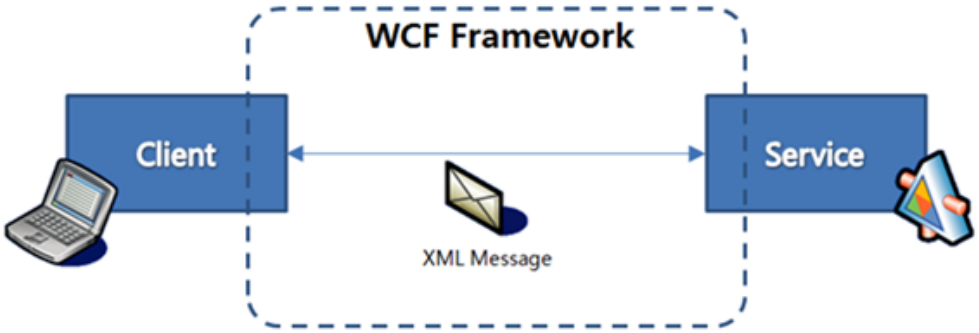
Chapter1. WCF 맛보기

01. WCF 걸음마

서비스 종점 – 서비스 종점을 구성하는 서비스 주소, 바인딩, 계약의 개념 서비스 구현에 필요한 서비스 호스트 서비스의 클라이언트를 구성하는 요소

WCF란?

- 웹 서비스 기반의 통신 프레임워크
  (callee, service) <==> (caller, client)
- 클라이언트와 서비스 사이에서 상호작용하며 둘 사이의 통신을 보다 쉽고 강력하게 해 주는 역할
- 메시징 프레임워크다.
  WCF기반을 통해 클라이언트와 서비스는 XML을 주고받는다.



[WCF 클라이언트와 서비스 그리고 WCF의 관계]

주소, 바인딩, 계약의 개념

WCF의 ABC (Address, Binding, Contract)

WCF에서의 서비스 주소란?

클라이언트가 서비스를 호출하기 위해 사용되는 주소
일반적으로 말하는 인터넷 주소
바인딩이 어떤 프로토콜을 사용하는 가에 따라 그 모양새가 조금씩 차이가 날수 있음을 유의
Ex) http://www.contoso.com/wcf/testservice.svc
net.tcp://appserver:2100/data/stockinfo.svc
net.msmq://server/AsyncService

WCF에서의 바인딩

WCF 서비스를 네트워크를 통해 호출할 때 고려해야 할 다양한 요소들의 집합

네트워크 통신을 할 때 고려해야 할 사항 - 어떤 프로토콜을 사용할 것인가? (TCP, HTTP, FTP 등) - 데이터 포맷은 어떤 방식을 사용할 것인가? (바이너리, 텍스트, MIME 등) - 네트워크 보안을 적용할 것인가? 적용한다면 어떤 보안 방식을 취할 것인가?(SSL, 암호화 등) - 트랜잭션 처리, 비동기 전송 등의 진보된 기능을 사용할 것인가?

즉, WCF에서의 바인딩은 통신에 필요한 다양한 요소들을 정의

- WCF 바인딩은 HTTP,. TCP, 명명된 파이프, MMSQ, P2P 등의 다양한 프로토콜 중 하나를 사용하면서 인터넷 표준, 바이너리/텍스트 메시지 인코딩 등의 네트워킹에 필요한 요소를 켜거나 끌 수 있다.

WCF에서의 계약

서비스에 대한 인터페이스

어떤 기능을 제공하고 각 기능의 매개변수가 어떠한지를 기술

```
public interface IBankingContract
{
    decimal Deposit(string account_id, decimal amount);
}
```

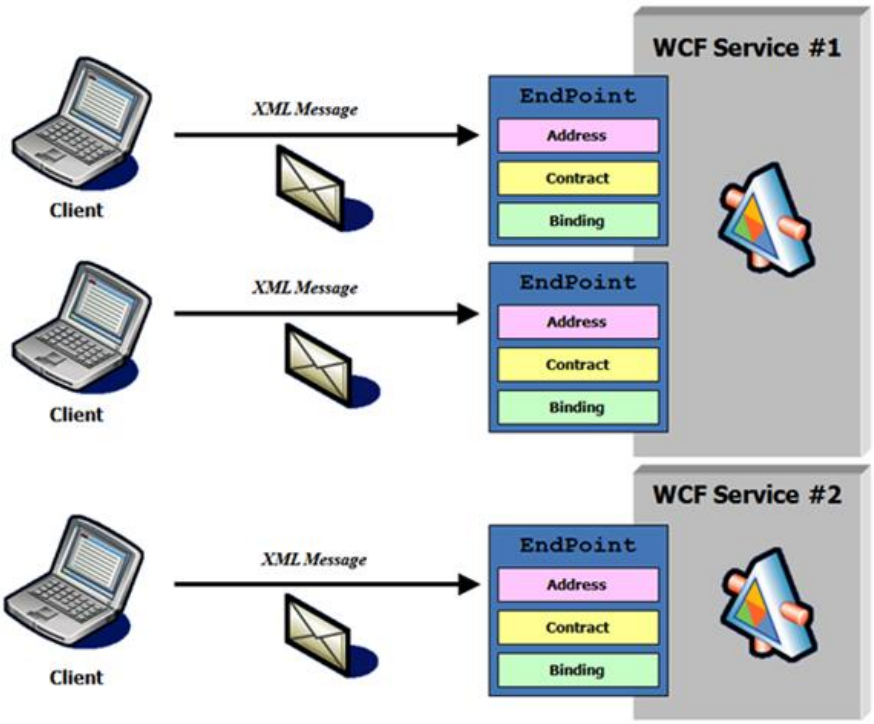
종점의 개념

WCF 서비스의 핵심적인 요소는 주소, 바인딩, 계약이다.

WCF 에서는 이들 세 가지 요소를 합쳐서 서비스 종점(endpoint)이라 부르며 WCF 서비스는 반드시 하나 이상의 종점을 갖는다.

서비스가 복수 개의 종점을 제공할 수 있다는 것은 이전의 닷넷 기반 통신 기술들인 ASP.NET 웹 서비스(ASMX), 닷넷 리모팅, COM+ 이 따라올 수 없는 WCF만의 독특하고 유연한 장점이다.

[service] 1개의 서비스가 2개의 종점을 가진다고 가정 (HTTP기반 바인딩 / TCP 기반 바인딩)
[client1] HTTP 프로토콜을 사용하는 클라이언트는 HTTP 기반의 종점을 통해 서비스 호출
[client2] TCP 프로토콜을 사용하는 클라이언트는 TCP 기반의 종점을 통해서 서비스를 호출
*) WCF 서비스는 어떤 바인딩을 사용하는가에 따라 서비스 구현코드나 방식이 달라지지 않는다.
*) 클라이언트 입장에서는 성능적으로 우수한 프로토콜을 골라서 사용할 수 있다.



[그림] 주소, 바인딩, 계약, 그리고 종점

서비스 호스트와 클라이언트

[서비스 호스트]

WCF를 이용하여 서비스를 구현하면 이 서비스는 WCF 호스트에 의해 호스팅되어야만 클라이언트의 호출을 받을 수 있다.

호스팅 환경은 일반 닷넷 EXE 어플리케이션 혹은 IIS를 이용하여 구축한다.

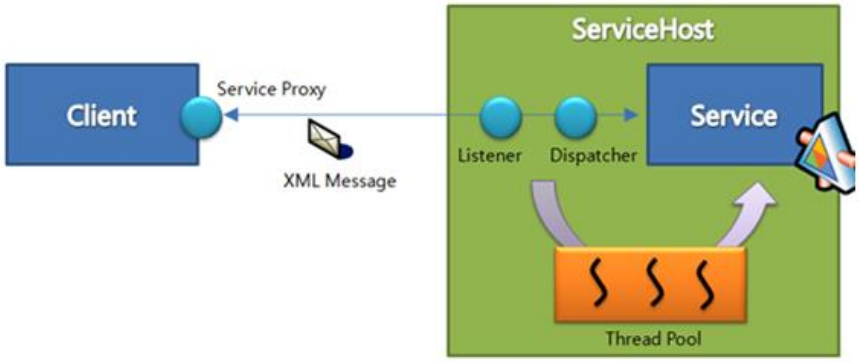
WCF가 제공하는 호스트는 다중 스레드 풀(multi-thread pool)을 완벽하게 지원한다.(클라이언트의 동시 호출(concurrent call)

에 대한 고민을 해결해 줌

[클라이언트]

서비스 호출시 WCF 프레임워크의 도움을 받는다.

- 서비스 계약으로부터 서비스가 제공하는 메소드를 알게 됨
- WCF 런타임과 유틸리티 도구(tool)를 이용하여 서비스의 계약 정보로부터 서비스에 대한 프록시 클래스를 생성한다.
- WCF 런타임이 적절하게 서비스를 찾고 XML 메시지를 전송할 수 있도록 호출하고자 하는 서비스의 종점에 대한 정보를 제공한다.



[그림] 서비스 프록시와 서비스 호스트

[서비스 프록시와 서비스 호스트 작동 방식]

1) [client] 서비스 프록시를 통해 서비스 호출
2) [proxy] WCF런타임을 통해 닷넷 메소드 호출을 XML 메시지로 변경
3) [servicehost] 서비스 호스트에 의해 호스팅된 서비스는 바인딩이 지정하는 네트워크 프로토콜의 리스너를 구동 XML 메시지를 수신
4) [listener] 수신된 XML 메시지를 디스패처에게 전달
5) [dispatcher] 메시지 내용을 해석하여 적절한 서비스의 메소드를 호출

*) 이러한 작업은 서비스 호스트, 보다 정확하게 말해 WCF 런타임이 제공하는 스레드 풀의 한 스레드를 이용하게 되므로 동시에 여러 클라이언트가 서비스를 호출하더라도 동시 액세스가 가능하다.

02. Hello World 서비스 구현

WCF 서비스 구현

Visual Studio – HelloSample '빈 솔루션' ➔ Hello World 서비스 프로그램 작성

1) HelloWorldService – 클래스 라이브러리 ➔ 위에서 작성한 서비스를 호스트하는 프로그램과 클라이언트는 모두 위 서비스의 계약, 즉 서비스의 인터페이스를 참조해야 한다. 따라서 서비스는 클라이언트와 서비스 호스트가 모두 참조할 수 있는 라이브러리 프로젝트로 작성할 것이다.

2) System.ServiceModel 어셈블리를 참조 ➔ WCF의 핵심 어셈블리

3) Hello World 서비스 구현

```
using System.ServiceModel;

namespace HelloWorldService
{
        // 1. 서비스 계약 선언
        [ServiceContract]
        public interface IHelloWorld
        {
                [OperationContract]
                string SayHello();
        }
        //2. 서비스 타입 구현
        public class HelloWorldWCFService : IHelloWorld
        {
                public string SayHello()
                {
                        return "Hello WCF World !";
                }
        }
}
```

3.1) WCF서비스 구현의 첫번째는 서비스 계약을 선언하는 것이다.

- 서비스 노출 계약인 IHelloWorld 인터페이스를 정의함
- 계약을 위한 인터페이스 선언은 반드시 System.ServiceModel 네임스페이스의 ServiceContract 특성을 인터페이스에 명시해야 한다.
- 서비스에서 사용될 매소드 역시 OperationContract 특성을 추가해야 한다.
  만약 이 특성이 명시되어 있지 않으면 해당 메서드를 계약의 일부로 간주하지 않는다.
  이러한 정책을 Opt-in 정책이라고 함

3.2) WCF서비스 구현의 두번째는 서비스 타입 구현이다.

WCF 서비스 호스트 구현

구현된 서비스를 호스팅하는 서비스 호스트 작성
서비스 호스트에 의해 호스팅될 때에만 클라이언트의 요청을 받아들일 수 있다.
서비스 호스트가 어떤 서비스를 호스팅하기 위해서는
  이 서비스가 어떤 주소를 갖는지,
  어떤 바인딩에 의해 클라이언트와 통신할 것인지
를 알아야 한다.
따라서 서비스가 호스팅되기 위해서는 서비스의 주소, 바인딩, 계약을 나타내는 서비스 종점을 정의하고 이 서비스 종점을

ServiceHost에 추가하는 과정을 거치게 된다.

1) ConsoleApplication 생성 ➔ 프로젝트명 : HelloWorldHost
2) 참조 구성 : HelloWorldService(기존 작성된 서비스 프로그램) , System.ServiceModel 추가
3) 서비스 호스트에 대한 코드 작성

(서비스 호스트는 System.ServiceModel 네임스페이스의 ServiceHost 클래스를 직접 사용하는 방식과 이 클래스에서 파생된 클래스를 사용할 수 있다.)

```
using System.ServiceModel;
using HelloWorldService;
using System.ServiceModel.Description;

namespace HelloWorldHost
{
        // Hello World 서비스에 대한 WCF 호스트 구현
        class HostApp
        {
                static void Main(string[] args)
                {
                        BasicHost();
                }
                static void BasicHost()
                {
                        // 다음 코드가 정상적으로 작동하기 위해서는 app.config 파일의
                        // <services> 요소를 comment 로 막아 두어야 한다.
                        ServiceHost host = new ServiceHost(typeof(HelloWorldWCFService),
                                new Uri("http://localhost/wcf/example/helloworldservice"));

                        host.AddServiceEndpoint(
                                typeof(IHelloWorld),            // service contract
                                new BasicHttpBinding(),         // service binding
                                "");                            // relative address
                      // ServiceMetadataBehavior 설정
                      // http://localhost/wcf/example/helloworldservice?wsdl   WSDL문서 획득

                       ServiceMetadataBehavior behavior =
                       host.Description.Behaviors.Find<ServiceMetadataBehavior>();
                      if (behavior == null)
                      {
                         behavior = new ServiceMetadataBehavior();
                         host.Description.Behaviors.Add(behavior);
                       }
                       behavior.HttpGetEnabled = true;

                          host.Open();
                          Console.WriteLine("Press Any key to stop the service");
                          Console.ReadKey(true);
                          host.Close();
                }
        }
}
```

- NetTcpBinding 은 별도로 TCP 포트를 지정해 주지 않으면 808 포트를 사용하도록 되어 있다

- ServiceHost 클래스의 인스턴스 생성(호스트하고자 하는 서비스 타입, 서비스 베이스 주소)
- 서비스 종점을 서비스 호스트에 추가(주소, 바인딩, 계약)➔ 주소는 서버베이스 주소를 그대로 사용

  BasicHttpBinding() : HTTP 프로토콜을 사용하는 바인딩으로서 기존 ASP.NET 웹 서비스와의 호환성을 위해 WCF가 기본적으로 제공하는 바인딩이다.(웹 서비스 보안을 사용할 수 없고, 트랜잭션, 콜백과 같은 WCF 제공 고급

기술을 사용할 수 없는 간단한 바인딩 임)

*) WSHttpBinding, WSDualHttpBinding : HTTP프로토콜사용, 트랜잭션 등 다양한 기능을 사용할 수 있슴
NetTcpBinding, NetNamedPipeBinding : TCP 혹은 명명된 파이프 프로토콜 기반의 고성능 바인딩
NetMsmqBinding : 비동기 통신이 가능한 바인딩
NetPeerTcpBinding : p2P 통신이 가능한 바인딩

*) 서비스 종점이 어떤 바인딩을 사용하는 가에 따라서 서비스 주소는 달라진다.
http로 시작하는 URL 서비스 주소 : BasicHttpBinding, WSHttpBinding, WSDualHttpBinding
net, tcp로 시작하는 서비스 주소 : NetTcpBinding

- 서비스 종점이 추가되었으면 서비스는 실제 클라이언트의 호출을 받을 준비가 끝난다.
Open() 메서드를 호출함으로써 서비스 호스트는 리스너를 구동하고 클라이언트 호출을 수신한다.
*) OPen메서드는 WCF 런타임이 서비스를 위해 리스터와 디스패처 등 다양한 준비를 하도록 구성한 후에 곧바로 제어를 반환한다.
*) 클라이언트의 호출을 실제로 처리하는 것은 디스패처에 의해 선택된 작업 스레드가 된다.
서비스 호스트를 설정하고 구동한 스레드는 다른 작업을 수행할 수 있다.
- 구동된 서비스는 Close() 메소드 호출에 의해 종료된다.
*) WCF 런타임은 더 이상 클라언트 호출을 수신하지 않게 되며, 이미 수신된 클라이언트 호출의 처리가 끝나길 기다린다. 클라이언트의 모든 호출이 처리되면 구성된 리스너와 디스패처를 제거하고 할당한 자원들을 모두 해제한다.

WCF 서비스 테스트

1) HelloWorldHost 애플리케이션 실행
해당 서비스의 주소는 Http://localhost/wcf/example/helloworldservice 이다.
2) 브라우저에서 해당 주소를 입력 후 실행

03. Hello World 서비스 클라이언트 구현

WCF는 XML 웹 서비스 표준으로 구현하였기 때문에 클라이언트를 작성하기 위해서 반드시 WCF를 사용해야 하는 것은 아니다.

클라이언트 콘솔 어플리케이션 구현

1) console application 생성 ➔ name : HelloWorldClient
2) WCF 런타임을 위한 System.ServiceModel 어셈블리를 참조
3) HelloWorldService : 구현해 놓은 Service 라이브러리를 참조 ➔ 서비스 계약인 인터페이스에 대한 정보를 얻기 위함
   추후 서비스 참조로 변경가능
4) 코드 작성
   클라이언트 입장에서 통신하기 위해 알아야 할 내용은 서비스 종점이다.
   (서비스 종점은 서비스 인터페이스를 기술하는 계약, 프로토콜이나 보안 등의 다양한 네트워킹 및 메시징 기술 바인딩, 서비스의 인터넷 상의 위치를 기술하는 주소를 포함한다.)

```
using System.ServiceModel;
using HelloWorldService;
using system.serviceModel.description;

namespace HelloWorldClient
{
	// Hello World 서비스 클라이언트 구현
	class ClientApp
	{
		static void Main(string[] args)
		{
			InvokeUsingHTTP();
		}

		// BasicHttpBinding을 사용하는 클라이언트 예제 코드
		static void InvokeUsingHTTP()
		{
			Uri uri = new Uri("http://localhost/wcf/example/helloworldservice");
			ServiceEndpoint ep = new ServiceEndpoint(
				ContractDescription.GetContract(typeof(IHelloWorld)),
				new BasicHttpBinding(),
				new EndpointAddress(uri));

			ChannelFactory<IHelloWorld> factory = new ChannelFactory<IHelloWorld>(ep);
			IHelloWorld proxy = factory.CreateChannel();
			string result = proxy.SayHello();
			(proxy as IDisposable).Dispose();
			Console.WriteLine(result);
		}
	}
}
```

*) system.serviceModel.description 네임스페이스의 ServiceEndPoint 클래스
서비스 종점을 기술하는 데 사용되는 클래스임(계약, 바인딩, 주소를 포함하는 모든 정보를 가지고 있다.)
생성자의 필수 매개변수는 서비스의 계약 정보를 나타내는 ContractDescription 클래스 객체이다.

*) ContractDescription 클래스는 서비스 계약이 어떤 닷넷 인터페이스 타입인가에 대한 정보와 서비스 계약에 포함될 수 있는 다양한 속성들(네임스페이스, 세션 사용 여부, 암호화 여부 등)에 대한 정보를 포함한다.

만약, 서비스 구현시 인터페이스에 네임스페이스를 다음과 같이 설정했다면 ContractDescription 객체의 Namespace 속성은 인터페이스에 설정된 네임스페이스를 반환한다.

```
// 1. 서비스 계약 선언
[ServiceContract(Namespace=http://www.simpleisbest.net/wcf/helloworld) ]
public interface IHelloWorld
{
        [OperationContract]
        string SayHello();
}

contractDescription desc = ContractDescription.Getcontract(typeof(IHelloWorld));
console.WriteLine(desc.Namespace);
//출력결과
// http://www.simpleisbest.net/wcf/helloworld
```

*) ContractDescription.GetContract(typeof(IHelloWorld)) : , IHelloWOrld 인터페이스 계약에 대한 속성 정보를 포함하는 ContractDescription 객체를 얻어낸다.

new BasicHttpBinding(), : 서비스와 동일한 바인딩을 사용한다.

new EndpointAddress(uri)); : 서비스 주로로써 가장 단순한 서비스 종점 주소를 나타내고 있다.

*) WCF의 채널 팩토리를 생성한다.

WCF의 내부 구조는 다양한 XML 웹 서비스 통신에 사용될 수 있는 다양한 채널이 메시지를 순차적으로 처리해 나가도록 구성되어 있다.

- WCF를 통해 클라이언트가 서비스를 호출하면
- 닷넷의 메소드 호출이 XML 메시지로 변환하는 작업을 수행하는 채널을 거친다.
- 필요하다면 메시지에 추가적인 보안 설정을 수행하는 채널도 통과한다.
- 최종적으로 메시지를 트랜스포트 프로토콜을 사용하여 네트워크로 흘러보내는 트랜스포트 채널을 통과하게 된다.

➔ 이러한 채널들은 서비스 계약에 주어진 보안,트랜잭션, 세션 등의 속성과 바인딩에 설정된 메시지 인코딩, 트랜잭션 처리 등에 의해 채널의 파이프라인 형태로 구성되어진다.

여러 채널들중 필요한 채널들의 파이프라인을 구성해 주는 역할을 하는 것이 채널 팩토리 객치이며 ChannelFactory<T> 클래스에 의해 구현되어 있다.

제네릭 타입인 이 클래스는 서비스의 계약 인터페이스를 타입 매개변수로 취하여 WCF의 내부 채널들을 구성하고 이 채널들을 대표하는 프록시 객체를 생성하여 반환해 주는 역할을 수행한다.

```
IHelloWorld proxy = factory.CreateChannel();    // 프록시 객체 생성 코드
```

⇨ 프록시 객체는 서비스의 계약 인터페이스 타입을 갖게 되며,

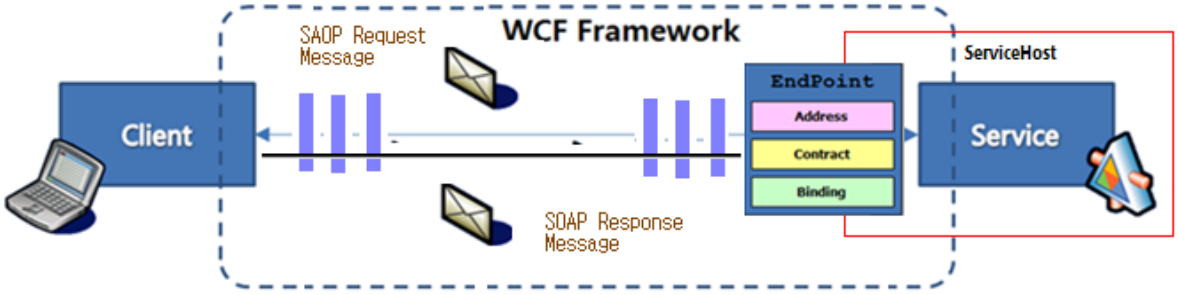
⇨ 프록시 객체는 인터페이스 메소드 호출을 XML 메시지로 변환하고 다양한 채널들을 통과하면서 이 XML매시지를 처리하고 최종적으로 이 XML, 메시지를 서비스에 전달하게 될 것이다.

```
(proxy as IDisposable).Dispose();
```

⇨ 프록시 객체는 반드시 IDispose 메소드를 호출하여 통신에 할당된 시스템 자원을 해제해 주어야 한다.

완성된 Hello World 예제

클라이언트 프로그램 수행 ....



[WCF의 역할]

*) 변환된 RequestMessage의 XML 형태

IHelloWorld 인터페이스의 SayHello 메소드를 호출하는 것은 단순히 닷넷 인터페이스의 메소드를 호출하는 것으로 보이지만 서비스의 계약 인터페이스에 대한 메소드 호출은 다음과 같은 XML 메시지로 변환된다.

```
<s:Envelope  xmlns:s="http://schemas.xmlsoap.org/soap/envelope"/>
      <s:Body>
            <SayHello  xmlns="http://www.simpleisbest.net/wcf/helloworld"/>
      </s:Body>
</s:Envelope>
```

*) Hello World 서비스가 반환한 결과 SOAP 메시지

```
<s:Envelope  xmlns:s="http://schemas.xmlsoap.org/soap/envelope"/>
      <s:Body>
            <SayHelloResponse  xmlns="http://www.simpleisbest.net/wcf/helloworld"/>
                  <SayHelloResult>Hello  WCF World!</SayHelloResult>
            </SayHelloResponse>
      </s:Body>
</s:Envelope>
```

[정리]

WCF는 SOAP 메시지 기반의 웹 서비스를 구현하기 쉽게 해 주는 프레임워크로서 닷넷 프로그래밍적인 요소를 XML 웹 서비스 요소로 바꾸어 줄 뿐만 아니라 보다 간편한 프로그래밍 모델을 제시해 준다.

WCF서비스와 WCF 클라이언트는 WCF 런타임과 클래스 라이브러리를 이용하여 서비스와 클라이언트 프록시를 구성하고 클라이언트가 프록시를 통해 서비스를 호출하면 서비스 호출은 여러 채널을 통과함에 따라 XML 기반의 SOAP 메시지로 변환되고 이 메시지가 서비스에 도착하게 된다.

서비스 측의 채널들은 수신된 SOAP 메시지를 처리하여 다시 서비스 메소드 호출로 변환해 주며
서비스 메소드 수행 후
서비스 수행 결과는 다시 채널을 통과하면서 XML 기반의 SOAP 결과 메시지로 변환된다.
클라이언트 측의 채널들은 결과 XML을 처리하여 프록시의 리턴값으로 변환하여 최종적으로 클라이언트에게 전달할 것이다.

종점의 추가

단순히 서비스 호스트에 ServiceEndPointer를 하나 더 추가하면 된다.
기존 종점과는 서비스 계약이 다르던가, 주소가 다르던가, 혹은 바인딩이 달라야 한다. 이는 서비스 호스트의 리스너가 종점을 구별하기 위한 최소한의 조건이기 때문이다.

*) 서비스 호스트에서 종점 추가 코드

```
static void Main(string[] args)
{
      BasicHost();
}
static void BasicHost()
{
      ServiceHost host = new ServiceHost(typeof(HelloWorldWCFService),
            new Uri("http://localhost/wcf/example/helloworldservice"),
          new Uri("net.tcp://localhost/wcf/example/hellowordservice"));
      host.AddServiceEndpoint(
            typeof(IHelloWorld),          // service contract
```

```
                    new BasicHttpBinding(),                 // service binding
                    "");                                                    // relative address
            host.AddServiceEndpoint(
                    typeof(IHelloWorld),            // service contract
                    new NetTcpBinding(),                    // service binding
                host.Open();
                Console.WriteLine("Press Any key to stop the service");
                Console.ReadKey(true);
                host.Close();
        }
```

*) HTTP와 TCP를 모두 사용하는 클라이언트 예제

```
        static void Main(string[] args)
        {
                InvokeUsingHTTP();
                InvokeUsingTCP();
        }
        // BasicHttpBinding을 사용하는 클라이언트 예제 코드
        static void InvokeUsingHTTP()
        {
                Uri uri = new Uri("http://localhost/wcf/example/helloworldservice");
                ServiceEndpoint ep = new ServiceEndpoint(
                        ContractDescription.GetContract(typeof(IHelloWorld)),
                        new BasicHttpBinding(),
                        new EndpointAddress(uri));

                ChannelFactory<IHelloWorld> factory = new ChannelFactory<IHelloWorld>(ep);
                IHelloWorld proxy = factory.CreateChannel();
                string result = proxy.SayHello();
                (proxy as IDisposable).Dispose();
                Console.WriteLine(result);
        }
        // NetTcpBinding을 사용하는 클라이언트 예제 코드
        static void InvokeUsingTCP()
        {
                Uri uri = new Uri("net.tcp://localhost/wcf/example/helloworldservice");
                ServiceEndpoint ep = new ServiceEndpoint(
                        ContractDescription.GetContract(typeof(IHelloWorld)),
                        new NetTcpBinding(),
                        new EndpointAddress(uri));

                ChannelFactory<IHelloWorld> factory = new ChannelFactory<IHelloWorld>(ep);
                IHelloWorld proxy = factory.CreateChannel();
                string result = proxy.SayHello();
                (proxy as IDisposable).Dispose();
                Console.WriteLine(result);
        }
```

*) 하나의 서비스가 2개 이상의 종점이 필요한 이유?

대개의 경우 클라이언트는 하나의 바인딩을 사용하여 서비스를 호출한다.

인트라넷 상의 WCF를 기반으로 하는 클라이언트는 상호 운영성이나 방화벽 통과보다는 성능적인 측면과 풍부한 기능을 갖는 프로토콜과 바인딩을 사용하는 것이 유리하다. (NetTcpBinding)

반면에 방화벽 바깥세상에서는 인트라넷 상에 존재하는 서비스를 호출하고자 한다면 방화벽 통과가 용이한 HTTP 프로토콜을 쓰는 것이 좋다.

비록 인트라넷 내에 존재하는 클라이언트일지라도 닷넷 플랫폼이 아닌 Java 클라이언트는 WCF가 사용하는 TCP 기반의

NetTcpBinding을 이해하지 못할 것이기 때문에 XML과 HTTP를 사용하는 BasicHttpBinding을 사용해야 할 것이다.

서비스 주소

서비스 주소는 바인딩이 어떤 트랜스포트를 사용하는가에 따라 달라진다.

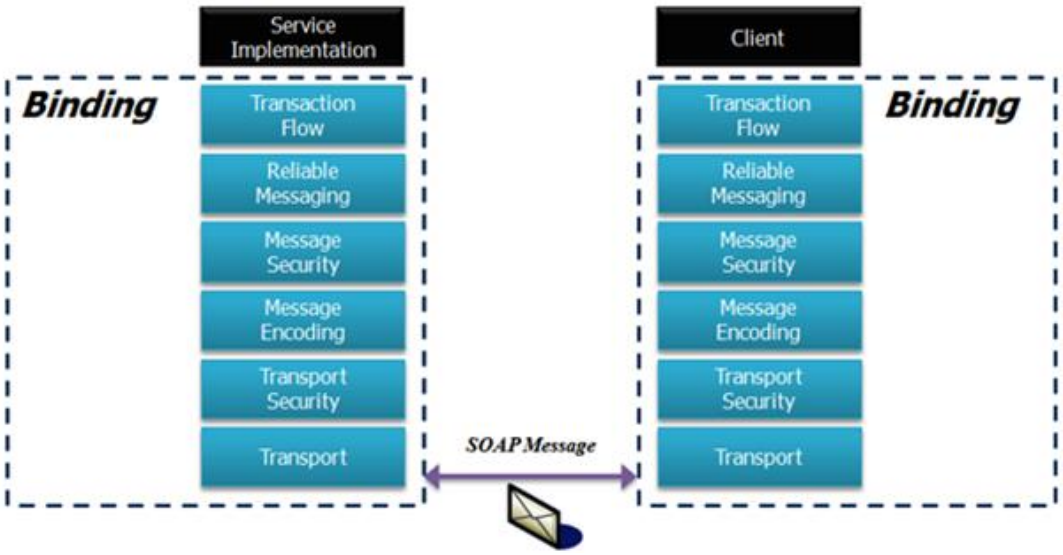
트랜스포트란 XML SOAP 메시지를 네트워크 상에서 전송해 주는 매개체 프로토콜을 말한다.

WCF에서 지원하는 트랜스포트에는 HTTP, TCP, 명명된 파이프, MSMQ, P2P 프로토콜 등이 있다.

이들 외 UDP나 SMTP를 사용하는 트랜스포트를 사용하도록 커스터마이징이 가능하지만 상당한 양의 코드를 작성해야 한다.

트랜스포트	바인딩	스킴
HTTP(S)	BasicHttpBinding, WSHttpBinding, WSDualHttpBinding, WSFederationBinding	http 혹은 https
TCP	NetTcpBinding	net.tcp
명명된파이프	NetNamePipeBinding	net.pipe
MSMQ	NetMsmqBinding, MsmqIntergrationBiding	net.msmq
P2P	NetPeerTcpBinding	net.p2p

바인딩에 대한 첨언



[그림] 바인딩 개념

WCF 에서 바인딩을 몇 마디로 설명하기란 쉽지 않은 일이다. 그래도 굳이 이야기 하자면 서비스와 클라이언트가 네트워크를 통해 통신하기 위해 필요한 트랜스포트 종류, 프로토콜 설정, 보안과 같은 특정 기능의 사용 여부 등을 명세 하는 통신 설명자(descriptor) 정도로 이야기 할 수 있겠다.

쉽게 이야기 하자면 서비스와 클라이언트가 네트워크 상에서 어떻게 통신할 것을 기술하는 것이 되겠다.

네트워크 통신 상에서 요구되는 다양한 선택 사항과 몇몇 기능들을 WCF 는 이미 프레임워크 수준에서 구현해 놓았기 때문에 개발자는 이들을 조합하여 웹 서비스 메시지 전송에 적용하면 되는 것이다. 이 때 WCF 가 제공하는 트랜스포트 프로토콜, 보안, 메시지 인코딩(message encoding), 메시지 수준의 보안, 신뢰할 수 있는 메시징(reliable messaging), 트랜잭션 흐름(transaction flow) 등의 통신 요소들을 조합해 놓은 것을 WCF 에서는 바인딩이란 용어를 사용한다고 보면 되겠다.

위의 그림에서 한 가지 놓쳐서는 안될 것은 서비스와 클라이언트가 동일한 바인딩을 사용해야만 한다는 점이다.

바인딩은 특정 기능이 그 바인딩에 포함되어 있는가에 대한 여부와 각 기능이 선택할 수 있는 옵션을 포함한다. 예를 들어, 어떤 바인딩은 분산 트랜잭션을 아예 사용하지 못하는 반면 또 어떤 바인딩은 분산 트랜잭션을 사용할 수 있지만 다양한 분산 트랜잭션 프로토콜 중 하나만을 지원하거나 두 개 이상의 분산 트랜잭션 프로토콜 중 하나를 선택할 수도 있다. 이처럼

바인딩은 여러 바인딩 요소(binding element)들 중 몇 개의 조합으로 구성되고 각 바인딩 요소에는 그 바인딩 요소에서 선택할 수 있는 선택 사항들이 또 존재한다는 것이 되겠다.

즉 BasicHttpBinding 은 HTTP 트랜스포트를 사용하는 기본적인 바인딩으로써 분산 트랜잭션이나 신뢰할 수 있는 메시징을 사용할 수 없다. 반면 NetTcpBinding 은 트랜잭션과 신뢰할 수 있는 메시징을 모두 지원한다. 그리고 BasicHttpBinding 과 NetTcpBinding 은 메시징 인코딩을 사용하지만 BasicHttpBinding 은 메시지 인코딩 옵션 중 텍스트 인코딩 혹은 MTOM(Message-Transmission Optimization Mechanism) 인코딩 중 하나를 선택할 수 있고 NetTcpBinding 은 바이너리 인코딩만을 사용하도록 구성 되어 있다.

기본적으로 WCF 에서 바인딩은 서비스에서 사용할 수 있는 다양한 기능들의 조합일 뿐이므로 이론적으로 다양한 조합이 나올 수 있으며 실제로 개발자가 자신이 필요한 기능들만을 조합하여 사용자 정의 바인딩(custom binding)을 만들 수도 있다. 하지만 상당한 노력을 요구하는 커스텀 바인딩을 매번 만들어야 한다면 대단한 낭비가 아닐 수 없다. 따라서 WCF 에서는 가장 많이 사용될 것으로 예상되는 기능들을 조합하여 대표적인 바인딩을 기본적으로 제공한다. 이들이 바로 BasicHttpBinding, WSHttpBinding, WSDualHttpBinding, WSFederationBinding, NetTcpBinding, NetNamedPipeBinding, NetMsmqBinding, MsmqIntegrationBinding, NetPeerTcpBinding 으로써 표준 바인딩 혹은 시스템 바인딩이라고도 한다.

바인딩	간략한 설명
BasicHttpBinding	WS-I 프로파일을 만족하는 웹 서비스와 호환되는 HTTP 기반의 기본 웹 서비스 가장 상호호환성이 높지만 신뢰할 수 있는 메시징, 트랜잭션 등의 기능은 사용할 수 없다. 이 바인딩은 ASP.NET 웹 서비스와 WSE(Web Service Extension)과의 호환성을 갖는다.
WSHttpBinding	다양한 웹 서비스 표준인 WS-*를 대부분 만족하는 풀 스펙의 HTTP 기반 바인딩으로써 보안, 트랜잭션, 신뢰할 수 있는 메시징을 모두 지원한다. 웹 서비스 표준들을 준수하므로 이들을 준수하는 다른 웹 서비스 혹은 웹 서비스 클라이언트들과 호환된다.
WSDualHttpBinding	WSHttpBinding에 일부 제약이 가해지지만 서비스가 클라이언트를 호출하는 콜백(callback) 기능을 갖춘 바인딩이다. 상호 운영성은 그다지 높지 않다.
WSFederationBinding	WS-Federation 표준을 구현하는 바인딩으로써 서로 다른 인증 시스템을 갖춘 서비스들이 서로를 인증하고 메시지에 대한 보안을 설정하기 위해 제공되는 바인딩이다.
NetTcpBinding	TCP 트랜스포트를 사용하는 바인딩으로써 WCF가 제공하는 모든 기능을 사용할 수 있으면서도 성능적으로 우수한 바인딩이다. 기존의 닷넷 리모팅에 비유할 수 있지만 XML 메시지를 사용한다는 점이 크게 다르다. 클라이언트와 서비스가 모두 WCF를 사용해야 하므로 상호운영성이 높지 않다.
NetNamedPipeBinding	단일 컴퓨터에서 사용하는 것을 목적으로 제공되는 바인딩으로써 가장 빠른 성능을 내는 바인딩이다.
NetMsmqBinding	트랜스포트로서 MSMQ를 사용하는 바인딩으로써 클라이언트와 서비스가 네트워크에 연결되어 있지 않더라도 SOAP 메시지가 전달될 수 있는 진정한 비동기 메시지 전달 능력을 갖춘 바인딩이다.
MsmqIntegrationBinding	레거시 MSMQ서버 및 클라이언트와 통신하기 위해 제공되는 바인딩이다.
NetPeerTcpBinding	P2P 네트워킹을 사용하는 바인딩으로써 서비스와 클라이언트가 P2P 통신이 가능하게 해준다.

[표] WCF 표준 바인딩들

*) 기존 sample에서 binding 사용 고찰

그렇다면 지금까지 우리가 예제 코드에서 사용한 바인딩은 어떻게 된 것일까? 우리가 지금까지 보아온 예제코드는 달랑 바인딩 객체들(BasicHttpBinding과 NetTcpBinding)를 생성하는 것으로 끝나지 않았는가? 그렇다. 바인딩은 기본 설정 값들을 가지고 있기 때문에 별도의 설정을 수행하지 않아도 기본 설정을 사용하여 작동하도록 되어 있다. 예를 들어 BasicHttpBinding 은 메시지 수준 보안은 "사용하지 않음"이 기본값이며 메시지 인코딩은 텍스트 인코딩이 기본값이다. 또한 트랜스포트 수준의 보안 역시 "사용하지 않음"이 기본 설정이다. 반면 NetTcpBinding은 메시지 수준 보안이 "사용하지 않음"이 기본값이며 메시지 인코딩은 바이너리 인코딩이, 트랜스포트 수준의 보안은 Windows 보안을 사용하는 것이 기본값이다. 따라서 지금까지 우리는 바인딩의 기본 설정만을 이용한 것이 되겠다.

다음 코드 조각은 서비스 호스트에 서비스 종점을 명시할 때 BasicHttpBinding의 메시지 인코딩을 MIME 기반의 MTOM 인코딩을 사용할 것을 설정하고 있다 . BasicHttpBinding 객체를 먼저 생성한 후에 MessageEncoding 속성을 설정하는 것에 주목하기 바란다.

```
ServiceHost host = new ServiceHost(typeof(HelloWorldWCFService));
BasicHttpBinding binding = new BasicHttpBinding();
```

```
binding.MessageEncoding = WSMessageEncoding.Mtom;
host.AddServiceEndpoint(
    typeof(IHelloWorld), // service contract
    binding, // service binding
    "http://localhost/wcf/example/helloworldservice");
...
```

이 서비스를 호출하는 클라이언트 역시 동일하게 BasicHttpBinding을 사용해야 함은 물론이요, 메시지 인코딩 옵션을 MTOM으로 해주지 않으면 안 된다. 다음 코드 조각은 클라이언트 측의 바인딩 설정을 보여 주고 있다. 서비스 호스트와 비슷하게 바인딩 객체를 생성한 후에 필요한 설정을 수행하고 있음이 지금까지 우리가 보아온 예제 코드와 다른 점임을 살펴보도록하자.

```
BasicHttpBinding binding = new BasicHttpBinding();
binding.MessageEncoding = WSMessageEncoding.Mtom;

Uri uri = new Uri("http://localhost/wcf/example/helloworldservice");
ServiceEndpoint ep = new ServiceEndpoint(
    ContractDescription.GetContract(typeof(IHelloWorld)),
    binding,
    new EndpointAddress(uri));
...
```

04. 요약

WCF의 기본 개념들 이해

- 서비스 종점
- 주소
- 바인딩 계약
- 서비스 호스트