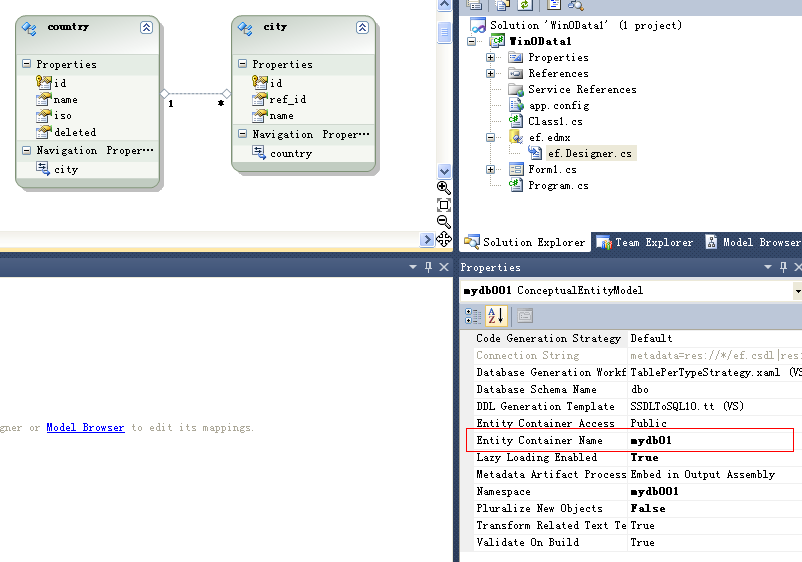
**WCF Data Service**

1. **Create ADO.Net Entity**



1. 宿主服务类: 继承自 DataService< T >:

using System.Data.Services;

using System.Data.Services.Common;

public class mydbsevive : DataService<mydb01>

{

public static void InitializeService(DataServiceConfiguration config)

{

config.SetEntitySetAccessRule("\*", EntitySetRights.All);

config.SetServiceOperationAccessRule("\*", ServiceOperationRights.All);

config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2;

}

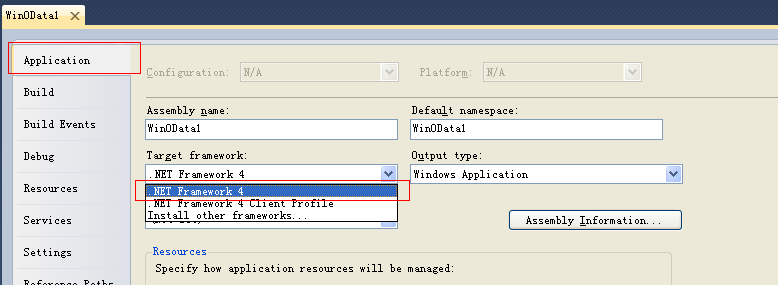
}

**宿主服务:** DataServiceHost 或者 WebServiceHost, DataServiceHost派生自WebServiceHost

DataServiceHost : [System.Data.Services 命名空间](ms-help://MS.MSDNQTR.v90.chs/fxref_system.data.services/html/79416b62-11ea-149d-11f2-011ae9cc15b0.htm)

WebServiceHost: [System.ServiceModel.Web 命名空间](ms-help://MS.MSDNQTR.v90.chs/fxref_system.servicemodel.web/html/887434a8-e424-8451-556c-8f43f026d783.htm)

如果找不到, 相应的命名空间, 请到 Project Property 里修改 Target Framework:



using System.Data.Services;

using System.Data.Services.Common;

using System.ServiceModel;

using System.ServiceModel.Channels;

using System.ServiceModel.Web;

代码:

Uri[] ba = new Uri[]{ new Uri("http://localhost:6455/dataservice") };

wsh = new DataServiceHost(typeof(mydbsevive), ba);

WebHttpBinding whb = new WebHttpBinding();

wsh.AddServiceEndpoint(typeof(System.Data.Services.IRequestHandler), whb, "web1");

wsh.Open();

或者使用:

Uri[] ba = new Uri[]{ new Uri("http://localhost:6455/dataservice") };

wsh = new WebServiceHost(typeof(mydbsevive), ba);

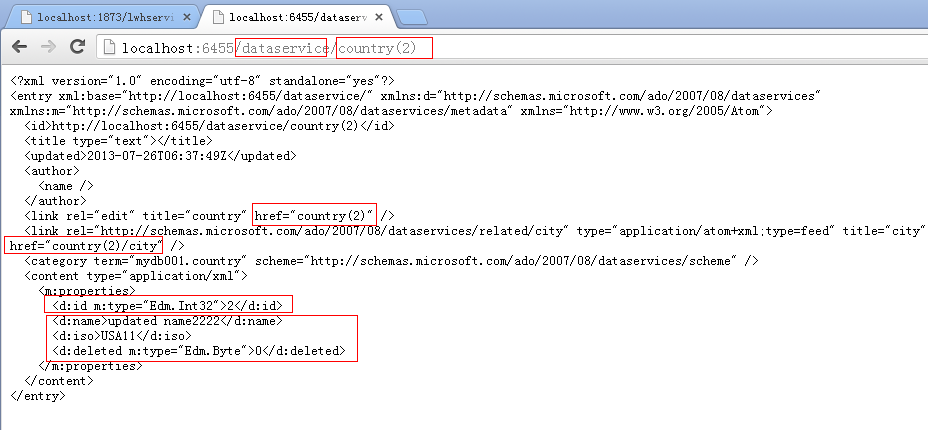
WebHttpBinding whb = new WebHttpBinding();

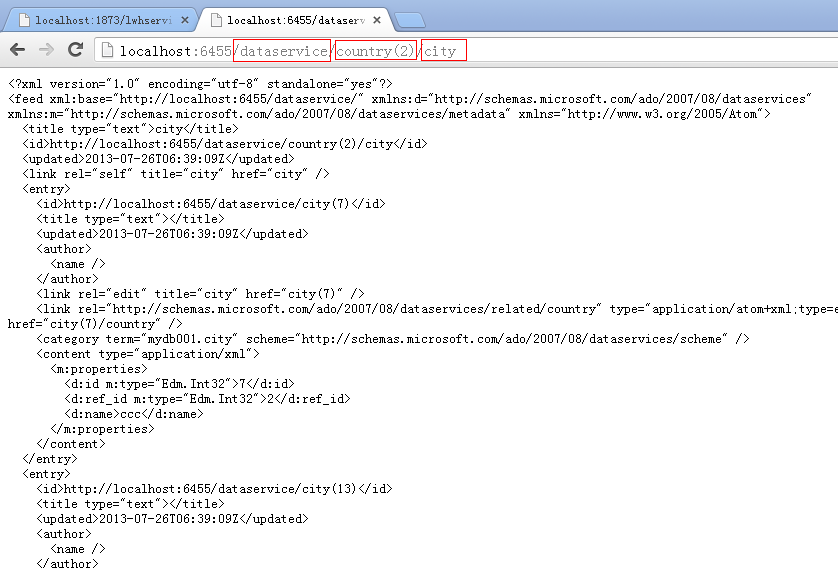
wsh.AddServiceEndpoint(typeof(System.Data.Services.IRequestHandler), whb, "web1");

wsh.Open();

服务的Container







或者Host IIS ASP.NET

<%@ ServiceHost Language="C#" Factory="System.Data.Services.DataServiceHostFactory, System.Data.Services, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" Service="webdata1.lwhservice" %>

public class lwhservice : DataService<db01>

{

// This method is called only once to initialize service-wide policies.

public static void InitializeService(DataServiceConfiguration config)

{

// TODO: set rules to indicate which entity sets and service operations are visible, updatable, etc.

// Examples:

config.SetEntitySetAccessRule("\*", EntitySetRights.All);

config.SetServiceOperationAccessRule("\*", ServiceOperationRights.All);

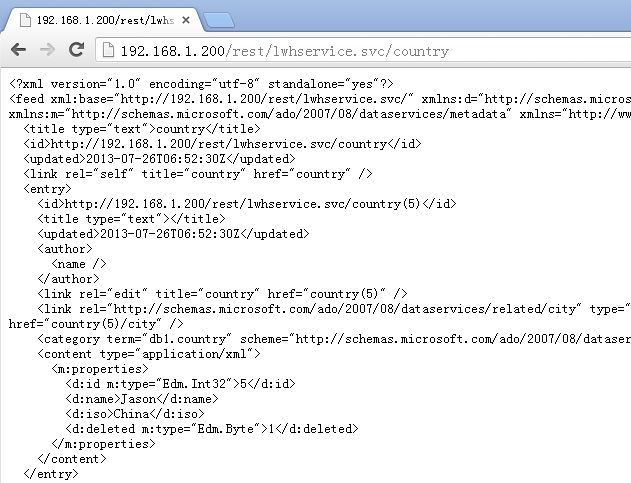
config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2;

}

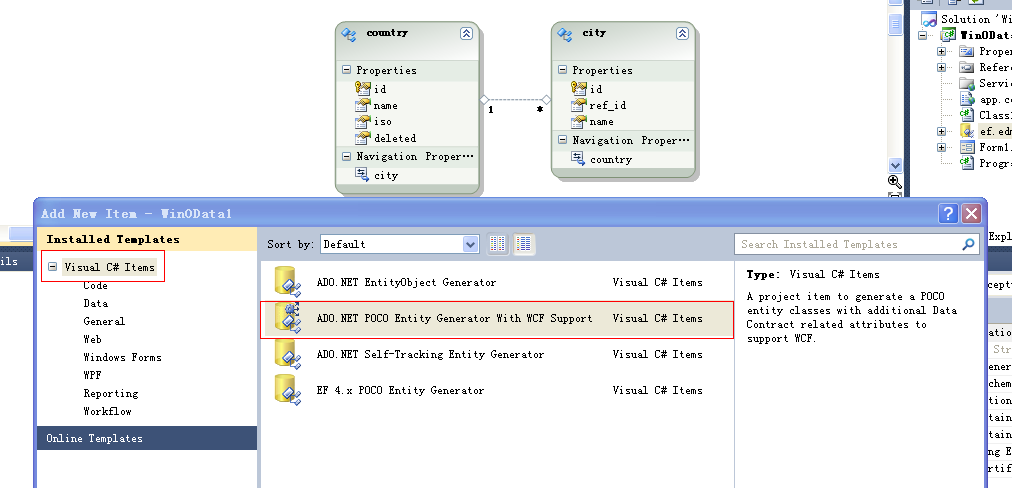
}

如果使用原始的 Entity Framework: 作为 DataService:

<http://192.168.1.200/rest/lwhservice.svc/country>



如果使用转化摸版：



访问数据时，则出错：



**解决办法：**

在自动生成的代码里， 找到 ObjectContext 类， 添加一句到所有的Constructor 里：

this.ContextOptions.ProxyCreationEnabled = false;

public partial class db01 : ObjectContext

{

public const string ConnectionString = "name=db01";

public const string ContainerName = "db01";

#region Constructors

public db01()

: base(ConnectionString, ContainerName)

{

this.ContextOptions.ProxyCreationEnabled = false;

this.ContextOptions.LazyLoadingEnabled = true;

}

public db01(string connectionString)

: base(connectionString, ContainerName)

{

this.ContextOptions.ProxyCreationEnabled = false;

this.ContextOptions.LazyLoadingEnabled = true;

}

public db01(EntityConnection connection)

: base(connection, ContainerName)

{

this.ContextOptions.ProxyCreationEnabled = false;

this.ContextOptions.LazyLoadingEnabled = true;

}

#endregion

自定义的DataService: 数据源可以根据自己需要灵活定制:

1. Entity Class: 必须有[DataServiceKeyAttribute("id")] 指明 Key , 否则出错.

[DataServiceKeyAttribute("id")]

public class cnt {

public int id { get; set; } 必须是属性, 以 get; set; 来获取设置. 否则无法暴露给客户端.

public string name {get; set;} 必须是属性,

public cnt()

{

}

}

1. 定义数据契约: [数据服务必须是](file:///\\数据服务必须是) IQueryable< T >

public class cntlist {

public static IList<cnt> \_cntlist;

public cntlist() {

\_cntlist = new List<cnt> {

new cnt() { id=10, name="William" },

new cnt() { id=20, name= "Lilian" }

};

}

public IQueryable<cnt> country { \\获取数据时 URI 的一部分 <http://localhost:6455/dataservice/web1/country>

get

{

return \_cntlist.AsQueryable<cnt>();

}

}

}

1. 定义服务类:

using System.Data.Services;

using System.Data.Services.Common;

namespace WinOData1

{

public class mydbsevive : DataService<cntlist>

{

public static void InitializeService(DataServiceConfiguration config)

{

config.SetEntitySetAccessRule("\*", EntitySetRights.All);

config.SetServiceOperationAccessRule("\*", ServiceOperationRights.All);

config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2;

}

}

}

1. 宿主服务: 原理同上: 也可以使用 DataServiceHost

Uri[] ba = new Uri[]{ new Uri("http://localhost:6455/dataservice") };

wsh = new WebServiceHost(typeof(mydbsevive), ba);

WebHttpBinding whb = new WebHttpBinding();

wsh.AddServiceEndpoint(typeof(System.Data.Services.IRequestHandler), whb, "web1");

wsh.Open();

**自定义 ling to sql:**

[Table(Name = "country")]

[DataServiceKeyAttribute("id")] // 必须有DataServiceKey , 否则出错:

public class cctt

{

private int \_id;

[Column(Name = "id", Storage = "\_id", IsPrimaryKey = true, IsDbGenerated = true, DbType = "int")]

public int id

{

get { return \_id; }

set { this.\_id = value; }

}

private string \_name;

[Column(Name = "name",Storage="\_name", DbType = "varchar(50)")]

public string name

{

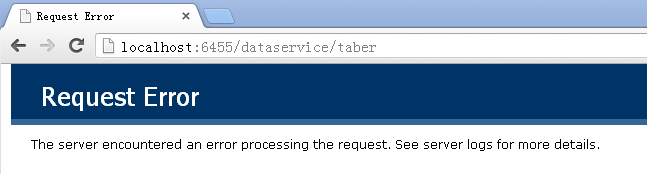
get { return this.\_name; }

set { this.\_name = value; }

}

}

如果没有: [DataServiceKeyAttribute("id")] 则出错:

****

public class cntlist {

public static IList<cnt> \_cntlist;

public cntlist() {

\_cntlist = new List<cnt> {

new cnt() { id=10, name="William" },

new cnt() { id=20, name= "Lilian" }

};

}

public IQueryable<cnt> country {

get

{

return \_cntlist.AsQueryable<cnt>();

}

}

public IQueryable<cctt> taber

{

get

{

SqlConnection scon = new SqlConnection(@"Server=CANSER\MYSQL2008;Database=mydb01;uid=lwh;pwd=lwh123;");

DataContext db = new DataContext(scon);

Table<cctt> cc = db.GetTable<cctt>();

return cc.AsQueryable<cctt>();

}

}

}

**服务类:**

public class mydbsevive : DataService<cntlist>

{

public static void InitializeService(DataServiceConfiguration config)

{

config.SetEntitySetAccessRule("\*", EntitySetRights.All);

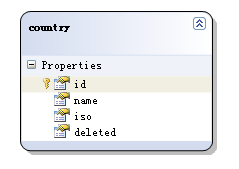
config.SetServiceOperationAccessRule("\*", ServiceOperationRights.All);

config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2;

}

}

**如果是自动生成的 xxx.dbml**

****

**必须手工修改各个entity class 的属性:**

[global::System.Data.Linq.Mapping.TableAttribute(Name="dbo.country")]

[DataServiceKey("id")]

public partial class country : INotifyPropertyChanging, INotifyPropertyChanged

{ …. }

[DataServiceKey("id")] 非常关键, 否则出错:

/// 整个自动生成的 DataContext 而就是对应 数据库DB

[global::System.Data.Linq.Mapping.DatabaseAttribute(Name="mydb01")]

public partial class mybusDataContext : System.Data.Linq.DataContext

{

private static System.Data.Linq.Mapping.MappingSource mappingSource = new AttributeMappingSource();

#region Extensibility Method Definitions

partial void OnCreated();

partial void Insertcountry(country instance);

partial void Updatecountry(country instance);

partial void Deletecountry(country instance);

#endregion

public mybusDataContext() :

base(global::WinOData1.Properties.Settings.Default.mydb01ConnectionString, mappingSource)

{

OnCreated();

}

public mybusDataContext(string connection) :

base(connection, mappingSource)

{

OnCreated();

}

public mybusDataContext(System.Data.IDbConnection connection) :

base(connection, mappingSource)

{

OnCreated();

}

public mybusDataContext(string connection, System.Data.Linq.Mapping.MappingSource mappingSource) :

base(connection, mappingSource)

{

OnCreated();

}

public mybusDataContext(System.Data.IDbConnection connection, System.Data.Linq.Mapping.MappingSource mappingSource) :

base(connection, mappingSource)

{

OnCreated();

}

public System.Data.Linq.Table<country> countries

{

get

{

return this.GetTable<country>();

}

}

}

**而服务类如下:**

public class mydbsevive : DataService<WinOData2.mybusDataContext>

{

public static void InitializeService(DataServiceConfiguration config)

{

config.SetEntitySetAccessRule("\*", EntitySetRights.All);

config.SetServiceOperationAccessRule("\*", ServiceOperationRights.All);

config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2;

}

}

**启动服务如下:**

Uri[] ba = new Uri[]{ new Uri("http://localhost:6455/dataservice") };

wsh = new WebServiceHost(typeof(mydbsevive), ba);

WebHttpBinding whb = new WebHttpBinding();

wsh.AddServiceEndpoint(typeof(System.Data.Services.IRequestHandler), whb, "web1");

wsh.Open();

**而对于 Entity Framework 的 XXX.edmx 则不需要**[DataServiceKey("id")],

**XXX.edmx 直接用于 DataService**

public class mydbsevive : DataService<mydb01>

{

public static void InitializeService(DataServiceConfiguration config)

{

config.SetEntitySetAccessRule("\*", EntitySetRights.All);

config.SetServiceOperationAccessRule("\*", ServiceOperationRights.All);

config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2;

}

}

**如果使用其他模板转化, 则需要注意:**

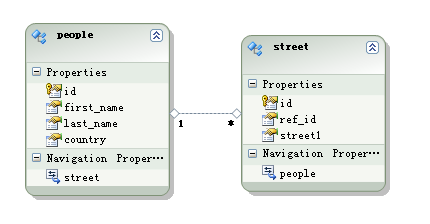
public partial class db01 : ObjectContext **的构造函数里记得设置:**

this.ContextOptions.ProxyCreationEnabled = false;

ADO.Net Data Service 中更新和查询一对多实体集合的注意事项：.

相关网站: <http://www.doc88.com/p-181436060108.html>

举例：



lwhSer.svc

<%@ ServiceHost Language="C#" Factory="System.Data.Services.DataServiceHostFactory, System.Data.Services, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" Service="webEF.lwhSer" %>

lwhSer.svc.cs

using System;

using System.Collections.Generic;

using System.Data.Services;

using System.Data.Services.Common;

using System.Linq;

using System.ServiceModel.Web;

using System.Web;

namespace webEF

{

public class lwhSer : DataService<mydb01>

{

public static void InitializeService(DataServiceConfiguration config)

{

config.SetEntitySetAccessRule("\*", EntitySetRights.All);

config.SetServiceOperationAccessRule("\*", ServiceOperationRights.All);

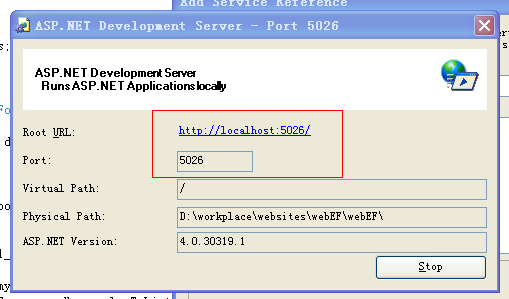
config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2;

}

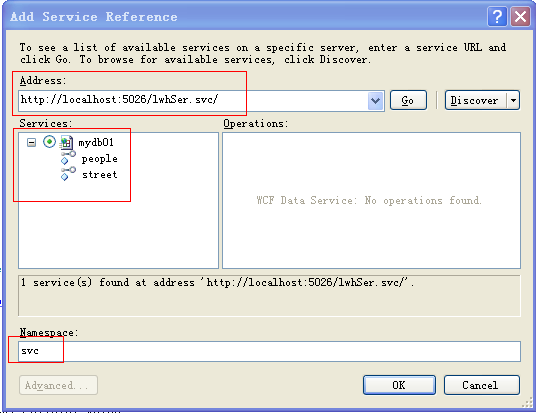
}

}

在本机调试运行： lwhSer.svc



在客户端程序添加服务：



public svc.mydb01 db;

db = new svc.mydb01(new Uri("http://localhost:5026/lwhSer.svc"));

我们看正常的插入操作：

svc.people np = new svc.people

{

first\_name = "William",

last\_name = "liu",

country = "USA"

};

svc.street ns = new svc.street

{

street1 = "Burlington"

};

np.street.Add(ns); // 通过关联表添加记录

db.AddTopeople(np);

db.SaveChanges();

**注意：**

上述代码执行时不会报任何错误， 但是问题在于： 它只插入了主表的记录 people. 而附表 street 则忽略没有插入。

**原因：这是为什么呢**：因为ADO.Net Data Service 是基于http 的一个服务， 考虑到数据量以及实际应用的情况， 所以默认情况下， 它不会将一个对象所关联的其他对象都加载进来。反过来讲也一样， 它不会自动将对象关联起来。也就是说添加的 street 对象其实没有跟主表 people 关联。既然没有关联， 自然不会更新。

如何实现： **DataServiceContext** 提供了方法: **AddRelatedObject**

svc.people np = new svc.people

{

first\_name = "William",

last\_name = "liu",

country = "USA"

};

svc.street ns = new svc.street

{

street1 = "Burlington"

};

svc.street ns1 = new svc.street

{

street1 = "Burnaby"

};

svc.street ns2 = new svc.street

{

street1 = "Canada Way"

};

db.AddTopeople(np);

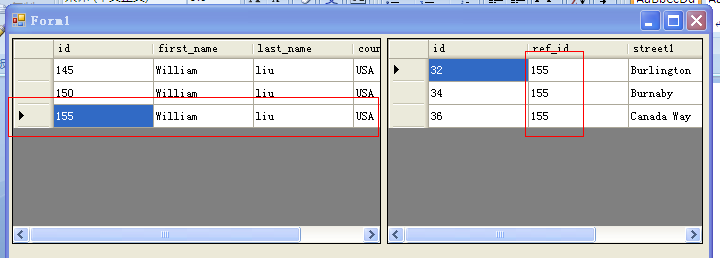
db.AddRelatedObject(np, "street", ns);

db.AddRelatedObject(np, "street", ns1);

db.AddRelatedObject(np, "street", ns2);

db.SaveChanges();

以上实现 一对多关联的插入操作：



**理解延迟加载：**LoadProperty( object, property\_name) 解决方法。

考虑到数据量的问题， LINQ to SQL 或者 Entity Framework默认是不会自动加载一个对象所有关联的对象的。这有显而易见的好处。如果全部加载非但费力，也不是用户所需要的。

延迟加载当然也有一个问题： 就是它需要多次发起数据库查询。 这是无法避免的。因为毕竟它需要加载过来。但这一点可以让人理解，至少它是按需要发起这些请求。

svc.people np = db.people.Where(p => p.id == 155).SingleOrDefault();

MessageBox.Show(string.Format("id:{0} count:{1}", np.id, np.street.Count));

如果这样查询： 我们发现 np.street.Count 属性始终返回 0

svc.people np = db.people.Where(p => p.id == 155).SingleOrDefault();

db.LoadProperty(np, "street");

MessageBox.Show(string.Format("id:{0} count:{1}", np.id, np.street.Count));

如果这样查询： 我们发现 np.street.Count 属性如实返回附表记录个数：3

**修改操作：**

svc.people np = db.people.Where(p => p.id == 155).SingleOrDefault();

np.first\_name = np.first\_name + ":Chg:" + DateTime.Now.ToLongTimeString();

db.LoadProperty(np, "street");

svc.street ns = np.street.FirstOrDefault();

ns.street1 = "change:street";

db.UpdateObject(ns); // 必须对修改的实体，呼叫 UpdateObject 目的是将状态标注为： Modified

db.UpdateObject(np); // 所有被修改过的实体，都需要呼叫UpdateObject, 不管是主表还是附表的各个记录实体。

db.SaveChanges();

**修改变换关联表：把 160 的子记录 移动到 180 上**

svc.people np = db.people.Where(p => p.id == 160).SingleOrDefault();

db.LoadProperty(np, "street");

svc.people np1 = db.people.Where(p => p.id == 180).SingleOrDefault();

foreach (svc.street ss in np.street)

{

ss.ref\_id = np1.id; // 改变外键的值。来修改所属

db.UpdateObject(ss);

}

np.street.Clear(); //只是为了清楚客户端的已经转移的记录。

db.SaveChanges();

**修改查询返回的单个个体：**

IEnumerable<svc.people> nnp = db.Execute<svc.people>(new Uri("people(180)", UriKind.Relative));

svc.people np = nnp.FirstOrDefault();

np.first\_name = "Change to Samsame";

db.UpdateObject(np);

db.SaveChanges();

**删除操作：**

svc.people np = db.people.Where(p => p.id == 165).SingleOrDefault();

np.first\_name = np.first\_name + ":Chg:" + DateTime.Now.ToLongTimeString();

db.LoadProperty(np, "street");

foreach (svc.street ss in np.street)

{

db.DeleteObject(ss);

}

db.DeleteObject(np);

db.SaveChanges();

必须先删除明细， 再删除主表

LoadProperty(master\_obj, “child\_object\_name”);

AddRelatedObject( master\_obj, “name of child object”, child\_object);

**各种方法支持异步操作：**

db.BeginExecute<svc.people>(new Uri("http://localhost:5026/lwhSer.svc/people"), callbk, db);

private void callbk(IAsyncResult e)

{

svc.mydb01 obj = (svc.mydb01)e.AsyncState;

IEnumerable<svc.people> npl = obj.EndExecute<svc.people>(e);

//this.dg2.DataSource = npl.ToList();

MessageBox.Show("excute:" + npl.Count().ToString() );

}

db.BeginSaveChanges(callsave, db);

private void callsave(IAsyncResult e)

{

svc.mydb01 obj = (svc.mydb01)e.AsyncState;

obj.EndSaveChanges(e);

MessageBox.Show("Save:" + e.IsCompleted.ToString());

}

**通过URI 灵活查询：**

IEnumerable<svc.people> res = db.Execute<svc.people>(new Uri("http://192.168.1.200/youtest/lwhSer.svc/people(180)"));

this.label1.Text = res.FirstOrDefault().first\_name;

返回类型都是 IEnumerable<T> :

IEnumerable<string> res = db.Execute<string>(new Uri("http://192.168.1.200/youtest/lwhSer.svc/people(180)/first\_name"));

this.label1.Text = res.FirstOrDefault();

分析返回类型：

// first\_name 是 string 类型， 查询返回值就是 IEnumerable<string>, 我们知道返回只有一个值，

// FirstOrDefault 就是string 类型

**拦截器： 查询拦截器， 修改拦截器：**

**查询拦截器：**

using System.Data.Services.Common;

using System.Linq;

using System.ServiceModel.Web;

using System.Web;

using System.Linq.Expressions; // Expression 所在的命名空间

namespace webEF

{

public class lwhSer : DataService<mydb01>

{

public static void InitializeService(DataServiceConfiguration config)

{

config.SetEntitySetAccessRule("\*", EntitySetRights.All);

config.SetServiceOperationAccessRule("\*", ServiceOperationRights.All);

config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2;

}

// 第一个拦截器

[QueryInterceptor("people")]

public Expression<Func<people, bool>> ReturnPeopleFilter()

{

return p => p.country == "China";

}

// 第二个拦截器

[QueryInterceptor("people")]

public Expression<Func<people, bool>> ReturnPeopleFilter1()

{

return p => p.last\_name == "Liu";

}

}

}

对同一个实体集合定义两个拦截器： 他们的条件是“与”， 即同时满足两个条件。

**改变拦截器：**

[ChangeInterceptor("people")]

public void ChangeCountry(people e, UpdateOperations op)

{

if (op == UpdateOperations.Add)

{

e.country = e. country + ":" + " By Server:" + DateTime.Now.ToLongTimeString();

}

}

[ChangeInterceptor("people")]

public void ChangeCountry1(people e, UpdateOperations op)

{

if (op == UpdateOperations.Add)

{

e.country = e. country + ":" + DateTime.Now.ToOADate().ToString();

}

}

如果是累计叠加， 则最后值是两者运算出来的和：

――――――――――――――――――――――――――――――――――

[ChangeInterceptor("people")]

public void ChangeCountry(people e, UpdateOperations op)

{

if (op == UpdateOperations.Add)

{

e.first\_name = " By Server:" + DateTime.Now.ToLongTimeString();

}

}

[ChangeInterceptor("people")]

public void ChangeCountry1(people e, UpdateOperations op)

{

if (op == UpdateOperations.Add)

{

e.first\_name = " second:" + DateTime.Now.ToOADate().ToString();

}

}

如果是赋值， 则后面定义的覆盖前面定义的值。

使用拦截器的注意事项：

1. 如果有查询拦截器： 如果插入操作， 该值不符合查询拦截器， 则出错。 修改则没问题， 记录被过滤掉

svc.people np = new svc.people

{

first\_name = "William",

last\_name = "liu",

country = "USA" // 错误在此： 拦截器是 country == “China”

};

svc.street ns = new svc.street

{

street1 = "Burlington"

};

svc.street ns1 = new svc.street

{

street1 = "Burnaby"

};

svc.street ns2 = new svc.street

{

street1 = "Canada Way"

};

db.AddTopeople(np);

db.AddRelatedObject(np, "street", ns);

db.AddRelatedObject(np, "street", ns1);

db.AddRelatedObject(np, "street", ns2);

db.SaveChanges();

对于修改则没有问题， 只是记录将不在出现， 被过滤掉：

IEnumerable<svc.people> nnp = db.Execute<svc.people>(new Uri("people(145)", UriKind.Relative));

svc.people np = nnp.FirstOrDefault();

np.first\_name = "change:wm";

np.country = "Mexican"; // 修改， 并不符合拦截器的值， 所以会被过滤掉。

db.UpdateObject(np);

db.SaveChanges();

1. 如果有修改拦截器： 如果插入操作， 该值不符合查询拦截器， 则出错。 修改则没问题， 记录被过滤掉

[ChangeInterceptor("people")]

public void ChangeCountry(people e, UpdateOperations op)

{

if (op == UpdateOperations.Add)

{

e.country = e. country + ":" + " By Server:" + DateTime.Now.ToLongTimeString();

}

}

执行上面的插入操作，会出错。 因为值不符合查询拦截器的值。

对于修改操作则没有问题， 只是记录被过滤掉， 不再出现而已.

**查询语法:**

db = new svc.mydb01(new Uri("http://192.168.1.200/youtest/lwhSer.svc"));

DataServiceQuery<svc.people> nppp = db.people.AddQueryOption("$orderby", "first\_name desc")

.AddQueryOption("$skip",6)

.AddQueryOption("$top",3);

IEnumerable<svc.people> nnp = db.Execute<svc.people>(new Uri("people?$skip=7&$top=3", UriKind.Relative));

svc.people np = nnp.FirstOrDefault();

MessageBox.Show("Street count:" + np.first\_name);