서비스의 동시성(Concurrency)

이번 포스팅의 주제는 WCF 의 Behaviors 중에서 서비스의 동시성(Concurrency)을 컨트롤 할 수 있는 Behavior 입니다.

Behavior 는 서비스가 동작할 때(그러니깐 런타임 시) 동작에 영향을 끼치는 클래스들로, 서비스 클래스의 특성으로 지정하거나, 환경 설정파일을 통해 지정할 수 있습니다.

Behavior 와 관련된 여러 가지 내용 중 이번 포스팅에선 **동시성(Concurrency)**에 대해서 얘기해보도록 하겠습니다.

동시성이라 함은, 여러 task 들이 동시에 동작하는 것을 말합니다.

동시성은 시스템의 throughput(출력률)에 큰 영향을 끼칩니다. 일정 시간 동안 처리할 수 있는 작업의 양이 커지기 때문이죠.

WCF 에서는 동시성을 컨트롤할 수 있는 두 종류의 behavior 가 있습니다. 바로 **"InstanceContextMode"** 와 **"ConcurrencyMode"** 입니다.

1. InstanceContextMode

생성되는 서비스의 인스턴스를 조절할 수 있는 behavior 로 다음과 같은 세 종류의 값으로 설정할 수 있습니다.

- **Single** : 이 값은 서비스로 들어오는 모든 요청을 하나의 인스턴스에서 처리하도록 설정합니다.
- **PerCall** : 서비스로 들어오는 요청마다 서비스의 인스턴스가 만들어지도록 하기 위한 설정입니다.
- **PerSession** : 클라이언트 세션마다의 서비스 인스턴스를 생성하기 위한 설정이며, 만약 세션을 사용하지 않는 채널일 때, 이 값으로 설정이 된다면, PerCall 과 같은 방식으로 동작합니다.

그리고, InstanceContextMode 의 기본값은 PerSession 으로 따로 어떠한 값도 설정되어 있지 않은 경우엔 세션 수에 따라 서비스 인스턴스가 생성됩니다.

2. ConcurrencyMode

하나의 서비스 인스턴스 내에서 동작하는 스레드를 통한 동시성을 컨트롤하는 behavior 입니다.
다음은 ConcurrencyMode 에서 설정할 수 있는 값에 대한 설명입니다.

- **Single** : 하나의 서비스 인스턴스 내에 오로지 하나의 스레드만이 동작하도록 설정하는 값입니다. 따라서, 이 값으로 설정되어 있는 경우엔 스레딩 문제를 고려하지 않아도 된다는 장점이 있습니다.
- **Reentrant** : 이 설정 역시 하나의 서비스 인스턴스에서 하나의 스레드만이 동작하도록 하는 설정값입니다. 하지만, 이 설정값이 Single 과 다른 점은 하나의 스레드가 동작하는 도중에 다른 작업이 처리될 수 있다는 것입니다. 이 작업의 처리가 완료되면 이 전의 작업이 계속해서 동작됩니다.
- **Multiple** : 하나의 서비스 인스턴스에서 하나 이상의 스레드가 동작할 수 있도록 하는 설정입니다. 이 값으로 설정되어 있는 경우엔 여러 개의 스레드에서 서비스 개체를 변경할 수 있기 때문에 항상 동기화와 상태 일관성을 처리해 주어야 합니다.

ConcurrencyMode 와 InstanceContextMode 의 값을 적절하게 조합하면, 서비스의 기능에 맞게 동시성과 인스턴스 관리를 할 수 있습니다.

네~ 이제 InstanceContextMode 와 ConcurrencyMode 의 값을 적절하게 조합하여 서비스에 적용하는 실습을 해보도록 하겠습니다.

우선, 가장 먼저 세션을 사용하지 않는 환경에서 InstaceContextMode 와 ConcurrencyMode 의 기본값을 사용한 서비스를 구현해보겠습니다. InstanceContextMode 의 기본값은 PerSession 이며, ConcurrencyMode 의 기본값은 Single 입니다. 이 기본값은 따로 설정해주지 않아도 적용된다는거 아시죠?

다음은 서비스를 구현한 클래스의 코드 입니다.

```
class ProductService : IProductService
{
    ProductService()
    {
        Console.WriteLine("{0}: 서비스의 새로운 인스턴스 생성!!", DateTime.Now);
    }

    public Product GetProduct()
    {
        Console.WriteLine("{0} : GetProduct 호출, Thread Id {1}", DateTime.Now,
                          Thread.CurrentThread.ManagedThreadId);
        Thread.Sleep(5000);

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

        return p;
    }
}
```

저번 포스팅에서 사용했던 서비스의 코드를 살짝 수정 해보았습니다.
서비스 클래스 생성자를 만들어 단순하게 인스턴스가 생성되었다는 메시지를 출력해주는 코드를 추가하였구요,
GetProduct 메서드 내에서는 현재 스레드의 ID 값을 출력해주는 코드를 추가하였습니다.

다음은 이 서비스를 호출하는 클라이언트 코드입니다. 코드를 보시면 아시겠지만 클라이언트에서 서비스 메서드를 비동기로 호출하고 있습니다.

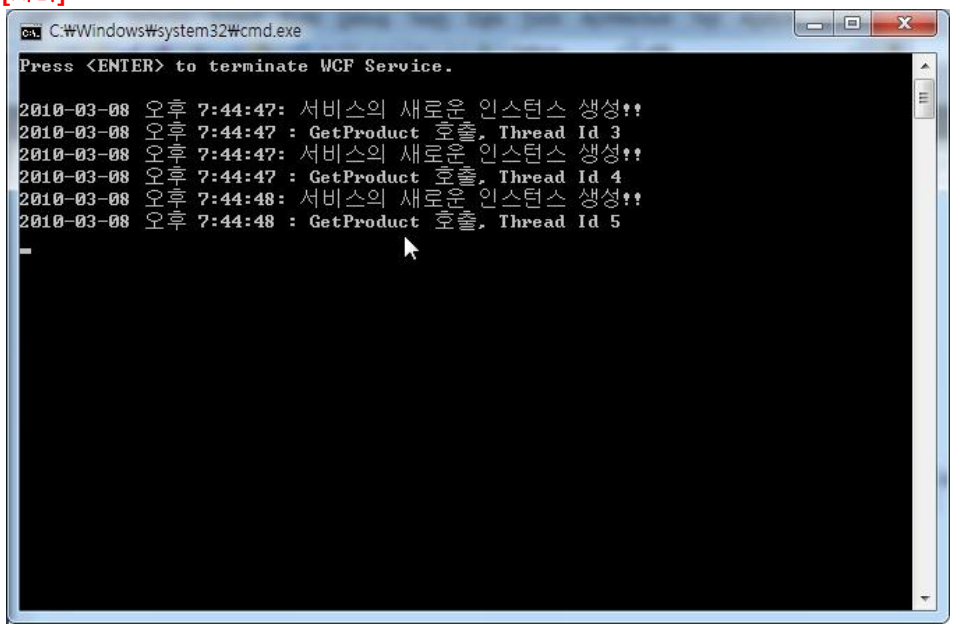
```
namespace MySvcAsyncClient
{
    class Program
    {
        static int c = 0;
        static void Main(string[] args)
        {
            ProductServiceClient proxy = new ProductServiceClient();
            for (int i = 0; i < 3; i++)
            {
                Console.WriteLine("{0}: GetProduct 메서드 호출", DateTime.Now);
                proxy.BeginGetProduct(GetProductInfoCallback, proxy);
                Thread.Sleep(100);
                Interlocked.Increment(ref c);
            }
            while (c > 0)
            {
                Thread.Sleep(100);
            }
        }

        static void GetProductInfoCallback(IAsyncResult ar)
        {
            ProductInfo productInfo = ((ProductServiceClient)ar.AsyncState)
                                        .EndGetProduct(ar);
```

```
            Console.WriteLine("{0} : ProductName : {1}",
                                        DateTime.Now, productInfo.Name);
            Interlocked.Decrement(ref c);
        }
    }
}
```

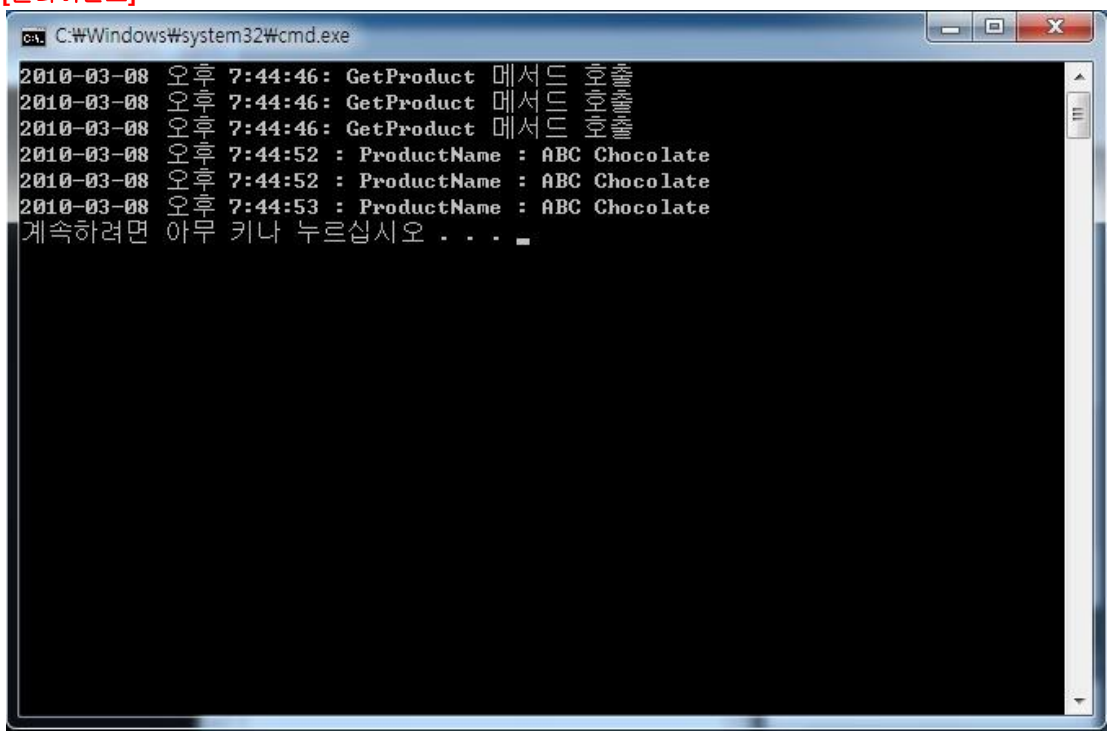
Main 메소드 내에서는 for 문을 사용하여 3 번 반복하여 GetProduct 메소드를 비동기로 호출하고 있으며, 각각의 비동기 호출에 의한 작업이 끝이 나면 AsyncCallback 대리자인 GetProductInfoCallback 메소드가 호출되며, 서비스에서 받은 Product 데이터를 화면에 출력해줍니다.

다음은 이 코드에 대한 결과 화면입니다.

[서버]



[클라이언트]



클라이언트 측 결과 화면을 보면 동시에 서비스의 메소드를 세번 호출하는 것을 확인할 수 있습니다. 그리고 6 초 정도의 시간 후에 차례대로 결과값을 가져와서 출력하는 것을 볼 수 있습니다.

서비스 측 결과 화면을 확인해 보면, **각 호출마다 생성자를 통해 새로운 인스턴스를 생성하고, 인스턴스 내에 하나의 스레드를 통해 GetProduct 메소드를 호출**하는 것을 확인할 수 있습니다.

여기서 잠깐 의문이 들지도 모르겠습니다.

InstanceContextMode 의 기본값은 PerSession 이라고 했는데 왜 서버에선 클라이언트의 호출마다 새로운 인스턴스를 생성한 것일까요?

답은 아주 간단합니다. 서비스를 호스팅할 때 사용했던 binding 의 종류가 BasicHttpBinding 이었던 것 기억하시나요? **BasicHttpBinding 의 경우엔 세션을 사용하지 않기 때문에, 이 경우엔 실제로 InstanceContextMode.PerCall 과 같은 형식으로 동작**하게 되는 것입니다.

기본값으로 설정한 경우를 알아봤으니, 이번엔 두 모드의 값을 바꿔서 서비스에 적용해보겠습니다.

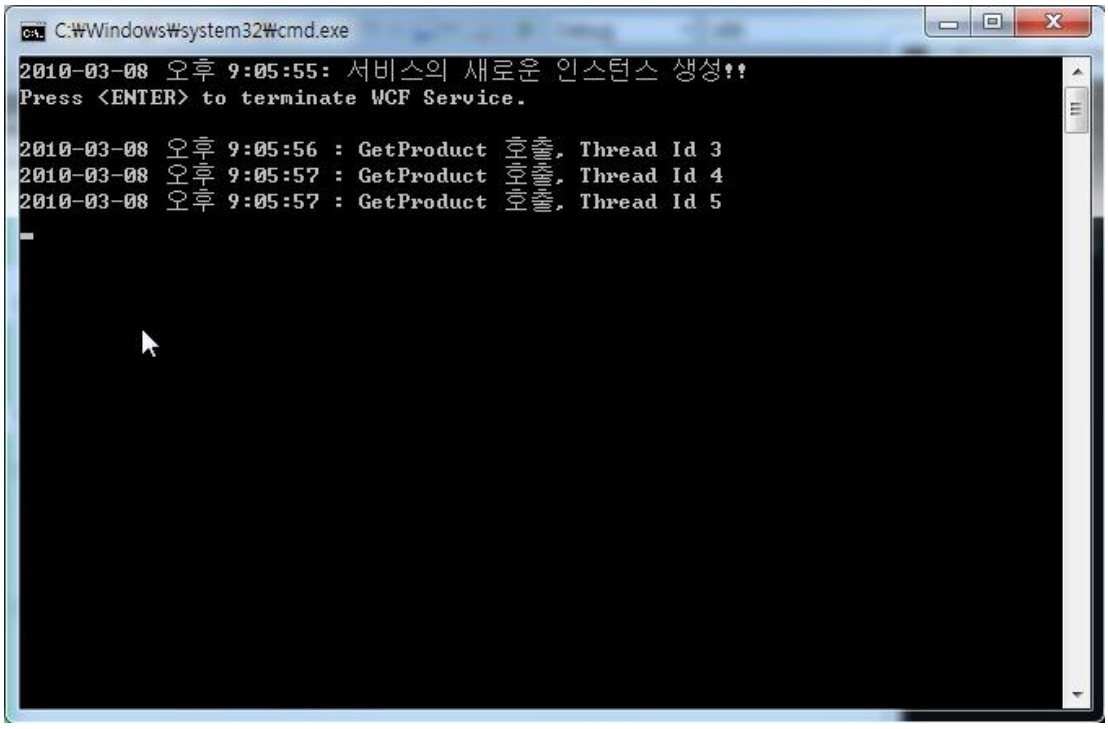
인스턴스는 모든 호출에 대해 하나만 생성하도록하고, 스레드의 갯수는 하나 이상으로 만들 수 있게끔 설정한 후에 결과값을 살펴보죠~

앞에서 한번 언급했지만 서비스의 Behavior 를 적용하는 방법은 서비스 클래스에 특성으로 설정하는 방법과 config 파일에 설정하는 방법이 있습니다. 여기서는 클래스에 특성으로 설정하는 방법을 사용해보겠습니다.

서비스 클래스의 코드를 다음과 같이 굵은 글씨로 적용된 부분만을 추가해보죠~

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

    ... 생략 ...
}
```

이렇게만 수정한 후에 솔루션을 실행시켜 보면, 클라이언트 측 화면은 변화가 없지만 서버 측 결과 화면은 다음과 같이 변화된 것을 확인하실 수 있을겁니다.



인스턴스가 하나만 생성되었다는 점이죠. 아~ 그러고보니 동작한 스레드의 ID 값들이 모두 다른 것도 보이네요. 이 말은 곧, **하나의 인스턴스에 여러 개의 스레드가 생성되었다는** 것을 의미하는 것이겠죠.

앞에서 설정했던 InstanceContextMode 의 값과 ConcurrencyMode 의 값이 어떻게 서비스의 동시성에 적용되었는지 이해가 가실겁니다.