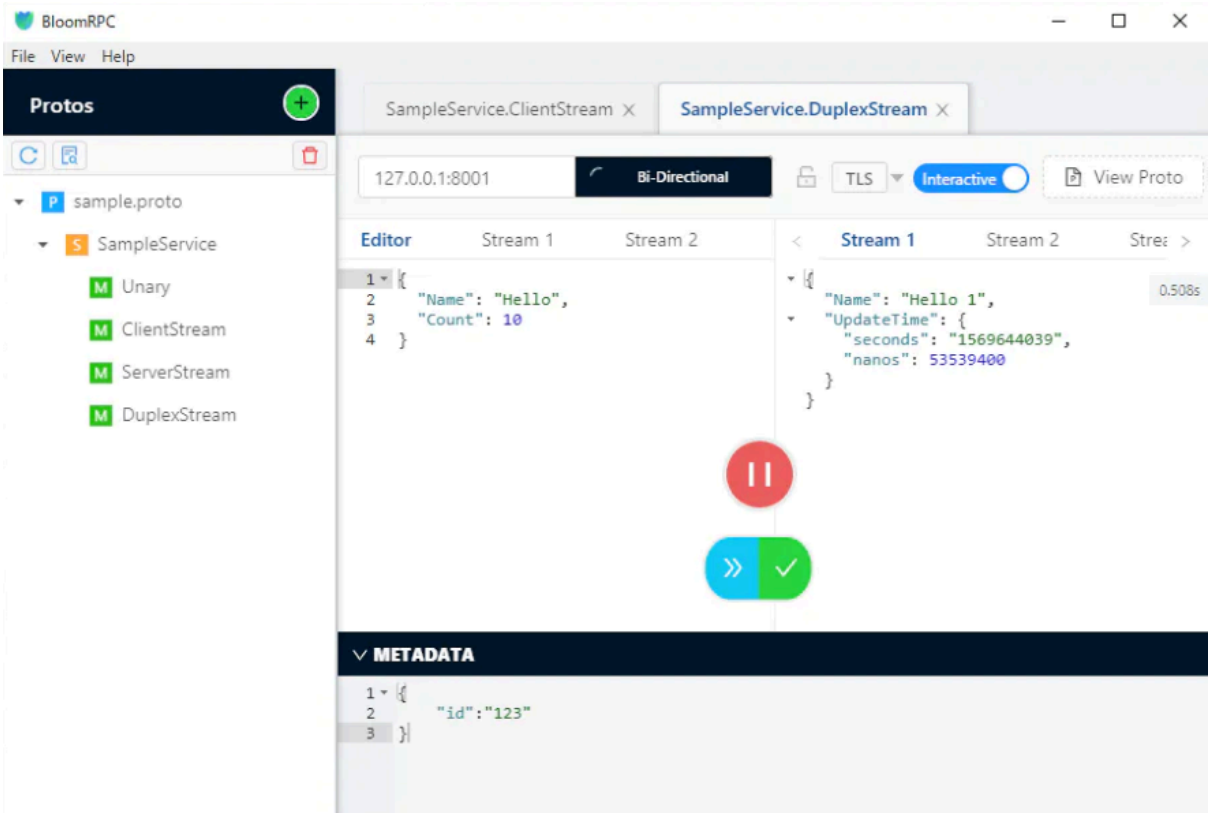
출처: https://qiita.com/mxProject/items/116eda6819560ff5660a

BloomRPC ?

【깃 허브】BloorRPC

gRPC용 GUI 클라이언트이다.
JSON 형식으로 기술한 요청 데이터를 실행중인 서비스에 보내고, 응답 데이터의 내용을 확인 할 수 있다.
스트림에도 대응하고 있다.

스크린 샷



설치

각 플랫폼의 설치가 공개되어 있다.

사용법

1. Protos 오른쪽 상단의 + 버튼을 클릭하고 원하는 proto 파일을 로드한다.
2. 실행하려는 메소드를 트리에서 선택한다.
3. 서비스 엔드 포인트를 입력한다.
4. 요청 란 편집기에 요청 데이터를 기술한다.
5. 요청 란 및 응답 필드 사이에 있는 실행 버튼을 클릭한다.
6. 응답 란에 응답 데이터가 표시된다.

스트림 작업

Interactive / manual에 의해 스트림 송수신 조작이 바뀐다.

요청 스트림 전송

인터렉티브

1. 실행 버튼을 누르면 스트림 전송이 시작되고 편집기에 작성된 요청이 전송된다.
2. 요청 측에 탭이 추가되어 전송한 요청의 내용이 표시된다.
3. Push 버튼을 누를 때마다 편집기에 기술된 요청이 전송된다.
4. Commit 버튼을 누르면 스트림 전송이 완료된다.

manual

- Push / Commit 버튼은 표시되지 않고, 한번만 전송된다. 요청 측의 편집기에 어떻게 묘사하는지 알 수 없었다. 요청을 배열로 작성하면 응답에는 빈 배열을 반환하여 기대했던 결과가 없었다.

응답 스트림을 수신

인터렉티브

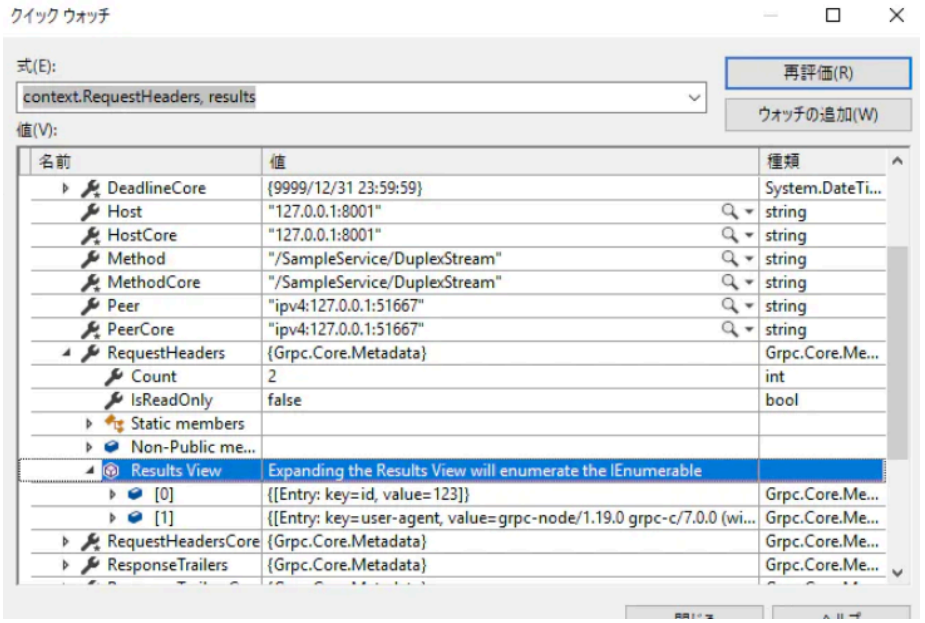
- 서비스에서 응답을 수신 할 때마다 탭이 추가되어 응답의 내용이 표시된다.

manaul

- 서비스에서 응답을 수신 할 때마다 덮어쉬운다. 탭은 추가되지 않는다.

요청 헤더

METADATA 란에서 요청 헤더에 저장된 데이터를 지정할 수 있다.
아래 그림은 Visual Studio에서 디버깅 실행했을 때 요청 헤더의 내용이다. METADATA 란에 기술 한 "id": "123"가 포함 되어 있는 것을 알 수 있다.



확인에 사용한 proto 파일

```
syntax = "proto3";

import "timestamp.proto";

option csharp_namespace = "SampleIDL";

message SampleRequest
{
    string Name = 1;
    int32 Count = 2;
}

message SampleResponse
{
    string Name = 1;
    google.protobuf.Timestamp UpdateTime = 2;
}

message SampleResponseList
{
    repeated SampleResponse Items = 1;
}

service SampleService
{
    rpc Unary(SampleRequest) returns (SampleResponseList);
    rpc ClientStream(stream SampleRequest) returns (SampleResponseList);
    rpc ServerStream(SampleRequest) returns (stream SampleResponse);
    rpc DuplexStream(stream SampleRequest) returns (stream
SampleResponse);
}
```

확인에 사용된 서비스 구현

SampleRequest.Count에서 지정된 수만큼 SampleResponse를 생성해서 돌려준다.

```
class SampleServiceImpl : SampleService.SampleServiceBase
{
    internal SampleServiceImpl() : base()
    {
    }
```

```
    public async override Task<SampleResponseList> Unary(SampleRequest
request, ServerCallContext context)
    {
        SampleResponseList list = new SampleResponseList();

        foreach (SampleResponse response in GetResponses(request))
        {
            await RamdomDelayAsync();
            list.Items.Add(response);
        }

        return list;
    }

    public async override Task<SampleResponseList>
ClientStream(IAsyncStreamReader<SampleRequest> requestStream,
ServerCallContext context)
    {
        SampleResponseList list = new SampleResponseList();

        while (await requestStream.MoveNext())
        {
            foreach (SampleResponse response in
GetResponses(requestStream.Current))
            {
                await RamdomDelayAsync();
                list.Items.Add(response);
            }
        }

        return list;
    }

    public async override Task ServerStream(SampleRequest request,
IServerStreamWriter<SampleResponse> responseStream, ServerCallContext
context)
    {
        foreach (SampleResponse response in GetResponses(request))
        {
            await RamdomDelayAsync();
            await responseStream.WriteAsync(response);
        }
    }

    public async override Task
```

```
DuplexStream(IAsyncStreamReader<SampleRequest> requestStream,
IServerStreamWriter<SampleResponse> responseStream, ServerCallContext
context)
    {
        while (await requestStream.MoveNext())
        {
            foreach (SampleResponse response in
GetResponses(requestStream.Current))
            {
                await RamdomDelayAsync();
                await responseStream.WriteAsync(response);
            }
        }
    }

    private IEnumerable<SampleResponse> GetResponses(SampleRequest
request)
    {
        for (int i = 0; i < request.Count; ++i)
        {
            yield return new SampleResponse()
            {
                Name = string.Format("{0} {1}", request.Name, i + 1),
                UpdateTime =
Timestamp.FromDateTimeOffset(DateTimeOffset.Now)
            };
        }
    }

    private Task RamdomDelayAsync()
    {
        return Task.Delay(random.Next(1000));
    }
    static Random random = new Random();

}
```