

第59期-性能优化方案

2019年8月21日 11:32

谈谈性能问题

EF Core 是一种快速而令人满意的 ORM 数据访问框架，但随着 Web 应用程序越来越频繁的访问，性能变得越来越重要，但让人诟病的性能问题一直是很多程序员热聊的话题，实际情况并不是性能差，而是需要我们掌握如何规避陷阱和避开影响性能的坑。

纯手共执行一个 SQL 语句和用 ORM 框架性能基本相当，只要优化得好，性能问题可以忽略，一般大系统的性能问题都是数据库设计问题，与使用何种数据访问技术无直接关系，EF Core 中也可以直接执行 SQL 语句。

使用上下文实例池

```
services.AddDbContext<EmployeeContext>(options => options.UseSqlServer(connection));  
services.AddDbContextPool<EmployeeContext>(options => options.UseSqlServer(connection));
```

如果使用 AddDbContextPool 方法，那么在控制器请求 DbContext 实例时，我们会首先检查池中是否有可用的实例。请求处理完成后，实例的任何状态都将被重置，并且实例本身会返回池中。

从概念上讲，此方法类似于 ADO.NET 连接池的运行原理，并具有节约 DbContext 实例初始化成本的优势。

<https://docs.microsoft.com/zh-cn/ef/core/what-is-new/ef-core-2.0#dbcontext-pooling>

[*Use DbContextPooling to improve the performance: .Net Core 2.1 feature*](#)

显式编译查询

EF 早期版本以及 LINQ to SQL 中已经提供手动或显式编译的查询 API，允许应用程序缓存查询转换，使其可仅被计算一次并执行多次。

尽管 EF Core 通常可基于查询表达式的哈希表示法自动编译和缓存查询，但是使用此机制可绕过哈希计算和缓存查询，允许应用程序通过调用委托来使用已编译查询，从而实现性能小幅提升。

```
// Create an explicitly compiled query  
private static Func<CustomerContext, int, Customer> _customerById =  
    EF.CompileQuery((CustomerContext db, int id) =>  
        db.Customers  
            .Include(c => c.Address)  
            .Single(c => c.Id == id));  
  
// Use the compiled query by invoking it  
using (var db = new CustomerContext())  
{  
    var customer = _customerById(db, 147);  
}
```

```
}
```

测试 EF Core 使用编译查询提高性能

多活动结果集数据库连接复用

多活动结果集 (MARS) 是一项允许对单个连接执行多个批处理的功能。在以前的版本中，在单个连接上一次只能执行一个批处理。使用 MARS 执行多个批处理并不意味着同时执行操作。

在连接字符串中添加：MultipleActiveResultSets=True 即可启用 MARS 特性。

Multiple Active Result Sets (MARS)

如果未启用 MARS 连接复用，多次打开应用程序，在 SQL Server Management Studio 中使用 sp_who 命令会查询当前存在多个活动的连接。

如果启用了 MARS 连接复用，进行上述操作，发现只有一个活动连接访问数据库，因为连接被复用。

优先使用 Async 异步方法

EF Core 中提供了很多形如 xxxAsync 的异步方法，推荐使用这些方法提高吞吐量和性能，减少不必要的延时等待。

批处理语句

```
optionbuilder.UseSqlServer(sConnectionString , b => b.MaxBatchSize(1));
```

Entity Framework Core 批处理语句性能测试

添加索引提高数据查询性能

索引是对数据库表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息。

```
modelBuilder.Entity<Blog>().HasIndex(b => b.Url).IsUnique();  
modelBuilder.Entity<Person>().HasIndex(p => new { p.FirstName, p.LastName });
```

避免 N+1 查询

模型中可使用导航属性来加载相关实体。有三种常见的 O/RM 模式可用于加载关联数据。预先加载表示从数据库中加载关联数据，作为初始查询的一部分。显式加载表示稍后从数据库中显式加载关联数据。延迟加载表示在访问导航属性时，从数据库中以透明方式加载关联数据。

```
var blogs = context.Blogs.Include(blog => blog.Posts).ThenInclude(post => post.Author).ToList();
```

但是，任何技术都有两面性，优势不可能让你占完了，使用 N+1 模式的优点是可以单独缓存部分数据。

跟踪与非跟踪查询

跟踪行为决定了 EF Core 是否将有关实体实例的快照信息保留在其更改跟踪器中。如果已跟踪某个实体，则该实体中检测到的任何更改都会在 SaveChanges() 期间永久保存到数据库。

当决定只查询数据，不更改数据时，非跟踪查询十分有用，非跟踪查询的执行会更快，因为无需为查询实体设置快照跟踪信息。如果不需要更新从数据库中检索到的实体，则应优先使用非跟踪查询。

```
var blogs = context.Blogs.AsNoTracking().ToList();

context.ChangeTracker.QueryTrackingBehavior = QueryTrackingBehavior.NoTracking;
```

关闭 DetectChanges 状态同步

当从数据库进行查询数据时，上下文便捕获了每个实体属性的快照（数据库值，原始值，当前值），当调用 SaveChanges 时，在内部会自动调用 DetectChanges 方法，此方法将扫描上下文中所有实体，并比较当前属性值和存储在快照中的原始属性值，如果被找到的属性值发生了改变，此时EF将会与数据库进行交互，进行数据更新。

会导致自动调用 DetectChanges 方法：Find、Local、Remove、Add、Update、Attach、SaveChanges 和 Entry 等。

但是，自动同步状态会频繁调用，可手动关闭以上方法的自动同步，当数据都修改好后，一次性手动同步。

```
context.ChangeTracker.AutoDetectChangesEnabled = false;
//执行操作后手动同步状态
context.ChangeTracker.DetectChanges();
```

数据缓存

缓存可以减少重复内容的多次生成成本，从而显著提高应用程序的性能和可伸缩性。缓存最适用于不经常更改的数据，生成成本很高的数据。通过缓存，可以比从数据源返回的数据的副本速度快得多。

微软提供：内存缓存和分布式缓存解决方案，详见 ASP.NET Core 系列课程。

EF Core 的高性能二级查询缓存开源库：[EntityFrameworkCore.Cacheable](#)

```
var cacheableQuery = cacheableContext.Books
    .Include(d => d.Pages)
    .ThenInclude(d => d.Lines)
    .Where(d => d.ID == 200)
    .Cacheable(TimeSpan.FromSeconds(60));
```

使用 EF.Functions.Link 模糊查询

EF Core 支持使用 `StartsWith`、`Contains` 和 `EndsWith` 方法进行模糊查询，这些方法被翻译成 LIKE 语句，但为了提高性能也提供了一个 `EF.Functions.Like` 方式，这种方式生成的 SQL 语句性能更优。

```
var result= context.Blogs.Where(b => EF.Functions.Like(b.BlogName, "%xcode%"))
```

Entity Framework Core Like 查询揭秘

DbFunctionAttribute 标量函数

EF Core 支持映射数据库中定义的函数，可以在 LINQ 查询中使用，该功能支持将数据库标量函数映射到方法存根，使其可用于 LINQ 查询并转换为 SQL。

在 `DbContext` 上声明静态方法，并使用 `DbFunctionAttribute` 对其批注：

```
public class BloggingContext : DbContext
{
    [DbFunction]
    public static int PostReadCount(int blogId)
    {
        throw new Exception();
    }
}
```

此类方法会自动注册，对 LINQ 查询方法的调用可转换为 SQL 中的函数调用：

```
var query =context.Posts.Where(p=>BloggingContext.PostReadCount(p.Id) > 5)
```