演练：使用 WCF 序列化 POCO 代理（实体框架）

不能直接通过 Windows Communication Foundation (WCF) 对 POCO 代理类型进行序列化或反序列化，原因是 [DataContractSerializer](http://technet.microsoft.com/zh-cn/magazine/system.runtime.serialization.datacontractserializer%28VS.100%29.aspx) 序列化引擎只能序列化和反序列化已知类型。 代理类型是一个未知类型。 有关更多信息，请参见[使用 POCO 实体（实体框架）](http://technet.microsoft.com/zh-cn/magazine/dd456853%28VS.100%29.aspx)主题中的“序列化 POCO 代理”一节。 若要将 POCO 代理序列化为 POCO 实体，请在序列化过程中使用 [ProxyDataContractResolver](http://technet.microsoft.com/zh-cn/magazine/system.data.objects.proxydatacontractresolver%28VS.100%29.aspx) 类将代理类型映射到 POCO 类型。

本主题中的示例演示了如何通过定义一个在内部使用 **ProxyDataContractResolver** 将代理类型映射到 POCO 类型的特性类（将被应用于服务操作），指示 **DataContractSerializer** 在服务操作中使用 **ProxyDataContractResolver** 类。 此示例还演示了如何将此特性类与作为 WCF 应用程序中服务协定的一部分的方法相关联。

本主题中的示例使用在[如何：定义 POCO 实体（实体框架）](http://technet.microsoft.com/zh-cn/magazine/dd456872%28VS.100%29.aspx)中定义的 POCO 类，以及在[如何：自定义建模和映射文件以使用自定义对象（实体框架）](http://technet.microsoft.com/zh-cn/magazine/bb738625%28VS.100%29.aspx)中定义的基于 AdventureWorks 的数据模型。

**创建包含 POCO 类的类库项目。**

1. 创建一个名为 **POCOAdventureWorksModel** 的新类库项目。
2. 移除添加到项目中的默认源代码文件。
3. 添加一个名为 **AdventureWorksModel** 的空模型。 若要创建空模型，请参见[How to: Create a New .edmx File](http://technet.microsoft.com/zh-cn/magazine/cc716703%28VS.100%29.aspx)主题中的“创建空的 .edmx 文件”部分。 按照[自定义 AdventureWorks .edmx 文件（实体框架）](http://technet.microsoft.com/zh-cn/magazine/ee853287%28VS.100%29.aspx)中的步骤修改该模型。
4. 禁用 .edmx 文件的代码生成。 在 [ADO.NET Entity Data Model Designer](http://technet.microsoft.com/zh-cn/magazine/cc716685%28VS.100%29.aspx)（实体设计器）中打开 .edmx 文件。 右击设计器图面并选择“属性”。 在**“属性”**窗口中选择**“代码生成策略”**属性，并选择 **None**。 如果**“属性”**窗口不可见，请按 F4。
5. 将 app.config 文件添加到类库项目。 右击**“POCOAdventureWorksModel”**，单击**“添加”**，再单击**“新建项”**。
6. 在**“添加新项”**对话框中，选择**“常规”**模板并选择**“应用程序配置文件”**。 将以下位于 **configuration** 标记之间的代码复制在应用程序配置文件中。 如有必要，修改 **Data Source** 的值。

Xml

<connectionStrings>

<add name="AdventureWorksEntities"

connectionString="metadata=res://\*/AdventureWorksModel.csdl|res://\*/AdventureWorksModel.ssdl|res://\*/AdventureWorksModel.msl;

provider=System.Data.SqlClient;provider connection string=&quot;

Data Source=(local);Initial Catalog=AdventureWorks;

Integrated Security=True;MultipleActiveResultSets=True&quot;"

providerName="System.Data.EntityClient" />

</connectionStrings>

1. 添加对 **System.Runtime.Serialization** 库的引用。 WCF 的在可序列化的实体类型上使用的 **DataContract** 和 **DataMember** 特性均需要此库。
2. 向名为 POCOClasses 的项目添加一个新类。 将[基于 AdventureWorks 模型的可序列化的 POCO 类](http://technet.microsoft.com/zh-cn/magazine/ee853286%28VS.100%29.aspx)中的代码添加到文件。 这包含实体类型和对象上下文定义。
3. 编译该项目。

**创建和配置 WCF 项目。**

1. 在名为 **POCOAdventureWorksService** 的类库项目所在的解决方案中创建一个 **WCF Service Application** 项目。
2. 添加对 **System.Data.Entity** 库的引用。
3. 添加对 **POCOAdventureWorksModel** 项目（在其中定义模型的项目）的引用。
4. 将连接字符串添加到 .config 文件，以便实体框架 运行时可以找到元数据。 在您的 **POCOAdventureWorksModel** 项目中打开 app.config 文件，复制 **connectionStrings** 元素，并将其添加为 Web.config 文件的 **configuration** 元素的子元素。
5. 创建一个新类并将其命名为 ApplyDataContractResolverAttribute。
6. 将以下命名空间添加到文件的开头：

C#

using System.Data.Objects;

using System.ServiceModel.Description;

using System.ServiceModel.Channels;

1. 将为新类生成的代码替换为以下代码：

C#

public class ApplyDataContractResolverAttribute : Attribute, IOperationBehavior

{

public ApplyDataContractResolverAttribute()

{

}

public void AddBindingParameters(OperationDescription description, BindingParameterCollection parameters)

{

}

public void ApplyClientBehavior(OperationDescription description, System.ServiceModel.Dispatcher.ClientOperation proxy)

{

DataContractSerializerOperationBehavior dataContractSerializerOperationBehavior =

description.Behaviors.Find<DataContractSerializerOperationBehavior>();

dataContractSerializerOperationBehavior.DataContractResolver =

new ProxyDataContractResolver();

}

public void ApplyDispatchBehavior(OperationDescription description, System.ServiceModel.Dispatcher.DispatchOperation dispatch)

{

DataContractSerializerOperationBehavior dataContractSerializerOperationBehavior =

description.Behaviors.Find<DataContractSerializerOperationBehavior>();

dataContractSerializerOperationBehavior.DataContractResolver =

new ProxyDataContractResolver();

}

public void Validate(OperationDescription description)

{

// Do validation.

}

}

1. 打开服务接口文件。 默认情况下，该文件名为 **IService1**。
2. 将 POCOAdventureWorksModel 命名空间添加到文件的开头。 在此命名空间中定义的 POCO 类型。
3. 将定义服务接口文件的代码替换为以下代码：

C#

[ServiceContract]

public interface IService1

{

[OperationContract]

[ApplyDataContractResolver]

void UpdateOrder(Order updated);

[OperationContract]

[ApplyDataContractResolver]

Order GetOrder(int OrderID);

}

1. 打开服务源代码。 默认情况下，该代码名为 Service1.srv.cs（或 .vb）
2. 将 POCOAdventureWorksModel 命名空间添加到文件的开头。
3. 将定义服务类的代码替换为以下代码：

C#

public class Service1 : IService1

{

public void UpdateOrder(Order updated)

{

using (POCOAdventureWorksEntities context =

new POCOAdventureWorksEntities())

{

// Attach the original order to the context by querying the database.

// Alternatively, you can require that the updated object be returned along with the original object from the client.

// This means the client would need to clone the original object.

Order original = context.Orders.SingleOrDefault(o => o.SalesOrderID == updated.SalesOrderID);

// Apply changes to the order object.

context.Orders.ApplyCurrentValues(updated);

context.SaveChanges();

}

}

public Order GetOrder(int OrderID)

{

using (POCOAdventureWorksEntities context = new POCOAdventureWorksEntities())

{

// You can disable the proxy creation

// by setting context.ContextOptions.ProxyCreationEnabled to false

context.ContextOptions.LazyLoadingEnabled = false;

// The order was created as a POCO proxy object.

// But it will be recieved on the client as a pure POCO.

Order order = context.Orders.SingleOrDefault(o => o.SalesOrderID == OrderID);

return order;

}

}

}

1. 编译该项目。

**测试服务。**

1. 创建一个控制台应用程序。 为项目名称键入 **POCOAdventureWorksTest**。
2. 添加对 **POCOAdventureWorksModel** 项目的引用。
3. 添加对 **POCOAdventureWorksService** 服务的引用。 在**“解决方案资源管理器”**中，右击引用文件夹并选择**“添加服务引用”**。
4. 打开 app.config 文件，并将连接字符串添加到该文件中。 打开 **POCOAdventureWorksModel** 的 app.config 文件，复制 **connectionStrings** 元素，并将其添加为 Web.config 文件的 **configuration** 元素的子元素。
5. 打开包含主函数的文件。
6. 将以下命名空间（服务和 POCO 类型在其中定义）添加到文件的开头：

C#

Service1Client client = new Service1Client();

int orderId = 43680;

Order order = client.GetOrder(orderId);

Console.WriteLine(order.DueDate);

// Modify order.

order.DueDate = DateTime.Now;

// Update order in the database.

client.UpdateOrder(order);

1. 将代码替换为以下代码。 请注意，尽管该服务能够对 POCO 代理进行序列化，但是客户端接收的是纯 POCO 对象。

C#

Service1Client client = new Service1Client();

int orderId = 43680;

Order order = client.GetOrder(orderId);

Console.WriteLine(order.DueDate);

// Modify order.

order.DueDate = DateTime.Now;

// Update order in the database.

client.UpdateOrder(order);