بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

Connection Resiliency

مدرس : مهندس افشین رفوآ

دوره آموزش MVC

Connection Resiliency (تلاش های مجدد خودکار برای رفع خطاهای موقتی) و Command Interception (رهگیری کوئری های ارسالی به پایگاه داده) به وسیله ی EF در MVC

تاکنون برنامه به طور محلی در نسخه ی Express سرور اطلاعات اینترنتی (IIS Express) در رایانه ی توسعه ی اپلیکیشن شما اجرا می شد. برای اینکه دیگران بتوانند از برنامه ی شما استفاده کنند، بایستی آن را بر روی یک ارائه دهنده ی میزبان وب (web hosting provider) و پایگاه داده را بر روی سرویس دهنده ی پایگاه داده (database provider) مستقر کنید.

در این فصل، با دو ویژگی جدید EF که حین مستقرسازی پایگاه داده در محیط CLOUD بسیار بکار می آیند، آشنا خواهید شد: 1. Connection resiliency (تلاش های مجدد خودکار برای خطاهای موقتی)، 2. Command interception (گرفتن جلوی query های ارسالی به پایگاه داده جهت تغییر یا ثبت گزارش و log کردن آن ها).

فعال سازی connection resiliency

هنگامی که برنامه را بر روی Windows Azure مستقر می کنید، درواقع آن را دارید بر روی پایگاه داده ی Windows Azure SQL پیاده می کنید که یک سرویس پایگاه داده cloud محسوب می شود. خطاهای

موقتی اتصال معمولا در شرایطی که سعی می کنید به cloud database service (سرویس پایگاه داده ی cloud) متصل شوید در مقایسه با زمانی که web server (سرویس دهنده ی وب) و database server (سرویس دهنده ی پایگاه داده) شما هر دو به طور مستقیم و در یک data center (مرکز داده ای) یکسان به هم وصل هستند، بیشتر رخ می دهند. حتی اگر یک cloud web server و یک cloud database service در یک data center واحد میزبانی (host) می شوند، اتصالات شبکه ای بیشتری بین آن ها وجود دارد که ممکن است دچار خطا شود، مانند بارگذاری متوازن (load balancer = بارکاری یا workload را بین چندین منبع همچون رایانه یا درایو دیسک و غیره .. توزیع می کند).

همچنین باید توجه داشت که یک cloud service معمولا بین چندین کاربر مشترک می باشد، بدین معنا که پاسخگویی آن تحت تاثیر تمامی کاربرانی که از آن استفاده می کنند قرار دارد. دسترسی به پایگاه داده ممکن است تحت تاثیر رخداد throttling قرار گیرد. Throttling به زمانی اشاره دارد که شما سعی می کنید به database service بیش از حد تعیین شده در موافقت نامه سطح خدمات (SLA) دسترسی پیدا کنید و throttling خطا صادر می کند.

بیشتر خطاهای اتصال که حین دسترسی به cloud service رخ می دهد، موقتی هستند، بدین معنا که با گذشت مدت زمان محدودی خودشان برطرف می شوند. بنابراین اگر تلاش شما برای اجرای یک عملیات مربوط به پایگاه داده با خطای موقتی رو به رو شد، کافی است مدتی بعد مجددا امتحان کنید، عملیات با موفقیت انجام خواهد شد. امکان connection resiliency که در ویرایش 6 تکنولوژی EF معرفی شد، پروسه ی امتحان کردن برای یافتن و برطرف ساختن خطاهای موقتی را به صورت خودکار مدیریت می کند تا مشتری (customer) بسیاری از این دست خطاها را مشاهده نکند و بدین وسیله تجربه ی روان برای کاربران به ارمغان می آورد.

امکان connection resiliency باید به درستی برای یک database service پیکربندی شود:

1. باید طوری تنظیم شده باشد که استثناها یا به عبارتی خطاهای احتمالی را بشناسد. خطاهایی باید retry شوند که از قطع یا از دست رفت موقتی اتصال در شبکه نشات می گیرد، نه خطاهایی که توسط یک اشکال در کدنویسی برنامه رخ می دهد.

2. باید گونه ای پیکربندی شده باشد تا مدت زمان مناسبی را بین تلاش های مجدد برای اجرای عملیات ناموفق صبر کند. به عنوان مثال مدت زمان بیشتری را می توان بین تلاش های مجدد برای اجرای عملیات دست جمعی (batch process) صبر کرد (در نظر گرفت) تا یک صفحه ی وب که کاربر صرفا منتظر دریافت پاسخ از سرور می باشد.

3. باید تعداد دفعات مناسبی پروسه را تکرار کند و در صورت ناموفق بودن دست از تلاش بردارد. تعداد دفعات تلاش مجدد بایستی در اجرای عملیات دسته جمعی (batch process) در مقایسه با یک برنامه ی آنلاین بیشتر باشد.

این دسته از تنظیمات را می توانید برای هر محیط بانک اطلاعاتی که توسط EF provider پشتیبانی می شود، به صورت دستی پیکربندی کنید. اما آن دسته از تنظیماتی که برای برنامه ی آنلاین که از Windows Azure SQL Database بهره می گیرد، مناسب می باشد از قبل برای شما پیکربندی شده و ما همین تنظیمات را برای برنامه ی Contoso University پیاده سازی می کنیم.

تنها کاری که لازم است برای فعال سازی connection resiliency انجام دهید، ایجاد یک کلاس در assembly می باشد که از کلاس DbConfiguration مشتق می شود (به ارث گرفته می شود)، سپس در آن کلاس SQL Database execution strategy را تنظیم کنید که در EF معادل همان retry policy می باشد.

1. پوشه ی DAL را باز کرده و یک فایل کلاس به نام SchoolConfiguration.cs به آن اضافه کنید.

2. کد زیر را جایگزین template code کنید:

```
using System.Data.Entity;
using System.Data.Entity.SqlServer;

namespace ContosoUniversity.DAL
{
  public class SchoolConfiguration : DbConfiguration
  {
    public SchoolConfiguration()
    {
      SetExecutionStrategy("System.Data.SqlClient", () => new SqlAzureExecutionStrategy());
```

```
    }
  }
}
```

EF آن کدی را که در کلاس مشتق شده از DbConfiguration می یابد، به صورت خودکار اجرا می کند. می توانید با استفاده از کلاس DbConfiguration عملیات و تسک های مربوط پیکربندی را که می بایست در فایل Web.config انجام دهید، حال در کد اجرا کنید.

3. در فایل StudentController.cs، یک دستور using به فضای نام System.Data.Entity.Infrastructure اضافه کنید.

```
using System.Data.Entity.Infrastructure;
```

4. تمامی قطعه کدهای مدیریت خطای catch را که خطاها (استثناها) را ضبط (catch) می کنند، به گونه ای تغییر دهید که کلیه ی خطاهای RetryLimitExceededException را ضبط کنند. مثال:

```
catch (RetryLimitExceededException /* dex */)
{
        //ثبت خطا (به جای عبارت dex نام متغیر را قرار داده  و آن را از حالت Comment خارج کنید و یک خط  Log در این قسمت قرار دهید.)
  ModelState.AddModelError("", "Unable to save changes. Try again, and if the problem persists see your system administrator.");

    }
```

در اینجا با استفاده از DataException فقط خطاهایی را شناسایی می کردیم که احتمالا موقتی بوده، سپس یک پیام خطای دوستانه ی " try again " به کاربر نمایش می دادیم. اما حالا که retry policy را فعال سازی کرده ایم، خطاهایی که ممکن است موقتی باشند، قطعا بارها و بارها تکرار شده و با عدم موفقیت مکرر در قالب یک خطا یا استثنای واقعی RetryLimitExceededException قرار گرفته و بازگردانده می شود.

فعال سازی امکان command interception

پس از فعال سازی retry policy، چگونه مطمئن شویم که retry policy به گونه ی مورد انتظار عمل می کند یا خیر؟ این که کاری کنید که خطای موقتی به زور رخ دهد، کار ساده ای نخواهد بود به خصوص زمانی که از مرورگر و به صورت محلی برنامه را اجرا می کنید. گنجاندن خطاهای موقتی حقیقی در یک unit test (آزمایش واحد = روشی است برای آزمودن واحدهای کوچکی از کد منبع برنامه و اطمینان از درست کار کردن آن‌ها) اتوماتیک نیز

دو چندان دشوار خواهد بود. به منظور تست امکان connection resiliency، به یک روش نیاز دارید که در آن query های ارسالی از EF به SQL Server را intercept (به عبارتی جلوی آن ها را گرفته، سپس آن ها تغییر داده یا ثبت) کرده، سپس پاسخ SQL Server را با یک خطایی که معمولا از نوع موقتی است جایگزین کنید.

همچنین می توانید با intercept کردن query برای ثبت یا تغییر آن ها، یک روش بسیار کارامد و پسندیده را برای برنامه های Cloud پیاده سازی کنید: ثبت تاخیر (latency) و موفقت یا عدم موفقیت فراخوانی سرویس های خارجی همچون database services. EF6 یک رابط برنامه سازی کاربردی ثبت گزارش اختصاصی (logging API) ارائه می دهد که فرایند ثبت گزارش یا LOGGING را سهل می سازد، اما در این بخش از مقاله ی آموزشی حاضر با نحوه ی استفاده از امکان interception تکنولوژی EF برای ثبت گزارش و شبیه سازی خطاهای موقتی به صورت مستقیم، آشنا خواهید شد.

ایجاد یک کلاس و رابط logging (interface)

بهترین روش برای ثبت گزارش (گرفتن logging)، استفاده از یک interface است؛ بهتر است بجای فراخوانی ها مکرر از System.Diagnostics.Trace یا ایجاد یک کلاس logging از یک interface جهت ثبت گزارش استفاده شود. این کار تغییر مکانیزم logging را در صورت نیاز در آینده، سهل می سازد. بنابراین در این بخش، یک رابط logging و همچنین یک کلاس برای پیاده سازی آن ایجاد خواهیم کرد.

1. یک پوشه در پروژه ی خود به نام Logging ایجاد می کنیم.

2. در پوشه ی Logging، یک فایل کلاس به نام ILogger.cs ایجاد کرده، سپس template code را با کد زیر جایگزین نمایید:

```
using System;

namespace ContosoUniversity.Logging
{
  public interface ILogger
  {
    void Information(string message);
    void Information(string fmt, params object[] vars);
    void Information(Exception exception, string fmt, params object[] vars);
```

```
        void Warning(string message);
        void Warning(string fmt, params object[] vars);
        void Warning(Exception exception, string fmt, params object[] vars);

        void Error(string message);
        void Error(string fmt, params object[] vars);
        void Error(Exception exception, string fmt, params object[] vars);

        void TraceApi(string componentName, string method, TimeSpan timespan);
        void TraceApi(string componentName, string method, TimeSpan timespan, string properties);
        void TraceApi(string componentName, string method, TimeSpan timespan, string fmt, params object[] vars);

    }
}
```

این interface سه لایه ی tracing برای نمایش اهمیت نسبی گزارشات (logs) و یک لایه ویژه ی ارائه ی اطلاعات مربوط به تاخیر (latency) برای فراخوانی های سرویس خارجی (external service call) همچون database query ها فراهم می نماید. متدهای logging دارای نسخه های overload شده هستند که امکان ارسال exception به آن ها را فراهم می نماید. این به این دلیل است که اطلاعات خطا از قبیل stack trace (رهگیری خطاهای تودرتو) و inner exceptions (خطاهای داخلی) بجای اینکه در هر بار فراخوانی متد logging سرار برنامه انجام شود، به طور قابل اطمینان توسط کلاسی که interface را پیاده سازی می شود، انجام گیرد.

متدهای TraceApi به شما این امکان را می دهد که latency (میزان نهفتگی یا تاخیر) هر بار فراخوانی سرویس های خارجی همچون SQL Database را رهگیری (track) کنید.

3. در پوشه ی Logging، یک فایل کلاس به نام Logger.cs ایجاد کرده و template code را با کد زیر جایگزین نمایید:

```
using System;
using System.Diagnostics;
using System.Text;

namespace ContosoUniversity.Logging
{
  public class Logger : ILogger
  {

    public void Information(string message)
    {
      Trace.TraceInformation(message);
    }
```

```
public void Information(string fmt, params object[] vars)
{
  Trace.TraceInformation(fmt, vars);
}

public void Information(Exception exception, string fmt, params object[] vars)
{
  Trace.TraceInformation(FormatExceptionMessage(exception, fmt, vars));
}

public void Warning(string message)
{
  Trace.TraceWarning(message);
}

public void Warning(string fmt, params object[] vars)
{
  Trace.TraceWarning(fmt, vars);
}

public void Warning(Exception exception, string fmt, params object[] vars)
{
  Trace.TraceWarning(FormatExceptionMessage(exception, fmt, vars));
}

public void Error(string message)
{
  Trace.TraceError(message);
}

public void Error(string fmt, params object[] vars)
{
  Trace.TraceError(fmt, vars);
}

public void Error(Exception exception, string fmt, params object[] vars)
{
  Trace.TraceError(FormatExceptionMessage(exception, fmt, vars));
}

public void TraceApi(string componentName, string method, TimeSpan timespan)
{
  TraceApi(componentName, method, timespan, “”);
}

public void TraceApi(string componentName, string method, TimeSpan timespan, string fmt, params object[] vars)
{
  TraceApi(componentName, method, timespan, string.Format(fmt, vars));
}
```

آدرس آموزشگاه : تهران - خیابان شریعتی - بالا تر از خیابان ملک - جنب بانک صادرات - پلاک - 165 واحد7
88146323 - 88446780 - 88146330

```
public void TraceApi(string componentName, string method, TimeSpan timespan, string properties)
{
  string message = String.Concat("Component:", componentName, ";Method:", method, ";Timespan:",
timespan.ToString(), ";Properties:", properties);
  Trace.TraceInformation(message);
}

private static string FormatExceptionMessage(Exception exception, string fmt, object[] vars)
{

 //قالب ساده خطا:جهت دریافت نسخه جامع تر به این آدرس مراجعه کنید.
  // http://code.msdn.microsoft.com/windowsazure/Fix-It-app-for-Building-cdd80df4
  var sb = new StringBuilder();
  sb.Append(string.Format(fmt, vars));
  sb.Append(" Exception: ");
  sb.Append(exception.ToString());
  return  sb.ToString();
}
}
    }
```

این پیاده سازی با استفاده از System.Diagnostics، عملیات trace را انجام می دهد. این امکان یک قابلیت توکار یا درون ساخته ی NET. است که ایجاد و استفاده از اطلاعات tracing (= استفاده ی تخصصی از logging جهت ثبت اطلاعات مربوط به اجرای یک برنامه) را سهل می سازد. " listener " ها (گوش فراخوان های) فراوانی وجود دارد، که شما می توانید همراه با System.Diagnostics tracing برای نوشتن گزارشات در فایل ها یا نوشتن آن ها در محل ذخیره سازی blob (مجموعه ای از داده های دودویی که در قالب یک موجودیت واحد ذخیره می شوند) در Azure استفاده کنید.

در یک برنامه ی تولیدی (production app) ممکن است بخواهید از tracing package (پکیج های رد گیری و ثبت اطلاعات اجرای برنامه) ای به غیر از System.Diagnostics استفاده کنید، برای این منظور می توانید از ILogger interface کمک بگیرید. این رابط تغییر به یک مکانیزم tracing دیگر را برای شما آسان می سازد.

ایجاد کلاس های interceptor

در این مرحله کلاس هایی را ایجاد می کنیم که EF هر بار که یک query به پایگاه داده ارسال می کند، صدا می زند، یکی برای شبیه سازی خطاهای موقتی و دیگری ویژه ی ثبت گزارش یا logging. کلاس هایی که ایجاد می کنیم، کلاس های interceptor هستند که بایستی از کلاس DbCommandInterceptor مشتق (به ارث بری) شوند. در این کلاس ها بازنویسی متدها را (method override) write می کنیم که به هنگام اجرای

query ها به صورت خودکار فراخوانی می‌شوند. در این متدها می توانید query ای که در حال ارسال به پایگاه داده است را مورد بررسی قرار داده و ثبت گزارش (log) کنید، همچنین می توانید query را پیش از اینکه به پایگاه داده فرستاده شود تغییر دهید و یا بدون اینکه query را به پایگاه داده ارسال کنید، خود چیزی را به عنوان خروجی به EF برگردانید.

1. جهت ایجاد کلاس interceptor به منظور ثبت گزارش هر SQL Query ای که به پایگاه داده ارسال می شود، یک فایل کلاس به نام SchoolInterceptorLogging.cs داخل پوشه ی DAL ایجاد کرده، سپس template code زیر را با کد زیر جایگزین کنید:

```
    using System;
using System.Data.Common;
using System.Data.Entity;
using System.Data.Entity.Infrastructure.Interception;
using System.Data.Entity.SqlServer;
using System.Data.SqlClient;
using System.Diagnostics;
using System.Reflection;
using System.Linq;
using ContosoUniversity.Logging;

namespace ContosoUniversity.DAL
{
  public class SchoolInterceptorTransientErrors : DbCommandInterceptor
  {
    private int _counter = 0;
    private ILogger _logger = new Logger();

    public override void ReaderExecuting(DbCommand command, DbCommandInterceptionContext<DbDataReader>
interceptionContext)
    {
      bool throwTransientErrors = false;
      if (command.Parameters.Count > 0 && command.Parameters[0].Value.ToString() == "%Throw%")
      {
        throwTransientErrors = true;
        command.Parameters[0].Value = "%an%";
        command.Parameters[1].Value = "%an%";
      }

      if (throwTransientErrors && _counter < 4)
      {
        _logger.Information("Returning transient error for command: {0}", command.CommandText);
        _counter++;
        interceptionContext.Exception = CreateDummySqlException();
      }
```

```
    }

    private SqlException CreateDummySqlException()
    {
                        //این یک نمونه از Sql Serverای که شما تلاش میکنید به آن وصل بشوید این نسخه کد گذاری شده را پشتیبانی نمیکند.
        var sqlErrorNumber = 20;

        var sqlErrorCtor = typeof(SqlError).GetConstructors(BindingFlags.Instance | BindingFlags.NonPublic).Where(c
=> c.GetParameters().Count() == 7).Single();
        var sqlError = sqlErrorCtor.Invoke(new object[] { sqlErrorNumber, (byte)0, (byte)0, "", "", "", 1 });

        var errorCollection = Activator.CreateInstance(typeof(SqlErrorCollection), true);
        var addMethod = typeof(SqlErrorCollection).GetMethod("Add", BindingFlags.Instance |
BindingFlags.NonPublic);
        addMethod.Invoke(errorCollection, new[] { sqlError });

        var sqlExceptionCtor = typeof(SqlException).GetConstructors(BindingFlags.Instance |
BindingFlags.NonPublic).Where(c => c.GetParameters().Count() == 4).Single();
        var sqlException = (SqlException)sqlExceptionCtor.Invoke(new object[] { "Dummy", errorCollection, null,
Guid.NewGuid() });

        return sqlException;
    }
  }
}
```

این کد فقط متد ReaderExecuting را بازنویسی (override) می کند. متد ذکر شده برای query هایی فراخوانده می شود که قادر به بازگردانی چندین سطر از داده هستند. اگر می خواهید امکان connection resiliency را برای دیگر نوع query ها (برای مثال query ای که تک مقدار برمی گرداند) آزمایش کنید، در آن صورت می توانید متدهای NonQueryExecuting و ScalarExecuting را بازنویسی کنید، دقیقا همان کاری که logging interceptor انجام می دهد. هنگامی که صفحه ی Student را اجرا کرده و "Throw" را به عنوان عبارت جستجو (search string) وارد می کنید، کد حاضر یک استثنا یا SQL Database exception ساختگی برای خطای شماره ی 20 ایجاد می کند، این نوع خطا اغلب تحت عنوان خطای موقتی شناخته می شود. دیگر شماره خطاهایی که در این دسته قرار می گیرند عبارتند از: 64، 233، 10053، 10054، 10060، 10928، 10929، 40197، 40501، abd، 40613 . گفتنی است این شماره خطاها ممکن است در ویرایش های جدید SQL Database متفاوت باشد.

کد، بجای اینکه query را اجرا کرده و نتایج query را بازگرداند، خطای مورد نظر را به EF برمی گرداند. خطای موقتی چهار بار برگردانده می شود، پس از آن کد به روال عادی ارسال query به پایگاه داده برمی گردد.

از آنجایی که همه چیز ثبت گزارش می شود، شما می توانید خودتان ببینید که EF چهار بار سعی می کند که query را اجرا کند تا در نهایت موفق به اجرای آن می شود. تنها تفاوتی که می توان در این برنامه مشاهده کرد این است که اجرا و نمایش صفحاتی که نتایج query را نشان می دهد بیشتر طول می کشد.

تعداد دفعاتی که EF سعی می کند query را اجرا کند، قابل تنظیم می باشد؛ کد به این خاطر چهار بار اجرای query را تکرار می کند که این مقدار، مقدار پیش فرض می باشد. اگر سیاست اجرا (execution policy) را تغییر دهید، در آن صورت کدی را که تعیین کننده ی تعداد دفعات تکرار خطاهای موقتی است را نیز تغییر می دهید. همچنین می توانید کد را به گونه ای تغییر دهید که خطاهای بیشتری را صادر کند و همچنین باعث شود که EF خطای RetryLimitExceededException را صادر (throw) کند.

مقداری که در کادر جستجو (Search box) وارد می کنید، command.Parameters[0] و command.Parameters[1] خواهد بود (یکی برای first name و دیگری برای last name). پس از یافت شدن "%Throw%"، "Throw" داخل آن پارامترها با واژه ی "an" جایگزین شده و از این طریق برخی از دانشجویان پیدا شده و بازیابی می شود. این تنها یک روش آسان برای تست connection resiliency است که بر پایه ی تغییر برخی ورودی ها به رابط کاربری برنامه (application UI) صورت می گیرد. شما همچنین می توانید کدی بنویسید که خطاهای موقتی برای تمامی query ها و بروز رسانی ها ایجاد می کند، همان طور که بعدها در توضیحات متد DbInterception.Add مشاهده می کنید.

4. فایل Global.asax را باز کرده و دستور using زیر را به آن اضافه کنید:

```
using ContosoUniversity.DAL;
using System.Data.Entity.Infrastructure.Interception;
```

5. خط های رنگی شده را به متد Application_Start اضافه نمایید:

```
protected void Application_Start()
    {
      AreaRegistration.RegisterAllAreas();
      FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
      RouteConfig.RegisterRoutes(RouteTable.Routes);
      BundleConfig.RegisterBundles(BundleTable.Bundles);
      DbInterception.Add(new SchoolInterceptorTransientErrors());
      DbInterception.Add(new SchoolInterceptorLogging());
    }
```

کدهای هایلایت شده ی مثال بالا، باعث می شوند که کد interceptor به محض اینکه EF کوئری ها را به پایگاه داده فرستاد، همان زمان اجرا شود. توجه داشته باشید که به دلیل ایجاد کلاس های مجزای interceptor برای شبیه سازی خطاهای موقتی (transient errors) و ثبت گزارش (logging)، می توانید هر یک را به صورت مستقل فعال یا غیرفعال کنید.

می توانید با درج کردن متد DbInterception.Add به هر جایی از کد خود، interceptor هایی را اضافه کنید؛ لزومی ندارد که متد مزبور را حتما در متد Application_Start لحاظ نمایید. روش دگیری که وجود دارد این است که کد مورد نظر را در کلاس DbConfiguration خود (که قبلا ایجاد کرده بودید) جای گذاری نمایید تا بتوانید سیاست اجرا (execution policy) را تنظیم و پیکربندی کنید.

```
    public class SchoolConfiguration : DbConfiguration
    {
      public SchoolConfiguration()
      {
        SetExecutionStrategy("System.Data.SqlClient", () => new SqlAzureExecutionStrategy());
        DbInterception.Add(new SchoolInterceptorTransientErrors());
        DbInterception.Add(new SchoolInterceptorLogging());
      }
            }
```

هنگامی که از این کد استفاده می کنید، لازم است دقت داشته باشید که متد DbInterception.Add را برای یک interceptor بیش از یکبار اجرا نکنید، زیرا در آن صورت با نمونه های اضافه بر سازمان interceptor مواجه می شوید. به عنوان مثال، چنانچه logging interceptor را دوبار اضافه کنید، در آن صورت به ازای هر SQL query دو log مشاهده خواهید کرد.

Interceptor ها به ترتیب registeration یا ثبت اجرا می شوند (ترتیب فراخوانی متد DbInterception.Add). این ترتیب بسته به عملیاتی که در interceptor انجام می دهید، از میزان اهمیت خاصی برخوردار است. برای مثال، ممکن است interceptor دستور SQL ای را که در خاصیت CommandText دریافت می کند، تغییر دهد. اگر interceptor دستور SQL را تغییر داد، در آن صورت interceptor بعدی دستور اصلاح شده ی SQL را دریافت می کند و این دستور به دست فرمان اصلی SQL نمی رسد.

کد شبیه ساز خطای موقتی را به گونه ای نوشته اید که به شما اجازه دهد با وارد کردن یک مقدار متفاوت در رابط کاربری (UI)، خطای موقتی ایجاد کنید. یا به روش دیگر، می توان کد interceptor را گونه ای تنظیم کرد تا همیشه رشته یا توالی خطاهای موقتی را ایجاد کند بدون اینکه به دنبال مقدار پارامتر خاصی بگردد. در این موقعیت می توانید interceptor را فقط در زمانی که می خواهید خطاهای موقتی ایجاد شود، اضافه نمایید. توجه داشته باشید که اگر می خواهید از این روش بهره بگیرید، بایستی تا اتمام شروع به کار پایگاه داده (database initialization) صبر کنید، سپس interceptor را اضافه کنید. به عبارتی دیگر، قبل از شروع به ایجاد خطاهای موقتی، اول یک عملیات ساده مانند اجرای query بر روی یکی از entity set های خود انجام دهید. EF در طول شروع به کار پایگاه داده (database initialization) چندین query اجرا می کند. این query ها در تراکنش (transaction) اجرا نمی شوند، بنابراین چنانچه خطاها حین شروع به کار پایگاه داده ایجاد شوند، ممکن است باعث شود context (بستر اجرا) در وضعیت ناپایدار و inconsistent قرار گیرد.

مشاهده و بررسی logging و connection resiliency

1. با زدن دکمه ی F5، برنامه را در مد debug اجرا کرده، سپس تب Students را کلیک نمایید.

2. در محیط ویژوال، می توانید خروجی tracing را در پنجره ی Output مشاهده کنید. شاید مجبور باشید با نوار پیمایش (scroll) تعدادی خط کد جاوا اسکریپت را رد کرده تا به گزارشات ثبت شده توسط logger خود برسید.

در این پنجره می توانید SQL query های ارسال شده به پایگاه داده را مشاهده کنید. همچنین تعدادی query و دستور اولیه قابل رویت می باشد که EF با استفاده از آن ها راه اندازی شده و در

این میان نسخه ی پایگاه داده و جدول دربردارنده ی اطلاعات مربوط به تاریخچه ی انتقال (migration history) را مورد بررسی قرار می دهد. یک query برای صفحه بندی (paging) می بینید که توسط آن به تعداد دانشجویان (student) پی می بریم. در نهایت query ای را می بینید که اطلاعات مربوط به دانشجویان را بازیابی می کند.

Tahlildadeh

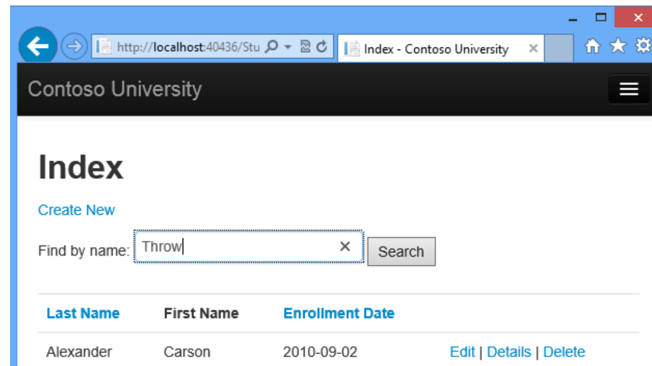
Output

Show output from: Debug

```
iisexpress.exe Information: 0 : Component:SQL
  Database;Method:SchoolInterceptor.ReaderExecuted;Timespan:00:00:00.0276705;Properties
  :Command: select serverproperty('EngineEdition'):
iisexpress.exe Information: 0 : Component:SQL
  Database;Method:SchoolInterceptor.ScalarExecuted;Timespan:00:00:00.0424968;Properties
  :Command: SELECT Count(*) FROM sys.databases WHERE [name]=N'ContosoUniversity2':
iisexpress.exe Information: 0 : Component:SQL
  Database;Method:SchoolInterceptor.ScalarExecuted;Timespan:00:00:00.0450531;Properties
  :Command:
SELECT Count(*)
FROM INFORMATION_SCHEMA.TABLES AS t
WHERE t.TABLE_TYPE = 'BASE TABLE'
    AND (t.TABLE_SCHEMA + '.' + t.TABLE_NAME IN
      ('dbo.Course','dbo.Department','dbo.Instructor','dbo.OfficeAssignment','dbo.Enrol
      lment','dbo.Student','dbo.CourseInstructor')
        OR t.TABLE_NAME = 'EdmMetadata'):
iisexpress.exe Information: 0 : Component:SQL
  Database;Method:SchoolInterceptor.ScalarExecuted;Timespan:00:00:00.0262633;Properties
  :Command: SELECT Count(*) FROM sys.databases WHERE [name]=N'ContosoUniversity2':
iisexpress.exe Information: 0 : Component:SQL
  Database;Method:SchoolInterceptor.ReaderExecuted;Timespan:00:00:00.0373571;Properties
  :Command: SELECT
    [GroupBy1].[A1] AS [C1]
    FROM ( SELECT
        COUNT(1) AS [A1]
        FROM [dbo].[__MigrationHistory] AS [Extent1]
    )  AS [GroupBy1]:
iisexpress.exe Information: 0 : Component:SQL
  Database;Method:SchoolInterceptor.ReaderExecuted;Timespan:00:00:00.0227459;Properties
  :Command: SELECT
    [GroupBy1].[A1] AS [C1]
    FROM ( SELECT
        COUNT(1) AS [A1]
        FROM [dbo].[__MigrationHistory] AS [Extent1]
        WHERE ([Extent1].[ContextKey] = @p__linq__0) AND (@p__linq__0 IS NOT NULL)
    )  AS [GroupBy1]:
iisexpress.exe Information: 0 : Component:SQL
  Database;Method:SchoolInterceptor.ReaderExecuted;Timespan:00:00:00.0298657;Properties
  :Command: SELECT TOP (1)
    [Project1].[C1] AS [C1],
    [Project1].[MigrationId] AS [MigrationId],
    [Project1].[Model] AS [Model]
    FROM ( SELECT
        [Extent1].[MigrationId] AS [MigrationId],
        [Extent1].[Model] AS [Model],
        1 AS [C1]
        FROM [dbo].[__MigrationHistory] AS [Extent1]
        WHERE ([Extent1].[ContextKey] = @p__linq__0) AND (@p__linq__0 IS NOT NULL)
    )  AS [Project1]
    ORDER BY [Project1].[MigrationId] DESC:
iisexpress.exe Information: 0 : Component:SQL
  Database;Method:SchoolInterceptor.ReaderExecuted;Timespan:00:00:00.0246607;Properties
  :Command: SELECT
    [GroupBy1].[A1] AS [C1]
    FROM ( SELECT
        COUNT(1) AS [A1]
        FROM [dbo].[Student] AS [Extent1]
    )  AS [GroupBy1]:
'iisexpress.exe' (CLR v4.0.30319: /LM/W3SVC/12/ROOT-1-130329870303366864): Loaded
  'EntityFrameworkDynamicProxies-ContosoUniversity'.
iisexpress.exe Information: 0 : Component:SQL
  Database;Method:SchoolInterceptor.ReaderExecuted;Timespan:00:00:00.0225311;Properties
  :Command: SELECT TOP (3)
    [Extent1].[ID] AS [ID],
    [Extent1].[LastName] AS [LastName],
    [Extent1].[FirstName] AS [FirstName],
    [Extent1].[EnrollmentDate] AS [EnrollmentDate]
    FROM ( SELECT [Extent1].[ID] AS [ID], [Extent1].[LastName] AS [LastName],
      [Extent1].[FirstName] AS [FirstName], [Extent1].[EnrollmentDate] AS
      [EnrollmentDate], row_number() OVER (ORDER BY [Extent1].[LastName] ASC) AS
      [row_number]
        FROM [dbo].[Student] AS [Extent1]
    )  AS [Extent1]
    WHERE [Extent1].[row_number] > 0
    ORDER BY [Extent1].[LastName] ASC:
```

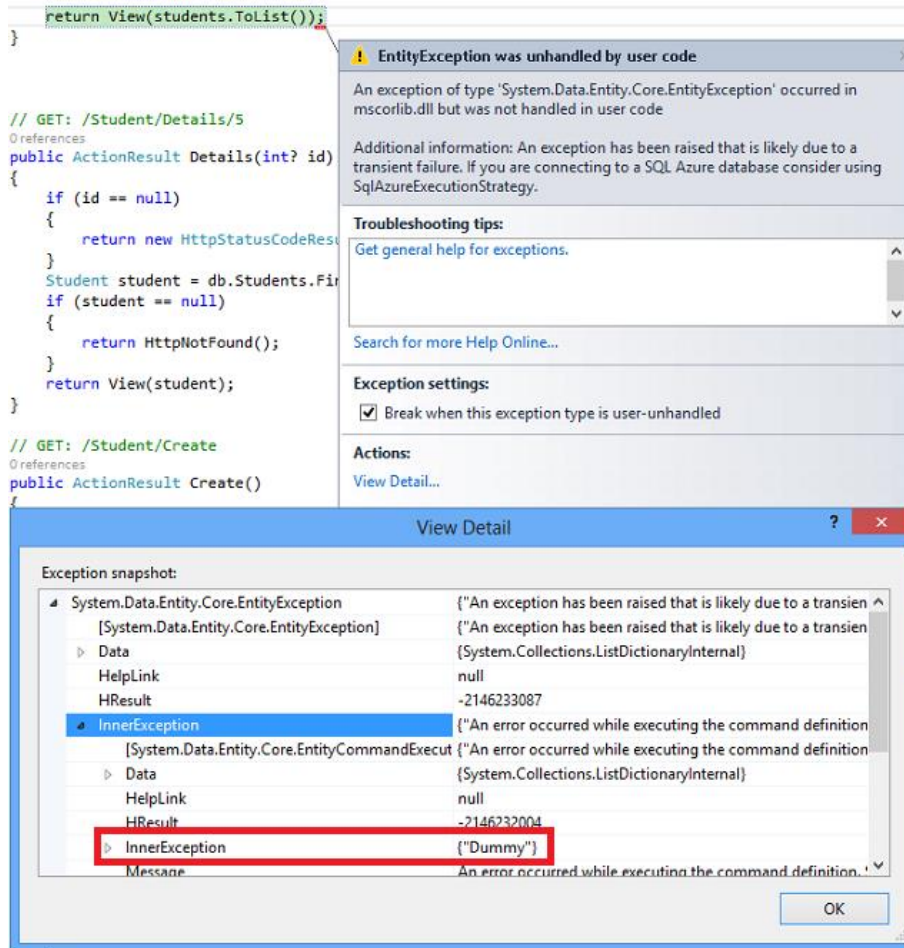
آدرس آموزش
88146323 - 88446780 - 88146330

3. در پنجره ی Students، واژه ی "Throw" را به عنوان عبارت جستجو (search string) وارد نموده، سپس Search را کلیک نمایید.



4. خواهید دید که مرورگر، زمانی که EF در حال اجرای مجدد query مورد نظر می باشد، به مدت چند ثانیه هنگ می کند. اولین retry یا تلاش مجدد به سرعت اتفاق می افتد، پس از اولین تلاش مجدد مدت زمانی که بایستی قبل از هر بار retry اضافی صبر کنید، به مراتب افزایش می یابد. این پروسه ی صبر کردن بیش از حد، قبل از هر بار تلاش مجدد، exponential backoff خوانده می شود.

با اجرا و نمایش صفحه، آن دانش آموزانی که نام آن ها شامل "an" می باشد نشان داده می شوند. حال اگر به پنجره ی output نگاه کنید، می بینید که query مورد نظر پنج بار اجرا شده که چهار بار اول منجر به بازیابی خطاهای موقتی شده است. برای هر خطای موقتی log ی را مشاهده می کنید که در زمان ایجاد کلاس SchoolInterceptorTransientErrors برای خطاهای موقتی نوشته بودید ("Returning transient error for command...") و زمانی که SchoolInterceptorLogging با یک استثنا مواجه شود log نوشته شده قابل مشاهده می باشد.



5. توضیح (comment) خط SetExecutionStrategy را در فایل SchoolConfiguration.cs حذف نمایید.

چکیده

در این مبحث با نحوه ی فعال سازی connection resiliency و ثبت گزارش دستورات SQL که توسط EF نوشته شده و به پایگاه داده ارسال می شود، آشنا شدید. در درس بعدی برنامه را روی اینترنت مستقر کرده و با استفاده از Code First Migrations پایگاه داده را نصب (deploy) می کنیم.