Бесшовная интеграция микросервисов в стиле функционального API

Райффайзен

Спикер

Павел Московой
Email: pavel@moskovoi.ru

- Проектирование архитектуры и back-end приложений для страховых продуктов банка
- Разработка микросервисов на .NET Core 3.1 и C# 8
- Функциональный подход при написании кода
- Более 8 лет опыта профессиональной разработки на C#

Цели

- Реализовать бесшовную интеграцию между микросервисами на платформе .NET Core
- Предложить сообществу функциональный фреймворк для реализации HTTP-клиентов и бесшовной интеграции сервисов

Задача

- Есть сервис, предоставляющий набор CRUD-операций над продуктами.
- Необходимо написать другой сервис, который берет продукты из первого и обогащает их дополнительными данными.

Products

GET	/products
POST	/products
GET	/products/{id}
PUT	/products/{id}
DELETE	/products/{id}

```
Product {
    id                  string($uuid)
    name*               string
    description*        string
}
```

Решение с HttpClient

```
private readonly HttpClient httpClient;

public ProductsServiceClient(HttpClient httpClient)
    => this.httpClient = httpClient;

public async Task<IReadOnlyCollection<Product>> GetProductsAsync()
{
    // Получаем ответ от сервиса продуктов
    var resonseStream = await httpClient.GetStreamAsync(ProductsReositoryUrl);

    // Десериализуем, используя System.Text.Json.JsonSerializer
    return await DeserializeAsync<IReadOnlyCollection<Product>>(resonseStream);
}

public async Task<Product> GetProductAsync(Guid id)
{
    var responseStream = await httpClient.GetStreamAsync($"{ProductsReositoryUrl}/{id}");
    return await DeserializeAsync<Product>(responseStream);
}
```

Регистрация в DI

```
public void ConfigureServices(IServiceCollection services)
    =>
    services
    .AddControllers().Services
    .AddHttpClient<IProductsServiceClient, ProductsServiceClient>(
        client =>
        {
            client.BaseAddress = new Uri(ProductsServiceUrl);
            client.DefaultRequestHeaders.Accept.Add(
                new MediaTypeWithQualityHeaderValue("application/json"));
        })
    .Services
    .AddSingleton<IProductModelMapper, ProductModelMapper>();
```

В чём проблема?

```
[HttpGet]
[ProducesResponseType(StatusCodes.Status200OK)]
0 references
public async Task<ActionResult<ProductModel[]>> GetAsync()
    =>
    (await productsServiceClient.GetProductsAsync())
    .Select(
        product => productModelMapper.Map(product))
    .ToArray();

[HttpGet("{guid}")]
[ProducesResponseType(StatusCodes.Status200OK)]
0 references
public async Task<ActionResult<ProductModel>> GetAsync(Guid guid)
{
    // Если продукт не будет найден, здесь будет Exception
    var product = await productsServiceClient.GetProductAsync(guid);
    return productModelMapper.Map(product);
}
```

Через обработку исключений

```
[HttpGet("{guid}")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
0 references
public async Task<ActionResult<ProductModel>> GetAsync(Guid guid)
{
    try
    {
        var product = await productsServiceClient.GetProductAsync(guid);
        return productModelMapper.Map(product);
    }
    catch (ProductNotFoundException)
    {
        return new NotFoundResult();
    }
}
```

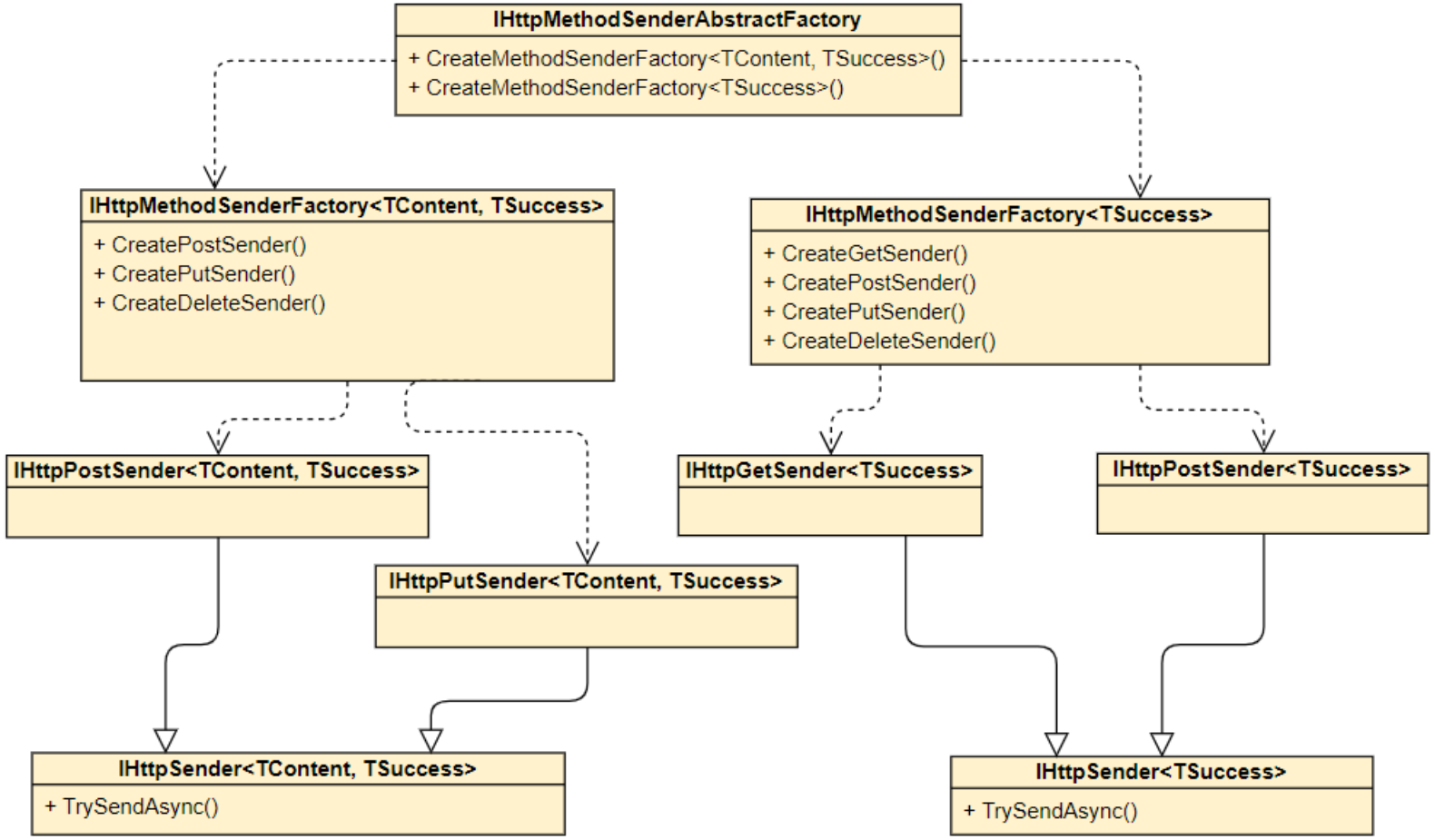
Интерфейс для работы с HTTP

```
public interface IHttpSender<TInputContent, TSuccessResult>
    where TInputContent : notnull
    where TSuccessResult : notnull
{
    5 references | ruamvp2, 61 days ago | 1 author, 2 changes
    public Task<IResult<TSuccessResult, IHttpMethodError>> TrySendAsync(
        string relativeUrl,
        TInputContent inputContent,
        CancellationToken cancellationToken = default)
        =>
        TrySendAsync(
            new HttpSenderRequest<TInputContent>(relativeUrl, inputContent),
            cancellationToken);

    ruamvp2, 61 days ago | 1 author, 1 change
    public Task<IResult<TSuccessResult, IHttpMethodError>> TrySendAsync(
        IHttpSenderRequest<TInputContent> request,
        CancellationToken cancellationToken = default);
}
```

Интерфейс для работы с HTTP

```
public interface IHttpSender<TSuccessResult>
    where TSuccessResult : notnull
{
    4 references | ruamvp2, 61 days ago | 1 author, 2 changes
    public Task<IResult<TSuccessResult, IHttpMethodError>> TrySendAsync(
        string relativeUrl,
        CancellationToken cancellationToken = default)
        =>
        TrySendAsync(new HttpSenderRequest(relativeUrl), cancellationToken);

    9 references | ruamvp2, 61 days ago | 1 author, 1 change
    public Task<IResult<TSuccessResult, IHttpMethodError>> TrySendAsync(
        IHttpSenderRequest request,
        CancellationToken cancellationToken = default);
}
```

Немного о реализации



Решение на нашем движке

```
private readonly IHttpGetSender<IReadOnlyCollection<Product>> productsGetSender;
private readonly IHttpGetSender<Product> productGetSender;
private readonly IHttpPostSender<Product, Product> productPostSender;
private readonly IHttpPutSender<Product, Product> productPutSender;
private readonly IHttpDeleteSender<Unit> productDeleteSender;

public ProductsServiceClient(in IHttpMethodSenderAbstractFactory abstractFactory)
{
    productsGetSender = abstractFactory
        .CreateMethodSenderFactory<IReadOnlyCollection<Product>>().CreateGetSender();
    productGetSender = abstractFactory
        .CreateMethodSenderFactory<Product>().CreateGetSender();
    productPostSender = abstractFactory
        .CreateMethodSenderFactory<Product, Product>().CreatePostSender();
    productPutSender = abstractFactory
        .CreateMethodSenderFactory<Product, Product>().CreatePutSender();
    productDeleteSender = abstractFactory
        .CreateMethodSenderFactory<Unit>().CreateDeleteSender();
}
```

Отправка запросов

```
private const string ProductsRepositoryUrl = "/products";

public async Task<IResult<IReadOnlyCollection<Product>, IHttpMethodError>> GetProductsAsync()
    => await productsGetSender.TrySendAsync(ProductsRepositoryUrl);

public async Task<IResult<Product, IHttpMethodError>> GetProductAsync(Guid id)
    => await productGetSender.TrySendAsync($"{ProductsRepositoryUrl}/{id}");

public async Task<IResult<Product, IHttpMethodError>> AddProductAsync(Product product)
    => await productPostSender.TrySendAsync(ProductsRepositoryUrl, product);

public async Task<IResult<Product, IHttpMethodError>> UpdateProductAsync(Product product)
    => await productPutSender.TrySendAsync($"{ProductsRepositoryUrl}/{product.Id}", product);

public async Task<IResult<Unit, IHttpMethodError>> DeleteProductAsync(Guid id)
    => await productDeleteSender.TrySendAsync($"{ProductsRepositoryUrl}/{id}");
```

Регистрация в DI

```
public void ConfigureServices(IServiceCollection services)
    =>
    services
    .AddControllers().Services
    .UseHttpClient(
        _ => new DefaultHttpClientConfiguration
        {
            BaseAddressUrl = ProductsServiceUrl,
            Timeout = TimeSpan.FromSeconds(5)
        })
    .UseRest()
    .MapService<IProductsServiceClient>(
        abstractFactory => new ProductsServiceClient(abstractFactory))
    .RegisterAsTransient()
    .AddSingleton<IProductModelMapper, ProductModelMapper>();
```

Регистрация в DI

```
public void ConfigureServices(IServiceCollection services)
    =>
    services
    .AddControllers().Services
    .UseHttpClient(
        _ => new DefaultHttpClientConfiguration
        {
            BaseAddressUrl = ProductsServiceUrl,
            Timeout = TimeSpan.FromSeconds(5)
        })
    .UseRest()
    .MapService<IProductsServiceClient>(
        abstractFactory => new ProductsServiceClient(abstractFactory))
    .RegisterAsTransient()
    .AddSingleton<IProductModelMapper, ProductModelMapper>();
```

Регистрация в DI

```
public void ConfigureServices(IServiceCollection services)
    =>
    services
    .AddControllers().Services
    .UseHttpClient(
        _ => new DefaultHttpClientConfiguration
        {
            BaseAddressUrl = ProductsServiceUrl,
            Timeout = TimeSpan.FromSeconds(5)
        })
    .UseRest()
    .MapService<IProductsServiceClient>(
        abstractFactory => new ProductsServiceClient(abstractFactory))
    .RegisterAsTransient()
    .AddSingleton<IProductModelMapper, ProductModelMapper>();
```

Регистрация в DI

```
public void ConfigureServices(IServiceCollection services)
    =>
    services
    .AddControllers().Services
    .UseHttpClient(
        _ => new DefaultHttpClientConfiguration
        {
            BaseAddressUrl = ProductsServiceUrl,
            Timeout = TimeSpan.FromSeconds(5)
        })
    .UseRest()
    .MapService<IProductsServiceClient>(
        abstractFactory => new ProductsServiceClient(abstractFactory))
    .RegisterAsTransient()
    .AddSingleton<IProductModelMapper, ProductModelMapper>();
```

Коллекция продуктов

```
[HttpGet]
[ProducesResponseType(StatusCodes.Status200OK)]
0 references
public async Task<ActionResult<ProductModel[]>> GetAsync()
    =>
    (await productsServiceClient.GetProductsAsync())
    .GetValueOrThrow()
    .Select(
        product => productModelMapper.Map(product))
    .ToArray();
```

Коллекция продуктов

```
[HttpGet]
[ProducesResponseType(StatusCodes.Status200OK)]
0 references
public async Task<ActionResult<ProductModel[]>> GetAsync()
    =>
    (await productsServiceClient.GetProductsAsync())
    .GetValueOrThrow()
    .Select(
        product => productModelMapper.Map(product))
    .ToArray();
```

Коллекция продуктов

```
[HttpGet]
[ProducesResponseType(StatusCodes.Status200OK)]
0 references
public async Task<ActionResult<ProductModel[]>> GetAsync()
    =>
    (await productsServiceClient.GetProductsAsync())
    .GetValueOrThrow()
    .Select(
        product => productModelMapper.Map(product))
    .ToArray();
```

Продукт по ИД

```
[HttpGet("{guid}")]
[ProducesResponseType(StatusCodes.Status200OK)]
0 references
public async Task<ActionResult<ProductModel>> GetAsync(Guid guid)
    =>
    (await productsServiceClient.GetProductAsync(guid))
    .MapValue(
        product => productModelMapper.Map(product))
    .Fold<ActionResult<ProductModel>>(
        onSuccess: productModel => productModel,
        onFailure: error => error.ErrorCode switch
        {
            HttpStatusCode.NotFound => ProductNotFound(),
            _ => throw new FailureException<IHttpMethodError>(error)
        });
```

Продукт по ИД

```
[HttpGet("{guid}")]
[ProducesResponseType(StatusCodes.Status200OK)]
0 references
public async Task<ActionResult<ProductModel>> GetAsync(Guid guid)
    =>
    (await productsServiceClient.GetProductAsync(guid))
    .MapValue(
        product => productModelMapper.Map(product))
    .Fold<ActionResult<ProductModel>>(
        onSuccess: productModel => productModel,
        onFailure: error => error.ErrorCode switch
        {
            HttpStatusCode.NotFound => ProductNotFound(),
            _ => throw new FailureException<IHttpMethodError>(error)
        });
```

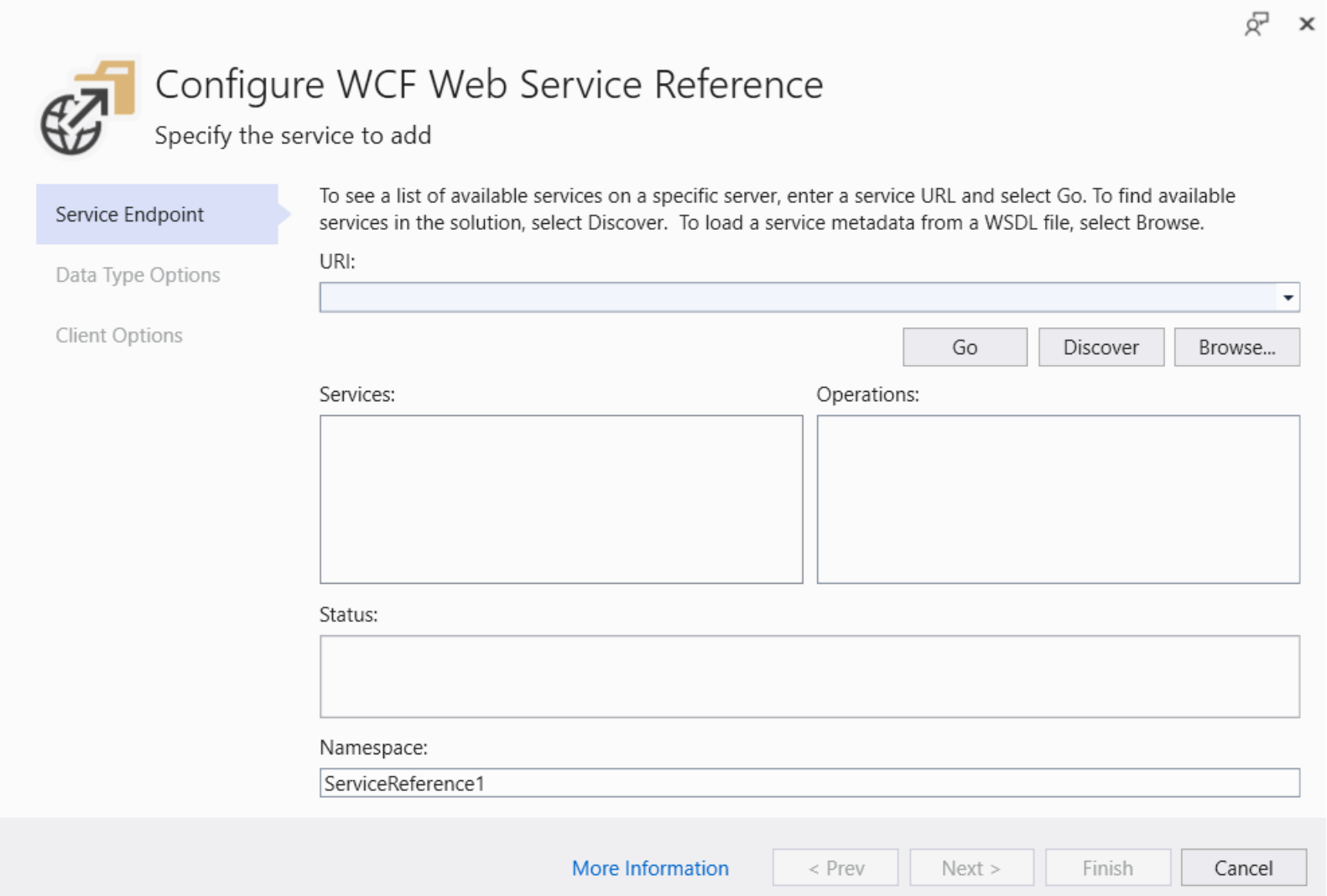
Продукт по ИД

```
[HttpGet("{guid}")]
[ProducesResponseType(StatusCodes.Status200OK)]
0 references
public async Task<ActionResult<ProductModel>> GetAsync(Guid guid)
    =>
    (await productsServiceClient.GetProductAsync(guid))
    .MapValue(
        product => productModelMapper.Map(product))
    .Fold<ActionResult<ProductModel>>(
        onSuccess: productModel => productModel,
        onFailure: error => error.ErrorCode switch
        {
            HttpStatusCode.NotFound => ProductNotFound(),
            _ => throw new FailureException<IHttpMethodError>(error)
        });
1 reference
private static NotFoundObjectResult ProductNotFound()
    => new NotFoundObjectResult(CreateNotFoundProblemDetails("Product was not found"));
```

Продукт по ИД

```
[HttpGet("{guid}")]
[ProducesResponseType(StatusCodes.Status200OK)]
0 references
public async Task<ActionResult<ProductModel>> GetAsync(Guid guid)
    =>
    (await productsServiceClient.GetProductAsync(guid))
    .MapValue(
        product => productModelMapper.Map(product))
    .Fold<ActionResult<ProductModel>>(
        onSuccess: productModel => productModel,
        onFailure: error => error.ErrorCode switch
        {
            HttpStatusCode.NotFound => ProductNotFound(),
            _ => throw new FailureException<IHttpMethodError>(error)
        });
```

Работа с SOAP



Автосгенерированный код

```
[System.CodeDom.Compiler.GeneratedCodeAttribute("Microsoft.Tools.ServiceModel.Svcutil", "2.0.1-preview-30514-0828")]
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.Xml.Serialization.XmlTypeAttribute(Namespace="http://www.raiffeisen.ru/types/SomeType/v1")]
0 references | 0 changes | 0 authors, 0 changes
public partial class ct_SomeType
{
    [System.Xml.Serialization.XmlElementAttribute(Order=0)]
    0 references | 0 changes | 0 authors, 0 changes
    public ct_TypedId accNumber
    {
        get
        {
            return this.accNumberField;
        }
        set
        {
            this.accNumberField = value;
        }
    }
```

WCF Web Service Reference Provider

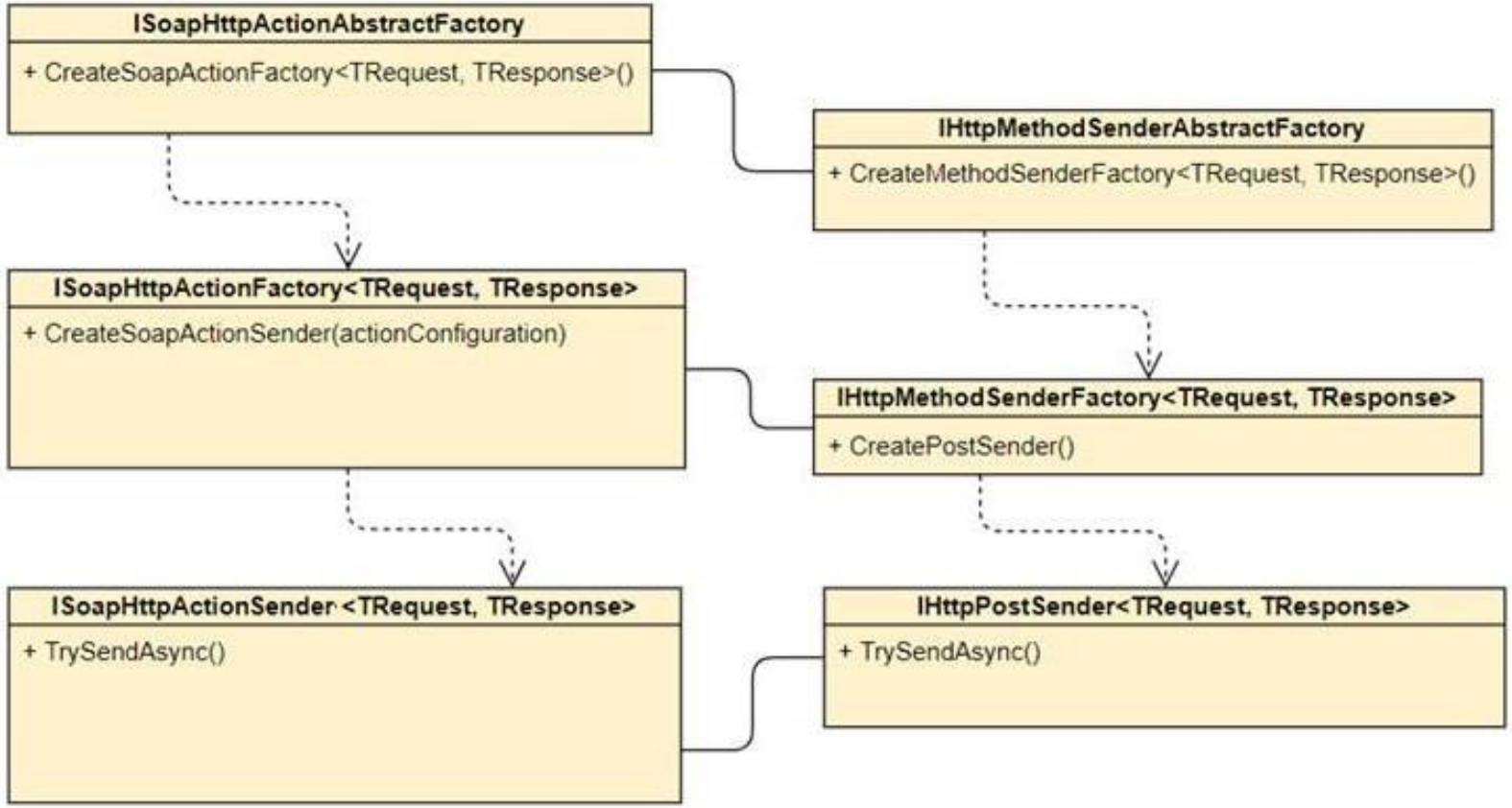
+ Создание сервис-клиента в удобном интерфейсе в несколько кликов.
+ Встроенная сериализация SOAP-сообщений.

- Автоматически созданный код может быть низкого качества
- Не соответствует nullable enable конвенции
- Разные подход для работы с REST и SOAP

Интерфейс для работы с SOAP

```
public interface ISoapHttpActionSender<TRequest, TResponse>
    where TRequest : notnull
    where TResponse : notnull
{

    Task<IResult<TResponse, IHttpMethodError>> TrySendAsync(
        TRequest request,
        CancellationToken token = default);
}
```

Немного о реализации



SOAP сервис-клиент

```
public sealed class ProductsServiceClient : IProductsServiceClient
{
    private readonly ISoapHttpActionSender<ProductRequest, ProductResponse> sender;

    public ProductsServiceClient(
        in ISoapHttpActionAbstractFactory abstractFactory)
        =>
        sender = abstractFactory
        .CreateSoapActionFactory<ProductRequest, ProductResponse>(
            new SoapHttpActionConfiguration
            {
                RelativeUrl = "/services/UpdateProduct"
            })
        .CreateSoapActionSender();

    public async Task<ProductResponse> UpdateProductAsync(ProductRequest request)
        => (await sender.TrySendAsync(request)).GetValueOrThrow();
}
```

SOAP сервис-клиент

```
public sealed class ProductsServiceClient : IProductsServiceClient
{
    private readonly ISoapHttpActionSender<ProductRequest, ProductResponse> sender;

    public ProductsServiceClient(
        in ISoapHttpActionAbstractFactory abstractFactory)
        =>
        sender = abstractFactory
        .CreateSoapActionFactory<ProductRequest, ProductResponse>(
            new SoapHttpActionConfiguration
            {
                RelativeUrl = "/services/UpdateProduct"
            })
        .CreateSoapActionSender();

    public async Task<ProductResponse> UpdateProductAsync(ProductRequest request)
        => (await sender.TrySendAsync(request)).GetValueOrThrow();
}
```

SOAP сервис-клиент

```
public sealed class ProductsServiceClient : IProductsServiceClient
{
    private readonly ISoapHttpActionSender<ProductRequest, ProductResponse> sender;

    public ProductsServiceClient(
        in ISoapHttpActionAbstractFactory abstractFactory)
        =>
        sender = abstractFactory
        .CreateSoapActionFactory<ProductRequest, ProductResponse>(
            new SoapHttpActionConfiguration
            {
                RelativeUrl = "/services/UpdateProduct"
            })
        .CreateSoapActionSender();

    public async Task<ProductResponse> UpdateProductAsync(ProductRequest request)
        => (await sender.TrySendAsync(request)).GetValueOrThrow();
}
```

Регистрация в DI

```
public void ConfigureServices(IServiceCollection services)
    =>
    services
    .AddControllers().Services
    .UseHttpClient(
        _ => new DefaultHttpClientConfiguration
        {
            BaseAddressUrl = ProductsServiceUrl,
            Timeout = TimeSpan.FromSeconds(30)
        })
    .UseSoapActions()
    .MapService<IProductsServiceClient>(
        abstractFactory => new ProductsServiceClient(abstractFactory))
    .RegisterAsTransient();
```

Регистрация в DI

```
public void ConfigureServices(IServiceCollection services)
    =>
    services
    .AddControllers().Services
    .UseHttpClient(
        _ => new DefaultHttpClientConfiguration
        {
            BaseAddressUrl = ProductsServiceUrl,
            Timeout = TimeSpan.FromSeconds(30)
        })
    .UseSoapActions()
    .MapService<IProductsServiceClient>(
        abstractFactory => new ProductsServiceClient(abstractFactory))
    .RegisterAsTransient();
```

Регистрация в DI

```
public void ConfigureServices(IServiceCollection services)
    =>
    services
    .AddControllers().Services
    .UseHttpClient(
        _ => new DefaultHttpClientConfiguration
        {
            BaseAddressUrl = ProductsServiceUrl,
            Timeout = TimeSpan.FromSeconds(30)
        })
    .UseSoapActions()
    .MapService<IProductsServiceClient>(
        abstractFactory => new ProductsServiceClient(abstractFactory))
    .RegisterAsTransient();
```

Регистрация в DI

```
public void ConfigureServices(IServiceCollection services)
    =>
    services
    .AddControllers().Services
    .UseHttpClient(
        _ => new DefaultHttpClientConfiguration
        {
            BaseAddressUrl = ProductsServiceUrl,
            Timeout = TimeSpan.FromSeconds(30)
        })
    .UseSoapActions()
    .MapService<IProductsServiceClient>(
        abstractFactory => new ProductsServiceClient(abstractFactory))
    .RegisterAsTransient();
```

Что в итоге?

- Гибкий flow для работы в функциональном стиле над результатом ответа на HTTP-запрос
- Возможность разделения неуспешных ответов от исключительных ситуаций
- Возможность работы с SOAP однотипно с REST
- Решение на основе HttpClient
- Встроенная сериализация, построенная на базе JsonSerializer (для REST)
- Подлинно функциональный механизм регистрации в DI

Спасибо
за внимание!

info@raiffeisen.ru
raiffeisen.ru

Райффайзен