

第81期-关于gRPC的相关配置

2020年1月13日 14:22

配置 TLS 证书和 HTTP2 协议

appsettings.json 配置文件

```
{
  "Kestrel": {
    "Endpoints": {
      "HttpsInlineCertFile": {
        "Url": "https://localhost:5001",
        "Protocols": "Http2",
        "Certificate": {
          "Path": "<path to .pfx file>",
          "Password": "<certificate password>"
        }
      }
    }
  }
}
```

硬编码配置

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureWebHostDefaults(webBuilder =>
        {
            webBuilder.ConfigureKestrel(options =>
            {
                options.Listen(IPAddress.Any, 5001, listenOptions =>
                {
                    listenOptions.Protocols =
                        HttpProtocols.Http2;
                    listenOptions.UseHttps("<path to .pfx
file>", "<certificate password>");
                });
            });
            webBuilder.UseStartup<Startup>();
        });
```

与 ASP.NET Core 集成

gRPC 服务直接使用 ASP.NET Core 依赖注入容器注入。

```
public class GreeterService : Greeter.GreeterBase
{
    public GreeterService(ILogger<GreeterService> logger)
    {
    }
}
```

从 gRPC 服务方法中解析 HttpContext 实例

```
public class GreeterService : Greeter.GreeterBase
{
    public override Task<HelloReply> SayHello(HelloRequest request,
ServerCallContext context)
    {
        var httpContext = context.GetHttpContext();
        var clientCertificate = httpContext.Connection.ClientCertificate;

        return Task.FromResult(new HelloReply
        {
            Message = "Hello " + request.Name + " from " +
clientCertificate.Issuer
        });
    }
}
```

gRPC 配置选项

配置服务端

选项	默认值	描述
MaxSendMessageSize	null	服务器发送的最大消息大小（以字节为单位）。尝试发送超过配置的最大消息大小的消息将导致异常。
MaxReceiveMessageSize	4 MB	服务器可接收的最大消息大小（以字节为单位）。如果服务器收到的消息超过此限制，则会引发异常。增大此值可使服务器接收更大的消息，但会对内存消耗产生负面影响。
EnableDetailedErrors	false	如果 true，则当服务方法中引发异常时，详细的异常消息将返回到客户端。默认值为 false。将 EnableDetailedErrors 设置为 true 可能会泄漏敏感信息。
CompressionProviders	gzip	用于压缩和解压缩消息的压缩提供程序的集合。可以创建自定义压缩提供程序并将其添加到集合中。默认配置的提供程序支持 gzip 压缩。
ResponseCompressionAlgorithm	null	压缩算法用于压缩从服务器发送的消息。该算法必须与 CompressionProviders 中的压缩提供程序匹配。为了使算

		法压缩响应，客户端必须通过在 grpc-accept 标头中发送来指示它支持该算法。
ResponseCompressionLevel	null	用于压缩从服务器发送的消息的压缩级别。
Interceptors	无	运行每个 gRPC 调用的侦听器的集合。拦截按注册顺序运行。全局配置的侦听器在为单个服务配置侦听器之前运行。

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddGrpc(options =>
    {
        options.EnableDetailedErrors = true;
        options.MaxReceiveMessageSize = 2 * 1024 * 1024; // 2 MB
        options.MaxSendMessageSize = 5 * 1024 * 1024; // 5 MB
    });
}
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddGrpc().AddServiceOptions<MyService>(options =>
    {
        options.MaxReceiveMessageSize = 2 * 1024 * 1024; // 2 MB
        options.MaxSendMessageSize = 5 * 1024 * 1024; // 5 MB
    });
}
```

配置客户端

选项	默认值	描述
HttpClient	新实例	用于进行 gRPC 调用的 HttpClient。可以将客户端设置为配置自定义 HttpClientHandler，或将其他处理程序添加到 gRPC 调用的 HTTP 管道。如果未指定 HttpClient，则将为该通道创建新的 HttpClient 实例。它将自动被释放。
DisposeHttpClient	false	如果 true 并且指定了 HttpClient，则在释放 GrpcChannel 时将释放 HttpClient 实例。
LoggerFactory	null	客户端用来记录有关 gRPC 调用的信息的 LoggerFactory。可以通过依赖关系注入或使用 LoggerFactory.Create 创建 LoggerFactory 实例。有关配置日志记录的示例，请参阅 .NET 上的 gRPC 中的日志记录和诊断 。
MaxSendMessageSize	null	可从客户端发送的最大消息大小（以字节为单位）。尝试发送超过配置的最大消息大小的消息将导致异常。
MaxReceiveMessageSize	4 MB	客户端可以接收的最大消息大小（以字节为单位）。如果客户端收到的消息超过此限制，则会引发异常。增大此值可使客户端接收更大的消息，但会对内存消耗产生负面影响。
Credentials	null	一个 ChannelCredentials 实例。凭据用于将身份验证元数据添加到

		gRPC 调用。
CompressionProviders	gzip	用于压缩和解压缩消息的压缩提供程序的集合。 可以创建自定义压缩提供程序并将其添加到集合中。 默认配置的提供程序支持gzip压缩。

```
static async Task Main(string[] args)
{
    var channel = GrpcChannel.ForAddress("https://localhost:5001", new
GrpcChannelOptions
    {
        MaxReceiveMessageSize = 5 * 1024 * 1024, // 5 MB
        MaxSendMessageSize = 2 * 1024 * 1024 // 2 MB
    });

    var client = new Greeter.GreeterClient(channel);

    var reply = await client.SayHelloAsync(new HelloRequest { Name =
"GreeterClient" });

    Console.WriteLine("Greeting: " + reply.Message);
}
```

服务端日志配置

由于 gRPC 服务托管在 ASP.NET Core 上，因此它使用 ASP.NET Core 日志记录系统。在默认配置中，gRPC 记录的信息非常小，但这可以进行配置。

配置文件配置

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information",
      "Grpc": "Debug"
    }
  }
}
```

硬编码配置

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
        {
            logging.AddFilter("Grpc", LogLevel.Debug);
        })
        .ConfigureWebHostDefaults(webBuilder =>
```

```

    {
        webBuilder.UseStartup<Startup>();
    });

```

环境变量配置

Logging:LogLevel:Grpc = Debug

客户端日志配置

如果要从 ASP.NET Core 的应用程序调用 gRPC 服务，则可以通过依赖关系注入（DI）来解析记录器工厂：

```

[ApiController]
[Route("[controller]")]
public class GreetingController : ControllerBase
{
    private ILoggerFactory _loggerFactory;

    public GreetingController(ILoggerFactory loggerFactory)
    {
        _loggerFactory = loggerFactory;
    }

    [HttpGet]
    public async Task<ActionResult<string>> Get(string name)
    {
        var channel = GrpcChannel.ForAddress("https://localhost:5001",
            new GrpcChannelOptions { LoggerFactory = _loggerFactory });
        var client = new Greeter.GreeterClient(channel);

        var reply = await client.SayHelloAsync(new HelloRequest { Name =
name });
        return Ok(reply.Message);
    }
}

```

如果你的应用未使用依赖注入容器，则可以使用 LoggerFactory 创建日志记录：

```

var loggerFactory = LoggerFactory.Create(logging =>
{
    logging.AddConsole();
    logging.SetMinimumLevel(LogLevel.Debug);
});

var channel = GrpcChannel.ForAddress("https://localhost:5001",
    new GrpcChannelOptions { LoggerFactory = loggerFactory });

```

```
var client = Greeter.GreeterClient(channel);
```