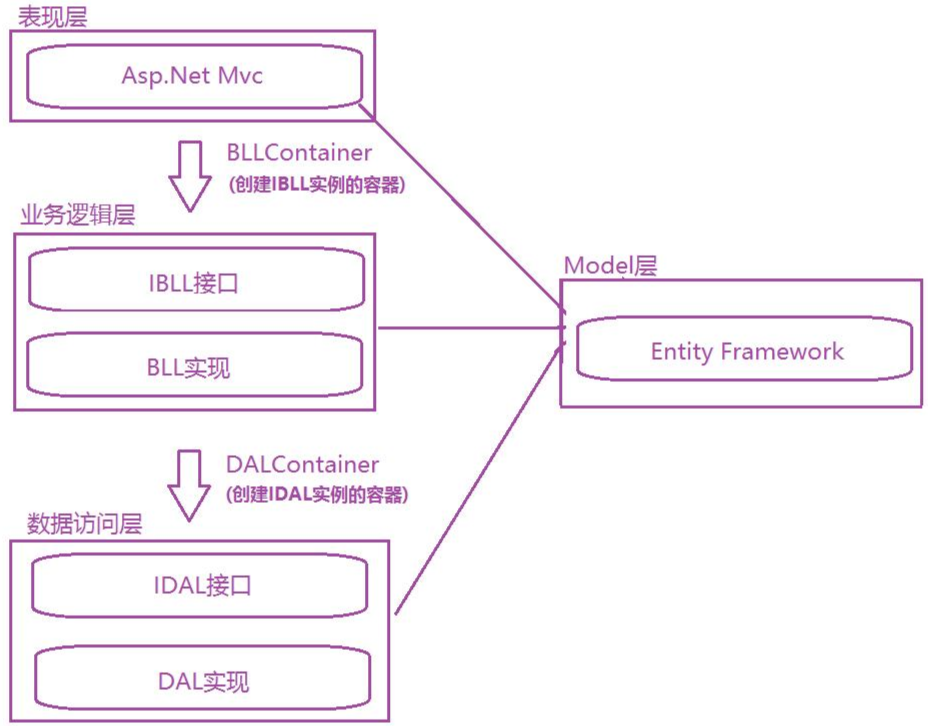
[Asp.Net MVC+EF+三层架构的完整搭建过程](http://www.cnblogs.com/zzqvq/p/5816091.html)

**架构图:**



**使用的数据库:**

一张公司的员工信息表，测试数据



解决方案项目设计：

1.新建一个空白解决方案名称为Company

2.在该解决方案下,新建解决方案文件夹(UI,BLL,DAL,Model) 当然还可以加上common

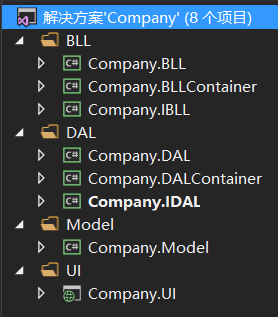
3.分别在BLL,DAL,Model 解决方案文件夹下创建类库项目

(1).BLL解决方案文件夹: Company.BLL、Company.IBLL、Company.BLLContainer

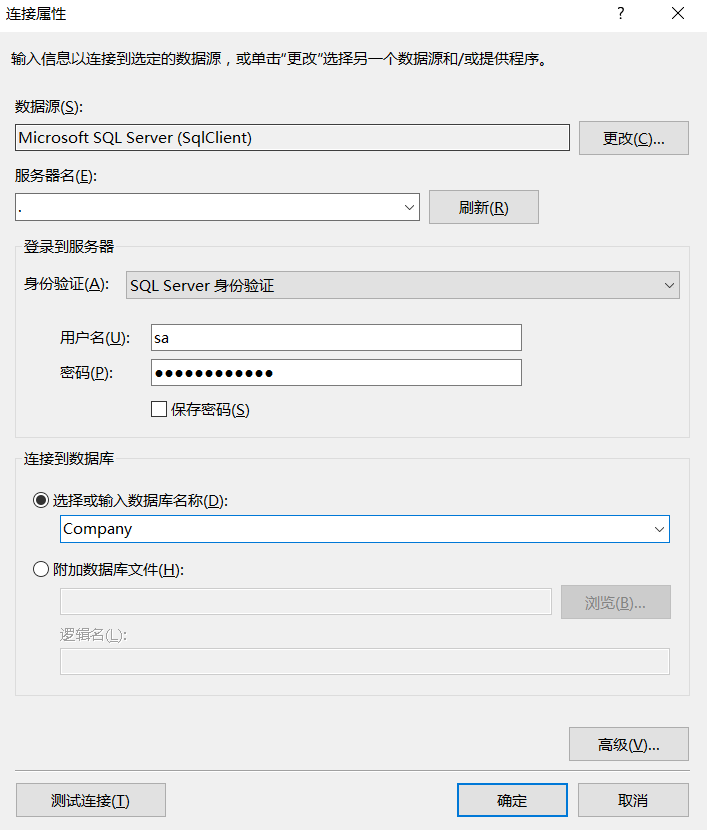
(2).DAL解决方案文件夹: Company.DAL、Company.IDAL、Company.DALContainer

(3).Model解决方案文件夹:Company.Model

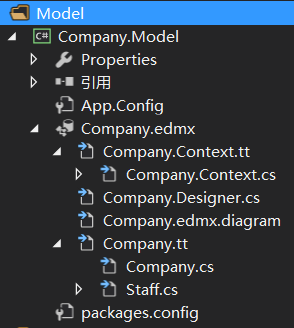
4.在UI 解决方案文件夹下添加一个ASP.NET Web应用程序,名称为Company.UI，选择我们的Mvc模板. 如图:



Model层: 选中Company.Model,右键=>添加=>新建项=>添加一个ADO.NET实体数据模型名称为Company=>选择来自数据库的EF设计器=>新建连接=>选择我们的Company数据库填入相应的内容



选择我们的Staff表，完成后如图:



这时Model层已经完成.我们的数据库连接字符串以及ef的配置都在App.Config里,但我们项目运行的是我们UI层的Web应用程序,所以我们这里要把App.Config里的配置复制到UI层的Web.Config中 数据访问层: 因为每一个实体都需要进行增删改查,所以我们这里封装一个基类.选中Company.IDAL,右键=>添加一个名称为IBaseDAL的接口=>写下公用的方法签名

[复制代码](javascript:void(0);)

著作权归作者所有。

商业转载请联系作者获得授权，非商业转载请注明出处。

作者：卷猫

链接：http://anneke.cn/ArticleInfo/Detial?id=11

来源：Anneke.cn

using System;

using System.Collections.Generic;

using System.Linq;

using System.Linq.Expressions;

using System.Text;

using System.Threading.Tasks;

namespace Company.IDAL

{

public partial interface IBaseDAL<T> where T : class, new()

{

void Add(T t);

void Delete(T t);

void Update(T t);

IQueryable<T> GetModels(Expression<Func<T, bool>> whereLambda);

IQueryable<T> GetModelsByPage<type>(int pageSize, int pageIndex, bool isAsc, Expression<Func<T, type>> OrderByLambda, Expression<Func<T, bool>> WhereLambda);

/// <summary>

/// 一个业务中有可能涉及到对多张表的操作,那么可以将操作的数据,打上相应的标记,最后调用该方法,将数据一次性提交到数据库中,避免了多次链接数据库。

/// </summary>

bool SaveChanges();

}

}

[复制代码](javascript:void(0);)

基类接口封装完成.然后选中Company.IDAL,右键=>添加一个名称为IStaffDAL的接口=>继承自基类接口

[复制代码](javascript:void(0);)

著作权归作者所有。

商业转载请联系作者获得授权，非商业转载请注明出处。

作者：卷猫

链接：http://anneke.cn/ArticleInfo/Detial?id=11

来源：Anneke.cn

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Company.Model;

namespace Company.IDAL

{

public partial interface IStaffDAL:IBaseDAL<Staff>

{

}

}

[复制代码](javascript:void(0);)

IDAL完成，接下来是DAL 选中Company.DAL=>右键=>添加一个类,名称为:BaseDAL,该类是我们对IBaseDAL具体的实现,我们这里需要用到ef上下文对象，所以添加引用EntityFramework.dll和EntityFramework.SqlServer.dll(这里是ef6版本不同引用的dll也可能不同) 上面说到我们这里要用到ef下上文对象,我们这里不能直接new,因为这样的话可能会造成数据混乱,所以要让ef上下文对象保证线程内唯一。 我们选中Company.DAL=>右键=>添加一个类.名称为DbContextFactory.通过这个类才创建ef上下文对象.

[复制代码](javascript:void(0);)

著作权归作者所有。

商业转载请联系作者获得授权，非商业转载请注明出处。

作者：卷猫

链接：http://anneke.cn/ArticleInfo/Detial?id=11

来源：Anneke.cn

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Runtime.Remoting.Messaging;

using System.Text;

using System.Threading.Tasks;

using Company.Model;

namespace Company.DAL

{

public partial class DbContextFactory

{

/// <summary>

/// 创建EF上下文对象,已存在就直接取,不存在就创建,保证线程内是唯一。

/// </summary>

public static DbContext Create()

{

DbContext dbContext = CallContext.GetData("DbContext") as DbContext;

if (dbContext==null)

{

dbContext=new CompanyEntities();

CallContext.SetData("DbContext",dbContext);

}

return dbContext;

}

}

}

[复制代码](javascript:void(0);)

EF上下文对象创建工厂完成,这时我们来完成我们的BaseDAL

[复制代码](javascript:void(0);)

著作权归作者所有。

商业转载请联系作者获得授权，非商业转载请注明出处。

作者：卷猫

链接：http://anneke.cn/ArticleInfo/Detial?id=11

来源：Anneke.cn

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Data.Entity.Migrations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Company.IDAL;

using System.Linq.Expressions;

namespace Company.DAL

{

public partial class BaseDAL<T> where T : class, new()

{

private DbContext dbContext = DbContextFactory.Create();

public void Add(T t)

{

dbContext.Set<T>().Add(t);

}

public void Delete(T t)

{

dbContext.Set<T>().Remove(t);

}

public void Update(T t)

{

dbContext.Set<T>().AddOrUpdate(t);

}

public IQueryable<T> GetModels(Expression<Func<T, bool>> whereLambda)

{

return dbContext.Set<T>().Where(whereLambda);

}

public IQueryable<T> GetModelsByPage<type>(int pageSize, int pageIndex, bool isAsc,

Expression<Func<T, type>> OrderByLambda, Expression<Func<T, bool>> WhereLambda)

{

//是否升序

if (isAsc)

{

return dbContext.Set<T>().Where(WhereLambda).OrderBy(OrderByLambda).Skip((pageIndex - 1) \* pageSize).Take(pageSize);

}

else

{

return dbContext.Set<T>().Where(WhereLambda).OrderByDescending(OrderByLambda).Skip((pageIndex - 1) \* pageSize).Take(pageSize);

}

}

public bool SaveChanges()

{

return dbContext.SaveChanges() > 0;

}

}

}

[复制代码](javascript:void(0);)

BaseDAL完成后,我们在添加一个类名称为StaffDAL,继承自BaseDAL，实现IStaffDAL接口

[复制代码](javascript:void(0);)

著作权归作者所有。

商业转载请联系作者获得授权，非商业转载请注明出处。

作者：卷猫

链接：http://anneke.cn/ArticleInfo/Detial?id=11

来源：Anneke.cn

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Company.IDAL;

using Company.Model;

namespace Company.DAL

{

public partial class StaffDAL:BaseDAL<Staff>,IStaffDAL

{

}

}

[复制代码](javascript:void(0);)

StaffDAL完成后，我们要完成的是DALContainer,该类库主要是创建IDAL的实例对象,我们这里可以自己写一个工厂也可以通过一些第三方的IOC框架,这里使用Autofac 1.选中DALContainer=>右键=>管理Nuget程序包=>搜索Autofac=>下载安装对应,net版本的AutoFac 2.安装完成后,我们在DALContainer下添加一个名为Container的类.

[复制代码](javascript:void(0);)

著作权归作者所有。

商业转载请联系作者获得授权，非商业转载请注明出处。

作者：卷猫

链接：http://anneke.cn/ArticleInfo/Detial?id=11

来源：Anneke.cn

using Autofac;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Company.DAL;

using Company.IDAL;

namespace Company.DALContainer

{

public class Container

{

/// <summary>

/// IOC 容器

/// </summary>

public static IContainer container = null;

/// <summary>

/// 获取 IDal 的实例化对象

/// </summary>

/// <typeparam name="T"></typeparam>

/// <returns></returns>

public static T Resolve<T>()

{

try

{

if (container == null)

{

Initialise();

}

}

catch (System.Exception ex)

{

throw new System.Exception("IOC实例化出错!" + ex.Message);

}

return container.Resolve<T>();

}

/// <summary>

/// 初始化

/// </summary>

public static void Initialise()

{

var builder = new ContainerBuilder();

//格式：builder.RegisterType<xxxx>().As<Ixxxx>().InstancePerLifetimeScope();

builder.RegisterType<StaffDAL>().As<IStaffDAL>().InstancePerLifetimeScope();

container = builder.Build();

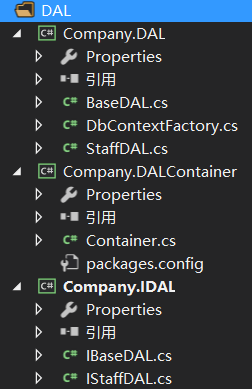
}

}

}

[复制代码](javascript:void(0);)

这时候我们数据访问层已经完成,结构如下



**业务逻辑层:**

选中Company.IBLL,右键=>添加一个接口,名称为IBaseService,里面也是封装的一些公用方法

[复制代码](javascript:void(0);)

著作权归作者所有。

商业转载请联系作者获得授权，非商业转载请注明出处。

作者：卷猫

链接：http://anneke.cn/ArticleInfo/Detial?id=11

来源：Anneke.cn

using System;

using System.Collections.Generic;

using System.Linq;

using System.Linq.Expressions;

using System.Text;

using System.Threading.Tasks;

namespace Company.IBLL

{

public partial interface IBaseService<T> where T:class ,new()

{

bool Add(T t);

bool Delete(T t);

bool Update(T t);

IQueryable<T> GetModels(Expression<Func<T, bool>> whereLambda);

IQueryable<T> GetModelsByPage<type>(int pageSize, int pageIndex, bool isAsc, Expression<Func<T, type>> OrderByLambda, Expression<Func<T, bool>> WhereLambda);

}

}

[复制代码](javascript:void(0);)

IBaseService完成后,我们继续添加一个接口，名称为IStaffService,继承自IBaseService

[复制代码](javascript:void(0);)

著作权归作者所有。

商业转载请联系作者获得授权，非商业转载请注明出处。

作者：卷猫

链接：http://anneke.cn/ArticleInfo/Detial?id=11

来源：Anneke.cn

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Company.Model;

namespace Company.IBLL

{

public partial interface IStaffService : IBaseService<Staff>

{

}

}

[复制代码](javascript:void(0);)

Company.IBLL,完成后,我们开始完成BLL 选中Company.BLL=>右键=>添加一个类,名称为BaseService，这个类是对IBaseService的具体实现. 这个类需要调用IDAL接口实例的方法，不知道具体调用哪一个IDAL实例,我这里只有一张Staff表，也就只有一个IStaffDAL的实例,但是如果我们这里有很多表的话，就有很多IDAL接口实例，这时我们的基类BaseService不知道调用哪一个，但是继承它的子类知道. 所以我们这里把BaseService定义成抽象类,写一个IBaseDAL的属性，再写一个抽象方法,该方法的调用写在 BaseService默认的无参构造函数内,当BaseService创建实例的时候会执行这个抽象方法,然后执行子类重写它的方法 为IBaseDAL属性赋一个具体的IDAL实例对象.

[复制代码](javascript:void(0);)

著作权归作者所有。

商业转载请联系作者获得授权，非商业转载请注明出处。

作者：卷猫

链接：http://anneke.cn/ArticleInfo/Detial?id=11

来源：Anneke.cn

using System;

using System.Collections.Generic;

using System.Linq;

using System.Linq.Expressions;

using System.Text;

using System.Threading.Tasks;

using Company.IDAL;

namespace Company.BLL

{

public abstract partial class BaseService<T> where T : class, new()

{

public BaseService()

{

SetDal();

}

public IBaseDAL<T> Dal{get;set;};

public abstract void SetDal();

public bool Add(T t)

{

Dal.Add(t);

return Dal.SaveChanges();

}

public bool Delete(T t)

{

Dal.Delete(t);

return Dal.SaveChanges();

}

public bool Update(T t)

{

Dal.Update(t);

return Dal.SaveChanges();

}

public IQueryable<T> GetModels(Expression<Func<T, bool>> whereLambda)

{

return Dal.GetModels(whereLambda);

}

public IQueryable<T> GetModelsByPage<type>(int pageSize, int pageIndex, bool isAsc,

Expression<Func<T, type>> OrderByLambda, Expression<Func<T, bool>> WhereLambda)

{

return Dal.GetModelsByPage(pageSize, pageIndex, isAsc, OrderByLambda, WhereLambda);

}

}

}

[复制代码](javascript:void(0);)

基类BaseService完成后，我们去完成子类StaffService,添加一个类名称为StaffService,继承BaseService,实现IStaffService,重写父类的抽象方法,为父类的IBaseDAL属性赋值

[复制代码](javascript:void(0);)

著作权归作者所有。

商业转载请联系作者获得授权，非商业转载请注明出处。

作者：卷猫

链接：http://anneke.cn/ArticleInfo/Detial?id=11

来源：Anneke.cn

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Company.IBLL;

using Company.IDAL;

using Company.Model;

namespace Company.BLL

{

public partial class StaffService : BaseService<Staff>, IStaffService

{

private IStaffDAL StaffDAL = DALContainer.Container.Resolve<IStaffDAL>();

public override void SetDal()

{

Dal = StaffDAL;

}

}

}

[复制代码](javascript:void(0);)

子类完成后,我们选中BLLContainer添加一个名为Container的类,添加对Autofac.dll 的引用,该类是创建IBLL的实例

[复制代码](javascript:void(0);)

著作权归作者所有。

商业转载请联系作者获得授权，非商业转载请注明出处。

作者：卷猫

链接：http://anneke.cn/ArticleInfo/Detial?id=11

来源：Anneke.cn

using Autofac;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Company.BLL;

using Company.IBLL;

namespace Company.BLLContainer

{

public class Container

{

/// <summary>

/// IOC 容器

/// </summary>

public static IContainer container = null;

/// <summary>

/// 获取 IDal 的实例化对象

/// </summary>

/// <typeparam name="T"></typeparam>

/// <returns></returns>

public static T Resolve<T>()

{

try

{

if (container == null)

{

Initialise();

}

}

catch (System.Exception ex)

{

throw new System.Exception("IOC实例化出错!" + ex.Message);

}

return container.Resolve<T>();

}

/// <summary>

/// 初始化

/// </summary>

public static void Initialise()

{

var builder = new ContainerBuilder();

//格式：builder.RegisterType<xxxx>().As<Ixxxx>().InstancePerLifetimeScope();

builder.RegisterType<StaffService>().As<IStaffService>().InstancePerLifetimeScope();

container = builder.Build();

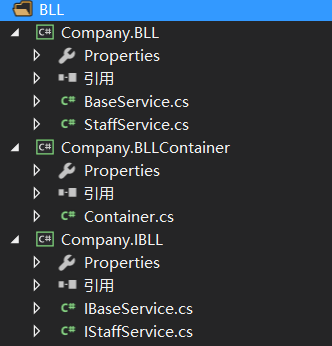
}

}

}

[复制代码](javascript:void(0);)

业务逻辑层完成0

,

**表现层:**

添加对EntityFramework.SqlServer.dll 的引用,不然会报错. 我们这里写个简单的增删改查测试一下,过程就不具体描述了,

**视图:**

[复制代码](javascript:void(0);)

著作权归作者所有。

商业转载请联系作者获得授权，非商业转载请注明出处。

作者：卷猫

链接：http://anneke.cn/ArticleInfo/Detial?id=11

来源：Anneke.cn

@using Company.Model

@model List<Company.Model.Staff>

@{

Layout = null;

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>Index</title>

</head>

<body>

<div>

<h1>简单的画一个表格展示数据</h1>

<table border="1" cellpadding="0" cellspacing="0">

<tr>

<th>ID</th>

<th>姓名</th>

<th>年龄</th>

<th>性别</th>

</tr>

@foreach (Staff staff in Model)

{

<tr>

<td>@staff.Id</td>

<td>@staff.Name</td>

<td>@staff.Age</td>

<td>@staff.Sex</td>

</tr>

}

</table>

<hr/>

<h1>增加</h1>

<form action="@Url.Action("Add", "Home")" method="POST">

姓名:<input name="Name"/>

年龄:<input name="Age"/>

性别:<input name="Sex"/>

<button type="submit">提交</button>

</form>

<hr/>

<h1>修改</h1>

<form action="@Url.Action("update", "Home")" method="POST">

Id:<input name="Id" />

姓名:<input name="Name"/>

年龄:<input name="Age"/>

性别:<input name="Sex"/>

<button type="submit">提交</button>

</form>

<hr />

<h1>删除</h1>

<form action="@Url.Action("Delete", "Home")" method="POST">

Id:<input name="Id" />

<button type="submit">提交</button>

</form>

</div>

</body>

</html>

[复制代码](javascript:void(0);)

控制器:

[复制代码](javascript:void(0);)

著作权归作者所有。

商业转载请联系作者获得授权，非商业转载请注明出处。

作者：卷猫

链接：http://anneke.cn/ArticleInfo/Detial?id=11

来源：Anneke.cn

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using Company.IBLL;

using Company.Model;

namespace Company.UI.Controllers

{

public class HomeController : Controller

{

private IStaffService StaffService = BLLContainer.Container.Resolve<IStaffService>();

// GET: Home

public ActionResult Index()

{

List<Staff>list = StaffService.GetModels(p => true).ToList();

return View(list);

}

public ActionResult Add(Staff staff)

{

if (StaffService.Add(staff))

{

return Redirect("Index");

}

else

{

return Content("no");

}

}

public ActionResult Update(Staff staff)

{

if (StaffService.Update(staff))

{

return Redirect("Index");

}

else

{

return Content("no");

}

}

public ActionResult Delete(int Id)

{

var staff = StaffService.GetModels(p => p.Id == Id).FirstOrDefault();

if (StaffService.Delete(staff))

{

return Redirect("Index");

}

else

{

return Content("no");

}

}

}

}

[复制代码](javascript:void(0);)

完整Demo https://github.com/zhenzhenkeai/Asp.Net-MVC-EF-Demo

可参见记事宝里的Asp.Net-MVC-EF-Demo.rar