

第12期-数据访问技术方案选型

2019年12月6日 09:31

对于几乎任何软件应用程序，数据访问都是重要的部分，ASP.NET Core 支持各种数据访问技术，包括 Entity Framework Core（以及 Entity Framework 6），并可使用任何 .NET 数据访问框架，比如 ADO.NET 原生对象，选择使用哪种数据访问框架，具体取决于应用程序的需求。对基础设施仓储的抽象，并在基础设施中实现这种抽象，有助于生成松散耦合的可测试软件。

关系型数据库与非关系型数据库

非关系型数据库和关系型数据库区别，优势比较？

结构化、半结构化和非结构化数据

Entity Framework Core（适用于关系数据库）

EF Core 是一种支持 .NET 开发人员将对象保存到数据源或从数据源中保存的对象关系映射程序 (O/RM)。它不要求提供开发人员通常需要编写的大部分数据访问代码。与 ASP.NET Core 一样，EF Core 经过完全重新编写以支持模块化跨平台应用程序。将其添加到应用程序作为 NuGet 包，在启动中配置它，并在需要的任何位置通过依赖关系注入请求它。

配置 EF Core，提取和存储数据，提取关联数据，封装数据，**弹性连接**，事务支持。

外部资源（如 SQL 数据库）偶尔可能不可用。如果暂时不可用，应用程序可使用重试逻辑避免引发异常。此方法通常称为连接弹性。可以实现指数退避算法的重试技术，方法是尝试重试，等待时间呈指数级增长，直到达到最大重试次数。该技术接受以下事实：云资源可能出现短时间间歇性不可用情况，导致某些请求失败。

```
// Startup.cs from any ASP.NET Core Web API
public class Startup
{
    public IServiceProvider ConfigureServices(IServiceCollection services)
    {
        //...
        services.AddDbContext<OrderingContext>(options =>
        {
            options.UseSqlServer(Configuration["ConnectionString"],
                sqlServerOptionsAction: sqlOptions =>
                {
                    sqlOptions.EnableRetryOnFailure(
                        maxRetryCount: 5,
                        maxRetryDelay: TimeSpan.FromSeconds(30),

```

```

        errorNumbersToAdd: null));
    });
}

```

选择 EF Core 还是微型 ORM?

尽管对于管理持久性以及在大多数情况下封装应用程序开发者提供的数据库详细信息而言，EF Core 是绝佳选择，但它不是唯一的选择。另一个热门的开源替代选择是一种所谓的微型 ORM，即 Dapper。微型 ORM 是不具有完整功能的轻量级工具，用于将对象映射到数据结构。Dapper 在设计上主要关注性能，而不是完全封装用于检索和更新数据的基础查询。因为它不提取开发人员的 SQL，Dapper“更接近于原型”，使开发人员可以编写要用于给定数据访问操作的确切查询。

EF Core 具有两个重要功能，使其有别于 Dapper，并且增加其性能开销。第一个功能是从 LINQ 表达式转换为 SQL。将缓存这些转换，但即便如此，首次执行它们时仍会产生开销。第二个功能是对实体进行更改跟踪（以便生成高效的更新语句）。通过使用 `AsNotTracking` 扩展，可对特定查询关闭此行为。EF Core 还会生成通常非常高效的 SQL 查询，并且从性能角度看，任何情况下都能完全接受，但如果需要执行对精确查询的精细化控制，也可使用 EF Core 传入自定义 SQL（或执行存储过程）。在这种情况下，Dapper 的性能仍然优于 EF Core，但只有略微优势。Julie Lerman 在其 2016 年 5 月的 MSDN 文章 Dapper、Entity Framework 和混合应用 中展示了部分性能数据。可访问 Dapper 网站，查看各种数据访问方法的其他性能基准数据。

```

// EF Core
private readonly CatalogContext _context;
public async Task<IEnumerable<CatalogType>> GetCatalogTypes()
{
    return await _context.CatalogTypes.ToListAsync();
}

// Dapper
private readonly SqlConnection _conn;
public async Task<IEnumerable<CatalogType>> GetCatalogTypesWithDapper()
{
    return await _conn.QueryAsync<CatalogType>("SELECT * FROM CatalogType");
}

```

因为它提供较少封装，Dapper 要求开发人员深入了解如何存储其数据，如何对其进行高效查询，以及如何编写更多代码来提取它。模型发生更改时，不是单纯地创建新的迁移（另一种 EF Core 功能），并/或在 `DbContext` 的某一位置更新映射信息，而是必须更新受影响的所有查询。这些查询没有编译时间保证，因此它们可能在运行时中断，以应对模型或数据库的更改，增加快速检测错误的难度。Dapper 可提供极快的性能，以补偿这些方面付出的代价。

对于大多数应用程序和几乎所有应用程序的大多数组件而言，EF Core 提供可接受的性能。因此，其开发人员工作效率优势很可能大于性能开销这一点上的劣势。对于可受益于缓存的查询，实际查询执行的时间可能只占很小的百分比，因此可以忽略较小的查询性能差异。

选择 SQL 还是 NoSQL

传统上，SQL Server 等关系数据库占领了持久性数据存储的市场，但它们不是唯一可行的解决方案。NoSQL 数据库（如 MongoDB）可提供用于存储对象的不同方法。另一种选择是序列化整个对象图并存储结果，而不是将对象映射到表格和行。此方法的优点（至少在起初阶段）是简单和高性能。使用 Key 存储单个序列化对象当然比将对象分解为多个表格（其中的关系以及更新和行可能自上次从数据库中检索该对象以来已更改）更简单。同样，从基于密钥的存储中提取和反序列化单个对象通常比复杂联接或多个数据库查询（完全编写关系数据库中的相同对象所需）更快速、更简单。此外，由于缺少锁定、事务或固定的架构，这使得 NoSQL 数据库非常适合在多台计算机中缩放，以支持大型数据集。

但是，NoSQL 数据库（人们通常这么称呼该数据库）也有其缺点。关系数据库利用规范化强制实施一致性并避免数据重复。这可减少数据库的总大小，确保共享数据更新在整个数据库中立即可用。在关系数据库中，“地址”表可能按 ID 引用“国家/地区”表，这样一来，如果国家/地区名称发生更改，地址记录将受益于更新，而无需自行更新。但是，对于 NoSQL 数据库，众多存储对象中的“地址”及其相关的“国家/地区”可能会被序列化。如果对国家/地区或区域名称进行更新，将需要更新所有此类对象，而不是单独的某行。关系数据库还可通过强制实施外键等规则，确保关系完整性。NoSQL 数据库通常不提供此类数据约束。

NoSQL 数据库需要处理的另一复杂性是版本控制。对象的属性发生更改时，可能无法从存储的过去版本进行反序列化。因此，需要更新具有对象的序列化（以前）版本的所有现有对象，才能符合其新架构。从概念上讲，这与关系数据库没有什么差异，架构更改有时需要更新脚本或映射更新。但是，需要修改的条目数量通常多于 NoSQL 方法，因为存在更多的重复数据。

可以在 NoSQL 数据库中存储多个对象版本，而这是固定架构关系数据库通常不支持的功能。但是，在这种情况下，应用程序代码需要考虑以前对象版本的存在，这增加了额外的复杂性。NoSQL 数据库通常不会强制实施 ACID，这意味着它在性能和可伸缩性方面优于关系数据库。它们非常适合特别大型的数据集和对象，不适合规范化表格结构中的存储。根据最适合的情景分别加以利用，单个应用程序能同时获得关系数据库和 NoSQL 数据库带来的好处。

第三方云数据库方案

阿里云、Azure 和腾讯云等。