان IoC است که پیشتر در مورد آن بحث شد.

تزریق وابستگی‌ها یا DI چیست؟

تزریق وابستگی‌ها یکی از انواع IoC بوده که در آن ایجاد و انقیاد (binding) یک وابستگی، در خارج از کلاسی که به آن نیاز دارد صورت می‌گیرد. روش‌های متفاوتی برای ارائه این وابستگی و هله سازی شده در خارج از کلاس به آن وجود دارد که در ادامه مورد بررسی قرار خواهند گرفت.

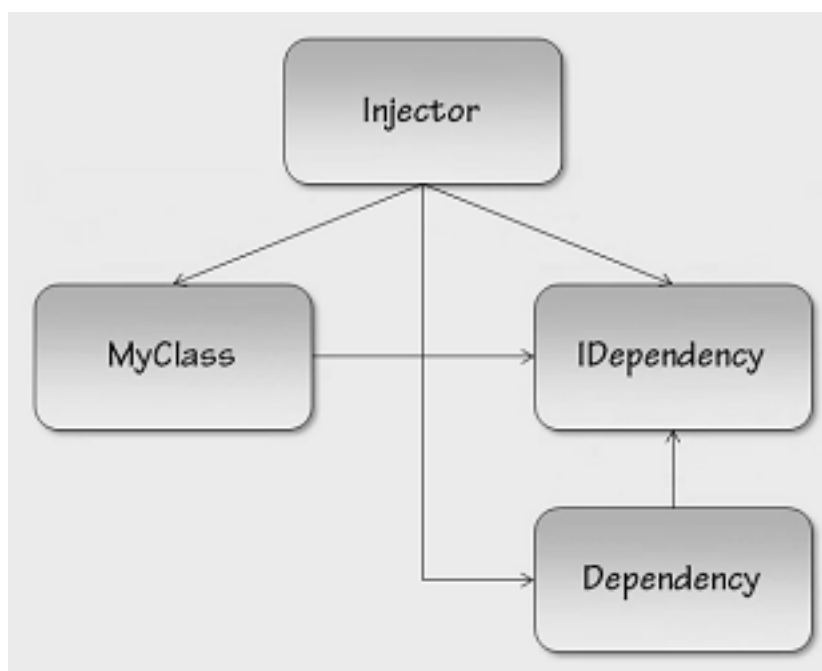
یک مثال: بسته غذایی را که با خود به سر کار می‌برید در نظر بگیرید. تمام اجزای مورد نیاز تشکیل دهنده یک بسته غذا عموماً داخل آن قرار گرفته و حمل می‌شوند. حالت عکس آن زمانی است که در محل کار به شما غذا می‌دهند. این دقیقاً همان حالتی است که تزریق وابستگی‌ها کار می‌کند؛ یک سری سرویس‌های خارجی، نیازهای کلاس جاری را برآورده می‌سازند.



در تصویر فوق یک طراحی مبتنی بر معکوس سازی کنترل‌ها را مشاهده می‌کنید. وابستگی‌های یک کلاس توسط اینترفیسی مشخص شده و سپس کلاسی وجود دارد که این وابستگی را پیاده سازی کرده است. همچنین در این تصویر یک خط منقطع از کلاس MyClass به کلاس Dependency رسم شده است. این خط، مربوط به حالتی است که خود کلاس، مستقیماً کار و هله سازی وابستگی مورد نیاز خود را انجام دهد؛ هر چند اینترفیسی نیز در این بین تعریف شده باشد. بنابراین اگر در بین کدهای این کلاس، چنین کدی مشاهده شد:

```
IDependency dependency= new Dependency();
```

تعریف dependency از نوع IDependency به معنای معکوس سازی کنترل نبوده و عملاً همان معماری سابق و متداول بکارگرفته شده است. برای بهبود این وضعیت، از تزریق وابستگی‌ها کمک خواهیم گرفت:



در اینجا یک کلاس دیگر به نام Injector اضافه شده است که قابلیت تزریق وابستگی‌ها را به کلاس نیازمند آن به روش‌های مختلفی دارا است. کار کلاس Injector، وهله سازی MyClass و همچنین وابستگی‌های آن می‌باشد؛ سپس وابستگی را در اختیار MyClass قرار می‌دهد.

تزریق وابستگی‌ها در سازنده کلاس

یکی از انواع روش‌های تزریق وابستگی‌ها، Constructor Injection و یا تزریق وابستگی‌ها در سازنده کلاس می‌باشد که متداول‌ترین نوع آن‌ها نیز به‌شمار می‌رود. در این حالت، وابستگی پس از وهله سازی، از طریق پارامترهای سازنده یک کلاس، در اختیار آن قرار می‌گیرند.
یک مثال:

```

public class Shopper
{
    private readonly ICreditCard _creditCard;
    public Shopper(ICreditCard creditCard)
    {
        _creditCard = creditCard;
    }
}

```

در اینجا شما یک کلاس خریدار را مشاهده می‌کنید که وابستگی مورد نیاز خود را به شکل یک اینترفیس از طریق سازنده کلاس دریافت می‌کند.

```

ICreditCard creditCard = new MasterCard();
Shopper shopper = new Shopper(creditCard);

```

برای نمونه، وهله‌ای از مستر کارتی که ICreditCard را پیاده سازی کرده باشد، می‌توان به سازنده این کلاس ارسال کرد. کار وهله سازی وابستگی و واژه کلیدی new به خارج از کلاس استفاده کننده از وابستگی منتقل شده است. بنابراین همانطور که ملاحظه می‌کنید، این مفاهیم آنچنان پیچیده نبوده و به همین سادگی قابل تعریف و اعمال هستند.

تزریق در خواص عمومی کلاس

Setter Injection و یا تزریق در خواص عمومی یک کلاس، یکی دیگر از روش‌های تزریق وابستگی‌ها است (setter در اینجا منظور همان get و set یک خاصیت می‌باشد). در حالت تزریق وابستگی‌ها در سازنده کلاس، امکان وهله سازی آن کلاس بدون ارسال وابستگی‌ها به سازنده آن ممکن نیست؛ اما در اینجا خیر و امکان وهله سازی و استفاده از یک کلاس، پیش از اعمال وابستگی‌ها نیز وجود دارد. بنابراین امکان تغییر و تعویض وابستگی‌ها پس از وهله سازی کلاس نیز میسر است.

```
public class Shopper
{
    public ICreditCard CreditCard { get; set; }
}

ICreditCard creditCard = new MasterCard();
Shopper shopper = new Shopper();
shopper.CreditCard = creditCard;
```

نمونه‌ای از این روش را در مثال فوق مشاهده می‌کنید. در این کلاس، وابستگی مورد نیاز از طریق یک خاصیت عمومی دریافت شده است.

تزریق اینترفیس‌ها

تزریق اینترفیس‌ها نیز یکی دیگر از روش‌های تزریق وابستگی‌ها است؛ اما آنچنان استفاده گسترده‌ای برخلاف دو روش قبل نیافته است. در این روش نه از سازنده کلاس استفاده می‌شود و نه از یک خاصیت عمومی. ابتدا یک اینترفیس که بیانگر ساختار کلاسی که قرار است تزریق شود ایجاد می‌گردد. سپس این اینترفیس را در کلاس وابستگی مورد نظر پیاده سازی خواهیم کرد. در این اینترفیس نیاز است متد خاصی تعریف شود تا کار تزریق وابستگی را انجام دهد. یک مثال:

```
public interface IDependOnCreditCard
{
    void Inject(ICreditCard creditCard);
}

public class Shopper : IDependOnCreditCard
{
    private ICreditCard creditCard;
    public void Inject(ICreditCard creditCard)
    {
        this.creditCard = creditCard;
    }
}

ICreditCard creditCard = new MasterCard();
Shopper shopper = new Shopper();
((IDependOnCreditCard)shopper).Inject(creditCard);
```

در اینجا یک اینترفیس به نام IDependOnCreditCard تعریف گردیده و متد خاصی را به نام Inject تدارک دیده است که نوعی از کردیت کارد را دریافت می‌کند. در ادامه کلاس خریدار، اینترفیس IDependOnCreditCard را پیاده سازی کرده است. به این ترتیب کلاس Injector تنها کافی است بداند تا این متد خاص را باید جهت تنظیم و تزریق وابستگی‌ها فراخوانی نماید. این روش نسبتاً شبیه به روش Setter injection است. این روش بیشتر می‌تواند جهت فریم ورک‌هایی که قابلیت یافتن کلیه کلاس‌های مشتق شده از IDependOnCreditCard را داشته و سپس می‌دانند که باید متد Inject آن‌ها را فراخوانی کنند، مناسب است.

نکاتی که باید حین کار با تزریق وابستگی‌ها در نظر داشت

الف) حین استفاده از تزریق وابستگی‌ها، وهله سازی به تاخیر افتاده وابستگی‌ها میسر نیست. برای مثال زمانی که یک وابستگی قرار است در سازنده کلاسی تزریق شود، وهله سازی آن باید پیش از نیاز واقعی به آن صورت گیرد. البته امکان استفاده از اشیاء Lazy دات نت 4 برای مدیریت این مساله وجود دارد اما در حالت کلی، دیگر همانند قبل و روش‌های متداول، وهله سازی تنها زمانی که نیاز به آن وابستگی خاص باشد، میسر نیست. به همین جهت باید به تعداد وابستگی‌هایی که قرار است در یک کلاس استفاده شوند نیز جهت کاهش مصرف حافظه دقت داشت.

ب) یکی از مزایای دیگر تزریق وابستگی‌ها، ساده‌تر شدن نوشتن آزمون‌های واحد است. زیرا تهیه Mocks در این حالت کار با اینترفیس‌ها بسیار ساده‌تر است. اما باید دقت داشت، کلاسی که در سازنده آن حداقل 10 اینترفیس را به عنوان وابستگی دریافت می‌کند، احتمالاً دچار مشکلاتی در طراحی و همچنین مصرف حافظه می‌باشد.

نظرات خوانندگان

نویسنده: فرشید علی اکبری
تاریخ: ۱۱:۵۶ ۱۳۹۲/۰۱/۲۶

سلام و درود بر شما آقای نصیری

من چون توی دات نت تازه کار هستم دوتا سؤال خدمتون دارم:

من توی برنامه ای که دارم می‌نویسم از الگوی UnitOfWork مطابق با آموزش‌های شما استفاده کردم درضمن برای تزریق وابستگی هم از StructerMap استفاده می‌کنم، برنامه Win Form هستش و توی Main یک کلاس Configuration رو که کارش Register کردن کلیه Interface و کلاس هاست رو فراخونی کردم.

اول اینکه : برای آزاد سازی منابع و استفاده بهینه از حافظه در حال استفاده از StructerMap شما چه پیشنهاد یا روشی رو معرفی می‌کنین؟

دوم اینکه : با ازدیاد و تعدد کلاسها و اینترفیس‌ها در حالیکه در ابتدای برنامه کلیه اونها رو با StructerMap رجیستر می‌کنیم و در هر جا که لازم باشه فقط از اونها یک نمونه میسازیم، اشکالی در روند عملیاتی و کاربری با اون نرم افزار پیش مشتری پیش نمی‌یاد؟ (مخصوصا در سیستم‌های یکپارچه و بزرگ از نظر حافظه).

سوم اینکه: آیا باید‌ها و نبایدهایی هم در استفاده از StructerMap وجود داره ؟

سپاسگزار شما هستم.

نویسنده: وحید نصیری
تاریخ: ۱۲:۲۴ ۱۳۹۲/۰۱/۲۶

- کار آزاد سازی منابع را به صورت خودکار GC انجام خواهد داد. فقط وهله سازی در اینجا توسط IoC Container انجام می‌شود. مابقی مسایل مانند قبل است.
- خیر. برای نمونه همین سایت جاری از StructerMap استفاده می‌کند و مشکلی هم از لحاظ میزان مصرف حافظه وجود ندارد. اساسا ایجاد چند شیء در سازنده یک کلاس آنچنان حافظه‌ای را مصرف نمی‌کنند مگر اینکه سازنده‌های این کلاس‌ها دارای تخصیص منابع قابل توجهی باشند. بنابراین سازنده‌های تعریف شده را سبک در نظر بگیرید و طراحی کنید.
- یک قسمت را به StructerMap اختصاص خواهیم داد؛ به صورت جداگانه.

نویسنده: فرشید علی اکبری
تاریخ: ۱۲:۲۷ ۱۳۹۲/۰۱/۲۶

سپاسگزار سرعت جوابگوئی شما هستم

امیدوارم هرچه زودتر شاهد مقاله مربوط به StructerMap از شما باشم.

نویسنده: محمد آزاد
تاریخ: ۰:۲۸ ۱۳۹۲/۰۴/۲۱

تزریق وابستگی رو تا چه سطحی باید انجام داد؟ یعنی رعایت کردن اون تو تمام سطوح نرم افزار باید انجام بشه؟ برای مثال کلاس زیر رو در نظر بگیرید که در لایه Entity وجود داره

```
class Parent
{
```

```
public IChild child {get;set;}
public Parent(IChild child)
{
    this.child =child;
}
}
```

آیا با اینکه کلاس پدر و فرزند در یک لایه مشترک هستند ، در اینجا ارزش داره که تزریق وابستگی رو انجام بدیم ؟حجم کد و کار بالا بالا نمیره؟ یا اینکه پیچیدگی زیاد نمیشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۲۱ ۰:۳۴

در دو مطلب زیر به سؤال شما پاسخ داده شده:
- [الگوی معکوس سازی کنترل چیست؟](#) (آیا هرجایی باید اینترفیس تعریف کرد؟ و کجا؟ اصلا این معکوس سازی چی هست و چه هدفی رو دنبال می‌کنه)
- [مراحل Refactoring یک قطعه کد برای اعمال تزریق وابستگی‌ها](#) (در کدهای موجود چطور باید این الگو را پیاده سازی کرد و نحوه تشخیص آن به چه صورتی است)

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۰۵/۲۱ ۱۲:۲۰

" حین استفاده از تزریق وابستگی‌ها، وهله سازی به تاخیر افتاده وابستگی‌ها میسر نیست "
فکر کنم منظور تان فقط هنگام تزریق وابستگی از طریق constructor است؟ چون چنانچه از روش تزریق setter استفاده کنیم این مشکل حل میشود. درست برداشت کرده ام؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۱ ۱۲:۴۶

- خیر. [در حالت setter](#) هم IoC Container (حین استفاده از ابزارهای متداول تزریق وابستگی‌ها) تمام وابستگی‌ها را در حین وهله سازی کلاس ایجاد می‌کند (چه استفاده شوند یا خیر؛ چون اگر اینکار را انجام ندهد استفاده کننده خطای null reference exception را حتما دریافت خواهد کرد. از این جهت که فقط در حالت استفاده [از کلاس‌های Lazy است](#) که یک محصور کننده، ارجاع واقعی به شیء را مدیریت می‌کند و به تاخیر می‌اندازد). [در حالت به تاخیر افتاده](#) ، یک وابستگی فقط زمانی وهله سازی می‌شود که ارجاعی به آن در مسیر منطقی اجرای برنامه وجود داشته باشد؛ در غیراینصورت از وهله سازی آن وابستگی استفاده نشده، صرفنظر خواهد شد.
- برای آزمایش این مساله یک break point را در سازنده کلاس‌های مورد استفاده قرار دهید.

قبل از اینکه وارد بحث استفاده از کتابخانه‌های بسیار غنی IoC Container موجود شویم، بهتر است یک نمونه ساده آن‌ها را طراحی کنیم تا بهتر بتوان با عملکرد و ساختار درونی آن‌ها آشنا شد.

IoC Container چیست؟

IoC Container، فریم ورکی است برای انجام تزریق وابستگی‌ها. در این فریم ورک امکان تنظیم اولیه وابستگی‌های سیستم وجود دارد. برای مثال زمانیکه برنامه از یک IoC Container، نوع اینترفیس خاصی را درخواست می‌کند، این فریم ورک با توجه به تنظیمات اولیه‌اش، کلاسی مشخص را بازگشت خواهد داد. IoC Containerهای قدیمی‌تر، برای انجام تنظیمات اولیه خود از فایل‌های کانفیگ استفاده می‌کردند. نمونه‌های جدیدتر آن‌ها از روش‌های Fluent interfaces برای مشخص سازی تنظیمات خود بهره می‌برند.

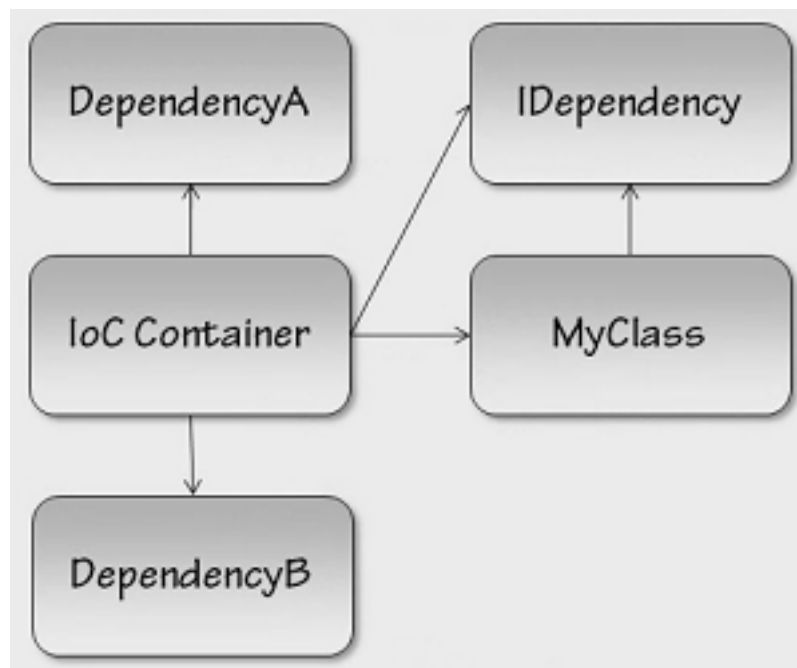
زمانیکه از یک IoC Container در کدهای خود استفاده می‌کنید، مراحل چند رخ خواهند داد:

الف) کد فراخوان، از IoC Container، یک شیء مشخص را درخواست می‌کند. عموماً اینکار با درخواست یک اینترفیس صورت می‌گیرد؛ هرچند محدودیتی نیز نداشته و امکان درخواست یک کلاس از نوعی مشخص نیز وجود دارد.

ب) در ادامه IoC Container به لیست اشیاء قابل ارائه توسط خود نگاه کرده و در صورت وجود، وهله سازی شیء درخواست شده را انجام و نهایتاً شیء مطلوب را بازگشت خواهد داد.

در این بین زنجیره‌ی وابستگی‌های مورد نیاز نیز وهله سازی خواهند شد. برای مثال اگر وابستگی اول به وابستگی دوم برای وهله سازی نیاز دارد، کار وهله سازی وابستگی‌های وابستگی دوم نیز به صورت خودکار انجام خواهند شد. (این موردی است که بسیاری از تازه واردان به این بحث تا یکبار آن‌را امتحان نکنند باور نخواهند کرد!)

ج) سپس کد فراخوان وهله دریافتی را مورد پردازش قرار داده و سپس شروع به استفاده از متدها و خواص آن خواهد نمود.



در تصویر فوق محل قرارگیری یک IoC Container را مشاهده می‌کنید. یک IoC Container در مورد تمام وابستگی‌های مورد نیاز،

اطلاعات لازم را دارد. همچنین این فریم ورک در مورد کلاسی که قرار است از وابستگی‌های سیستم استفاده نماید نیز مطلع است؛ به این ترتیب می‌تواند به صورت خودکار در زمان و هله سازی آن، نوع‌های وابستگی‌های مورد نیاز آن را در اختیارش قرار دهد. برای مثال در اینجا MyClass، وابستگی مشخص شده در سازنده خود را به نام IDependency از IoC Container درخواست می‌کند. سپس این IoC Container بر اساس تنظیمات اولیه خود، یکی از وابستگی‌های A یا B را بازگشت خواهد داد.

آغاز به کار ساخت یک IoC Container نمونه

در ابتدا کدهای آغازین مثال بحث جاری را در نظر بگیرید:

```
using System;

namespace DI01
{
    public interface ICreditCard
    {
        string Charge();
    }

    public class Visa : ICreditCard
    {
        public string Charge()
        {
            return "Charging with the Visa!";
        }
    }

    public class MasterCard : ICreditCard
    {
        public string Charge()
        {
            return "Swiping the MasterCard!";
        }
    }

    public class Shopper
    {
        private readonly ICreditCard creditCard;

        public Shopper(ICreditCard creditCard)
        {
            this.creditCard = creditCard;
        }

        public void Charge()
        {
            var chargeMessage = creditCard.Charge();
            Console.WriteLine(chargeMessage);
        }
    }
}
```

در اینجا وابستگی‌های کلاس خریدار از طریق سازنده آن که متداول‌ترین روش تزریق وابستگی‌ها است، در اختیار آن قرار خواهد گرفت. یک اینترفیس کردیت کارت تعریف شده است به همراه دو پیاده سازی نمونه آن مانند مسترکارت و ویزا کارت. ساده‌ترین نوع فراخوانی آن نیز می‌تواند مانند کدهای ذیل باشد (تزریق وابستگی‌های دستی):

```
var shopper = new Shopper(new Visa());
shopper.Charge();
```

در ادامه قصد داریم این فراخوانی‌ها را اندکی هوشمندتر کنیم تا بتوان بر اساس تنظیمات برنامه، کار تزریق وابستگی‌ها صورت گیرد و به سادگی بتوان اینترفیس‌های متفاوتی را در اینجا درخواست و مورد استفاده قرار داد. اینجا است که به اولین IoC Container خود خواهیم رسید:

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```

namespace DI01
{
    public class Resolver
    {
        // کار ذخیره سازی و نگاشت از یک نوع به نوعی دیگر در اینجا توسط این دیکشنری انجام خواهد شد
        private Dictionary<Type, Type> dependencyMap = new Dictionary<Type, Type>();

        /// <summary>
        /// یک نوع خاص از آن درخواست شده و سپس بر اساس تنظیمات برنامه، کار و هله سازی
        /// نمونه معادل آن صورت خواهد گرفت
        /// </summary>
        public T Resolve<T>()
        {
            return (T)Resolve(typeof(T));
        }

        private object Resolve(Type typeToResolve)
        {
            Type resolvedType;

            // ابتدا بررسی می‌شود که آیا در تنظیمات برنامه نگاشت متناظری برای نوع درخواستی وجود دارد؟
            if (!dependencyMap.TryGetValue(typeToResolve, out resolvedType))
            {
                // اگر خیر، کار متوقف خواهد شد
                throw new Exception(string.Format("Could not resolve type {0}",
                    typeToResolve.FullName));
            }

            var firstConstructor = resolvedType.GetConstructors().First();
            var constructorParameters = firstConstructor.GetParameters();
            // در ادامه اگر این نوع، دارای سازنده‌ی بدون پارامتری است
            // بلافاصله و هله سازی خواهد شد
            if (!constructorParameters.Any())
                return Activator.CreateInstance(resolvedType);

            var parameters = new List<object>();
            foreach (var parameterToResolve in constructorParameters)
            {
                // در اینجا یک فراخوانی بازگشتی صورت گرفته است برای و هله سازی
                // خودکار پارامترهای مختلف سازنده یک کلاس
                parameters.Add(Resolve(parameterToResolve.ParameterType));
            }
            return firstConstructor.Invoke(parameters.ToArray());
        }

        public void Register<TFrom, TTo>()
        {
            dependencyMap.Add(typeof(TFrom), typeof(TTo));
        }
    }
}

```

در اینجا کدهای کلاس Resolver یا همان IoC Container ابتدایی بحث را مشاهده می‌کنید. توضیحات قسمت‌های مختلف آن به صورت کامنت ارائه شده‌اند.

```

var resolver = new Resolver();
// تنظیمات اولیه
resolver.Register<Shopper, Shopper>();
resolver.Register<ICreditCard, Visa>();
// تزریق وابستگی‌ها و و هله سازی
var shopper = resolver.Resolve<Shopper>();
shopper.Charge();

```

در ادامه نحوه استفاده از IoC Container ایجاد شده را مشاهده می‌کنید. ابتدا کار تعریف نگاشت‌های اولیه انجام می‌شود. در این صورت زمانیکه متد Resolve فراخوانی می‌گردد، نوع درخواستی آن به همراه سازنده دارای آرگومانی از نوع ICreditCard و هله سازی شده و بازگشت داده خواهد شد. سپس با در دست داشتن یک و هله آماده، متد Charge آنرا فراخوانی خواهیم کرد.

بررسی نحوه استفاده از Microsoft Unity به عنوان یک IoC Container

Unity چیست؟

[Unity](#) یک فریم ورک IoC Container تهیه شده توسط مایکروسافت می باشد که آن را به عنوان جزئی از Enterprise Library خود قرار داده است. بنابراین برای دریافت آن یا می توان کل مجموعه Enterprise Library را دریافت کرد و یا به صورت مجزا به عنوان [یک بسته نیوگت](#) نیز قابل تهیه است. برای این منظور در خط فرمان پاورشل نیوگت در VS.NET دستور ذیل را اجرا کنید:

```
PM> Install-Package Unity
```

پیاده سازی مثال خریدار توسط Unity

همان مثال قسمت قبل را در نظر بگیرید. قصد داریم اینبار بجای IoC Container دست سازی که تهیه شد، پیاده سازی آن را به کمک MS Unity انجام دهیم.

```
using Microsoft.Practices.Unity;

namespace DI02
{
    class Program
    {
        static void Main(string[] args)
        {
            var container = new UnityContainer();

            container.RegisterType<ICreditCard, MasterCard>();

            var shopper = container.Resolve<Shopper>();
            shopper.Charge();
        }
    }
}
```

همانطور که ملاحظه می کنید، API آن بسیار شبیه به کلاس دست سازی است که در قسمت قبل تهیه کردیم. مطابق کدهای فوق، ابتدا تنظیمات IoC Container انجام شده است. به آن اعلام کرده ایم که در صورت نیاز به ICreditCard، نوع MasterCard را یافته و وهله سازی کن. با این تفاوت که Unity هوشمندتر بوده و سطر مربوط به ثبت کلاس Shoper ایی را که در قسمت قبل انجام دادیم، در اینجا حذف شده است.

سپس به این IoC Container اعلام کرده ایم که نیاز به یک وهله از کلاس خریدار داریم. در اینجا Unity کار وهله سازی های خودکار وابستگی ها و تزریق آن ها را در سازنده کلاس خریدار انجام داده و نهایتاً یک وهله قابل استفاده را در اختیار ادامه برنامه قرار خواهد داد.

یک نکته:

به صورت پیش فرض کار تزریق وابستگی ها در سازنده کلاس ها به صورت خودکار انجام می شود. اگر نیاز به Setter injection و مقدار دهی خواص کلاس وجود داشت می توان به نحو ذیل عمل کرد:

```
container.RegisterType<ICreditCard, MasterCard>(new InjectionProperty("propertyName", 5));
```

نام خاصیت و مقدار مورد نظر به عنوان پارامتر متد RegisterType باید تعریف شوند.

مدیریت طول عمر اشیاء در Unity

توسط یک IoC Container می توان یک وهله معمولی از شیء ایی را درخواست کرد و یا حتی طول عمر این وهله را به صورت Singleton معرفی نمود (یک وهله در طول عمر کل برنامه). در Unity اگر تنظیم خاصی اعمال نشود، هربار که متد Resolve

فراخوانی می‌گردد، یک وهله جدید را در اختیار ما قرار خواهد داد. اما اگر پارامتر متد RegisterType را با وهله‌ای از ContainerControlledLifetimeManager مقدار دهی کنیم:

```
container.RegisterType<ICreditCard, MasterCard>(new ContainerControlledLifetimeManager());
```

از این پس با هربار فراخوانی متد Resolve، در صورت نیاز به وابستگی از نوع ICreditCard، تنها یک وهله مشترک از MasterCard ارائه خواهد شد.

حالت پیش فرض مورد استفاده، بدون ذکر پارامتر متد RegisterType، مقدار TransientLifetimeManager می‌باشد.

نظرات خوانندگان

نویسنده: مهدی فرهانی
تاریخ: ۱۳۹۲/۰۱/۲۶ ۱:۰۶

به نظر شما از بین فریم ورک‌ها موجود کدام یک بهتره ؟ مخصوصاً مقایسه ای بین Unity ، Ninject و StrucuterMap اگر داشته باشیم خیلی بهتره

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۲۶ ۱:۱۳

من StructureMap رو ترجیح می‌دم. خیلی‌ها هم همین نظر رو دارند:

[IoC libraries compared](#)

[Which .NET Dependency Injection frameworks are worth looking into](#)

نویسنده: سیروس
تاریخ: ۱۳۹۲/۰۱/۲۸ ۱۸:۲۸

با وجود اینکه ما خودمان می‌تونیم مانند کد زیر کار و هله سازی را انجام دهیم

```
var shopper = new Shopper(new Visa());
shopper.Charge();
```

چه لزومی به استفاده از IoC Container و کد

```
var resolver = new Resolver();
//تنظیمات اولیه
resolver.Register<Shopper, Shopper>();
resolver.Register<ICreditCard, Visa>();
//تزریق وابستگی‌ها و و هله سازی
var shopper = resolver.Resolve<Shopper>();
shopper.Charge();
```

وجود دارد، شاید بگید : " اگر وابستگی اول به وابستگی دوم برای و هله سازی نیاز دارد، کار و هله سازی وابستگی‌های وابستگی دوم نیز به صورت خودکار انجام خواهند شد. " میشه یه مثال ملموس‌تر بزنیم.
من یه خورده گیج شدم!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۲۸ ۲۰:۵۹

پاسخ به این سؤال نیاز به مطالعه قسمت‌های بعدی دارد.

- هدف از قسمت جاری آشنایی اولیه با مراحل ابتدایی کار با یک IoC Container است.

- در حالت اول هنوز شما هستید که مسئول و هله سازی‌های اولیه می‌باشید و کار و هله سازی را به لایه‌ای دیگر واگذار نکرده‌اید.

- در کد حالت اول نمی‌شود این وابستگی ارسالی به سازنده کلاس را به سادگی تعویض کرد. در حالیکه با استفاده از یک IoC container فقط کافی است تنظیمات اولیه نگاشت‌های آنرا مشخص کنیم تا نوع کلاسی که باید در سازنده‌ها تزریق شوند مشخص شود. مزیت اینکار ساده‌تر شدن نوشتن آزمون‌های واحد و تهیه کلاس‌های Fake است؛ بدون نیازی به تغییری حتی در حد یک سطر در کدهای اصلی برنامه.

- در مورد و هله سازی خودکار چند سطح وابستگی‌ها، در قسمت‌های بعد تحت عنوان Object graph بیشتر بحث شده است و

مثال زده شده. همیشه با یک کلاس ساده ویزا مانند مثال فوق سر و کار نداریم. عموماً با سرویس‌هایی سر و کار داریم که

خودشان نیز از سرویس‌های دیگری استفاده می‌کنند. برای مثال یک سرویس ارسال ایمیل از سرویس کاربران برای دریافت

ایمیل‌های کاربران کمک می‌گیرد. وهله سازی تمام این وابستگی‌ها را در چند سطح می‌شود با استفاده از IoC Containers خودکار کرد و به کدهای نهایی بسیار تمیزتری رسید.

- در حالت اول، طول عمر یک شیء را نمی‌شود مشخص کرد (یا حداقل نیاز به کد نویسی قابل توجهی دارد). برای مثال با استفاده از یک IoC Container می‌شود وهله ایجاد شده را Singleton کرد تا در سراسر برنامه یک وهله از آن استفاده شود یا حتی می‌شود در طول یک درخواست رسیده وب، یک وهله را در اختیار تمام کلاس‌های درگیر قرار داد. به این ترتیب به سربار کمتری در حالت‌های خاصی مانند وهله سازیObjectContext یا DbContext در EF خواهیم رسید.

- زمان استفاده از IoC Container ها کارهای فراتری از تزریق وابستگی‌ها را هم می‌شود انجام داد. برای مثال فراخوانی‌های متدها را هم تحت نظر قرار داد (برنامه نویسی AOP یا جنبه گرا) و مثلاً بدون نوشتن کد اضافه‌ای در برنامه، خروجی متدها را کش کرد. AOP یک سری بحث مفصل را در طی یک دوره جدا به همراه دارد.

طوری با IoC Containers کار کنید که انگار وجود خارجی ندارند

تفاوت پایه‌ای که بین یک فریم ورک IoC و سایر فریم ورک‌ها وجود دارد، در معکوس شدن مسئولیت‌ها است. در اینجا لایه‌های مختلف برنامه شما نیستند که فریم ورک IoC را فراخوانی می‌کنند؛ بلکه این فریم ورک IoC است که از جزئیات ارتباطات و وابستگی‌های سیستم شما آگاه است و نهایتاً کار کنترل و هله سازی اشیاء مختلف را عهده دار خواهد شد. طول عمر آن‌ها را تنظیم کرده یا حتی در بعضی از موارد مانند برنامه نویسی جنبه‌گرا یا AOP، نسبت به تزئین این اشیاء یا دخالت در مراحل مختلف فراخوانی متدهای آن‌ها نیز نقش خواهد داشت. نکته‌ی مهم در اینجا، نا آگاهی برنامه از حضور آن‌ها است. بنابراین در پروژه شما اگر ماژول‌ها و لایه‌های مختلفی حضور دارند، تنها برنامه اصلی است که باید ارجاعی را به فریم ورک IoC داشته باشد و نه سایر لایه‌های سیستم. علت حضور آن در ریشه سیستم نیز تنها باید به اصطلاحا bootstrapping و اعمال تنظیمات مرتبط با آن خلاصه شود.

به عبارتی استفاده صحیح از یک فریم ورک IoC نباید به شکل الگوی Service Locator باشد؛ حالتی که در تمام قسمت‌های برنامه مدام مشاهده می‌کنید resolver.Resolve, resolver.Resolve و الی آخر. باید از این نوع استفاده از فریم ورک‌های IoC تا حد ممکن حذر شود و کدهای برنامه نباید وابستگی مستقیم ثانویه‌ای را به نام خود فریم ورک IoC پیدا کنند.

```
var container = BootstrapContainer();
var finder = container.Resolve<IDuplicateFinder>();
var processor = container.Resolve<IArgumentsParser>();

Execute( args, processor, finder );

container.Dispose();
```

نمونه‌ای از نحوه صحیح استفاده از یک IoC Container را مشاهده می‌کنید. تنها در سه نقطه است که یک IoC container باید حضور پیدا کند:

- الف) در آغاز برنامه برای اعمال تنظیمات اولیه و bootstrapping
- ب) پیش از اجرای عملی جهت و هله سازی وابستگی‌های مورد نیاز
- ج) پس از اجرای عمل مورد نظر جهت آزاد سازی منابع

نکته مهم اینجا است که در حین اجرای فرآیند، این فرآیند باید تا حد ممکن از حضور IoC container بی‌خبر باشد و کار تشکیل اشیاء باید خارج از منطق تجاری برنامه انجام شود: IoC container خود را صدا زنید؛ او شما را صدا خواهد زد. عنوان شد تا «حد ممکن». این تا حد ممکن به چه معنایی است؟ اگر کار و هله سازی اشیاء را می‌توانید تحت کنترل قرار دهید، مثلاً آیا می‌توانید در نحوه و هله سازی کنترلرها در ASP.NET MVC دخل و تصرف کرده و در زمان و هله سازی، اینکار را به یک IoC Container واگذار کنید؟ اگر بلی، دیگر به هیچ عنوانی نباید داخل کلاس‌های فراخوانی شده و تزریق شده به کنترلرهای برنامه اثری از IoC Container شما مشاهده شود. زیرا این فریم ورک‌ها اینقدر توانمند هستند که بتوانند تا چندین لایه از سیستم را واکاوی کرده و وابستگی‌های لازم را و هله سازی کنند.

اگر خیر (نمی‌توانید کار و هله سازی اشیاء را مستقیماً تحت کنترل قرار دهید)؛ مانند تهیه یک Role Provider سفارشی در ASP.NET MVC که کار و هله سازی این Role Provider را توسط موتور ASP.NET انجام می‌شود و در این بین امکان دخل و تصرفی هم در آن ممکن نیست، آنگاه مجاز است داخل این کلاس ویژه از متدهای container.Resolve استفاده کرد؛ چون چاره‌ی دیگری وجود ندارد و IoC Container نیست که کار و هله سازی ابتدایی آن‌را عهده دار شده است. باید دقت داشت به این حالت خاص دیگر تزریق وابستگی‌ها گفته نمی‌شود؛ بلکه نام الگوی آن [Service locator](#) است. در Service locator یک کامپوننت خودش به دنبال وابستگی‌های مورد نیازش می‌گردد. در حالت تزریق وابستگی‌ها، یک کامپوننت وابستگی‌های مورد نیاز را درخواست می‌کند.

یک مثال:

```
public class ExampleClass
{
    private readonly IService _service;

    public ExampleClass()
    {
        _service = Container.Resolve<IService>();
    }

    public void DoSomething(int id)
    {
        _service.DoSomething(id);
    }
}
```

کاری که در اینجا انجام شده است نمونه اشتباهی از استفاده از یک IoC Container می‌باشد. به صرف اینکه مشغول به استفاده از یک IoC Container هستیم به این معنا نیست که واقعا الگوی معکوس سازی وابستگی‌ها را درست درک کرده‌ایم. در اینجا الگوی Service locator مورد استفاده است و نه الگوی تزریق وابستگی‌ها. به عبارتی در مثال فوق، کلاس ExampleClass وابسته است به یک وابستگی جدیدی به نام Container، علاوه بر وابستگی IService ایی که به او قرار است خدماتی را ارائه دهد. نمونه اصلاح شده کلاس فوق، تزریق وابستگی‌ها در سازنده کلاس به نحو زیر است:

```
public class ExampleClass
{
    private IService _service;

    public ExampleClass(IService service)
    {
        _service = service;
    }

    public void DoSomething(int id)
    {
        _service.DoSomething(id);
    }
}
```

در اینجا این کلاس است که وابستگی‌های خود را درخواست می‌کند و نه اینکه خودش به دنبال آن‌ها بگردد.

نمونه دیگری از کلاسی که خودش به دنبال یافتن و وهله سازی وابستگی‌های مورد نیازش است مثال زیر می‌باشد:

```
public class Search
{
    IDinner _dinner;
    public Search(): this(new Dinner())
    { }

    public Search(IDinner dinner)
    {
        _dinner = dinner;
    }
}
```

به این کار [poor man's dependency injection](#) هم گفته می‌شود؛ اولین سازنده از طریق یک default constructor سعی کرده است وابستگی‌های کلاس را، خودش تامین کند. باز هم کلاس می‌داند که به چه وابستگی خاصی نیاز دارد و عملا معکوس سازی وابستگی‌ها رخ نداده است. همچنین استفاده از این حالت زمانیکه کلاس Dinner خودش وابستگی به کلاس‌های دیگر داشته باشد، بسیار به هم ریخته و مشکل خواهد بود. مزیت استفاده از IoC Containers وهله سازی یک large object graph کامل است. به علاوه توسط IoC Containers مدیریت طول عمر اشیاء را نیز می‌توان تحت نظر قرار داد. برای مثال می‌توان به یک IoC Container گفت تنها یک وهله از DbContext را در طول یک درخواست ایجاد و آن‌را در اختیار لایه‌های مختلف برنامه قرار بده؛ چون نیاز داریم کاری که در طی یک درخواست انجام می‌شود، در داخل یک تراکنش انجام شده و همچنین بی‌جهت به ازای هر new DbConetxt جدید، یکبار اتصالی به بانک اطلاعاتی باز و بسته نشود (سرعت بیشتر، سربار کمتر).

نظرات خوانندگان

نویسنده: مهدی فرهانی
تاریخ: ۱۳۹۲/۰۱/۲۶ ۱:۱۴

اگر ترکیبی از Service Locator و poor man's dependency injection استفاده شود چه ایراداتی دارد ؟
مثلاً این کد

```
protected readonly IUnitOfWork UnitOfWork;

protected BaseOperation():this(ObjectFactory.GetInstance<IUnitOfWork>())
{
}
protected BaseOperation(IUnitOfWork uow)
{
    UnitOfWork = uow;
}
```

به غیر از این که کلاس مورد نظر به Container وابسته هست آیا ایراد دیگری هم هست یا خیر ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۲۶ ۱:۲۰

- این بهتر است. مدیریت طول عمر اشیاء (مثلاً ایجاد یک وهله در طی یک درخواست) و همچنین وهله سازی object graph در چند سطح به صورت خودکار توسط Service Locator هم انجام می‌شود.
- ولی در کل اگر امکان وهله سازی کلاس BaseOperation توسط IoC Container به صورت مستقل وجود دارد (چیزی مثل استفاده از DefaultControllerFactory در ASP.NET MVC) بهتر است اجازه بدید خود IoC Container کار تزریق وابستگی‌ها را به صورت خودکار انجام دهد و کلاس‌ها اطلاعی از وجود آن نداشته باشند.

نویسنده: مهدی فرهانی
تاریخ: ۱۳۹۲/۰۱/۲۶ ۱:۳۳

در اصل کلاس BaseOperation یک کلاس Abstract هست که بقیه Operation‌ها از این کلاس ارثی بری میکنند.

```
public abstract class BaseOperation : IPartikanOperation
```

و هیچ وهله سازی مستقیمی از آن در برنامه صورت نمی‌گردد.

```
public class UserOperations : BaseOperation, IUserOperations
{
    private readonly IUserService _userService;
    private readonly IMessageTemplateService _messageTemplateService;

    public UserOperations(IUserService userService, IMessageTemplateService messageTemplateService)
    {
        _userService = userService;
        _messageTemplateService = messageTemplateService;
    }
}
```

راه حلی که من استفاده کردم ، استفاده از پارمتر ورودی برای کلاس‌های فرزند هست

```
public UserOperations(IUnitOfWork uow,IUserService userService, IMessageTemplateService
messageTemplateService) : base(uow)
{
    _userService = userService;
    _messageTemplateService = messageTemplateService;
}
```

با توجه به اینکه هیچ وهله سازی از کلاس پایه صورت نمیگیره ،آیا لزومی دارد که وابستگی به Container از کلاس پایه گرفته شود ؟

نویسنده: وحید نصیری
تاریخ: ۹:۵۲ ۱۳۹۲/۰۱/۲۶

لطفا متن قسمت جاری را یکبار دیگر مطالعه بفرمائید. جواب صریحی را دریافت خواهید کرد.
(قسمت‌های وهله سازی خودکار وابستگی‌های کلاس‌های به هم وابسته (منظور از Object graph)؛ به علاوه امکان تعریف طول عمر یک شیء طوریکه هربار وهله سازی نشود (مثلا فقط در طول یک درخواست در تمام کلاس‌های وابسته به صورت یک وهله مشترک در دسترس باشد؛ مفید برای حالت استفاده از الگوی واحد کار). همچنین الگوی Service locator و فرق آن با تزریق وابستگی‌ها. مواردی که شاید یکی به نظر به رسند اما یکی نیستند)

نویسنده: رضا بزرگی
تاریخ: ۱:۲۶ ۱۳۹۲/۰۲/۰۷

لطفا در این مورد " تفاوت پایه‌ای که بین یک فریم ورک IoC و سایر فریم ورک‌ها وجود دارد، در معکوس شدن مسئولیت‌ها است. " بیشتر توضیح دهید.
- چه چیزی عامل برتری structuremap برای انتخاب شماست. و کلا چه تفاوت‌هایی با هم دارند. مثلا با ninject.
- آیا با توجه به ویژگی‌های جدید نسخه 3 unity که به تازگی منتشر شده و از طرفی بومی بودن اون، میشه گفت ارزش امتحان کردن داره یا خیر.
ممنونم.

نویسنده: وحید نصیری
تاریخ: ۸:۷ ۱۳۹۲/۰۲/۰۷

- هر دوره قسمت اختصاصی رو داره به نام « [پرسش و پاسخ](#) » برای طرح این نوع سؤالات خارج از موضوع مطلب جاری، اما مرتبط با عنوان دوره.
- در مورد معکوس شدن مسئولیت‌ها به تفصیل در سه قسمت اول [این دوره](#) مطلب نوشته شده است؛ پیش از شروع به کد نویسی.
- من StructureMap رو ترجیح می‌دم. خیلی‌ها هم همین نظر رو دارند:
[IoC libraries compared](#)
[Which .NET Dependency Injection frameworks are worth looking into](#)
- این مورد بیشتر سلیقه‌ای هست.

نویسنده: وحید م
تاریخ: ۲۳:۴۰ ۱۳۹۲/۰۷/۱۹

با سلام
بنده structuremap را از نوگت گرفتم پوشه ای برایم ایجاد شده که حاوی دوکلاس بود یکی IoC.cs و SmDependencyResolver.cs سوالی که داشتم آیا IOC همان servicelocator است؟
آیا ObjectFactory.GetInstance همان کار servicelocator را انجام می‌دهد؟
آیا `var processor = container.Resolve<IArgumentsParser>` هم همان کار servicelocator را انام می‌دهد

نویسنده: وحید نصیری
تاریخ: ۲۳:۴۸ ۱۳۹۲/۰۷/۱۹

- خیر. به زبان ساده اگر وابستگی‌ها از طریق سازنده کلاس یا خواص آن در اختیار کلاس قرار گیرنده و در این بین ابزاری یا کتابخانه‌ای این تزریق را انجام دهد، به آن ابزار IoC Container می‌گویند. اگر در یک کلاس مستقیما از امکانات IoC Container

برای دریافت وابستگی‌ها استفاده شود، الگوی Service locator نام دارد و در این حالت خود IoC Container یک وابستگی در طراحی شما به حساب می‌آید.

- بله و خیر. بله اگر مستقیماً داخل یک کلاس مثلاً لایه سرویس یا یک کنترلر و امثال آن استفاده شود. اگر از آن در یک کلاس فکتوری مانند که کار وهله سازی مثلاً کنترلرها و امثال آن را عهده دار است، استفاده شود دیگر الگوی Service locator نیست و تزریق وابستگی‌های استاندارد است.
- بله و خیر. مانند قبل.

نویسنده: ناظم

تاریخ: ۱۳۹۲/۱۱/۱۵ ۱۴:۱۳

با سلام؛ یعنی اگر من در یک برنامه mvc و در یک کنترلر، از IoC container مستقیماً برای تولید اشیا استفاده کنم در واقع از تزریق وابستگی‌ها استفاده نمیکنم و دارم از الگوی Service locator استفاده میکنم؟

```
public partial class Test : Controller
{
    private IUnitOfWork _uow;
    private IService _Service;

    public Test()
    {
        _uow = ObjectFactory.GetInstance<IUnitOfWork>();
        _userService = ObjectFactory.GetInstance<IService>();
    }

    // Other Methods
}
```

مثلاً در کلاس Service که اصلاً از IoC container مستقیماً استفاده نشده، و هنگام ایجاد شی در کنترلر به صورت خودکار وابستگی‌ش تامین میشود، الگوی تزریق وابستگی‌ها درست پیاده شد؟

نویسنده: وحید نصیری

تاریخ: ۱۳۹۲/۱۱/۱۵ ۱۴:۲۱

خیر. این روش service locator است و در MVC قابل بهبود است. یک مطلب کامل در مورد آن داریم:
« [تزریق خودکار وابستگی‌ها در برنامه‌های ASP.NET MVC](#) »

نویسنده: ناظم

تاریخ: ۱۳۹۲/۱۱/۱۵ ۱۴:۳۸

برای این که مطمئن بشم که آیا DI رو به صورت صحیح پیاده کردم یا نه آیا چک لیستی یا روشی برای این کار هست؟

برای مثال الان دارم فکر میکنم که چطور این کارو تو winForms میشه انجام داد، پس از انجام، آیا کارم درست هست یا نه؟ روشی برای حصول اطمینان هست؟

نویسنده: وحید نصیری

تاریخ: ۱۳۹۲/۱۱/۱۵ ۱۴:۴۱

مطالب و مفاهیم همین مقاله جاری و سطر اول آن یعنی «طوری با IoC Containers کار کنید که انگار وجود خارجی ندارند» کافی است.

[StructureMap](#) یکی از IoC container های بسیار غنی سورس باز نوشته شده برای دات نت فریم ورک است. امکان تنظیمات آن توسط کدنویسی و یا همان Fluent interfaces، به کمک فایل‌های کانفیگ XML و همچنین استفاده از ویژگی‌ها یا Attributes نیز میسر است. امکانات جانبی دیگری را نیز مانند یکی شدن با فریم ورک‌های [Dynamic Proxy](#) برای ساده سازی فرآیندهای برنامه نویسی جنبه‌گرا یا AOP، دارا است. در ادامه قصد داریم با نحوه استفاده از این فریم ورک IoC بیشتر آشنا شویم.

دریافت StructureMap

برای دریافت آن نیاز است دستور پاورشل ذیل را در کنسول [نیوگت](#) ویزوال استودیو فراخوانی کنید:

```
PM> Install-Package structuremap
```

البته باید دقت داشت که برای استفاده از StructureMap نیاز است به خواص پروژه مراجعه و سپس حالت Client profile را به Full profile تغییر داد تا برنامه قابل کامپایل باشد (در برنامه‌های دسکتاپ البته)؛ از این جهت که StructureMap ارجاعی را به اسمبلی استاندارد System.Web دارد.

آشنایی با ساختار برنامه

ابتدا یک برنامه کنسول را آغاز کرده و سپس یک Class library جدید را به نام Services نیز به آن اضافه کنید. در ادامه کلاس‌ها و اینترفیس‌های زیر را به Class library ایجاد شده، اضافه کنید. سپس از طریق نیوگت به روشی که گفته شد، StructureMap را به پروژه اصلی (ونه پروژه Class library) اضافه نمائید و Target framework آن را نیز در حالت Full قرار دهید بجای حالت Client profile.

```
namespace DI03.Services
{
    public interface IUsersService
    {
        string GetUserEmail(int userId);
    }
}

namespace DI03.Services
{
    public interface IEmailsService
    {
        void SendEmailToUser(int userId, string subject, string body);
    }
}

using System;

namespace DI03.Services
{
    public class UsersService : IUsersService
    {
        public UsersService()
        {
            // هدف صرفا نمایش وهله سازی خودکار این وابستگی است
            Console.WriteLine("UsersService ctor.");
        }

        public string GetUserEmail(int userId)
        {
            // برای مثال دریافت از بانک اطلاعاتی و بازگشت یک نمونه جهت آزمایش برنامه
            return "name@site.com";
        }
    }
}
```



```

}
using System;
namespace DI03.Services
{
    public class EmailService: IEmailService
    {
        private readonly IUsersService _userService;
        public EmailService(IUsersService userService)
        {
            Console.WriteLine("EmailService ctor.");
            _userService = userService;
        }

        public void SendEmailToUser(int userId, string subject, string body)
        {
            var email = _userService.GetUserEmail(userId);
            Console.WriteLine("SendEmailTo({0})", email);
        }
    }
}

```

در لایه سرویس برنامه، یک سرویس کاربران و یک سرویس ارسال ایمیل تدارک دیده شده‌اند. سرویس کاربران بر اساس آی دی یک کاربر، برای مثال از بانک اطلاعاتی ایمیل او را بازگشت می‌دهد. سرویس ارسال ایمیل، نیاز به ایمیل کاربری برای ارسال ایمیلی به او دارد. بنابراین وابستگی مورد نیاز خود را از طریق تزریق وابستگی‌ها در سازنده کلاس و وهله سازی شده در خارج از آن (معکوس سازی کنترل)، دریافت می‌کند. در سازنده‌های هر دو کلاس سرویس نیز از Console.WriteLine استفاده شده‌است تا زمان وهله سازی خودکار آن‌ها را بتوان بهتر مشاهده کرد. نکته مهمی که در اینجا وجود دارد، بی‌خبری لایه سرویس از وجود IoC Container مورد استفاده است.

استفاده از لایه سرویس و تزریق وابستگی‌ها به کمک StructureMap

```

using DI03.Services;
using StructureMap;
namespace DI03
{
    class Program
    {
        static void Main(string[] args)
        {
            // تنظیمات اولیه برنامه که فقط یکبار باید در طول عمر برنامه انجام شود
            ObjectFactory.Initialize(x =>
            {
                x.For<IEmailService>().Use<EmailService>();
                x.For<IUsersService>().Use<UsersService>();
            });

            // نمونه‌ای از نحوه استفاده از تزریق وابستگی‌های خودکار
            var emailService = ObjectFactory.GetInstance<IEmailService>();
            emailService.SendEmailToUser(userId: 1, subject: "Test", body: "Hello!");
        }
    }
}

```

کدهای برنامه را به نحو فوق تغییر دهید. در ابتدا نحوه سیم کشی‌های آغازین برنامه را مشاهده می‌کنید. برای مثال کدهای ObjectFactory.Initialize باید در متدهای آغازین یک پروژه قرار گیرند و تنها یکبار هم نیاز است فراخوانی شوند. به این ترتیب IoC Container ما زمانیکه قرار است object graph مربوط به IEmailService درخواستی را تشکیل دهد، خواهد دانست ابتدا به سازنده‌ی کلاس EmailService می‌رسد. در اینجا برای وهله سازی این کلاس به صورت خودکار، باید وابستگی‌های آن را نیز وهله سازی کند. بنابراین بر اساس تنظیمات آغازین برنامه می‌داند که باید از کلاس UsersService برای تزریق خودکار وابستگی‌ها در سازنده کلاس ارسال ایمیل استفاده نماید. در این حالت اگر برنامه را اجرا کنیم، به خروجی زیر خواهیم رسید:

```

UserService ctor.
EmailsService ctor.
SendEmailTo(name@site.com)

```

بنابراین در اینجا با مفهوم Object graph نیز آشنا شدیم. فقط کافی است وابستگی‌ها را در سازنده‌های کلاس‌ها تعریف کرده و سیم‌کشی‌های آغازین صحیحی را نیز در ابتدای برنامه معرفی نمائیم. کار وهله سازی چندین سطح با تمام وابستگی‌های متناظر با آن‌ها در اینجا به صورت خودکار انجام خواهد شد و نهایتاً یک شیء قابل استفاده بازگشت داده می‌شود. ابتدایی‌ترین مزیت استفاده از تزریق وابستگی‌ها امکان تعویض آن‌ها است؛ خصوصاً در حین Unit testing. اگر کلاسی برای مثال قرار است با شبکه کار کند، می‌توان پیاده سازی آن‌را با یک نمونه اصطلاحاً Fake جایگزین کرد و در این نمونه تنها نتیجه‌ی کار را بازگشت داد. کلاس‌های لایه سرویس ما تنها با اینترفیس‌ها کار می‌کنند. این تنظیمات قابل تغییر اولیه IoC container مورد استفاده هستند که مشخص می‌کنند چه کلاس‌هایی باید در سازنده‌های کلاس‌ها تزریق شوند.

تعیین طول عمر اشیاء در StructureMap

برای اینکه بتوان طول عمر اشیاء را بهتر توضیح داد، کلاس سرویس کاربران را به نحو زیر تغییر دهید:

```

using System;

namespace DI03.Services
{
    public class UserService : IUserService
    {
        private int _i;
        public UserService()
        {
            // هدف صرفاً نمایش وهله سازی خودکار این وابستگی است؛
            Console.WriteLine("UserService ctor.");
        }

        public string GetUserEmail(int userId)
        {
            _i++;
            Console.WriteLine("i:{0}", _i);
            // برای مثال دریافت از بانک اطلاعاتی و بازگشت یک نمونه جهت آزمایش برنامه؛
            return "name@site.com";
        }
    }
}

```

به عبارتی می‌خواهیم بدانیم این کلاس چه زمانی وهله سازی مجدد می‌شود. آیا در حالت فراخوانی ذیل،

```

// نمونه‌ای از نحوه استفاده از تزریق وابستگی‌های خودکار
var emailsService1 = ObjectFactory.GetInstance<IEmailsService>();
emailsService1.SendEmailToUser(userId: 1, subject: "Test1", body: "Hello!");

var emailsService2 = ObjectFactory.GetInstance<IEmailsService>();
emailsService2.SendEmailToUser(userId: 1, subject: "Test2", body: "Hello!");

```

ما شاهد چاپ عدد 2 خواهیم بود یا عدد یک:

```

UserService ctor.
EmailsService ctor.
i:1
SendEmailTo(name@site.com)
UserService ctor.
EmailsService ctor.
i:1
SendEmailTo(name@site.com)

```

همانطور که ملاحظه می‌کنید، به ازای هربار فراخوانی ObjectFactory.GetInstance، یک وهله جدید ایجاد شده است. بنابراین مقدار i در هر دو بار مساوی عدد یک است.

اگر به هر دلیلی نیاز بود تا این رویه تغییر کند، می‌توان بر روی طول عمر اشیاء تشکیل شده نیز تاثیر گذار بود. برای مثال تنظیمات آغازین برنامه را به نحو ذیل تغییر دهید:

```
// تنظیمات اولیه برنامه که فقط یکبار باید در طول عمر برنامه انجام شود
ObjectFactory.Initialize(x =>
{
    x.For<IEmailsService>().Use<EmailsService>();
    x.For<IUsersService>().Singleton().Use<UsersService>();
});
```

اینبار اگر برنامه را اجرا کنیم، به خروجی ذیل خواهیم رسید:

```
UsersService ctor.
EmailsService ctor.
i:1
SendEmailTo(name@site.com)
EmailsService ctor.
i:2
SendEmailTo(name@site.com)
```

بله. با Singleton معرفی کردن تنظیمات UsersService، تنها یک وهله از این کلاس ایجاد خواهد شد و نهایتاً در فراخوانی دوم ObjectFactory.GetInstance، شاهد عدد i مساوی 2 خواهیم بود (چون از یک وهله استفاده شده است).

حالت‌های دیگر تعیین طول عمر مطابق متدهای زیر هستند:

```
Singleton()
HttpContextScoped()
HybridHttpOrThreadLocalScoped()
```

با انتخاب حالت HttpContext، به ازای هر HttpContext ایجاد شده، کلاس معرفی شده یکبار وهله سازی می‌گردد. در حالت ThreadLocal، به ازای هر Thread، وهله‌ای متفاوت در اختیار مصرف کننده قرار می‌گیرد. حالت Hybrid ترکیبی است از حالت‌های HttpContext و ThreadLocal. اگر برنامه وب بود، از HttpContext استفاده خواهد کرد در غیراینصورت به ThreadLocal سوئیچ می‌کند.

شاید بپرسید که کاربرد مثلاً HttpContextScoped در کجا است؟ در یک برنامه وب نیاز است تا یک وهله از DbContext (مثلاً Entity framework) را در اختیار کلاس‌های مختلف لایه سرویس قرار داد. به این ترتیب چون هربار new Context صورت نمی‌گیرد، هربار هم اتصال جداگانه‌ای به بانک اطلاعاتی باز نخواهد شد. نتیجه آن رسیدن به یک برنامه سریع، با سربار کم و همچنین کار کردن در یک تراکنش واحد است. چون هربار فراخوانی new Context به معنای ایجاد یک تراکنش جدید است.

همچنین در این برنامه وب قصد نداریم از حالت طول عمر Singleton استفاده کنیم، چون در این حالت یک وهله از Context در اختیار تمام کاربران سایت قرار خواهد گرفت (و DbContext به صورت Thread safe طراحی نشده است). نیاز است به ازای هر کاربر و به ازای طول عمر هر درخواست، تنها یکبار این وهله سازی صورت گیرد. بنابراین در این حالت استفاده از HttpContextScoped توصیه می‌شود. به این ترتیب در طول عمر کوتاه Object graph‌های تشکیل شده، فقط یک وهله از DbContext ایجاد و استفاده خواهد شد که بسیار مقرون به صرفه است. مزیت دیگر مشخص سازی طول عمر به نحو HttpContextScoped، امکان Dispose خودکار آن به صورت زیر است:

```
protected void Application_EndRequest(object sender, EventArgs e)
{
    ObjectFactory.ReleaseAndDisposeAllHttpScopedObjects();
}
```

اگر نام اینترفیس‌های شما فقط یک I در ابتدا بیشتر از نام کلاس‌های متناظر با آن‌ها دارد، مثلاً مانند I Test و کلاس Test هستند؛ فقط کافی است از قراردادهای پیش فرض StructureMap برای اسکن یک یا چند اسمبلی استفاده کنیم:

```
// تنظیمات اولیه برنامه که فقط یکبار باید در طول عمر برنامه انجام شود
ObjectFactory.Initialize(x =>
{
    //x.For<IEmailsService>().Use<EmailsService>();
    //x.For<IUsersService>().Singleton().Use<UsersService>();
    x.Scan(scan =>
    {
        scan.AssemblyContainingType<IEmailsService>();
        scan.WithDefaultConventions();
    });
});
```

در این حالت دیگر نیازی نیست به ازای اینترفیس‌های مختلف و کلاس‌های مرتبط با آن‌ها، تنظیمات اضافه‌تری را تدارک دید. کار یافتن و برقراری اتصالات لازم در اینجا خودکار خواهد بود.

دریافت مثال قسمت جاری

[DI03.zip](#)

نظرات خوانندگان

نویسنده: فرشید علی اکبری
تاریخ: ۱۱:۱۱ ۱۳۹۲/۰۱/۲۷

سلام

بسیار عالی ... مفاهیم اساسی و پایه ای برای درک بهتر استفاده از StructureMap و مخصوصا قسمت scan آن برای من خیلی جالب بود.

نویسنده: فرشید علی اکبری
تاریخ: ۱۳:۴۲ ۱۳۹۲/۰۱/۲۷

مجددا خسته نباشید

مهندس جان کد زیر رو نگاه کنین :

```
ObjectFactory.Configure(c =>
{
    c.For<IUnitOfWork>().CacheBy(InstanceScope.Hybrid).Use<My_Context>();
    c.For<ICityService>().Use<EFCityService>();
    c.For<ICountryService>().Use<EFCountryService>();
});
```

درحالیکه Resharper میگوید که CachBy منسوخ شده و باید از LifecycleIs استفاده کنی میخوام بدونم چطوری باید تغییرش بدم که حالت InstanceScope.Hybrid هم براش تعیین شده باشه؟ درضمن آگاه رفرتس خوبی برای تسلط به کار با StructureMap درنظر دارین رو محبت کنین ولینکش رو بذارین ممنون میشم. با تشکر.

نویسنده: وحید نصیری
تاریخ: ۱۳:۴۸ ۱۳۹۲/۰۱/۲۷

- مآخذ خوب، [مستندات رسمی](#) آن است.

- توضیح دادم در متن. از متد HybridHttpOrThreadLocalScoped استفاده کنید. تمام این حالت‌ها خلاصه شدن به سه متد زیر: (حالت هیبرید، بسته به نوع ویندوزی یا وب بودن برنامه به صورت خودکار نوع بهینه رو انتخاب می‌کنه)

```
Singleton()
HttpContextScoped()
HybridHttpOrThreadLocalScoped()
//مثال
x.For<IUsersService>().HybridHttpOrThreadLocalScoped().Use<UsersService>();
```

نویسنده: رضا بزرگی
تاریخ: ۲۰:۳۰ ۱۳۹۲/۰۲/۰۷

تعیین طول عمر اشیاء در حالت تنظیمات اولیه خودکار به چه صورت است؟ آیا در این روش میشود برای هر کلاس طول عمرهای متفاوتی تعریف کرد؟

نویسنده: وحید نصیری
تاریخ: ۲۱:۲۹ ۱۳۹۲/۰۲/۰۷

به این ترتیب:

```
scan.ConnectImplementationsToTypesClosing(typeof(ITestService))
    .OnAddedPluginTypes(y => y.HybridHttpOrThreadLocalScoped());
```

نویسنده: فرشید علی اکبری
تاریخ: ۹۰:۵۲ ۱۳۹۲/۰۲/۱۸

من تا روز قبل هیچ مشکلی در استفاده از StructureMap نداشتم و از کار کردن باهاش لذت می بردم ولی با استفاده از NuGet دیروز آخرین نسخه StructureMap رو از سایت گرفتم و موقع اجرای برنامه به محض اولین وهله سازی (uow) اشکال میگیره و پیغام Error in the application میده.... ولی اگه دستی و بدون استفاده از StructureMap وهله سازی کنم کارها روی روال پیش میره.... Error Code =205 و توی سایتها زیاد سرچ زدم ولی موارد قید شده راهگشای این مشکل نبود...درضمن توی Error Details هم پیغام زیر رو نمایش میده :

```
StructureMap Exception Code: 205
Missing requested Instance property "path" for InstanceKey "fcfc9943-37d0-45d3-aebc-dad30fe29e59"
```

که اگه ConnectionString من مشکل داره پس چرا بدون استفاده از StructureMap اون وهله سازی میکنه؟
و کدهای ایجاد و وهله سازی :

```
ObjectFactory.Initialize(x =>
{
x.For<IUnitOfWorkCentralSystem>().HybridHttpOrThreadLocalScoped().Use<ContextCentralSystem>();
    x.For<IYearService>().Use<YearService>();
    x.For<IUserService>().Use<UsersService>();
});
var _uow = ObjectFactory.GetInstance<IUnitOfWorkCentralSystem>();
```

پیشاپیش ممنون از همکاری شما.

نویسنده: وحید نصیری
تاریخ: ۱۰:۱۷ ۱۳۹۲/۰۲/۱۸

آیا کلاس ContextCentralSystem دارای سازنده ای است با پارامتر path که باید مقدار دهی شود؟ اگر بله روش کار شبیه به کد زیر است:

```
x.For<IUnitOfWorkCentralSystem>()
    .HybridHttpOrThreadLocalScoped()
    .Use<ContextCentralSystem>()
    .Ctor<string>("path").Is("....."); // مقدار دهی سازنده با پارامتر رشته ای خاص
```

نویسنده: وحید نصیری
تاریخ: ۱۱:۷ ۱۳۹۲/۰۲/۱۸

ضمن اینکه روش دیگری نیز برای بازگشت دادن یک وهله، وجود دارد:

```
x.For<IUnitOfWork>().HybridHttpOrThreadLocalScoped().Use(() =>
{
    var ctx = new Sample07Context();
    ctx.Database.Connection.ConnectionString = "...";
    return ctx;
});
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۲ ۱۳:۴۰

[مقایسه‌ای بین توانمندی‌های کتابخانه‌های IoC Container مختلف در دات نت . ماخذ](#)

نویسنده: سیدعلی
تاریخ: ۱۳۹۲/۰۷/۰۱ ۱۱:۱۷

[این هم یک مقایسه دیگر که فکر می‌کنم جمع بندی آن به عهده توسعه دهندگان و انتخاب آنها بستگی دارد.](#)

نویسنده: سیروان عقیفی
تاریخ: ۱۳۹۲/۰۷/۰۱ ۲۲:۴۵

آیا جهت خودکار سازی تنظیمات اولیه باید ابتدا یک نمونه از اینترفیس هایمان را به StructureMap معرفی کنیم تا به صورت خودکار کار اسکن اسمبلی هایمان را انجام دهد؟
مثلاً در مثال شما جهت اسکن اسمبلی‌ها :

```
x.Scan(scan =>
{
    scan.AssemblyContainingType<IEmailsService>();
    scan.WithDefaultConventions();
});
```

به عنوان مثال شما IEmailService را تعریف کرده اید.

درست متوجه شدم؟

و سوال دیگر آیا امکان مشخص کردن Namespace جهت اسکن اسمبلی‌ها توسط StructureMap وجود دارد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۷/۰۱ ۲۳:۵۴

- IEmailsService (یا کلا هر نوع مشخصی) می‌تواند در یک اسمبلی دلخواه جداگانه باشد. ذکر آن کار اسکن را سریعتر و دقیق‌تر می‌کند. فقط یک نوع علامتگذاری است؛ این اسمبلی خاص رو بگرد، نه جای دیگری را.
- بله. برای نمونه باید IAssemblyScanner را پیاده سازی کنید. [اطلاعات بیشتر](#)
و یا یک مثال در اینجا [StructureMap - Don't Scan All Assemblies](#)

نویسنده: سیدعلی
تاریخ: ۱۳۹۲/۰۸/۲۸ ۱۹:۱۵

سلام

من در یک پروژه که دارای دو قسمت ویندوز سرویس و وب می‌باشد از تزریق وابستگی با StructureMap استفاده می‌کنم که دارای دو تعریف از UOW برای این دو هستم می‌خواستم بدونم به چه شکلی آن را تعریف کنم که در هر زمان که لازم بود یک نمونه از آن را با توجه به پروژه ای که هستم (ویندوز سرویس یا وب) داشته باشم و آیا اصولاً این کار امکان پذیر است؟

```
x.For<UnitOfWorkScopeBase<TModelUnitOfWork>>>()
    .HttpContextScoped()
    .Use<PerThreadUnitOfWorkScope<TModelUnitOfWork>>();
x.For<UnitOfWorkScopeBase<TModelUnitOfWork>>>()
    .HybridHttpOrThreadLocalScoped()
    .Use<PerThreadUnitOfWorkScope<TUnitOfWork>>();
```

با تشکر از شما

نویسنده: وحید نصیری

تاریخ: ۱۹:۲۵ ۱۳۹۲/۰۸/۲۸

- در متن بحث شده: « حالت Hybrid ترکیبی است از حالت‌های HttpContext و ThreadLocal. اگر برنامه وب بود، از HttpContext استفاده خواهد کرد در غیراینصورت به ThreadLocal سوئیچ می‌کند.»
- یعنی HybridHttpContextOrThreadLocalScoped هر دو مورد را به صورت خودکار پوشش می‌دهد.
- ضمناً روش تشخیص کلی زمینه برنامه جاری، اینکه وب است یا ویندوز به صورت زیر است:

```
using System.Web;

bool IsInWeb
{
    get
    {
        return HttpContext.Current != null;
    }
}
```

این مورد ارجاعی را به اسمبلی استاندارد System.Web نیاز دارد.

نویسنده: سیدعلی

تاریخ: ۱۹:۴۸ ۱۳۹۲/۰۸/۲۸

با تشکر از پاسخگویتان بله این مطالب را می‌دانستم شاید منظورم را اشتباه رساندم اینکه از کدام دو حالت UOW خودم استفاده کنم به کدام شکل تعریف می‌شود یعنی در زمانی که از HybridHttpContextOrThreadLocalScoped استفاده می‌کنم PerThreadUnitOfWorkScope یا PerRequestUnitOfWorkScope باشد؟ خودش این دو را با توجه به وب یا ویندوز سرویس بودن تشخیص دهد. با توجه به اینکه این دو Context را به عنوان ورودی دریافت می‌کنند. که اگر دریافت نمی‌کردن نحوه تعریف آنها شاید به این شکل صحیح بود.

```
x.For<UnitOfWorkScopeBase<TrackingModelUnitOfWork>>()
    .HybridHttpContextOrThreadLocalScoped()
    .Use(() =>
    {
        IsWeb ?new PerRequestUnitOfWorkScope<TrackingModelUnitOfWork>(context) : new
        PerThreadUnitOfWorkScope<TrackingModelUnitOfWork>(context)
    });
```

در بالای آن Context را تعریف کردم اما نمی‌دانم مورد بالا را به چه نحوی تعریف کنم.

```
x.For<LightSpeedContext<TUnitOfWork>>().Singleton().Use(() =>
{
    var ctx = new LightSpeedContext<TrackingModelUnitOfWork>()
    {
        ConnectionString = "MyConnection",
        QuoteIdentifiers = true,
        DataProvider = DataProvider.SqlServer2012
    };
    return ctx;
});
```

ممنون از راهنمایتان

نویسنده: وحید نصیری

تاریخ: ۲۰:۵۲ ۱۳۹۲/۰۸/۲۸

- نیازی به PerRequestUnitOfWorkScope و PerThreadUnitOfWorkScope کتابخانه‌های ثالث در حین کار با StructureMap نیست. خود این IoC Container قابلیت مدیریت طول عمر اشیاء را دارد. HttpContextScoped آن یعنی مدیریت طول عمر یک شیء و زنده نگه داشتن آن در طول یک درخواست یا Request. بنابراین نیازی نیست یکبار StructureMap این کار را انجام دهد و یکبار دیگر هم کتابخانه‌ی ثالث دیگری که مثلاً PerRequestUnitOfWorkScope در آن تعریف شده؛ کار اضافی است. (بحث «تعیین طول

عمر اشیاء در StructureMap» در متن فوق)

- فقط از HybridHttpOrThreadLocalScoped استفاده کنید تا هر دو حالت برنامه‌های وب و ویندوز را با یک تنظیم پوشش دهید.
- نیازی هم به بررسی IsInWeb یاد شده نیست. خود StructureMap به صورت توکار این کار را انجام می‌دهد.
- نیازی نیست تا کار وهله سازی را در قسمت Use انجام دهید (کار اضافی است). فقط نام کلاس آنرا ذکر کنید کافی است.

```
x.For<IMyInterface>().HybridHttpOrThreadLocalScoped().Use<MyClass>();
```

در این حالت کلاس MyClass، هر سازنده‌ای داشته باشد، با توجه به سایر x.For-Use های نوشته شده به صورت خودکار سازنده‌های آن‌ها توسط StructureMap وهله سازی می‌شوند. تا n سطح هم باشد، کار وهله سازی آن‌ها خودکار است به شرطی که در تنظیمات StructureMap ذکر کنید، هر تزریق اینترفیس صورت گرفته در سازنده کلاسی با چه کلاسی مرتبط است یعنی x.For-Use های کاملی باید داشته باشید.

نویسنده: رضا گرمارودی
تاریخ: ۱۳۹۲/۰۹/۲۵ ۸:۵۱

سلام؛ برای اعمال توکار دات نت چه کار باید کرد. مثلاً زمانی که CustomRole یا CustomMemberShip داریم و متد سازنده ما کانتکس و کلاس‌های دیتالایر را به عنوان پارامتر ورودی می‌گیره، این گونه موارد و نمی‌تونم تزریق وابستگی‌ها را انجام داد. من اشتباه می‌کنم یا راه دیگه ای داره؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۵ ۹:۳۱

در قسمت بررسی الگوی [Service locator](#) توضیح داده شده‌است. جایی که نمی‌توانید کار وهله سازی اشیاء را مستقیماً تحت کنترل قرار دهید، نیاز است از Service locator استفاده کنید. در حین کار با StructureMap اگر متد ObjectFactory.GetInstance مستقیماً داخل کدهای کلاس بکار گرفته شود، مفهوم Service locator را دارد.

نویسنده: ابوالفضل رجب پور
تاریخ: ۱۳۹۲/۰۹/۲۶ ۱۳:۳۹

من از structure در پروژه م به صورتی که توضیح دادم استفاده کردم. در یه مورد خاص null هست. وقتی نیاز به پارشال اکشن دارم که در کنترل دیگری قرار داره، درست کار میکنه سیم کشی‌ها و هیچ چیزی نال نیست.، اما وقتی نیاز دارم که پارشالی از اکشن کنترل جاری که در حال رندر هست، استفاده کنم، نال هست همه‌ی اینترفیس‌ها. سازنده کنترلر هم فراخونی نمیشه. ساختار کنترلر به این صورت هست:

```
public partial class ContactController : Controller
{
    private IGroupsBusiness _groupsBusiness;
    private IContactsBusiness _contactsBusiness;

    public ContactController(IContactsBusiness contactsBusiness, IGroupsBusiness groupsBusiness)
    {
        _groupsBusiness = groupsBusiness;
        _contactsBusiness = contactsBusiness;
    }

    public virtual ActionResult View(int id)
    {
        var model = _contactsBusiness.Select(id);
        return View(model);
    }

    public virtual ActionResult ViewGroups(int contactId)
    {
        var model = _groupsBusiness.SelectById(contactId);
    }
}
```

```
        return PartialView(model);  
    }  
}
```

ابتدا view اجرا میشه و سیم کشی برقرار هست. داخل ویو ارجاعی به اکشن viewgroups داره. اما این بار نال هست و به مشکل برمیخورم.

من توی ویو نوشتم

```
@{ Html.RenderAction(MVC.Contact.ViewGroups(Model.Id)); }
```

اگر این اکشن رو بذارم داخل کنترلر دیگه و صداش بزنم کار میکنه.
آیا نباید کد بالا درست کار بکنه؟

نویسنده: وحید نصیری
تاریخ: ۱۴:۳۷ ۱۳۹۲/۰۹/۲۶

جهت رعایت بهتر نظم در سایت:
- هر دوره در سایت، یک [قسمت مخصوص](#) پرسش و پاسخ‌های شخصی مرتبط با آن دوره دارد.
- دوره جاری یک [قسمت مجزای MVC](#) دارد.
- نحوه ارسال یک [گزارش خطای خوب](#) را هم یکبار مطالعه کنید. ارسال stack trace و اصل خطای حاصل خیلی مهم است و بدون آن پاسخ دادن از راه دور، بسیار مشکل.

نویسنده: ابوالفضل رجب پور
تاریخ: ۱۳:۴۰ ۱۳۹۲/۱۰/۱۴

تشکر
مشکل حل شد.
خطای منطقی بود. به اکشن به اسم View دارم در کنترلر. و بعد وقتی در اکشن‌های دیگه‌ی این کنترلر، return view رو صدا میزدم که اطلاعات نمایش داده بشه، این متد رو صدا میزد و بعد به حلقه بی پایان و در انتها خطای نامعلوم از طرف structuremap صادر میشد

نویسنده: vici
تاریخ: ۲۲:۱۰ ۱۳۹۲/۱۱/۰۶

سلام؛ وقتی برنامه اجرا میشه مسیر برنامه به این صورته؟
ابتدا به یه وهله از اینترفیس Email می‌سازه وبعد متد SendEmailToUser صدا زده میشه ، داخل این متد متوجه میشه که به وهله ایی از کلاس user نیاز داره پس به صورت خودکار وهله سازی رو انجام میده ، مقدار لازمه برگشت داده میشه و بقیه عملیات درسته؟

نویسنده: وحید نصیری
تاریخ: ۲۲:۱۷ ۱۳۹۲/۱۱/۰۶

بله. البته از اینترفیس وهله سازی نمی‌شود. بر اساس تنظیمات ObjectFactory.Initialize ، می‌داند که درخواست رسیده به IEmailService باید به کمک کلاس EmailsService وهله سازی شود و همینطور الی آخر.

پیشنیاز این بحث، مطلب «استفاده از StructureMap به عنوان یک IoC Container» می‌باشد که پیشتر در این سری مطالعه کردید (در حد نحوه نصب StructureMap و آشنایی با تنظیمات اولیه آن)

ابتدا ساختار بحث جاری را به نحو زیر در نظر بگیرید:

```
namespace DI04.Services
{
    public interface IAccounting
    {
        void CreateInvoice(int orderId, int count);
    }
}

namespace DI04.Services
{
    public interface ISales
    {
        bool ShippingAllowed(int orderId);
    }
}

namespace DI04.Services
{
    public interface IOrderHandler
    {
        void Handle(int orderId, int count);
    }
}

using System;

namespace DI04.Services
{
    public class Accounting : IAccounting
    {
        public Accounting()
        {
            Console.WriteLine("Accounting ctor.");
        }

        public void CreateInvoice(int orderId, int count)
        {
            // ...
        }
    }
}

using System;

namespace DI04.Services
{
    public class Sales : ISales
    {
        public Sales()
        {
            Console.WriteLine("Sales ctor.");
        }

        public bool ShippingAllowed(int orderId)
        {
            // فقط جهت آزمایش سیستم
            return false;
        }
    }
}

using System;

namespace DI04.Services
{

```

```
public class OrderHandler : IOrderHandler
{
    private readonly IAccounting _accounting;
    private readonly ISales _sales;
    public OrderHandler(IAccounting accounting, ISales sales)
    {
        Console.WriteLine("OrderHandler ctor.");
        _accounting = accounting;
        _sales = sales;
    }

    public void Handle(int orderId, int count)
    {
        if (_sales.ShippingAllowed(orderId))
        {
            _accounting.CreateInvoice(orderId, count);
        }
    }
}
```

در اینجا کار مدیریت سفارشات در کلاس OrderHandler انجام می‌شود. این کلاس دارای دو وابستگی تزریق شده در سازنده کلاس می‌باشد.

در متد Handle، اگر مجوز کار توسط متد ShippingAllowed صادر شد، آنگاه کار نهایی توسط متد CreateInvoice باید صورت گیرد. با توجه به اینکه تزریق وابستگی‌ها در سازنده کلاس صورت می‌گیرد، نیاز است پیش از وهله سازی کلاس OrderHandler، هر دو وابستگی آن وهله سازی و تزریق شوند. در حالیکه در مثال جاری هیچگاه به وهله‌ای از نوع IAccounting نیاز نخواهد شد؛ زیرا متد ShippingAllowed در این مثال، فقط بر false بر می‌گرداند.

و از این نمونه‌ها زیاد هستند. کلاس‌هایی با تعداد متدهای بالا و تعداد وابستگی‌های قابل توجه که ممکن است در طول عمر شیء وهله سازی شده این کلاس، تنها به یکی از این وابستگی‌ها نیاز شود و نه به تمام آن‌ها. راه حلی برای این مساله در دات نت 4 با معرفی کلاس Lazy ارائه شده است؛ به این نحو که اگر برای مثال در اینجا accounting را Lazy تعریف کنیم، تنها زمانی وهله سازی خواهد شد که به آن نیاز باشد و نه پیش از آن.

```
private readonly Lazy<IAccounting> _accounting;
```

سؤال: Lazy loading تزریق وابستگی‌ها را چگونه می‌توان توسط StructureMap فعال ساخت؟

ابتدا تعاریف کلاس OrderHandler را به نحو زیر بازنویسی می‌کنیم:

```
using System;

namespace DI04.Services
{
    public class OrderHandlerLazy : IOrderHandler
    {
        private readonly Lazy<IAccounting> _accounting;
        private readonly Lazy<ISales> _sales;
        public OrderHandlerLazy(Lazy<IAccounting> accounting, Lazy<ISales> sales)
        {
            Console.WriteLine("OrderHandlerLazy ctor.");
            _accounting = accounting;
            _sales = sales;
        }

        public void Handle(int orderId, int count)
        {
            if (_sales.Value.ShippingAllowed(orderId))
            {
                _accounting.Value.CreateInvoice(orderId, count);
            }
        }
    }
}
```

در اینجا سازنده‌های کلاس، به صورت Lazy معرفی شده‌اند. دسترسی به فیلدهای sales و accounting نیز اندکی تغییر کرده‌اند و اینبار از طریق خاصیت Value آن‌ها باید انجام شود. مرحله نهایی هم اندکی تغییر در نحوه معرفی تنظیمات اولیه StructureMap است:

```
using System;
using DI04.Services;
using StructureMap;

namespace DI04
{
    class Program
    {
        static void Main(string[] args)
        {
            // تنظیمات اولیه برنامه که فقط یکبار باید در طول عمر برنامه انجام شود
            ObjectFactory.Initialize(x =>
            {
                x.For<IOrderHandler>().Use<OrderHandlerLazy>();

                // Lazy loading
                x.For<Lazy<IAccounting>>().Use(c => new Lazy<IAccounting>(c.GetInstance<Accounting>));
                x.For<Lazy<ISales>>().Use(c => new Lazy<ISales>(c.GetInstance<Sales>));
            });

            var orderHandler = ObjectFactory.GetInstance<IOrderHandler>();
            orderHandler.Handle(orderId: 1, count: 10);
        }
    }
}
```

به این ترتیب زمانیکه برنامه به sales.Value می‌رسد آنگاه نیاز به وهله سازی شیء متناظر با آن‌را خواهد داشت که در اینجا از طریق متد GetInstance به آن ارسال خواهد گردید.

خروجی برنامه در این حالت:

```
OrderHandlerLazy ctor.
Sales ctor.
```

همانطور که مشاهده می‌کنید، هرچند کلاس OrderHandlerLazy دارای دو وابستگی تعریف شده در سازنده کلاس است، اما تنها وابستگی Sales آن زمانیکه به آن نیاز شده، وهله سازی گردیده است و خبری از وهله سازی کلاس Accounting نیست (چون مطابق تعاریف کلاس‌های برنامه هیچگاه به مسیر accounting.Value نخواهیم رسید؛ بنابراین نیازی هم به وهله سازی آن نخواهد بود).

دریافت مثال این قسمت

[DI04.zip](#)

نظرات خوانندگان

نویسنده: صابر فتح الهی
تاریخ: ۱۳:۳۴ ۱۳۹۲/۱۰/۱۷

سلام
یک سوال
من در یک برنامه MVC
چند کلاس دارم که در سازنده‌های آن کلاس‌های دیگر به صورت lazy تزریق میشود.
حال زمانی که کلاس مورد نظر فراخوانی می‌شود با خطای 202 به منزله عدم وجود سازنده پیش فرض مواجه می‌شوم در حالی
که تمامی کلاسها را به صورت lazy به StructureMap معرفی کرده‌ام.

نویسنده: وحید نصیری
تاریخ: ۱۴:۱۲ ۱۳۹۲/۱۰/۱۷

خطای 202 به معنای ناقص بودن سیم‌کشی‌های آغازین برنامه شما است. نمونه‌اش در بحث مرتبط با MVC [مطرح شده‌است](#) .

نویسنده: صابر فتح الهی
تاریخ: ۱۲:۲۸ ۱۳۹۲/۱۰/۱۸

سلام
سیم‌کشی‌های من درست بود
پارامترهارو در یک کلاس کپسوله کردم و به کنترلر پاس دادم درست شد. ظاهراً خطای غیر منطقی هست چون هیچ چیزی تغییر
نکرد

همانطور که در قسمت‌های قبل عنوان شد، دو نوع متداول تزریق وابستگی‌ها وجود دارند:

الف) تزریق وابستگی‌ها در سازنده کلاس

ب) تزریق وابستگی‌ها در خواص عمومی کلاس‌ها یا Setters injection

حالت الف متداول‌ترین است و بیشتر زمانی کاربرد دارد که کار وهله سازی یک کلاس را می‌توان راسا انجام داد. اما در فرم‌ها یا یوزرکنترل‌های ASP.NET Web forms به صورت پیش فرض کار وهله سازی فرم‌ها و یوزرکنترل‌ها توسط موتور ASP.NET انجام می‌شود و در این حالت اگر بخواهیم از تزریق وابستگی‌ها استفاده کنیم، مدام به همان روش معروف Service locator و استفاده از container.Resolve در تمام قسمت‌های برنامه می‌رسیم که آنچنان روش مطلوبی نیست.

اما ... در ASP.NET Web forms می‌توان وهله سازی فرم‌ها را نیز تحت کنترل قرار داد، که برای آن دو روش زیر وجود دارند:

الف) یک کلاس مشتق شده را از کلاس پایه [PageHandlerFactory](#) تهیه کنیم. این کلاس را پیاده سازی کرده و نهایتاً بجای وهله ساز پیش فرض فرم‌های موتور داخلی ASP.NET، در فایل وب کانفیگ برنامه استفاده کنیم. یک نمونه از پیاده سازی آن را [در اینجا](#) می‌توانید مشاهده کنید.

مشکلی که این روش دارد سازگاری آن با حالت Full trust است. یعنی برنامه شما در یک هاست Medium trust (اغلب هاست‌های خوب) اجرا نخواهد شد.

ب) روش دوم، استفاده از یک Http Module است برای اعمال Setter injection ها، به صورت خودکار. اکنون که حالت الف را همه جا نمی‌توان بکار برد یا به عبارتی نمی‌توان وهله سازی فرم‌ها را راسا در دست گرفت، حداقل می‌توان خواص عمومی اشیاء صفحه تولید شده را مقدار دهی کرد که در ادامه، این روش را بررسی می‌کنیم.

تهیه ماژول انجام Setters injection به صورت خودکار در برنامه‌های ASP.NET Web forms به کمک StructureMap

پیشنیاز این بحث، مطلب «استفاده از StructureMap به عنوان یک IoC Container» می‌باشد که پیشتر مطالعه کردید (در حد نحوه نصب StructureMap و آشنایی با تنظیمات اولیه آن)

```
using System.Collections;
using System.Web;
using System.Web.UI;
using StructureMap;

namespace DI05.Core
{
    /// <summary>
    /// تسهیل در کار تزریق خودکار وابستگی‌ها در سطح فرم‌ها و یوزرکنترل‌ها
    /// </summary>
    public class StructureMapModule : IHttpModule
    {
        public void Dispose()
        { }

        public void Init(HttpApplication app)
        {
            app.PreRequestHandlerExecute += (sender, e) =>
            {
                var page = HttpContext.Current.Handler as Page; // The Page handler
                if (page == null)
                    return;

                WireUpThePage(page);
                WireUpAllUserControls(page);
            };
        }

        private static void WireUpAllUserControls(Page page)
        {
            // در اینجا هم کار سیم کشی یوزر کنترل‌ها انجام می‌شود
            page.InitComplete += (initSender, evt) =>
            {
```

```

var thisPage = (Page)initSender;
foreach (Control ctrl in getControlTree(thisPage))
{
    // فقط یوزر کنترل‌ها بررسی شدند
    // بررسی شوند شرط را حذف کنید
    if (ctrl is UserControl)
    {
        ObjectFactory.BuildUp(ctrl);
    }
}
};
}

private static void WireUpThePage(Page page)
{
    ObjectFactory.BuildUp(page); // برقراری خودکار سیم‌کشی‌ها در سطح صفحات
}

private static IEnumerable getControlTree(Control root)
{
    foreach (Control child in root.Controls)
    {
        yield return child;
        foreach (Control ctrl in getControlTree(child))
        {
            yield return ctrl;
        }
    }
}
}
}
}

```

در این ماژول، کار با `HttpContext.Current.Handler` شروع می‌شود که دقیقاً معادل با وهله‌ای از یک صفحه یا فرم می‌باشد. اکنون که این وهله را داریم، فقط کافی است متد `ObjectFactory.BuildUp` مربوط به `StructureMap` را روی آن فراخوانی کنیم تا کار `Setter injection` را انجام دهد. مرحله بعد یافتن یوزر کنترل‌های احتمالی قرار گرفته بر روی صفحه و همچنین فراخوانی متد `ObjectFactory.BuildUp`، بر روی آن‌ها می‌باشد. پس از تهیه ماژول فوق، باید آن‌را در فایل وب کانفیگ برنامه معرفی کرد:

```

<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0" />

    <httpModules>
      <add name="StructureMapModule" type="DI05.Core.StructureMapModule"/>
    </httpModules>
  </system.web>

  <system.webServer>
    <modules runAllManagedModulesForAllRequests="true">
      <add name="StructureMapModule" type="DI05.Core.StructureMapModule"/>
    </modules>
    <validation validateIntegratedModeConfiguration="false" />
  </system.webServer>
</configuration>

```

مثالی از نحوه استفاده از `StructureMapModule` تهیه شده

فرض کنید لایه سرویس برنامه دارای اینترفیس‌ها و کلاس‌های زیر است:

```

namespace DI05.Services
{
    public interface IUsersService
    {
        string GetUserEmail(int id);
    }
}

namespace DI05.Services
{

```



```
public class UsersService: IUserService
{
    public string GetUserEmail(int id)
    {
        // فقط جهت بررسی تزریق وابستگی‌ها/
        return "test@test.com";
    }
}
```

کار تنظیمات اولیه آن‌ها را در فایل global.asax.cs برنامه انجام خواهیم داد:

```
using System;
using StructureMap;
using DI05.Services;

namespace DI05
{
    public class Global : System.Web.HttpApplication
    {
        private static void initStructureMap()
        {
            ObjectFactory.Initialize(x =>
            {
                x.For<IUserService>().Use<UsersService>();

                x.SetAllProperties(y =>
                {
                    y.OfType<IUserService>();
                });
            });
        }

        protected void Application_Start(object sender, EventArgs e)
        {
            initStructureMap();
        }

        void Application_EndRequest(object sender, EventArgs e)
        {
            ObjectFactory.ReleaseAndDisposeAllHttpScopedObjects();
        }
    }
}
```

در اینجا فقط باید دقت داشت که ذکر SetAllProperties الزامی است. از این جهت که از روش Setter injection در حال استفاده هستیم.

مرحله آخر هم استفاده از سرویس‌های برنامه به شکل زیر است:

```
using System;
using DI05.Services;

namespace DI05
{
    public partial class Default : System.Web.UI.Page
    {
        public IUserService UsersService { set; get; }

        protected void Page_Load(object sender, EventArgs e)
        {
            lblEmail1.Text = string.Format("From Default Page: {0}", UsersService.GetUserEmail(1));
        }
    }
}
```

همانطور که ملاحظه می‌کنید در این فرم، هیچ خبری از وجود IoC Container مورد استفاده نیست و کار وهله سازی و مقدار دهی سرویس مورد استفاده به صورت خودکار توسط Http Module تهیه شده انجام می‌شود.

دریافت مثال کامل قسمت جاری

[DI05.zip](#)

یک نکته‌ی تکمیلی

برای ارتقاء نکات مطلب جاری به نگارش سوم StructureMap نیاز است موارد ذیل را لحاظ کنید:
الف) نصب بسته‌ی وب آن

```
PM> Install-Package structuremap.web
```

ب) `HttpContextLifecycle.DisposeAndClearAll()` حذف شده را به متد جدید `ReleaseAndDisposeAllHttpScopedObjects` تغییر دهید.
ج) `x.SetAllProperties` را به `x.Policies.SetAllProperties` ویرایش کنید.

نظرات خوانندگان

نویسنده: فرهاد یزدان پناه
تاریخ: ۲۲:۴۱ ۱۳۹۲/۰۱/۲۷

وقت بخیر مهندس نصیری. من شخصا بیشتر کلاس‌های اصلی مربوط به ASP.NET رو سفارشی کردم (Page, UserControl, HttpHandler و ...) و در سازنده عملیات‌های مربوط به سیم کشی! رو انجام دادم. (قبلا در مباحث مربوط به Entity Framework این روش رو توضیح داده بودید) ولی به نظرم این روش HttpModule خیلی منظمتره. مسئله ای که به نظر می‌تونه کمک کنه اینه که کاش فقط Page ها رو سیم کشی نمی‌کردید (همه چیز حتی HttpHandler ها هم میشه همینجا کلکشون کنده بشه و همچنین WebControl ها) در هر حال دستتون درد نکنه.

نویسنده: فرهاد یزدان پناه
تاریخ: ۲۱:۲۷ ۱۳۹۲/۰۲/۰۸

وقت بخیر در حالاتی که کنترل‌هایی دارای کالکشن‌هایی از کنترل‌های دیگر باشند (مثل GridView که دارای ستون‌هایی است - هرچند که ستون کنترل نیست) این موارد سیم‌کشی نمی‌شوند چون جزو درخت Page نیستند.

نویسنده: وحید نصیری
تاریخ: ۲۱:۵۹ ۱۳۹۲/۰۲/۰۸

شما داخل کنترل‌های قرار گرفته داخل GridView نیاز به تزریق وابستگی‌ها دارید؟ مثلا یک ستون آن دارای سلول‌هایی از جنس یوزر کنترل است که داخل این نیاز هست تزریق صورت گیرد؟

نویسنده: فرهاد یزدان پناه
تاریخ: ۱۸:۵۹ ۱۳۹۲/۰۲/۱۰

تقریبا!

من یک کنترل آکاردیون دارم که دارای ساختاری شبه زیر است:

```
<Common:HomeAccordion runat="server">
  <Common:HomeAccordionPane runat="server" HeaderText="Pane #1">
    <Common:HomeAccordionItem runat="server" Text="Item #1"/>
    <Common:HomeAccordionItem runat="server" Text="Item #1"/>
  </Common:HomeAccordionPane>
  <Common:HomeAccordionPane runat="server" HeaderText="Pane #2">
    <Common:HomeAccordionItem runat="server" Text="Item #1"/>
    <Common:HomeAccordionItem runat="server" Text="Item #1"/>
  </Common:HomeAccordionPane>
</Common:HomeAccordion>
```

حال گاهی من از کنترل اصلی AccordionPane ارث برده و کنترل‌هایی را ایجاد می‌کنم که به صورت خودکار از سرویس‌های امنیتی استفاده کرده و آیتم‌های لازم (که کاربر جاری به آنها دسترسی دارد) را اضافه می‌کنم.

```
<Common:HomeAccordion runat="server">
  <DF:HomeAccordionPaneBasic runat="server" />
  <DF:HomeAccordionPaneSystem runat="server" />
  <DF:HomeAccordionPaneMyAccount runat="server" />
</Common:HomeAccordion>
```

گویا این اقلام (که معمولا می‌توانند کنترل هم نباشند - چون توسط والد رندر می‌شوند) جزو درخت صفحه نیستند - استفاده از ParseChildren در کنترل والد اجازه افزودن مجموعه کنترل‌هایی را می‌دهد)

نویسنده: وحید نصیری
تاریخ: ۱۹:۱۹ ۱۳۹۲/۰۲/۱۰

- نیاز به مثال کامل برای دیباگ هست.

- در کل اگر از روش مطرح شده در مطلب جاری جواب نگرفتید یا کمبود داشت، در سازنده کلاس، متد زیر را فراخوانی کنید:

```
ObjectFactory.BuildUp(this);
```

نویسنده: فرهاد یزدان پناه
تاریخ: ۱۹:۲۴ ۱۳۹۲/۰۲/۱۰

ممنون.

طبق چیزی که قبلا خودتون فرموده بودید (در دوره EF) من هم از همین روش استفاده می‌کنم. فقط کلاس‌های Page، UserControl، و چند کلاس دیگر را (که به عنوان کلاس‌های من قرار دارند) در سازنده انجام دادم. فقط برای HttpModule (چون همانند شی Application دارای یک نمونه است) در متد Init کارهای لازم رو انجام دادم.

نویسنده: فرهاد یزدان پناه
تاریخ: ۱۹:۲۴ ۱۳۹۲/۰۲/۱۰

سعی می‌کنم یک مثال که موارد لازم در اون دخیل باشه رو براتون ارسال کنم (در همین پست به عنوان پیوست)

نویسنده: vici
تاریخ: ۲۲:۲۸ ۱۳۹۲/۱۱/۰۷

سلام؛ تا قبل از این قسمت رو خوب متوجه شدم ولی توی این قسمت کلاس StructureMapModule اصلا متوجه نشدم چیه؟ یه مقدار راهنمایی می‌کنید؟ ممنون

نویسنده: وحید نصیری
تاریخ: ۲۲:۴۹ ۱۳۹۲/۱۱/۰۷

در مطلب «[بایدها و نیایدهای استفاده از IoC Containers](#)» عنوان شد که تا حد ممکن نباید کدهای مرتبط با یک IoC Container داخل کدهای متداول ما ظاهر شوند. کار کلاس StructureMapModule تمیز کردن فایل‌های code behind از وجود IoC Container مورد نظر است.

نویسنده: vici
تاریخ: ۲۳:۰۱ ۱۳۹۲/۱۱/۰۷

ممنون، یعنی این کلاس ثابت هست؟ در صورتی که نیاز به ویژگی یا تنظیم دیگری نداشته باشیم همین کلاس کفایت می‌کنه؟

نویسنده: وحید نصیری
تاریخ: ۲۳:۰۵ ۱۳۹۲/۱۱/۰۷

کار این کلاس، در مثالی که زده شد (انتهای مطلب) این است که UsersService درخواستی را به صورت خودکار و هله سازی و قابل استفاده می‌کند؛ بدون اینکه جایی اثری از StructureMap در این فایل code behind دیده شود.

نویسنده: فواد کریمی
تاریخ: ۱۸:۳۸ ۱۳۹۲/۱۲/۲۱

با سلام؛ من در ویندوز اپلیکیشن از این ساختار استفاده میکنم و از فرم‌های Devexpress استفاده میکنم. در کلاس BasePage روی دستور ObjectFactory this خطای زیر رو میده

```
An unhandled exception of type 'StructureMap.StructureMapException' occurred in StructureMap.dll
Additional information: Error in the application.
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۲۲ ۰:۳۹

این خطای کلی بدون مشخص بودن کدهای شما قابل بررسی نیست. [اطلاعات بیشتر](#)
معمولاً در VS.NET اگر بر روی جزئیات بیشتر استثنای رخ داده کلیک کنید، مقادیر تو در تو آن مانند inner exception حاصل، اطلاعات بیشتری را به همراه دارند.

نویسنده: ابوالفضل علیاری
تاریخ: ۱۳۹۳/۰۱/۰۶ ۱۲:۲۸

برای کار با ef و استفاده از UOW، لایه سرویس باید از روشی که در EF#12 آموزش دادید نوشته بشود؟
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۰۶ ۱۲:۵۶

مطلب و دوره جاری، پیشنهادی است برای درک جزئیات مطلبی که [به آن اشاره کردید](#).

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۱۳ ۰:۵۳

به روز سانی

روش ارتقاء به نگارش سوم StructureMap به انتهای بحث اضافه شد.

نویسنده: سیروان عقیفی
تاریخ: ۱۳۹۳/۰۲/۲۵ ۲۲:۳۲

ممنون، ظاهراً با MVC5 سازگار نیست، ربطش رو نمی‌دونم ولی با MVC5 تست کردم مشکل داشت (از مقدار بازگشتی توسط متد GetControllerInstance اشکال می‌گرفت)، با تعویض لایه وب به ورژن MVC4 مشکلم حل شد. مثالی تکمیلی مربوط به قسمت 12 سری EF شما هم برای ورژن MVC4 بود.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۲۶ ۱:۴۳

با MVC5 هم تستش کردم. مشکلی نبود. در GetControllerInstance فقط باید بررسی کنید که آیا controllerType نال هست یا خیر. اگر نال بود، یعنی یک آدرس یافت نشد در برنامه دارید:

```
if (controllerType == null && requestContext.HttpContext.Request.Url != null)
    throw new InvalidOperationException(string.Format("Page not found: {0}",
requestContext.HttpContext.Request.Url.AbsoluteUri.ToString(CultureInfo.InvariantCulture)));
```

هدف از این قسمت، ارائه راه حلی برای حالت تزریق وابستگی‌ها در سازنده‌های کنترلرهای ASP.NET MVC به صورت خودکار است. به صورت پیش فرض، ASP.NET MVC به کنترلرهایی نیاز دارد که سازنده آن‌ها فاقد پارامتر باشند. از این جهت که بتواند به صورت خودکار آن‌ها را و هله سازی کرده و مورد استفاده قرار دهد. بنابراین به نظر می‌رسد که در اینجا نیز به همان روش معروف استفاده از الگوی Service locator و تکرار مدام کدهایی مانند `ObjectFactory.GetInstance` در سراسر برنامه خواهیم رسید که آنچنان مطلوب نیست.

اما ... در ASP.NET MVC می‌توان و هله ساز پیش فرض کنترلرها را با پیاده سازی کلاس `DefaultControllerFactory` به طور کامل تعویض کرد. یعنی اگر در اینجا بجای و هله ساز پیش فرض، از و هله سازی انجام شده توسط `IoC Container` خود بتوانیم استفاده کنیم، آنگاه کار تزریق وابستگی‌ها در سازنده‌های کنترلرها نیز خودکار خواهد گردید.

```
public class StructureMapControllerFactory : DefaultControllerFactory
{
    protected override IController GetControllerInstance(RequestContext requestContext, Type
controllerType)
    {
        if (controllerType == null)
            throw new InvalidOperationException(string.Format("Page not found: {0}",
requestContext.HttpContext.Request.Url.AbsoluteUri.ToString(CultureInfo.InvariantCulture)));
        return ObjectFactory.GetInstance(controllerType) as Controller;
    }
}
```

در کدهای فوق نمونه‌ای از این پیاده سازی را با استفاده از امکانات `StructureMap` ملاحظه می‌کنید. به این ترتیب در زمان و هله سازی خودکار یک کنترلر، اینبار `StructureMap` وارد عمل شده و وابستگی‌های برنامه را مطابق تعاریف `ObjectFactory.Initialize` ذکر شده، به سازنده کلاس کنترلر تزریق می‌کند. برای استفاده از این `ControllerFactory` جدید تنها کافی است بنویسیم:

```
protected void Application_Start()
{
    //Set current Controller factory as StructureMapControllerFactory
    ControllerBuilder.Current.SetControllerFactory(new StructureMapControllerFactory());
}
```

و ... همین!

اکنون نوشتن یک چنین کنترلرهایی که سازنده آن‌ها دارای پارامتر است، مجاز خواهد بود و تزریق وابستگی‌ها در سازنده‌ها به صورت خودکار توسط `IoC Container` مورد استفاده انجام می‌شود.

```
public partial class LoginController : Controller
{
    readonly IUsersService _userService;
    public LoginController(IUsersService userService)
    {
        _userService = userService;
    }
}
```

بدیهی است سایر مسایل مانند تنظیمات اولیه `IoC Container`، تهیه لایه سرویس و غیره مانند قبل است و تفاوتی نمی‌کند.

روش دوم تزریق خودکار وابستگی‌ها در برنامه‌های ASP.NET MVC

روش پیاده سازی و تعویض `DefaultControllerFactory` پیش فرض، متداول‌ترین روش خودکار سازی تزریق وابستگی‌ها در ASP.NET MVC است. روش دیگری نیز بر اساس پیاده سازی اینترفیس توکار `IDependencyResolver` معرفی شده در ASP.NET MVC 3.0 به بعد، وجود دارد. این روش علاوه بر ASP.NET MVC در کنترلرهای مخصوص Web API نیز کاربرد دارد. حتی `SignalR` نیز دارای

کلاس پایه‌ای به نام `DefaultDependencyResolver` با امضای مشابه `IDependencyResolver` است.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using StructureMap;

namespace Prog
{
    public class StructureMapDependencyResolver : IDependencyResolver
    {
        public object GetService(Type serviceType)
        {
            if (serviceType.IsAbstract || serviceType.IsInterface || !serviceType.IsClass)
                return ObjectFactory.TryGetInstance(serviceType);
            return ObjectFactory.GetInstance(serviceType);
        }

        public IEnumerable<object> GetServices(Type serviceType)
        {
            return ObjectFactory.GetAllInstances(serviceType).Cast<object>();
        }
    }
}
```

یک نمونه از پیاده سازی آن را به کمک `StructureMap` در اینجا ملاحظه می‌کنید. برای ثبت آن در برنامه خواهیم داشت:

```
protected void Application_Start()
{
    DependencyResolver.SetResolver(new StructureMapDependencyResolver());
}
```

در Web API باید `GlobalConfiguration.Configuration.DependencyResolver` تنظیم شود. البته `IDependencyResolver` آن در فضای نام دیگری به نام `System.Web.Http.Dependencies` قرار گرفته است؛ اما کلیات آن تفاوتی نمی‌کند.

نظرات خوانندگان

نویسنده:

مولانا

تاریخ:

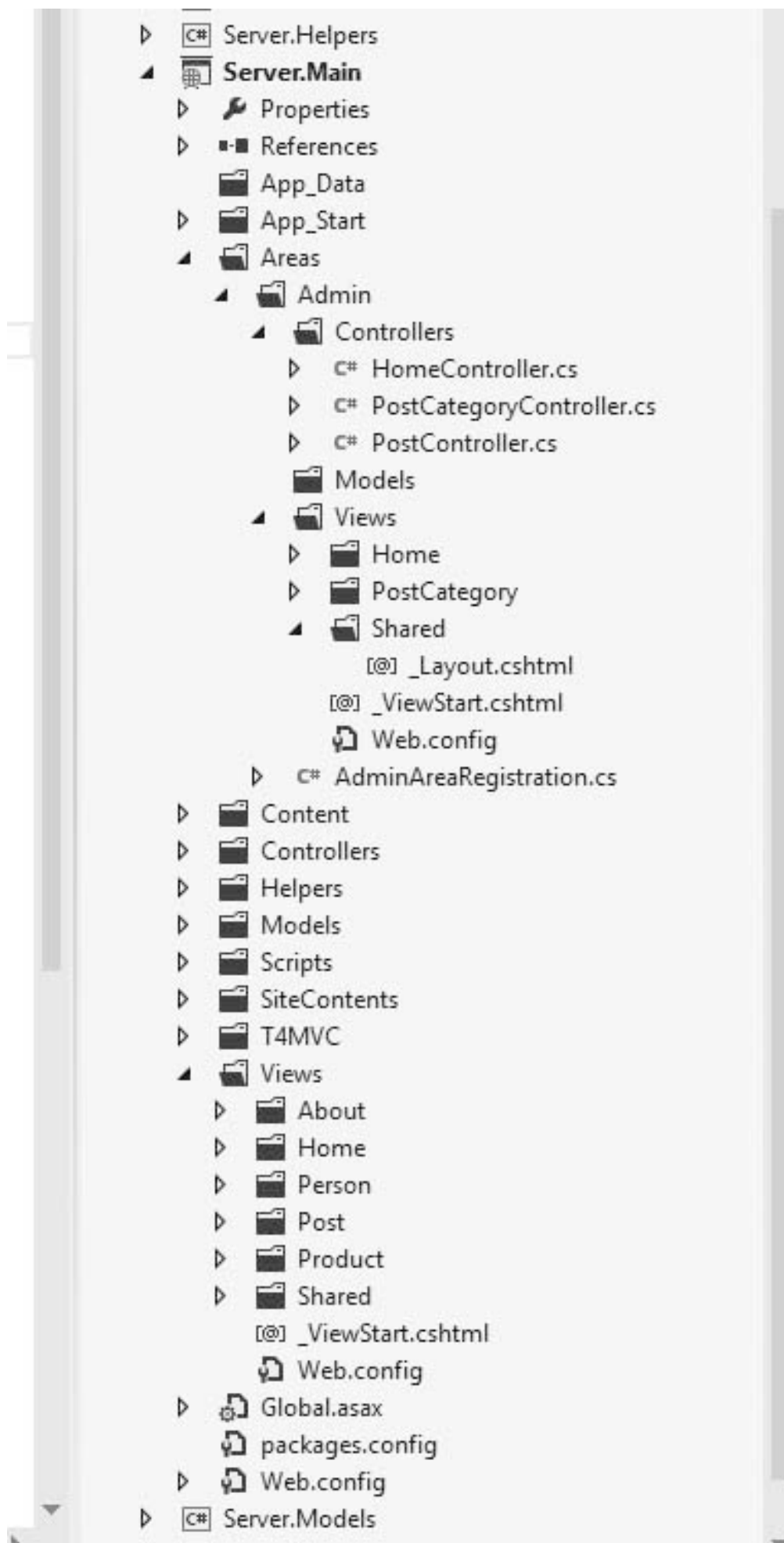
۱۰:۱۳ ۱۳۹۲/۰۲/۰۸

باسلام.

هنگامی که آدرس <http://localhost:2215/admin> را در مرورگر وارد میکنم با خطای زیر روبرو می‌شوم.

```
The IControllerFactory 'Server.Helpers.StructureMapControllerFactory' did not return a controller for the name 'admin'
```

ساختار پروژه من هم بصورت زیر است:



کد global.asax هم بصورت زیر است:

```
public class MvcApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        Database.SetInitializer<EshoppingContext>(null);

        #region Mvc

        AreaRegistration.RegisterAllAreas();
        WebApiConfig.Register(GlobalConfiguration.Configuration);
        FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
        RouteConfig.RegisterRoutes(RouteTable.Routes);
        BundleConfig.RegisterBundles(BundleTable.Bundles);
        MvcHandler.DisableMvcResponseHeader = true;

        #endregion

        initStructureMap();
    }
    protected void Application_EndRequest(object sender, EventArgs e)
    {
        ObjectFactory.ReleaseAndDisposeAllHttpScopedObjects();
    }
    static void initStructureMap()
    {
        ObjectFactory.Initialize(x =>
        {
            x.For<IUnitOfWork>().HttpContextScoped().Use(() => new EshoppingContext());
            x.For<IProductCategoryService>().Use<ProductCategoryService>();
            x.For<IProductService>().Use<ProductService>();
            x.For<IPersonService>().Use<PersonService>();
            x.For<IPersonAttachmentService>().Use<PersonAttachmentService>();
            x.For<IPostService>().Use<PostService>();
            x.For<IPostCategoryService>().Use<PostCategoryService>();
            x.For<ISliderItemService>().Use<SliderItemService>();
        });
        ControllerBuilder.Current.SetControllerFactory(new StructureMapControllerFactory());
    }
}
```

اطلاعات RouteConfig هم بصورت زیر است:

```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new
            {
                controller = MVC.Home.Name,
                action = MVC.Home.ActionNames.Index,
                id = UrlParameter.Optional
            },
            namespaces: new[] { "Server.Main.Controllers" }
        );
    }
}
```

و کد StructureMapControllerFactory بصورت زیر است:

```
public class StructureMapControllerFactory : DefaultControllerFactory
{
    protected override IController GetControllerInstance(RequestContext requestContext, Type
controllerType)
    {
        if (controllerType == null)
        {
            return null;
        }
        return ObjectFactory.GetInstance(controllerType) as Controller;
    }
}
```

با تشکر.

نویسنده:

وحید نصیری

تاریخ:

۱۱:۴۰ ۱۳۹۲/۰۲/۰۸

- اگر مطلب را مطالعه کرده باشید، نحوه صحیح استفاده از Controller factory به این صورت است:

```
if (controllerType == null)
    throw new InvalidOperationException(string.Format("Page not found: {0}",
requestContext.HttpContext.Request.Url.AbsoluteUri.ToString(CultureInfo.InvariantCulture)));
```

به عمد این استثناء صادر می‌شود تا مشخص شود اگر controllerType نال است، حتما مسیری بوده که در سیستم وجود ندارد و این مورد برای خطایابی در دراز مدت حائز اهمیت است. به صورت خودکار هم توسط ELMAH لاگ می‌شود و برای بررسی‌های بعدی مورد استفاده خواهد بود.

- اگر اینکار رو انجام بدید خواهید دید که مسیر ذکر شده توسط شما به صورت یافت نشد اعلام می‌گردد. چرا؟

```
public override void RegisterArea(AreaRegistrationContext context)
{
    context.MapRoute(
        "Admin_default",
        "Admin/{controller}/{action}/{id}",
        new { action = "Index", id = UrlParameter.Optional }
    );
}
```

چون در مسیریابی پیش فرض یک Area نام کنترلر پیش فرض ذکر نشده (در فایل AdminAreaRegistration.cs). نام action هست اما نام controller نیست. به همین جهت یا باید این مسیریابی رو اصلاح کنید و یا اینکه مسیر کامل را به نحو زیر وارد نمائید:

```
http://localhost:2215/admin/home
```





















نویسنده:

پویا امینی

تاریخ:

۱:۱۲ ۱۳۹۲/۰۴/۲۴

با سلام؛ من در استفاده از این روش در پروژه خودم به مشکل بر خوردم ممنون میشم راهنمایی بفرمایید. ساختار لایه بندی من به صورت زیر است.

- ▶  Configuration
- ▶  DataLayer
- ▶  DomainClasses
- ▶  ServiceLayer
- ▶  **Web**
 - ▶  Properties
 - ▶  References
 - ▶  App_Data
 - ▶  Content
 - ▶  Controllers
 - ▶  HomeController.cs
 - ▶  Images
 - ▶  Models
 - ▶  Scripts
 - ▶  Views
 - ▶  Class1.cs
 - ▶  Global.asax
 - ▶  Global.asax.cs
 - ▶  packages.config
 - ▶  Web.config

در پروژه وب یک کلاس ایجاد کردم و کدهای زیر را در آن نوشتم

```
public class StructureMapControllerFactory : DefaultControllerFactory
{
    protected override IController GetControllerInstance(RequestContext requestContext, Type
controllerType)
    {
        if (controllerType == null)
            throw new InvalidOperationException(string.Format("Page not found: {0}",
requestContext.HttpContext.Request.Url.AbsoluteUri.ToString(CultureInfo.InvariantCulture)));
        return ObjectFactory.GetInstance(controllerType) as Controller;
    }
}
```

و کدهای کنترلر من به صورت زیر است

```
public class HomeController : Controller
{
    public ITABMPCREWSERVICE aa { get; set; }

    public ActionResult Index()
    {
        TABMPCREWS tt = new TABMPCREWS()
        {
            DTLASTUPDATEDDATE = DateTime.Now,
            INTOTRATE = 122,
            INTRATE = 215,
            VCCODEDESCRIPTION = "fff858699",
            VCCODEVALUE = "fff858699",
            VCLASTUSERID = "fff858699",
            INTCREWCODE = 105652
        };
        aa.Add(tt);

        ViewBag.Message = "Welcome to ASP.NET MVC!";
        return View();
    }

    public ActionResult About()
    {
        ViewBag.Message = "Your app description page.";
        return View();
    }

    public ActionResult Contact()
    {
        ViewBag.Message = "Your contact page.";
        return View();
    }
}
```

و در فایل Global در متد Application_Start() کد زیر را اضافه کردم

```
ControllerBuilder.Current.SetControllerFactory(new StructureMapControllerFactory());
```

و در لایه Service کلاس TABMPCREWSERVICE به صورت زیر است

```
public class TABMPCREWSERVICE : ITABMPCREWSERVICE
{
    private IUnitOfWork _uow;
    private IDbSet<TABMPCREWS> _tabmpcrews;
    public TABMPCREWSERVICE(IUnitOfWork uow)
    {
        this._uow = uow;
    }
}
```

```

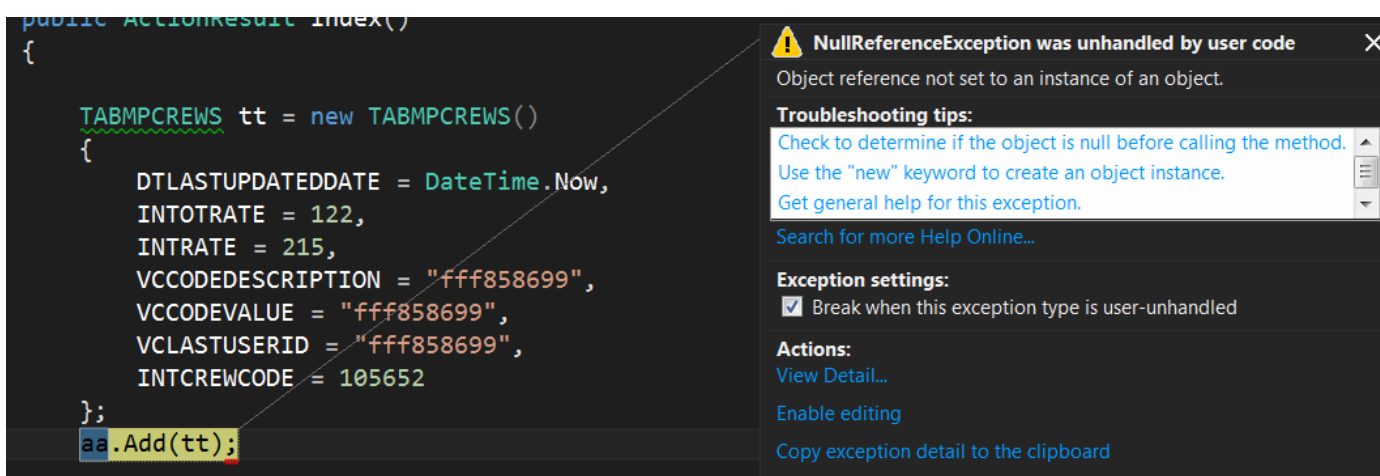
        _tabmpcrews = uow.Set<TABMPCREW>();
    }

    public int Add(TABMPCREW personnel)
    {
        int rowEffect = 0;

        _tabmpcrews.Add(personnel);
        rowEffect = _uow.SaveChanges();
        return rowEffect;
    }
}

```

و زمانیکه پروژه را اجرا می‌کنم به این خطا بر می‌خورم



همه چیز رو چک کردم ولی دلیلی برای این خطا پیدا نکردم . ممنون میشم راهنمایی کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۲۴ ۸:۴۲

دو نوع تزریق وابستگی‌ها وجود دارد: الف) در سازنده کلاس ب) در خواص تعریف شده شما روش دوم را انتخاب کردید. نیازی به اینکار در MVC نیست و روش مرجع، روش الف است که نمونه‌ای از آن را در کلاس LoginController بحث فوق ملاحظه می‌کنید. اگر می‌خواهید تزریق وابستگی‌ها در خواص یک کلاس صورت گیرد، نیاز به یک سری تنظیمات اضافه‌تر وجود دارد که [در بحث وب فرم‌ها](#) مطرح شده (تنظیم SetAllProperties در متد initStructureMap آن).

نویسنده: پویا امینی
تاریخ: ۱۳۹۲/۰۴/۲۵ ۰:۱۸

من می‌خواهم از روش اول استفاده کنم برای همین کنترلر خودم را به صورت زیر تغییر دادم

```

public class HomeController : Controller
{
    public readonly ITABMPCREWService aa ;
    public HomeController(ITABMPCREWService tabmpcrewService)
    {
        aa = tabmpcrewService;
    }

    public ActionResult Index()
    {
        TABMPCREWS tt = new TABMPCREWS()

```

```

    {
        DTLASTUPDATEDDATE = DateTime.Now,
        INTOTRATE = 122,
        INTRATE = 215,
        VCCODEDESCRIPTION = "fff858699",
        VCCODEVALUE = "fff858699",
        VCLASTUSERID = "fff858699",
        INTCREWCODE = 105652
    };
    aa.Add(tt);

    ViewBag.Message = "Welcome to ASP.NET MVC!";
    return View();
}

public ActionResult About()
{
    ViewBag.Message = "Your app description page.";

    return View();
}

public ActionResult Contact()
{
    ViewBag.Message = "Your contact page.";

    return View();
}
}
}

```

ولی خطای زیر را دریافت می‌کنم

.StructureMap Exception Code: 202 No Default Instance defined for PluginFamily ServiceLayer

ممنون میشم من را راهنمایی کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۲۵ ۰:۳۵

این خطا یعنی تنظیمات اولیه ناقصی دارید. مراجعه کنید به مطلب «[استفاده از StructureMap به عنوان یک IoC Container](#)» برای توضیحات بیشتر در مورد نحوه تعریف ObjectFactory.Initialize و ارتباط دادن اینترفیس‌ها به کلاس‌های متناظر.

نویسنده: پویا امینی
تاریخ: ۱۳۹۲/۰۴/۲۵ ۱:۲۶

من مطلب بالا را مطالعه کردم و با توجه به مطلب بالا کدهای خودم را به صورت زیر تغییر دادم

لایه Service

```

namespace ServiceLayer.EFServices
{
    public class TABMPCREWSERVICE : ITABMPCREWSERVICE
    {
        private IUnitOfWork _uow;
        private IDbSet<TABMPCREWS> _tabmpcrews;

        public TABMPCREWSERVICE(IUnitOfWork uow)
        {
            this._uow = uow;
            _tabmpcrews = uow.Set<TABMPCREWS>();
        }

        public int Add(TABMPCREWS personnel)
        {
            int rowEffect = 0;

```

```

        _tabmpcrews.Add(personnel);
        rowEffect = _uow.SaveChanges();
        return rowEffect;
    }
}

```

و اینترفیس

```

namespace ServiceLayer.Interface
{
    public interface ITABMPCREWSERVICE
    {
        int Add(TABMPCREWS personnel);
    }
}

```

و فایل Global

```

protected void Application_Start()
{
    AreaRegistration.RegisterAllAreas();

    RegisterGlobalFilters(GlobalFilters.Filters);
    RegisterRoutes(RouteTable.Routes);
    ObjectFactory.Initialize(x =>
    {
        x.For<ITABMPCREWSERVICE>().Use<TABMPCREWSERVICE>();
        // x.For<IUsersService>().Use<UsersService>();
    });
    ControllerBuilder.Current.SetControllerFactory(new StructureMapControllerFactory());
    // initStructureMap();
}

```

9

```

public class StructureMapControllerFactory : DefaultControllerFactory
{
    protected override IController GetControllerInstance(RequestContext requestContext, Type controllerType)
    {
        if (controllerType == null)
            throw new InvalidOperationException(string.Format("Page not found: {0}",
requestContext.HttpContext.Request.Url.AbsoluteUri.ToString(CultureInfo.InvariantCulture)));
        return ObjectFactory.GetInstance(controllerType) as Controller;
    }
}

```

و کد کنترلر

```

public class HomeController : Controller
{
    public readonly ITABMPCREWSERVICE aa ;
    public HomeController(ITABMPCREWSERVICE tabmpcrewService)
    {
        aa = tabmpcrewService;
    }

    public ActionResult Index()
    {
        TABMPCREWS tt = new TABMPCREWS()
        {
            DTLASTUPDATEDDATE = DateTime.Now,
            INTOTRATE = 122,
            INTRATE = 215,
            VCCODEDESCRIPTION = "fff858699",
            VCCODEVALUE = "fff858699",
            VCLASTUSERID = "fff858699",

```



```

        INTCREWCODE = 105652
    };
    aa.Add(tt);

    ViewBag.Message = "Welcome to ASP.NET MVC!";
    return View();
}

public ActionResult About()
{
    ViewBag.Message = "Your app description page.";
    return View();
}

public ActionResult Contact()
{
    ViewBag.Message = "Your contact page.";
    return View();
}
}

```

اما زمان اجرا این خطا رو بهم میده و از این خط خطا می‌گیره

Server Error in '/' Application.

StructureMap Exception Code: 202

No Default Instance defined for PluginFamily ServiceLayer.Interface.ITABMPCREWService, CmmsServiceLayer, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: StructureMap StructureMapException: StructureMap Exception Code: 202
No Default Instance defined for PluginFamily ServiceLayer.Interface.ITABMPCREWService, CmmsServiceLayer, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null

Source Error:

```

Line 17:         if (controllerType == null)
Line 18:             throw new InvalidOperationException(string.Format("Page not found: {0}", requestContext.HttpContext.Request.Url.AbsoluteUri.ToString(CultureInfo.))
Line 19:             return ObjectFactory.GetInstance(controllerType) as Controller;
Line 20:         }
Line 21:     }

```

Source File: c:\Akpc\Cmms\CmmsWeb\Class1.cs **Line:** 19

Stack Trace:

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۲۵ ۷:۵۸

عرض کردم تعاریف ObjectFactory.Initialize و ارتباط دادن اینترفیس‌ها به کلاس‌های متناظر شما ناقص است. الان TABMPCREWService خودش دارای یک وابستگی تزریق شده در سازنده آن به نام IUnitOfWork است که تعاریف مرتبط با آن در قسمت ObjectFactory.Initialize ذکر نشدند. یعنی این IoC Container نمی‌دونه برای وهله سازی کلاس TABMPCREWService زمانیکه به IUnitOfWork رسید از چه کلاسی باید استفاده کند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۱۵ ۱۳:۳۹

یک نکته تکمیلی

اگر به هر دلیلی یک کلاس پایه کنترلر را ایجاد کردید که در آن ExecuteCore تعریف شده است، این متد چه با تزریق وابستگی‌ها

و چه بدون آن فراخوانی نخواهد شد. (روش صحیح مدیریت این مسایل در ASP.NET MVC استفاده از فیلترها است و نه ارث بری؛ چون در طراحی ASP.NET MVC مباحث AOP به صورت خودکار توسط فیلترها پیاده سازی می‌شوند)

```
public class MyBaseController : Controller
{
    /// <summary>
    /// from
    http://forums.asp.net/t/1776480.aspx/1?ExecuteCore+in+base+class+not+fired+in+MVC+4+beta
    /// </summary>
    protected override bool DisableAsyncSupport
    {
        get { return true; }
    }

    protected override void ExecuteCore()
    {
        base.ExecuteCore();
    }
}
```

برای حل این مشکل باید DisableAsyncSupport را اضافه کنید تا ExecuteCore تحریف شده، اجرا گردد (جزو تغییرات MVC4 است).

نویسنده: ابوالفضل علیاری
تاریخ: ۱۳۹۲/۱۱/۲۲ ۱۹:۱۱

سلام؛ کلاس StructureMapControllerFactory در کجای برنامه باید پیاده سازی شود؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۲ ۱۹:۱۵

- در یک کلاس یا یک کتابخانه کمکی.
- محل فراخوانی و معرفی آن مهم است که در بحث هم ذکر شده: Application_Start

نویسنده: سیروان عقیفی
تاریخ: ۱۳۹۳/۰۲/۱۹ ۲۱:۲

سلام؛ من در پروژه ام (یک برنامه ASP.NET MVC) یک کنترلر Web Api ایجاد کردم؛ در این حالت تنظیم DependencyResolver به چه صورت است؟ یعنی همزمان هم باید به این دو صورت تنظیم شود؟

```
protected void Application_Start()
{
    DependencyResolver.SetResolver(new StructureMapDependencyResolver());
    GlobalConfiguration.Configuration.DependencyResolver = new
    StructureMapDependencyResolver(container);
}
```

در این حالت پیاده سازی StructureMapDependencyResolver به چه صورت خواهد بود؟
ممنون.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۱۹ ۲۳:۵

مانند همان توضیحات انتهای بحث است. فقط در متد BeginScope باید this بازگشت داده شود:

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Web.Http.Dependencies;
using StructureMap;

public class StructureMapDependencyResolver : IDependencyResolver
{
    public IDependencyScope BeginScope()
    {
        return this;
    }

    public object GetService(Type serviceType)
    {
        if (serviceType.IsAbstract || serviceType.IsInterface || !serviceType.IsClass)
            return ObjectFactory.Container.TryGetInstance(serviceType);
        return ObjectFactory.GetInstance(serviceType);
    }

    public IEnumerable<object> GetServices(Type serviceType)
    {
        return ObjectFactory.GetAllInstances(serviceType).Cast<object>();
    }

    public void Dispose()
    {
    }
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۲۰ ۱:۴۶

یک نکته‌ی تکمیلی در مورد Web API

Mark Seemann توصیه کرده‌است که از IDependencyResolver استفاده نکنید. روش دیگری برای کار با Web API مورد تأیید ایشان است:

[Dependency Injection and Lifetime Management with ASP.NET Web API](#)

[Dependency Injection in ASP.NET Web API with Castle Windsor](#)

پایده سازی این نکته با StructureMap در اینجا:

[Better way to configure StructureMap in ASP.NET WebAPI](#)

نویسنده: سیروان عفیفی
تاریخ: ۱۳۹۳/۰۲/۲۰ ۸:۲۰

ممنون! مراحل رو به این صورت انجام دادم :

```
static void InitStructureMap()
{
    ObjectFactory.Initialize(x =>
    {
        x.For<IUnitOfWork>().HybridHttpOrThreadLocalScoped().Use<MyContext>();
        x.Scan(scan =>
        {
            scan.AssemblyContainingType<INewsService>();
            scan.WithDefaultConventions();
        });
    });

    ControllerBuilder.Current.SetControllerFactory(new StructureMapControllerFactory());
    var container = ObjectFactory.Container;
    GlobalConfiguration.Configuration.Services
        .Replace(typeof(IHttpControllerActivator),
            new StructureMapHttpControllerActivator(container));
}
```

پایده سازی StructureMapHttpControllerActivator به همان صورت که در لینک معرفی کردید انجام دادم. ممنون از شما.

برای رسیدن به الگوی معکوس سازی وابستگی‌ها عموماً مراحل زیر طی می‌شوند:

- الف) در متدهای لایه جاری خود واژه‌های کلیدی new و همچنین کلیه فراخوانی‌های استاتیک را بیابید.
- ب) وهله سازی این‌ها را به یک سطح بالاتر (نقطه آغازین برنامه) منتقل کنید. اینکار باید بر اساس اتکای به Abstraction و برای مثال استفاده از اینترفیس‌ها صورت گیرد.
- ج) اینکار را آنقدر تکرار کنید تا دیگر در کدهای لایه جاری خود واژه کلیدی new یا فراخوانی متدهای استاتیک مشاهده نشود.
- د) در آخر وهله سازی object graph مورد نیاز را به یک IoC Container محول کنید.

یک مثال: ابتدا بررسی یک قطعه کد متداول

```
using System.Net;
using System.Text;
using System.Text.RegularExpressions;
using System.Web.Mvc;

namespace DI06.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            string result = string.Empty;
            using (var client = new WebClient { Encoding = Encoding.UTF8 })
            {
                result = client.DownloadString("http://www.dotnettips.info/");
            }
            var match = new Regex(@"(?s)<title>(.*?)</title>", RegexOptions.IgnoreCase).Match(result);
            var title = match.Groups[1].Value.Trim();

            ViewBag.PageTitle = title;
            return View();
        }
    }
}
```

فرض کنید یک برنامه ASP.NET MVC را به نحو فوق تهیه کرده‌ایم. در کدهای کنترلر آن قصد داریم محتویات HTML یک صفحه خاص را دریافت و سپس عنوان آن را استخراج کرده و نمایش دهیم.

مشکلات کد فوق:

- الف) قرار گرفتن منطق تجاری پیاده سازی کدها مستقیماً داخل کدهای یک اکشن متد؛ این مساله در دراز مدت به تکرار شدید کدها منجر خواهد شد که نهایتاً قابلیت نگهداری آن را کاهش می‌دهند.
- ب) در این کد حداقل دو بار واژه کلیدی new ذکر شده است. مورد اول یا new WebClient، از همه مهم‌تر است؛ از این جهت که نوشتن آزمون واحد را برای این کنترلر بسیار مشکل می‌کند. آزمون‌های واحد باید سریع و بدون نیاز به منابع خارجی، قابل اجرا باشند. تعویض آن هم مطابق کدهای تدارک دیده شده کار ساده‌ای نیست.

بهبود کیفیت قطعه کد متداول فوق با استفاده از الگوی معکوس سازی وابستگی‌ها

در اصل معکوس سازی وابستگی‌ها عنوان کردیم لایه بالایی سیستم نباید مستقیماً به لایه‌های زیرین در حال استفاده از آن، وابسته باشد. این وابستگی باید معکوس شده و همچنین بر اساس Abstraction یا برای مثال استفاده از اینترفیس‌ها صورت گیرد. به همین منظور یک پروژه دیگر را از نوع Class library، مثلاً به نام DI06.Services به Solution جاری اضافه می‌کنیم.

```
namespace DI06.Services
{
```

```
public interface IWebClientServices
{
    string FetchUrl(string url);
    string GetWebpageTitle(string url);
}

using System.Net;
using System.Text;
using System.Text.RegularExpressions;

namespace DI06.Services
{
    public class WebClientServices : IWebClientServices
    {
        public string FetchUrl(string url)
        {
            using (var client = new WebClient { Encoding = Encoding.UTF8 })
            {
                return client.DownloadString(url);
            }
        }

        public string GetWebpageTitle(string url)
        {
            var html = FetchUrl(url);
            var match = new Regex(@"(?s)<title>(.*?)</title>", RegexOptions.IgnoreCase).Match(html);
            return match.Groups[1].Value.Trim();
        }
    }
}
```

در این لایه، سرویس WebClient ایی را تدارک دیده‌ایم. این سرویس می‌تواند محتوای HTML یک آدرس را برگرداند و یا عنوان آن آدرس خاص را استخراج نماید.

هنوز کار معکوس سازی وابستگی‌ها رخ نداده است. صرفاً اندکی تمیزکاری و انتقال پیاده سازی منطق تجاری به یک سری کلاس‌هایی با قابلیت استفاده مجدد صورت گرفته است. به این ترتیب اگر باگی در این کدها وجود داشته باشد و همچنین از آن در چندین نقطه برنامه استفاده شده باشد، اصلاح این کلاس مرکزی، به یکباره تمامی قسمت‌های مختلف برنامه را تحت تاثیر مثبت قرار داده و از تکرار کدها و فراموشی احتمالی بهبود قسمت‌های مشابه جلوگیری می‌کند. کار معکوس سازی وابستگی‌ها در یک لایه بالاتر صورت خواهد گرفت:

```
using System.Web.Mvc;
using DI06.Services;

namespace DI06.Controllers
{
    public class HomeController : Controller
    {
        readonly IWebClientServices _webClientServices;
        public HomeController(IWebClientServices webClientServices)
        {
            _webClientServices = webClientServices;
        }

        public ActionResult Index()
        {
            ViewBag.PageTitle = _webClientServices.GetWebpageTitle("http://www.dotnettips.info/");
            return View();
        }
    }
}
```

اینبار کنترلر Home را توسط تزریق وابستگی‌ها در سازنده کنترلر، بازنویسی کرده‌ایم. کد نهایی بسیار تمیزتر از حالت قبل است. دیگر پیاده سازی متد GetWebpageTitle مستقیماً داخل یک اکشن متد قرار نگرفته است. همچنین این کنترلر اصلاً نمی‌داند که قرار است از کدام پیاده سازی اینترفیس IWebClientServices استفاده کند. اگر در تنظیمات اولیه IoC Container مورد استفاده، کلاس WebClientServices ذکر شده باشد، از آن استفاده خواهد کرد؛ یا اگر حتی کلاس Fake WebClientServices نیز معرفی گردد (جهت انجام آزمون غیر وابسته به new WebClient)، باز هم بدون کوچکترین تغییری در کدهای آن قابل استفاده خواهد بود.

در مورد نحوه تنظیمات اولیه یک IoC Container و یا پیشنیازهای ASP.NET MVC جهت آماده شدن برای تزریق خودکار وابستگی‌ها در سازنده کنترلرها، پیشتر مطالبی را در این سری مطالعه کرده‌اید؛ در اینجا نیز اصول مورد استفاده یکی است و تفاوتی نمی‌کند.

نظرات خوانندگان

نویسنده: اکبر بابامرادی
تاریخ: ۱۶:۲۶ ۱۳۹۲/۰۴/۱۸

لطفا مثالی جهت درک مفاهیم معکوس سازی، در کنترلرهایی که با دیتابیس ارتباط دارند و یک مدل به View ارسال می‌کنند، بزنید.

نویسنده: وحید نصیری
تاریخ: ۱۶:۲۹ ۱۳۹۲/۰۴/۱۸

- اصول آن تفاوتی نمی‌کند. مراحل تشکیل لایه سرویس، برپایی IOC و همچنین تزریق وابستگی‌ها در سازنده کنترلرها یکی است.
+ مراجعه کنید به [قسمت 12 سری EF Code first](#). یک مثال در این مورد (کار با دیتابیس در MVC، وب فرم‌ها و برنامه‌های ویندوزی به همراه تزریق وابستگی‌ها) وجود دارد و همچنین قابل دریافت است.

در این قسمت قصد داریم همانند کنترلرهای در ASP.NET MVC، کار تزریق وابستگی‌ها را در متدهای سازنده ViewModelهای WPF بدون استفاده از الگوی Service locator انجام دهیم؛ برای مثال:

```
public class TestViewModel
{
    private readonly ITestService _testService;
    public TestViewModel(ITestService testService) // تزریق وابستگی در سازنده کلاس
    {
        _testService = testService;
    }
}
```

و همچنین کار اتصال یک ViewModel، به View متناظر آن را نیز خودکار کنیم. قراردادی را نیز در اینجا بکار خواهیم گرفت: نام تمام Viewهای برنامه به View ختم می‌شوند و نام ViewModelها به ViewModel. برای مثال **Test View** و **Test ViewModel** معرف یک View و ViewModel متناظر خواهند بود.

ساختار کلاس‌های لایه سرویس برنامه

```
namespace DI07.Services
{
    public interface ITestService
    {
        string Test();
    }
}

namespace DI07.Services
{
    public class TestService: ITestService
    {
        public string Test()
        {
            return "برای آزمایش";
        }
    }
}
```

یک پروژه WPF را آغاز کرده و سپس یک پروژه Class library دیگر را به نام Services با دو کلاس و اینترفیس فوق، به آن اضافه کنید. هدف از این کلاس‌ها صرفاً آشنایی با نحوه تزریق وابستگی‌ها در سازنده یک کلاس ViewModel در WPF است.

علامتگذاری ViewModelها

در ادامه یک اینترفیس خالی را به نام IViewModel مشاهده می‌کنید:

```
namespace DI07.Core
{
    public interface IViewModel // از این اینترفیس خالی برای یافتن و علامتگذاری ویوو مدل‌ها استفاده می‌کنیم
    {
    }
}
```

از این اینترفیس برای علامتگذاری ViewModelهای برنامه استفاده خواهد شد. این روش، یکی از انواع روش‌هایی است که در مباحث Reflection برای یافتن کلاس‌هایی از نوع مشخص استفاده می‌شود. برای نمونه کلاس TestViewModel برنامه، با پیاده سازی IViewModel، به نوعی نشانه گذاری نیز شده است:


```
using DI07.Services;
using DI07.Core;

namespace DI07.ViewModels
{
    public class TestViewModel : IViewModel // علامتگذاری ویوو مدل
    {
        private readonly ITestService _testService;
        public TestViewModel(ITestService testService) // تزریق وابستگی در سازنده کلاس
        {
            _testService = testService;
        }

        public string Data
        {
            get { return _testService.Test(); }
        }
    }
}
```

تنظیمات آغازین IoC Container مورد استفاده

در کلاس استاندارد App برنامه WPF خود، کار تنظیمات اولیه StructureMap را انجام خواهیم داد:

```
using System.Windows;
using DI07.Core;
using DI07.Services;
using StructureMap;

namespace DI07
{
    public partial class App
    {
        protected override void OnStartup(StartupEventArgs e)
        {
            base.OnStartup(e);

            ObjectFactory.Configure(cfg =>
            {
                cfg.For<ITestService>().Use<TestService>();

                cfg.Scan(scan =>
                {
                    scan.TheCallingAssembly();
                    // Add all types that implement IView into the container,
                    // and name each specific type by the short type name.
                    scan.AddAllTypesOf<IViewModel>().NameBy(type => type.Name);
                    scan.WithDefaultConventions();
                });
            });
        }
    }
}
```

در اینجا عنوان شده است که اگر نیاز به نوع ITestService وجود داشت، کلاس TestService را و هله سازی کن. همچنین در ادامه از قابلیت اسکن این IoC Container برای یافتن کلاس‌هایی که IViewModel را در اسمبلی جاری پیاده سازی کرده‌اند، استفاده شده است. متد NameBy، سبب می‌شود که بتوان به این نوع‌های یافت شده از طریق نام کلاس‌های متناظر دسترسی یافت.

اتصال خودکار ViewModel‌ها به View‌های برنامه

```
using System.Windows.Controls;
using StructureMap;
```

```

namespace DI07.Core
{
    /// <summary>
    /// Stitches together a view and its view-model
    /// </summary>
    public static class ViewModelFactory
    {
        public static void WireUp(this ContentControl control)
        {
            var viewName = control.GetType().Name;
            var viewModelName = string.Concat(viewName, "Model"); // قرار داد نامگذاری ما است
            control.Loaded += (s, e) =>
            {
                control.DataContext = ObjectFactory.GetNamedInstance<IViewModel>(viewModelName);
            };
        }
    }
}

```

اکنون که کار علامتگذاری `IViewModel`ها انجام شده و همچنین `IoC Container` ما می‌داند که چگونه باید آن‌ها را در اسمبلی جاری جستجو کند، مرحله بعدی، ایجاد کلاسی است که از این تنظیمات استفاده می‌کند. در کلاس `ViewModelFactory`، متد `WireUp`، وهله‌ای از یک `View` را دریافت کرده، نام آن را استخراج می‌کند و سپس بر اساس قراردادی که در ابتدای بحث وضع کردیم، نام `ViewModel` متناظر را یافته و سپس زمانیکه این `View` بارگذاری می‌شود، به صورت خودکار `DataContext` آن را به کمک `StructureMap` وهله سازی می‌کند. این وهله سازی به همراه تزریق خودکار وابستگی‌ها در سازنده کلاس `ViewModel` نیز خواهد بود.

استفاده از کلاس `ViewModelFactory`

در ادامه کدهای `TestView` و پنجره اصلی برنامه را مشاهده می‌کنید:

```

<UserControl x:Class="DI07.Views.TestView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="300">
    <Grid>
        <TextBlock Text="{Binding Data}" />
    </Grid>
</UserControl>

<Window x:Class="DI07.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:Views="clr-namespace:DI07.Views"
    Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Views:TestView />
    </Grid>
</Window>

```

در فایل `Code behind` مرتبط با `TestView` تنها کافی است سطر فراخوانی `this.WireUp` اضافه شود تا کار تزریق وابستگی‌ها، وهله سازی `ViewModel` متناظر و همچنین مقدار دهی `DataContext` آن به صورت خودکار انجام شود:

```

using DI07.Core;

namespace DI07.Views
{
    public partial class TestView
    {
        public TestView()
        {
            InitializeComponent();
            this.WireUp(); // تزریق خودکار وابستگی‌ها و یافتن ویوو مدل متناظر
        }
    }
}

```

```
    }  
  }  
}
```

دریافت پروژه کامل این قسمت

[DI07.zip](#)

نظرات خوانندگان

نویسنده: شاهین کیاست
تاریخ: ۱۷:۵۵ ۱۳۹۲/۰۲/۰۵

مدیریت طول عمر DbContext به کمک StructureMap در برنامه‌های WPF و الگوی MVVM چگونه است ؟

نویسنده: وحید نصیری
تاریخ: ۱۹:۵۲ ۱۳۹۲/۰۲/۰۵

- هر دوره قسمت اختصاصی رو داره به نام « [پرسش و پاسخ](#) » برای طرح این نوع سؤالات خارج از موضوع مطلب جاری، اما مرتبط با عنوان دوره.

- در این مورد DbContext در همان پرسش و پاسخ‌های قسمت 12 سری EF بحث شده. [اینجا](#) برای تکرار:

«... در یک برنامه مبتنی بر MVVM، مدیریت طول عمر یک context در طول عمر ViewModel برنامه است. در یک برنامه ویندوزی تا زمانیکه یک فرم باز است، اشیاء آن تخریب نخواهند شد. بنابراین مدیریت context در برنامه‌های ویندوزی «دستی» است. در زمان شروع فرم context شروع خواهد شد، زمان تخریب/بستن آن، با بستن یا dispose یک context، خودبخود اتصالات هم قطع خواهند شد.

بنابراین در برنامه‌های وب «context/session per http request» داریم؛ در برنامه‌های ویندوزی «context per operation or per form». یعنی می‌تونید بسته به معماری برنامه ویندوزی خود، context را در سطح یک فرم تعریف کنید و مدیریت؛ و یا در سطح یک عملیات کوتاه مانند یک کلیک ...»

نویسنده: وحید نصیری
تاریخ: ۱۱:۳۹ ۱۳۹۲/۰۲/۲۷

یک نکته جالب!

```
public class FrameFactory : Frame
{
    protected override void OnContentChanged(object oldContent, object newContent)
    {
        base.OnContentChanged(oldContent, newContent);
        ((FrameworkElement)newContent).WireUp(); // مرتبط سازی و و هله سازی ویوو مدل مرتبط
    }
}
```

کار تزریق وابستگی‌ها و و هله سازی ویوو مدل مرتبط انجام خواهد شد

می‌شود یک کنترل فریم سفارشی ایجاد کرد. سپس در متد OnContentChanged فرصت تزریق خودکار وابستگی‌ها به صفحه‌ای که در حال اضافه شدن و نمایش است وجود خواهد داشت.

نویسنده: وحید نصیری
تاریخ: ۱۳:۱۳ ۱۳۹۲/۰۵/۲۲

چند مثال تکمیلی دیگر

[Restructuring your legacy code using MVVM and Dependency Injection with Unity](#)

[IOC Containers and MVVM](#)

[Using Structuremap to resolve ViewModels](#)

[Implementing MVVM Light with Structure Map](#)

عنوان: تزریق وابستگی‌ها و سناریوهای بسیار متعدد موجود

نویسنده: وحید نصیری

تاریخ: ۱۰:۷ ۱۳۹۲/۰۷/۲۲

آدرس: www.dotnettips.info

برچسب‌ها: Design patterns, Dependency Injection, IoC

تعدادی از پرکاربردترین حالت‌های تزریق وابستگی‌ها را در دوره جاری بررسی کردیم. برای مثال چگونه می‌توان تزریق وابستگی‌های یک کنترلر ASP.NET MVC را [خودکار کرد](#)، یا در [وب فرم‌ها](#) وضعیت چگونه است. اما در حین حل مسایلی از این دست، به سناریوهای بسیار متعددی برخورد خواهید خورد. برای مثال تزریق وابستگی‌های خودکار در یک کنترلر را آموختیم؛ در مورد فیلترها و Action Result‌های سفارشی چطور باید رفتار کرد؟ در WCF چطور؟ در هندلرهای وب فرم‌ها چطور؟ و بسیاری از حالات دیگر. البته تمام این موارد را توسط الگوی [Service locator](#) که شامل استفاده مستقیم از امکانات و هله سازی یک IoC Container، در کلاس مدنظر است، می‌توان حل کرد؛ اما باید تا حد امکان از این روش با توجه به اینکه خود IoC Container را تبدیل به یک وابستگی مدفون شده در کلاس‌های ما می‌کند، پرهیز نمود.

اگر به دنبال کتابخانه‌ای هستید که بسیاری از این سناریوها را پیاده سازی کرده است، کتابخانه Autofac پیشنهاد می‌شود. حتی اگر علاقمند به استفاده از آن نباشید، می‌توان از نحوه پیاده سازی‌های مختلف آن در مورد حالت‌های مختلف خودکار سازی تزریق وابستگی‌ها، ایده گرفت و سپس این کدها را با IoC Container مورد علاقه خود پیاده سازی کرد.

صفحه خانگی Autofac

<http://code.google.com/p/autofac>

<http://autofac.org>

بسته نیوگت

<http://www.nuget.org/packages/Autofac>

[محلی برای ایده گرفتن](#) مثلاً در مورد فیلترهای ASP.NET MVC

و در مورد نحوه استفاده از آن‌ها، نیاز است [آزمون‌های واحد این پروژه](#) را بررسی کنید و یا [مستندات پروژه](#) را مطالعه کنید. همچنین بررسی [لیست مستندات کلی آن](#) نیز بسیار مفید است

به صورت خلاصه، هرجایی در مورد تزریق وابستگی‌های خودکار جهت پرهیز از استفاده مستقیم از الگوی Service locator ایده‌ای نداشتید، سورس پروژه Autofac را بررسی کنید.

پ.ن.

[سایت bitbucket](#) امکان import کامل مخازن کد Google code را نیز دارد (در صورتیکه دسترسی شما به گوگل کد محدود است).

عموما در معکوس سازی مسئولیت‌ها و واگذاری آن‌ها به لایه‌های دیگر، از اینترفیس‌ها استفاده می‌شود. روش دیگری را که در اینجا می‌توان بکار گرفته استفاده از Delegates است بجای اینترفیس‌ها. از این جهت که یک Delegate در عمل می‌تواند به صورت یک [Anonymous Interface](#) عمل کند. در بسیاری از مواردی که اینترفیس شما تنها از یک متد تشکیل می‌شود، می‌توان عملکرد آن‌را با یک Delegate جهت ساده سازی فرآیند تزریق وابستگی‌ها تعویض کرد.

یک مثال از تعویض اینترفیس‌های تک متدی با Delegates

```
public interface IAuthentication
{
    bool IsUserAuthenticated(string userName, string password);
}

public class AuthenticationService : IAuthentication
{
    public bool IsUserAuthenticated(string userName, string password)
    {
        return userName == "Vahid" && password == "123";
    }
}

public class LoginController
{
    private readonly IAuthentication _authentication;
    public LoginController(IAuthentication authentication)
    {
        _authentication = authentication;
    }

    // ...
}
```

مثال بسیار متداول فوق را در نظر بگیرید. در لایه سرویس برنامه، اینترفیس و کلاس پیاده سازی کننده منطق اعتبارسنجی کاربران را تدارک دیده‌ایم. نهایتاً جایی در سطحی بالاتر از این توانمندی قرار است استفاده شود. مثلاً در کلاس LoginController. نکته مهم اینترفیس IAuthentication، تک متدی بودن آن است. به همین جهت تعریف کلاس LoginController را به شکل زیر نیز می‌توان بازنویسی کرد:

```
public class LoginController
{
    private readonly Func<string, string, bool> _authenticationStrategy;
    public LoginController(Func<string, string, bool> authenticationStrategy)
    {
        _authenticationStrategy = authenticationStrategy;
    }

    // ...
}
```

در این حالت نیز کلاس LoginController استفاده کننده از Delegate تعریف شده، از نحوه و استراتژی اعتبارسنجی بی‌خبر است و پیاده سازی آن به کلاسی دیگر که قرار است از LoginController استفاده کند، واگذار می‌شود.

اگر در برنامه‌ی وب خود از ELMAH استفاده می‌کنید و استثنایی در روال رخدادگردان Application_Start واقع در فایل global.asax.cs رخ دهد، این استثناء در لاگ‌های ELMAH ظاهر نمی‌شود و صرفاً در لاگ‌های خود ویندوز قابل مطالعه خواهد بود؛ از این جهت که HttpContext.Current در این روال هنوز ایجاد نشده‌است. همچنین اگر از IoC Containers استفاده می‌کنید یا آغاز بانک اطلاعاتی ORM خود را در روال Application_Start قرار داده‌اید، تمام این‌ها نیز آغاز نشده باقی خواهند ماند و عملاً تا ری استارت بعدی برنامه یا IIS در سرور به صورت دستی، برنامه و سایت قابل استفاده نخواهد بود.

سؤال: آیا می‌توان از ری‌استارت دستی IIS، به ری‌استارت خودکار رسید؟

پاسخ: بلی. فراخوانی متد HttpRuntime.UnloadAppDomain در یک برنامه‌ی ASP.NET سبب ری استارت آن خواهد شد. بنابراین می‌توان این متد را در Application_Start برنامه، جهت ری‌استارت خودکار آن در صورت شکست اولیه Application_Start قرار داد:

```
void Application_Start(object sender, EventArgs e)
{
    try
    {
        // startup stuff
    }
    catch
    {
        HttpRuntime.UnloadAppDomain(); // آغاز مجدد آن با درخواست بعدی می‌شود
        throw;
    }
}
```

اهمیت این مساله از آنجا ناشی می‌شود که ممکن است در آن لحظه‌ی خاص، بار سرور زیاد بوده باشد یا حتی میزان حافظه مهبیای جهت آغاز برنامه کافی نبوده باشد؛ و نه الزاماً مشکل منطقی در کدهای برنامه. فراخوانی HttpRuntime.UnloadAppDomain سبب خواهد شد تا برنامه شانس مجدد اجرای دیگری را بدون نیازی به ری استارت دستی IIS بیابد.

نظرات خوانندگان

نویسنده: میثم هوشمند
تاریخ: ۱۳۹۲/۱۰/۲۴ ۰:۳۳

به هیچ وجه نمی‌توان خطاهای رخ داده در این بخش را catch کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۲۴ ۱:۰۶

چرا، در همین try/catch نوشته شده می‌شود اینکار را انجام داد؛ اما فایده‌ای ندارد چون قسمت آغازین برنامه ناقص است و بعد از آن برنامه قابل استفاده نخواهد بود. مثلاً تنظیمات نگاشت‌های IoC Container و یا ORM انجام نشده‌اند. بنابراین catch آن حاصلی نخواهد داشت و عملاً برنامه نیاز به ری‌استارت دستی پیدا می‌کند. چون به ظاهر پروسه IIS آن در حال اجرا است، اما قسمت‌های مختلف برنامه پاسخ نمی‌دهند.

یک برنامه‌ی WinForms را در نظر بگیرید که از دو فرم تشکیل شده است.

فرم اول کار نمایش فرم 2 را به عهده دارد.

فرم دوم کار ارسال ایمیل را انجام می‌دهد. این ایمیل نیز از طریق سرویس ذیل فراهم می‌شود:

```
namespace WinFormsIoc.Services
{
    public interface IEmailsService
    {
        void SendEmail(string from, string to, string title, string message);
    }
}

namespace WinFormsIoc.Services
{
    public class EmailsService : IEmailsService
    {
        public void SendEmail(string from, string to, string title, string message)
        {
            //todo: ...
        }
    }
}
```

پیاده سازی متد SendEmail در اینجا مدنظر نیست. نکته‌ی مهم، مدیریت تامین و تزریق وابستگی‌های تعریف شده در سازنده‌ی آن است:

```
public partial class Form2 : Form
{
    private readonly IEmailsService _emailsService;
    public Form2(IEmailsService emailsService)
    {
        _emailsService = emailsService;
        InitializeComponent();
    }
}
```

احتمالا شاید عنوان کنید که در فرم اول، زمانیکه نیاز است فرم دوم نمایش داده شود، می‌نویسیم `new Form2` و در پارامتر آن با استفاده از متد `ObjectFactory.GetInstance` سازنده‌ی آن را فراهم می‌کنیم:

```
var form2 = new Form2(ObjectFactory.GetInstance<IEmailsService>());
form2.Show();
```

و یا اگر مدتی با IoC Containers کار کرده باشید، شاید پیشنهاد دهید که فقط بنویسید:

```
var form2 = ObjectFactory.GetInstance<Form2>();
form2.Show();
```

و همین! به صورت خودکار اگر `n` پارامتر تزریق شده هم در سازنده‌ی فرم دوم وجود داشته باشند، بر اساس تنظیمات اولیه‌ی IoC Container مورد استفاده، نمونه سازی شده و برنامه بدون مشکل کار خواهد کرد.

مشکل! این دو راه حل هیچکدام به عنوان تزریق وابستگی‌ها شناخته نمی‌شوند و به [الگوی Service locator](#) معروف هستند. مشکل آن‌ها این است که کدهای ما در حال حاضر وابستگی مستقیمی به IoC container مورد استفاده پیدا کرده‌اند. در حالت اول ما خودمان دستی درخواست داده‌ایم که کدام وابستگی باید و هله سازی شود و در حالت دوم همانند حالت اول، کدهای `ObjectFactory.GetInstance`، مختص به یک IoC Container خاص است. نحوه‌ی صحیح کار با IoC Container ها باید به این نحو

باشد که یکبار در آغاز برنامه تنظیم شوند و در ادامه سایر کلاس‌های برنامه طوری کار کنند که انگار IoC Container ایی وجود خارجی ندارد.

راه حل: ObjectFactory.GetInstance را کپسوله کنید.

```
using System.Windows.Forms;

namespace WinFormsIoc.IoC
{
    public interface IFormFactory
    {
        T Create<T>() where T : Form;
    }
}

using System.Windows.Forms;
using StructureMap;

namespace WinFormsIoc.IoC
{
    public class FormFactory : IFormFactory
    {
        public T Create<T>() where T : Form
        {
            return ObjectFactory.GetInstance<T>();
        }
    }
}
```

در اینجا یک اینترفیس را تعریف کرده‌ایم که متد ایجاد وهله‌ای از یک فرم را ارائه می‌دهد. پیاده سازی آن در برنامه‌ای که از StructureMap استفاده می‌کند، مطابق کلاس FormFactory است. اگر IoC Container دیگری باشد، فقط باید این پیاده سازی را تغییر دهید و نه کل برنامه را. اکنون برای استفاده از آن، IFormFactory را در سازنده‌ی کلاسی که نیاز دارد فرم‌های دیگر را نمایش دهد، تزریق می‌کنیم:

```
using System;
using System.Windows.Forms;
using WinFormsIoc.IoC;

namespace WinFormsIoc
{
    public partial class Form1 : Form
    {
        private readonly IFormFactory _formFactory;
        public Form1(IFormFactory formFactory)
        {
            _formFactory = formFactory;
            InitializeComponent();
        }

        private void btnShowForm2_Click(object sender, EventArgs e)
        {
            var form2 = _formFactory.Create<Form2>();
            form2.Show();
        }
    }
}
```

در کدهای فوق، فرم اول برنامه را ملاحظه می‌کنید که قرار است فرم دوم را نمایش دهد. IFormFactory در سازنده‌ی آن تزریق شده‌است. با فراخوانی متد Create آن، فرم دوم برنامه به همراه تمام وابستگی‌های تزریق شده‌ی در سازنده‌ی آن وهله سازی می‌شوند.

نکته‌ی مهم این کدها عدم وابستگی مستقیم آن به هیچ نوع IoC Container خاصی است. این فرم اصلاً نمی‌داند که IoC Container ایی در برنامه وجود دارد یا خیر.

مشکل! با تغییر سازنده‌ی Form1 برنامه دیگر کامپایل نمی‌شود!

اگر فایل Program.cs را باز کنید، یک چنین سطری را دارد:

```
Application.Run(new Form1());
```

چون سازنده‌ی فرم یک، اکنون پارامتر جدیدی پیدا کرده‌است، در اینجا می‌توان ObjectFactory.GetInstance را مستقیماً بکار برد (در این حالت خاص که مرتبط است به کلاس آغازین برنامه، با توجه به اینکه وهله سازی آن مستقیماً و خارج از کنترل ما انجام می‌شود، دیگر چاره‌ای نداریم و مجبور هستیم از الگوی Service locator استفاده کنیم).

```
Application.Run(ObjectFactory.GetInstance<Form1>());
```

مثال کامل این بحث را از اینجا می‌توانید دریافت کنید

[WinFormsIoc.zip](#)

اگر با برنامه‌های وب و StructureMap کار کرده باشید، حتما از متد جدید `HttpContextLifecycle.DisposeAndClearAll` و متد قدیمی `ObjectFactory.ReleaseAndDisposeAllHttpScopedObjects` آن نیز برای `Dispose` خودکار کلیه اشیاء `IDisposable` در `Application_EndRequest` استفاده کرده‌اید. البته شرط استفاده از متدهای یاد شده نیز این است که طول عمر اشیاء `IDisposable` به صورت `Http Scoped` تعریف شده باشند:

```
x.For<IUnitOfWork>().HybridHttpOrThreadLocalScoped().Use<MyContext>();
```

سؤال: برای سایر حالات چطور؟ در یک برنامه‌ی ویندوزی که `Http Scoped` در آن معنا ندارد چکار باید کرد؟

پاسخ: در اینجا حداقل دو راه حل وجود دارد:

الف) استفاده از nested containers

```
using (var container = ObjectFactory.Container.GetNestedContainer())
{
    var uow = container.GetInstance<IUnitOfWork>();
}
```

قابلیتی از نگارش 2.6 استراکچرمپ به آن اضافه شده‌است به نام [nested containers](#) که هدف از آن `Dispose` خودکار کلیه اشیاء `Transient` از نوع `IDisposable` است. در اینجا منظور از `Transient` این است که طول عمر شیء مدنظر به صورت `Singleton`، `HttpContext scoped` و یا `ThreadLocal scoped` تعریف نشده باشد (هیچ نوع `caching` خاصی به طول عمر آن اعمال نشده باشد). در مثال فوق، پس از پایان کار قطعه‌ی `using` نوشته شده، به صورت خودکار کلیه اشیاء `IDisposable` یافت شده و `Dispose` می‌شوند.

ب) نگاهی به پشت صحنه‌ی متد `DisposeAndClearAll`

اگر اشیاء `IDisposable` شما با طول عمر `HybridHttpOrThreadLocalScoped` معرفی شده باشند (و `Transient` نباشند)، [با دستور ذیل](#) چه در برنامه‌های ویندوزی و چه در برنامه‌های وب، کلیه‌ی آن‌ها یافت شده و به صورت خودکار `Dispose` می‌شوند:

```
new HybridLifecycle().FindCache(null).DisposeAndClear();
```

متد `HttpContextLifecycle.DisposeAndClearAll` فقط مختص است به برنامه‌های وب. اگر نیاز به متدی دارید که در هر دو حالت برنامه‌های وب و ویندوزی کار کند، از روش `HybridLifecycle` فوق استفاده نمایید.

بنابراین به صورت خلاصه

اگر طول عمر شیء `IDisposable` مدنظر به صورت هیبرید تعریف شده‌است، از متد `DisposeAndClear` موجود در `HybridLifecycle` می‌توان استفاده کرد. اگر طول عمر شیء `IDisposable` مورد استفاده، معمولی است و هیچ نوع `caching` خاصی برای آن در نظر گرفته نشده‌است، می‌توان از روش `nested containers` برای رها سازی خودکار منابع آن کمک گرفت.

در انتهای مطلب « [تزریق خودکار وابستگی‌ها در برنامه‌های ASP.NET MVC](#) » اشاره‌ای کوتاه به روش DependencyResolver توکار Web API شد که این روش پس از بررسی‌های بیشتر ([^](#) و [^](#)) به دلیل ماهیت service locator بودن آن و همچنین از دست دادن Context جاری Web API، مردود اعلام شده و استفاده از IHttpControllerActivator توصیه می‌گردد. در ادامه این روش را توسط Structure map 3 پیاده سازی خواهیم کرد.

پیش نیازها

- شروع یک پروژه‌ی جدید وب با پشتیبانی از Web API
- نصب دو بسته‌ی نیوگت مرتبط با Structure map 3

```
PM>install-package structuremap
PM>install-package structuremap.web
```

پیاده سازی IHttpControllerActivator توسط Structure map

```
using System;
using System.Net.Http;
using System.Web.Http.Controllers;
using System.Web.Http.Dispatcher;
using StructureMap;

namespace WebApiDISample.Core
{
    public class StructureMapHttpControllerActivator : IHttpControllerActivator
    {
        private readonly IContainer _container;
        public StructureMapHttpControllerActivator(IContainer container)
        {
            _container = container;
        }

        public IHttpController Create(
            HttpRequestMessage request,
            HttpControllerDescriptor controllerDescriptor,
            Type controllerType)
        {
            var nestedContainer = _container.GetNestedContainer();
            request.RegisterForDispose(nestedContainer);
            return (IHttpController)nestedContainer.GetInstance(controllerType);
        }
    }
}
```

در اینجا نحوه‌ی پیاده سازی IHttpControllerActivator را توسط StructureMap مشاهده می‌کنید. نکته‌ی مهم آن استفاده از [NestedContainer](#) آن است. معرفی آن به متد request.RegisterForDispose سبب می‌شود تا کلیه کلاس‌های IDisposable نیز در پایان کار به صورت خودکار رها سازی شده و نشتی حافظه رخ ندهد.

معرفی StructureMapHttpControllerActivator به برنامه

فایل WebApiConfig.cs را گشوده و تغییرات ذیل را در آن اعمال کنید:

```
using System.Web.Http;
using System.Web.Http.Dispatcher;
using StructureMap;
using WebApiDISample.Core;
using WebApiDISample.Services;
```

```

namespace WebApiDISample
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            // IoC Config
            ObjectFactory.Configure(c => c.For<IEmailsService>().Use<EmailsService>());

            var container = ObjectFactory.Container;
            GlobalConfiguration.Configuration.Services.Replace(
                typeof(IHttpControllerActivator), new StructureMapHttpControllerActivator(container));

            // Web API routes
            config.MapHttpAttributeRoutes();
            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}

```

در ابتدا تنظیمات متداول کلاس‌ها و اینترفیس‌ها صورت می‌گیرد. سپس نحوه‌ی معرفی StructureMapHttpControllerActivator را به GlobalConfiguration.Configuration.Services مخصوص Web API ملاحظه می‌کنید. این مورد سبب می‌شود تا به صورت خودکار کلیه وابستگی‌های مورد نیاز یک Web API Controller به آن تزریق شوند.

تهیه سرویسی برای آزمایش برنامه

```

namespace WebApiDISample.Services
{
    public interface IEmailsService
    {
        void SendEmail();
    }
}

using System;

namespace WebApiDISample.Services
{
    /// <summary>
    /// سرویسی که دارای قسمت دیسپوز نیز هست
    /// </summary>
    public class EmailsService : IEmailsService, IDisposable
    {
        private bool _disposed;

        ~EmailsService()
        {
            Dispose(false);
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        public void SendEmail()
        {
            //todo: send email!
        }

        protected virtual void Dispose(bool disposeManagedResources)
        {
            if (_disposed) return;
            if (!disposeManagedResources) return;

            //todo: clean up resources here ...
        }
    }
}

```

```

        _disposed = true;
    }
}

```

در اینجا یک سرویس ساده ارسال ایمیل را بدون پیاده سازی خاصی مشاهده می‌کنید. نکته‌ی مهم آن استفاده از IDisposable در این کلاس خاص است (ضروری نیست؛ صرفاً جهت بررسی بیشتر اضافه شده‌است). اگر در کدهای برنامه، یک چنین کلاسی وجود داشت، نیاز است متد Dispose آن نیز توسط IoC Container فراخوانی شود. برای آزمایش آن یک break point را در داخل متد Dispose قرار دهید.

استفاده از سرویس تعریف شده در یک Web API Controller

```

using System.Web.Http;
using WebApiDISample.Services;

namespace WebApiDISample.Controllers
{
    public class ValuesController : ApiController
    {
        private readonly IEmailsService _emailsService;
        public ValuesController(IEmailsService emailsService)
        {
            _emailsService = emailsService;
        }

        // GET api/values/5
        public string Get(int id)
        {
            _emailsService.SendEmail();
            return "_emailsService.SendEmail(); called!";
        }
    }
}

```

در اینجا مثال ساده‌ای را از نحوه‌ی تزریق سرویس ارسال ایمیل را در ValuesController مشاهده می‌کنید. تزریق وهله‌ی مورد نیاز آن، به صورت خودکار توسط StructureMapHttpControllerActivator که در ابتدای بحث معرفی شد، صورت می‌گیرد.

فراخوانی متد Get آن را نیز توسط کدهای سمت کاربر ذیل انجام خواهیم داد:

```

<h2>Index</h2>
@section scripts
{
    <script type="text/javascript">
        $(function () {
            $.getJSON('/api/values/1?timestamp=' + new Date().getTime(), function (data) {
                alert(data);
            });
        });
    </script>
}

```

درون متد Get کنترلر، یک break point قرار دهید. همچنین داخل متد Dispose لایه سرویس نیز جهت بررسی بیشتر یک break point قرار دهید.

اکنون برنامه را اجرا کنید. هنگام فراخوانی متد Get، وهله‌ی سرویس مورد نظر، نال نیست. همچنین متد Dispose نیز به صورت خودکار فراخوانی می‌شود.

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید

