# [Entity Framework在WCF中序列化的问题](http://www.cnblogs.com/Gyoung/p/3153875.html)

**问题描述**

如果你在WCF中用Entity Framework来获取数据并返回实体对象，那么对下面的错误一定不陌生。

*接收对 http://localhost:5115/ReService.svc 的 HTTP 响应时发生错误。这可能是由于服务终结点绑定未使用 HTTP 协议造成的。*

*这还可能是由于服务器中止了 HTTP 请求上下文(可能由于服务关闭)所致。有关详细信息，请参见服务器日志。*

这就是因为在返回数据的时候，序列化失败，导致WCF服务自动停止了。

**为什么会序列化失败**

为了方便说明，我们先做个示例来重现这个错误。

默认情况下，Entity Framework为了支持它的一些高级特性(延迟加载等)，默认将自动生成代理类是设置为true，即

public MyContext()

{

this.Configuration.ProxyCreationEnabled = true;

}

这样，如果我们的实体中包含其它实体的导航属性，则EF会自动的为这个实体生成代理类。

[复制代码](javascript:void(0);)

[DataContract(IsReference=true)]

public class Student

{

public Student()

{

this.Teachers = new HashSet<Teacher>();

}

[DataMember]

public int ID { get; set; }

[DataMember]

public virtual string Name { get; set; }

[DataMember]

public virtual ICollection<Teacher> Teachers { get; set; }

}

[DataContract(IsReference = true)]

public class Teacher

{

[DataMember]

public int ID { get; set; }

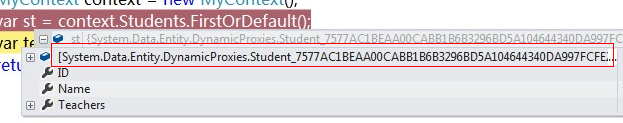
[DataMember]

public virtual string Name { get; set; }

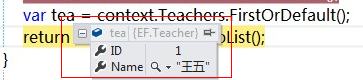
}

[复制代码](javascript:void(0);)

观察上面两个实体，Student中有对Teacher的导航属性，而Teacher则没有。我们看看通过EF对获取这两个对象有什么不同的情况



我们可以看到EF为Student生成了值为System.Data.Entity.DynamicProxies.Student\_...的代理实体



而对于Teacher，返回的就是我们所定义的实体。

如果我们在WCF中分别定义一个契约，来返回这两个实体会怎么样呢？

[OperationContract]

Student GetStudent();

[OperationContract]

Teacher GetTeacher();

实现方法

[复制代码](javascript:void(0);)

public Student GetStudent()

{

using (MyContext context = new MyContext())

{

return context.Students.FirstOrDefault();

}

}

public Teacher GetTeacher()

{

using (MyContext context = new MyContext())

{

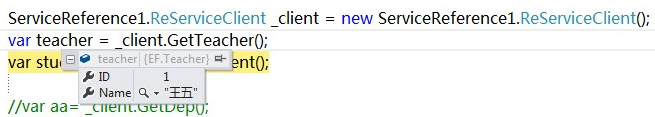
return context.Teachers.FirstOrDefault();

}

}

[复制代码](javascript:void(0);)

调用 WCF进行测试，我们可以很好的得到GetTeacher()的值，如图



但是，当调用GetStudent()方法，从服务端返回结果到客户端时确报错了。



嗯，没错，就是刚开始我说的那个错误。但，这是为什么呢。我们明明在Student中加了DataContract和DataMember关键字啊。

原因就是EF自动为Student生成了代理类，WCF序列化的其实是EF生成的那个代理类，而不是我们自己定义的Student，而代理类并没有标识这是一个可以序列化的实体。

**解决方法**

**1.禁用代理类**

既然原因是EF生成了代理类，那我们把它禁用了就可以了嘛。也很简单，只要将生成代理的配置设置为false即可。

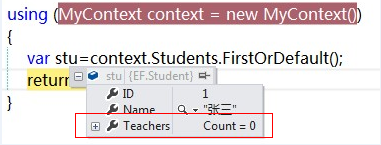
public MyContext()

{

this.Configuration.ProxyCreationEnabled = false;

}

禁用后，看看通过EF获取Student是怎么样的。



没错，代理类没了，但是我们不能直接通过导航属性来获取Teacher了。这可是杀敌一千，自损八百啊。有没有更好的办法呢？

**2 反序列化**

既然代理类是由实体序列化而来的，我们就可以在返回数据前将代理类序列化成我们所需要的实体。

[复制代码](javascript:void(0);)

public Student GetStudent()

{

using (MyContext context = new MyContext())

{

var stu=context.Students.FirstOrDefault();

var serializer = new DataContractSerializer(typeof(Student), new DataContractSerializerSettings()

{

DataContractResolver = new ProxyDataContractResolver()

});

using (var stream = new MemoryStream())

{

// 反序列化

serializer.WriteObject(stream, stu);

stream.Seek(0, SeekOrigin.Begin);

var newStu = (Student)serializer.ReadObject(stream);

return newStu;

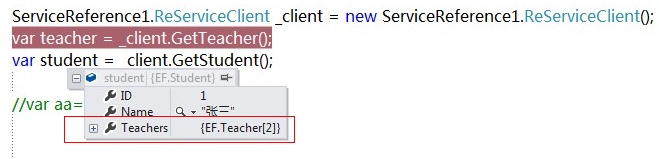
}

}

}

[复制代码](javascript:void(0);)

通过这个方法，再测试一下.



不错，没有报错，并且成功的得到了我们想要的结果。

但每个方法都要这样序列化一下，是不是很麻烦，有没有更好的方法。

答案肯定有，我们可以通过自定义Attribute，加在服务契约上面，标识通过这个服务返回的方法都要进行反序列化。

[复制代码](javascript:void(0);)

public class ProxyDataContractResolver: DataContractResolver

{

private XsdDataContractExporter \_exporter = new XsdDataContractExporter();

public override Type ResolveName( string typeName, string typeNamespace, Type declaredType,

DataContractResolver knownTypeResolver)

{

return knownTypeResolver.ResolveName(

typeName, typeNamespace, declaredType, null);

}

public override bool TryResolveType(Type dataContractType,Type declaredType,

DataContractResolver knownTypeResolver,

out XmlDictionaryString typeName,

out XmlDictionaryString typeNamespace)

{

Type nonProxyType = ObjectContext.GetObjectType(dataContractType);

if (nonProxyType != dataContractType)

{

// Type was a proxy type, so map the name to the non-proxy name

XmlQualifiedName qualifiedName = \_exporter.GetSchemaTypeName(nonProxyType);

XmlDictionary dictionary = new XmlDictionary(2);

typeName = new XmlDictionaryString(dictionary,

qualifiedName.Name, 0);

typeNamespace = new XmlDictionaryString(dictionary,

qualifiedName.Namespace, 1);

return true;

}

else

{

// Type was not a proxy type, so do the default

return knownTypeResolver.TryResolveType(

dataContractType,

declaredType,

null,

out typeName,

out typeNamespace);

}

}

}

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

public class ApplyProxyDataContractResolverAttribute : Attribute, IOperationBehavior

{

public void AddBindingParameters(OperationDescription description, BindingParameterCollection parameters)

{

}

public void ApplyClientBehavior(OperationDescription description, ClientOperation proxy)

{

DataContractSerializerOperationBehavior

dataContractSerializerOperationBehavior =

description.Behaviors.Find<DataContractSerializerOperationBehavior>();

dataContractSerializerOperationBehavior.DataContractResolver = new ProxyDataContractResolver();

}

public void ApplyDispatchBehavior(OperationDescription description, DispatchOperation dispatch)

{

DataContractSerializerOperationBehavior

dataContractSerializerOperationBehavior = description.Behaviors.Find<DataContractSerializerOperationBehavior>();

dataContractSerializerOperationBehavior.DataContractResolver = new ProxyDataContractResolver();

}

public void Validate(OperationDescription description)

{

}

}

[复制代码](javascript:void(0);)

类ApplyProxyDataContractResolverAttribute就是我们想要的结果。现在我们只要在定义服务契约的时候，加上ApplyProxyDataContractResolver关键字就可以了。

[复制代码](javascript:void(0);)

[OperationContract]

[ApplyProxyDataContractResolver]

Student GetStudent();

[OperationContract]

[ApplyProxyDataContractResolver]

Teacher GetTeacher();

[复制代码](javascript:void(0);)

**扩展**

对于继承类的序列化，要在基类用KnownType属性来标识

[复制代码](javascript:void(0);)

[KnownType(typeof(ClassB))]

[KnownType(typeof(ClassA))]

[DataContract]

public class BaseClass

{

}

[DataContract]

public class ClassA : BaseClass

{

}

[DataContract]

public class ClassB : BaseClass

{

}

[复制代码](javascript:void(0);)

PS：虽然这样可以解决问题，但是多一层序列化会影响效率，希望EF的后续版本可以解决问题吧。