عنوان: مقاومت اتصال و اتصالات بهبودپذیر در Entity framework 6
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۲۹ ۱۲:۱۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

Timeouts، Deadlocks و قطعی‌های احتمالی و موقت اتصال به بانک اطلاعاتی در شبکه، جزئی از ساختار دنیای واقعی هستند. در EF 6 برای پیاده سازی سعی مجدد در اتصال و انجام مجدد عملیات، ویژگی خاصی تحت عنوان connection resiliency اضافه شده‌است که در ادامه مثالی از آن‌را بررسی خواهیم کرد.

پیاده سازی‌های پیش فرض موجود

برای پیاده سازی منطق سعی مجدد در اتصال، باید اینترفیس IDbExecutionStrategy پیاده سازی شود. در EF 6 حداقل 4 نوع پیاده سازی پیش فرض از آن به صورت توکار ارائه شده‌است :

الف) DefaultExecutionStrategy : حالت پیش فرض است و در صورت بروز مشکل، سعی مجددی را در اتصال، به عمل نخواهد آورد.

ب) DefaultSqlExecutionStrategy : برای کارهای درونی EF از آن استفاده می‌شود. سعی مجددی در اتصال قطع شده نخواهد کرد؛ اما جزئیات خطاهای بهتری را در اختیار مصرف کننده قرار می‌دهد.

ج) DbExecutionStrategy : هدف از آن تهیه یک کلاس پایه است برای نوشتن استراتژی‌های سعی مجدد سفارشی.

د) SqlAzureExecutionStrategy : یک نمونه DbExecutionStrategy سفارشی تهیه شده برای ویندوز اژور است. برای فعال سازی و تعریف آن نیز باید به نحو ذیل عمل کرد:

```
public class MyConfiguration : DbConfiguration
{
    public MyConfiguration()
    {
        SetExecutionStrategy("System.Data.SqlClient", () => new SqlAzureExecutionStrategy());
    }
}
```

تهیه یک DbExecutionStrategy سفارشی برای SQL Server

همانطور که عنوان شد، هدف از کلاس DbExecutionStrategy، تهیه یک کلاس پایه، جهت نوشتن منطق سعی مجدد در اتصال به بانک اطلاعاتی است و این مورد از دیتابیسی به دیتابیس دیگر می‌تواند متفاوت باشد؛ زیرا خطاهایی را که ارائه می‌دهند، یکسان و یک دست نیستند. در ادامه یک پیاده سازی سفارشی را از DbExecutionStrategy، جهت SQL Server مرور خواهیم کرد:

```
public class SqlServerExecutionStrategy : DbExecutionStrategy
    {
        public SqlServerExecutionStrategy()
        { }

        public SqlServerExecutionStrategy(int maxRetryCount, TimeSpan maxDelay)
            : base(maxRetryCount, maxDelay)
        { }

        protected override bool ShouldRetryOn(Exception ex)
        {
            var sqlException = ex as SqlException;
            if (sqlException == null)
                return false; // don't retry

            foreach (var error in sqlException.Errors.Cast<SqlError>())
            {
                switch (error.Number)
                {
                    case 1205: // Deadlock
                    case -1: // Timeout
                    case -2: // Timeout
                        return true; // retry
                }
```

```
            }
            return false;
        }
    }
```

در اینجا کار با بازنویسی متد ShouldRetryOn شروع می‌شود. این متد اگر پس از بررسی استثنای دریافتی، مقدار true را برگرداند، به معنای نیاز به سعی مجدد در اتصال است و برعکس. سازنده پیش فرض این کلاس طوری تنظیم شده‌است که 5 بار سعی مجدد کند؛ با فواصل زمانی 7 ثانیه. اگر می‌خواهید این زمان را صریحا تعیین کنید باید متد GetNextDelay کلاس پایه را نیز بازنویسی کرد:

```
protected override TimeSpan? GetNextDelay(Exception lastException)
  {
        return base.GetNextDelay(lastException);
  }
```

در ادامه برای استفاده از آن خواهیم داشت:

```
public class MyDbConfiguration : DbConfiguration
    {
        public MyDbConfiguration()
        {
            SetExecutionStrategy("System.Data.SqlClient", () => new SqlServerExecutionStrategy());
        }
    }
```

این کلاس به صورت خودکار توسط EF از اسمبلی جاری استخراج شده و استفاده خواهد شد. بنابراین نیازی نیست جایی معرفی شود. فقط باید در کدها حضور داشته باشد. همچنین ذکر System.Data.SqlClient نیز ضروری است؛ از این جهت که خطاهای بازگشت داده شده مانند 1205 و امثال آن، در بانک‌های اطلاعاتی مختلف، می‌توانند کاملا متفاوت باشند.