**Chapter4. 계약과 직렬화**

|  |
| --- |
| WCF 서비스 : 자신이 제공하는 다양한 기능을 클라이언트에게 알리기 위한 방법으로 계약을 사용한다.  계약에는 서비스가 제공하는 메소드들과 이들이 요구하는 매개변수들이 데이터 타입, 리턴 타입이 포함된다.   * 이를 서비스 계약이라 함   자신이 사용하는 커스텀 데이터 타입에 대한 정보를 알려줄 필요가 있다.   * 이를 데이터 계약이라 함   \*) 전체 서비스 계약에는 서비스 계약과 데이터 계약이 포함되어 있다.    닷넷 데이터 타입을 XML 스키마로 바꾸는 직렬화/역직렬화에 대한 개념을 이해하자. |

1. **서비스 계약**

|  |
| --- |
| 작업 계약  인터페이스 계약과 클래스 계약  계약에 대한 별칭  서비스 계약 상속 |

\*) 서비스 계약 정의 시 public, internal 등의 닷넷 영역 한정자는 닷넷 플랫폼에서만 의미를 가질 뿐 서비스에 의해 노출

되는 경우에는 아무런 의미를 갖지 않는다.

따라서 굳이 WCF에서는 서비스 계약에 public 한정자를 붙일 필요가 없다.

**작업계약**

계약 내에 정의된 서비스의 작업(operation)들은 닷넷의 메소드와 매핑된다.

즉 WCF에서는 메소드가 서비스의 작업이 되며, OperationContract 특성을 메소드에 명시함으로써 정의된다.

|  |
| --- |
| // 서비스 계약 정의  [ServiceContract]  interface IBookStore  {  [OperationContract]  int Order(string userid, string isbn, int amount);  bool Cancel(int orderid);  } |

\*) 실제 클라이언트 측에서 프록시 코드를 자동 생성하면 IBookStore 인터페이스의 Order 메서드만 존재한다.

**인터페이스 계약과 클래스 계약**

클래스 또한 ServiceContract 특성이 명시되어 있다면 WCF의 계약으로 정의될 수 있다.

|  |
| --- |
| using System;  using System.ServiceModel;  using System.ServiceModel.Description;  namespace ClassContractService  {  [ServiceContract(Namespace="http://noname.com/bookstore")]  class BookStore  {  private static int id = 0;  // 도서의 단가를 반환한다.  private int GetUnitPrice(string isbn)  {  // 고정값... -\_-;  return 35000;  }  // 도서 주문  [OperationContract]  protected int Order(string userid, string isbn, int amount)  {  int total = GetUnitPrice(isbn) \* amount;  id += 1;  // 주문 처리 로직  // (단순 출력만... -\_-; )  Console.WriteLine(  "Book Order: orderid={0}\n userid={1}, isbn={2}, amount={3}, total={4}",  id, userid, isbn, amount, total);  // 주문 ID 반환  return id;  }  // 주문 취소  [OperationContract]  public bool Cancel(int orderid)  {  // 주문 취소 로직  // (단순 출력만... ^^;  Console.WriteLine("Cancel Order: orderid={0}", orderid);  return true;  }  }  class ServiceApp  {  static void Main(string[] args)  {  Console.WriteLine("Class Contract Service Example...");  ServiceHost host = new ServiceHost(  typeof(BookStore),  new Uri("http://localhost/wcf/example/bookstoreservice"));  // 종점 추가  host.AddServiceEndpoint(  typeof(BookStore),  new BasicHttpBinding(),  "");  // WSDL 을 위한 HTTP GET enable  ServiceMetadataBehavior behavior = new ServiceMetadataBehavior();  behavior.HttpGetEnabled = true;  host.Description.Behaviors.Add(behavior);  host.Open();  Console.WriteLine("Press Any Key to Stop Service...");  Console.ReadKey(true);  host.Close();  }  }  } |

\*) 클래스가 서비스 계약으로 사용되면 계약 인터페이스 대신 계약 클래스를 그대로 사용하여 호스팅 코드를 작성하면

된다.

\*) 클라이언트는 프록시를 생성하게 되면 서비스 계약이 인터페이스인지 혹은 클래스인지 구분하지 않는다. 단, 서비스가

노출하는 계약은 클라이언트에게 항상 인터페이스로 간주되고 계약의 이름을 이용하여 클라이언트에서 사용할 인터페

이스를 선언하도록 되어 있다.

|  |
| --- |
| // BookStore Class 계약에 대한 클라이언트 측 인터페이스 정의  [ServiceContractAttribute(Namespace="http://noname.com/bookstore")]  publc interface BookStore  {  [OperationContractAttribute(…) ]  int Order(string userid, string isbn, int amount);  [OperationContractAttribute(…) ]  Bool Cancel(int ordered);  } |

🡺 ASMX를 통신 메커니즘으로 사용했었고 이것을 WCF 기반의 시스템으로 마이그레이션하고자 하는 경우 사용될 수 있

다.

**계약에 대한 별칭**

ServiceContract 특성과 OperationContract 특성은 모두 Name 속성을 가지고 있고, 별칭을 설정하는 데 사용된다.

Name속성은 메타데이터를(WSDL) 통해 서비스 외부에 노출되는 계약의 이름과 작업의 이름을 설정하는 데 사용된다.

|  |
| --- |
| using System;  using System.ServiceModel;  using System.ServiceModel.Description;  using System.Collections;  namespace ContractAliasingServiceApp  {  [ServiceContract(Namespace = "http://noname.com/bookstore", Name = "IMyBookStore")]  interface IBookStore  {  [OperationContract(Name="PlaceOrder")]  int Order(string userid, string isbn, int amount);  [OperationContract(Name="CancelOrder")]  bool Cancel(int orderid);  [OperationContract(Name="SearchByTitle")]  string[] Search(string title);  [OperationContract(Name="SearchByTitleAndAuthor")]  string[] Search(string title, string author);  }  // 서비스 타입 구현  [ServiceBehavior(IncludeExceptionDetailInFaults = true)]  class BookStoreService : IBookStore  {  private static int id = 0;  private static string[] titles = new string[] {  "WCF 10일만에 정복", "10일동안 하면 개똥이 만큼 WCF 한다", "WCF Programming",  "Programming WCF", "Learning WCF", "WCF Step by Step" };  private static string[] authors = new string[] {  "김정복", "김개똥", "김혜영",  "김영철", "김철수", "김말똥" };  // 도서 단가 반환  // (고정 값... -\_-; )  private int GetUnitPrice(string isbn)  {  return 35000;  }  // 도서 주문  public int Order(string userid, string isbn, int amount)  {  int total = GetUnitPrice(isbn) \* amount;  id += 1;  // 주문 처리 로직  // (단순 출력만... -\_-; )  Console.WriteLine(  "Book Order: orderid={0}\n userid={1}, isbn={2}, amount={3}, total={4}",  id, userid, isbn, amount, total);  // 주문 ID 반환  return id;  }  // 주문 취소  public bool Cancel(int orderid)  {  // 주문 취소 로직  // (단순 출력만... ^^; )  Console.WriteLine("Cancel Order: orderid={0}", orderid);  return true;  }  // 도서명 검색  public string[] Search(string title)  {  // 단순 검색 -\_-;  ArrayList results = new ArrayList();  foreach (string t in titles) {  if (t.StartsWith(title) == true) {  results.Add(t);  }  }  return (string[])results.ToArray(typeof(string));  }  // 도서명, 저자 검색  public string[] Search(string title, string author)  {  // 단순 검색 -\_-;  ArrayList results = new ArrayList();  for(int i=0; i < titles.Length; i++) {  if (titles[i].StartsWith(title) == true && authors[i].StartsWith(author)) {  results.Add(titles[i]);  }  }  return (string[])results.ToArray(typeof(string));  }  }  class ServiceApp  {  static void Main(string[] args)  {  Console.WriteLine("Contract Aliasing Service Example...");  ServiceHost host = new ServiceHost(  typeof(BookStoreService),  new Uri("http://localhost/wcf/example/bookstoreservice"));  // 종점 추가  host.AddServiceEndpoint(  typeof(IBookStore),  new BasicHttpBinding(),  "");  // WSDL 을 위한 HTTP GET enable  ServiceMetadataBehavior behavior = new ServiceMetadataBehavior();  behavior.HttpGetEnabled = true;  host.Description.Behaviors.Add(behavior);  host.Open();  Console.WriteLine("Press Any Key to Stop Service...");  Console.ReadKey(true);  host.Close();  }  }  } |

\*) 별칭을 사용한 계약에 대한 클라이언트 프록시 코드

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Text;  using ContractAliasingClientApp.BookStore;  namespace ContractAliasingClientApp  {  class ClientApp  {  static void Main(string[] args)  {  Console.WriteLine("BookStore (Contract Aliasing) Client Application");  int orderid;  using (MyBookStoreClient proxy = new MyBookStoreClient()) {  orderid = proxy.PlaceOrder("GoPl", "999-9-999-99999-9", 2);  Console.WriteLine("Place Order. Order ID = {0}", orderid);  string[] results = proxy.SearchByTitle("WCF");  Console.WriteLine("Search result :");  foreach (string title in results) {  Console.WriteLine("\t{0}", title);  }  }  }  }  } |

\*) 메소드 오버로드와 별칭

닷넷 언어들은 오버로딩을 지원한다. 하지만 서비스는 특정 기술을 가정하지 않는다.

따라서 서비스 계약에는 오버로딩을 포함시킬 수 없다. 이 때 별칭을 사용해 다른 이름을 계약에 적용시킬 수 있다.

**서비스 계약 상속**

\*) 인터페이스 상속

|  |
| --- |
| using System;  using System.ServiceModel;  using System.ServiceModel.Description;  using System.Collections;  namespace ServiceInheritanceServiceApp  {  [ServiceContract(Namespace="http://noname.com/bookstore")]  interface IBookStore  {  [OperationContract]  int Order(string userid, string isbn, int amount);  [OperationContract]  bool Cancel(int orderid);  [OperationContract]  string[] Search(string title);  }  [ServiceContract(Namespace="http://noname.com/bookstoreV2")]  interface IBookStore2 : IBookStore  {  [OperationContract]  string[] SearchEx(string title);  }  // 서비스 타입 구현  [ServiceBehavior(IncludeExceptionDetailInFaults = true)]  class BookStoreService : IBookStore  {  private static int id = 0;  protected static string[] titles = new string[] {  "WCF 10일만에 정복", "10일동안 하면 개똥이 만큼 WCF 한다", "WCF Programming",  "Programming WCF", "Learning WCF", "WCF Step by Step" };  protected static string[] authors = new string[] {  "김정복", "김개똥", "김혜영",  "김영철", "김철수", "김말똥" };  // 도서 단가 반환  // (고정 값... -\_-; )  private int GetUnitPrice(string isbn)  {  return 35000;  }  // 도서 주문  public int Order(string userid, string isbn, int amount)  {  int total = GetUnitPrice(isbn) \* amount;  id += 1;  // 주문 처리 로직  // (단순 출력만... -\_-; )  Console.WriteLine(  "Book Order: orderid={0}\n userid={1}, isbn={2}, amount={3}, total={4}",  id, userid, isbn, amount, total);  // 주문 ID 반환  return id;  }  // 주문 취소  public bool Cancel(int orderid)  {  // 주문 취소 로직  // (단순 출력만... ^^; )  Console.WriteLine("Cancel Order: orderid={0}", orderid);  return true;  }  // 도서명 검색  public string[] Search(string title)  {  // 단순 검색 -\_-;  ArrayList results = new ArrayList();  foreach (string t in titles) {  if (t.StartsWith(title) == true) {  results.Add(t);  }  }  return (string[])results.ToArray(typeof(string));  }  }  // 파생 인터페이스에 대한 서비스 타입 구현  [ServiceBehavior(IncludeExceptionDetailInFaults = true)]  class BookStoreService2 : BookStoreService, IBookStore2  {  // 확장된 도서명 검색 -\_-;  public string[] SearchEx(string title)  {  ArrayList results = new ArrayList();  foreach (string t in titles) {  if (t.IndexOf(title) >= 0) {  results.Add(t);  }  }  return (string[])results.ToArray(typeof(string));  }  }  class ServiceApp  {  static void Main(string[] args)  {  Console.WriteLine("Contract Inheritance Service Example...");  ServiceHost host = new ServiceHost(  typeof(BookStoreService2),  new Uri("http://localhost/wcf/example/bookstoreservice"));  // 종점 추가  host.AddServiceEndpoint(  typeof(IBookStore2),  new BasicHttpBinding(),  "");  // WSDL 을 위한 HTTP GET enable  ServiceMetadataBehavior behavior = new ServiceMetadataBehavior();  behavior.HttpGetEnabled = true;  host.Description.Behaviors.Add(behavior);  host.Open();  Console.WriteLine("Press Any Key to Stop Service...");  Console.ReadKey(true);  host.Close();  }  }  } |

\*) 구현상속