Include all navigation properties using Reflection in generic repository using EF Core

Asked 2 years, 10 months ago Active 3 months ago Viewed 5k times

8

3

I'm working on creating a generic repository for an EF Core project to avoid having to write CRUD for all models. A major roadblock I've hit is navigation properties not being loaded since Core doesn't yet support lazy loading and the generic class obviously can't define .Include statements for class specific properties.

I'm trying to do something like this for my Get method to include all the properties dynamically:

```
public virtual T Get(Guid itemId, bool eager = false)
        {
            IQueryable<T> querySet = _context.Set<T>();

            if (eager)
            {
                foreach (PropertyInfo p in typeof(T).GetProperties())
                {
                    querySet = querySet.Include(p.Name);
                }
            }

            return querySet.SingleOrDefault(i => i.EntityId == itemId);
        }
```

But it throws an error when including properties that are not navigation properties.

I found this answer which is about the same thing but its for EF 5 and involves methods that are not present in EF core:

EF5 How to get list of navigation properties for a domain object

Is it possible to accomplish the same thing in EF Core?

c# entity-framework-core system.reflection

asked Jun 21 '17 at 17:26
Valuator

1,807 ● 9 ● 29

just a suggestion but instead of having a parameter for eager loading have instead a lambda expression of the properties you want to eagerly load? Parameter something like. Expression<Func<T, object>> includes. – Rob White Jun 21 '17 at 17:37

I suppose that's an option, I wanted to avoid having developers have to specify a bunch of stuff when calling the repo though. – Valuator Jun 21 '17 at 17:40

2 Answers

Active | Oldest | Votes

18

Working with metadata in EF Core is much easier than in previous EF versions. The `DbContext` class provides `Model` property which provides access to

> The metadata about the shape of entities, the relationships between them, and how they map to the database.

The code which does what you ask could be like this:

```
public virtual IQueryable<T> Query(bool eager = false)
{
    var query = _context.Set<T>().AsQueryable();
    if (eager)
    {
        foreach (var property in
_context.Model.FindEntityType(typeof(T)).GetNavigations())
            query = query.Include(property.Name);
    }
    return query;
}

public virtual T Get(Guid itemId, bool eager = false)
{
    return Query(eager).SingleOrDefault(i => i.EntityId == itemId);
}
```

Please note that although this does what you asked for, it's quite limited generic approach since it eager loads only the direct navigation properties of the entity, i.e. does not handle loading nested navigation properties with `ThenInclude`.

answered Jun 21 '17 at 18:20

Ivan Stoev

IS 135k ● 8 ● 141 ● 195

based on your answer, on this part `return Query(eager).SingleOrDefault(i => i.EntityId == itemId);`. What is `EntityId` referring to? – warheat1990 Jun 17 '18 at 9:38

@warheat1990 This is coming from OP code. I'm assuming there is some generic constraint on `T`, like base class or interface having `Guid EntityId { get; }` property. – Ivan Stoev Jun 17 '18 at 11:04

This solution was working earlier. After migrating to EF Core 2.1.3. I get the following exception "One or more errors occurred. (No coercion operator is defined between types 'Microsoft.EntityFrameworkCore.Query.Internal.AnonymousObject' and 'YourEntity'.)" Any thoughts? – Rakesh Sep 22 '18 at 15:07

@Rakesh It still works for me in 2.1.3. You can post a question with repro. – Ivan Stoev Sep 22 '18 at 15:49

-1

```
    private List<PropertyInfo> GetNavigationProperties<T>(GesFormaContext
_gesFormaContext)
    {
        List<PropertyInfo> propertyInfos = new List<PropertyInfo>();
        _gesFormaContext.Model.GetEntityTypes().Select(x =>
x.GetNavigations()).ToList().ForEach(entityTypes =>
        {
            entityTypes.ToList().ForEach(property =>
            {
                propertyInfos.AddRange(typeof(T).GetProperties().Where(x =>
x.PropertyType == property.PropertyInfo.PropertyType).ToList());
            });

        });


        return propertyInfos;
    }
```

answered Jan 17 at 14:54

Majdi Siala
1

1 While this code may accurately answer the question, it's best to include an explanation with it so other users can familiarize. – Christine Jan 17 at 17:00