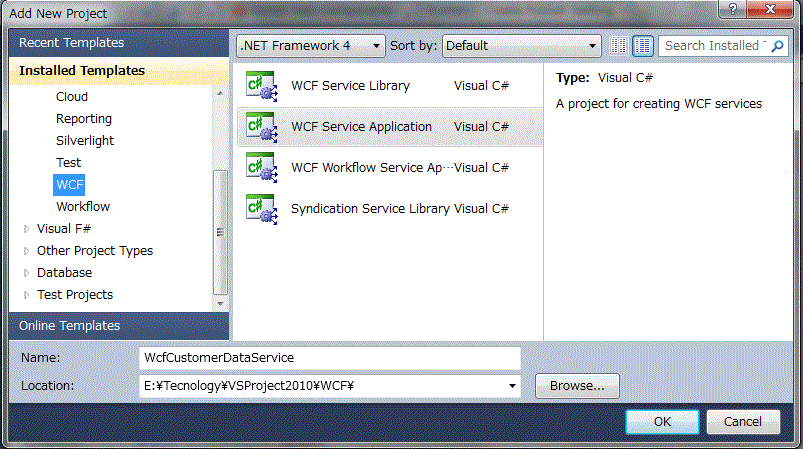
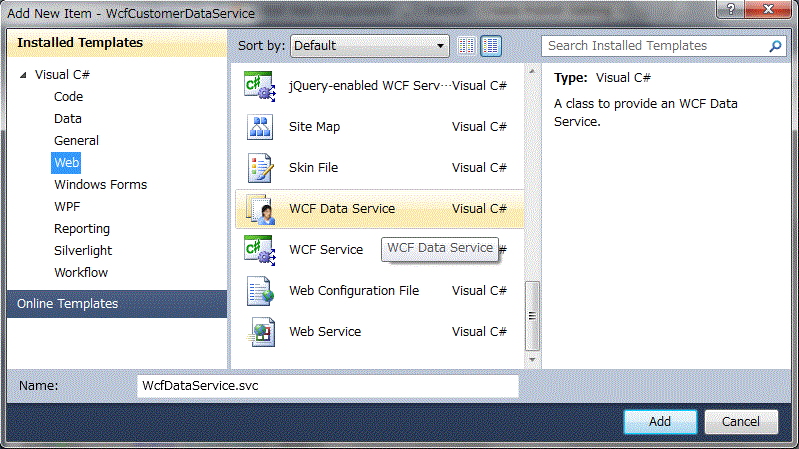
**[WCF实例 —— 自定义DataService数据模型(1)](http://blog.csdn.net/fangxinggood/article/details/6601385)**

分类： [[05] WCF](http://blog.csdn.net/fangxinggood/article/category/767778) 2011-07-13 01:34 2945人阅读 [评论](http://blog.csdn.net/fangxinggood/article/details/6601385#comments)(13) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/fangxinggood/article/details/6601385#report)

[wcf](http://blog.csdn.net/tag/details.html?tag=wcf)[service](http://blog.csdn.net/tag/details.html?tag=service)[class](http://blog.csdn.net/tag/details.html?tag=class)[string](http://blog.csdn.net/tag/details.html?tag=string)[reference](http://blog.csdn.net/tag/details.html?tag=reference)[application](http://blog.csdn.net/tag/details.html?tag=application)

ADO.NET DataService 默认是暴露EDM数据模型(e.g. Entity Framework) 以提供一种基于OData协议的数据服务，但也支持自定义的数据模型，可以选择别的数据源来实现一个DataService。当然作为自定义的数据模型， 查询需要返回实现 IQueryable 接口的对象，更新则需要实现 IUpdatable 接口。  
  
先创建一个 WCF Service Application  
  
删掉默认的 Service1.svc 和 IService.cs，添加一个 WcfDataService.svc 项目  
  
查看 SVC 文件的 Markup，你会发现它加上 DataServiceHostFactory：

<%@ ServiceHost Language="C#" Factory="System.Data.Services.DataServiceHostFactory, System.Data.Services, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" Service="WcfCustomerDataService.WcfDataService1" %>

如果在开发非IIS Host应用的时候，则使用 WebServiceHost 来寄宿服务。详细参考：[Hosting the Data Service (ADO.NET Data Services)](http://msdn.microsoft.com/en-us/library/cc668805%28v=VS.90%29.aspx)

**1.定义实体：实体里必须使用 DataServiceKeyAttribute 标注Key。否则DataService运行报错。**

**[csharp]** [view plaincopy](http://blog.csdn.net/fangxinggood/article/details/6601385)

1. [DataServiceKeyAttribute("OrderId")]
2. public class Order
3. {
4. public int OrderId { get; set; }
5. public string Customer { get; set; }
6. public IList<Item> Items { get; set; }
7. }
9. [DataServiceKeyAttribute("Product")]
10. public class Item
11. {
12. public string Product { get; set; }
13. public int Quantity { get; set; }
14. }

**2. 定义数据模型：**  
(1) 在 DataModel 的静态构造方法中，添加了一些初始数据，在这里你可以任意扩展数据的来源。  
(2) 定义了两个属性：Orders 和 Items， 它们返回 IQueryable<T> 的结果。

**[csharp]** [view plaincopy](http://blog.csdn.net/fangxinggood/article/details/6601385)

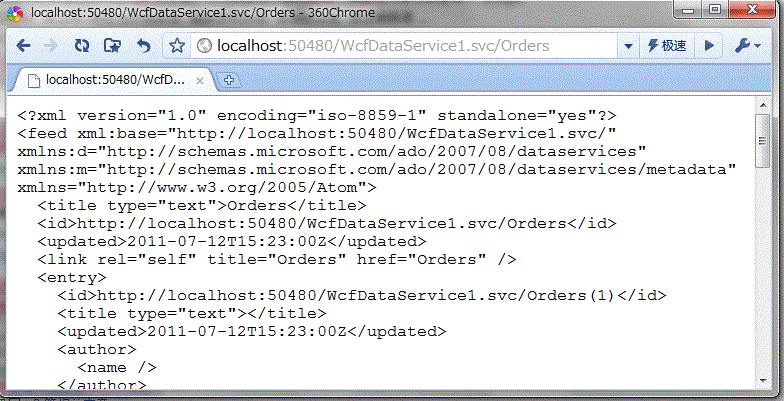
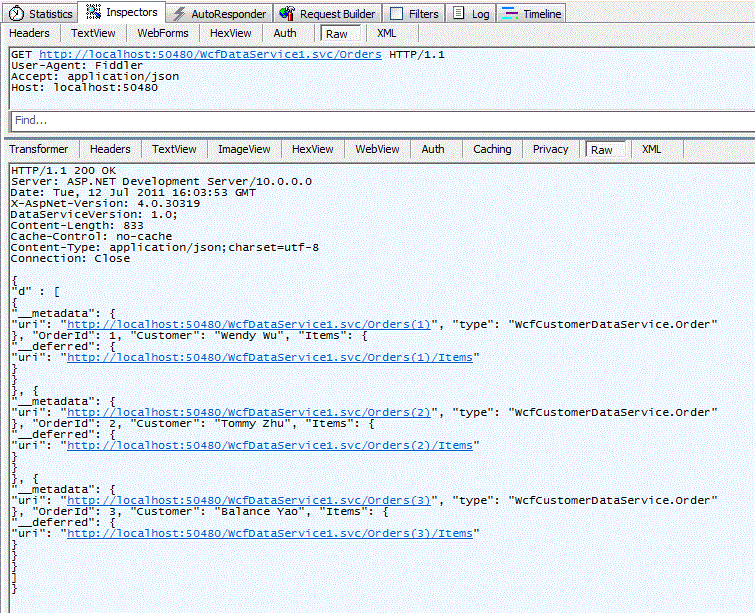
1. public class DataModel
2. {
3. #region Populate Service Data
4. static IList<Order> \_orders;
5. static IList<Item> \_items;
6. static DataModel()
7. {
8. \_orders = new List<Order> {
9. new Order(){ OrderId=1, Customer = "Wendy Wu", Items = new List<Item>()},
10. new Order(){ OrderId=2, Customer = "John Gu", Items = new List<Item>()},
11. new Order(){ OrderId=3, Customer = "Balance Yao", Items = new List<Item>()}
12. };
14. \_items = new List<Item> {
15. new Item(){ Product="Chang", Quantity = 4 },
16. new Item(){ Product="Aniseed Syrup", Quantity=5 },
17. new Item(){ Product="Toy", Quantity=7 },
18. new Item(){ Product="Car", Quantity=1 },
19. new Item(){ Product="Ball", Quantity=6}
20. };
22. \_orders[0].Items.Add(\_items[0]);
23. \_orders[1].Items.Add(\_items[1]);
24. \_orders[1].Items.Add(\_items[2]);
25. \_orders[2].Items.Add(\_items[3]);
26. \_orders[2].Items.Add(\_items[4]);
27. }
28. #endregion
30. public IQueryable<Order> Orders
31. {
32. get { return \_orders.AsQueryable<Order>(); }
33. }
35. public IQueryable<Item> Items
36. {
37. get { return \_items.AsQueryable<Item>(); }
38. }
39. }

**3. 实现Service**

**[csharp]** [view plaincopy](http://blog.csdn.net/fangxinggood/article/details/6601385)

1. public class WcfDataService1 : DataService<DataModel>
2. {
3. public static void InitializeService(DataServiceConfiguration config)
4. {
5. config.SetEntitySetAccessRule("\*", EntitySetRights.All);
6. config.SetServiceOperationAccessRule("\*", ServiceOperationRights.All);
7. config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2;
8. }
9. }

运行，在浏览器里输入 <http://localhost:50480/WcfDataService1.svc/Orders> 就能看到结果：返回Atom协议的XML数据。

  
PS：Data Service 通过客户端Request的**Accept** 来决定返回格式。  
比如我用 Fiddler 来查询，使用Accept: application/json 那么返回的就是Json格式的数据：  
****

当然除了在 DataModel 中实现属性的方式提供数据查询服务以外，还可以在 Service 中实现方法提供服务。  
比如在上面的 WcfDataService1 里加入下面的方法：调用时按照OData协议写Query

**[csharp]** [view plaincopy](http://blog.csdn.net/fangxinggood/article/details/6601385)

1. [WebGet]
2. [SingleResult]
3. public Order TopOrder()
4. {
5. return this.CurrentDataSource.Orders
6. .OrderByDescending(o => o.Items.Sum(i => i.Quantity)).First();
7. }

Data Service 对服务操作有一些具体的限制

* 此方法只能接受 [in] 参数。如果对参数进行定义，则每个参数的类型必须为基元类型。
* 此方法必须返回 void、IEnumerable<T>、IQueryable<T>、T 或基元类 (如整数或字符串)。T 必须为一个类，此类表示数据服务将公开的数据模型中的某个实体类型。若要支持查询选项（如排序、分页和筛选），服务操作方法应返回 IQueryable<T>。
* 必须用 [WebGet] 或 [WebInvoke] 属性为此方法添加标注。[WebGet] 使调用方能够通过使用 GET 请求调用此方法；[WebInvoke] 使调用方能够通过使用 PUT、POST 或 DELETE 请求调用此方法。
* 可 以用 SingleResultAttribute 为服务操作添加批注，指定此方法的返回值是一个实体而不是一个实体集。这一区别确定了生成的响应序列化。例如，当使用 AtomPub 序列化时，单个资源类型实例将表示为一个 entry 元素，而单个实例集将表示为一个 feed 元素。

4. 客户端调用  
创建一个Console工程，添加Service Reference 生成客户端代理，调用DataService:

**[csharp]** [view plaincopy](http://blog.csdn.net/fangxinggood/article/details/6601385)

1. static void Main(string[] args)
2. {
3. var svcUri = new Uri("http://localhost:50480/WcfDataService1.svc");
4. //Query
5. var ctx = new DataSvc.DataModel(svcUri);
6. var order = ctx.Orders.Where(o => o.OrderId == 3).First();
7. Console.WriteLine(order.Customer);
8. //Query by customer method 'TopOrder'
9. var topOrder = ctx.Execute<DataSvc.Order>(new Uri("/TopOrder", UriKind.Relative)).First();
10. Console.WriteLine(topOrder.Customer);
11. Console.Read();
12. }

[**WCF实例 —— 自定义DataService数据模型(2)**](http://blog.csdn.net/fangxinggood/article/details/6604446)

分类： [[05] WCF](http://blog.csdn.net/fangxinggood/article/category/767778) 2011-07-14 02:03 1954人阅读 [评论](http://blog.csdn.net/fangxinggood/article/details/6604446#comments)(20) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/fangxinggood/article/details/6604446#report)

[wcf](http://blog.csdn.net/tag/details.html?tag=wcf)[object](http://blog.csdn.net/tag/details.html?tag=object)[string](http://blog.csdn.net/tag/details.html?tag=string)[exception](http://blog.csdn.net/tag/details.html?tag=exception)[null](http://blog.csdn.net/tag/details.html?tag=null)[数据库](http://blog.csdn.net/tag/details.html?tag=%e6%95%b0%e6%8d%ae%e5%ba%93)

在前一篇 [DataService数据模型(1)](http://blog.csdn.net/fangxinggood/article/details/6601385)  里介绍了如何实现一个内存数据集合的 DataService，接下来研究一下如何实现更新操作。  
  
这里偷了个懒，直接让 DataModel 实现 [IDataServiceUpdateProvider](http://msdn.microsoft.com/en-us/library/system.data.services.providers.idataserviceupdateprovider%28VS.100%29.aspx) 接口，而 IDataServiceUpdateProvider 实际继承于 IUpdatable 接口。为了实现数据的增加，修改，删除，主要实现下面几个方法：

**[csharp]** [view plaincopy](http://blog.csdn.net/fangxinggood/article/details/6604446)

1. //增加
2. object CreateResource(string containerName, string fullTypeName);
3. //修改
4. void SetValue(object targetResource, string propertyName, object propertyValue);
5. //删除
6. void DeleteResource(object targetResource);
7. //保存
8. void SaveChanges();
9. //回滚
10. void ClearChanges();

PS：实体关联关系的一系列操作，比较复杂，之后讨论。

另外需要提到的是  IDataServiceUpdateProvider 接口设计中支持批量特性，这就允许在一个事务中一次更新很多的资源。换句话说可以在 SaveChangeds() 方法调用前，调用 IDataServiceUpdateProvider.SetValue() 或者 CreateResource() 多次。这似乎看起来比较简单，但是对接口的实现却影响很大。在某个方法实现上，不可以立即将请求反映到数据源上，而是记录所发生的事情并在最后一次性的提交所有操作。

如果数据存放在数据库中，那么数据库系统会自动的在事务中记录所有的命令操作(比如：Entity Framewok)。但在这个例子中使用了内存对象存放数据，所以需要记录在 SaveChanges() 前所发生的一切。  
  
因此在 DataModel 中，增加了 **List<Action> \_actions** 来记录 SaveChanges() 之前的操作(请求)。  
然后在 CreateResource, SetValue, DeleteResource 时都只是将操作暂存在 \_actions 里。  
SaveChanges 时 ForEach 所有暂存的 Action，一口气调用。 ClearChanges 则简单的把 \_actions 清空。

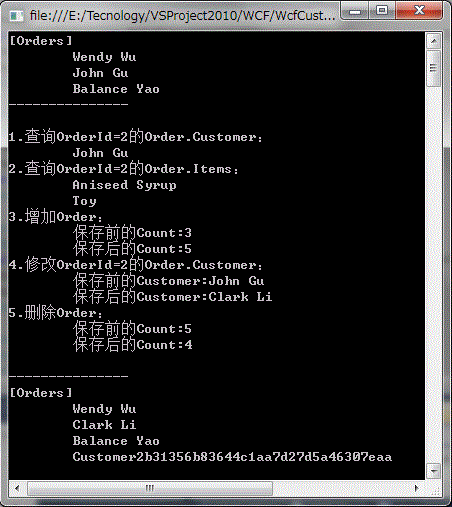
**[csharp]** [view plaincopy](http://blog.csdn.net/fangxinggood/article/details/6604446)

1. public class DataModel : IDataServiceUpdateProvider
2. {
3. #region Populate Service Data
4. static IList<Order> \_orders;
5. static IList<Item> \_items;
7. List<Action> \_actions = new List<Action>();
9. static DataModel()
10. {
11. \_orders = new List<Order> {
12. new Order(){ OrderId=1, Customer = "Wendy Wu", Items = new List<Item>()},
13. new Order(){ OrderId=2, Customer = "John Gu", Items = new List<Item>()},
14. new Order(){ OrderId=3, Customer = "Balance Yao", Items = new List<Item>()}
15. };
17. \_items = new List<Item> {
18. new Item(){ Product="Chang", Quantity = 4 },
19. new Item(){ Product="Aniseed Syrup", Quantity=5 },
20. new Item(){ Product="Toy", Quantity=7 },
21. new Item(){ Product="Car", Quantity=1 },
22. new Item(){ Product="Ball", Quantity=6 }
23. };
25. \_orders[0].Items.Add(\_items[0]);
26. \_orders[1].Items.Add(\_items[1]);
27. \_orders[1].Items.Add(\_items[2]);
28. \_orders[2].Items.Add(\_items[3]);
29. \_orders[2].Items.Add(\_items[4]);
30. }
31. #endregion
33. public IQueryable<Order> Orders
34. {
35. get { return \_orders.AsQueryable<Order>(); }
36. }
38. public IQueryable<Item> Items
39. {
40. get { return \_items.AsQueryable<Item>(); }
41. }
43. #region NotImplementedMethods
44. public void SetConcurrencyValues(object resourceCookie, bool? checkForEquality, IEnumerable<KeyValuePair<string, object>> concurrencyValues)
45. {
46. throw new NotImplementedException();
47. }
49. public void AddReferenceToCollection(object targetResource, string propertyName, object resourceToBeAdded)
50. {
51. //PropertyInfo pi = targetResource.GetType().GetProperty(propertyName);
52. //if (pi == null)
53. //    throw new Exception("Can't find property");
54. //IList collection = (IList)pi.GetValue(targetResource, null);
55. //collection.Add(resourceToBeAdded);
56. }
58. public void RemoveReferenceFromCollection(object targetResource, string propertyName, object resourceToBeRemoved)
59. {
60. //PropertyInfo pi = targetResource.GetType().GetProperty(propertyName);
61. //if (pi == null)
62. //    throw new Exception("Can't find property");
63. //IList collection = (IList)pi.GetValue(targetResource, null);
64. //collection.Remove(resourceToBeRemoved);
65. }
67. public void SetReference(object targetResource, string propertyName, object propertyValue)
68. {
69. //((IUpdatable)this).SetValue(targetResource, propertyName, propertyValue);
70. }
71. #endregion
73. public object CreateResource(string containerName, string fullTypeName)
74. {
75. Type t = Type.GetType(fullTypeName, true);
76. object resource = Activator.CreateInstance(t);
77. \_actions.Add(() =>
78. {
79. if (containerName == "Orders")
80. \_orders.Add(resource as Order);
81. if (containerName == "Items")
82. \_items.Add(resource as Item);
83. });
84. return resource;
85. }
87. public object GetResource(IQueryable query, string fullTypeName)
88. {
89. object resource = query.Cast<object>().SingleOrDefault();
91. // fullTypeName can be null for deletes
92. if (fullTypeName != null && resource.GetType().FullName != fullTypeName)
93. throw new ApplicationException("Unexpected type for this resource.");
94. return resource;
95. }
97. public void DeleteResource(object targetResource)
98. {
99. if (targetResource.GetType() == typeof(Order))
100. {
101. \_orders.Remove(targetResource as Order);
102. return;
103. }
104. if (targetResource.GetType() == typeof(Item))
105. {
106. \_items.Remove(targetResource as Item);
107. return;
108. }
109. throw new NotSupportedException("Type not found");
110. }
112. public void SetValue(object targetResource, string propertyName, object propertyValue)
113. {
114. \_actions.Add(() =>
115. {
116. targetResource
117. .GetType()
118. .GetProperties()
119. .Single(p => p.Name == propertyName)
120. .GetSetMethod()
121. .Invoke(targetResource, new[] { propertyValue });
122. });
123. }
125. public object GetValue(object targetResource, string propertyName)
126. {
127. var value = targetResource
128. .GetType()
129. .GetProperties()
130. .Single(p => p.Name == propertyName)
131. .GetGetMethod()
132. .Invoke(targetResource, new object[] { });
133. return value;
134. }
136. public object ResetResource(object resource)
137. {
138. return resource;
139. }
141. public object ResolveResource(object resource)
142. {
143. return resource;
144. }
146. public void SaveChanges()
147. {
148. \_actions.ForEach(a => a());
149. // Handler Primary Keys
150. var newOrders = \_orders.Where(o => o.OrderId == 0).ToList();
151. var maxOrderId = \_orders.Max(o => o.OrderId);
152. foreach (var order in newOrders)
153. order.OrderId = ++maxOrderId;
154. }
156. public void ClearChanges()
157. {
158. \_actions.Clear();
159. }
160. }

客户端调用：

**[csharp]** [view plaincopy](http://blog.csdn.net/fangxinggood/article/details/6604446)

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Data.Services.Client;
7. namespace ConsoleApplication1
8. {
9. class Program
10. {
11. static void Main(string[] args)
12. {
13. var svcUri = new Uri("http://localhost:50480/WcfDataService1.svc");
14. var ctx = new DataSvc.DataModel(svcUri);
16. Console.WriteLine("[Orders]");
17. ctx.Orders.ToList().ForEach(o => Console.WriteLine("\t{0}", o.Customer));
18. Console.WriteLine("---------------");
19. Console.WriteLine();
21. var order = ctx.Orders.Where(o => o.OrderId == 2).First();
22. Console.WriteLine("1.查询OrderId=2的Order.Customer：");
23. Console.WriteLine("\t{0}", order.Customer);
24. // 括号里的"2"是对应Order主键而不是Index
25. var items = ctx.Execute<DataSvc.Item>(new Uri("/Orders(2)/Items", UriKind.Relative)).ToList();
26. Console.WriteLine("2.查询OrderId=2的Order.Items：");
27. items.ForEach(it => Console.WriteLine("\t{0}", it.Product));
29. Console.WriteLine("3.增加Order：");
30. Console.WriteLine("\t保存前的Count:" + ctx.Orders.Count());
31. var addOrder1 = new DataSvc.Order { Customer = "Customer" + Guid.NewGuid().ToString("N") };
32. ctx.AddToOrders(addOrder1);
33. var addOrder2 = new DataSvc.Order { Customer = "Customer" + Guid.NewGuid().ToString("N") };
34. ctx.AddToOrders(addOrder2);
35. ctx.SaveChanges();
36. Console.WriteLine("\t保存后的Count:" + ctx.Orders.Count());
38. Console.WriteLine("4.修改OrderId=2的Order.Customer：");
39. Console.WriteLine("\t保存前的Customer:" + order.Customer);
40. order.Customer = "Clark Li";
41. ctx.SaveChanges();
42. var query = ctx.Orders.Where(o => o.OrderId == 2).First();
43. Console.WriteLine("\t保存后的Customer:" + query.Customer);
45. Console.WriteLine("5.删除Order：");
46. Console.WriteLine("\t保存前的Count:" + ctx.Orders.Count());
47. var delOrder1 = ctx.Orders.Where(o => o.OrderId == addOrder1.OrderId).First();
48. ctx.DeleteObject(delOrder1);
49. ctx.SaveChanges();
50. Console.WriteLine("\t保存后的Count:" + ctx.Orders.Count());
52. Console.WriteLine();
53. Console.WriteLine("[Orders]");
54. ctx.Orders.ToList().ForEach(o => Console.WriteLine("\t{0}", o.Customer));
55. Console.WriteLine("---------------");
57. Console.Read();
58. }
59. }
60. }

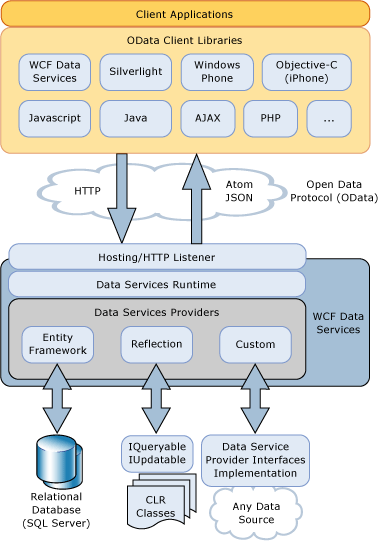
运行结果：  


[**WCF实例 —— 自定义DataService数据模型(3)**](http://blog.csdn.net/fangxinggood/article/details/6624345)

分类： [[05] WCF](http://blog.csdn.net/fangxinggood/article/category/767778) 2011-07-22 00:43 1548人阅读 [评论](http://blog.csdn.net/fangxinggood/article/details/6624345#comments)(6) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/fangxinggood/article/details/6624345#report)

[wcf](http://blog.csdn.net/tag/details.html?tag=wcf)[object](http://blog.csdn.net/tag/details.html?tag=object)[string](http://blog.csdn.net/tag/details.html?tag=string)[exception](http://blog.csdn.net/tag/details.html?tag=exception)[query](http://blog.csdn.net/tag/details.html?tag=query)[inheritance](http://blog.csdn.net/tag/details.html?tag=inheritance)

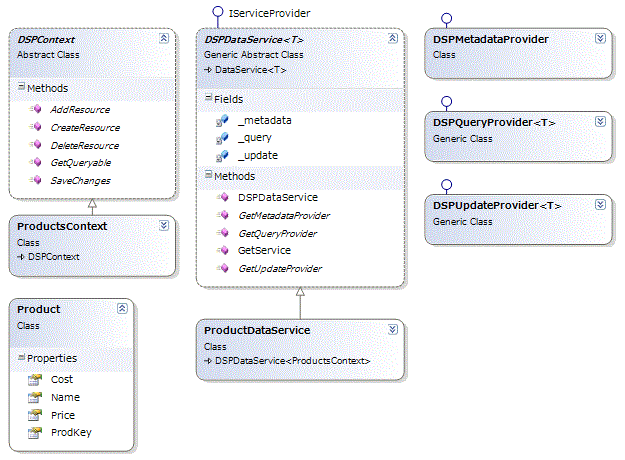
接着研究 WCF DataService 的实现机制。先上张MSDN的DataService体系结构图：

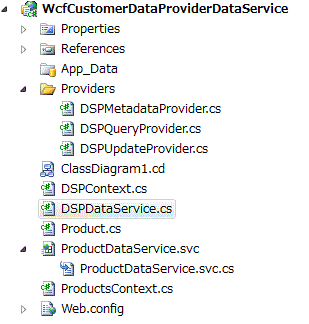
  
【1.用反射数据提供程序实现的DataService】

在前面的文章里直接定义了 WcfService 继承于 DataService<T> ， 它使用的是：ReflectionServiceProvider 提供程序通过使用反射来推断数据模型。  
以下列表显示了反射提供程序如何推断数据模型：

* 实体容器 – 将数据作为可返回 IQueryable 实例的属性公开的类。对基于反射的数据模型进行寻址时，实体容器表示服务的根。对给定命名空间仅支持一个实体容器类。
* 实体集 – 返回 IQueryable 实例的属性视为实体集。在查询中，将把实体集作为资源直接进行寻址。实体容器中只有一个属性才能返回给定类型的 IQueryable 实例。
* 实体类型 – 实体集返回的 IQueryable 的类型 T。反射提供程序将作为继承层次结构一部分的类转换为等效的实体类型层次结构
* 实体键 – 作为实体类型的每个数据类必须具有一个键属性。该属性具有 DataServiceKeyAttribute 特性 ([DataServiceKeyAttribute])。

【2.自定义Provider实现的DataService】

如何自定义一个 ServiceProvider 来实现 DataService  (具体参考：<http://blogs.msdn.com/b/alexj/archive/tags/dsp/>)  
  




DSPDataService<T> 继承于 DataService<T> 实现 IServiceProvider 接口，而 ProductDataService 继承于它，返回 IDataServiceMetadataProvider, IDataServiceQueryProvider, IDataServiceUpdateProvider 的实例。它是 DataService 的入口。而这3个接口顾名思义分别提供：元数据，查询，更新的功能。

**DSPDataService.cs**

**[csharp]** [view plaincopy](http://blog.csdn.net/fangxinggood/article/details/6624345)

1. public abstract class DSPDataService<T> : DataService<T>, IServiceProvider where T : DSPContext
2. {
3. private IDataServiceMetadataProvider \_metadata;
4. private IDataServiceQueryProvider \_query;
5. private IDataServiceUpdateProvider \_update;
7. public DSPDataService()
8. {
9. \_metadata = GetMetadataProvider(typeof(T));
10. \_query = GetQueryProvider(\_metadata);
11. \_update = GetUpdateProvider(\_metadata, \_query);
12. }
14. public object GetService(Type serviceType)
15. {
16. if (serviceType == typeof(IDataServiceMetadataProvider))
17. return \_metadata;
18. else if (serviceType == typeof(IDataServiceQueryProvider))
19. return \_query;
20. else if (serviceType == typeof(IDataServiceUpdateProvider))
21. return \_update;
22. else
23. return null;
24. }
26. public abstract IDataServiceMetadataProvider GetMetadataProvider(Type dataSourceType);
27. public abstract IDataServiceQueryProvider GetQueryProvider(IDataServiceMetadataProvider metadata);
28. public abstract IDataServiceUpdateProvider GetUpdateProvider(IDataServiceMetadataProvider metadata, IDataServiceQueryProvider query);
29. }

**DSPContext.cs**

**[csharp]** [view plaincopy](http://blog.csdn.net/fangxinggood/article/details/6624345)

1. public abstract class DSPContext
2. {
3. public abstract IQueryable GetQueryable(ResourceSet set);
4. public abstract object CreateResource(ResourceType resourceType);
5. public abstract void AddResource(ResourceType resourceType, object resource);
6. public abstract void DeleteResource(object resource);
7. public abstract void SaveChanges();
8. }

**可以看到上面两个都是抽象类，具体实现类是：ProductDataService.svc 和 ProductsContext.cs。这儿 ProductsContext.cs 就是自定义数据模型的容器。  
  
ProductDataService.svc.cs**  
(1) 重写了 DataService.CreateDataSource 返回 ProductsContext 实例  
(2) 重写了 DSPDataService.GetMetadataProvider 返回 DSPMetadataProvider 实例  
(3) 重写了 DSPDataService.GetQueryProvider 返回 DSPQueryProvider 实例  
(4) 重写了 DSPDataService.GetUpdateProvider 返回 DSPUpdateProvider 实例  
注意：GetMetadataProvider 部分的代码，这里手动添加了对应的 ResourceType(及包含的 ResourceProperty)

**[csharp]** [view plaincopy](http://blog.csdn.net/fangxinggood/article/details/6624345)

1. using System;
2. using System.Collections.Generic;
3. using System.Data.Services;
4. using System.Data.Services.Common;
5. using System.Linq;
6. using System.ServiceModel.Web;
7. using System.Web;
8. using System.Data.Services.Providers;
10. namespace WcfCustomerDataProviderDataService
11. {
12. public class ProductDataService : DSPDataService<ProductsContext>
13. {
14. // This method is called only once to initialize service-wide policies.
15. public static void InitializeService(DataServiceConfiguration config)
16. {
17. config.SetEntitySetAccessRule("\*", EntitySetRights.All);
18. config.SetServiceOperationAccessRule("\*", ServiceOperationRights.All);
19. config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2;
20. }
22. protected override ProductsContext CreateDataSource()
23. {
24. ProductsContext context = new ProductsContext();
25. context.Products.Add(
26. new Product
27. {
28. ProdKey = 1,
29. Name = "Bovril",
30. Cost = 4.35M,
31. Price = 6.49M
32. });
33. context.Products.Add(
34. new Product
35. {
36. ProdKey = 2,
37. Name = "Marmite",
38. Cost = 4.97M,
39. Price = 7.21M
40. });
41. return context;
42. }
44. public override IDataServiceMetadataProvider GetMetadataProvider(Type dataSourceType)
45. {
46. DSPMetadataProvider metadata = new DSPMetadataProvider();
47. var productType = new ResourceType(
48. typeof(Product), // CLR type backing this Resource
49. ResourceTypeKind.EntityType, // Entity, ComplexType etc
50. null, // BaseType
51. "Namespace", // Namespace
52. "Product", // Name
53. false // Abstract?
54. );
55. var prodKey = new ResourceProperty(
56. "ProdKey",
57. ResourcePropertyKind.Key |
58. ResourcePropertyKind.Primitive,
59. ResourceType.GetPrimitiveResourceType(typeof(int))
60. );
61. var prodName = new ResourceProperty(
62. "Name",
63. ResourcePropertyKind.Primitive,
64. ResourceType.GetPrimitiveResourceType(typeof(string))
65. );
66. var prodPrice = new ResourceProperty(
67. "Price",
68. ResourcePropertyKind.Primitive,
69. ResourceType.GetPrimitiveResourceType(typeof(Decimal))
70. );
71. productType.AddProperty(prodKey);
72. productType.AddProperty(prodName);
73. productType.AddProperty(prodPrice);
75. metadata.AddResourceType(productType);
76. metadata.AddResourceSet(new ResourceSet("Products", productType));
77. return metadata;
78. }
80. public override IDataServiceQueryProvider GetQueryProvider(IDataServiceMetadataProvider metadata)
81. {
82. return new DSPQueryProvider<ProductsContext>(metadata);
83. }
85. public override IDataServiceUpdateProvider GetUpdateProvider(IDataServiceMetadataProvider metadata, IDataServiceQueryProvider query)
86. {
87. return new DSPUpdateProvider<ProductsContext>(metadata, query);
88. }
89. }
90. }

**ProductsContext.cs**实现对数据的增，删，改，查

**[csharp]** [view plaincopy](http://blog.csdn.net/fangxinggood/article/details/6624345)

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Web;
5. using System.Data.Services.Providers;
7. namespace WcfCustomerDataProviderDataService
8. {
9. public class ProductsContext : DSPContext
10. {
11. private List<Product> \_products = new List<Product>();
13. public override IQueryable GetQueryable(ResourceSet set)
14. {
15. if (set.Name == "Products")
16. return Products.AsQueryable();
17. throw new NotSupportedException(string.Format("{0} not found", set.Name));
18. }
20. public List<Product> Products
21. {
22. get { return \_products; }
23. }
25. public override object CreateResource(ResourceType resourceType)
26. {
27. if (resourceType.InstanceType == typeof(Product))
28. {
29. return new Product();
30. }
31. throw new NotSupportedException(string.Format("{0} not found", resourceType.FullName));
32. }
34. public override void AddResource(ResourceType resourceType, object resource)
35. {
36. if (resourceType.InstanceType == typeof(Product))
37. {
38. Product p = resource as Product;
39. if (p != null)
40. {
41. Products.Add(p);
42. return;
43. }
44. }
45. throw new NotSupportedException("Type not found");
46. }
48. public override void DeleteResource(object resource)
49. {
50. if (resource.GetType() == typeof(Product))
51. {
52. Products.Remove(resource as Product);
53. return;
54. }
55. throw new NotSupportedException("Type not found");
56. }
58. public override void SaveChanges()
59. {
60. var prodKey = Products.Max(p => p.ProdKey);
61. foreach (var prod in Products.Where(p => p.ProdKey == 0))
62. prod.ProdKey = ++prodKey;
63. }
65. }
66. }

**DSPMetadataProvider.cs**

**[csharp]** [view plaincopy](http://blog.csdn.net/fangxinggood/article/details/6624345)

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Web;
5. using System.Data.Services.Providers;
6. using System.Reflection;
7. using System.Data.Services;
9. namespace WcfCustomerDataProviderDataService
10. {
11. public class DSPMetadataProvider : IDataServiceMetadataProvider
12. {
13. private Dictionary<string, ResourceType> \_resourceTypes
14. = new Dictionary<string, ResourceType>();
15. private Dictionary<string, ResourceSet> \_resourceSets
16. = new Dictionary<string, ResourceSet>();
17. //private Dictionary<string, ServiceOperation> serviceOperations
18. //            = new Dictionary<string, ServiceOperation>();
20. public DSPMetadataProvider() { }
22. public void AddResourceType(ResourceType type)
23. {
24. type.SetReadOnly();
25. \_resourceTypes.Add(type.FullName, type);
26. }
28. public void AddResourceSet(ResourceSet set)
29. {
30. set.SetReadOnly();
31. \_resourceSets.Add(set.Name, set);
32. }
34. public string ContainerName
35. {
36. get { return "Container"; }
37. }
39. public string ContainerNamespace
40. {
41. get { return "Namespace"; }
42. }
44. public IEnumerable<ResourceType> GetDerivedTypes(ResourceType resourceType)
45. {
46. // We don't support type inheritance yet
47. yield break;
48. }
50. public ResourceAssociationSet GetResourceAssociationSet(
51. ResourceSet resourceSet,
52. ResourceType resourceType,
53. ResourceProperty resourceProperty)
54. {
55. throw new NotImplementedException("No relationships.");
56. }
58. public bool HasDerivedTypes(ResourceType resourceType)
59. {
60. // We don’t support inheritance yet
61. return false;
62. }
64. public IEnumerable<ResourceSet> ResourceSets
65. {
66. get { return this.\_resourceSets.Values; }
67. }
69. public IEnumerable<ServiceOperation> ServiceOperations
70. {
71. // No service operations yet
72. get { yield break; }
73. }
75. public bool TryResolveResourceSet(string name, out ResourceSet resourceSet)
76. {
77. return \_resourceSets.TryGetValue(name, out resourceSet);
78. }
80. public bool TryResolveResourceType(string name, out ResourceType resourceType)
81. {
82. return \_resourceTypes.TryGetValue(name, out resourceType);
83. }
85. public bool TryResolveServiceOperation(string name, out ServiceOperation serviceOperation)
86. {
87. // No service operations are supported yet
88. serviceOperation = null;
89. return false;
90. }
92. public IEnumerable<ResourceType> Types
93. {
94. get { return this.\_resourceTypes.Values; }
95. }
96. }
97. }

**DSPQueryProvider.cs**

**[csharp]** [view plaincopy](http://blog.csdn.net/fangxinggood/article/details/6624345)

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Web;
5. using System.Data.Services.Providers;
6. using System.Reflection;
7. using System.Globalization;
9. namespace WcfCustomerDataProviderDataService
10. {
11. public class DSPQueryProvider<T> : IDataServiceQueryProvider where T : DSPContext
12. {
13. private T \_currentDataSource;
14. private IDataServiceMetadataProvider \_metadata;
16. public DSPQueryProvider(IDataServiceMetadataProvider metadata)
17. {
18. // TODO: Complete member initialization
19. \_metadata = metadata;
20. }
22. public object CurrentDataSource
23. {
24. get { return \_currentDataSource; }
25. set { \_currentDataSource = value as T; }
26. }
28. public object GetOpenPropertyValue(object target, string propertyName)
29. {
30. throw new NotImplementedException();
31. }
33. public IEnumerable<KeyValuePair<string, object>> GetOpenPropertyValues(object target)
34. {
35. throw new NotImplementedException();
36. }
38. public object GetPropertyValue(object target, ResourceProperty resourceProperty)
39. {
40. throw new NotImplementedException();
41. }
43. public IQueryable GetQueryRootForResourceSet(ResourceSet resourceSet)
44. {
45. return \_currentDataSource.GetQueryable(resourceSet);
46. }
48. public ResourceType GetResourceType(object target)
49. {
50. Type type = target.GetType();
51. return \_metadata.Types.Single(t => t.InstanceType == type);
52. }
54. public object InvokeServiceOperation(ServiceOperation serviceOperation, object[] parameters)
55. {
56. return null;
57. }
59. public bool IsNullPropagationRequired
60. {
61. get { throw new NotImplementedException(); }
62. }
63. }
64. }

**DSPUpdateProvider.cs**

**[csharp]** [view plaincopy](http://blog.csdn.net/fangxinggood/article/details/6624345)

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Web;
5. using System.Data.Services.Providers;
7. namespace WcfCustomerDataProviderDataService
8. {
9. public class DSPUpdateProvider<T> : IDataServiceUpdateProvider where T : DSPContext
10. {
11. private IDataServiceMetadataProvider \_metadata;
12. private IDataServiceQueryProvider \_query;
13. private List<Action> \_actions;
15. public DSPUpdateProvider(IDataServiceMetadataProvider metadata, IDataServiceQueryProvider query)
16. {
17. \_metadata = metadata;
18. \_query = query;
19. \_actions = new List<Action>();
20. }
22. public T GetContext()
23. {
24. return (\_query.CurrentDataSource as T);
25. }
27. public void SetConcurrencyValues(object resourceCookie, bool? checkForEquality, IEnumerable<KeyValuePair<string, object>> concurrencyValues)
28. {
29. throw new NotImplementedException();
30. }
32. public void AddReferenceToCollection(object targetResource, string propertyName, object resourceToBeAdded)
33. {
34. throw new NotImplementedException();
35. }
37. public void ClearChanges()
38. {
39. \_actions.Clear();
40. }
42. public object CreateResource(string containerName, string fullTypeName)
43. {
44. ResourceType type = null;
45. if (\_metadata.TryResolveResourceType(fullTypeName, out type))
46. {
47. var context = GetContext();
48. var resource = context.CreateResource(type);
49. \_actions.Add(() => context.AddResource(type, resource));
50. return resource;
51. }
52. throw new Exception(string.Format("Type {0} not found", fullTypeName));
53. }
55. public void DeleteResource(object targetResource)
56. {
57. \_actions.Add(() => GetContext().DeleteResource(targetResource));
58. }
60. public object GetResource(IQueryable query, string fullTypeName)
61. {
62. var enumerator = query.GetEnumerator();
63. if (!enumerator.MoveNext())
64. throw new Exception("Resource not found");
65. var resource = enumerator.Current;
66. if (enumerator.MoveNext())
67. throw new Exception("Resource not uniquely identified");
69. if (fullTypeName != null)
70. {
71. ResourceType type = null;
72. if (!\_metadata.TryResolveResourceType(fullTypeName, out type))
73. throw new Exception("ResourceType not found");
74. if (!type.InstanceType.IsAssignableFrom(resource.GetType()))
75. throw new Exception("Unexpected resource type");
76. }
77. return resource;
78. }
80. public object GetValue(object targetResource, string propertyName)
81. {
82. var value = targetResource
83. .GetType()
84. .GetProperties()
85. .Single(p => p.Name == propertyName)
86. .GetGetMethod()
87. .Invoke(targetResource, new object[] { });
88. return value;
89. }
91. public void RemoveReferenceFromCollection(object targetResource, string propertyName, object resourceToBeRemoved)
92. {
93. throw new NotImplementedException();
94. }
96. public object ResetResource(object resource)
97. {
98. \_actions.Add(() => ReallyResetResource(resource));
99. return resource;
100. }
102. public void ReallyResetResource(object resource)
103. {
104. // Create an new 'blank' instance of the resource
105. var clrType = resource.GetType();
106. ResourceType resourceType = \_metadata.Types.Single(t => t.InstanceType == clrType);
107. var resetTemplate = GetContext().CreateResource(resourceType);
109. // Copy non-key property values from the 'blank' resource
110. foreach (var prop in resourceType.Properties.Where(p => (p.Kind & ResourcePropertyKind.Key)
111. != ResourcePropertyKind.Key))
112. {
113. // Obviously for perf reasons you could might want to
114. // cache the result of these reflection calls.
115. var clrProp = clrType.GetProperties().Single(p => p.Name == prop.Name);
116. var defaultPropValue = clrProp.GetGetMethod().Invoke(resetTemplate, new object[] { });
117. clrProp.GetSetMethod().Invoke(resource, new object[] { defaultPropValue });
118. }
119. }
121. public object ResolveResource(object resource)
122. {
123. return resource;
124. }
126. public void SaveChanges()
127. {
128. \_actions.ForEach(a => a());
129. GetContext().SaveChanges();
130. }
132. public void SetReference(object targetResource, string propertyName, object propertyValue)
133. {
134. throw new NotImplementedException();
135. }
137. public void SetValue(object targetResource, string propertyName, object propertyValue)
138. {
139. \_actions.Add(
140. () => ReallySetValue(
141. targetResource,
142. propertyName,
143. propertyValue)
144. );
145. }
147. public void ReallySetValue(object targetResource, string propertyName, object propertyValue)
148. {
149. targetResource
150. .GetType()
151. .GetProperties()
152. .Single(p => p.Name == propertyName)
153. .GetSetMethod()
154. .Invoke(targetResource, new[] { propertyValue });
155. }
156. }
157. }

(2011/7/25 Updated)

遗留问题：通过自定义Provider并不能实现类似使用反射数据提供程序实现的DataService的完整功能，至少比如像  http://localhost:51396/ProductDataService.svc/Products()?$filter=Name eq 'fx'

这 样的查询功能都不支持。根据 DataService<T> 源码分析还需要实现比如：TryResolveServiceOperation，InvokeServiceOperation 以及 IDataServicePagingProvider 接口 等等。DataService<T> 水深啊，需要进一步研究。