本章还是继续来完善下仓储的设计。

## **一、对仓储接口以及实现基类的完善**

### **1、仓储实现基类的所有方法加上virtual关键字，方便具体的仓储在特定需求的时候override基类的方法。**

　　//仓储的泛型实现类

public class EFBaseRepository<TEntity> : IRepository<TEntity> where TEntity : AggregateRoot

{

[Import(typeof(IEFUnitOfWork))]

public IEFUnitOfWork UnitOfWork { get; set; }

public EFBaseRepository()

{

Regisgter.regisgter().ComposeParts(this);

}

public virtual IQueryable<TEntity> Entities

{

get { return UnitOfWork.context.Set<TEntity>(); }

}

public virtual TEntity GetByKey(object key)

{

return UnitOfWork.context.Set<TEntity>().Find(key);

}

public virtual IQueryable<TEntity> Find(Expression<Func<TEntity, bool>> express)

{

Func<TEntity, bool> lamada = express.Compile();

return UnitOfWork.context.Set<TEntity>().Where(lamada).AsQueryable<TEntity>();

}

public virtual int Insert(TEntity entity)

{

UnitOfWork.RegisterNew(entity);

return UnitOfWork.Commit();

}

public virtual int Insert(IEnumerable<TEntity> entities)

{

foreach (var obj in entities)

{

UnitOfWork.RegisterNew(obj);

}

return UnitOfWork.Commit();

}

public virtual int Delete(object id)

{

var obj = UnitOfWork.context.Set<TEntity>().Find(id);

if (obj == null)

{

return 0;

}

UnitOfWork.RegisterDeleted(obj);

return UnitOfWork.Commit();

}

public virtual int Delete(TEntity entity)

{

UnitOfWork.RegisterDeleted(entity);

return UnitOfWork.Commit();

}

public virtual int Delete(IEnumerable<TEntity> entities)

{

foreach (var entity in entities)

{

UnitOfWork.RegisterDeleted(entity);

}

return UnitOfWork.Commit();

}

public virtual int Delete(Expression<Func<TEntity, bool>> express)

{

Func<TEntity, bool> lamada = express.Compile();

var lstEntity = UnitOfWork.context.Set<TEntity>().Where(lamada);

foreach (var entity in lstEntity)

{

UnitOfWork.RegisterDeleted(entity);

}

return UnitOfWork.Commit();

}

public virtual int Update(TEntity entity)

{

UnitOfWork.RegisterModified(entity);

return UnitOfWork.Commit();

}

}

### **2、查询和删除增加了传参lamada表达式的方法**

仓储接口：

　　public interface IRepository<TEntity> where TEntity : AggregateRoot

{

//...........

#region 公共方法

/// <summary>

/// 根据lamada表达式查询集合

/// </summary>

/// <param name="selector">lamada表达式</param>

/// <returns></returns>

IQueryable<TEntity> Find(Expression<Func<TEntity, bool>> express);

/// <summary>

/// 根据lamada表达式删除对象

/// </summary>

/// <param name="selector"> lamada表达式 </param>

/// <returns> 操作影响的行数 </returns>

int Delete(Expression<Func<TEntity, bool>> express);

//..........

}

### **3、对于涉及到多张表需要连表的查询机制，我们还是通过神奇的Linq来解决。例如我们有一个通过角色取角色对应的菜单的接口需求。**

在菜单的仓储接口里面：

　　 /// <summary>

/// 菜单这个聚合根的仓储接口

/// </summary>

public interface IMenuRepository:IRepository<TB\_MENU>

{

IQueryable<TB\_MENU> GetMenusByRole(TB\_ROLE oRole);

}

对应仓储实现：

　　[Export(typeof(IMenuRepository))]

public class MenuRepository:EFBaseRepository<TB\_MENU>,IMenuRepository

{

public IQueryable<TB\_MENU> GetMenusByRole(TB\_ROLE oRole)

{

var queryrole = UnitOfWork.context.Set<TB\_ROLE>().AsQueryable();

var querymenu = UnitOfWork.context.Set<TB\_MENU>().AsQueryable();

var querymenurole = UnitOfWork.context.Set<TB\_MENUROLE>().AsQueryable();

var lstres = from menu in querymenu

from menurole in querymenurole

from role in queryrole

where menu.MENU\_ID == menurole.MENU\_ID &&

menurole.ROLE\_ID == role.ROLE\_ID &&

role.ROLE\_ID == oRole.ROLE\_ID

select menu;

return lstres;

}

}