**Service 구현하기**

먼저 새로운 프로젝트를 선택하고, 콘솔 응용 프로그램을 선택한다. 그 후에 System.ServiceModel 이라는 네임스페이스를 선택해서 추가한다. 네임스페이스를 참조 했다면 다음에 새로운 ChatService 라는 클래스를 추가하도록 하자. ChatService라는 클래스가 추가가 되었다면 그럼 먼저 클라이언트에서 서버로 보내고 서버에서 클라이언트로 콜백할 서비스 인터페이스를 정의해 보도록 하겠다.

|  |
| --- |
| #region 1. 메세지관련Contract InterFace (클라이언트->서버)  /// ////////////////////////////////////////////////////////////////////////  /// InstanceContextMode.PerSession이므로세션을사용함  /// -->그렇기때문에신뢰할수있는전송세션이용해야함  /// ////////////////////////////////////////////////////////////////////////  [ServiceContract(SessionMode = SessionMode.Required, CallbackContract = typeof(IChatCallback))]  interface IChat  {  /// ////////////////////////////////////////////////////////////////////////  /// IsTerminating : 메서드가호출되고반환되는시점에세션을자동종료될지여부  /// IsInitiating : 세션이최초로맺어지는단계에서호출이가능여부  /// ////////////////////////////////////////////////////////////////////////  [OperationContract(IsOneWay = false, IsInitiating = true, IsTerminating = false)]  string[] Join(string name);  [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]  void Say(string msg);  [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]  void Whisper(string to, string msg);  [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = true)]  void Leave();  }  #endregion  #region 2. 클라이언트에콜백할CallBackContract (서버->클라이언트)  interface IChatCallback  {  [OperationContract(IsOneWay = true)]  void Receive(string senderName, string message);  [OperationContract(IsOneWay = true)]  void ReceiveWhisper(string senderName, string message);  [OperationContract(IsOneWay = true)]  void UserEnter(string name);  [OperationContract(IsOneWay = true)]  void UserLeave(string name);  }  #endregion |

<리스트 1> Service Contract 정의

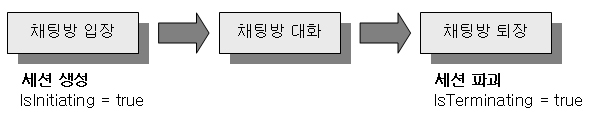
IChat 이라는 인터페이스와 IChatCallback이라는 두 개의 인터페이스를 추가하였다. 먼저 IChat 인터페이스 먼저 살펴보도록 하자. ServiceContract에 다음과 같은 옵션을 추가하였다.

[ServiceContract(SessionMode = SessionMode.Required, CallbackContract = typeof(IChatCallback))]

즉, 세션을 요구한다는 옵션을 준 것이고, CallbackContract 옵션은 클라이언트로 메시지를 보낼 때의 계약을 IChatCallback 인터페이스로 지정하고 있는 것이다. 그 다음 각 메서드 위에 선언된 OperationContract의 옵션들을 살펴보자.

|  |  |
| --- | --- |
| 이름 | 설명 |
| IsOneWay | 단 방향 통신으로 설정할지의 여부 |
| IsTerminating | 메서드가 호출되고 반환되는 시점에 세션을 자동 종료될지의 여부 |
| IsInitiating | 세션이 맺어지는 최초에만 호출할 수 있게 할지의 여부 |

IsOneWay는 앞에서 설명했듯이 단 방향으로 메시지를 전달하게 된다. 별도의 요청을 기다릴 필요가 없기 때문에 서비스를 가볍게 하기 위한 필수적인 옵션이 될 것이다. IsTerminating과 IsInitiating 옵션은 인스턴스 관리와 상호 연결되는 옵션이다. 먼저 채팅을 하기 위한 순서를 살펴보자. 즉, 처음에 세션이 맺어지는 최초에만 호출을 하게 하려면 IsInitiating 속성을 true로 설정을 해야 할 것이고, 마지막으로 호출되는 메서드에서 자동으로 연결을 종료하기 원한다면 IsTerminating 속성을 true로 설정하면 될 것이다. 두 개의 속성 다 기본 값은 false이다. 다음 그림을 참고한다면 앞에서 설정한 OperationContract 옵션을 이해하는데 수월할 것이다.

  
< 그림 9> 채팅 프로그램으 흐름

이번 강좌에서는 위의 WCF의 디자인 패턴과 Contract의 설정들이다. 채팅에 관한 로직들은 지면상 다 다룰 수는 없다. 때문에 필자가 자세하게 주석을 달아 놓았기 때문에 주석들을 참고한다면 충분히 이해할 수 있을 것이다. 그럼 이제 위의 인터페이스를 상속받은 클래스의 구현을 살펴보도록 하자.

|  |
| --- |
| #region 3. 메세지타입&이벤트Agrs정의  //메세지타입  public enum MessageType { Receive, UserEnter, UserLeave, ReceiveWhisper };  //ChatEventArgs 이벤트Arg에포함될항목  public class ChatEventArgs : EventArgs  {  public MessageType msgType;  public string name;  public string message;  }  #endregion  #region 4. 실제서비스구현(클라이언트->서버)  /// /////////////////////////////////////////////////////////////////////  /// InstanceContextMode -> PerSession:세션과같은인스턴스  /// ConcurrencyMode -> Multiple: 멀티스레드지원  ////////////////////////////////////////////////////////////////////////  [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession, ConcurrencyMode = ConcurrencyMode.Multiple)]  public class ChatService : IChat  {  #region 4.1. 전역변수및COMMON 메서드  //동기화작업을위해서가상의객체생성  private static Object syncObj = new Object();  //콜백이벤트로실행  IChatCallback callback = null;  //콜백델리게이트와이벤트선언  public delegate void ChatEventHandler(object sender, ChatEventArgs e);  public static event ChatEventHandler ChatEvent;  //키와값을가지고있는컬렉셔, Key=string 타입, 값은ChatEventHandler를가지고있는다.  static Dictionary<string, ChatEventHandler> chatters = new Dictionary<string, ChatEventHandler>();  //키가될이름  private string name;  //채팅이벤트  private ChatEventHandler myEventHandler = null;  /// <summary>  /// 전체클라이언트들에게이벤트를전달한다.  /// </summary>  /// <param name="e"></param>  private void BroadcastMessage(ChatEventArgs e)  {  //이벤트  ChatEventHandler temp = ChatEvent;  if (temp != null)  {  //현재이벤트들을전달한다.  foreach (ChatEventHandler handler in temp.GetInvocationList())  {  handler.BeginInvoke(this, e, new AsyncCallback(EndAsync), null);  }  }  }  /// <summary>  /// 클라이언트에게이벤트를전달한다.  /// </summary>  /// <param name="ar"></param>  private void EndAsync(IAsyncResult ar)  {  ChatEventHandler d = null;  try  {  System.Runtime.Remoting.Messaging.AsyncResult asres = (System.Runtime.Remoting.Messaging.AsyncResult)ar;  d = ((ChatEventHandler)asres.AsyncDelegate);  d.EndInvoke(ar);  }  catch  {  ChatEvent -= d;  }  }  /// <summary>  /// 이벤트발생  /// </summary>  /// <param name="sender"></param>  /// <param name="e"></param>  private void MyEventHandler(object sender, ChatEventArgs e)  {  try  {  //클라이언트에게보내기  switch (e.msgType)  {  case MessageType.Receive:  callback.Receive(e.name, e.message);  break;  case MessageType.ReceiveWhisper:  callback.ReceiveWhisper(e.name, e.message);  break;  case MessageType.UserEnter:  callback.UserEnter(e.name);  break;  case MessageType.UserLeave:  callback.UserLeave(e.name);  break;  }  }  catch//에러가발생했을경우  {  Leave();  }  }  #endregion  #region 4.2. JOIN 메서드  /// <summary>  /// \* JOIN 메서드  /// 특정한사용자가방에처음들어왔을때사용자의이름을받아서  /// 처리한다.  /// </summary>  /// <param name="name">사용자이름</param>  /// <returns>같은이름이없을경우에는사용자리스트를반환한다</returns>  public string[] Join(string name)  {  myEventHandler = new ChatEventHandler(MyEventHandler);  lock (syncObj)  {  if (!chatters.ContainsKey(name))//이름이기존채터에있는지검색한다.  {  //이름과이벤트를추가한다.  this.name = name;  chatters.Add(name, MyEventHandler);  //사용자에게보내줄채널을설정한다.  callback = OperationContext.Current.GetCallbackChannel<IChatCallback>();  //UserEnter 라는이벤트를전달한다  ChatEventArgs e = new ChatEventArgs();  e.msgType = MessageType.UserEnter;  e.name = name;  BroadcastMessage(e);  //델리게이터추가  ChatEvent += myEventHandler;  //사용자리스트를보내준다.  string[] list = new string[chatters.Count];  lock (syncObj)  {  chatters.Keys.CopyTo(list, 0);  }  return list;  }  else //이미사용자가사용하고있는이름일경우  {  return null;  }  }  }  #endregion  #region 4.3. Say메서드  public void Say(string msg)  {  ChatEventArgs e = new ChatEventArgs();  e.msgType = MessageType.Receive;  e.name = this.name;  e.message = msg;  BroadcastMessage(e);  }  #endregion  #region 4.4. 귓속말  public void Whisper(string to, string msg){  ChatEventArgs e = new ChatEventArgs();  e.msgType = MessageType.ReceiveWhisper;  e.name = this.name;  e.message = msg;  try  {  ChatEventHandler chatterTo;  lock (syncObj)  {  chatterTo = chatters[to];  }  chatterTo.BeginInvoke(this, e, new AsyncCallback(EndAsync), null);  }  catch (KeyNotFoundException)  {  }  }  #endregion  #region 4.5. 방나가기  public void Leave(){  if (this.name == null) return;  lock (syncObj)  {  chatters.Remove(this.name);  }  ChatEvent -= myEventHandler;  //새로운이벤트발생  ChatEventArgs e = new ChatEventArgs();  e.msgType = MessageType.UserLeave;  e.name = this.name;  BroadcastMessage(e);  }  #endregion  }  #endregion |

<리스트 2> Service 구현 코드

큰 개념만 살펴 보도록 하겠다. 먼저 서비스는 Static 전역변수를 만들어서 클라이언트를 관리한다. 그리고 특정 요청이 있을 때 그 메시지를 이벤트로 전달하게 된다. 클라이언트에서 서비스로의 요청하는 동작은 Join, Say, Whisper, Leave 4개의 메서드로 구현되고, 서버에서 클라이언트로 콜백 되는 메서드는 Receive, ReceiveWhisper, UserEnter, UserLeave로 정의 되어있다. 그럼 이제 App.Config 파일을 추가해서 Endpoint를 설정하고, Program.cs 서비스를 작동시켜 보도록 하자.

|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?>  <configuration>  <appSettings>  <add key="addr" value="net.tcp://localhost:22222/chatservice" />  </appSettings>  <system.serviceModel>  <services>  <service name="NikeSoftChat.ChatService">  <endpoint address=""  binding="netTcpBinding"  bindingConfiguration="DuplexBinding"  contract="NikeSoftChat.IChat" />  </service>  </services>  <bindings>  <netTcpBinding>  <binding name="DuplexBinding" maxConnections="10000">  <reliableSession enabled="true" />  <security mode="None" />  </binding>  </netTcpBinding>  </bindings>  </system.serviceModel>  </configuration> |

<리스트 3> App.Config설정

|  |
| --- |
| static void Main(string[] args){  //Address  Uri uri = new Uri(ConfigurationManager.AppSettings["addr"]);  //Contract-> Setting  //Binding -> App.Config  ServiceHost host = new ServiceHost(typeof(NikeSoftChat.ChatService), uri);  //오픈  host.Open();  Console.WriteLine("채팅서비스를시작합니다. {0}", uri.ToString());  Console.WriteLine("멈추시려면엔터를눌러주세요..");  Console.ReadLine();  //서비스  host.Abort();  host.Close();  }  } |

<리스트 4> Program.cs 코드

App.Config에 보면 DuplexBinding이라고 해서 Binding의 옵션을 주고 있다. Standard Binding이라고 해서 Binding의 설정을 바꾸는 것이 불가능 한 것이 아니라는 것을 보여주고 있는 것이다. 지금 설정을 모두 Configuration으로 하여 코드를 작성하였지만 cs 코드로도 위와 같은 메시지를 작성하는 것이 가능하다. 하지만 차후에 서비스 관리를 위해서 Configuration을 이용하는 것을 추천하는 바이다.

이렇게 코드를 모두 작성 되었다면 서비스 프로그램은 모두 작성된 것이다. 서비스를 정리해 보자면 Per-Session 방식으로 인스턴스 관리를 하고 있고, netTcpBinding을 이용해서 Duplex통신을 하는 서비스로 정리할 수 있다.

**Client 구현하기**

앞서 설명했듯이 이번 채팅 프로그램의 핵심은 WCF의 서비스를 구현하는 디자인과 설정이다. 지면관계상 클라이언트의 코드는 모두 다루지 못한다. 소스를 보면서 학습을 하기 원한다면 필자의 홈페이지 HOONS 닷넷(http://www.hoonsbara.com)에서 와서 채팅 프로그램 소스를 다운 받아 학습할 것을 권한다. 이번 클라이언트 프로그램은 WCF와 연동하는 부분만 살펴보도록 하자. 먼저 클라이언트의 프로그램은 Windows C#으로 구성 되어있다. 클라이언트에서는 Add Service Reference 메뉴를 통해서 프록시 클래스를 자동 생성해도 되지만 이번 예제에서는 클라이언트에서 통신하는 부분을 직접 작성하였다. 통신 부분의 코드를 살펴 보도록 하자.

|  |
| --- |
| public void Receive(string senderName, string message){  AppendText(senderName + ": " + message + Environment.NewLine);  }  public void ReceiveWhisper(string senderName, string message){  AppendText(senderName + " whisper: " + message + Environment.NewLine);  }  public void UserEnter(string name){  AppendText("User " + name + " enter at " + DateTime.Now.ToString() + Environment.NewLine);  lstChatters.Items.Add(name);  }  public void UserLeave(string name){  AppendText("User " + name + " leave at " + DateTime.Now.ToString() + Environment.NewLine);  lstChatters.Items.Remove(name);  AdjustWhisperButton();  } |

<리스트 5> 채팅 클라이언트 코드

ChatForm은 채팅 프로그램의 메인 폼이고, 앞에서 정의한 IChatCallback 인터페이스를 상속 받았다. 그리고 앞의 코드들은 모두 채팅 서비스 서버에서 메시지가 왔을 때 실행되는 메서드들이다. 만약 클라이언트가 오랜 시간동안 요청을 보내지 않았을 경우에는 서버에서는 세션이 자동으로 사라지게 되어 있다. 그 시간은 기본적으로 설정이 되어 있으며 사용자가 Behaviors의 Throttling 구성을 통해서 설정할 수 있다.