C#基础系列：序列化效率比拼 - 文章 - 伯乐在线

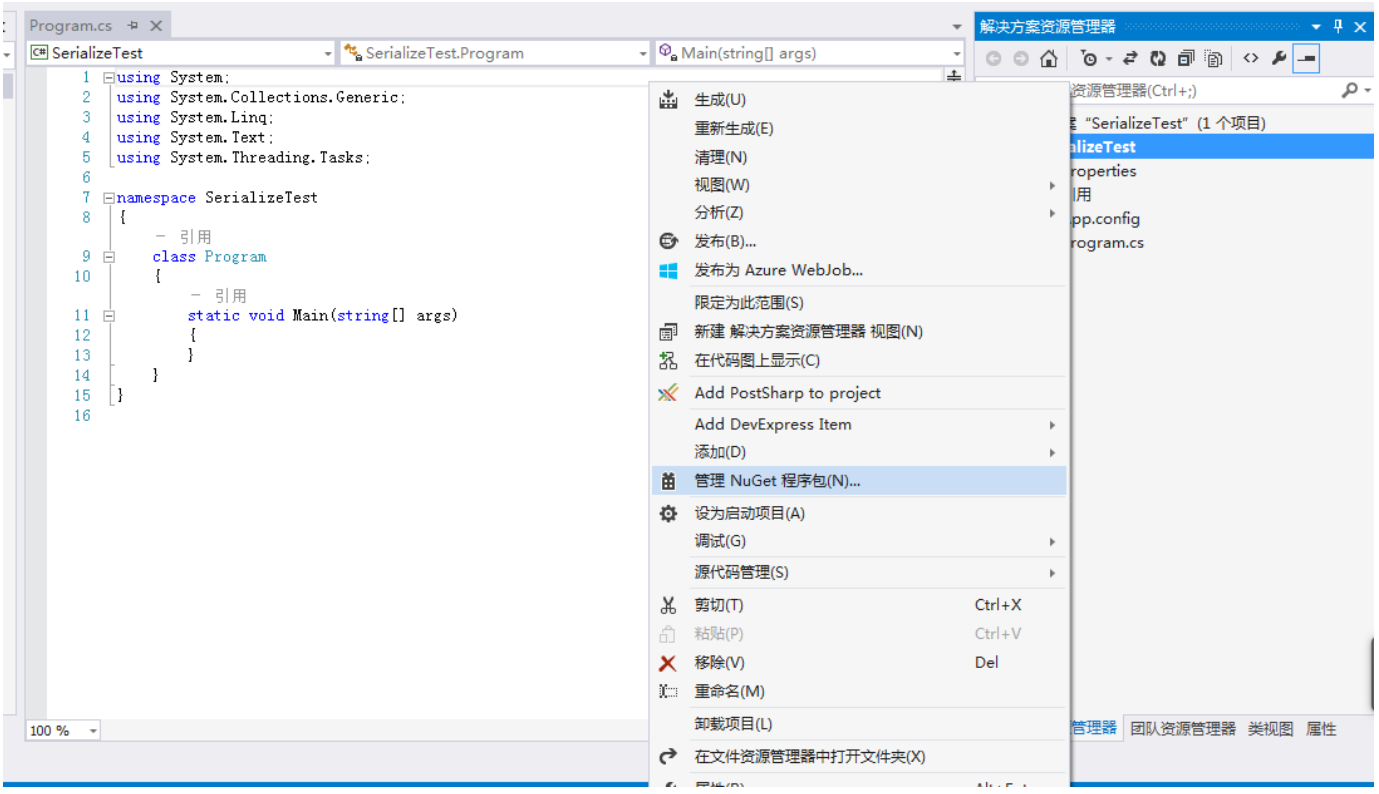
前言：作为开发人员，对象的序列化恐怕难以避免。楼主也是很早以前就接触过序列化，可是理解都不太深刻，对于用哪种方式去做序列化更是随波逐流——项目中原来用的什么方式照着用就好了。可是这么多年自己对于这东西还是挺模糊的，今天正好有时间，就将原来用过的几种方式总结了下，也算是做一个记录，顺便做了下性能测试。楼主算了下，从使用序列化到现在，用到的无非下面几种方式：(1)JavaScriptSerializer方式;(2)DataContract方式;(3)Newtonsoft.Json.

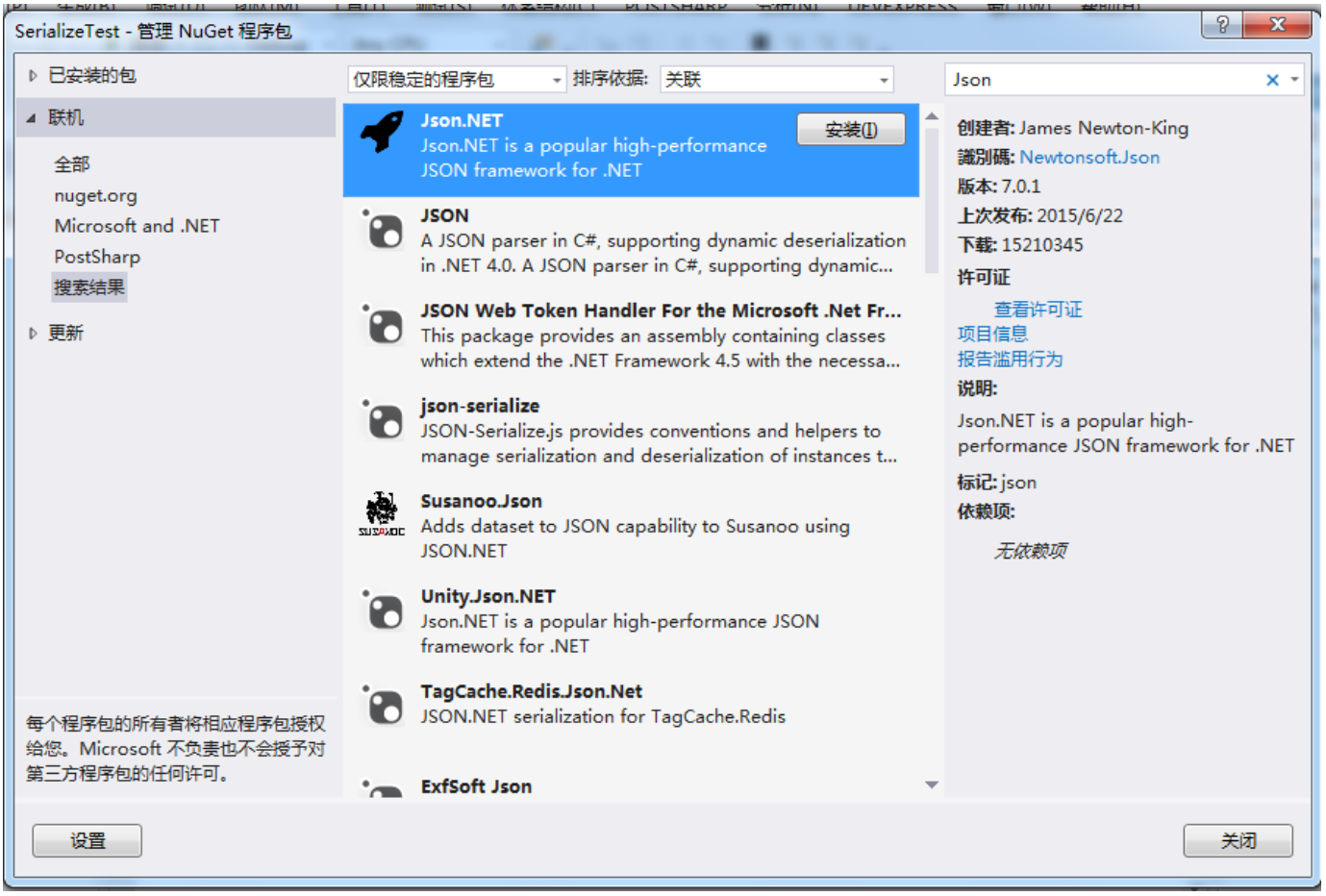
1、准备工作：要对这三种方式分别作测试，必须要将相应的内库引用进来。

（1）JavaScriptSerializer这个类是.Net内置的，属于System.Web.Script.Serialization这个命名空间下面。需要引用System.Web.Extensions这个dll。

（2）DataContract方式也是.net内置的，主要使用的DataContractJsonSerializer这个类，属于System.Runtime.Serialization.Json这个命名空间。需要引用System.Runtime.Serialization这个dll。

（3）Newtonsoft.Json是第三方的dll，但是Visual Studio 对它做了很好的支持。使用方式有两种：一种是去网上下载最新的dll，然后添加引用即可；第二种是直接使用NuGet安装这个包。方式如下：





按照步骤安装即可。

2、类库准备完毕，还需要提供几个通用的方法。自己分别封装了JavaScriptSerializer和DataContract方式两个方法，代码如下：

C#

```
        #region DataContract序列化
        public static class DataContractExtensions
        {
                /// <summary>
                /// 将对象转化为Json字符串
                /// </summary>
                /// <typeparam name="T">对象类型</typeparam>
                /// <param name="instanse">对象本身</param>
                /// <returns>JSON字符串</returns>
                public static string ToJsonString<T>(this T instanse)
                {
                        try
                        {
                                DataContractJsonSerializer js = new
DataContractJsonSerializer(typeof(T));
                                using (MemoryStream ms = new MemoryStream())
                                {
                                        js.WriteObject(ms, instanse);
```

```
                                ms.Flush();
                                ms.Seek(0, SeekOrigin.Begin);
                                StreamReader sr = new StreamReader(ms);
                                return sr.ReadToEnd();
                        }
                }
                catch
                {
                        return String.Empty;
                }
        }
        /// <summary>
        /// 将字符串转化为JSON对象，如果转换失败，返回default(T)
        /// </summary>
        /// <typeparam name="T">对象类型</typeparam>
        /// <param name="s">字符串</param>
        /// <returns>转换值</returns>
        public static T ToJsonObject<T>(this string s)
        {
                try
                {
                        DataContractJsonSerializer js = new
DataContractJsonSerializer(typeof(T));
                        using (MemoryStream ms = new MemoryStream())
                        {
                                StreamWriter sw = new StreamWriter(ms);
                                sw.Write(s);
                                sw.Flush();
                                ms.Seek(0, SeekOrigin.Begin);
                                return (T)js.ReadObject(ms);
                        }
                }
                catch
                {
                        return default(T);
                }
        }
}
#endregion
#region JavaScriptSerializer方式序列化
public static class JavascriptExtentions
{
```

```
                public static string ToScriptJsonString<T>(this T instanse)
                {
                        try
                        {
                                JavaScriptSerializer js = new JavaScriptSerializer();
                                return js.Serialize(instanse);
                        }
                        catch
                        {
                                return String.Empty;
                        }
                }
                public static T ToScriptJsonObject<T>(this string s)
                {
                        try
                        {
                                JavaScriptSerializer js = new JavaScriptSerializer();
                                return js.Deserialize<T>(s);
                        }
                        catch
                        {
                                return default(T);
                        }
                }
        }
        #endregion
```

C#

```
public class Newtonsoft_Common
        {
                #region 序列化
                // 将对象(包含集合对象)序列化为Json
                public static string SerializeObjToJson(object obj)
                {
                        string strRes = string.Empty;
                        try
                        {
                                strRes = JsonConvert.SerializeObject(obj);
                        }
                        catch
                        { }
                        return strRes;
                }
```

```
//将xml转换为json
public static string SerializeXmlToJson(System.Xml.XmlNode node)
{
        string strRes = string.Empty;
        try
        {
                strRes = JsonConvert.SerializeXmlNode(node);
        }
        catch
        { }
        return strRes;
}
//支持Linq格式的xml转换
public static string SerializeXmlToJson(System.Xml.Linq.XNode node)
{
        string strRes = string.Empty;
        try
        {
                strRes = JsonConvert.SerializeXNode(node);
        }
        catch
        { }
        return strRes;
}
#endregion
#region 反序列化
//将json反序列化为实体对象(包含DataTable和List<>集合对象)
public static T DeserializeJsonToObj<T>(string strJson)
{
        T oRes = default(T);
        try
        {
                oRes = JsonConvert.DeserializeObject<T>(strJson);
        }
        catch
        { }
        return oRes;
}
//将Json数组转换为实体集合
public static List<T> JsonLstToObjs<T>(List<string> lstJson)
{
        List<T> lstRes = new List<T>();
```

```
                try
                {
                        foreach (var strObj in lstJson)
                        {
                                //将json反序列化为对象
                                var oRes = JsonConvert.DeserializeObject<T>
(strObj);
                                lstRes.Add(oRes);
                        }
                }
                catch
                { }
                return lstRes;
        }
        #endregion
}
```

C#

```
public static List<Person> GetPersons()
        {
                var lstRes = new List<Person>();
                for (var i = 0; i < 50000; i++)
                {
                        var oPerson = new Person();
                        oPerson.Name = "李雷" + i;
                        oPerson.Age = 20;
                        oPerson.IsChild = i % 5 == 0 ? true : false;
                        oPerson.Test1 = "aaaaaa";
                        oPerson.Test2 = i.ToString() ;
                        oPerson.Test3 = i.ToString();
                        oPerson.Test4 = i.ToString();
                        oPerson.Test5 = i.ToString();
                        oPerson.Test6 = i.ToString();
                        oPerson.Test7 = i.ToString();
                        oPerson.Test8 = i.ToString();
                        oPerson.Test9 = i.ToString();
                        oPerson.Test10 = i.ToString();
                        lstRes.Add(oPerson);
                }
                return lstRes;
        }
        public static DataTable GetDataTable()
        {
```

```
                var dt = new DataTable("dt");
                dt.Columns.Add("Age", Type.GetType("System.Int32"));
                dt.Columns.Add("Name", Type.GetType("System.String"));
                dt.Columns.Add("Sex", Type.GetType("System.String"));
                dt.Columns.Add("IsChild", Type.GetType("System.Boolean"));
                for (var i = 0; i < 1000; i++)
                {
                        DataRow dr = dt.NewRow();
                        dr["Age"] = i + 1;
                        dr["Name"] = "Name" + i;
                        dr["Sex"] = i % 2 == 0 ? "男" : "女";
                        dr["IsChild"] = i % 5 > 0 ? true : false;
                        dt.Rows.Add(dr);
                }
                return dt;
        }
```

3、测试开始之前，先介绍下，本篇测试分别通过强类型对象和若类型的DataTable分别去做序列化和反序列化的测试。测试代码：

C#

```
static void Main(string[] args)
        {
                #region 强类型对象
                var lstRes = GetPersons();
                #region JavaScriptSerializer序列化方式
                var lstScriptSerializeObj = new List<string>();
                Stopwatch sp_script = new Stopwatch();
                sp_script.Start();
                foreach (var oPerson in lstRes)
                {
                        lstScriptSerializeObj.Add(oPerson.ToScriptJsonString<Person
>());
                }
                sp_script.Stop();
                Console.WriteLine("JavaScriptSerializer序列化方式序列化" +
lstScriptSerializeObj.Count + "个对象耗时：" + sp_script.ElapsedMilliseconds + "毫秒");
                lstRes.Clear();
                Stopwatch sp_script1 = new Stopwatch();
                sp_script1.Start();
                foreach (var oFrameSerializeObj in lstScriptSerializeObj)
                {
                        lstRes.Add(oFrameSerializeObj.ToScriptJsonObject<Person>
```

```
());
                        }
                        sp_script1.Stop();
                        Console.WriteLine("JavaScriptSerializer序列化方式反序列化" +
lstScriptSerializeObj.Count + "个对象耗时：" + sp_script1.ElapsedMilliseconds + "毫秒");
                        #endregion
                        #region DataContract序列化方式
                        var lstFrameSerializeObj = new List<string>();
                        Stopwatch sp = new Stopwatch();
                        sp.Start();
                        foreach (var oPerson in lstRes)
                        {
                                lstFrameSerializeObj.Add(oPerson.ToJsonString<Person>());
                        }
                        sp.Stop();
                        Console.WriteLine("DataContract序列化方式序列化" +
lstFrameSerializeObj.Count + "个对象耗时：" + sp.ElapsedMilliseconds + "毫秒");
                        lstRes.Clear();
                        Stopwatch sp1 = new Stopwatch();
                        sp1.Start();
                        foreach (var oFrameSerializeObj in lstFrameSerializeObj)
                        {
                                lstRes.Add(oFrameSerializeObj.ToJsonObject<Person>());
                        }
                        sp1.Stop();
                        Console.WriteLine("DataContract序列化方式反序列化" +
lstFrameSerializeObj.Count + "个对象耗时：" + sp1.ElapsedMilliseconds + "毫秒");
                        #endregion
                        #region Newtonsoft
                        var lstNewtonsoftSerialize = new List<string>();
                        Stopwatch sp2 = new Stopwatch();
                        sp2.Start();
                        foreach (var oPerson in lstRes)
                        {
                                lstNewtonsoftSerialize.Add(JsonConvert.SerializeObject(oPer
son));
                        }
                        sp2.Stop();
                        Console.WriteLine("Newtonsoft.Json方式序列化" +
lstNewtonsoftSerialize.Count + "个对象耗时：" + sp2.ElapsedMilliseconds + "毫秒");
                        lstRes.Clear();
                        Stopwatch sp3 = new Stopwatch();
```

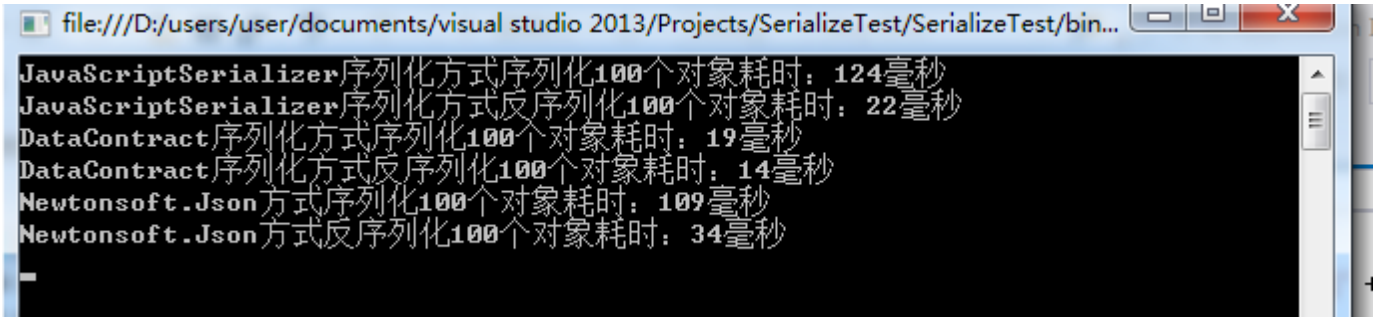
```
                sp3.Start();
                foreach (var oNewtonsoft in lstNewtonsoftSerialize)
                {
                        lstRes.Add(JsonConvert.DeserializeObject<Person>
(oNewtonsoft));
                }
                sp3.Stop();
                Console.WriteLine("Newtonsoft.Json方式反序列化" +
lstNewtonsoftSerialize.Count + "个对象耗时：" + sp3.ElapsedMilliseconds + "毫秒");
                #endregion
                #endregion
                #region 弱类型DataTable
                /*var dt = GetDataTable();
                #region JavaScriptSerializer序列化方式                    var
lstScriptSerializeObj = new List<string>();                    Stopwatch sp_script =
new Stopwatch();                    sp_script.Start();                    var
strRes = dt.ToScriptJsonString<DataTable>
();                    sp_script.Stop();                    Console.WriteLine("Java
ScriptSerializer序列化方式序列化" + lstScriptSerializeObj.Count + "个对象耗时：" +
sp_script.ElapsedMilliseconds + "毫
秒");                    dt.Clear();                    Stopwatch sp_script1 = new
Stopwatch();                    sp_script1.Start();                    dt =
strRes.ToScriptJsonObject<DataTable>
();                    sp_script1.Stop();                    Console.WriteLine("Jav
aScriptSerializer序列化方式反序列化" + lstScriptSerializeObj.Count + "个对象耗时：" +
sp_script1.ElapsedMilliseconds + "毫
秒");                    #endregion                    #region DataContract序列化方
式                    var lstFrameSerializeObj = new List<string>
();                    Stopwatch sp = new
Stopwatch();                    sp.Start();                    strRes =
dt.ToJsonString<DataTable>
();                    sp.Stop();                    Console.WriteLine("DataContrac
t序列化方式序列化" + lstFrameSerializeObj.Count + "个对象耗时：" + sp.ElapsedMilliseconds +
"毫秒");                    dt.Clear();                    Stopwatch sp1 = new
Stopwatch();                    sp1.Start();                    dt =
strRes.ToJsonObject<DataTable>
();                    sp1.Stop();                    Console.WriteLine("DataContra
ct序列化方式反序列化" + lstFrameSerializeObj.Count + "个对象耗时：" +
sp1.ElapsedMilliseconds + "毫
秒");                    #endregion                    #region
Newtonsoft                    var lstNewtonsoftSerialize = new List<string>
();                    Stopwatch sp2 = new
```

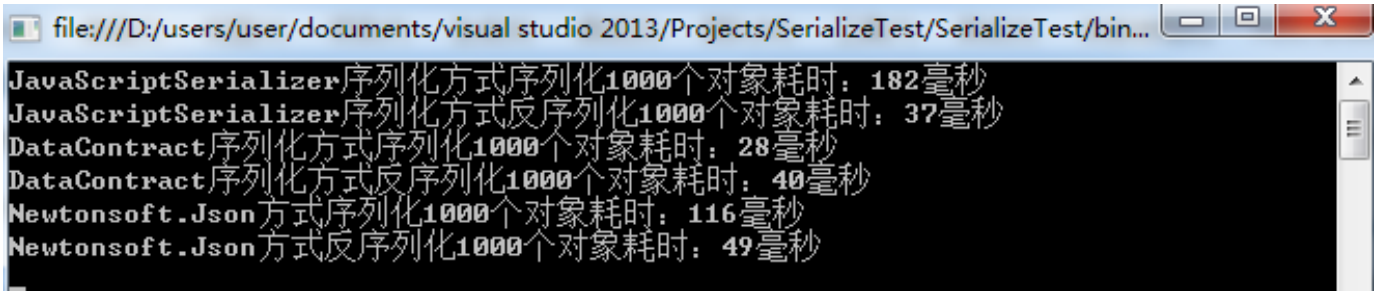
```
Stopwatch();                    sp2.Start();                    strRes =
JsonConvert.SerializeObject(dt);                    sp2.Stop();
Console.WriteLine("Newtonsoft.Json方式序列化" + lstNewtonsoftSerialize.Count + "个对象耗
时：" + sp2.ElapsedMilliseconds + "毫
秒");                    dt.Clear();                    Stopwatch sp3 = new
Stopwatch();                    sp3.Start();                    dt =
JsonConvert.DeserializeObject<DataTable>
(strRes);                    sp3.Stop();                    Console.WriteLine("Newt
onsoft.Json方式反序列化" + lstNewtonsoftSerialize.Count + "个对象耗时：" +
sp3.ElapsedMilliseconds + "毫秒");                    #endregion*/
                    #endregion
                    Console.ReadLine();
                }
```

4、测试结果：

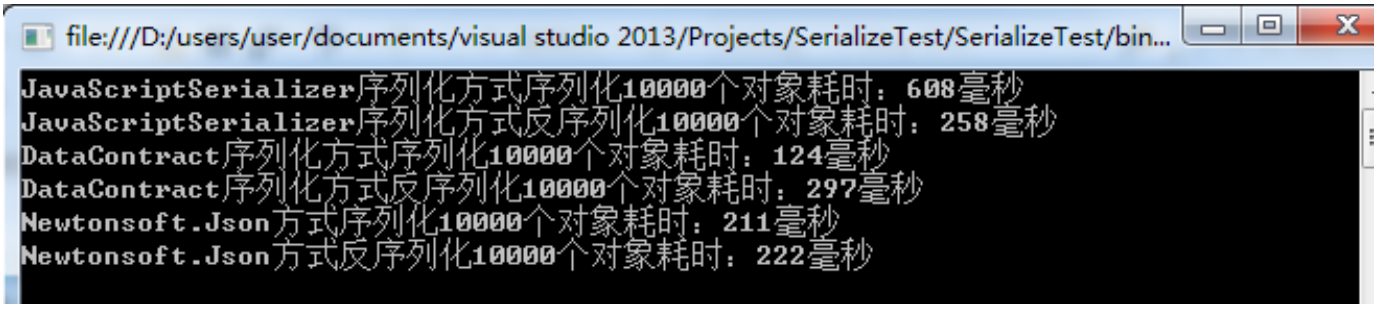
先说强类型对象的结果：

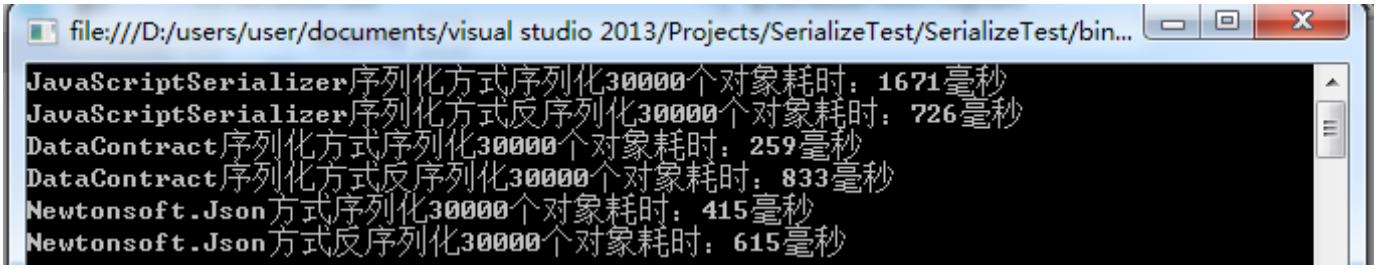
（1）集合数量100和1000时，序列化和反序列化三种方式差别不大：

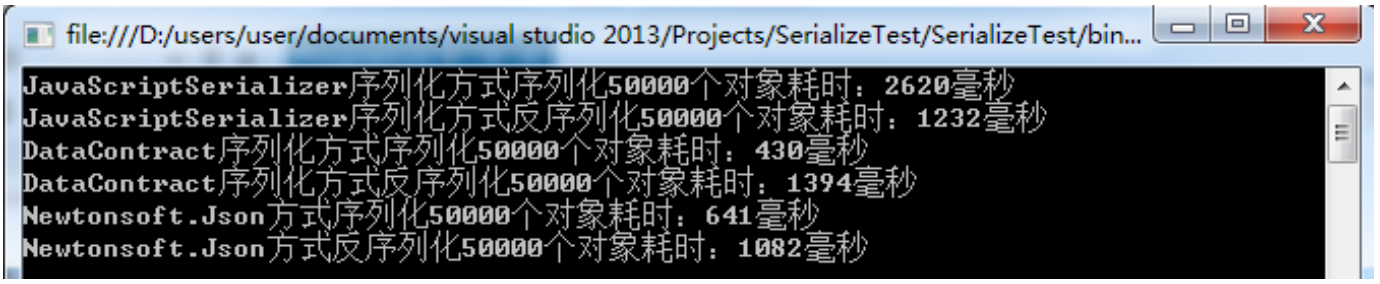




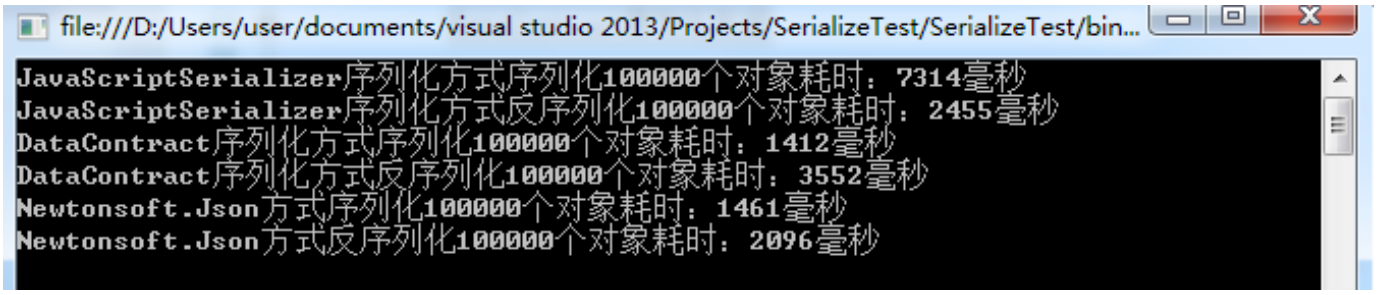
（2）当超过10000时，

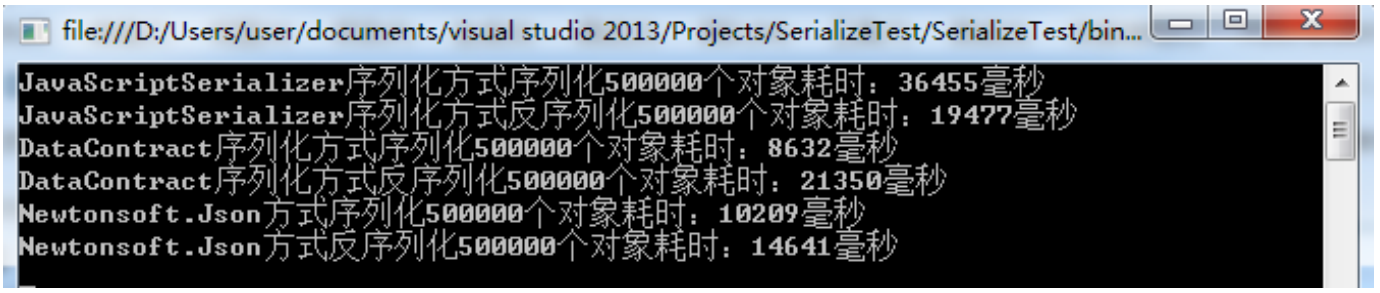






（3）继续加大数据量





弱类型DataTable的测试结果：

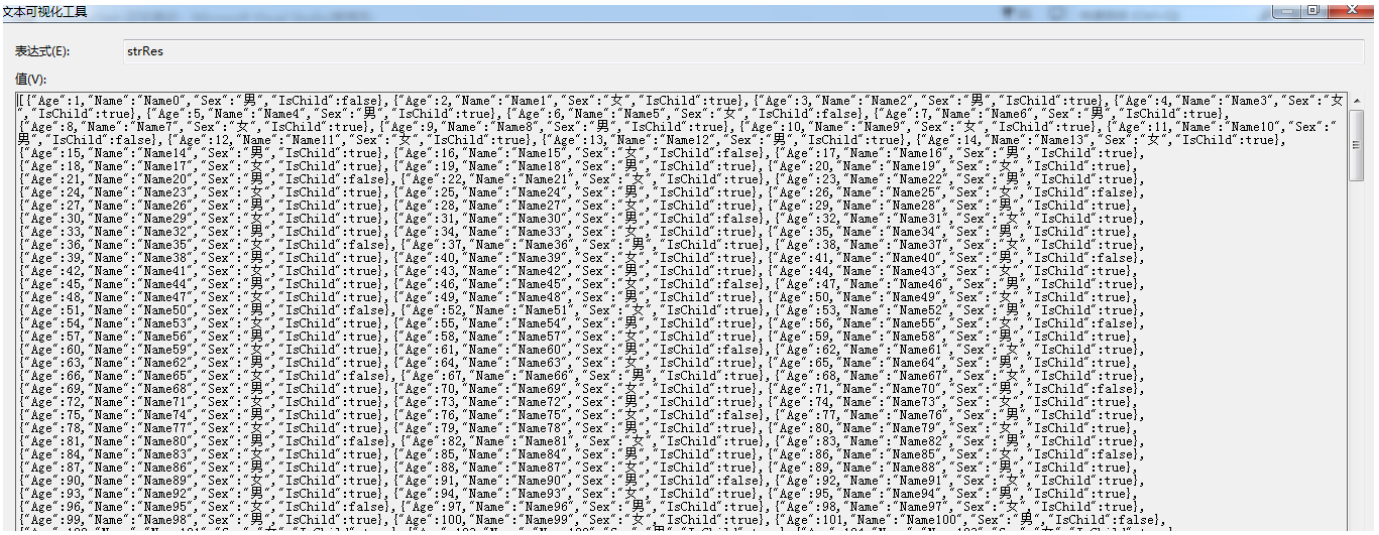
JavaScriptSerializer方式直接报错：

```
1 个引用
public static string ToScriptJsonString<T>(this T instanse)
{
    try
    {
        JavaScriptSerializer js = new JavaScriptSerializer();
        return js.Serialize(instanse);
    }
    catch
    {
        ret
    }
}

1 个引用
public stat
```

DataContract方式需要提供DataTable的表名，序列化得到是DataTable的Xml

Newtonsoft.Json方式可以实现和Json数据的序列化和反序列化。



5、测试总结：

（1）总的来说，DataContract和Newtonsoft.Json这两种方式效率差别不大，随着数量的增加

JavaScriptSerializer的效率相对来说会低些。

（2）对于DataTable的序列化，如果要使用json数据通信，使用Newtonsoft.Json更合适，如果是用xml做持久化，使用DataContract合适。

（3）随着数量的增加JavaScriptSerializer序列化效率越来越低，反序列化和其他两种相差不大。

（4）后来发现当对象的DataTime类型属性不赋值时，DataContract和JavaScriptSerializer这两种方式序列化都会报错，而用Newtonsoft.Json方式可以正常序列化。所以看来在容错方便，还是Newtonsoft.Json比较强。

以上只是楼主自己做的简单测试，可能存在不够严谨的地方，望各位大虾拍砖指正~~

附上源码：源码下载。