POCO Proxy and Lazy Loading

Patrick Desjardins C#, Entity Framework Oct, 05, 2011 No Comments

POCO objects using Entity Framework as ORM require the creation of a proxy. When the proxy is created, the rest is exactly the same as if you were using standard entity from the object context.

Creating the proxy

In fact, the proxy will be generated by the Entity Framework (in runtime). To be more accurate, each of the POCO classes have a proxy. This proxy will derive from your POCO classes. This will let the Entity Framework keeping track of the state change and enable the use of lazy loading. Since the proxy class derive from POCO classes, these one must not be sealed, private or abstract.

We have said that the proxy is created in the runtime and this is a good thing because we can enable and disable the proxy. If you desire to enable the proxy, you must use

Search

Search Find

Categories

- Ado.Net
  - Dapper.Net
  - Entity Framework
  - Linq To Sql
- ASP
  - ASP.MVC
  - ASP.Net
  - Identity
  - Telerik Asp.Net MVC
  - VNext
- Best Practices
  - UI
- C#
  - Async

the *ProxyCreationEnabled* to true. This property reside inside the context object, inside the context option.

```
var db = new NorthWindContext();
db.ContextOptions.ProxyCreationEnabled = true; //Enable the proxy creati
```

This is not always enabled because in some case, like while using serialization with WCF, only *[known]* class can be serialized. This won't be the case of the runtime generated proxy.

From here, you need to do some changes in your POCO class and the change varies depending if you want only the lazy loading or also the change tracking.

Lazy Loading

To have lazy loading enable the navigation properties (the one that link to an other object that need to be loaded or a collection to be loaded) must be public and virtual and not sealed. This way, it's possible for the proxy to change some call to add the lazy loaded statement.

Change Tracking

The first step is to make you POCO class legit for lazy loading. So, all information in the previous paragraph must remain true. Each collection of object must return a type that derive from a generic ICollection. It's also require to use the CreateObject method instead of using new to create your class.

```
var db = new NorthWindContext();
var myPoco = db.CreateObject<MyPocoObject>();
```

You can fine further information on MSDN.

Complete example

Let's make this theory in practice. We are gonna use the Northwind database and the table Customers and Orders.

Preparation

Lets create a new ADO.NET entity data model and use the generator creating the model for us. Do not forget to remove the Custom Tool text on the Edmx file.

After that, lets create the 2 POCO classes.

```
namespace PocoAndLazy.POCO
{
    public class Customer
    {
        public string CustomerID { get; set; }
        public string CompanyName { get; set; }
        public string ContactName { get; set; }
        public string ContactTitle { get; set; }
        public string Address { get; set; }
        public string City { get; set; }
        public string Region { get; set; }
        public string PostalCode { get; set; }
        public string Country { get; set; }
        public string Phone { get; set; }
        public string Fax { get; set; }

        public virtual List<Order> Orders { get; set; } //Virtual + ICo
    }
}
```

and

```
namespace PocoAndLazy.POCO
{
    public class Order
    {
        public int OrderID { get; set; }
        public string CustomerID { get; set; }
        public int EmployeeID { get; set; }
        public DateTime? OrderDate { get; set; }
        public DateTime? RequiredDate { get; set; }
        public DateTime? ShippedDate { get; set; }
        public int? ShipVia { get; set; }
        public decimal? Freight { get; set; }
        public string ShipName { get; set; }
        public string ShipAddress { get; set; }
        public string ShipCity { get; set; }
        public string ShipRegion { get; set; }
        public string ShipCountry { get; set; }
        public string ShipPostalCode { get; set; }
    }
}
```

After the ObjectContext class.

```
namespace PocoAndLazy
{
    public class ModelContext : ObjectContext
    {
```

- NoSql
- Redis
- Web
  - Angular
  - BootStrap
  - CSS
  - Flow
  - Gulp
  - Javascript
  - Jest
  - Nodejs
  - Owin
  - React
  - React-Hooks
  - Redux
  - TypeScript
  - Webpack
- WebServices
  - GraphQL
  - OData
  - WCF
  - WebApi

Archives

- December 2019
- June 2019
- May 2019
- April 2019

```
        private ObjectSet<Customer> customers;
        private ObjectSet<Order> orders;

        public ModelContext(): base("name=NorthwindEntities", "Northwin
        {
            customers = CreateObjectSet<Customer>();
            orders = CreateObjectSet<Order>();
        }

        public ObjectSet<Customer> Customers
        {
            get
            {
                return customers;
            }
        }

        public ObjectSet<Order> Order
        {
            get
            {
                return orders;
            }
        }

    }
}
```

And lets do a quick test.

```
ModelContext db = new ModelContext();
var bigCustomers = db.Customers.Where(c => c.Orders.Count > 20);
foreach (var customer in bigCustomers)
{
    Debug.WriteLine("Customer#" + customer.CustomerID);
}
```

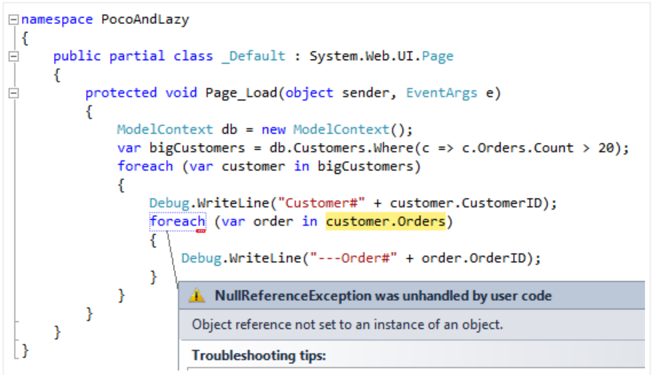
Output:

```
Customer#ERNSH
Customer#QUICK
Customer#SAVEA
```

This display the list of customer that have over 10 orders. I have not given the explication of how to create POCO objets with Entity Framework here because this is covered in an other article. The important information is that we have now a stable structure to continue to the core of the goal : lazy loading.

Currently, no order can be show if we loop through the list of order of each of these 3 clients. The reason is the default value is eager loading the use of *Include* is missing and no explicit loading with *Load* is provided.

```
ModelContext db = new ModelContext();
var bigCustomers = db.Customers.Where(c => c.Orders.Count > 20);
foreach (var customer in bigCustomers)
{
  Debug.WriteLine("Customer#" + customer.CustomerID);
  foreach (var order in customer.Orders)
  {
    Debug.WriteLine("---Order#" + order.OrderID);
  }
}
```



```
namespace PocoAndLazy
{
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            ModelContext db = new ModelContext();
            var bigCustomers = db.Customers.Where(c => c.Orders.Count > 20);
            foreach (var customer in bigCustomers)
            {
                Debug.WriteLine("Customer#" + customer.CustomerID);
                foreach (var order in customer.Orders)
                {
                    Debug.WriteLine("---Order#" + order.OrderID);
                }
            }
        }
    }
}
```

NullReferenceException was unhandled by user code

Object reference not set to an instance of an object.

Troubleshooting tips:

POCO Eager Loading

Of course we can add in the constructor of the Customer the initialization of the Orders collection.

```
public Customer()
{
    Orders = new List<Order>();
}
```

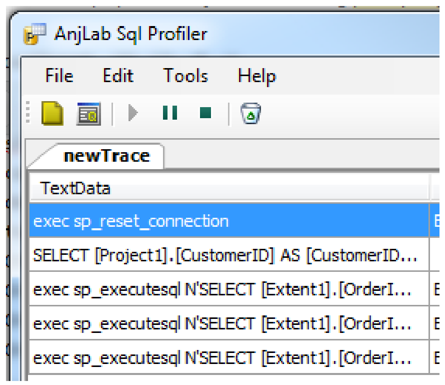
But, you and me understand that it still does not load the list of orders. Let for fun just enable the Lazy Loading.

```
ModelContext db = new ModelContext();
db.ContextOptions.LazyLoadingEnabled = true;
var bigCustomers = db.Customers.Where(c => c.Orders.Count > 20);
foreach (var customer in bigCustomers)
{
  Debug.WriteLine("Customer#" + customer.CustomerID);
  foreach (var order in customer.Orders)
  {
    Debug.WriteLine("---Order#" + order.OrderID);
  }
}
```

We can not see in the output:

```
Customer#ERNSH
---Order#10258
---Order#10263
---Order#10351
---Order#10368
---...
Customer#QUICK
---Order#10273
---Order#10285
---Order#10286
---...
Customer#SAVEA
---Order#10324
---Order#10393
---Order#10398
--- ...
```

If we check the SQL profiler we see N+1 call to the database (1 to get all customers and 3 to gets each of their orders).



```
exec sp_reset_connection
SELECT [Project1].[CustomerID] AS [CustomerID...
exec sp_executesql N'SELECT [Extent1].[OrderI...
exec sp_executesql N'SELECT [Extent1].[OrderI...
exec sp_executesql N'SELECT [Extent1].[OrderI...
```

SQL profiler showing the 1+3 call.

With the use of eager loading, a single query is done.

```
protected void Page_Load(object sender, EventArgs e)
{
  ModelContext db = new ModelContext();
  db.ContextOptions.LazyLoadingEnabled = false;
  var bigCustomers = db.Customers.Include("Orders").Where(c => c.Orders
  foreach (var customer in bigCustomers)
  {
    Debug.WriteLine("Customer#" + customer.CustomerID);
    foreach (var order in customer.Orders)
```

- April 2014
- March 2014
- February 2014
- January 2014
- December 2013
- November 2013
- October 2013
- September 2013
- August 2013
- July 2013
- June 2013
- May 2013
- April 2013
- March 2013
- February 2013
- January 2013
- December 2012
- November 2012
- October 2012
- September 2012
- August 2012
- July 2012
- June 2012
- May 2012
- April 2012
- March 2012
- February 2012
- January 2012
- December 2011
- November 2011

```
    {
      Debug.WriteLine("---Order#" + order.OrderID);
    }
  }
}
```

- October 2011
- September 2011
- August 2011

```sql
SELECT
[Project2].[C1] AS [C1],
[Project2].[CustomerID] AS [CustomerID],
[Project2].[CompanyName] AS [CompanyName],
[Project2].[ContactName] AS [ContactName],
...
FROM ( SELECT
  [Project1].[CustomerID] AS [CustomerID],
  [Project1].[CompanyName] AS [CompanyName],
  [Project1].[ContactName] AS [ContactName],
  [Project1].[ContactTitle] AS [ContactTitle],
...
  1 AS [C1],
  [Extent3].[OrderID] AS [OrderID],
  [Extent3].[CustomerID] AS [CustomerID1],
  [Extent3].[EmployeeID] AS [EmployeeID],
  [Extent3].[OrderDate] AS [OrderDate],
  [Extent3].[RequiredDate] AS [RequiredDate],
  [Extent3].[ShippedDate] AS [ShippedDate],
...
  FROM   (SELECT
    [Extent1].[CustomerID] AS [CustomerID],
    [Extent1].[CompanyName] AS [CompanyName],
    [Extent1].[ContactName] AS [ContactName],
    [Extent1].[ContactTitle] AS [ContactTitle],
...
    (SELECT
      COUNT(1) AS [A1]
      FROM [dbo].[Orders] AS [Extent2]
      WHERE [Extent1].[CustomerID] = [Extent2].[CustomerID]) AS [C1]
    FROM [dbo].[Customers] AS [Extent1] ) AS [Project1]
  LEFT OUTER JOIN [dbo].[Orders] AS [Extent3] ON [Project1].[CustomerID
  WHERE [Project1].[C1] > 20
)  AS [Project2]
ORDER BY [Project2].[CustomerID] ASC, [Project2].[C2] ASC
```

So without Lazy Loading, nothing is shown until explicit load is called, with Lazy loading N+1 query is done to the database and with Eager loading a single query is done to the database.

If you like my article, think to buy my annual book, professionally edited by a proofreader. directly from me or on Amazon. I also wrote a TypeScript book called Holistic TypeScript

.Net Knowledge Book Volume 6
TypeScript, React and Redux
Patrick Desjardins
Holistic TypeScript
Patrick Desjardins
First Edition
TypeScript 3.0 Quick Start Guide
Packt

« Entity Framework and the Connection String

Decimal literal and Float literal »

Leave a Reply

Comment

Name

Email

Website

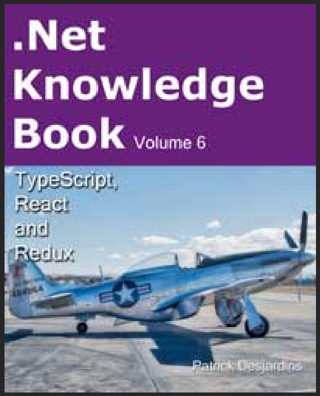
Post Comment

This site uses Akismet to reduce spam. Learn how your comment data is processed.

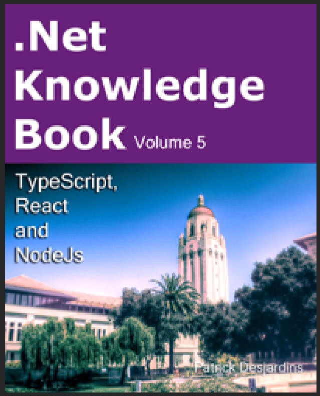
Disclamer

As of the date of 2017-07-08, Patrick Desjardins has been employee by Netflix. Before, from 2014-08-01 to 2017-07-08, employee by Microsoft Corporation. The views expressed in this blog are those of the author, Mr. Desjardins, and do not necessarily reflect those of Netflix or Microsoft Corporation.

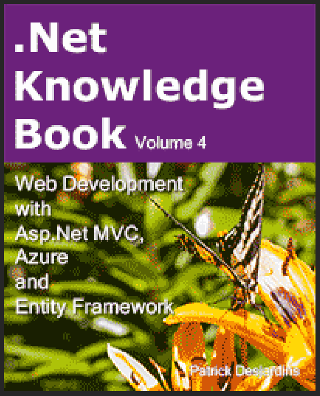
2018 Book



2017 Book



2016 Book


.Net Knowledge Book Volume 4
Web Development with Asp.Net MVC, Azure and Entity Framework
Patrick Desjardins