عنوان: **تفاوت بین IQueryable و IEnumerable در حین کار با ORMs**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۸/۰۷ ۱۷:۴۴:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ

متد زیر را که یکی از اشتباهات رایج حین استفاده از LINQ خصوصا جهت Binding اطلاعات است، در نظر بگیرید:

```
IQueryable<Customer> GetCustomers()
```

این متد در حقیقت هیچ چیزی را Get نمی‌کند! نام اصلی آن GetQueryableCustomers و یا GetQueryObjectForCustomersاست.
IQueryable قلب LINQ است و تنها بیانگر یک عبارت (expression) از رکوردهایی می‌باشد که مد نظر شما است و نه بیشتر.

```
IQueryable<Customer> youngCustomers = repo.GetCustomers().Where(m => m.Age < 15);
```

برای مثال زمانیکه یک IQueryable را همانند مثال فوق فیلتر می‌کنید نیز هنوز چیزی از بانک اطلاعاتی یا منبع داده‌ای دریافت نشده است. هنوز هیچ اتفاقی رخ نداده است و هنوز رفت و برگشتی به منبع داده‌ای صورت نگرفته است.
به آن باید به شکل یک expression builder نگاه کرد و نه لیستی از اشیاء فیلتر شده‌ی ما. به این مفهوم، deferred execution (اجرای به تاخیر افتاده) نیز گفته می‌شود (باید دقت داشت که IQueryable هم یک نوع IEnumerable است به علاوه expression trees که مهم‌ترین وجه تمایز آن نیز می‌باشد).
برای مثال در عبارت زیر تنها در زمانیکه متد ToList فراخوانی می‌شود، کل عبارت LINQ ساخته شده، به عبارت SQL متناظر با آن ترجمه شده، اطلاعات از دیتابیس اخذ گردیده و حاصل به صورت یک لیست بازگشت داده می‌شود:

```
IList<Competitor> competitorRecords =  competitorRepository
  .Competitors
  .Where(m => !m.Deleted)
  .OrderBy(m => m.countryId)
  .ToList(); //فقط اینجا است که اس کیوال نهایی تولید می‌شود
```

در مورد IEnumerable ها چطور؟

```
IEnumerable<Product> products = repository.GetProducts();
var productsOver25 = products.Where(p => p.Cost >= 25.00);
```

دو سطر فوق به این معنا است:
لطفا ابتدا به بانک اطلاعاتی رجوع کن و تمام رکوردهای محصولات موجود را بازگشت بده. سپس بر روی این حجم بالای اطلاعات، محصولاتی را که قیمت بالای 25 دارند، فیلتر کن.

اگر همین دو سطر را با IQueryable بازنویسی کنیم چطور؟

```
IQueryable<Product> products = repository.GetQueryableProducts();
var productsOver25 = products.Where(p => p.Cost >= 25.00);
```

در سطر اول تنها یک عبارت LINQ ساخته شده است و بس. در سطر دوم نیز به همین صورت. در طی این دو سطر حتی یک رفت و برگشت به بانک اطلاعاتی صورت نخواهد گرفت. در ادامه اگر این اطلاعات به نحوی Select شوند (یا ToList فراخوانی شود، یا در طی یک حلقه برای مثال Iteration ایی روی این حاصل صورت گیرد یا موارد مشابه دیگر)، آنگاه کوئری SQL متناظر با عبارت LINQ فوق ساخته شده و بر روی بانک اطلاعاتی اجرا خواهد شد.

بدیهی است این روش منابع کمتری را نسبت به حالتی که تمام اطلاعات ابتدا دریافت شده و سپس فیلتر می‌شوند، مصرف می‌کند (حالت بازگشت تمام اطلاعات ممکن است شامل 20000 رکورد باشد، اما حالت دوم شاید فقط 5 رکورد را بازگشت دهد).

سؤال: پس IQueryable بسیار عالی است و از این پس کلا از IEnumerable ها دیگر نباید استفاده کرد؟
خیر! توصیه اکید طراحان این است که لطفا تا حد امکان متدهایی که IQueryable بازگشت می‌دهند ایجاد نکنید! IQueryable یعنی اینکه این نقطه‌ی آغازین کوئری در اختیار شما، بعد برو هر کاری که دوست داشتی با آن در طی لایه‌های مختلف انجام بده و هر زمانیکه دوست داشتی از آن یک خروجی تهیه کن. خروجی IQueryable به معنای مشخص نبودن زمان اجرای نهایی کوئری و همچنین مبهم بودن نحوه‌ی استفاده از آن است. به همین جهت متدهایی را طراحی کنید که IEnumerable بازگشت می‌دهند اما در بدنه‌ی آن‌ها به نحو صحیح و مطلوبی از IQueryable استفاده شده است. به این صورت حد و مرز یک متد کاملا مشخص می‌شود. متدی که واقعا همان فیلتر کردن محصولات را انجام می‌دهد، همان 5 رکورد را بازگشت خواهد داد؛ اما با استفاده از یک لیست یا یک IEnumerable و نه یک IQueryable که پس از فراخوانی متد نیز به هر نحو دلخواهی قابل تغییر است.

نظرات خوانندگان

نویسنده: Afshar Mohebbi
تاریخ: ۱۳۸۹/۰۸/۰۷ ۲۲:۰۱:۴۷

جالب بود. من هم چند وقت پیش به این موضوع برخورد کرده بودم: http://stackoverflow.com/questions/3949823/why-skip-and-take-does-not-work-when-passing-through-a-method

حتی یک مطلب کوچولو هم برای آن آماده کرده و در سیستم اتوماتیک وبلاگم برای انتشار گذاشته‌ام.

نویسنده: سامان نام نیک
تاریخ: ۱۳۸۹/۰۸/۰۸ ۱۱:۱۴:۳۸

مدت ها بود که سوال فوق در ذهنم بود
از توضیح مختصر و مفیدتون ممنون

نویسنده: علی اقدم
تاریخ: ۱۳۸۹/۰۸/۰۹ ۱۲:۱۳:۵۱

کاملا درسته، به خاطر Tricky بودن IQueryable شدیدا توصیه می کنم که اگر معماری چند لایه کار می کنید اصلا لایه Bussiness داده ها رو به صورت IQueryable به UI پاس نکنه و در عوض می تونید از IList استفاده کنید.

آقای نصیری بسیار جالب و آموزنده بود،ممنون

نویسنده: کیان
تاریخ: ۱۳۹۱/۰۶/۲۸ ۱۶:۲۰

آیا می‌شه به نوع **IList** بسنده کرد یا کاملا بسته به جایی که استفاده می‌کنیم ممکنه فرق کنه این قضیه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۲۸ ۱۶:۲۸

بستگی به مکان استفاده داره. اگر قرار است دو یا چند جستجو را انجام دهید، اینکارها باید با IQueryable داخل یک متد انجام شود، اما خروجی متد فقط باید لیست حاصل باشد؛ نه IQueryable ایی که انتهای آن باز است و سبب نشتی لایه سرویس شما در لایه‌های دیگر خواهد شد. IQueryable فقط یک expression است. هنوز اجرا نشده. زمانیکه First، ToList و امثال آن روی این عبارت فراخوانی شود تبدیل به SQL شده و سپس بر روی بانک اطلاعاتی اجرا می‌شود. به این deferred execution یا اجرای به تعویق افتاده گفته می‌شود.
اگر این عبارت را در اختیار لایه‌های دیگر قرار دهید، یعنی انتهای کار را بازگذاشته‌اید و حد و حدود سیستم شما مشخص نیست. شما اگر IQueryable بازگشت دهید، در لایه‌ای دیگر می‌شود یک join روی آن نوشت و اطلاعات چندین جدول دیگر را استخراج کرد؛ درحالیکه نام متد شما GetUsers بوده. بنابراین بهتر است به صورت صریح اطلاعات را به شکل List بازگشت دهید، تا انتهای کار باز نمانده و طراحی شما نشتی نداشته باشد.
طراحی یک لایه سرویس که خروجی IQueryable دارد نشتی دار درنظر گرفته شده و توصیه نمی‌شود. اصطلاحا leaky abstraction هم به آن گفته می‌شود؛ چون طراح نتوانسته حد و مرز سیستم خودش را مشخص کند و همچنین نتوانسته سازوکار درونی آن‌را به خوبی کپسوله سازی و مخفی نماید.

نویسنده: رضا بزرگی
تاریخ: ۱۳۹۱/۰۶/۲۹ ۹:۱۱

تفاوت بازگشت متد از نوع List و IList در اینجا چیست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۲۹ ۹:۳۱

این‌ها بیشتر مباحث طراحی API است. اگر از List استفاده کنید، مصرف کننده کتابخانه شما مجبور است فقط از List استفاده کند. List صرفا یک پیاده سازی خاص از IList است.

اگر از اینترفیس و قرارداد IList استفاده شود، آزادی عمل بیشتری را در اختیار مصرف کننده خواهید گذاشت. در اینجا مصرف کننده می‌تواند از هر پیاده سازی دلخواهی از IList برای کار با API شما استفاده کند. حتی مواردی که در زمان طراحی API اصلی وجود خارجی نداشته‌اند و بعدها پیاده سازی خواهند شد.