Lazy Loading of Related Data

Article • 10/12/2021

Lazy loading with proxies

The simplest way to use lazy-loading is by installing the Microsoft.EntityFrameworkCore.Proxies package and enabling it with a call to `UseLazyLoadingProxies`. For example:

C#

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    => optionsBuilder
        .UseLazyLoadingProxies()
        .UseSqlServer(myConnectionString);
```

Or when using AddDbContext:

C#

```
.AddDbContext<BloggingContext>(
    b => b.UseLazyLoadingProxies()
          .UseSqlServer(myConnectionString));
```

EF Core will then enable lazy loading for any navigation property that can be overridden--that is, it must be `virtual` and on a class that can be inherited from. For example, in the following entities, the `Post.Blog` and `Blog.Posts` navigation properties will be lazy-loaded.

C#

```
public class Blog
{
    public int Id { get; set; }
    public string Name { get; set; }

    public virtual ICollection<Post> Posts { get; set; }
}

public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
```

```
    public string Content { get; set; }

    public virtual Blog Blog { get; set; }
}
```

> ⚠ **Warning**
>
> Lazy loading can cause unneeded extra database roundtrips to occur (the so-called N+1 problem), and care should be taken to avoid this. See the **performance section** for more details.

Lazy loading without proxies

Lazy-loading without proxies work by injecting the `ILazyLoader` service into an entity, as described in Entity Type Constructors. For example:

C#

```
public class Blog
{
    private ICollection<Post> _posts;

    public Blog()
    {
    }

    private Blog(ILazyLoader lazyLoader)
    {
        LazyLoader = lazyLoader;
    }

    private ILazyLoader LazyLoader { get; set; }

    public int Id { get; set; }
    public string Name { get; set; }

    public ICollection<Post> Posts
    {
        get => LazyLoader.Load(this, ref _posts);
        set => _posts = value;
    }
}

public class Post
{
```

```
    private Blog _blog;

    public Post()
    {
    }

    private Post(ILazyLoader lazyLoader)
    {
        LazyLoader = lazyLoader;
    }

    private ILazyLoader LazyLoader { get; set; }

    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog
    {
        get => LazyLoader.Load(this, ref _blog);
        set => _blog = value;
    }
}
```

This method doesn't require entity types to be inherited from or navigation properties to be virtual, and allows entity instances created with `new` to lazy-load once attached to a context. However, it requires a reference to the `ILazyLoader` service, which is defined in the Microsoft.EntityFrameworkCore.Abstractions package. This package contains a minimal set of types so that there is little impact in depending on it. However, to completely avoid depending on any EF Core packages in the entity types, it's possible to inject the `ILazyLoader.Load` method as a delegate. For example:

C#

```
public class Blog
{
    private ICollection<Post> _posts;

    public Blog()
    {
    }

    private Blog(Action<object, string> lazyLoader)
    {
        LazyLoader = lazyLoader;
    }
```

```
    private Action<object, string> LazyLoader { get; set; }

    public int Id { get; set; }
    public string Name { get; set; }

    public ICollection<Post> Posts
    {
        get => LazyLoader.Load(this, ref _posts);
        set => _posts = value;
    }
}

public class Post
{
    private Blog _blog;

    public Post()
    {
    }

    private Post(Action<object, string> lazyLoader)
    {
        LazyLoader = lazyLoader;
    }

    private Action<object, string> LazyLoader { get; set; }

    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog
    {
        get => LazyLoader.Load(this, ref _blog);
        set => _blog = value;
    }
}
```

The code above uses a `Load` extension method to make using the delegate a bit cleaner:

C#

```
public static class PocoLoadingExtensions
{
    public static TRelated Load<TRelated>(
        this Action<object, string> loader,
        object entity,
        ref TRelated navigationField,
        [CallerMemberName] string navigationName = null)
        where TRelated : class
```

```
    {
        loader?.Invoke(entity, navigationName);

        return navigationField;
    }
}
```

> **Note**
>
> The constructor parameter for the lazy-loading delegate must be called "lazyLoader". Configuration to use a different name than this is planned for a future release.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback