Edit on 14:59 25 December 2020

When I configuring the basic poco data models using EntityFramework Core, I came across some problems, I set a BaseDao to be inherited by other models, the BaseDao model contains the navigation properties that refer to the user who created and deleted them. I set these properties carefully and completely by using both guid and navigation property, and decorate them with required on creatorId and ForeignKey data annotation, but things came out not worked as well as I imagined, the cascade problems becomes a big issue, I need to dig deep to understand and solve them. This issue points me the explicit direction : I have weakness on both EntityFramework Core and Database Infrastructure. I need to get it done well.

Edit on 8:01 26 December 2020

I’m starting use the dao models. But here reveals another problems, I want the dbcontext layer be transparent to the service layer, which means the service layer can easily get or set the data in database just by using the dao classes with out explicitly access the dbcontext. When do things like this, I start to considering the concrete processing flow of Asp.net, I wander whether the requests are processed each by each in a single thread or they are separated in multiple thread, and which parameters are reused and which are not. By refereeing to the company’s project code, the TheradLocal class starts to jump into my vision. To further progress my project, I need to learn the processing flow of Asp.net the how to handle the threads elegantly. This website seems fit my requirements :

<https://www.codemag.com/article/0511061/A-Low-Level-Look-at-ASP.NET-Architecture#:~:text=ASP.NET%20is%20a%20request,HTTP%20or%20the%20Web%20server>.

Edit on 7:59 27 December 2020

When I tweaking the user model, I add the lastLoginTime, registerTime and deactivatedTime to the model. After I configured all the things well and start to run the server, something weird happened. The Create and Delete function are ok, but only the update function sometimes is ok, but the rest time always do not hit my target. After the long time bug shooting, I noted that the EntityFramework does not automatically update the entity even though you have modified the IQueryable<T> data by hand, you still need to call the context.DbSet<T>.Update function to apply those changes. Why sometimes the update function is ok is that in my service, all operations are executed in one transaction, and some operation after that update calls the update function which include the changes did before. So that makes it more weirder.

Something more bad happened, my authorization dead again for no reason. I have to figure it out.

Edit on 4:55 28 December 2020

Here comes another head-aching problem: there is a base class and a subclass, both used the Autofac property injection. One property injected in base class and one property injected in subclass. My controller use the subclass’s property to finished some function, but when it comes to some common function, the subclass will call the base class’s injected property to finish it’s work. These thing works well in subclass, the property gets successfully injected. But in the base class, the property was not injected as I will. Those property are types that registered in the startUp by using RegisterAssemblyTypes, and both are contains with Impl for convenience. Also, these two property are generic typed. I don’t know for now how to solve this problem.

Edit on 12:52 29 December 2020

Finally the problem have figured out. There has been a problem that changes made in set<T>() not affect the database for a while. After a manually add another call to context.SaveChanges() in the manager layer, everything starts to works fine. The issue here is that the context used in manager layer isn’t the same object used in serviceInterceptor, which caused the changes made in the first context will not be tracked by the one used in ServiceInterceptor, which means the context.SaveChanges and transaction.Commit in ServiceInterceptor actually does nothing. The main reason why this issue emerges is located on the DbContextHolder class. I’ll check that later.

Edit on 2:19 30 December 2020

The problem issued before has been solved. The process how I located the problem and solved that is very interesting. I have been doubt that the context used in manager aren’t the same got called in serviceInterceptor for a while. First of all, I think that is because of the manager and service has their own dbcontext which makes them different. But after log out the thread.ManagedThreadId, I found that they are actually the same. That’s really got me. But after a while, I thought why I didn’t just directly compare the two context used in manager and interceptor? So I add an extra property called Tag in DbContext I made and give it a random integer. And then log them out to check. The reason emerged. After the first use of the manager, the context in baseManager are persist. No matter how the thread change between different thread. The context hold by the baseManager stays the same with the one it firstly used. The reason caused that is I directly reference the DbContextHolder.Context by using a property in the baseManager, the right way to do that is to set an expression body in that property to dynamically get the DbContextHolder.Context.