

第44期-使用事务

2019年7月6日 12:21

默认事务行为

默认情况下，如果数据库提供程序支持事务，单个 SaveChanges() 调用中的所有变更都会在一个事务中被提交。如果其中任何一个变更失败了，那么事务就会回滚，没有任何变更会被应用到数据库。这意味着 SaveChanges() 能够确保要么成功保存，要么在发生错误时不对数据库做任何修改。

示例代码: /TransactionSample/DefaultTransaction/Sample.cs

```
using (var context = new BloggingContext())
{
    context.Blogs.Add(new Blog { Url = "https://www.xcode.me/dotnet" });

    context.Blogs.Add(new Blog { Url = "https://www.xcode.me/visualstudio", BlogId = 123 });

    context.SaveChanges();
}
```

以上代码 BlogId 为自增列，不允许设置显式值，所以第二条插入语句将引发异常，这将导致整个事务回滚，第一条添加数据虽然成功，但事务回滚，数据库无任何数据添加成功。

```
context.Database.AutoTransactionsEnabled = false;
```

使用微软 SQL Server Profiler 工具监控事务。

对于大部分应用程序来说，默认的事务行为已经够用了。只有在应用程序需求认为有必要时你才需要手动去控制事务。

控制事务（相同DbContext上下文）

示例代码: /TransactionSample/ControllingTransaction/Sample.cs

```
using (var context = new BloggingContext())
{
    using (var transaction = context.Database.BeginTransaction())
    {
        try
        {
            context.Blogs.Add(new Blog { Url = "https://www.xcode.me/dotnet" });
            context.SaveChanges();

            context.Blogs.Add(new Blog { Url = "https://www.xcode.me/visualstudio" });
            context.SaveChanges();

            context.Database.ExecuteSqlCommand("INSERT INTO [Blogs]([Url]) VALUES({0})",
                "https://www.xcode.me/efcore");

            // Commit transaction if all commands succeed, transaction will auto-rollback
            // when disposed if either commands fails
            transaction.Commit();
        }
    }
}
```

```

        catch (Exception)
        {
            // TODO: Handle failure
        }
    }
}

```

以上三条添加语句共享同一个事务，最后使用 `transaction.Commit()` 统一提交，三条全部执行成功，则影响到数据库，如果任何一个命令失败，则在事务被回收 (`Dispose`) 时会自动回滚，对数据库无影响。

控制事务（跨DbContext上下文）

示例代码: `/TransactionSample/SharingTransaction/Sample.cs`

共享 `DbConnection` 要求能够在构造上下文实例时传入链接对象。实现外部提供 `DbConnection` 的最简单方式是避免使用 `DbContext.OnConfiguring` 方法来配置上下文，并且在其外部创建 `DbConetxtOptions`，然后将其传递给上下文类型的构造方法。

```

public class BloggingContext : DbContext
{
    public BloggingContext(DbContextOptions<BloggingContext> options) : base(options)
    { }
    public DbSet<Blog> Blogs { get; set; }
}

```

另一种替代方案是仍然使用 `DbContext.OnConfiguring`，但是接受并保存一个 `DbConnection`，然后在 `DbContext.OnConfiguring` 中使用它。

```

public class BloggingContext2 : DbContext
{
    private DbConnection _connection;
    public BloggingContext2(DbConnection connection)
    {
        _connection = connection;
    }
    public DbSet<Blog> Blogs { get; set; }
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(_connection);
    }
}

```

现在可以创建共享同一个连接的多个上下文了。之后可以使用 `DbContext.Database.UseTransaction(DbTransaction)` API 来在一个事务中收集这些上下文。

```

var options = new DbContextOptionsBuilder<BloggingContext>().UseSqlServer(new
SqlConnection(connectionString)).Options;

using (var context1 = new BloggingContext(options))
{
    using (var transaction = context1.Database.BeginTransaction())
    {
        try
        {
            context1.Blogs.Add(new Blog { Url = "https://www.xcode.me/dotnet" });
            context1.SaveChanges();
        }
    }
}

```

```

using (var context2 = new BloggingContext(options))
{
    context2.Database.UseTransaction(transaction.GetDbTransaction());

    context2.Database.ExecuteSqlCommand("INSERT INTO [Blogs] ([Url]) VALUES ({0})",
"https://www.xcode.me/efcore");

    context1.Blogs.Add(new Blog { Url = "https://www.xcode.me/aspnetcore" });
    context1.SaveChanges();
}

// Commit transaction if all commands succeed, transaction will auto-rollback
// when disposed if either commands fails
transaction.Commit();
}
catch (Exception)
{
    // TODO: Handle failure
}
}
}

```

使用外部 DbTransactions 事务

如果你正在使用多个数据访问技术来访问关系数据库，那么你可能会想要在这些不同技术执行的操作中共享事务。

以下代码样例显示了如何在同一个事务中执行一个 ADO.NET SqlClient 操作和一个 Entity Framework Core 操作。

示例代码: */TransactionSample/ExternalDbTransaction/Sample.cs*

```

using (var connection = new SqlConnection(connectionString))
{
    connection.Open();

    using (var transaction = connection.BeginTransaction())
    {
        try
        {
            // Run raw ADO.NET command in the transaction
            var command = connection.CreateCommand();
            command.Transaction = transaction;
            command.CommandText = "DELETE FROM dbo.Blogs";
            command.ExecuteNonQuery();

            // Run an EF Core command in the transaction
            var options = new DbContextOptionsBuilder<BloggingContext>
                ().UseSqlServer(connection).Options;

            using (var context = new BloggingContext(options))
            {
                context.Database.UseTransaction(transaction);
                context.Blogs.Add(new Blog { Url = "https://www.xcode.me/dotnet" });
                context.SaveChanges();
            }

            // Commit transaction if all commands succeed, transaction will auto-rollback
            // when disposed if either commands fails
            transaction.Commit();
        }
    }
}

```

```

    }
    catch (Exception)
    {
        // TODO: Handle failure
    }
}

```

使用 System.Transactions 环境事务

如果需要跨较大作用域进行协调，则可以使用环境事务。

示例代码: /TransactionSample/AmbientTransaction/Sample.cs

```

using (var scope = new TransactionScope(TransactionScopeOption.Required, new TransactionOptions { IsolationLevel = IsolationLevel.ReadCommitted }))
{
    using (var connection = new SqlConnection(connectionString))
    {
        connection.Open();

        try
        {
            // Run raw ADO.NET command in the transaction
            var command = connection.CreateCommand();
            command.CommandText = "DELETE FROM dbo.Blogs";
            command.ExecuteNonQuery();

            // Run an EF Core command in the transaction
            var options = new DbContextOptionsBuilder<BloggingContext>
                ().UseSqlServer(connection).Options;

            using (var context = new BloggingContext(options))
            {
                context.Blogs.Add(new Blog { Url = "https://www.xcode.me/dotnet" });
                context.SaveChanges();
            }

            // Commit transaction if all commands succeed, transaction will auto-rollback
            // when disposed if either commands fails
            scope.Complete();
        }
        catch (Exception)
        {
            // TODO: Handle failure
        }
    }
}

```

示例代码: /TransactionSample/CommitableTransaction/Sample.cs

自版本 2.1 起，.NET Core 中的 System.Transactions 实现将不包括对分布式事务的支持，因此不能使用 TransactionScope 或 CommittableTransaction 来跨多个资源管理器协调事务。

主要分布式事务需要依赖于 Windows 系统的 MSDTC 服务，但 .NET Core 要实现跨平台，基于跨平台的分布式事务没有统一的标准，后续版希望改进。