реализация в .NET Core

Патудин Иван

Краткое содержание

- Что такое gRPC
- Сравнение с REST и WCF
- Язык Protocol buffer
- Особенности gRPC и работа с браузером
- gRPC в .NET Core 3
- Пример сервиса на REST и gRPC и средства тестирования

Что такое gRPC

- gRPC - Google Remote Procedure Calls
- “Высокопроизводительный фреймворк для удаленного вызова процедур с открытым исходным кодом”
- Contract-first, использует protobuf для определения интерфейса
- Multiplatform
- HTTP 2.0

Зачем это нужно

{REST}


GRPC


{REST}

gRPC

- Фреймворк удаленного вызова процедур
- Protocol Buffers. Автогенерация клиента и сервера
- HTTP/2; Streaming
- Ограниченные возможности по работе напрямую с браузером
- Ограниченный набор инструментов тестирования

REST

- Предназначен прежде всего для управления ресурсами
- OpenAPI
- Работа только в формате запрос-ответ
- Отличное взаимодействие с браузерами
- Большой набор средств, включая отладку из браузера


GRPC


Microsoft
.NET
WCF

gRPC

- Теперь официально поддерживается;
- Contract-first
- Прекрасные кроссплатформенные возможности

WCF

- Устаревающая технология
- Code-first;
- Слабая возможность межплатформенного взаимодействия

Benchmarks

```
[ServiceContract]
public interface ISomeService
{
    [OperationContract]
    IEnumerable<CompositeType> GetUserData(int min, int max);
}
```

```
public class SomeService : ISomeService
{
    public IEnumerable<CompositeType> GetUserData(int min, int max)
    {
        var response = new CompositeType[max - min];
        for (int i = min; i < max; i++)
        {
              response[i] = new CompositeType {IntValue = i, StringValue = i.ToString() };
        }
        return response;
    }
}
```

```
var someService = new SomeService();
var host = new ServiceHost(someService);

var binding = new NetTcpBinding();
host.AddServiceEndpoint(
    typeof(ISomeService), binding,
    "net.tcp://localhost:5000/serv"
);
```

```
[GlobalSetup]
public void Setup()
{
    _client = new SomeService.SomeServiceClient(
        new NetTcpBinding(),
        new EndpointAddress("net.tcp://localhost:5000/serv")
    );
}
```

```
[Benchmark]
public int Wcf()
{
    var response =  _client.GetUserData(0, 42);

    var sum = 0;
    foreach (var item in response)
    {
        sum += item.IntValue;
    }
    return sum;
}
```

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
Wcf	211.5 us	1.625 us	1.440 us	0.7324	-	-	3.52 KB

```
syntax = "proto3";

package GrpcService;

service SomeGrpcService {
    rpc  GetUserData(GetDataRequest) returns (GetDataResponse) {}
}

message GetDataStreamRequest {
    int32 min = 1;
    int32 max = 2;
}

message GetDataResponse {
    repeated CompositeType value = 1;
}

message CompositeType {
    bool BoolValue = 1;
    string StringValue = 2;
    int32 IntValue = 3;
}
```

```
public class SomeService : SomeService.SomeServiceBase
{
    public override Task<GetDataStreamResponse> GetUserData(GetDataStreamRequest request, ServerCallContext context)
    {
        var response = new GetDataStreamResponse();
        foreach (int i = min; i < max; i++)
        {
            response.Value.Add(new CompositeType {IntValue = i, StringValue = i.ToString() });
        }

        return Task.FromResult(response);
    }
}
```

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapGrpcService<SomeService>();
    });
}
```

```
[Benchmark]
public async Task<int> Grpc()
{
    var response = await _client.GetDataStreamAsync(
        new GetDataStreamRequest {Min = 0, Max = 42}
    );

    var sum = 0;
    foreach (var item in response.Value)
    {
        sum += item.IntValue;
    }
    return sum;
}
```

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
Grpc	190.1 us	1.984 us	1.856 us	1.4648	-	-	1.14 KB

gRPC vs WCF

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
WCF	211.5 us	1.625 us	1.440 us	0.7324	-	-	3.52 KB
gRPC	190.1 us	1.984 us	1.856 us	1.4648	-	-	1.14 KB

- gRPC выделяет меньше памяти
- gRPC хоть и незначительно но быстрее (20us)

Protobuf

Protocol buffers (protobuf)

gRPC использует contract-first подход для разработки API.

- Язык описания интерфейса сервиса
- Формат сериализации используемый по умолчанию
- Используя строгую типизацию полей и бинарный формат
- Быстрая сериализации/десериализации

Protocol buffers (protobuf)

```
syntax = "proto3";
```

```
import "models/realty.proto";
```

```
//something very important!
option csharp_namespace = "DowntownRealty";
```

```
message  RealtyRequest{
   int64 id = 1;
}
```

```
message  RealtyResponse{
   RealtyAd realty = 1;
}
```

```
service DowntownRealty{
   rpc GetRealtyById (RealtyRequest) returns (RealtyResponse);
   rpc GetRealtyList (RealtyListRequest) returns (RealtyListResponse);
   rpc GetUserById (stream UserRequest) returns (stream UserResponse);
}
```

Protocol buffers (protobuf)

```
message RealtyAd{
   int32  id = 1;
   RealtyType type =2;
   string topic = 3;
   string message  = 4;
   string phone = 5;
}


enum RealtyType {
    OTHER = 0;
    COMMERCIAL = 1;
    LIVING = 2;
  }
```

Типы данных protobuf

Protocol buffers (protobuf)

- reserved

```
message User {
   string id = 1;
   reserved 2, 3;
   int32 age = 4;
}
```

- any

```
import "google/protobuf/any.proto";

message ErrorStatus {
  string message = 1;
  repeated google.protobuf.Any details = 2;
}
```

c#: object

Protocol buffers (protobuf)

- maps

```
message SampleMessage {
    map<string, Project> projects = 3;
}
```

c#: `Dictionary<string, Project>`

- oneof

```
message SampleMessage {
  oneof test {
    string name = 1;
    SubMessage message = 2;
  }
}
```

```
enum TestOneofCase
{
  None = 0,
  Name = 1,
  Message = 2
}

public TestOneofCase TestCase { get; }
public void ClearTest();
public string Name { get; set; }
public SubMessage Message { get; set; }

switch (change.TestCase) {
        case TestOneofCase.None:
                return;
        case TestOneofCase.Name: FormatName(change.Name);
                break;
        case TestOneofCase.Message: FormatMessage(change.Message);
                break;
        default: throw new ArgumentException("Unknown instrument type");
}
```

Nullable

```
import "google/protobuf/wrappers.proto

message Person {
      ...
      google.protobuf.Int32Value age = 5;
}


message Int32Value {
  // The int32 value.
  int32 value = 1;
}
```

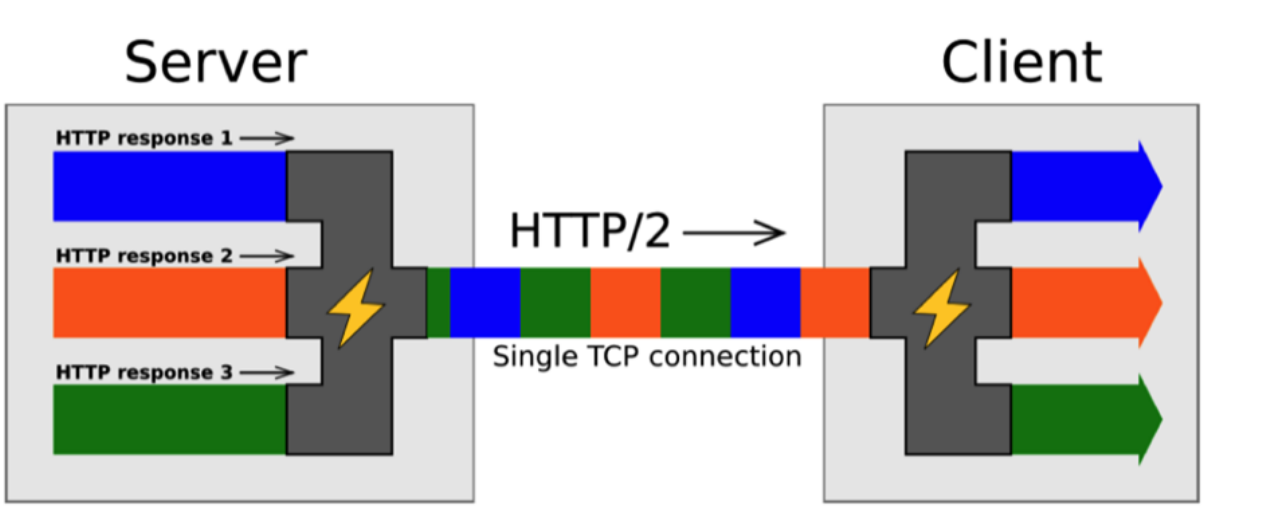
Особенности HTTP/2

Особенности HTTP/2

- Протокол HTTP/2 является бинарным
- Может отправлять не запрошенные данные в рамках соединения
- Решена проблема Head-of-line blocking
- Мультиплексирование в HTTP/2

Особенности HTTP/2

- Мультиплексирование в HTTP/2



Устройство

Варианты взаимодействия

Unary RPC

```
rpc SayHello(HelloRequest) returns (HelloResponse);
```

Client streaming RPC

```
rpc LotsOfGreetings(stream HelloRequest) returns (HelloResponse)
```

Server streaming

```
rpc LotsOfReplies(HelloRequest) returns (stream HelloResponse)
```

Bidirectional streaming RPC

```
rpc BidiHello(stream HelloRequest) returns (stream HelloResponse)
```

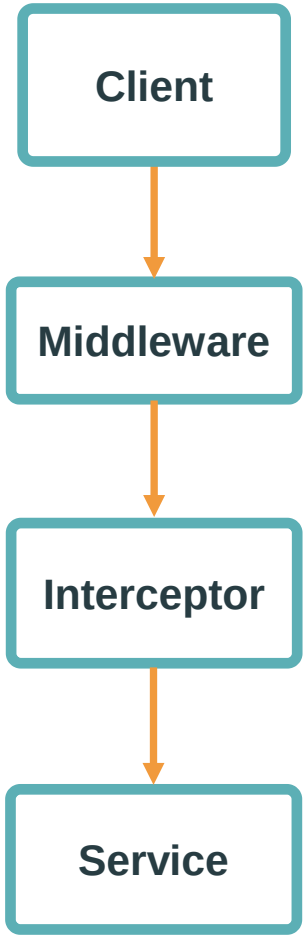
Подключение

- Deadlines (Timeouts)
- Metadata
- Channels
- RPC termination

Interceptors


Client
Middleware
Interceptor

```
 public abstract class Interceptor
 {

        public delegate TResponse BlockingUnaryCallContinuation<TRequest, TResponse>(TRequest
request, ClientInterceptorContext<TRequest, TResponse> context) ……


        public delegate AsyncUnaryCall<TResponse> AsyncUnaryCallContinuation<TRequest,
TResponse>(TRequest request, ClientInterceptorContext<TRequest, TResponse> context) ……


//…… …… //


        public virtual Task<TResponse> UnaryServerHandler<TRequest, TResponse>(TRequest request,
ServerCallContext context, UnaryServerMethod<TRequest, TResponse> continuation) ……


public virtual Task DuplexStreamingServerHandler<TRequest, TResponse>(ServerCallContext context,
DuplexStreamingServerMethod<TRequest, TResponse> continuation) ……

}
```


Client
Middleware
Interceptor
Service

Exceptions

Exceptions

`class RpcException`

Ограниченный набор статусов: `enum StatusCode`

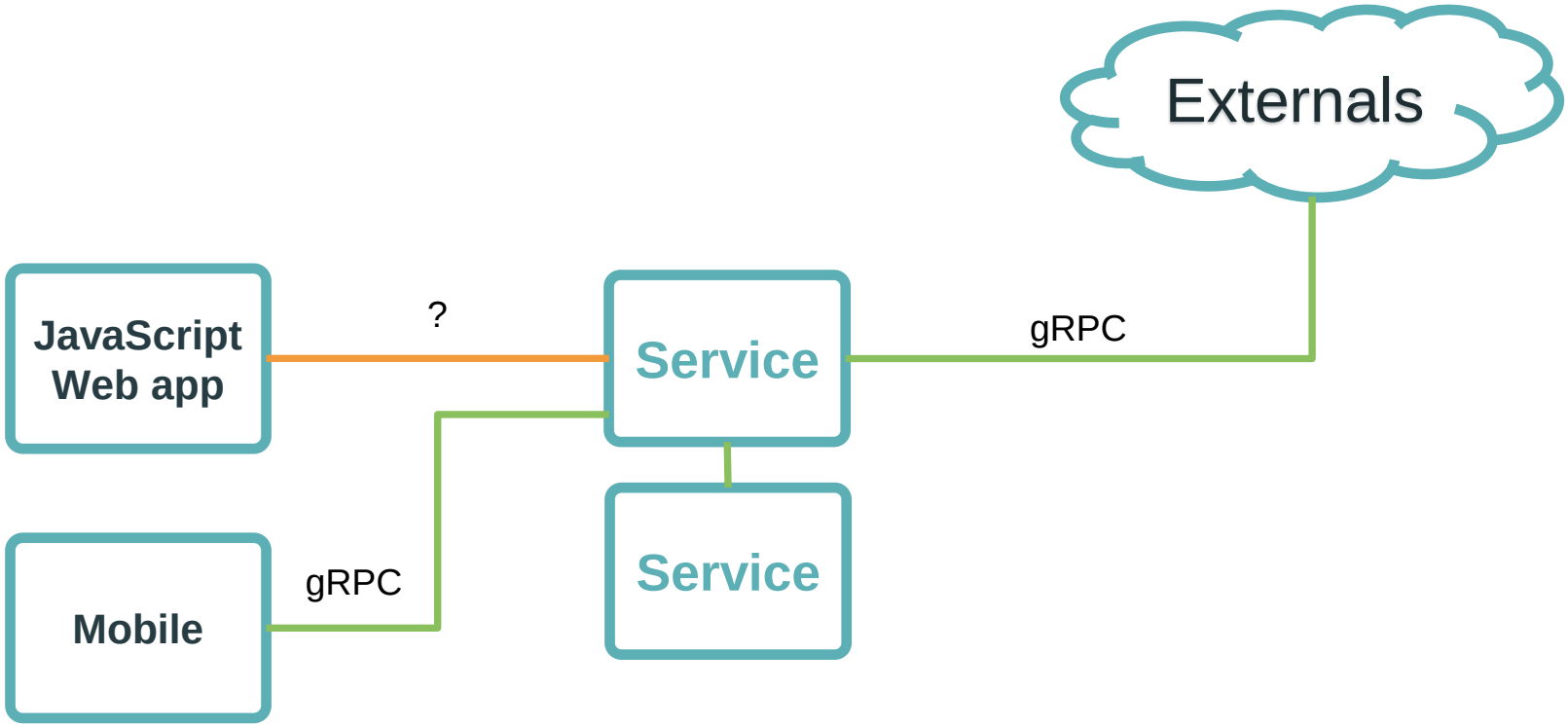
Использование метаданных

```
//... ...
if (!ValidateUser(user))
{
     var metadata = new Metadata  {  { "User", user.Identity.Name }  };
     throw new RpcException(new Status(StatusCode.PermissionDenied, "Permission
denied"), metadata);
}
```

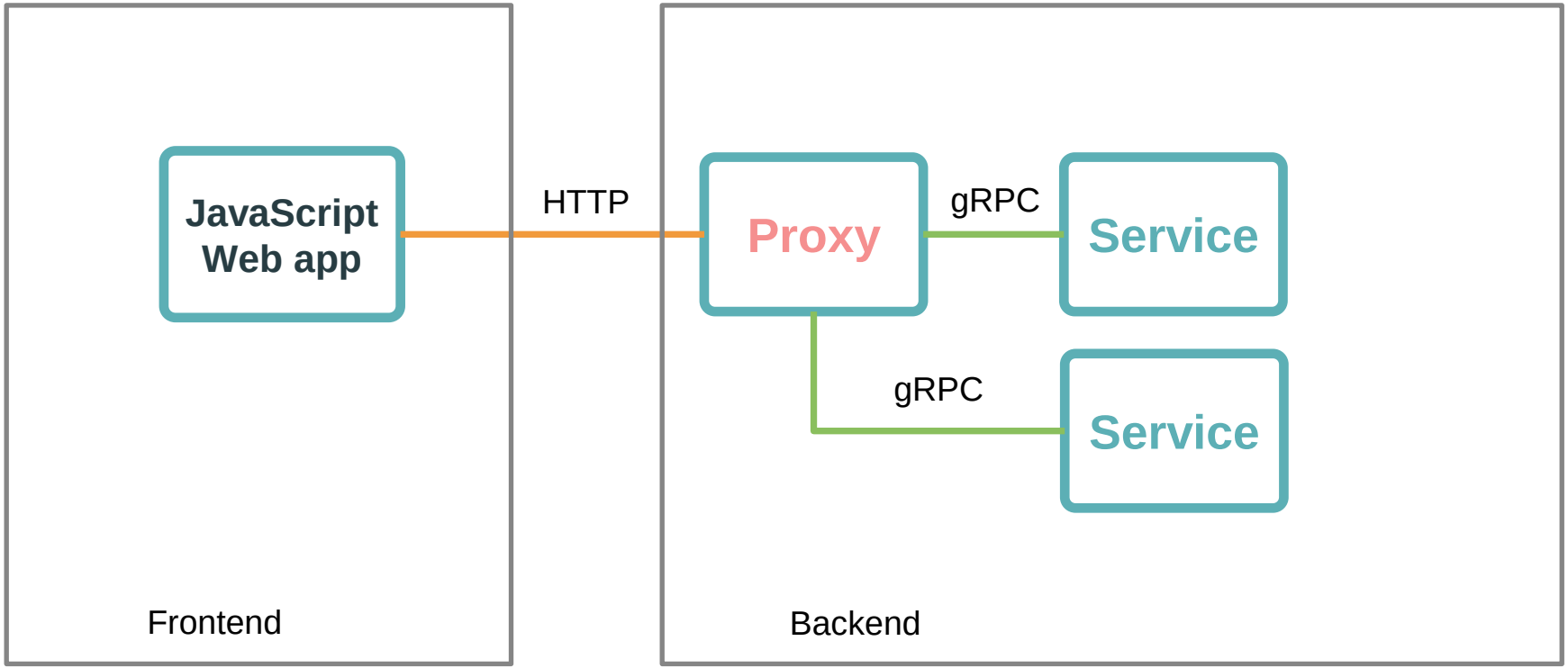
```
try
{
     //... ...
} catch (RpcException ex) when (ex.StatusCode == StatusCode.PermissionDenied)
{
    var userEntry = ex.Trailers.FirstOrDefault(e => e.Key == "User");
} catch (RpcException) { // Handle any other error type ... }
```

Взаимодействие из браузера

Взаимодействие из браузера



Взаимодействие из браузера

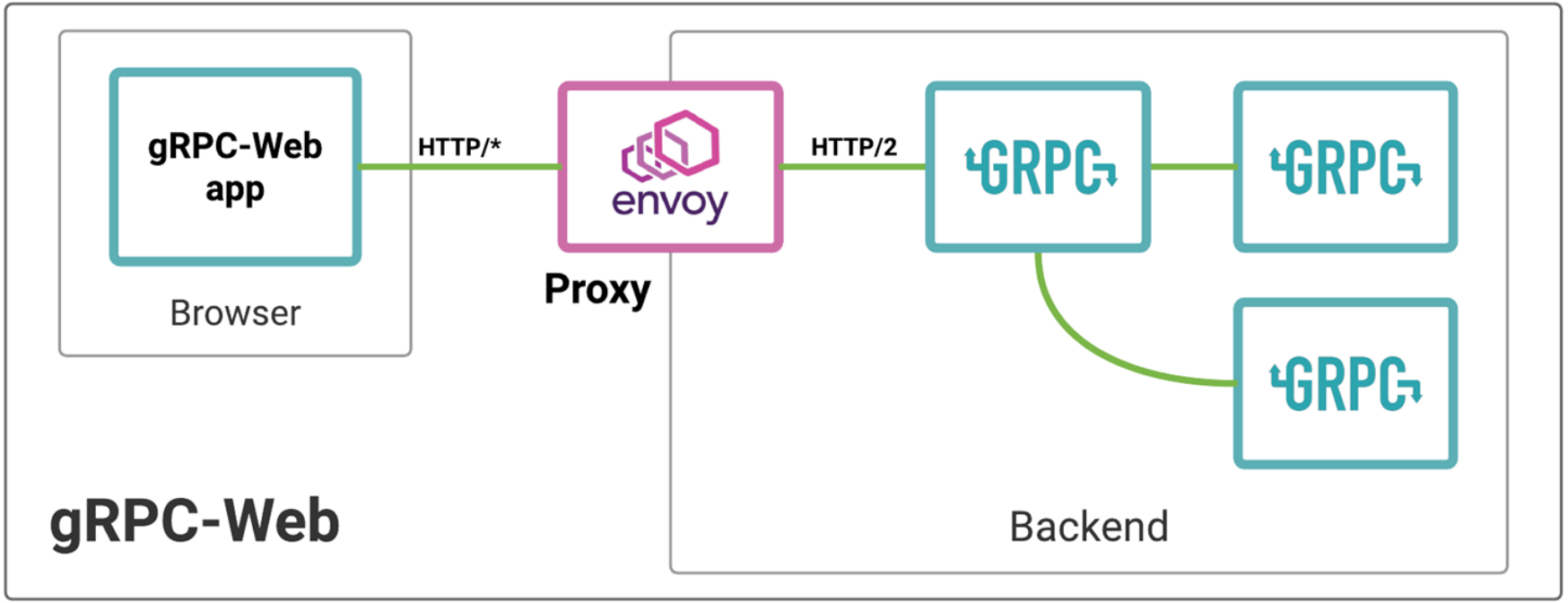


gRPC-Web Client

gRPC-Web Client

- Является прокси между браузером и сервисами
- Берет на себя задачи по сериализации/десериализации

Google: gRPC-Web Client



Google: gRPC-Web Client

- Запускается вручную
- Unary calls and Server-side streaming
- Работает с Deadline
- Production ready

Roadmap:

- Bidi Support
- Интеграция с React, Vue и Angular
- Web UI Support
- Использование разных прокси-клиентов (кроме **Envoy**)

Microsoft .NET: gRPC-Web Client

```
Install-Package Grpc.AspNetCore.Web
```

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseRouting();

    app.UseGrpcWeb();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapGrpcService<GreeterService>().EnableGrpcWeb();
    });
}
```

Microsoft .NET: gRPC-Web Client

Решение для Blazor WebAssembly:

- `Install-Package Grpc.Net.Client.Web`
- `Install-Package Grpc.Net.Client`

```
var handler = new GrpcWebHandler(GrpcWebMode.GrpcWebText, new HttpClientHandler());
var channel = GrpcChannel.ForAddress("https://localhost:5001", new GrpcChannelOptions
{
    HttpClient = new HttpClient(handler)
});
```

```
var client = new TicketerService.TicketerClient(channel);
var response = await client.GetTickets();
```

Microsoft .NET: gRPC-Web Client

- Клиентский код из JS приложения будет аналогичен решению от Google
- Быстро конфигурируется и запускается для Blazor WebAssembly
- Unary calls and Server-side streaming
- Может требовать настройки CORS

Кроссплатформенность

Особенности gRPC: Кроссплатформенность

GO

C/C++, PHP, C#,
Objective-C


.NET
Core
+
GRPC

gRPC C-core и .NET Core gRPC

- Вместо C-библиотеки для реализации http/2 использует нативную реализацию в .Net Core 3
- Вместо создания класса Server используется Kestrel
- Больше не используется GrpcEnvironment
- ILogger

.NET Core gRPC: Visual Studio

- Поддержка proto формата
- REST и gRPC
- Документация по миграции с grpc-c на .NET Core gRPC

Безопасность

Аутентификация

Call credentials

- JWT Bearer Token
- OAuth 2.0
- OpenID Connect
- Azure Active Directory
- IdentityServer
- WS-Federation

Channel credentials

- Сертификаты

Channel credentials

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureWebHostDefaults(webBuilder =>
        {
          var serverCert = ObtainServerCertificate();

          webBuilder.UseStartup<Startup>()
              .ConfigureKestrel(kestrelServerOptions => {

                    opt.ClientCertificateMode = ClientCertificateMode.RequireCertificate;

                    // Verify that client certificate was issued by same CA as server certificate
                    opt.ClientCertificateValidation = (certificate, chain, errors) =>
                        certificate.Issuer == serverCert.Issuer;

                  });
              });
        });
```

Channel credentials

```
var cert = ObtainServerCertificate();


var handler = new HttpClientHandler();
handler.ClientCertificates.Add(cert);
var httpClient = new HttpClient(handler);


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


var someServiceClient = new SomeService.SomeServiceClient(channel);
var response = await someServiceClient.SayHelloAsync(new HelloRequest { Name = “Hi“ }));
```

Call credentials: server

```
[Authorize]
public class TicketerService : Ticketer.TicketerBase, ITicketerService
{
    [Authorize]
    public override Task<BuyTicketsResponse> BuyTickets(BuyTicketsRequest request, ServerCallContext context)
    {
        var user = context.GetHttpContext().User;

        return Task.FromResult(new BuyTicketsResponse
        {
            Success = _ticketRepository.BuyTickets(user.Identity.Name, request.Count)
        });
    }
```

Call credentials: server

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddGrpc();

    var signingKey = ObtainSigningKeySomehow();

    services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
        .AddJwtBearer(options =>
        {
             options.TokenValidationParameters =
                new TokenValidationParameters
                {
                    ValidateLifetime = true,
                    IssuerSigningKey = signingKey
                };
        });
}
```

Call credentials: client

```
public async Task BuyTicketsAsync(string name)
{
    var headers = new Metadata
    {
        { "Authorization", $"Bearer {_userToken}" }
    };
    var request = new GetRequest
    {
        Time = DateTime.Now.AddDays(1),
        Name = name
    };
    var response = await _ticketsService.BuyTickets(request, headers);
}
```

Server reflection

Server reflection

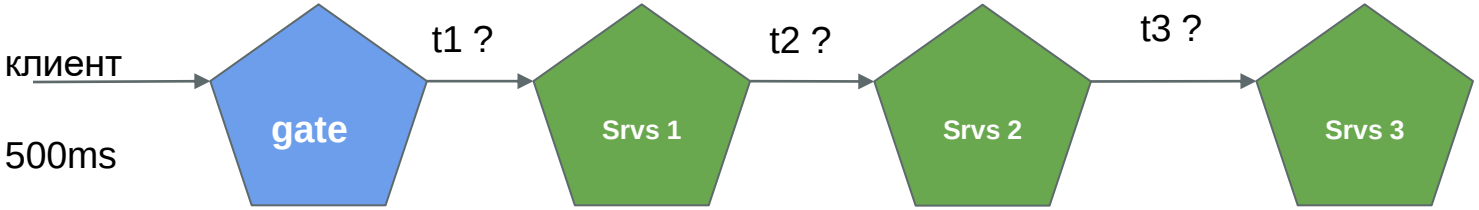
- Рантайм информация о сервисе
- Используется gRPC CLI для получения данных о сервере
- NuGet Grpc.Reflection
- Не позволяет автоматически генерировать proto-file

Server reflection

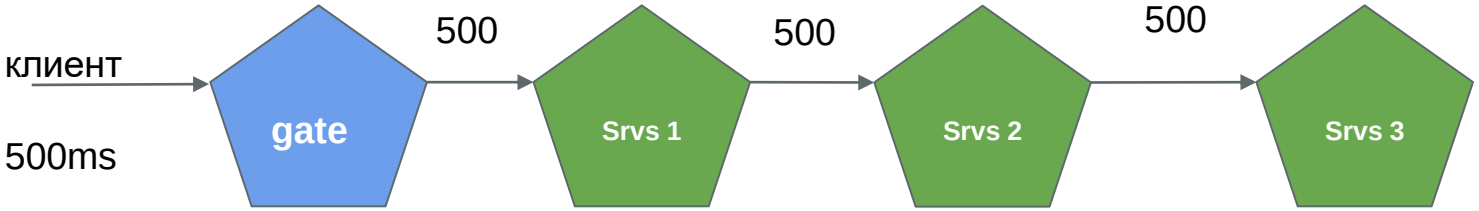
```
public void ConfigureServices(IServiceCollection services)
{
     services.AddGrpc();

     services.AddGrpcReflection();
}


public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapGrpcService<RealtyService>();
        endpoints.MapGrpcReflectionService();
    });
}
```

Deadline
&
ContextPropagationToken

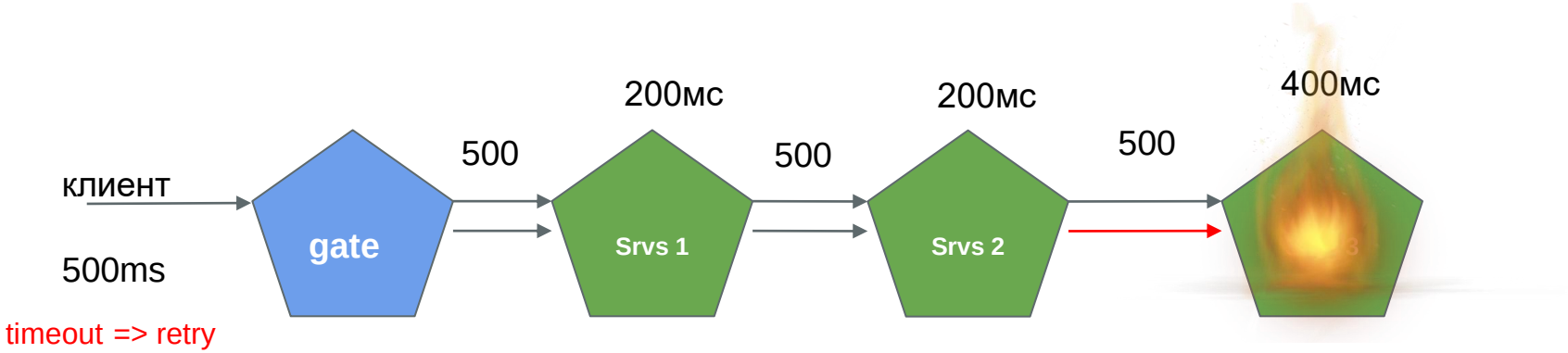
ContextPropagationToken



ContextPropagationToken



ContextPropagationToken



ContextPropagationToken

- Grpc.Net.ClientFactory

```
services
    .AddGrpcClient<Greeter.GreeterClient>(o =>
    {
        o.Address = new Uri("https://localhost:5001");
    })
    .EnableCallContextPropagation();
```

```
var channel = new Channel("localhost", ServerPort, ChannelCredentials.Insecure);
var client = new DowntownRealtyClient(channel);
client.GetRealtyList(new RealtyListRequest() { Type = RealtyType.House },
                         new CallOptions(deadline: DateTime.Now.AddSeconds(90)));
```

Немного кода

Summarize

- отличная альтернатива WCF
- облегченное межплатформенное взаимодействие
- хорошая скорость и гибкое взаимодействие клиент-сервера

- Не работает с IIS
- инструменты для тестирования

Патудин Иван

https://github.com/grem0087/gRpcNext