

WCF 서비스의 동시성

Implementing a Singleton

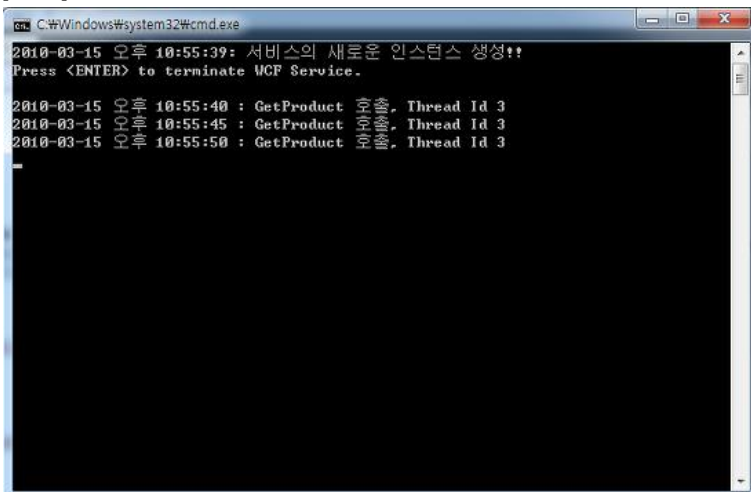
단 하나의 서비스 인스턴스만이 생성되며, 인스턴스에서 동작하는 스레드 역시, 단 하나만 생성되게 하는 경우에 대해 먼저 살펴보겠습니다.

저번 포스팅에서 사용했던 서비스 클래스의 코드를 다음과 같이 굵은 글씨체 부분만을 수정하여 서비스를 실행해 보시기 바랍니다.

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single,  
                 ConcurrencyMode = ConcurrencyMode.Single)]  
class ProductService : IProductService  
{  
    ProductService()  
    {  
        Console.WriteLine("{0}: 서비스의 새로운 인스턴스 생성!!", DateTime.Now);  
    }  
    ... 생략 ...  
}
```

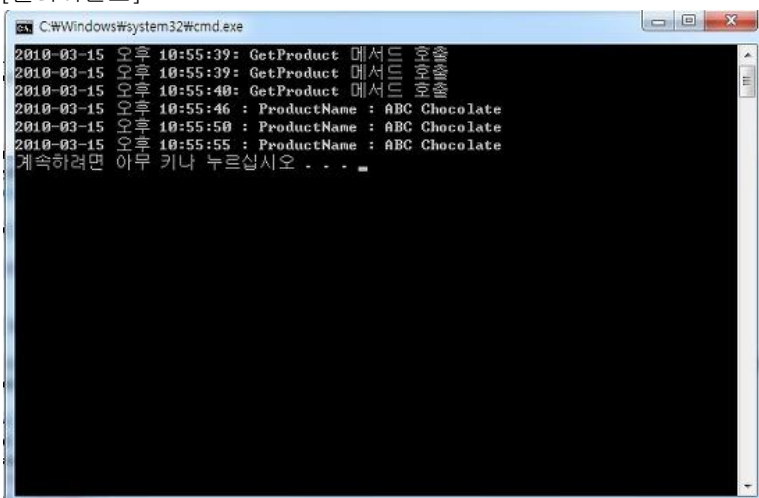
이 코드를 실행하면 다음과 같은 결과를 확인할 수 있을 것입니다.

[서버]



```
C:\Windows\system32\cmd.exe  
2010-03-15 오후 10:55:39: 서비스의 새로운 인스턴스 생성!!  
Press <ENTER> to terminate WCF Service.  
2010-03-15 오후 10:55:40 : GetProduct 호출. Thread Id 3  
2010-03-15 오후 10:55:45 : GetProduct 호출. Thread Id 3  
2010-03-15 오후 10:55:50 : GetProduct 호출. Thread Id 3  
-
```

[클라이언트]



```
C:\Windows\system32\cmd.exe  
2010-03-15 오후 10:55:39: GetProduct 메서드 호출  
2010-03-15 오후 10:55:39: GetProduct 메서드 호출  
2010-03-15 오후 10:55:40: GetProduct 메서드 호출  
2010-03-15 오후 10:55:46 : ProductName : ABC Chocolate  
2010-03-15 오후 10:55:50 : ProductName : ABC Chocolate  
2010-03-15 오후 10:55:55 : ProductName : ABC Chocolate  
계속하려면 아무 키나 누르십시오 . . .
```

결과가 여러분이 예상했던 것과 일치하나요?

이 경우 역시 서비스의 인스턴스는 서버 측 결과 화면을 통해 **클라이언트의 요청의 수와는 관계없이 하나 만** **생성됨**을 알 수 있습니다.

그리고, 클라이언트에서 서비스를 호출하는 방식으로 비동기 방식을 사용했던 것 기억 하실 겁니다. 결과 그림만을 보서는 잘 모를 수도 있지만, 이 때문에 클라이언트 측 결과 화면을 보면, 지연시간 없이(각 호출에 대한 결과를 받지 않아도) 서비스를 연속적으로 세 번 호출함을 볼 수 있습니다.

하지만, 이 호출에 대한 결과는 각각 5 초간의 지연시간을 두고 화면에 출력하는데, 이는 서버측 결과 화면을 보면 그 이유를 알 수 있습니다. 만약, 서비스 인스턴스가 여러 개의 스레드를 만들어 요청을 처리했다면, 클라이언트의 각 요청에 대한 결과를 거의 동시에 받을 수 있겠지만, 이 경우에는 **하나의 스레드 만** **동작하기 때문에 각 요청을 한번에 하나씩** 처리할 수 있어 이러한 결과를 얻을 수 있는 것이죠. 참고로, 각 요청에 대한 처리는 **FIFO(First In First Out)의 순서로 동작**합니다.

서비스 인스턴스에서 단 하나의 스레드 만이 동작한다는 것은 서버 측 결과에서 Thread ID 값이 3 으로 동일한 것을 봐도 증명이 가능합니다. 가끔, 이 thread id 의 값이 각 요청마다 다른 값이 나올 수도 있습니다. 그렇다고 잘못된 결과값은 아닙니다. **ConcurrencyMode.Single** 은 **한번에 하나의 스레드만이 동작한다는 것을** 명시하는 거지, 단 하나의 스레드만이 만들어진다는 의미는 아니거든요~, 약간 헷갈릴 수도 있는 부분인 것 같으니, 꼭 명심해주세요

이 경우처럼 싱글 인스턴스, 싱글 스레드는 한번에 하나의 클라이언트 요청만을 처리할 수 있기 때문에 throughput 을 감소시킨다는 단점이 있지만, 반면에 시스템 자원(resource)에 동시 접근 같은 문제가 일어나지 않아서, 이에 대한 추가적인 관리가 필요하지 않다는 장점도 있습니다.

Session-Level Instances

이번에는 세션 모드가 적용되었을 때, 서비스의 인스턴스가 어떻게 생성되는지 한번 살펴보도록 하겠습니다. 우선, 서비스에서 세션을 지원하도록 하기 위해 서비스 계약의 특성값을 다음과 같이 수정합니다.

```
[ServiceContract(Namespace = "http://RuAAService.co.kr",
                SessionMode = SessionMode.Required)]
interface IProductService
{
    [OperationContract]
    Product GetProduct();
}
```

SessionMode 에 Required 값을 적용하여 이 서비스가 세션 모드를 지원해준다는 것을 명시해 줍니다. 다음은 ServiceBehavior 특성에서 **InstanceContextMode**의 값을 **PerSession**으로, **ConcurrencyMode**의 값을 **Multiple**로 수정을 해줍니다. 그리고 GetProduct 메서드가 한 인스턴스에서 몇 번 호출이 이루어지는지를 체크하기 위해 n_Calls 라는 이름의 필드를 추가해 주었습니다. lockThis 필드는 n_Calls 필드의 값을 증가시킬 때 다른 스레드에서 동시에 n_Calls의 값을 바꾸지 못하도록 하기 위한 목적으로 선언해주었습니다.

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession,
                ConcurrencyMode = ConcurrencyMode.Multiple)]
class ProductService : IProductService
{
    object lockThis = new object();
    private int n_Calls = 0;

    ... 중간 생략 ...

    public Product GetProduct()
    {
        Console.WriteLine("{0} : GetProduct 호출, Thread Id {1}",
                        DateTime.Now, Thread.CurrentThread.ManagedThreadId);

        Thread.Sleep(1000);

        Product p = new Product();
        p.ProductId = 1234;
        p.ProductName = "ABC Chocolate";
        p.Price = 1500.0;
        p.Company = "Lottee";
        p.CreateDate = DateTime.Parse("2010-01-22");

        lock (lockThis)
        {
            p.calls = ++n_Calls;
        }

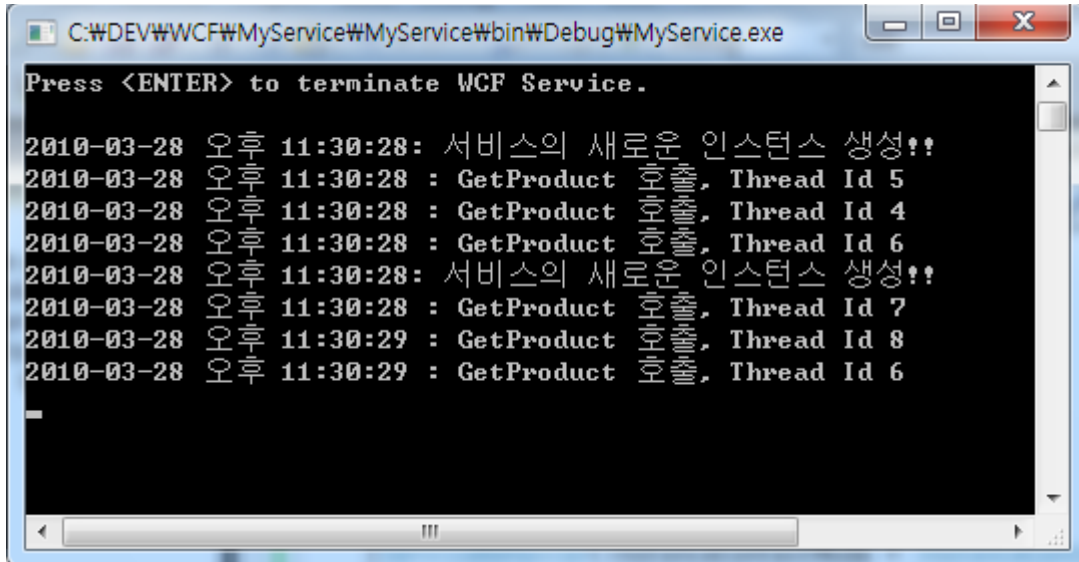
        return p;
    }
}
```

아, 이런~ 저와 같은 방법으로 수정을 한 후에 실행 시키면, 아마 예외가 발생하실겁니다.

예외의 이유는 간단합니다. Service Contract 에서 세션 모드의 값을 Required 로 설정을 해놓았는데, **실제 서비스를 호스팅할 때 세션을 지원하지 않는 BasicHttpBinding 을 사용했기 때문**입니다. 따라서, **BasicHttpBinding 을 WSHHttpBinding 으로 수정**해주시면 이 예외는 발생하지 않을 것입니다.

자~ 이 부분 수정을 다 하셨다면, 다시 실행을 해보도록 하겠습니다.
세션모드의 지원을 확인하기 위해서 클라이언트 어플리케이션을 두 개 연속해서 실행을 시킵니다.

다음은 서버 어플리케이션의 실행 화면입니다.

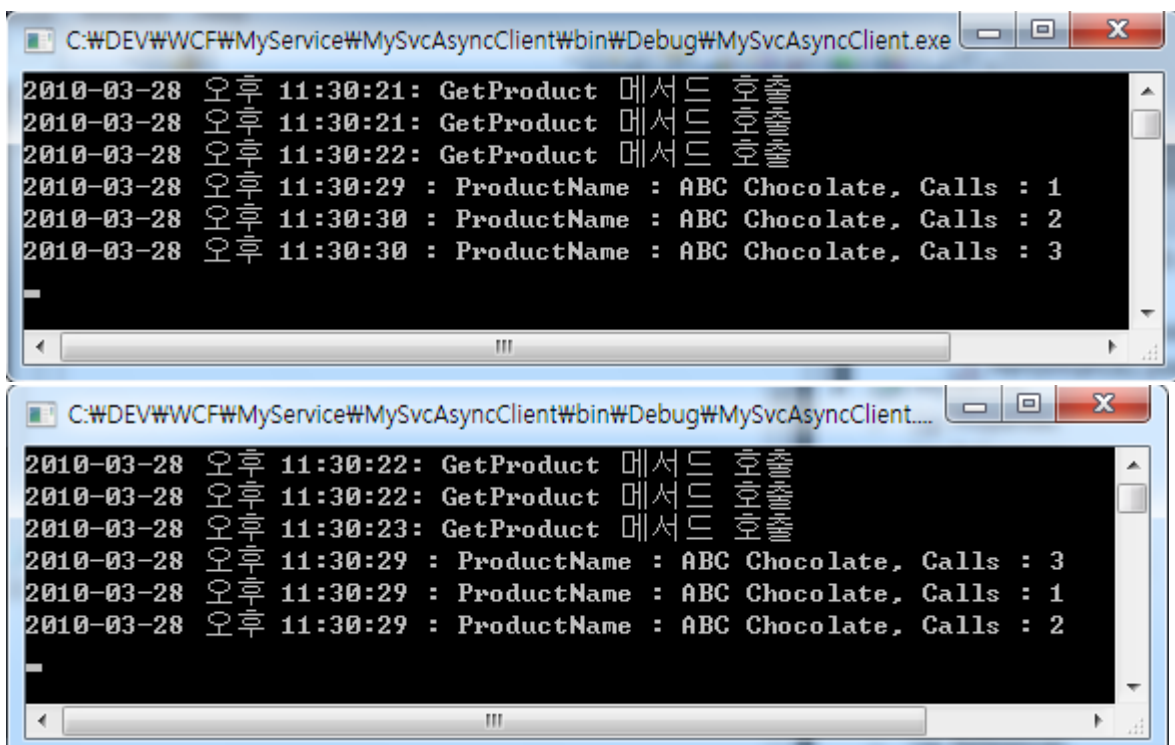


```
C:\DEV\WCF\MyService\MyService\bin\Debug\MyService.exe
Press <ENTER> to terminate WCF Service.

2010-03-28 오후 11:30:28: 서비스의 새로운 인스턴스 생성!!
2010-03-28 오후 11:30:28 : GetProduct 호출, Thread Id 5
2010-03-28 오후 11:30:28 : GetProduct 호출, Thread Id 4
2010-03-28 오후 11:30:28 : GetProduct 호출, Thread Id 6
2010-03-28 오후 11:30:28: 서비스의 새로운 인스턴스 생성!!
2010-03-28 오후 11:30:28 : GetProduct 호출, Thread Id 7
2010-03-28 오후 11:30:29 : GetProduct 호출, Thread Id 8
2010-03-28 오후 11:30:29 : GetProduct 호출, Thread Id 6
```

인스턴스가 두 개 생성된 것을 확인할 수 있는데요,, 왜 인스턴스가 두 개 생성되었을까요?? 당연히 클라이언트 어플리케이션이 두 개 실행이 되었기 때문입니다. 세션을 지원하는 서비스이니깐요,,

자~ 다음은 두 클라이언트 어플리케이션의 실행화면입니다.



```
C:\DEV\WCF\MyService\MySvcAsyncClient\bin\Debug\MySvcAsyncClient.exe
2010-03-28 오후 11:30:21: GetProduct 메서드 호출
2010-03-28 오후 11:30:21: GetProduct 메서드 호출
2010-03-28 오후 11:30:22: GetProduct 메서드 호출
2010-03-28 오후 11:30:29 : ProductName : ABC Chocolate, Calls : 1
2010-03-28 오후 11:30:30 : ProductName : ABC Chocolate, Calls : 2
2010-03-28 오후 11:30:30 : ProductName : ABC Chocolate, Calls : 3

C:\DEV\WCF\MyService\MySvcAsyncClient\bin\Debug\MySvcAsyncClient.exe
2010-03-28 오후 11:30:22: GetProduct 메서드 호출
2010-03-28 오후 11:30:22: GetProduct 메서드 호출
2010-03-28 오후 11:30:23: GetProduct 메서드 호출
2010-03-28 오후 11:30:29 : ProductName : ABC Chocolate, Calls : 3
2010-03-28 오후 11:30:29 : ProductName : ABC Chocolate, Calls : 1
2010-03-28 오후 11:30:29 : ProductName : ABC Chocolate, Calls : 2
```

각 클라이언트는 호출한 횟수가 1~3 인 것을 확인할 수 있습니다. 이것은 **각 클라이언트 마다의 서비스 인스턴스가 따로 데이터를 유지한다는 것**을 의미하는 것이죠.