08. gRPC 실용사례 C# (mono)

본 문서 에서는 C#(mono) 에서 gRPC 를 사용하여 Client 프로그램을 작성 하는 방법을 소개 한다. Ubuntu 에서 C# 을 사용하기 위해서는 mono 를 설치 하여야 하고 .net Framwork 을 사용하기 위해서는 `dotnet` Package 를 설치 하여야 한다. 본 문서 에서는 `mono` 설치와 `dotnet` 을 설치 하는 방법도 같이 기술 하도록 하였다.

1. mono 설치

1.1 mono 설치

Ubuntu 에서 mono 설치 하는 방법은 아래 명령으로 간단히 설치 할 수 있다.

```
sudo apt-get install -y mono-devel mono-complete
```

1.2 mono test 프로그램 작성

vi 편집기 등을 이용하여 다음 소스 코드를 작성 하고 파일명을 `hello.cs` 로 저장 한다.

```
using System;

namespace NoName
{
   public class HelloWorld
   {
      public static void Main(string[] args)
      {
         Console.WriteLine("Hello World");
      }
   }
}
```

1.3 프로그램 컴파일

mono 에서 소스를 컴파일 하는 방법은 `mcs` 를 이용하여 수행 하면 된다. `hello.cs` 를 컴파일 하는 방법은 다음 명령을 실행 하면 된다.

```
mcs hello.cs
```

- 위 명령을 실행 하면 hello.exe 파일이 생성 된다.
- mcs 는 특별한 옵션을 주지 않고 실행 하면 확장자 `.exe` 를 붙인 실행 파일을 생성 한다.
- mcs 에서 출력 파일을 지정 하려면 옵션을 `-out:FILE` 처럼 지정 할 수 있다.
- 위 프로그램을 `hello` 파일로 출력 파일을 지정 하려면 `mcs -out:hello hello.cs` 로 하면 된다.

1.4 프로그램 실행

생성된 프로그램의 실행은 다음과 같다.

```
hello

or

./hello

or

mono hello
```

실행 결과 `Hello World` 이 Console 에 표시된다. hello 는 진정한 binary 실행 파일이 아니기 때문에 `ldd` 명령 또는 을 통해 확인 하면 `not a dynamic executable` 이 나오고 `nm` 명령으로 확인 하면 `nm: hello: no symbols` 가 표시 된다.

2. dotnet 설치

C# gRPC 는 .NetFramework 위에서 실행 되므로 Ubuntu 에 `dotnet` 을 설치 하여야 한다. 설치 과정은 다음과 같다.

2.1 설치 script download

[여기] (https://docs.microsoft.com/ko-kr/dotnet/core/tools/dotnet-install-script) 에 접속 하여 `https://dot.net/v1/dotnet-install.sh` 를 click 하여 설치 스크립트를 download 한다.

2.2 `dotnet` 설치

다음 명령을 실행 하여 `dotnet` 을 설치 한다.

```
./dotnet-install.sh -c Current
```

설치가 완료 되면 HOME Directory 아래 `.dotnet/` directory 가 생성 된다.

2.3 환경 변수 수정

`dotnet` 을 임의의 위치 에서 쉽게 실행 할 수 있도록 환경 변수를 수정 한다.

```
cd ~

vi .profile

파일의 제일 아래 쪽 으로 이동 후

export PATH=$PATH:~/.dotnet

를 입력 후 저장 하고 vi 를 종료 한다.
```

- 변경된 환경 변수를 적용 하기 위해서는 현재 접속되어 있는 창을 닫고 Ubuntu 에 재 접속 하거나, `soure ./.profile` 을 실행 하면 된다.

2.4 `dotnet` 설치 완료 테스트

`dotnet` 이 정상적으로 설치 되어 있는지 확인 하기 위해서 `dotnet --version` 을 실행 하여 version 정보가 표시 되면 된다. 내 기발 환경 에서는 `2.1.500` 이 표시 된다.

3. Interface 를 위한 .proto 파일

실용사례 에서 공통으로 사용할 .proto 파일로 이름은 `xdb_grpc.proto` 로 작성 하였다.
`01. gRPC 실용사례 CPP` 에 있는 내용과 동일 하다.

```
syntax = "proto3";

// Option for Java.
option java_multiple_files = true;
option java_package = "com.hancomsechre.xecuredb.grpc";
option java_outer_classname = "XecuredbProto";

// Option for Ovjective-C.
option objc_class_prefix = "XDB";

package xdbproto;

// The Xecuredb service definition.
service Xecuredb {
  rpc Encrypt (EncRequest) returns (EncReply) {}
  rpc Decrypt (DecRequest) returns (DecReply) {}
  rpc CryptoHash (HashRequest) returns (HashReply) {}
}

// The request message containing the alias and plain text.
message EncRequest {
  string alias = 1;
  string plain = 2;
}

// The response message containing the err code and cipher text.
message EncReply {
  int32 err = 1;
  string msg = 2;
  string cipher = 3;
}

// The request message containing the alias and cipher text.
message DecRequest {
  string alias = 1;
  string cipher = 2;
}

// The response message containing the err code and plain text.
message DecReply {
```

```
  int32 err = 1;
  string msg = 2;
  string plain = 3;
}

// The request message containing the alogorithm id and plain text.
message HashRequest {
  int32 id = 1;
  string plain = 2;
}

// The response message containing the err code and hash string.
message HashReply {
  int32 err = 1;
  string msg = 2;
  string cipher = 3;
}
```

4. protoc 를 이용한 Interface 파일 생성

4.1 Interface 파일 생성

C#(mono) 에서 gRPC 를 사용하기 위해서는 protoc 를 이용하여 interface 파일을 생성 하여야 한다. 다음은 interface 파일을 생성 하기 위한 명령 이다. 작업을 수행 하기 전에 작업 현재 경로에 `XdbGrpc` directory 를 만든 후 스크립트를 실행 하여야 한다. `create_csharp_src.sh`

```
# C#
protoc -I../proto --csharp_out XdbGrpc --grpc_out XdbGrpc  --plugin=protoc-gen-grpc=`which grpc_csharp_plugin` xdb_grpc.proto
```

위 명령을 실행 하면 `XdbGrpc` directory 두개의 파일리 생성 되는데 다음은 생성된 파일 목록 이다.

`./XdbGrpc` directory 의 파일 목록

```
XdbGrpc.cs
XXdbGrpcGrpc.cs
```

4.2 C# Project 파일 생성

Interface 모듈을 컴파일 하기 위해서 project파일을 생성 한다. vi 등 편집기를 이용하여 다음 내용을 작성한 다음 `XdbGrpc.csproj` 파일로 저장 한다.

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <AssemblyTitle>XdbGrpc</AssemblyTitle>
    <TargetFrameworks>netcoreapp2.1</TargetFrameworks>
    <DebugType>portable</DebugType>
    <AssemblyName>XdbGrpc</AssemblyName>
    <PackageId>XdbGrpc</PackageId>
  </PropertyGroup>
```

```
  <ItemGroup>
    <PackageReference Include="Google.Protobuf" Version="3.6.1" />
    <PackageReference Include="Google.Protobuf.Tools" Version="3.6.1" />
    <PackageReference Include="Grpc" Version="1.14.1" />
    <PackageReference Include="Grpc.Tools" Version="1.14.1" />
  </ItemGroup>

</Project>
```

4.3 Interface 빌드

project 파일이 생성 되었으면 interface 소스를 build 한다.

```
dotnet build
```

빌드가 실행 되면 다음과 같은 message 가 표시되고 `bin` 및 `obj` directory 가 생성 된다.

```
Microsoft (R) Build Engine version 15.9.20+g88f5fadfbe for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

  Restoring packages for /home/grpctest/GRPC_XDB/C#/XdbGrpc/XdbGrpc.csproj...
  Generating MSBuild file
/home/grpctest/GRPC_XDB/C#/XdbGrpc/obj/XdbGrpc.csproj.nuget.g.props.
  Generating MSBuild file
/home/grpctest/GRPC_XDB/C#/XdbGrpc/obj/XdbGrpc.csproj.nuget.g.targets.
  Restore completed in 481.52 ms for
/home/grpctest/GRPC_XDB/C#/XdbGrpc/XdbGrpc.csproj.
  XdbGrpc ->
/home/grpctest/GRPC_XDB/C#/XdbGrpc/bin/Debug/netcoreapp2.1/XdbGrpc.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:03.69
```

5. Client 코드 작성

C#(mono) 를 이용한 client 는 `CsClient.cs` 로 작성 하였으며 컴파일을 위한 project 파일은 `CsClient.csproj` 로 작성 하였다.

5.1 소스 파일 `CsClient.cs`

```
using System;
using Grpc.Core;
using Xdbproto;

namespace CsClient
{
    class Program
```

```
    {
        public static void Main(string[] args)
        {
            Channel channel = new Channel("127.0.0.1:50052",
ChannelCredentials.Insecure);

            var client = new Xecuredb.XecuredbClient(channel);
            String strAlias = "normal";
            String strPlain = "1234567890123";

            var replyEnc = client.Encrypt(new EncRequest { Alias = strAlias,
Plain = strPlain });
            String strEnc = replyEnc.Cipher;

            var replyDec = client.Decrypt(new DecRequest { Alias = strAlias,
Cipher = strEnc });
            String strDec = replyDec.Plain;

            int algo = 6;
            var replyHash = client.CryptoHash(new HashRequest { Id = algo, Plain
= strPlain });
            String strHash = replyHash.Cipher;

            Console.WriteLine("Encrypt client received: " + strEnc);
            Console.WriteLine("Decrypt client received: " + strDec);
            Console.WriteLine("Hash    client received: " + strHash);

            channel.ShutdownAsync().Wait();
        }
    }
}
```

5.2 Project 파일 `CsClient.csproj`

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <AssemblyTitle>CsClient</AssemblyTitle>
    <TargetFrameworks>netcoreapp2.1</TargetFrameworks>
    <DebugType>portable</DebugType>
    <AssemblyName>CsClient</AssemblyName>
    <OutputType>Exe</OutputType>
    <PackageId>CsClient</PackageId>
  </PropertyGroup>

  <ItemGroup>
    <ProjectReference Include="..\XdbGrpc\XdbGrpc.csproj" />
  </ItemGroup>

</Project>
```

- 위 에서 주의깊게 보아야 할 부분은 `<ProjectReference Include="..\XdbGrpc\XdbGrpc.csproj" />` 부분으로 반드시 포함 하여야 한다.

6. 컴파일 및 실행

CsClient.csproj 가 있는 directory 에서 다음 명령을 이용하여 프로그램을 실행 시킨다.

```
dotnet run -f netcoreapp2.1
```

위 명령 중 `-f netcoreapp2.1` 부분은 설치된 framework 에 따라 달라진다.

`실행 결과 출력 내용`

```
Encrypt client received: AAGo8vet8aHoqsGorhx1CsXL
Decrypt client received: 1234567890123
Hash    client received: vKKOGisl4TfIP+4Oave9Hg9SvVYFg8oHobQvmUTFxQs=
```

7. Remark

위 프로그램은 gRPC examples 에 있는 Helloworld 를 참조 하여 작성 하였다. 이상으로 개발 환경이 준비된 모든 언어 에서 gRPC 를 이용한 Interface 를 모두 테스트 하였다. 테스트를 하는 과정 에서 보면 java, c# 의 경우 환경 구축이 제일 번거로우며, 나머지는 비교적 무난 하였다.

- 끝 -