Unit of Work

Unit of Work (UoW) manages many data change persistence operations within single business transaction. The operation could be occurred in different class, service, function, method and route. but all those operations must be in one logical call context with in same business transaction. In the end of the UoW it must perform commit or rollback action to ensure all data changes are committed persist or roll back completely.

# UoW requirement:

1. Tracking data objects affected by a business transaction.

2. Coordinates data persistence component for writing out data change.

3. Persistence ignorance.

# UoW types

1. Nested Unit of work

A nested UoW is a new unit of work initiated inside another UoW. Changes made within a nested UoW will be merged with the parent UoW. Each nested UoW must perform commit or rollback. The data change persistent only occurred in topmost UoW when all nested UoWs committed. Any nested UoW rollback will cause entire UoW rollback.

**Nested Unit of Work**

B3

UoW Begin

UoW End

Repository 31

Repository 32

UoW provider 1 /DbContext1

UoW provider 1 /DbContext2

UoW provider 2 /DbContext3

UoW Begin

UoW End

Repository 11

Repository 12

Repository 21

Repository 22

B1

Repository 13

B2

Repository 41

Repository 42

2. Isolated Unit of work

An isolated UoW is a new unit of work initiated inside another UoW. Changes made within isolated UoW will be persisted when the UoW committed. The change can be merged into parent UoW. The rollback and commit in isolated UoW will not affect to parent UoW transaction.

**Isolated Unit of Work**

UoW provider 1 /DbContext1

UoW provider 1 /DbContext2

UoW provider 2 /DbContext3

B3

UoW Begin

UoW End

Repository 31

Repository 32

UoW Begin

UoW End

Repository 11

Repository 12

Repository 21

Repository 22

B1

Repository 13

requireNew = true

3. Parallel Unit of Work

Multiple Units of Work operates in parallel (in different call context) and each UoW maintains own state change. The repositories without UoW will not join any UoW.

UoW provider 1 /DbContext1

UoW provider 1 /DbContext2

UoW provider 2 /DbContext3

Main block

B3

Repository 41

Repository 42

UoW Begin

UoW End

Repository 11

Repository 12

Repository 21

Repository 22

B1

Repository 13

B2

UoW Begin

UoW End

Repository 31

Repository 32

4. UoW per request

A simple UoW created for each request (Web request, Service request or message request) and completed after request completed.

a. Create an UoW onRequestBegin and bind UoW to request context

b. Apply UoW to persistance component

c. Complete the UoW onRequestEnd and unbind UoW from request context

requireNew = false

UoW provider 1 /DbContext1

UoW provider 1 /DbContext2

UoW provider 2 /DbContext3

UoW End

Repository 11

Repository 12

Repository 21

Repository 22

Repository 13

Bx1

UoW

UoW End

Repository 31

Bx2

UoW

UoW End

Repository 31

UoW Begin

requireNew = true

# UoW, Interface, Provider and Persistence Ignorance:

Unit of Work is a Persistence Ignorance component, but underline UoW implementation always be persistence specific. To decouple application UoW with actual persistence mechanism IoC is required to wire UoW implementation with interface.

The IUnitOfWork provides "Complete()" method which indicates a business transaction need be committed. The transaction will be rollback without "Complete()" in end of scope of UoW.

public interface IUnitOfWork:IDisposable

{

void Complete();

}

An application may have more than one IUnitOfWork implementation components for different persistence mechanism. Each IUnitOfWork component also implemented IUnitOfWorkProvider interface to generate UoW instance.

public interface IUnitOfWorkProvider:IProvider

{

IUnitOfWork Create(bool requireNew);

}

Below are some sample UoM implementation classes:

Sql DBClient UoW implementation

public class DbUnitOfWork:IUnitOfWork, IUnitOfWorkProvider

{

public DbUnitOfWork();

public DbUnitOfWork(bool requireNew);

public void Complete();

public void Dispose();

public IUnitOfWork Create(bool requireNew)

{

return new DbUnitOfWork(requireNew);

}

}

NHibernate UoW implementation

public class NHibernateUnitOfWork:IUnitOfWork, IUnitOfWorkProvider

{

public NHibernateUnitOfWork();

public NHibernateUnitOfWork(bool requireNew);

public void Complete();

public void Dispose();

public IUnitOfWork Create(bool requireNew)

{

return new NHibernateUnitOfWork(requireNew);

}

}

Entity Framework UoW implementation

public class EFUnitOfWork:IUnitOfWork, IUnitOfWorkProvider

{

public EFUnitOfWork();

public EFUnitOfWork(bool requireNew);

public void Complete();

public void Dispose();

public IUnitOfWork Create(bool requireNew)

{

return new EFUnitOfWork(requireNew);

}

}

The UnitOfWork class provides unified persistence ignorance UoW for all the UoW implementations.

public class UnitOfWork:IDisposable

{

public UnitOfWork(bool requireNew = false);

public void Complete();

}

The UnitOfWork instance will wire up all UoW implementation components through IoC in scope of UoW.

(Note: If the application UoW contains more than one UoW instances with different UoW provider. The UoW transaction completion may not be guaranteed if the failure occurred in one of persistence transaction commit operation.

)

# Multiple UoW Providers and Persistence transactions

UoW provider is a unit of work implementation for a specific type of persistence store (such as ADO.NET database, EF DbContext, NHibernate session or others). An UoW could contain many UoW instances in different UoW providers. Same provider may mange multiple persistence stores (databases) in different persistence transaction.

The UoW guarantee all persistence operations (could be in different persistence implementation) are under single unit of work. One of the persistence failure cause entire UoW rollback. All successful persistence operations commit the UoW completion.

There is one exception that UoW cannot guarantee all transactions are committed in single UoW if one of the transaction failures occurred in the moment of transaction operation.

**Unit of Work with multiple providers**

B1

UoW Begin

Repository 11

B2

Repository 41

Repository 42

Repository 12

X

Complete()

Committed

Committed(?)

Committed

UoW End

Repository 13

Repository 22

Repository 21

UoW provider 2 /DbContext3

UoW provider 1 /DbContext2

UoW provider 1 /DbContext1

B3

UoW Begin

UoW End

Repository 31

Repository 32

# UoW and Multi-Threads and Call Context

UoW perform a single transaction in one unit of call context (except “requireNew” child UoW). The call context is the path of the code calling from begin of the unit of work to end of the unit of work. If the UoW call context go through multi-threads the repository and uow under child thread will be automatically merged into parent uow in same call context. It is very useful when the main operation contains many async await task operations (see case 1). Those async await operations will be merged into parent UoW.

But in other side, some operation unwanted be in UoW also will be join the UoW automatically. To avoid this exception, application code need take care those persistence operations to wrap them into a “requireNew” UoW or using a non-UoW supported repository operation. (See case 2)

The UoW is not thread safe. The UoW is a call context specific. The repository evaluate UoW call context in the moment of repository operation. If a child thread is instantiated earlier than UoW and its repository operation performing after main thread UoW starting it could cause unwanted operation joins to the uow automatically (See case 3).

If the child thread persistence operation doesn’t want join UoW, create a “requestNew” UoW to perform isolated transaction in child thread.

Unexpected exception may raise in child thread if UoW completed earlier before persistence operation in child thread.

Try to scope UoW smaller to reduce the issue in multi-threads environment.

The framework use ContextManager to provide UoW CallContext management. You also can use “StaticThread” callcontext to provide isolated uow for different thread.

Case 1:

using (var unitwork = new UnitOfWork())

{

var task1 = AddDataAsync();

var task2 = UpdateDataAsync();

var task3 = DeleteDataAsync();

await task.WhenAll(task1,task2,task2);

unitwork.Complete();

}

Case 2:

using (var unitwork = new UnitOfWork())

{

...

LogAction(“Create AddDataAsync Task”);

await AddDataAsync();

LogAction(“Completed AddDataAsync”);

unitwork.Complete();

}

void LogAction(string text) {

using (var unitwork = new UnitOfWork(true))

{

...

LogRepository.Save (text);

unitwork.Complete();

}

}

Case 3:

NewThreadOperation ();

using (var unitwork = new UnitOfWork())

{

var task1 = AddDataAsync();

var task2 = UpdateDataAsync();

var task3 = DeleteDataAsync();

await task.WhenAll(task1,task2,task2);

unitwork.Complete();

}