|  |  |  |
| --- | --- | --- |
|  | |  | | --- | | **WCF : Duplex 패턴을 이용한 비동기 호출 구현**http://blogimgs.naver.net/imgs/nblog/spc.gif[ASP .NET, AJAX](http://blog.naver.com/PostList.nhn?blogId=knight50&categoryNo=2&from=postList&parentCategoryNo=2) http://blogimgs.naver.net/imgs/nblog/spc.gif  2010/08/12 08:18  복사<http://blog.naver.com/knight50/80113277134>  [*전용뷰어 보기*](http://blog.naver.com/PostView.nhn?blogId=knight50&logNo=80113277134&beginTime=0&jumpingVid=&from=search&redirect=Log&widgetTypeCall=true&topReferer=http%3A%2F%2Fsearch.naver.com%2Fsearch.naver%3Fsm%3Dtab_hty.top%26where%3Dnexearch%26ie%3Dutf8%26query%3D%25EA%25B3%2584%25EC%2595%25BD%25EC%2597%2590%2BDuplex%26x%3D0%26y%3D0) |   **WCF Duplex 패턴을 이용한 비동기 호출 구현**  *오늘은 WCF를 이용하여 Duplex 패턴을 이용한 콜백(Callback)과 비동기(Asynchronous) 처리에 대해 알아보도록 한다.* **\*\* 올린 글이 너무 헷갈리는 것 같아서 부리나케 내용을 좀 바꾸었다..사설을 많이 늘어 놓았다..2007.01.12..이전에 보신 분들은 다시 보기 바란다.-\_-**  \*\* 기존 WCF에서 소개한 EchoService 샘플을 확장하여 구현해 볼 것이다.  **미리 이해해야 할 내용 : 비동기?, 콜백?**  먼저 이해해야 할 내용이라 내가 이해하고 있는 바를 아래에 적었다. 이해가 안가거나 잘 모르겠다면, 비동기/콜백에 대한 다른 훌륭한 글들을 찾아 보기 바란다.  콜백은 일반적으로 호출하고자 하는 함수에 파라미터 등으로 특정 함수의 포인터 등을 전달하여 호출 함수가 수행되는 동안 전달된 함수포인터에 해당하는 함수를 되호출을 하도록 하는 방식이다. 이때 되호출 되는 함수를 콜백함수 할 것이고, 그 되호출 행위를 콜백이라 할 것이다....  이 콜백이 비동기 작업에 필요하다. 왜? 비동기 호출은 클라이언트 서버 호출 후 잊어버리고 있다가 서버가 작업완료 하면 클라이언트에게 통지를 해주는 식의 처리 방식을 취한다. 그런데 이때, 서버가 통지를 해주려고 하는데, 어디로 어떻게 통지를 할지 알아야 할 것이다. "어디로" 는 호출시점에 서버가 알아 차리고 그 정보를 유지 하면 되지 싶은데, 무엇을 어떻게 통지 할지는? .... 사전에 이에 대한 약속을 서로 해야 한다. "서버야, 호출 다 하면 나(클라이언트)한테 있는 어떤 놈을 어떻게 호출해줘..." 이런 식의 약속 말이다. 여기서 "어떤 놈"이 바로 콜백 함수 가 될 것이다.  이런식으로 비동기 작업의 한 요소로서 콜백이 사용되고 있다.  일반적인 예로서, ASP.NET2.0 웹 서비스를 생성하게 되면 기본적으로 동기/비동기 방식 모두 호출이 가능하다. (웹 메서드가 GetTime()인 경우) 동기적으로 호출하려면 클라이언트가 프록시를 통해 서버의 메서드명과 동일한 GetTime()을 호출하게 되면 클라이언트는 서버가 작업을 완료한 후 리턴할 때까지 대기해야 만 한다. (다른 작업을 할 수 없다. 그래서 작업이 동기화 되는 것이다.)  비동기적으로 호출하려면, 자동 생성된 프록시 내의 GetTimeAsync() 식의 비동기용 메서드를 사용하면 된다.   비동기 호출을 하기 전에 호출 완료에 대한 서버로부터의 통보(?)를 받을 것이 필요한데, 이것이 바로 콜백이다. 그래서 비동기 호출 전에 반드시 콜백을 마련해 두고 호출 객체에 적절히 등록해 주는 과정이 필요하다. ASP.NET 2.0 웹 서비스에 대한 콜백은 이벤트 핸들러 형태로 제공된다. 그래서 호출 완료에 대한 이벤트 핸들러, 즉 콜백을 등록하면 되는 것이다.  클라이언트가 이 메서드를 호출하여 비동기 호출을 시작하면, 서버 측에서는 이 요청을 수신 즉시 리턴하고 난 뒤 따로 요청된 작업을 계속 처리하게 된다. 그러므로 클라이언트는 대기시간 없이(비동기적으로) 바로 다른 업무를 처리할 수 있고, 서버는 요청된 처리가 완료된 후 클라이언트 측에 콜백(함수)를 호출해주어서 작업이 완료되었음을 알려준다. 결국 클라이언트 측에서는 콜 백 함수에서 작업 완료에 대한 처리를 해주면 된다.  아래에 ASP.NET2.0 웹서비스에 대해 비동기 호출을 하는 클라이언트 측의 간단한 샘플 코드를 소개한다.  private void btnInvokeGetTime\_Click(object sender, EventArgs e) { MyWebService service = new MyWebService(); **service.GetTimeCompleted += new GetTimeCompletedEventHandler(this.GetTimeCompleted);** service.**GetTimeAsync**(); // 위 비동기 호출은 바로 리턴되므로, 아래의 코드가 바로 실행된다.. ...  }  // Event Handler : Callback void **GetTimeCompleted**(object sender, GetTimeCompletedEventArgs args) { //Display the return value MessageBox.Show(args.Result); }  WCF에는 서버와 클라이언트 간의 세가지 메시지 교환 또는 호출하는 패턴이 존재하고 이를 구현한 각각의 서비스 형태가 존재한다.   □ **Request/Reply (Service)** : 요청 메시지(호출)와 그에 대한 응답 메시지(리턴)를 교환하는 패턴이다. (기본) 호출이 있은 후에 그에 관련된 응답이 처리되어야 하는 방식으로서, 가장 일반적인 동기적 호출 방식에 사용되는 메시지 교환 패턴으로 요청 메시지와 응답 메시지는 상호관련성을 가지게 된다. 클라이언트가 요청을 하게 되면 서버로부터 응답이 오기 전에 다른 작업을 처리할 수 없는 방식이다. 작업이 동기화 되기 위해, 클라이언트에서 서버호출(대기시작) -> 서버 처리시작 -> 서버 처리완료 -> 클라이언트에 리턴 -> 클라이언트 다음 작업 처리  □ **One-Way (Service)** : 단 방향의 메시지 전달(호출) 패턴이다.  클라이언트가 서버의 작업 처리 완료 때까지 대기할 수 없는(필요가 없는) 경우에 사용하는 패턴 중의 하나로서 클라이언트가 요청을 전달하고 난 뒤에 바로 다른 작업을 할 수 있다. 요청에 대한 SOAP 실패가 나도 이 실패를 처리하지 않는다. 서버로부터 리턴되는 값도 없다. 단지 전달만 하고 처리가 되든 말든 신경 쓰지 않는 방식이다.  □ **Duplex(Two-way) (Service)** : 양 방향의 메시지 전달(호출) 패턴이다. 따라서 클라이언트와 서버가 서로간에 호출이 가능한 패턴이다.   서비스에서 클라이언트 측으로 호출이 가능하므로 이를 이용해 콜백을 처리하여, 비동기 처리를 가능하게 할 수 있는 패턴이라고 짐작이 가능하다.  따라서 Duplex 서비스의 구성을 말해보자면, 클라이언트에서 서비스로 One-Way + 서비스에서 클라이언트로 On-Way + Callback 정도로 표현이 가능할 것이고 따라서 그 작동 방식은 아래와 같을 것이다. 클라이언트에서 서비스로 One-Way 방식으로 메시지를 전달하고(호출하고), 서비스상에서 처리 완료 후 다시 클라이언트로 One-Way 방식으로 메시지를 전달하는( 콜백을 호출하는) 방식으로 처리한다.  **흠.. 글을 올리고 보니 쫌 헷갈리네... 그럼 비동기 처리를 위해서는 반드시 아래에 소개되는 방식대로 Duplex 패턴으로 구현해야 하는 것이야?.. 그건 아니다,**  기존 ASP.NET 웹 서비스 처럼 일반 웹메서드를 구현하기만 하면 클라이언트 측에서는 자동 생성된 프록시를 이용해 동기/비동기 호출이 모두 가능한 건 아니지만,  **WCF역시 일반 Request/Reply 패턴의 오퍼레이션을 정의/구현한 후 약간의 추가작업(?) 만을 곁들이면, 클라이언트 측에서 동기/비동기 호출 모두 가능해 진다. (비동기만을 원한다면 굳이 아래와 같이 Duplex 패턴에 맞게 구현하지 않아도 된다)** --> 여기에 대해서는 다음 글에서 소개하겠다.약속!필자도 사실, 비동기 구현에 촛점을 맞추다가..어찌하다..보니 Duplex로 촛점이 옮겨 졌다..-\_-!!!  혼동해 선 안되는게, 비동기 처리를 위해 반드시 Duplex 패턴 형식으로 구현해야 한다고 오해하면 안된다.  그럼 비동기 처리에 대한 구현은 둘 중에 어떤게 좋은가?.. 아까 말한 일반 오퍼레이션에 약간의 추가작업이 좋아? 아니면 Duplex가 좋아? 아주 논리적으로 답할 수 없지만, 일반적인 비동기 호출이 원하는 것이라면 그냥 "약간의 추가작업"이 여러가지 면에서 효율적일 거라 본다.  그러나 Duplex가 좀 더 복잡한 구현을 해야 하는 만큼 개발자에게 더 많은 융통성을 제공하므로 요구사항이 단순히 비동기 호출만을 요구하지 않는다면, Duplex를 고려해야 할 것이다.  언뜻 해 보니까, Duplex는 서비스 상에서 클라이언트로의 콜백을 원하는 시점에 임으로 호출 가능하지만, "약간의 추가작업"은 자동으로 처리되는 부분이 많아서 그 사이에 개발자가 껴 들 수 없는것으로 보인다. 고로 Duplex가 비동기 구현을 위해 보다 융통성 있는, 그리고 강력한(?) 방법이라 할 수 있다.  [생각해보면....] Duplex와 "약간의 추가작업"은 메시지 교환 패턴이 다르다. 그런데 둘 다 비동기 처리가 가능하다. 왜? 내부적으로 비동기라는 숙제를 해결하는 방법이 다르다고 본다. Duplex는 위에서 언급한 대로, 작업 완료 후 콜백 처리를 직접 서비스에서 클라이언트로 호출해 버리는 형태라 직관적이다. 그런데 "약간의 추가작업" 같은 경우는 서비스에서 클라이언트로 호출을 하는 패턴이 아닌데... 짐작 컨데, 모든 일은 클라이언트의 프록시 내에서 이루어 진다고 본다. 클라이언트의 프록시가 서비스 호출 시 이를 바로 리턴해 주고, 자신은 홀로 남아(별도의 스레드로서) 서비스의 작업 완료를 감시하다가 완료하면, 사전에 등록된 콜백함수을 호출 해주는 식의 "북치고 장구치고" 신공을 펼치는 것이 아닐까... 왜냐면, 서비스가 클라이언트로 호출을 할 수 없으므로 오로지 할 수 잇는 것은 완료후 리턴을 해 주는 것 밖에 없지 않은가...그렇다면 비동기 구현을 위해 이 리턴을 잡아서 콜백해주는 수 밖에..  자, 이제 본론으로 돌아가서 글의 주제인 Duplex 패턴 구현을 통해 비동기 처리하는 것에 대해 알아보도록 하자.  **WCF Duplex 패턴 구현 : 비동기,콜백 구현**  **1. Duplex 패턴을 사용하는 Service Contract 구현**  Duplex 패턴을 구현하기 위해서는 서버가 클라이언트 호출에 사용할 콜 백에 대한 정보를 정의에 포함시켜야 한다.  이는 클라이언트가 콜백을 받기 위해 사용해야 하는 인터페이스 정의와 그 인터페이스 타입을 콜백으로 사용하겠다는 속성 정의 작업을 나타낸다.  따라서, WCF 서비스 계약 정의 시에 아래와 같이, **콜백에 대한 인터페이스**를 정의하고, 그 타입을 **CallbackContract**속성에 부여함으로써 이 서비스 계약이 지원하는 클라이언트 콜백을 정의하게 된다.  **1.1. 서비스 계약 정의**  [ServiceContract(Namespace="http://xdn-dcpark/WCF/echo/", **CallbackContract=typeof(IEchoServiceCallback)**)] public interface IEchoService { …  **// 비동기 호출에 사용할 오퍼레이션 : 비동기를 위해 One-Way 방식을 사용해 호출 시 즉시 리턴하도록 함** **[OperationContract(IsOneWay = true)] void EchoJobStart(string jobName);** }  **// 콜백 인터페이스 정의 : 클라이언트에서 콜백을 받기 위해, 이 인터페이스를 구현해야 함.** **public interface IEchoServiceCallback {**  **[OperationContract(IsOneWay = true)] void EchoJobEnd(string result);  }**  **1.2 서비스 Endpoint 구성**  <service behaviorConfiguration="returnFaults" name="EchoService"> <!--<endpoint binding="wsHttpBinding" name="httpEndPoint" contract="IEchoService" />--> **<endpoint binding="wsDualHttpBinding" name="dualHttpEndPoint" contract="IEchoService" />** <endpoint binding="netTcpBinding" name="tcpEndPoint" contract="IEchoService" /> <endpoint binding="netNamedPipeBinding" name="namedPipeEndPoint" contract="IEchoService" />  <host> <baseAddresses> <add baseAddress="net.pipe://xdn-dcpark/Server\_Console/EchoService" /> <add baseAddress="net.tcp://xdn-dcpark:20000/Server\_Console/EchoService" /> <add baseAddress="http://xdn-dcpark/Server\_Console/EchoService" /> </baseAddresses> </host> </service> </services> <behaviors> <serviceBehaviors> <behavior name="returnFaults" > <serviceDebug includeExceptionDetailInFaults="true" /> <serviceMetadata httpGetEnabled="true" /> **<serviceThrottling maxConcurrentSessions="10" maxConcurrentInstances="10" maxConcurrentCalls="10" />** </behavior> </serviceBehaviors> </behaviors>  \*\* 기존의 서비스 계약 구성에서는 wsHttpBinding 바인딩을 사용하였지만, Duplex 패턴을 위해서는 wsHttpBinding 대신, Duplex 를 위한 **wsDualHttpBindign을 사용해야 한다.**  \*\* 서버에 가해지는 부하를 일정 수준 이하로 제한하기(Throttling : 스로틀링) 하기 위한 설정으로 **serviceThrottling** 구성을 추가할 수 있다. 이렇게 함으로써, 서버가 과다한 부하로 다운되거나 하는 것을 방지할 수 있다. (DoS와 같은 공격으로부터 보호하는 수단이기도 하다.)  위 샘플 구성에서는 최대 동시 세션 수 10개, 최대 동시 서비스 인스턴스 수 10개, 최대 동시 호출 수 10개로 제한하였다. **1.3 서비스 계약 구현**  그리고 서비스 계약 구현 시에는 서비스 계약 정의 시에 명시한 콜백 타입의 객체를 획득하여(?) 이를 호출해 줌으로써, 클라이언트로의 콜백을 구현할 수 있다.  [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession,ConcurrencyMode = ConcurrencyMode.Multiple)] public class EchoService : IEchoService { **IEchoServiceCallback callback = null;** string result = string.Empty;  // 서비스 인스턴스 내에서의 상태정보 테스트를 위한 테스트 전역변수 private int callCount;  public EchoService() { result = string.Empty; **// 클라이언트로 향하는 콜백 객체는 현재의 OperationContext 객체의 GetCallbackChannel을 통해 획득할 수 있다. // 아마도 서비스 인스턴스가 생성되면서 계약에 명시된 콜백을 위한 채널이 자동으로 생성되어, 쉽게 사용할 수 있게 하나 보다. callback = OperationContext.Current.GetCallbackChannel<IEchoServiceCallback>();** }  //비동기로 호출된 오퍼레이션 구현. public void EchoJobStart(string jobName) { //상태 정보 변경 ++callCount; result = jobName + " is completed!!!■CALLCOUNT:" + callCount.ToString(); System.Threading.Thread.Sleep(5000);  **// 클라이언트의 콜백을 호출함. callback.EchoJobEnd(result);**  **// 막!! 한번 더 콜백을 호출할 수도 있다.. (클라이언트가 콜백을 받고 바로 프록시를 닫지 않고 있다면,..) // callback.EchoJobEnd(result);**  }  **2. 클라이언트에서 콜백 구현하기 및 비동기 서비스 호출**  Callback을 정의한 서비스 계약에 대한 프록시 코드를 생성하게 되면,(svcutil.exe를 통해) 두개의 인터페이스가 포함되어 있다. 하나는 서비스용, 하나는 콜백용.  **2.1 콜백 인터페이스 구현** 그 콜백용 인터페이스를 상속 받아서 이를 구현한다.  public class EchoServiceCallbackHandler : **IEchoServiceCallback** { public void EchoJobEnd(string result) { **// 콜백이 호출되었을 경우 할 일.** } }  **2.2 서비스 호출**  그리고 프록시 객체를 이용해 (비동기) 서비스 호출을 한다.  //전역 클라이언트 프록시 객체 변수 internal EchoServiceClient asyncClient = null; private void btnCallHttpDual\_Click(object sender, EventArgs e) { try { **//InstanceContext 객체를 생성. 이때, 콜백 객체를 전달. System.ServiceModel.InstanceContext site = new System.ServiceModel.InstanceContext(new EchoServiceCallbackHandler());**  // 생성된 InstanceContext객체와, 사용할 Endpoint 명을 전달하여 프록시 객체 생성 asyncClient = new EchoServiceClient(site, "dualHttpEndPoint");  **// 비동기 서비스 호출 : OneWay 방식이므로 바로 리턴됨. asyncClient.EchoJobStart("AsyncEchoJob1");** } catch (TimeoutException timeProblem) { asyncClient.Abort();  } catch (System.ServiceModel.CommunicationException commProblem) { asyncClient.Abort(); } }  서비스 계약에 정의된 오퍼레이션 EchoJobStart은 OneWay 방식이다. 따라서 클라이언트에서 호출하자마자 바로 리턴 되므로, 클라이언트는 호출 후 즉시 대기 없이 다른 작업이 가능하다. 즉 비동기로 호출된다. 그리고 서버측의 서비스 인스턴스가 작업을 완료 한 후, 클라이언트 측의 콜백, EchoJobEnd()을 호출해 주게 되고, 클라이언트는 이 호출을 받음으로서 서버측의 EchoJobStart가 완료되었음을 인지하여 그에 적절한 다른 작업을 수행하게 된다.  3. 샘플 테스트  \*\*샘플은 첨부파일로 첨부되어 있다.  3.1 WCF 서비스 호스트 어플리케이션인, Server\_Console을 먼저 실행한다.  http://www.ensimple.net/enSimple/UpDir/study_netfr3/WCF6_1.gif  \*\* DualHttpBinding용 Endpoint가 비동기 호출을 제공하는 놈이다.  3.2 클라이언트를 실행한다. : WCFClient\_WinForm.exe  3.3. wsDualHttpBinding(Async)로 호출 버튼을 클릭하여 비동기 호출을 한다. : EchoJobStart() 호출  http://www.ensimple.net/enSimple/UpDir/study_netfr3/WCF6_2.gif  호출 후 바로 리턴되므로 다른 버튼으로 netTcpBinding, netNamedPipeBinding을 호출할 수 있다.  약 5초 후 클라이언트 측 콜백이 호출된다. : (서버에 의해) EchoJobEnd()가 호출됨.  http://www.ensimple.net/enSimple/UpDir/study_netfr3/WCF6_3.gif  \*\* CALLCOUNT 값이 세션 별로 호출 시 마다 증가하는 것을 확인할 수 있다. PerSession 모드 이므로 세션 동안 서비스 인스턴스가 유지되므로 그 동안 상태 정보를 유지할 수 있다. PerCall이면 CALLCOUNT 값은 무조건 1이다. 왜 그런지는 아래의 설명을 참조하도록 한다.  **서비스의 작동 행태(Behavior) : InstaceContextMode, ConcurrencyMode**  서비스 계약 시에 ServiceBehavior 속성 정의를 통해 서비스 인스턴스(실행 객체)의 작동 행태를 정의할 수 있다.  위의 샘플에서는, 아래와 같았다. [ServiceBehavior(**InstanceContextMode** = InstanceContextMode.PerSession, **ConcurrencyMode** = ConcurrencyMode.Multiple)]  **InstanceContextMode**  서비스 인스턴스가 언제 생성될지를 명시하는 속성으로서, 인스턴스의 생명주기 유형을 결정한다.  InstanceContextMode는 PerCall, PerSession, Single 이렇게 세게의 옵션이 존재한다.   □ **PerCall** : 클라이언트가 호출 할 때마다 (새로운) 서비스 인스턴스가 생성되고 호출이 완료될 때, 소멸한다. 호출 마다 새로운 인스턴스가 생성되므로 호출 간에 서비스 인스턴스 내에서(메모리 상에서) 상태 정보를 유지하는 것은 불가하다.  □ **PerSession** : 세션(프록시가 생성되고, Close 될 때 까지의 시간) 마다 새로운 서비스 인스턴스가 생성된다. 세션이 종료(Close) 되면 소멸한다. 세션 내에서 일어나는 호출들 간에 서비스 인스턴스 내에서(메모리 상에서) 서로 상태 정보를 공유할 수 있다.  □ **Single** : 오직 하나의 서비스 인스턴스 만이 모든 클라이언트로 부터의 호출을 처리한다. 서비스 인스턴스가 소멸되기 전 까지 모든 클라이언트로부터의 모든 호출은 서로 상태 정보를 공유할 수 있다.   \*\* 첨부된 샘플에 간단한 상태 정보 유지에 대한 처리가 포함되어 있다. PerCall, PerSession에 대한 차이를 확인해 보기 바란다. \*\* 이와 관련해서 아주 훌륭한 글이 있다. 참조 바란다. 유경상 님의 블로그 : <http://www.simpleisbest.net/articles/1731.aspx>  **ConcurrecyMode**  ConcurrencyMode는 서비스 인스턴스가 멀티스레드로 작동할 것이지 여부를 지정하는 속성이다.  Multiple, Single, Reentrance 이렇게 세가지 모드를 지원한다.  □ **Multiple** : **Multi-threaded** 이다. 그래서 여러 호출을 동시에 처리할 수 있다. 그래서 여러 호출이 하나의 상태 정보를 동시에 공유할 수도 있다. 이때 그 상태 정보의 동기화는 개발자의 책임이다.  □ **Single** : **Single-threaded**이다. 한번에 하나의 호출만 처리한다. 서비스 인스턴스가 호출을 처리 중이면, 다른 호출은 이전 호출이 완료될 때까지 대기하게 될 것이다.  □ **Reentrance** : **Single-threaded**이다. 그러나 서비스 인스턴스가 **호출 처리 중에 다른 호출이 요청되면 이를 수용**할 수 있는 방식이다.  호출 처리 중에 다시 다른 호출을 처리하도록 하는 것이다. 별도의 스레드로 다른 호출을 처리하는 것이 아니라 하나의 스레드 내에서 여러 호출을 처리하는 것이므로 호출 간에 상태정보를 동기화하는 것이 중요할 수 있으며 역시 개발자 책임이다. 이는 서비스 내에서 다른 서비스를 호출하는 경우에 사용하게 되는데,  클라이언트에서 서비스1 호출 ▷서비스1에서 서비스2 호출 ▷서비스2에서 다시 서비스1을 호출(Call back) 과 같은 경우, 서비스 1이 Reentrance가 아니면 Dead lock이 발생하게 될 것이다..-\_-  *자 오늘은 여기까지 하자.. 시스템 오픈 때문에 밤 새고 나니 머리가 새~ 하다.*  *끝.*  **첨부파일** : http://www.ensimple.net/enSimple/images/file.gif[AsyncCallback(Duplex)WCFApp.zip (82Kbytes)](http://www.ensimple.net/enSimple/download.aspx?cnum=407&b_id=study_netfr3)  **[출처]** [WCF : Duplex 패턴을 이용한 비동기 호출 구현](http://blog.naver.com/knight50/80113277134)|**작성자** [좋은생각](http://blog.naver.com/knight50) |