00. WCF 개념

#WCF #SOAP

Quote >

URL : https://learn.microsoft.com/ko-kr/dotnet/framework/wcf/whats-wcf
URL : https://gostart.tistory.com/117
URL : https://blog.naver.com/islove8587/220970196394
URL : https://hongchange.tistory.com/entry/WCF%EB%9E%80-%EB%AC%B4%EC%97%87%EC%9D%B8%EA%B0%80
URL : https://vsts2010.tistory.com/179

WCF 개념

01. 정의

> *Windows Communication Foundation*

- Service Oriented Application 개발을 위한 Framework.
- WCF를 사용하면 데이터를 비동기 메시지로 서비스 엔드포인트 간에 전송 가능
  - 서비스 엔드포인트는 IIS에서 호스팅하는 계속 사용 가능한 서비스의 일부분일 수도 있고 애플리케이션에서 호스팅되는 서비스일 수도 있습니다.
  - 엔드포인트는 서비스 엔드포인트에서 데이터를 요청하는 서비스의 클라이언트일 수 있습니다
- 메시지는 XML로 전송되는 한 문자나 단어처럼 간단할 수도 있고 이진 데이터 스트림처럼 복잡할 수도 있습니다.

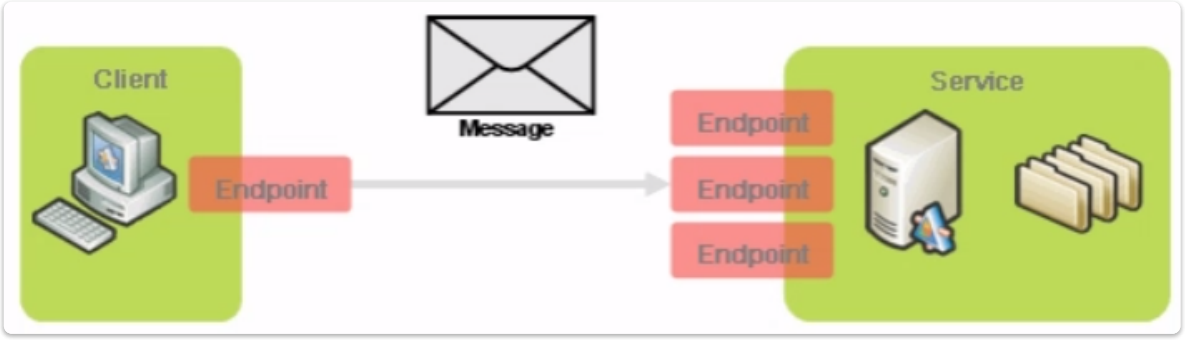
02. 기능

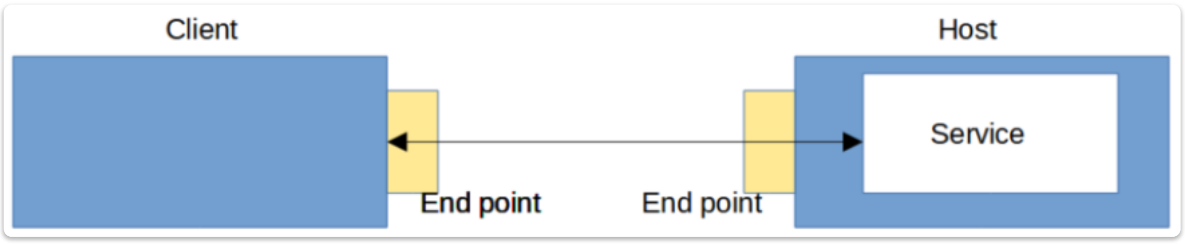
- 서비스 지향성
  - WS 표준을 사용하는 경우의 이점 중 하나는 WCF를 통해 서비스 지향 애플리케이션을 만들 수 있다는 것
    - SOA(서비스 지향 아키텍처) 방식에서는 웹 서비스를 사용하여 데이터를 보내고 받습니다
    - 애플리케이션이 서로 하드 코드되지 않고 느슨하게 결합된다

03. 개념

- 메시지(Message)
  - WCF를 통해서 전송되는 데이터 블럭으로 SOAP XML로 되어 있다.
- 엔드포인트(EndPoint)

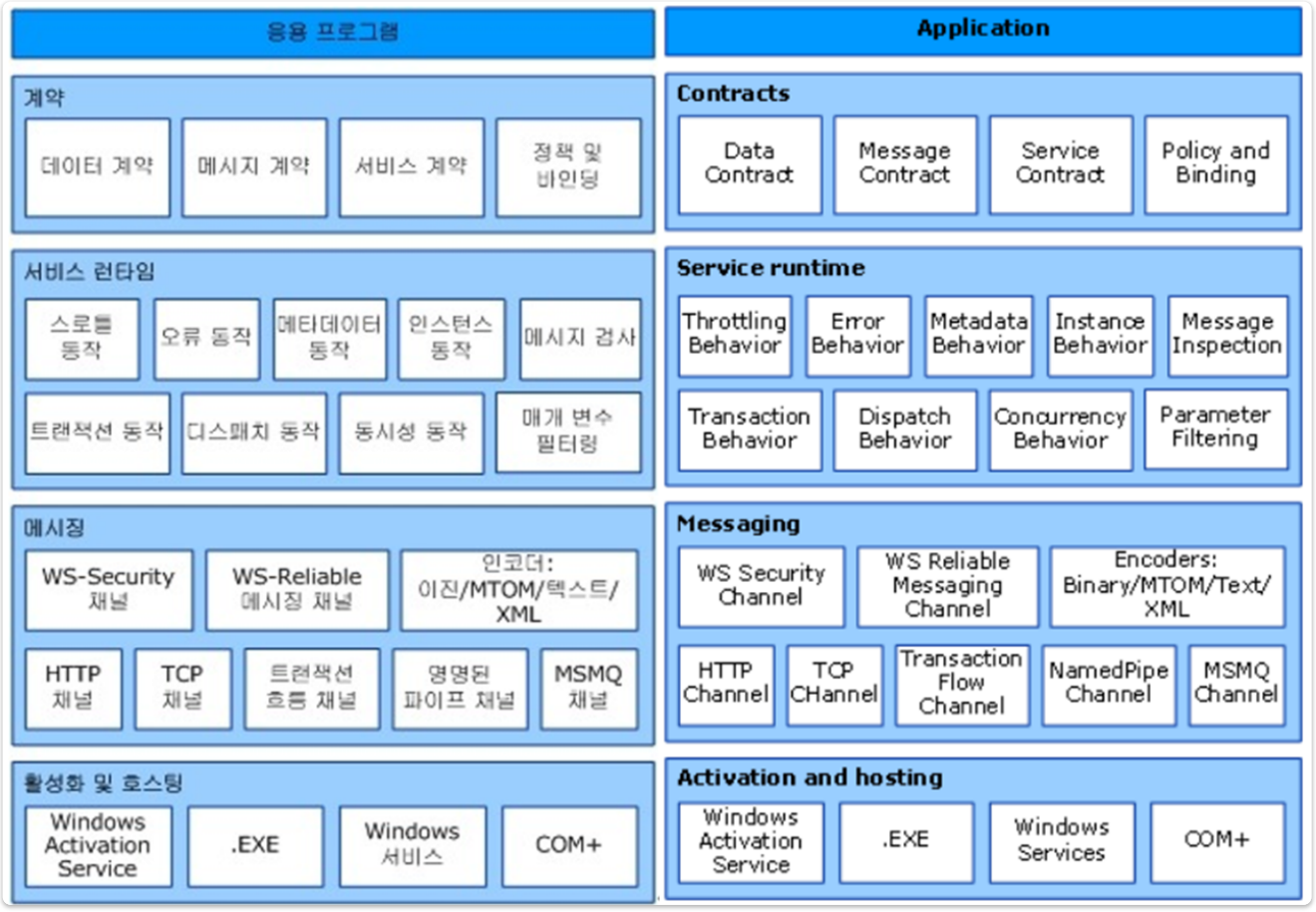
    - 메시지를 송,수신하는데 사용하는 주소, 메시지 세트 정의, 메시지 전송 방법 등을 정의한다.
    - WCF는 엔드포인트를 통해 서버(서비스)와 클라이언트 간의 메시지 교환을 한다.
    - Endpoint는 .Net 리모팅 채널과 포멧터를 합친 개념이라고 볼 수 있는데, 서버(서비스)에는 여러 개의 Endpoint를 구성할 수 있으며 클라이언트는 이중 하나 이상의 Endpoint와 메시지 교환을 하게 된다.





- 주소(Address)
    - 메시지를 수신할 정확한 위치를 URL형태로 진정한다.
- 바인딩(Binding)
    - 엔드포인터가 사용하게 될 프로토클, 메시지 인코딩 형식, 보안과 관련된 통신방식을 정의한다.
- 계약(Contracts)
    - 엔드포인터가 클라이언트에 제공하는 기능을 정의하며 인터페이스 이름으로 구성된다.
- 호스팅(Hosting)
    - 서비스를 운영하는 방식으로 자체호스팅, IIS 호스팅, WAS 호스팅등의 방법이 있다.
- 메타데이터(Metadata)
    - 서비스에서 서비스와 통신하기 위해 외부 엔터티가 이해해야 하는 서비스 특성을 설명
- 클라이언트 애플리케이션 (Client application)
    - 메시지를 하나 이상의 엔드포인트와 교환하는 프로그램
    - WCF 클라이언트의 인스턴스를 만들고 WCF 클라이언트의 메서드를 호출하여 시작합니다.
- WCF 클라이언트
    - 클라이언트 애플리케이션 구문으로 서비스 작업을 메서드로 노출합니다.
    - 모든 애플리케이션은 서비스를 호스트하는 애플리케이션을 포함하여 WCF 클라이언트를 호스트할 수 있습니다.
    - 다른 서비스의 WCF 클라이언트를 포함하는 서비스를 만들 수 있습니다.
- SOAP(Simple Object Access Protocol)
    - Header, Body로 구성된 xml 문서다.

04. 아키텍처



1) 계약(Contracts) 계층

계약(Contracts)계층은 서비스에서 전달할 정보에 대한 정의와 어떤 서비스를 제공할지를 선언한다.

- Data Contracts: 서비스 제공자가 송, 수신할 데이터 타입과 변수를 선언한다.
- Message Contract: 서비스의 송,수신되는 SOAP 메시지를 통제한다.
- Service Contract: 서비스의 엔드포인터(Endpoint)가 무엇을 제공하고 통신하는지를 선언한다.
- Policy and Binding: 프로토콜과 같이 통신에 사용할 정책들에 대하여 선언한다.

2) 서비스 실행(Service Runtime) 계층

서비스 런타임 계층은 서비스 운영 또는 서비스 런타임 중에 발생하는 서비스의 동작을 선언하고 관리한다.

- Throttling Behavior: 동시에 처리할 수 있는 메시지의 수를 정의한다.
- Error Behavior: 서비스 런타임 중에 오류가 발생할 경우 수행할 작업을 정의한다.
- Metadata Behavior: 메타 데이터가 외부에 노출되는지 여부를 정의한다.
- Instance Behavior: 메시지를 처리할 수 있는 인스턴스의 수를 정의한다.
- Message Inspection: 서비스 제공자가 메시지의 전체 또는 일부를 검사할 수 있도록 한다.
- Transaction Behavior: 서비스가 런타임중에 실패할 경우 드랜잭션을 롤백할 수 있는 기능을 제공한다.
- Dispatch Behavior: 메시지를 처리하고 처리하는 방법을 결정한다.
- Concurrency Behavior: 서비스또는 서비스의 인스턴스가 병렬로 처리하는 기능을 제공한다.
- Parameter Filtering: 메시지의 해더에 따라 사전 설정작업을 실행한다.

3) 메시지(Messaging) 계층

메시징 계층은 서비스 통신중에 사용할 수 있는 형식과 데이터 교환 패턴을 정의한다.

- WS Security Channel: WS 보안 형태의 메시지 전달을 가능하게 한다.
- WS Reliable Message Channel: WS 신뢰성 메시지 규격의 메시지 전달을 가능하게 한다.
- Encoders: Binary/MTOM/Text/XML: 메시지를 어떤 형태로 해석하는지를 정의한다.
- HTTP Channel: HTTP 프로토클을 통해서 메시지가 전달되도록 정의한다.
- TCP Channel: TCP 프로토클을 통해서 메시지가 전달되도록 정의한다.
- Transaction Flow Channel: 전송된 메시지 패턴을 관리한다.
- NamedPipe Channel: 프로세스간 통신을 가능하게 한다.
- MSMQ Channel: MSMQ를 가능하게 한다.

4) 활성화 및 호스팅(Activation and hosting) 계층

서비스 활성화를 어떤 형태로 할지를 결정한다.
- IIS;Internet Information Service: IIS를 사용하여 COM+를 등록하고 HTTP 프로토클을 사용한다.
- WAS;Windows Activation Service: Windows WAS이며IIS와 같이 제공되며 HTTP, HTTP 기반 통신이 가능하다.
- .EXE: 실행파일을 실행하여 서비스로 프로토클 선택과 자체 주소 체계를 설정하는데 있어 유연성을 제공한다.
- Windows Service: Windows Service에 등록하여 호스팅을 제공한다.

05. ABC 컴포넌트 모델

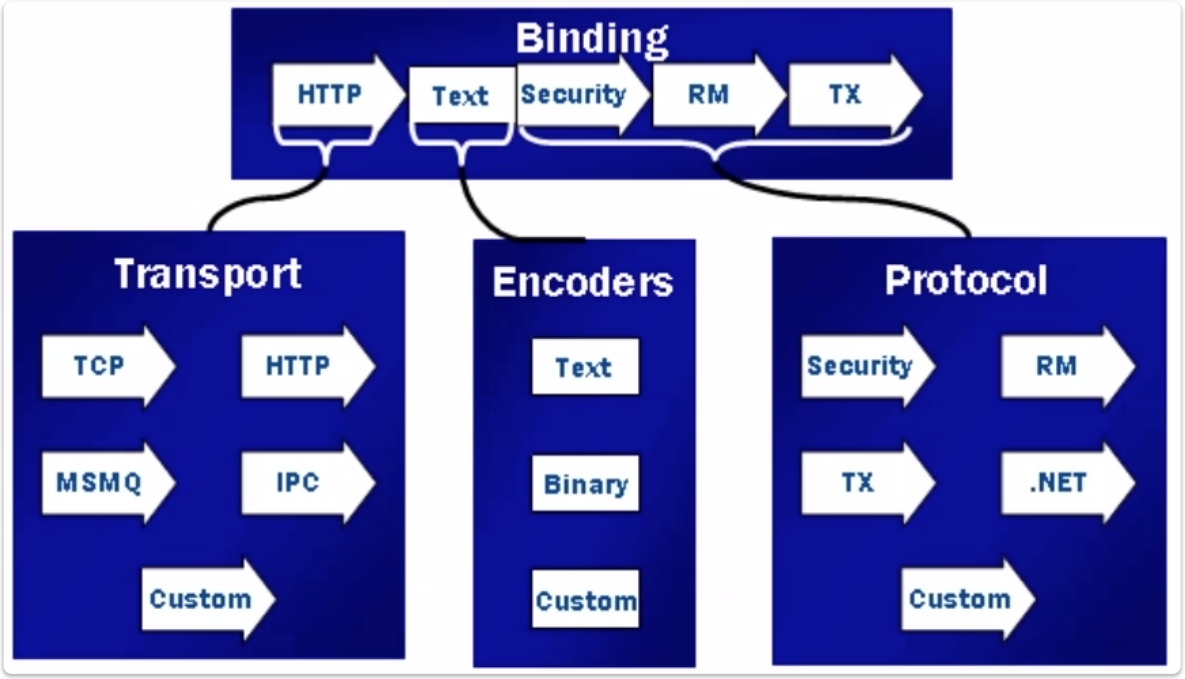
05.01. WCF

- WCF의 Endpoint는 서버와 클라이언트 간의 통신을 위한 포털 역할을 한다고 볼 수 있는데,
  이를 자세히 들여다 보면 통신을 위해 가장 기본적으로 아래와 같은 구성 요소로 이루어진다.
  - Where : 서비스의 위치
  - How : 통신 방식
  - What : 서비스
    즉, Endpoint는 아래와 같은 구성 요소로 이루어진다.

05.02. EndPoint 구성 요소

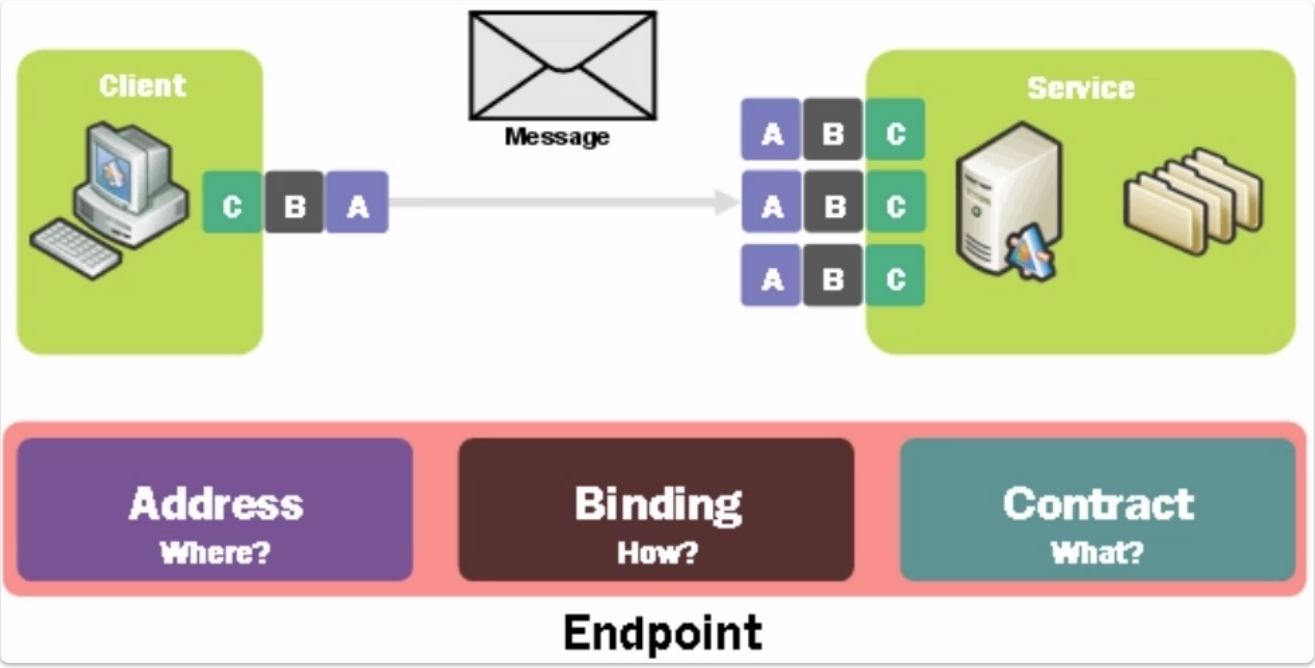
- Address
  - 서비스에 접근하기 위한 주소
- Bidning
  - 서비스 호출에 사용되는 Transport Protocol, 인증, 암호화, 메시지 인코딩, 세션 여부 등

- Binding에 의해 Address 형식이 결정되곤 함



- Contract
  - 서비스에 대한 인터페이스
  - 서비스 메소드 : Operation Contract
  - 관련 데이터 타입 : Data Contract

05.03 ABC Component Model



구성 요소	키워드	설명
Address	Where?	서비스가 어디에 위치해 있는가? 서비스 주소 및 연결 포트 정보 등
Binding	How?	서비스는 어떤 통신을 하는가? 통신 프로토콜, 인코딩 방법, 보안 요구사항 등
Contract	What?	서비스는 무엇을 제공하는가? 서비스가 제공하는 대상 등

기본 WCF 프로그래밍

01. Service Design & Contract

01.01. 계약 정의 및 MEP(Message Exchang Pattern)

01.01.01 요청/회신 Pattern (Default)

- Client에서 서비스 기능 요청 후 관련된 회신을 받는 패턴

```
// 기본 서비스 계약
[ServiceContract]
public interface IService
{
        // 서비스가 제공하는 기능 계약
        [OperationContract]
        Result Add1(double n1, double n2);
```

```
        [OperationContract]
        Result Add2(double n1, double n2);

        [OperationContract]
        void Add3(ref double n1, ref double n2);

        [OperationContract]
        void Add4(ref double n1, ref double n2, out Result res);

        [OperationContract]
        void Add5(double n1, double n2);
}

// 기본 데이터 계약
[DataContract] // 형식에 데이터 계약이 있음을 선언
public class Result
{
        private string _command = "";
        private double _resultVal = 0.0;

        [DataMember] // serialize되는 멤버(속성, 필드 또는 이벤트)를 정의
        public string Command
        {
                get { return _command; }
                set { _command = value; }
        }

        [DataMember]
        public double ResultVal
        {
                get { return _resultVal; }
                set { _resultVal = value; }
        }
}
```

01.01.02. 단방향 Pattern

- 작업이 완료될 때까지 기다리지 않아도 되고 SOAP 오류가 처리되지 않을 경우 작업에 단방향 메시지 패턴을 지정할 수 있습니다.
- 단방향 작업은 클라이언트가 작업을 호출하고 WCF에서 네트워크에 메시지를 작성한 후에 처리를 계속하는 작업입니다.

```
// 기본 서비스 계약
[ServiceContract]
public interface IService
```

```
{
        // 서비스가 제공하는 기능 계약
        // 단반향 설정
        [OperationContract(IsOneWay=true)]
        void Add5(double n1, double n2);
}

// 기본 데이터 계약
[DataContract] // 형식에 데이터 계약이 있음을 선언
public class Result
{
        private string _command = "";
        private double _resultVal = 0.0;

        [DataMember] // serialize되는 멤버(속성, 필드 또는 이벤트)를 정의
        public string Command
        {
                get { return _command; }
                set { _command = value; }
        }

        [DataMember]
        public double ResultVal
        {
                get { return _resultVal; }
                set { _resultVal = value; }
        }
}
```

01.01.03. 이중 Pattern

- 특징은 단방향 메시징을 사용하든 요청/회신 메시징을 사용하든 관계없이 서비스와 클라이언트 모두 서로 독립적으로 메시지를 보낼 수 있는 기능
- 클라이언트와 직접 통신해야 하는 서비스에 유용하며, 메시지를 교환하는 양측에 이벤트와 유사한 동작을 비롯하여 비동기 환경을 제공하는 데도 유용합니다.

```
// 기본 서비스 계약
[ServiceContract(CallbackContract=typeof(IServiceReply))]
public interface IService
{
        // 서비스가 제공하는 기능 계약
        // 단반향 설정
        [OperationContract(IsOneWay=true)]
        void Add1(double n1, double n2);
}
```

```
// 서비스가 클라이언트에서 호출할 수 있는 작업 집합을 정의하는 콜백 인터페이스를 만듭니다.
public interface IServiceReply
{
        [OperationContract(IsOneWay = true)]
        void Reply(Result res);
}

// 기본 데이터 계약
[DataContract] // 형식에 데이터 계약이 있음을 선언
public class Result
{
        private string _command = "";
        private double _resultVal = 0.0;

        [DataMember] // serialize되는 멤버(속성, 필드 또는 이벤트)를 정의
        public string Command
        {
                get { return _command; }
                set { _command = value; }
        }

        [DataMember]
        public double ResultVal
        {
                get { return _resultVal; }
                set { _resultVal = value; }
        }
}
```

01.02. 계약에 메시지 보호 수준 지정

```
// 기본 서비스 계약
[ServiceContract]
public interface IService
{
        // WSHttpBinding : 메시지가 암호화되고 디지털 방식 서명된 메시지 전송
        [OperationContract]
        Result Add1(double n1, double n2);

        // WSHttpBinding : 메시지는 암호화되지 않고 서명되지 않은 일반 텍스트로 전송
        [OperationContract(ProtectionLevel=ProtectionLevel.None)]
        Result Add2(double n1, double n2);

        // WSHttpBinding : 메시지는 암호화되지 않고 서명된 메시지 전송
        [OperationContract(ProtectionLevel=ProtectionLevel.Sign)]
        void Add3(ref double n1, ref double n2);
```

```
        // WSHttpBinding : 암호화되고 서명된 메시지 전송
        // 메시지 부분이 서명되기 전에 암호화되어 기밀성을 보장합니다.
        [OperationContract(ProtectionLevel=ProtectionLevel.EncryptAndSign)]
        void Add4(ref double n1, ref double n2, out Result res);
}
```

01.03. Session

01.03.01. Session

- Allowed : Specifies that the contract supports sessions if the incoming binding supports them.
  - 들어오는 바인딩이 세션을 지원하는 경우 계약이 세션을 지원하도록 지정합니다.
- NotAllowed : Specifies that the contract never supports bindings that initiate sessions.
  - 계약이 세션을 시작하는 바인딩을 지원하지 않음을 지정합니다.
- Required : Specifies that the contract requires a sessionful binding. An exception is thrown if the binding is not configured to support session.
  - 계약에 세션 바인딩이 필요함을 지정합니다. 세션을 지원하도록 바인딩이 구성되지 않은 경우 예외가 발생합니다.

01.03.02. Session Option

- IsInitiating : 메서드가 서버에서 세션을 시작할 수 있는 작업을 구현하는지 여부를 나타내는 값을 가져오거나 설정합니다(해당 세션이 있는 경우).
  - 작업이 서버에서 세션을 시작하도록 허용된 경우 `true`이고, 그렇지 않은 경우 `false`.
  - 기본값은 입니다 `true`.
- IsTerminating : 회신 메시지(있는 경우)를 보낸 후 서비스 작업의 결과로 서버에서 세션을 종료할지 여부를 나타내는 값을 가져오거나 설정합니다.
  - 작업의 결과로 서버에서 세션을 종료하면 `true`이고, 그렇지 않으면 `false`입니다.
  - 기본값은 `false`입니다.

```
[ServiceContract(SessionMode = SessionMode.Required)]
public interface ICalculator
{
        // session 시작
        [OperationContract(IsOneWay=true, IsInitiating=true, IsTerminating=false)]
        void Clear();

        [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]
        void FirstValue(double fir);
```

```
        [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]
        void Add();

        [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]
        void Subtract();

        [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]
        void Multiply();

        [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]
        void Divide();

        [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]
        void SecondValue(double sec);

        // session 종료
        [OperationContract(IsInitiating = false, IsTerminating = true)]
        double Calculate();
}

public interface IServiceReply
{
        [OperationContract(IsOneWay = true)]
        void Reply(double result);
}
```

02. Binding

> *바인딩은 WCF(Windows Communication Foundation) 서비스의 엔드포인트에 연결하기 위해 필요한 통신 세부 사항을 지정하는 데 사용되는 개체입니다.*

02.01. 바인딩이 정의하는 내용

- **프로토콜**
  - 신뢰할 수 있는 메시징 기능 또는 트랜잭션 컨텍스트 흐름 설정 중 하나인 사용되는 보안 메커니즘을 결정합니다.
- **인코딩**
  - 메시지 인코딩(예: 텍스트 또는 이진)을 결정합니다.
- **전송**
  - 사용할 내부 전송 프로토콜(예: TCP 또는 HTTP)을 결정합니다.

02.02. 시스템 제공 바인딩

- BasicHttpBinding: ASP.NET 웹 서비스 기반 서비스 등 웹 서비스에 연결하는 데 적합한, WS-I Basic Profile 사양을 준수하는 HTTP 프로토콜 바인딩입니다.

- WSHttpBinding: 엔드포인트에 연결하는 데 적합한, WS-* 프로토콜을 준수하는 상호 운용 가능한 바인딩입니다.
- NetNamedPipeBinding: .NET Framework를 사용하여 동일한 컴퓨터의 다른 WCF 엔드포인트에 연결합니다.
- NetMsmqBinding: .NET Framework를 사용하여 다른 WCF 엔드포인트와 큐에 대기된 메시지 연결을 만듭니다.
- NetTcpBinding: 이 바인딩은 HTTP 바인딩보다 뛰어난 성능을 제공하며 로컬 네트워크에서 사용하기에 적합합니다.

03. Endpoint

- WCF(Windows Communication Foundation) 서비스와의 모든 통신은 서비스의 *엔드포인트*를 통해 이루어집니다.
- 엔드포인트는 WCF 서비스가 제공하는 기능에 대한 클라이언트 액세스를 제공합니다.

03.01. 구조

> *각 엔드포인트에는 엔드포인트를 찾을 위치를 나타내는 주소, 클라이언트가 엔드포인트와 통신할 수 있는 방법을 지정하는 바인딩, 그리고 사용 가능한 메서드를 식별하는 계약이 포함되어 있습니다.*

- A, 주소 (Address)
  - 주소는 엔드포인트를 고유하게 식별하고 잠재 고객에게 서비스가 있는 위치를 알려줍니다.
- B, 바인딩 (Binding)
  - 바인딩은 엔드포인트와 통신하는 방법을 지정합니다.
  - 바인딩은 사용할 전송 프로토콜(예: TCP 또는 HTTP), 메시지에 사용할 인코딩(예: 텍스트 또는 이진), 필요한 보안 요구 사항(예: SSL[Secure Sockets Layer] 또는 SOAP 메시지 보안) 등을 포함하여 엔드포인트가 대상과 통신하는 방법을 지정
- C, 서비스 계약 (Contract)
  - 서비스 계약에서는 엔드포인트가 클라이언트에 노출하는 기능을 간략하게 설명합니다.
  - 계약은 클라이언트가 호출할 수 있는 작업, 작업을 호출하는 데 필요한 입력 매개변수 또는 데이터의 형식 및 메시지 형식, 클라이언트가 기대할 수 있는 처리 또는 응답 메시지의 종류 등을 지정합니다.

03.01.01 코드로 Endpoint 정의

```
//Specify the base address for Echo service.
Uri echoUri = new Uri("http://localhost:8000/");

//Create a ServiceHost for the Echo service. ServiceHost
serviceHost = new ServiceHost(typeof(Echo),echoUri);

// Use a predefined WSHttpBinding to configure the service.
WSHttpBinding binding = new WSHttpBinding();
```

```
// Add the endpoint for this service to the service host.
serviceHost.AddServiceEndpoint( typeof(IEcho), binding, echoUri );

// Open the service host to run it.
serviceHost.Open();
```

03.01.02 구성으로 Endpoint 정의

```
<configuration>
  <system.serviceModel>
    <services>
      <service name="UE.Samples.HelloService"
               behaviorConfiguration="HelloServiceBehavior">
        <endpoint address="/Address1"
                  binding="basicHttpBinding"
                  contract="UE.Samples.IHello"/>

        <endpoint address="mex"
                  binding="mexHttpBinding"
                  contract="IMetadataExchange" />
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="HelloServiceBehavior">
          <serviceMetadata httpGetEnabled="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

03.02. Metadata

> *- 서비스는 하나 이상의 메타데이터 엔드포인트를 게시하여 메타데이터를 게시합니다.*
> *- 서비스 메타데이터를 게시하면 WS-MetadataExchange(MEX) 및 HTTP/GET 요청과 같이 표준화된 프로토콜을 통해 메타데이터를 사용할 수 있습니다.*
> *- 메타데이터 엔드포인트는 주소, 바인딩 및 계약에 포함된 다른 서비스 엔드포인트와 유사하며 구성 또는 코드를 통해 서비스 호스트에 추가할 수 있습니다.*

Service

```
[ServiceContract]
public interface ISimpleService
{
```

```
        [OperationContract]
        string SimpleMethod(string msg);
}
class SimpleService : ISimpleService
{
        public string SimpleMethod(string msg)
        {
                Console.WriteLine("The caller passed in " + msg);
                return "Hello " + msg;
        }
}
```

03.02.01 코드로 Metadata 게시

- 클라이언트가 WS-MetadataExchange를 사용하는 메타데이터 또는 `?wsdl` 쿼리 문자열을 사용하는 HTTP/GET 요청을 검색할 수 있도록 메타데이터가 코드에서 WCF 서비스를 게시할 수 있는 방법을 보여 줍니다.

> ⚠ **Service Library를 참조한 후 해당 Host Console 프로그램 실행 시 >**
>
> Service Library 또한 IIS로 실행되기 때문에, 동일한 주소로 바인딩 시 이미 바인딩 되었다고 오류가 발생!
> 다른 주소로 재 맵핑해서 사용하시거나, 서비스 계약 및 구현을 콘솔로 옮겨서 Only Console만 실행

```
try
{
        // 콘솔 애플리케이션의 main 메서드 내에서 서비스 형식 및 기본 주소를 전달하여
        // ServiceHost개체를 인스턴스화합니다.
        ServiceHost svcHost = new ServiceHost(typeof(SimpleService), new
Uri("http://localhost:8001/MetadataSample"));

        // 서비스 호스트에 ServiceMetadataBehavior가 포함되어 있는지 여부를 확인합니다.
        // 포함되지 않은 경우에는 새 ServiceMetadataBehavior 인스턴스를 만듭니다.
        smb = svcHost.Description.Behaviors.Find<ServiceMetadataBehavior>();
        // If not, add one
        if (smb == null)
                smb = new ServiceMetadataBehavior();

        // HTTP/GET 요청을 사용하여 검색용 서비스 메타데이터를 게시할지 여부를
        // 타내는 값을 가져오거나 설정합니다.
        smb.HttpGetEnabled = true;

        /*
        ServiceMetadataBehavior에 MetadataExporter 속성이 포함됩니다.
        MetadataExporter에 PolicyVersion 속성이 포함됩니다.
        PolicyVersion 속성 값을 Policy15로 설정합니다.
        PolicyVersion 속성을 Policy12로 설정할 수도 있습니다.
        Policy15로 설정된 경우,
```

```
        메타데이터 내보내기는 WS-Policy 1.5를 준수하는 메타데이터로 정책 정보를 생성합니다.
        Policy12로 설정된 경우, 메타데이터 내보내기는 WS-Policy 1.2를 준수하는 정책 정보를 생성
합니다.
        */
        smb.MetadataExporter.PolicyVersion = PolicyVersion.Policy15;

        //서비스 호스트의 동작 컬렉션에 ServiceMetadataBehavior 인스턴스를 추가합니다.
        svcHost.Description.Behaviors.Add(smb);

        //서비스 호스트에 메타데이터 교환 엔드포인트를 추가합니다.
        svcHost.AddServiceEndpoint(ServiceMetadataBehavior.MexContractName,
MetadataExchangeBindings.CreateMexHttpBinding(),"mex");

        // 서비스 호스트에 애플리케이션 엔드포인트를 추가합니다.
        svcHost.AddServiceEndpoint(typeof(ISimpleService), new WSHttpBinding(), "");

        /*
        서비스 호스트를 열고 들어오는 호출을 기다립니다.
        사용자가 Enter 키를 누르면 서비스 호스트가 닫힙니다.
        */
        // Open the service host to accept incoming calls
        svcHost.Open();

        // The service can now be accessed.
        Console.WriteLine("The service is ready.");
        Console.WriteLine("Press <ENTER> to terminate service.");
        Console.WriteLine();
        Console.ReadLine();

        // Close the ServiceHostBase to shutdown the service.
        svcHost.Close();
}
catch (CommunicationException commProblem)
{
        Console.WriteLine("There was a communication problem. " + commProblem.Message);
        Console.Read();
}
```

03.02.02 구성로 Metadata 게시

- WCF 서비스가 메타데이터를 게시하여 클라이언트가 WS-MetadataExchange를 사용하는 메타데이터 또는 `?wsdl` 쿼리 문자열을 사용하는 HTTP/GET 요청을 검색할 수 있도록 구성하는 방법을 보여 줍니다.

```
<configuration>
  <system.serviceModel>
    <services>
      <service name="Metadata.Example.SimpleService"
               behaviorConfiguration="SimpleServiceBehavior">
        <endpoint address=""
                                binding="wsHttpBinding"
```

```
                                contract="Metadata.Example.ISimpleService" />

                <endpoint address="mex"
                                binding="mexHttpBinding"
                                contract="IMetadataExchange" />
      </service>
    </services>
    <behaviors>
          <serviceBehaviors>
            <behavior name="SimpleServiceBehavior">
              <serviceMetadata httpGetEnabled="True" policyVersion="Policy15" />
            </behavior>
          </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

04. Secure

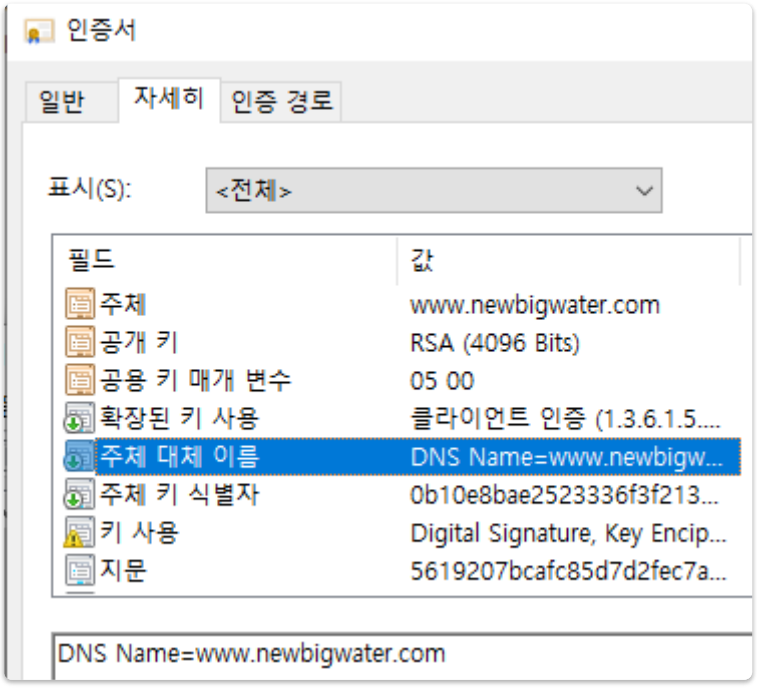
04.01. Username

- Service : WindowsBasedValidator

```
namespace GettingStartedLib
{
    public class WindowsBasedValidator : UserNamePasswordValidator
    {
        public override void Validate(string userName, string password)
        {
            if (userName == "test" && password == "test")
            {
                return;
            }

            throw new FaultException("Test account can be authenticated ONLY.");
        }
    }
}
```

- Service : App.config




```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

	<appSettings>
		<add key="aspnet:UseTaskFriendlySynchronizationContext" value="true" />
	</appSettings>
	<system.web>
		<compilation debug="true" />
	</system.web>
	<!— 서비스 라이브러리 프로젝트를 배포할 때 호스트의 app.config 파일에 구성 파일의 내용
을 추가해야 합니다.
  System.Configuration이 라이브러리에 대한 구성 파일을 지원하지 않습니다. -->
	<system.serviceModel>
		<diagnostics performanceCounters="Default" />
		<bindings>
			<wsHttpBinding>
				<binding name="websocketHttpBinding">
					<security mode="Message">
						<message clientCredentialType="UserName"/>
					</security>
				</binding>
			</wsHttpBinding>
		</bindings>
		<services>
			<service behaviorConfiguration="UserCertificate"
name="GettingStartedLib.CalculatorService">
				<endpoint address="" binding="wsHttpBinding"
bindingConfiguration="websocketHttpBinding"
                                  contract="GettingStartedLib.ICalculator">
					<identity>
						<dns value="www.newbigwater.com" />
					</identity>
				</endpoint>
```

```
                                <host>
                                        <baseAddresses>
                                                <add
baseAddress="http://localhost:8080/GettingStarted/CalculatorService" />
                                        </baseAddresses>
                                </host>
                        </service>
                </services>
                <behaviors>
                        <serviceBehaviors>
                                <behavior name="UserCertificate">
                                        <serviceMetadata httpGetEnabled="true"
httpsGetEnabled="true" />
                                        <serviceDebug includeExceptionDetailInFaults="true" />
                                        <serviceCredentials>
                                                <serviceCertificate
findValue="5619207BCAFC85D7D2FEC7A621458A9CBA863C3A"

storeLocation="LocalMachine" storeName="My" x509FindType="FindByThumbprint" />
                                                <userNameAuthentication
userNamePasswordValidationMode="Custom"

customUserNamePasswordValidatorType="GettingStartedLib.WindowsBasedValidator,
GettingStartedLib" />
                                        </serviceCredentials>
                                </behavior>
                        </serviceBehaviors>
                </behaviors>
        </system.serviceModel>

</configuration>
```

- Host

```
            // Step 1: Create a URI to serve as the base address.
            Uri baseAddress = new
Uri("http://localhost:8081/GettingStarted/CalculatorService");

            ServiceHost selfHost = new ServiceHost(typeof(CalculatorService), baseAddress);

            try
            {
                WSHttpBinding binding = new WSHttpBinding();
                binding.Security.Mode = SecurityMode.Message;
                binding.Security.Message.ClientCredentialType = MessageCredentialType.UserName;

                // Step 3: Add a service endpoint.
                selfHost.AddServiceEndpoint(typeof(ICalculator), binding, "");

                // Step 4: Enable metadata exchange.
                ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
```

```
                smb.HttpGetEnabled = true;
                selfHost.Description.Behaviors.Add(smb);

                selfHost.Credentials.UserNameAuthentication.UserNamePasswordValidationMode =
UserNamePasswordValidationMode.Custom;
                selfHost.Credentials.UserNameAuthentication.CustomUserNamePasswordValidator =
new WindowsBasedValidator();

                selfHost.Credentials.ServiceCertificate.SetCertificate(
                    System.Security.Cryptography.X509Certificates.StoreLocation.LocalMachine,
                    System.Security.Cryptography.X509Certificates.StoreName.My,

System.Security.Cryptography.X509Certificates.X509FindType.FindByThumbprint,
"5619207BCAFC85D7D2FEC7A621458A9CBA863C3A");

                // Step 5: Start the service.
                selfHost.Open();
                Console.WriteLine("The service is ready.");

                // Close the ServiceHost to stop the service.
                Console.WriteLine("Press <Enter> to terminate the service.");
                Console.WriteLine();
                Console.ReadLine();
                selfHost.Close();
            }
            catch (CommunicationException ce)
            {
                Console.WriteLine("An exception occurred: {0}", ce.Message);
                selfHost.Abort();
            }
```

- Client

```
InitializeComponent();
client = new CalculatorClient();

client.ClientCredentials.UserName.UserName = "test";
client.ClientCredentials.UserName.Password = "test";
```

04.02. Certification

- Service : App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

        <appSettings>
                <add key="aspnet:UseTaskFriendlySynchronizationContext" value="true" />
        </appSettings>
        <system.web>
```

```
                <compilation debug="true" />
        </system.web>
        <!-- 서비스 라이브러리 프로젝트를 배포할 때 호스트의 app.config 파일에 구성 파일의 내용
을 추가해야 합니다.
  System.Configuration이 라이브러리에 대한 구성 파일을 지원하지 않습니다. -->
        <system.serviceModel>
                <bindings>
                        <wsHttpBinding>
                                <binding name="websocketHttpBinding">
                                        <security mode="Message">
                                                <message clientCredentialType="Certificate"/>
                                        </security>
                                </binding>
                        </wsHttpBinding>
                </bindings>
                <services>
                        <service behaviorConfiguration="UserCertificate"
name="GettingStartedLib.CalculatorService">
                                <endpoint address="" binding="wsHttpBinding"
bindingConfiguration="websocketHttpBinding"
                                          contract="GettingStartedLib.ICalculator">
                                        <identity>
                                                <dns value="www.newbigwater.com" />
                                        </identity>
                                </endpoint>
                                <host>
                                        <baseAddresses>
                                                <add
baseAddress="http://localhost:8080/GettingStarted/CalculatorService" />
                                        </baseAddresses>
                                </host>
                        </service>
                </services>
                <behaviors>
                        <serviceBehaviors>
                                <behavior name="UserCertificate">
                                        <serviceMetadata httpGetEnabled="True"
httpsGetEnabled="True"/>
                                        <serviceDebug includeExceptionDetailInFaults="true"/>
                                        <serviceCredentials>
                                                <serviceCertificate
findValue="5619207BCAFC85D7D2FEC7A621458A9CBA863C3A" x509FindType="FindByThumbprint"
storeLocation="LocalMachine" storeName="My"/>
                                        </serviceCredentials>
                                </behavior>
                        </serviceBehaviors>
                </behaviors>
        </system.serviceModel>

</configuration>
```

- Host

```
Uri baseAddress = new Uri("http://localhost:8081/GettingStarted/CalculatorService");

            ServiceHost selfHost = new ServiceHost(typeof(CalculatorService), baseAddress);

            try
            {
                WSHttpBinding binding = new WSHttpBinding();
                binding.Security.Mode = SecurityMode.Message;
                binding.Security.Message.ClientCredentialType =
MessageCredentialType.Certificate;

                // Step 3: Add a service endpoint.
                selfHost.AddServiceEndpoint(typeof(ICalculator), binding, "");

                // Step 4: Enable metadata exchange.
                ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
                smb.HttpGetEnabled = true;
                selfHost.Description.Behaviors.Add(smb);

                selfHost.Credentials.ServiceCertificate.SetCertificate(
                    System.Security.Cryptography.X509Certificates.StoreLocation.LocalMachine,
                    System.Security.Cryptography.X509Certificates.StoreName.My,

System.Security.Cryptography.X509Certificates.X509FindType.FindByThumbprint,
"5619207BCAFC85D7D2FEC7A621458A9CBA863C3A");

                // Step 5: Start the service.
                selfHost.Open();
                Console.WriteLine("The service is ready.");

                // Close the ServiceHost to stop the service.
                Console.WriteLine("Press <Enter> to terminate the service.");
                Console.WriteLine();
                Console.ReadLine();
                selfHost.Close();
            }
            catch (CommunicationException ce)
            {
                Console.WriteLine("An exception occurred: {0}", ce.Message);
                selfHost.Abort();
            }
```

- Client

```
namespace GettingStartedClient
{
    public partial class MainForm : Form
    {
        CalculatorClient client = null;

        public MainForm()
```

```
        {
            InitializeComponent();
            client = new CalculatorClient();

            client.ClientCredentials.ClientCertificate.SetCertificate(
                System.Security.Cryptography.X509Certificates.StoreLocation.LocalMachine,
                System.Security.Cryptography.X509Certificates.StoreName.My,
                System.Security.Cryptography.X509Certificates.X509FindType.FindByThumbprint,
"5619207BCAFC85D7D2FEC7A621458A9CBA863C3A");
        }
    }
}
```