Тестовое задание .Net BackEnd Developer (Dec 2023)

Описание задания

Необходимо написать пакет генератора кода, который генерирует имплементацию сервиса Grpc.

На вход, разработчик создает:

1. Proto файл с описанием интерфейса:

```
service MyGrpcApi {
  rpc Method (MethodRequest request) returns (MethodResponse);
  rpc StreamMethod (MethodRequest request) returns (stream MethodResponse);
}
```

2. Класс реализации Grpc контроллера:

```
[GrpcMapping]
internal partial class MyAPIController : MyGrpcApi.MyGrpcApiBase
{
}
```

Для возможностей генерации имплементации этот класс должен иметь оформлен следующим образом:

1. Данный класс должен быть partial;
2. Данный класс должен быть с атрибутом GrpcMappingAttribute.

Кодогенератор должен для всех классов с атрибутом GrpcMappingAttribute, выполнять следующую логику:

1. Генерировать реализацию методов Grpc (смотри далее),
2. Проверить, что все методы из базового класса MyGrpcApi.MyGrpcApiBase реализованы в контроллере MyAPIController.

Interface mapping

Данный метод имплементации переводит реализацию метода в вызов интерфейса.
Указывается в какой интерфейс надо мапить метод:

```
[GrpcMapping]
internal partial class MyAPIController : MyGrpcApi.MyGrpcApiBase
{
  [GrpcInterface(typeof(ImplInterface), Method = nameof(ImplInterface.Method))]
  public override partial Task<MethodResponse> Method (MethodRequest request, ServerCallContext context);
}
```

Метод помечается атрибутом GrpcInterface с указанием:

1. Интерфейс который надо добавить в зависимости,
2. Method: метод для вызова. Если совпадает с именем метода в Grpc, то можно не указывать.

В результате должен генерироваться код:

```
partial class MyAPIController
{
    private readonly ImplInterface _serviceImplInterface;

    public MyAPIController (ImplInterface serviceImplInterface)
    {
        _serviceImplInterface = serviceImplInterface;
    }

    partial async Task<MethodResponse> Method(MethodRequest request, ServerCallContext context)
    {
        var result = await _serviceImplInterface.Method(
            request.Arg1,
            request.Arg2,
            ....
            context.CancellationToken);

        return new MethodResponse()
        {
            Response = result.Value
        };
    }
}
```

Данный код вызывает выполнение метода интерфейса и маппинг результата в выходное сообщение.

Stream mapping

Для маппинга в stream метод возвращает IAsyncEnumerable и мапят сообщения в stream. Методом необходимо указать какие сообщения в какие grpc messages преобразовывать.

```
[GrpcMapping]
internal partial class MyAPIController: MyGrpcApi.MyGrpcApiBase
{
    [GrpcInterface(typeof(ImplInterface),Method=nameof(ImplInterface.Method))]
    [GrpcStreamMapping(Notification=typeof(CommandNotification1),Message=typeof(MethodMessage1))]
    [GrpcStreamMapping(Notification=typeof(CommandNotification2),Message=typeof(MethodMessage2))]
    public override partial Task StreamMethod(MethodRequest request, IServerStreamWriter<MethodResponse> response,
ServerCallContext context);
}
```

В результате должен генерироваться код:

```
partial class MyAPIController
{
     private readonly ImplInterface _serviceImplInterface;

     public MyAPIController (ImplInterface serviceImplInterface)
     {
            _serviceImplInterface = serviceImplInterface;
     }

     partial async Task StreamMethod(MethodRequest request, IServerStreamWriter<MethodResponse> response, ServerCallContext
context)
     {
          await foreach(var notification in _serviceImplInterface.Method(
               request.Arg1,
               request.Arg2,
              ...
          context.CancellationToken))
          {
              switch(notification)
              {
                   case CommandNotification1 notification1:
                        await response.WriteAsync(new MethodMessage1(
                                                  notification1.Prop1,
                                                  notification1.Prop2,
                                                 ...
                                                 ));
                   break;

                   case CommandNotification2 notification2:
                       await response.WriteAsync(new MethodMessage2(
                                                  notification2.Prop1,
                                                  notification2.Prop2,
                                                 ...));
                   break;
                   default:
                       throw new InvalidCastException($"Unexpected notification type {notification.GetType().FullName}");
              }
          }
     }
}
```

Требования к решению:

1. Целевая платформа - минимум .NET 6
2. Сторонние библиотеки: можно использовать библиотеки с MIT лицензией если они носят вспомогательную роль.
3. Целевая операционная система: Windows 11

Критерии оценки:

1. Чистый код: грамотное распределение по проектам и классам, чистый стиль кодирования, понятные алгоритмы.
2. Хорошая архитектура: точки расширения, хорошая модель обработки, хорошая модель ограничений стратегии.
3. Хорошее понимание платформы и операционной системы: учет особенностей управления системными ресурсами.

Время на выполнение задания 4-6 часов.