### 什么是gRPC

gRPC是google开源的一个高性能、跨语言的RPC框架，基于HTTP2协议，采用ProtoBuf 定义的IDL。

gRPC 的主要优点是：

* 现代高性能轻量级 RPC 框架。
* 协定优先 API 开发，默认使用协议缓冲区，允许与语言无关的实现。
* 可用于多种语言的工具，以生成强类型服务器和客户端。
* 支持客户端、服务器和双向流式处理调用。
* 使用 Protobuf 二进制序列化减少对网络的使用。

这些优点使 gRPC 适用于：

* 效率至关重要的轻量级微服务。
* 需要多种语言用于开发的 Polyglot 系统。
* 需要处理流式处理请求或响应的点对点实时服务。

更多介绍请前往 https://grpc.io/docs/guides/

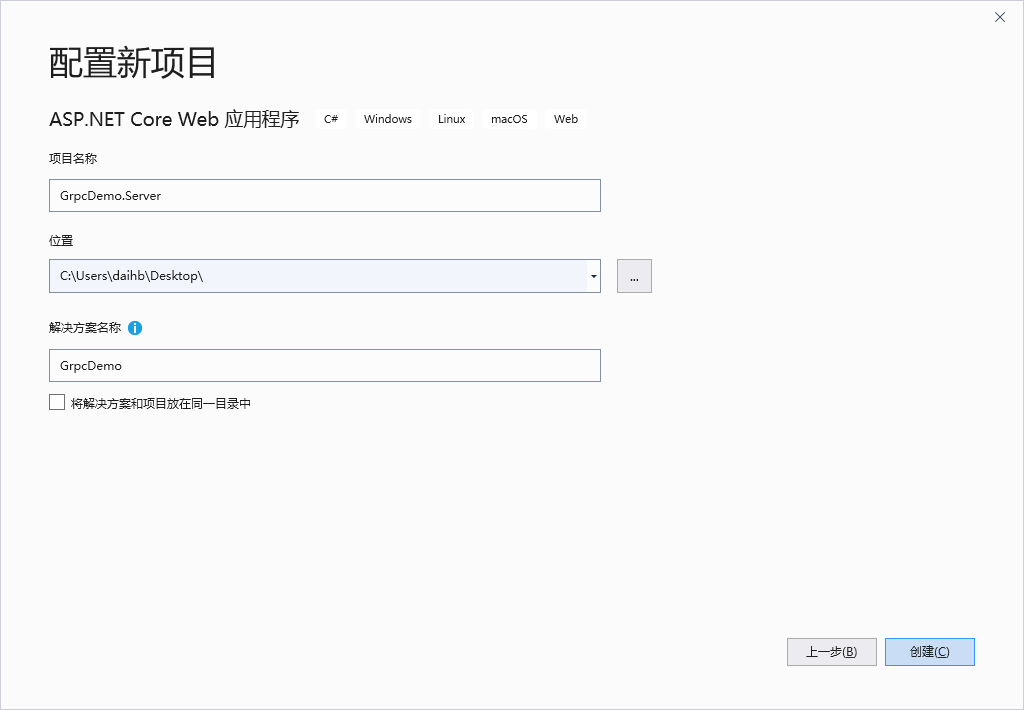
### 开始

.net core 3.0中已经加入了gRPC的项目模板，使用vs2019就可以创建gRpc项目。

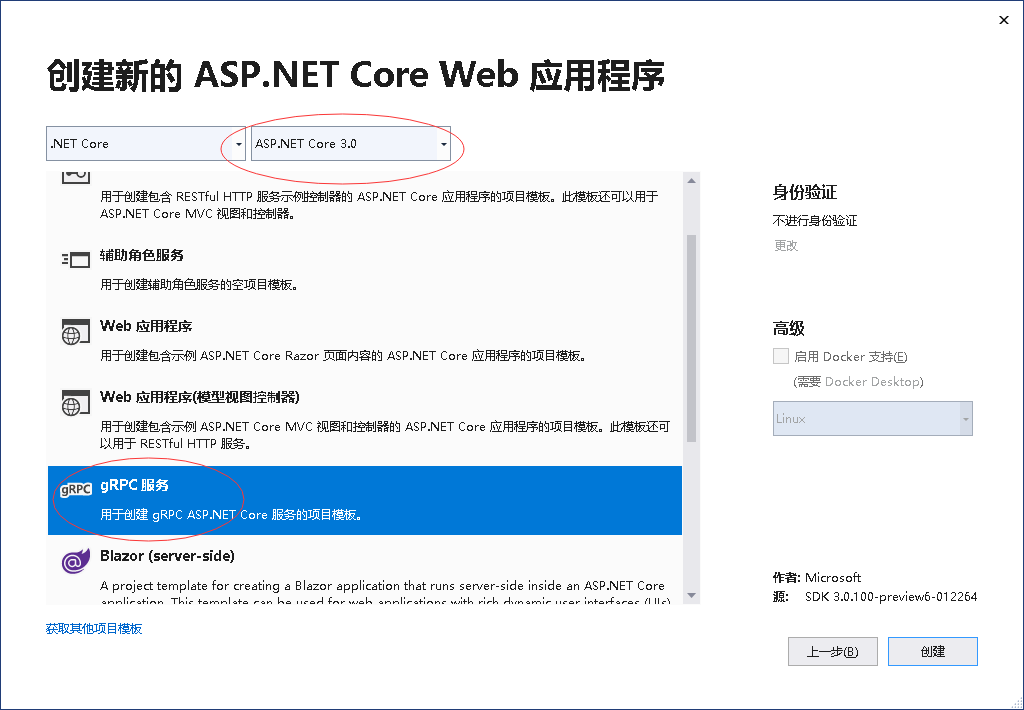
1.打开vs2019，创建一个asp .net core web项目



2.点击下一步，设置项目名称和项目路径和解决方案名称。项目名称就叫GrpcDemo.Server，路径我这里就放到桌面，解决方案名称就叫GrpcDmeo。



3.点击创建，框架选择asp .net core 3.0,模板选择gRPC服务。



点击创建就可以创建gRPC服务项目了，如果没有出现asp .net core 3.0框架选项或者grpc服务模板。请到工具-选项-环境-预览功能 里面将使用.NET Core SDK的预览勾选上。前提是要有下载.net core 3.0的sdk。

4.创建成功后会出现下图所示的项目结构。



还会默认安装三个依赖包，分别是

* [Grpc.AspNetCore.Server](https://www.nuget.org/packages/Grpc.AspNetCore.Server)
* [Google.Protobuf](https://www.nuget.org/packages/Google.Protobuf/)对于Protobuf消息API。
* [Grpc.Tools](https://www.nuget.org/packages/Grpc.Tools/) 这个包主要将proto文件生成C#类

首先解析一下gRPC的模板信息，在HelloGrpc.Server 服务的端项目中有如下几个文件

* greet.proto：*greet.proto* 文件定义 Greeter gRPC，且用于生成 gRPC 服务器资产。
* Services 文件夹：包含 Greeter 服务的实现。
* appSettings.json：包含配置数据，如 Kestrel 使用的协议。（熟悉ASP.NET Core的你一定很熟悉）
* Program.cs:包含 gRPC 服务的入口点。（熟悉ASP.NET Core的你一定很熟悉）
* Startup.cs:IWebHostBuilder的启动配置文件,包含配置应用行为的代码。（熟悉ASP.NET Core的你一定很熟悉）

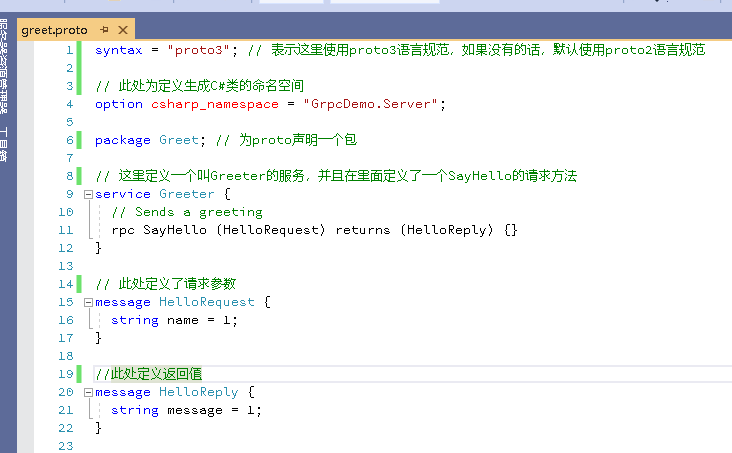
再详细看下每个文件内的内容。

### proto文件

GRPC使用约定优先的API开发方法。默认情况下，使用协议缓冲区(Protobuf)作为接口设计语言(IDL)。这个.proto文件包含：

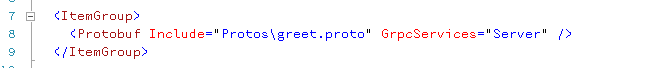
* GRPC服务的定义。
* 在客户端和服务器之间发送的消息。

有关Protobuf文件语法的更多信息，请参见[正式文件(原型)](https://developers.google.com/protocol-buffers/docs/proto3).如我们模板中创建的greet.proto 文件内容如下：



* 定义Greeter服务。
* 这个Greeter服务定义SayHello请求。
* SayHello发送HelloRequest消息并接收HelloReply信息：

我们来看看，这个*.proto*文件是如何包含在项目中的呢，其实，如果你打开项目的.csproj文件就会看到，通过将该文件添加到<Protobuf>的ItemGroup中即可，如下所示：



Include属性表示将greet.proto文件包含在我们的项目当中，GrpcServices="Server"表示仅在服务器项目中生成资源文件。但是这样远远不够，以后我们可能还会在这个文件夹中增加其他proto文件。所以我们需要改动一下。

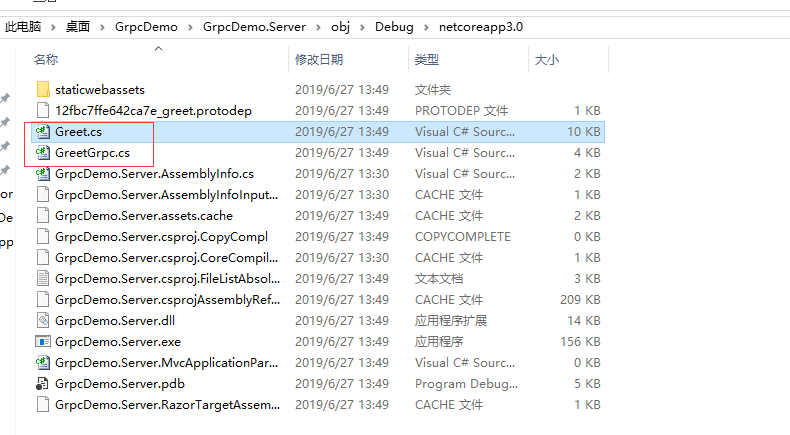
<ItemGroup>

<Protobuf Include="Protos\\*.proto" GrpcServices="Server" />

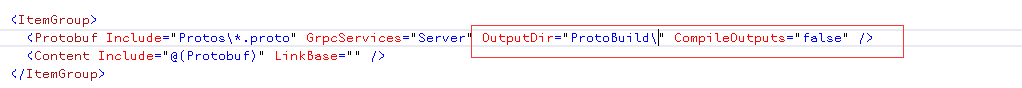
<Content Include="@(Protobuf)" LinkBase="" />

</ItemGroup>

这里表示将项目下的Protos文件夹所有以proto结尾的文件包含到项目中。再右键项目生成一下，然后打开项目的obj\Debug\netcoreapp3.0目录，就能发现两个生成的类文件。分别是Greet.cs和GreetGrpc.cs。有兴趣可以打开进行深入研究。



如果不想生成的文件放到这个目录，可以配置一下上面所说的<Protobuf>节点，配置一些具体的信息，可以配置文件的生成目录。



设置OutputDir属性就可以设置文件的输出目录，上面这里就设置了文件的输出目录为ProtoBuild目录，另外还设置了CompileOutputs="false"，这是为了防止止将生成的文件编译为程序集。编译一下，就可以看到文件出现在了ProtoBuild目录。



关于Protobuf的详细配置信息，可以参考下面的链接。里面还有各种奇奇怪怪的配置。

<https://github.com/grpc/grpc/blob/master/src/csharp/BUILD-INTEGRATION.md>

编译项目能够将proto文件生成类文件，是引用Grpc.Tools了这个工具包的原因。

### Services 文件夹中的具体的gRPC服务

下面再看一下services目录里的文件。我们知道Grpc.Tools工具包将根据.proto文件的定义翻译并生成对应的C#类型的文件。

对于服务器端资产，将生成一个抽象的服务基类型。基类型包含在.proto文件中包含的所有GRPC调用的定义。然后，您将创建从此基类型派生的具体服务实现，并实现GRPC调用的逻辑。对于前面描述的greet.proto示例，将生成包含虚拟SayHello方法的抽象GreeterBase类型。具体的实现GreeterService重写该方法并实现处理GRPC调用的逻辑。

正如GrpcDemo.Server项目中的Services\GreeterService.cs中的代码



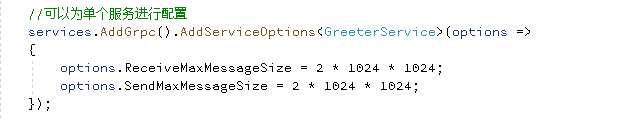
### Startup

在Startup中我们发现跟普通的ASP.NET Core程序有所不同，具体的如下图所示：在ConfigureServices 服务中引入了gRPC服务，然后在Configure配置了服务。

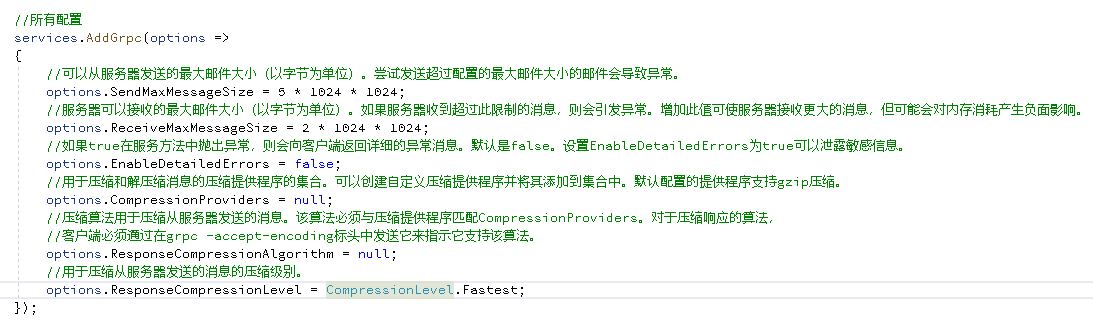


在StartUp的方法里也可以对服务进行一定配置。

可以对单个服务进行配置：



也可以对所有服务进行配置：



### 其他需要说明的内容

与ASP.NET Core 接口的集成

GRPC服务可以完全访问ASP.NETCore功能，如依赖注入(Di)和日志功能。例如，服务实现可以通过构造函数解析DI容器中的记录器服务：

public class GreeterService : Greeter.GreeterBase

{

public GreeterService(ILogger<GreeterService> logger)

{

}

}

默认情况下，GRPC服务可以解析具有任意生存期的其他DI服务(Singleton, Scoped, or Transient)。

在GRPC方法中解析HttpContext

GRPC 应用程序接口提供对某些HTTP/2消息数据的访问，例如method, host, header, and trailers。访问是通过ServerCallContext参数传递给每个GRPC方法：

public class GreeterService : Greeter.GreeterBase

{

public override Task<HelloReply>

SayHello(HelloRequest request, ServerCallContext context)

{

return Task.FromResult(new HelloReply

{

Message = "Hello " + request.Name

});

}

}

ServerCallContext不提供对所有ASP.NET 接口中HttpContext的完全访问。GetHttpContext扩展方法提供对表示ASP.NET API中底层HTTP/2消息的httpContext的完全访问：

public class GreeterService : Greeter.GreeterBase

{

Public override Task<HelloReply> SayHello(HelloRequest request, ServerCallContext context)

{

var httpContext = context.GetHttpContext();

return Task.FromResult(new HelloReply

{

Message = "Using https: " + httpContext.Request.IsHttps

});

}

}

请求体数据速率限制

默认情况下，Kestrel服务器设置为最小请求主体数据速率。对于客户端流式和双工流式的请求，此速率可能不满足，并且连接可能超时。当GRPC服务包括客户端流和双工流调用时，必须禁用最小请求正文数据速率限制：

public class Program

{

public static void Main(string[] args)

{

CreateHostBuilder(args).Build().Run();

}

public static IHostBuilder CreateHostBuilder(string[] args) =>

Host.CreateDefaultBuilder(args)

.ConfigureWebHostDefaults(webBuilder =>

{

webBuilder.UseStartup<Startup>();

webBuilder.ConfigureKestrel((context, options) =>

{

options.Limits.MinRequestBodyDataRate = null;

});

});

}

### 客户端调用

1.右键点击解决方案，新建一个控制台项目，名字叫GrpcDemo.Client。并使用命令安装依赖包。

dotnet add GrpcGreeterClient.csproj package Grpc.Net.Client –version 0.1.21-pre1

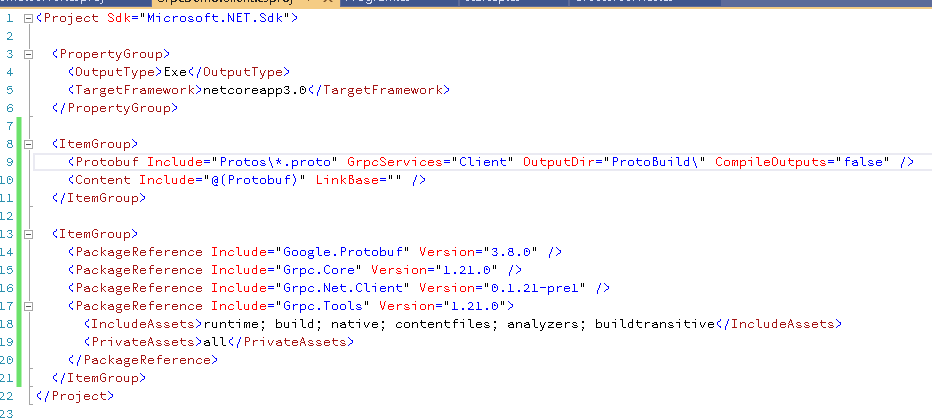
dotnet add GrpcGreeterClient.csproj package Google.Protobuf

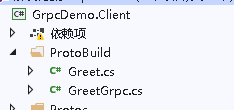
dotnet add GrpcGreeterClient.csproj package Grpc.Tools

2.将服务端项目的Protos文件夹复制到客户端项目，新建ProtoBuild文件夹存放proto编译后的类。



并且修改项目的csproj文件，修改为以下内容。与服务端不同的是Protobuf节点的GrpcServices="Client"属性，表示仅在客户端项目生成。

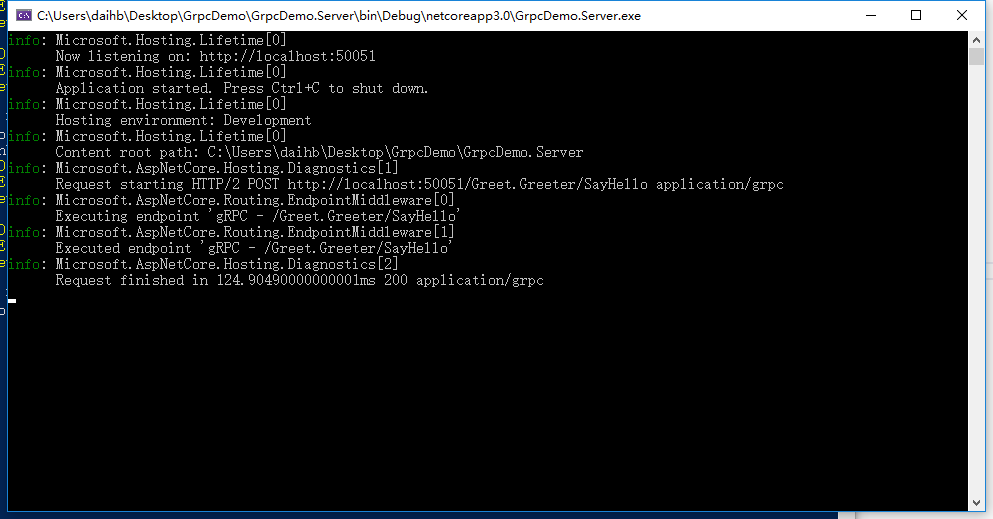


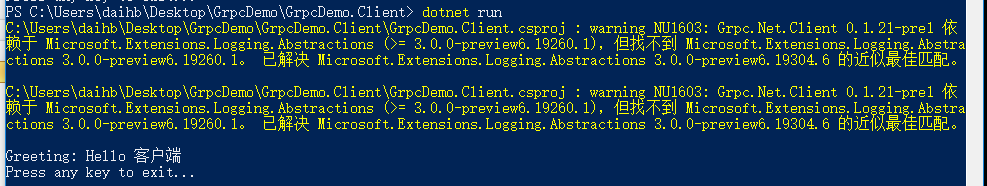
然后右键生成项目，这时候ProtoBuild目录下应该有生成的类文件了

在Program文件中编写以下代码进行调用服务端：



然后先启动服务端项目，再启动客户端，就能成功调用服务端的方法了。





可以看到成功的调用的服务端的服务，警告是由于包的问题，可忽略不计。

参考文章：<https://www.cnblogs.com/yilezhu/p/10631420.html>