عنوان: **Data Contracts and Circular References**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۳/۲۰ ۸:۲۰
آدرس: www.dotnettips.info
برچسب‌ها: WCF, Serialization, Circular Reference, DataContract

تشریح مسئله : در DataContractSerializer قابلیتی به عنوان سریالایز کردن objectها به صورت درختی وجود داردکه اصطلاحا به اون Circular References گفته می‌شود در این پست قصد دارم روش پیاده سازی، به همراه مزایای استفاده از این روش رو توضیح بدم.
نکته : آشنایی با مفاهیم اولیه WCF برای درک بهتر مطالب الزامی است.
در ابتدا لازم است تا مدل برنامه را تعریف کنیم. ابتدا یک پروژه از نوع WCF Service Application ایجاد کنید و مدل زیر را بسازید.

Employee#

```
[DataContract]
    public class Employee
    {
        [DataMember]
        public string Name { get; set; }

        [DataMember]
        public Employee Manager { get; set; }
    }
```

Department#

```
[DataContract]
    public class Department
    {
        [DataMember]
        public string DeptName { get; set; }

        [DataMember]
        public List<Employee> Staff { get; set; }
    }
```

در مدل Employee یک خاصیت از نوع خود کلاس Employee وجود دارد که برای پیاده سازی مدل به صورت درختی است. در مدل Department هم لیستی از کارمندان دپارتمان را ذخیره می‌کنیم و قصد داریم این مدل رو از سمت سرور به کلاینت انتقال دهیم و نوع سریالایز کردن WCF رو در این مورد مشاهده کنیم. ابتدا سرویس و Contract مربوطه را می‌نویسیم.

Contract#

```
[ServiceContract]
    public interface IDepartmentService
    {
        [OperationContract]
        Department GetOneDepartment();
    }
```

Service#

```
public class DepartmentService : IDepartmentService
    {
        public Department GetOneDepartment()
        {
            List<Employee> listOfEmployees = new List<Employee>();

              var masoud = new Employee() { Name = "Masoud" };
              var saeed = new Employee() { Name = "Saeed", Manager = masoud };
              var peyman = new Employee() { Name = "Peyman", Manager = saeed };
```

```
            var mostafa = new Employee() { Name = "Mostafa", Manager = saeed };

            return new Department() { DeptName = "IT", Staff = new List<Employee>() { masoud, saeed,
peyman, mostafa } };
        }
    }
```

همانطور که در سرویس بالا مشخص است لیستی از کارمندان ساخته شده که خود این لیست به صورت درختی است و بعضی از کارمندان به عنوان مدیر کارمند دیگر تعیین شد است. حال برای دریافت اطلاعات سمت کلاینت یک پروژه از نوع Console ایجاد کنید و از روش AddServiceReference سرویس مورد نظر را اضافه کنید و کدهای زیر را در کلاس Program کپی کنید.

```
class Program
    {
        static void Main( string[] args )
        {
            DepartmentServiceClient client = new DepartmentServiceClient();

            var result = client.GetOneDepartment();
            WriteDataToFile( result );

            Console.ReadKey();
        }

        private static void WriteDataToFile( Department data )
        {
            DataContractSerializer dcs = new DataContractSerializer( typeof( Department ) );
            var ms = new MemoryStream();
            dcs.WriteObject( ms, data );
            ms.Seek( 0, SeekOrigin.Begin );
            var sr = new StreamReader( ms );
            var xml = sr.ReadToEnd();
            string filePath = @"d:\\data.xml";
            if ( !File.Exists( filePath ) )
            {
                File.Create( filePath );
            }
                using ( TextWriter writer = new StreamWriter( filePath ) )
                {
                    writer.Write( xml );
                }
        }
```

یک متد به نام WriteDataToFile نوشتم که اطلاعات Department رو به فرمت Xml در فایل ذخیره می‌کند. بعد از اجرای برنامه خروجی مورد نظر در فایل Xml به صورت زیر است.

```
<Department xmlns="http://schemas.datacontract.org/2004/07/Service"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Name>IT</Name>
  <Staff>
    <Employee>
      <Manager i:nil="true"/>
      <Name>Masoud</Name>
    </Employee>
    <Employee>
      <Manager>
        <Manager i:nil="true"/>
        <Name>Masoud</Name>
      </Manager>
      <Name>Saeed</Name>
    </Employee>
    <Employee>
      <Manager>
        <Manager>
          <Manager i:nil="true"/>
          <Name>Masoud</Name>
        </Manager>
        <Name>Saeed</Name>
      </Manager>
      <Name>Peyman</Name>
    </Employee>
    <Employee>
      <Manager>
        <Manager>
```

```
          <Manager i:nil="true"/>
          <Name>Masoud</Name>
        </Manager>
        <Name>Saeed</Name>
      </Manager>
      <Name>Mostafa</Name>
    </Employee>
  </Staff>
</Department>
```

در فایل بالا مشاهده می‌کنید که تعداد تکرار Masoud به اندازه تعداد استفاده اون در Department است. در این قسمت قصد داریم که از Circular Referencing موجود در DataContractSerializer استفاده کنیم. برای این کار کافیست از خاصیت IsReference موجود در DataContract استفاده کنیم. پس مدل Employee به صورت زیر تغییر میباید:

```
[DataContract( IsReference = true )]
    public class Employee
    {
        [DataMember]
        public string Name { get; set; }

        [DataMember]
        public Employee Manager { get; set; }
    }
```

پروژه رو دوباره Run کنید و فایل xml ساخته شده به صورت زیر تغییر می‌کند.

```
<Department xmlns="http://schemas.datacontract.org/2004/07/Service"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Name>IT</Name>
  <Staff>
    <Employee z:Id="i1" xmlns:z="http://schemas.microsoft.com/2003/10/Serialization/">
      <Manager i:nil="true"/>
      <Name>Masoud</Name>
    </Employee>
    <Employee z:Id="i2" xmlns:z="http://schemas.microsoft.com/2003/10/Serialization/">
      <Manager z:Ref="i1"/>
      <Name>Saeed</Name>
    </Employee>
    <Employee z:Id="i3" xmlns:z="http://schemas.microsoft.com/2003/10/Serialization/">
      <Manager z:Ref="i2"/>
      <Name>Peyman</Name>
    </Employee>
    <Employee z:Id="i4" xmlns:z="http://schemas.microsoft.com/2003/10/Serialization/">
      <Manager z:Ref="i2"/>
      <Name>Mostafa</Name>
    </Employee>
  </Staff>
</Department>
```

کاملا واضح است که تعداد Masoud به عنوان Employee فقط یک بار است و از z:ref برای ارتباط بین Objectها استفاده می‌شود. در این روش فقط یک بار هر object سریالاز می‌شود و هر جا که نیاز به استفاده از object مربوطه باشد فقط یک ارجاع به آن خواهد شد.

مزایا : استفاده از این روش در هنگام عمل سریالایز داده‌های زیاد و زمانی که تعداد Objectهای موجود در ObjectGraph زیاد باشد باعث افزایش کارایی و سرعت انجام عملیات سریالایز می‌شود.