

[老老实实学WCF] 第八篇 实例化

老老实实学WCF

第八篇 实例化

通过上一篇的学习，我们简单地了解了会话，我们知道服务端和客户端之间可以建立会话连接，也可以建立非会话连接，通信的绑定和服务协定的ServiceContract 的SessionMode属性共同决定了连接是否是会话的。会话连接在会话保持阶段服务端可以记住客户端，而非会话连接则不会，相同客户端的多次调用会被认为是不同的客户端发起的。

会话这个特性是许多其他特性的基础，例如我们今天要学习的实例化。连接是否是会话对实例化的过程将产生不同的影响。今天我们就来研究这个问题。

1. 什么是实例化

那么，什么是实例化呢？我们知道，要调用一个类的方法，如果这个类不是静态类，首先要将这个类实例化从而获得这个类的一个对象，然后在这个对象上调用方法，这是面向对象的基本知识，而我们服务端上寄存的服务，也是一个类，是一个实现了服务接口的类，因此，客户端在调用这个类的方法时，服务端一定也要将这个类先实例化出一个对象，然后在这个对象上调用方法，将结果返回给客户端。

我们在这里探讨的实例化就是当客户端调用服务端服务方法时，服务端如何对服务类进行实例化的问题。

2. 实例化的几种模式

我们不禁要疑问，实例化不就是new嘛，new有什么可说的，直接实例化然后调用最后返回，这似乎没什么特别的地方。

new的话题我觉得是个挺深的问题，它也一直是面向对象编程理论中比较重要的一环。如何正确的控制实例化的时间和地点是许多设计模式着手解决的问题。我们在本地编程模型中不太多关注到它，可能常用的也就是为共享资源建立一个单例，或者为上层调用建立一个工厂。甚至随用随new的情况也都稀松平常。

然而我们应该重视实例化的方式，尤其在面向服务这样的模型中，实例化方式的不同会显著的影响性能和伸缩性。

按照最简单的考量，每次客户端的方法调用服务端都实例化一个对象然后为其服务，调用完就把对象销毁，也就是随用随new。这样看上去最简单明了，但是性能可能不太好，创建对象和销毁对象都是有开销的。如果客户端和服务端使用会话来连接，那么服务端就可以实例化一次，然后在整个会话期间都使用这一个对象为其服务，这样开销就小一些了，而且客户端是知道会话的存在的，他和服务端对于服务对象的状态是有共识的，这样可以重复的利用服务对象，直到会话结束才销毁服务对象。如果更进一步，如果创建对象的开销太大而且对象提供的服务并没有对不同客户端的差异，可以让服务端只实例化一次，这个对象将为所有的客户端服务，无论连接是不是会话，这个对象一直也不会被销毁，除非服务停止或重新启动。

综上所述，服务端实例化的方式有三种，分别是"每调用实例"，"每会话实例"和"单一实例"。

实例化模式的指定是通过配置服务类的ServiceBehavior属性中的InstanceContextMode属性来实现的。注意，是服务类的属性，而不是服务协定。

(1) 每调用实例

将服务类的ServiceBehavior属性中的InstanceContextMode属性设置为PerCall来启用这种模式

在这种实例化模式下，客户端对服务方法的每一次调用，服务端都会new一个实例，方法调用结束即销毁，这种模式可能要频繁的花费创建对象和销毁对象的花销，因此性能比较差，但是每次调用后服务对象被销毁，对象所把持的资源也就被释放(如文件啦、数据库连接啦)，因此伸缩性是最好的。

看下面的例子。

首先看加在服务类上的属性应该怎么写：

```
[csharp]
1. [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
```

我用的还是前几篇中用的IIS的例子，HelloWCFService.cs的源代码如下：

```
[csharp]
1.  using System;
2.  using System.ServiceModel;
3.
4.  namespace LearnWCF
5.  {
6.      [ServiceContract(SessionMode = SessionMode.Required)]
7.      public interface IHellwWCF
8.      {
9.          [OperationContract]
10.         string HelloWCF();
11.     }
12.
13.     [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
14.     public class HelloWCFService : IHellwWCF
15.     {
16.         private int _Counter;
17.         public string HelloWCF()
18.         {
19.             _Counter++;
20.             return "Hello, you called " + _Counter.ToString() + " time(s)";
21.         }
22.     }
23. }
```

Web.Config文件内容如下

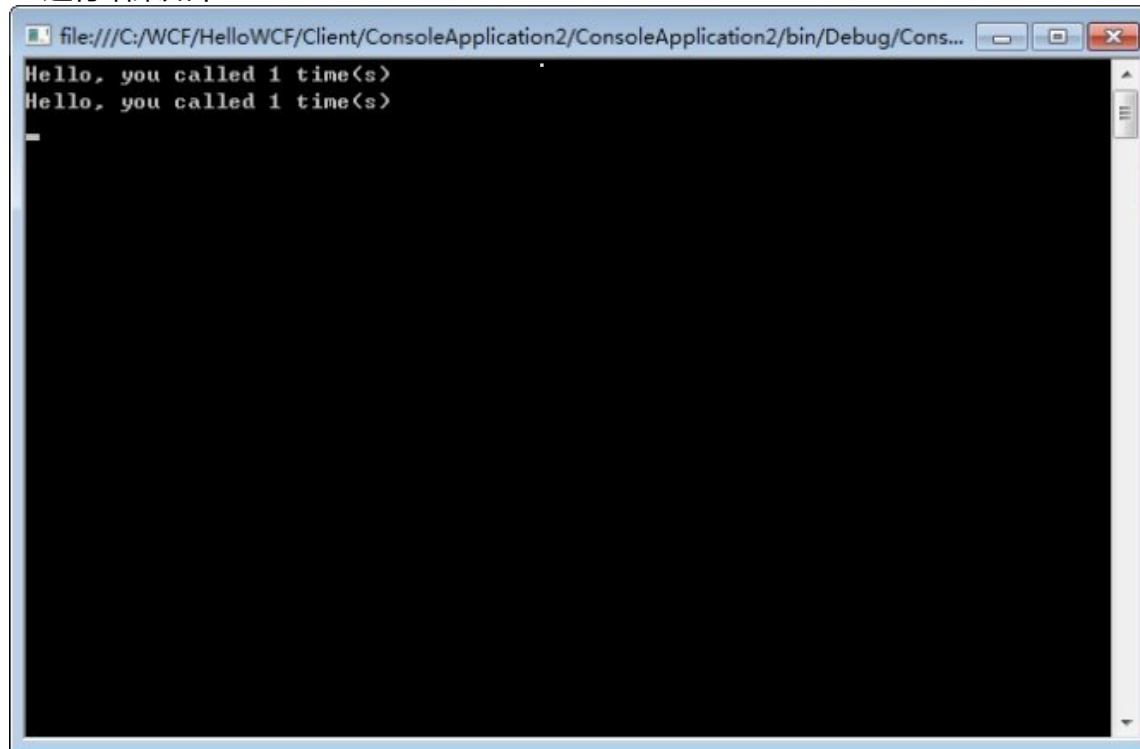
```
[html]
1.  <configuration>
2.      <system.serviceModel>
3.          <services>
4.              <service name="LearnWCF.HelloWCFService" behaviorConfiguration="metadataExchange">
5.                  <endpoint address="" binding="wsHttpBinding" contract="LearnWCF.IHellwWCF"/>
6.                  <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>
7.              </service>
8.          </services>
9.          <behaviors>
10.             <serviceBehaviors>
```

```
11.         <behavior name="metadataExchange">
12.             <serviceMetadata httpGetEnabled="true" />
13.         </behavior>
14.     </serviceBehaviors>
15. </behaviors>
16. </system.ServiceModel>
17. </configuration>
```

调用的客户端代码如下：

```
[csharp]
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5.
6. using System.ServiceModel;
7.
8. namespace ConsoleClient
9. {
10.     class Program
11.     {
12.         static void Main(string[] args)
13.         {
14.             Services.HelloWCFClient client = new Services.HelloWCFClient();
15.
16.             Console.WriteLine(client.HelloWCF());
17.             Console.WriteLine(client.HelloWCF());
18.
19.             client.Close();
20.
21.             Console.ReadLine();
22.         }
23.     }
24.
25.
26. }
```

F5运行结果如下：



```
file:///C:/WCF/HelloWCF/Client/ConsoleApplication2/ConsoleApplication2/bin/Debug/Cons...
Hello, you called 1 time(s)
Hello, you called 1 time(s)
```

可以看到，即使我们把服务协定的会话模式设置为Required(必须使用会话)，即这个时候服务端和客户端是在进行会话连接的时候，服务端还是为两次客户端调用分别实例化了对象，所以看到计数没有增长，每次都是1。

(2) 每会话实例

将服务类的ServiceBehavior属性的InstanceContextMode属性设置为PerSession时启用这种模式。

如果不指定，那么这个值是默认的配置。

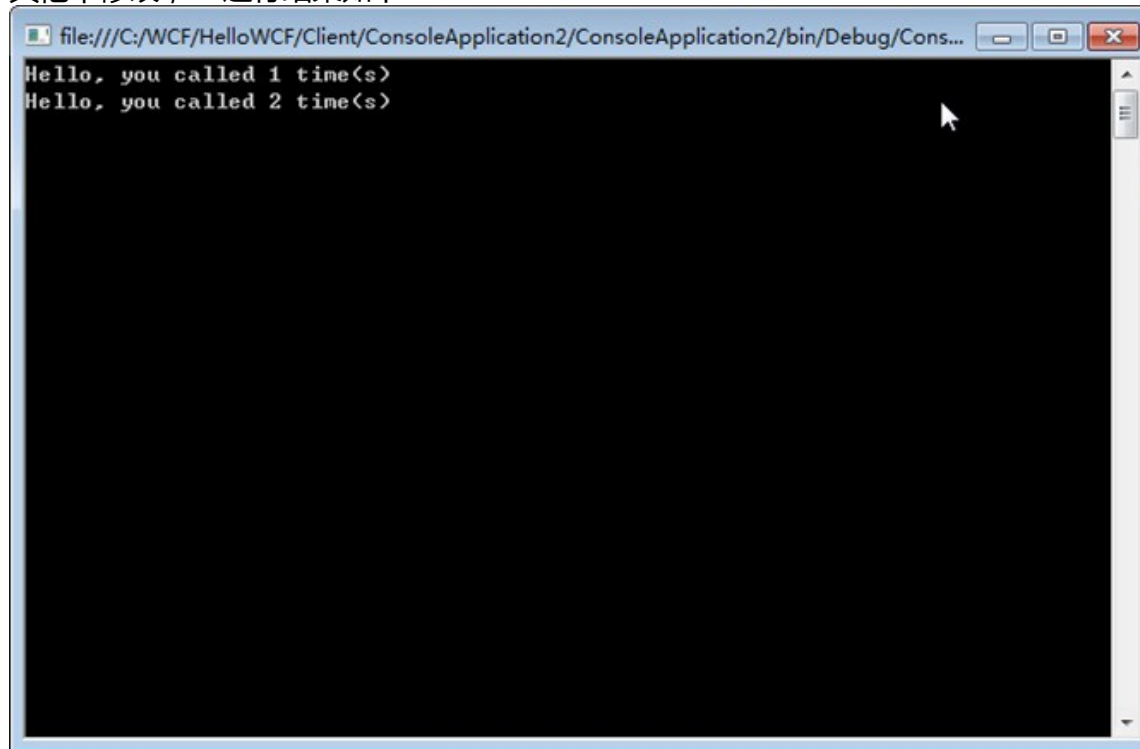
这种模式下服务端在受到会话的第一个调用时实例化对象，直到会话关闭才销毁对象，这样减少了对对象的建立和销毁开销，性能有一定提升，但是在会话期间申请的资源如文件和数据库连接直到会话结束才会释放，伸缩性有所下降。

看看给服务类的属性应该怎么写：

[csharp]

1. [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession)]

其他不修改，F5运行结果如下：



可以看到计数增长了，说明没有实例化新的对象，在整个会话中只有一个。设想一下，如果此时把服务协定的SessionMode属性设为NotAllowed会怎样呢，结果会跟PerCall一样，因为PerSession模式需要会话的支持，如果协定不支持会话，服务端就不可能记得住客户端了，也可以理解成每次调用都是一个全新的会话了。

(3) 单一实例

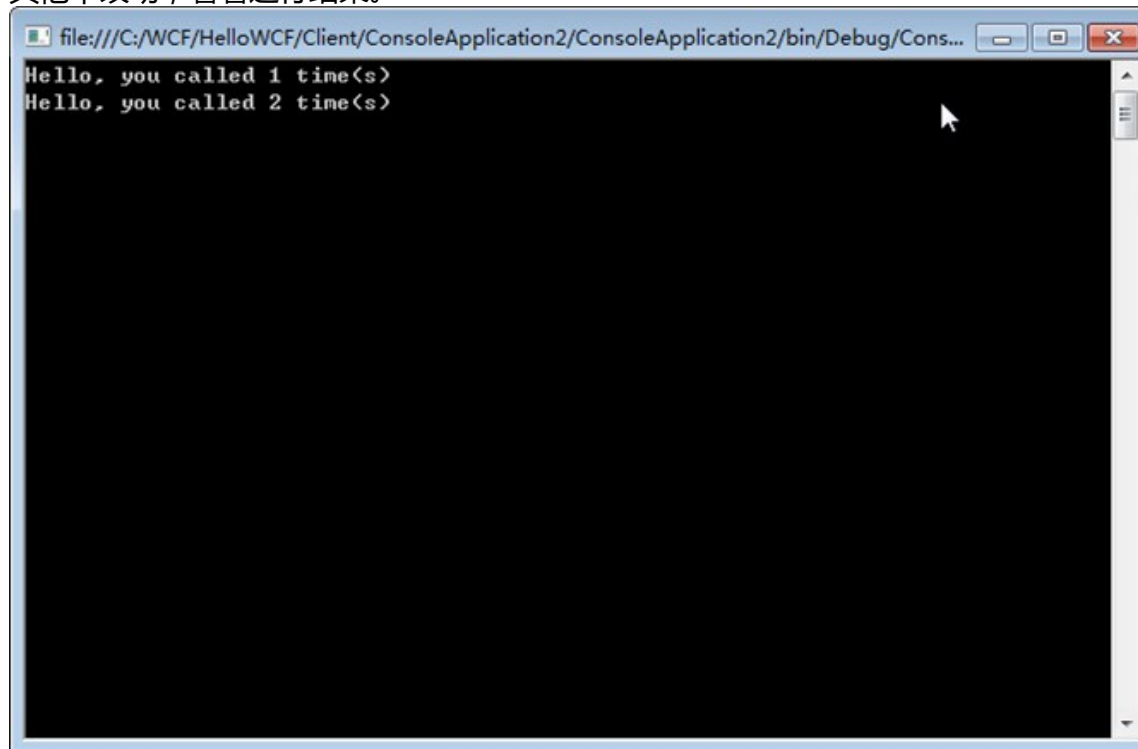
把服务类的ServiceBehavior属性的InstanceContextMode属性设置为Single来启用这种模式。

在这种模式下，服务类在受到首次调用的时候实例化，然后一直不销毁，直到服务宿主关闭，在这期间无论是会话连接还是非会话连接，服务端都用这个实例进行调用。这样实际上几乎没有创建和销毁对象的开销，但是如果一旦实例把持资源，则一直都不能释放，毫无伸缩性可言。

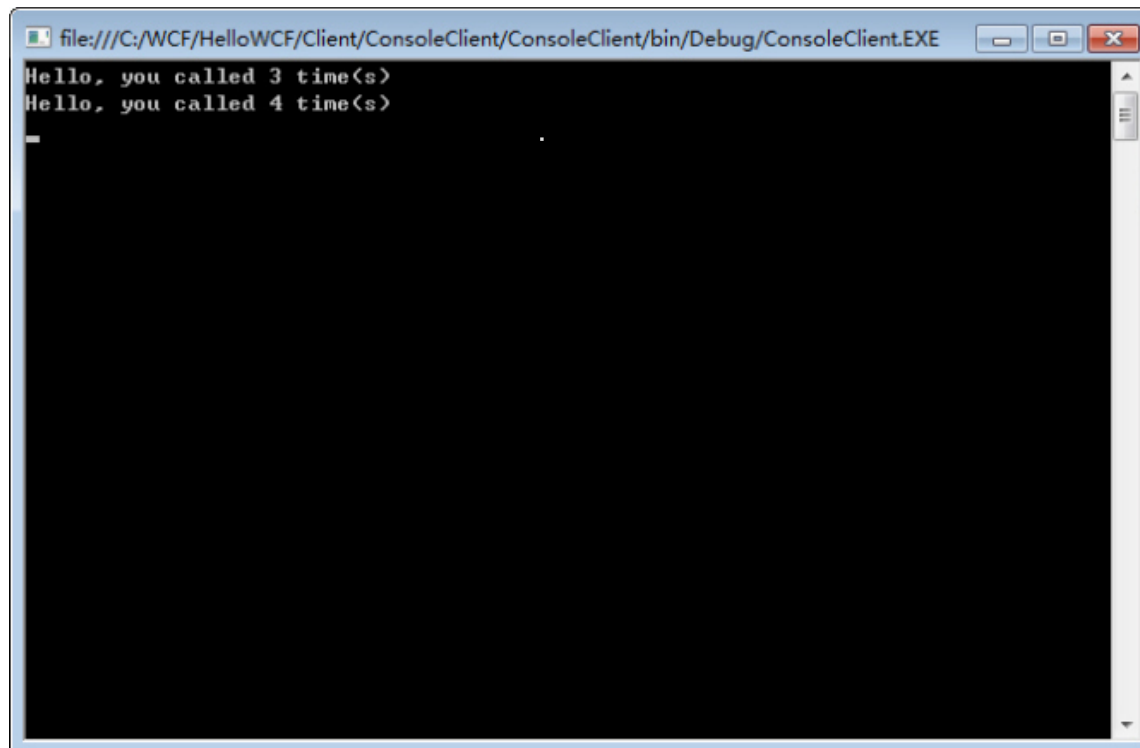
看看服务类的ServiceBehavior属性怎么写：

```
[csharp]
1. [ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]
```

其他不改动，看看运行结果。



似乎和PerSession没有什么区别，关闭客户端程序再运行一次，结果如下：



计数器继续增长了，客户端程序已经重新运行，很显然这是个新的会话，但是服务端的实例一直没有销毁，所以计数器就继续增长了。

在这种模式下，无论我们把协定的SessionMode改成什么值，结果都是一样的，因为服务端已经认准了一个实例，无论客户端连接是不是会话了。

3. 总结

通过今天的学习，我们了解到了服务类实例化的几种模式，从PerCall到Single，性能越来越好，伸缩性越来越差，我们在实际项目中可以按照自己应用的特点来进行选择，同时要注意会话模式和实例化模式之间互相的影响：

- (1) 如果协定不支持会话，那么PerSession相当于PerCall。
- (2) 如果服务类为PerCall，那么协定的是否支持会话结果相同。
- (3) 如果服务类为Single，那么协定是否支持会话结果相同。

其实还有一个并发的特性，稍复杂，我们后面再展开。