Hydrogen Sessions

#12 – Service Client

ServiceClient<T>

Esta é uma das classes abstratas mais interessantes do Hydrogen e assegura praticamente todas as funcionalidades das client libraries dos micro serviços (ou de qualquer outra Web API REST, mesmo que não seja produzida com o SDK Lithium).

Vale a pena explicá-la em detalhe porque permite aprofundar alguns tópicos referidos antes (as classes base por exemplo), perceber um pouco melhor o funcionamento das client libraries em geral e compreender alguns detalhes da ligação das aplicações ao Identity Server (OAuth). No processo serão visíveis várias boas práticas de programação.

O tema das client libraries é polémico. Há quem defenda que não devem existir porque contrariam o espírito dos micro serviços (loosely coupling). Há quem defenda que são úteis para assegurar comportamentos repetitivos (cross-cutting concerns).

No Lithium segue-se a segunda linha de pensamento porque:

- Pretendemos acelarar o desenvolvimento dos micro serviços (o developer não tem que programar uma série de coisas em cada serviço).
- Tiramos partido da geração de código (e dos objetivos inerentes).
- Pretendemos uniformizar o mais possível o comportamento dos serviços e a sua programação (afinal a PRIMAVERA é uma fábrica de software, onde não se faz artesanato).

Portanto, todos os micro serviços devem, em princípio, ter uma client lib e essa client lib C# deve ser sempre, em princípio, usada para comunicar com os serviços. Claro que haverá exceções (serviços que não precisam de ter client lib ou casos em que será necessário comunicar diretamente com a API via HTTP). São exceções!

Hoje as client libs suportam apenas REST e C#. Um dia destes podem estar disponíveis versões JavaScript ou gRPC.

O ServiceClient serve exclusivamente para o caso atual. REST e C#.

ServiceClient<T>

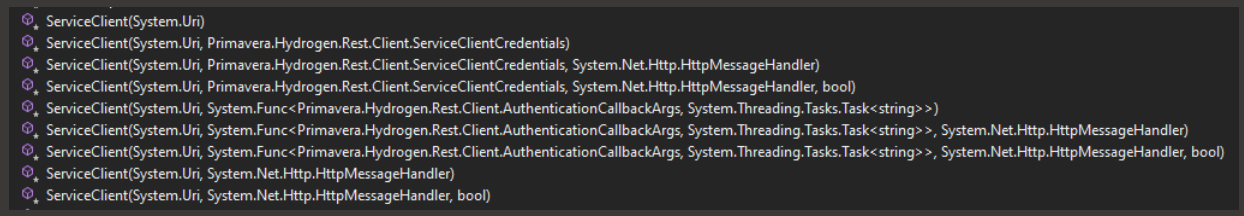
Esta classe está disponível em Primavera.Hydrogen.Rest.Client (runtime para clientes de API REST).

Eis a API:

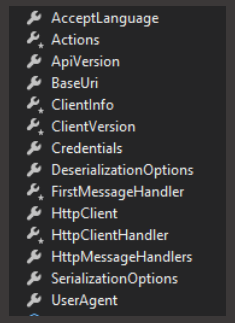
Definição da classe:

```
/// <summary>
/// Provides the base class for a service client.
/// A service client is a type that is used to communicate with a REST service.
/// </summary>
/// <typeparam name="T">The type of the concrete service client.</typeparam>
/// <seealso cref="IDisposable" />
93 references | Hugo Ribeiro, 41 days ago | 2 authors, 5 changes | 1 incoming change
public abstract partial class ServiceClient<T> : IDisposable
    where T : ServiceClient<T>
```

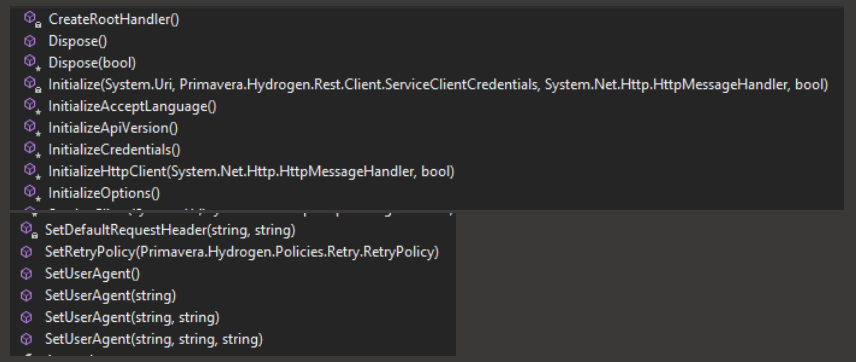
Construtores:



Propriedades:



Métodos:



IDisposable (HttpClient/HttpMessageHandler)

O ServiceClient é disposable, para controlar o lifetime do HttpMessageHandler usado para comunicar com a API (via HttpClient).

```
protected virtual void InitializeHttpClient(HttpMessageHandler handler, bool disposeHandler)
{
    // Set the HTTP message handler

    this.HttpClientHandler = handler;

    // Set the retry delegating handler

    RetryDelegatingHandler retryHandler = new RetryDelegatingHandler
    {
        InnerHandler = this.HttpClientHandler
    };

    // Create the HTTP client

    this.HttpClient = new HttpClient(retryHandler, disposeHandler);

    // Set the first message handler

    this.FirstMessageHandler = retryHandler;

    // Set user agent

    this.SetUserAgent();
}
```

```
protected virtual void Dispose(bool disposing)
{
    // Already disposed?

    if (this.disposed)
    {
        return;
    }

    // Called from Dispose()?

    if (disposing)
    {
        // Dispose HTTP client?

        if (this.HttpClient != null)
        {
            this.HttpClient.Dispose();
            this.HttpClient = null;
        }

        // Set to null other references

        this.HttpClientHandler = null;
        this.FirstMessageHandler = null;
    }

    // Set flag

    this.disposed = true;
}
```

```
string accessToken = (...)
using (MyServiceClient client = new MyServiceClient(baseUri, new AccessTokenCredentials(accessToken))
{
    // (...)
}
```

Este comportamento está relacionado com o tema dos construtores (ver adiante) e com a gestão do HttpClient (ver IHttpClientFactory).

Construtores (credenciais)

Por defeito, existem 8 construtores que permitem definir:

- O endereço base do serviço (baseUri).
- As credenciais (credentials) ou um callback para obter as credenciais (callback).
- O HttpMessageHandler que deve ser utilizado (handler). Se não for indicado um, será criado um quando necessário.
- Um valor para determinar se o handler deve ser disposed ou não no dispose do ServiceClient (disposeHandler).

Estes dois últimos parâmetros permitem assim, controlar o HttpMessageHandler utilizado, assim com o seu tempo de vida. O que permite:

- A utilização de IHttpMessageHandlerFactory.
- A realização de testes sobre as client libs (com mocks sobre o ServiceClient e/ou sobre o handler). Há inúmeros exemplos disso nos testes do Hydrogen.

Funcionalidades fundamentais

Esta classe abstrata implementa (e uniformiza) as seguintes funcionalidades:

- A autorização OAuth.
- Política de retry (RetryPolicy).
- Localização das respostas (AcceptLanguage).
- A invocação correta dos vários tipos de operação da API (ServiceClientActions).
- A serialização dos pedidos e a desserialização das respostas.
- O tratamento dos resultados.
- O tratamento de erros.
- O versionamento da API.
- A definição do user-agent (usado para identificar os pedidos recebidos na API)
- O tratamentos de outros headers nos pedidos e nas respostas.
- Etc.

Vejamos a maior parte dessas funcionalidades nos slides seguintes.

RetryPolicy

A invocação de endpoints HTTP está sujeita a erros transientes a que as aplicações podem reagir através de uma política de retry definida.

O ServiceClient implementa uma política por defeito:

```
protected virtual void InitializeHttpClient(HttpMessageHandler handler, bool disposeHandler)
{
    // Set the HTTP message handler

    this.HttpClientHandler = handler;

    // Set the retry delegating handler

    RetryDelegatingHandler retryHandler = new RetryDelegatingHandler
    {
        InnerHandler = this.HttpClientHandler
    };

    // Create the HTTP client

    this.HttpClient = new HttpClient(retryHandler, disposeHandler);

    // Set the first message handler

    this.FirstMessageHandler = retryHandler;

    // Set user agent

    this.SetUserAgent();
}
```

```
private static RetryPolicy GetDefaultRetryPolicy()
{
    ExponentialBackoffRetryStrategy retryStrategy = new ExponentialBackoffRetryStrategy(
        RetryDelegatingHandler.DefaultNumberOfAttempts,
        RetryDelegatingHandler.DefaultMinBackoff,
        RetryDelegatingHandler.DefaultMaxBackoff,
        RetryDelegatingHandler.DefaultBackoffDelta);

    return new RetryPolicy<HttpStatusCodeErrorDetectionStrategy>(retryStrategy);
}
```

```
public partial class HttpStatusCodeErrorDetectionStrategy : ITransientErrorDetectionStrategy
{
    #region Fields

    private readonly HttpStatusCode[] acceptedStatusCodes = new HttpStatusCode[]
    {
        HttpStatusCode.RequestTimeout,
        HttpStatusCode.BadGateway,
        HttpStatusCode.ServiceUnavailable,
        HttpStatusCode.GatewayTimeout
    };

    #endregion

    #region Public Methods

    /// <inheritdoc />
    27 references | 6/6 passing | Hugo Lourenço | 1 author, 1 change | 1 incoming change
    public bool IsTransient(Exception ex)
    {
        if (ex != null)
        {
            if (ex is HttpRequestWithStatusCodeException httpException)
            {
                return this.acceptedStatusCodes.Any(c => c == httpException.StatusCode);
            }
        }

        return false;
    }

    #endregion
}
```

Mas também permite que o “cliente” a altere:

```
public virtual void SetRetryPolicy(RetryPolicy retryPolicy)
{
    // If there is no retry policy specified, set a kind of null policy

    if (retryPolicy == null)
    {
        retryPolicy = RetryPolicy<IgnoreErrorDetectionStrategy>.FixedInterval(0);
    }

    // Get the current delegating handler

    RetryDelegatingHandler retryHandler = this.HttpMessageHandlers.OfType<RetryDelegatingHandler>().FirstOrDefault();
    if (retryHandler != null)
    {
        // Set the new policy

        retryHandler.RetryPolicy = retryPolicy;

        // Return

        return;
    }
```

AcceptLanguage

Esta propriedade é utilizada para definir o header Accept-Language em todos os pedidos à API, que depois o trata corretamente (ver https://github.com/PrimaveraDeveloper/lithium/blob/master/capabilities/localization.md).

```
/// <summary>
/// Gets or sets the preferred language for the response.
/// </summary>
13 references | 10/10 passing | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
public virtual string AcceptLanguage
{
    get
    {
        return this.fieldAcceptLanguage;
    }

    set
    {
        if (!this.fieldAcceptLanguage.EqualsNoCase(value))
        {
            this.fieldAcceptLanguage = value;
            this.SetDefaultRequestHeader(AcceptLanguageHeaderName, this.fieldAcceptLanguage);
        }
    }
}
```

```
private void SetDefaultRequestHeader(string name, string value)
{
    // Remove

    if (this.HttpClient.DefaultRequestHeaders.Contains(name))
    {
        this.HttpClient.DefaultRequestHeaders.Remove(name);
    }

    // Add if not null

    if (value != null)
    {
        this.HttpClient.DefaultRequestHeaders.TryAddWithoutValidation(name, value);
    }
}
```

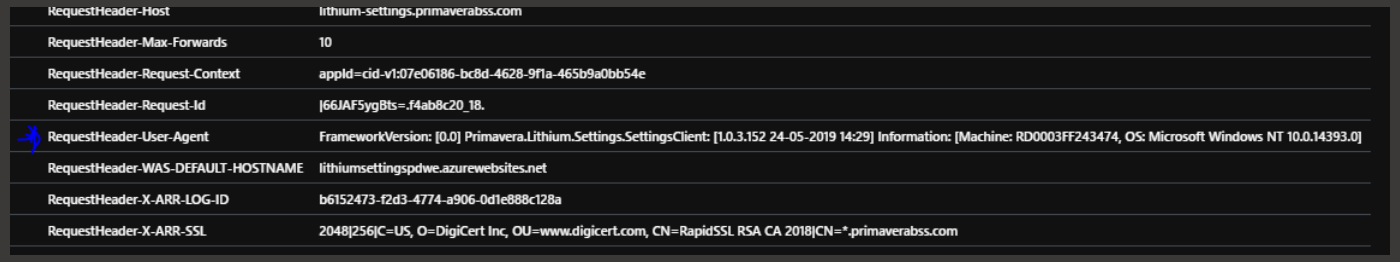
```
using FlatNoAuthServiceClient client = this.CreateFlatServiceClient();
client.AcceptLanguage = "es";
```

UserAgent

O user agente define o header User-Agent, enviado em todos os pedidos, e inclui:

- O nome da aplicação.
- A versão da aplicação.
- Informação adicional.

Esta informação é útil, na API, para identificar a origem dos pedidos, por exemplo, no AppInsights:



RequestHeader-Host	lithium-settings.primaverabss.com
RequestHeader-Max-Forwards	10
RequestHeader-Request-Context	appId=cid-v1:07e06186-bc8d-4628-9f1a-465b9a0bb54e
RequestHeader-Request-Id	|66JAF5ygBts=.f4ab8c20_18.
RequestHeader-User-Agent	FrameworkVersion: [0.0] Primavera.Lithium.Settings.SettingsClient: [1.0.3.152 24-05-2019 14:29] Information: [Machine: RD0003FF243474, OS: Microsoft Windows NT 10.0.14393.0]
RequestHeader-WAS-DEFAULT-HOSTNAME	lithiumsettingspdwe.azurewebsites.net
RequestHeader-X-ARR-LOG-ID	b6152473-f2d3-4774-a906-0d1e888c128a
RequestHeader-X-ARR-SSL	2048|256|C=US, O=DigiCert Inc, OU=www.digicert.com, CN=RapidSSL RSA CA 2018|CN=*.primaverabss.com

```
public bool SetUserAgent(string productName, string version, string info)
{
    // Validation

    SmartGuard.NotNullOrEmpty(() => productName, productName);
    SmartGuard.NotNullOrEmpty(() => version, version);
    SmartGuard.NotNullOrEmpty(() => info, info);

    // Disposed?

    if (this.disposed || this.HttpClient == null)
    {
        return false;
    }

    // Value

    string value = "{0}: [{1}]".Format(productName, version);
    value = string.Join(" ", value, "{0}: [{1}]".Format(InformationName, info));

    // Set user agent header

    this.SetDefaultRequestHeader(UserAgentHeaderName, value);

    // Result OK

    return true;
}
```

ServiceClientActions

Esta é a funcionalidade mais importante da client lib, para além da autorização. É aquela que uniformiza o comportamento dos pedidos de acordo com o seu tipo.

A cada ServiceClient<T> está associada uma instância de ServiceClientActions<T>, que disponibiliza os métodos adequados para chamar os endpoints da API de acordo com o tipo, dependendo se aceita parâmetros ou não, ou se devolve resultados ou não.

```
protected virtual ServiceClientActions<T> Actions
{
    get;
    set;
}
```

```
DeserializeResponseAsync(System.Net.Http.HttpRequestMessage, System.Net.Http.HttpResponseMessage)
DeserializeResponseAsync<TResult>(System.Net.Http.HttpRequestMessage, System.Net.Http.HttpResponseMessage, System.Threading.CancellationToken)
DeserializeResponseAsync<TResult>(System.Net.HttpStatusCode, System.Net.Http.HttpRequestMessage, System.Net.Http.HttpResponseMessage, System.Threading.CancellationToken)
ExecuteDeleteNoResultAsync(System.Uri, System.Net.HttpStatusCode, System.Threading.CancellationToken)
ExecuteDeleteNoResultAsync<TRequestBody>(System.Uri, TRequestBody, System.Net.HttpStatusCode, System.Threading.CancellationToken)
ExecuteGetAsync<TResult>(System.Uri, System.Net.HttpStatusCode, System.Threading.CancellationToken)
ExecuteGetNoResultAsync(System.Uri, System.Net.HttpStatusCode, System.Threading.CancellationToken)
ExecutePostAsync<TRequestBody, TResult>(System.Uri, TRequestBody, System.Net.HttpStatusCode, System.Threading.CancellationToken)
ExecutePostAsync<TResult>(System.Uri, System.Net.HttpStatusCode, System.Threading.CancellationToken)
ExecutePostNoResultAsync(System.Uri, System.Net.HttpStatusCode, System.Threading.CancellationToken)
ExecutePostNoResultAsync<TRequestBody>(System.Uri, TRequestBody, System.Net.HttpStatusCode, System.Threading.CancellationToken)
ExecutePutAsync<TRequestBody, TResult>(System.Uri, TRequestBody, System.Net.HttpStatusCode, System.Threading.CancellationToken)
ExecutePutAsync<TResult>(System.Uri, System.Net.HttpStatusCode, System.Threading.CancellationToken)
ExecutePutNoResultAsync(System.Uri, System.Net.HttpStatusCode, System.Threading.CancellationToken)
ExecutePutNoResultAsync<TRequestBody>(System.Uri, TRequestBody, System.Net.HttpStatusCode, System.Threading.CancellationToken)
HandleRequestErrorAsync(System.Net.Http.HttpRequestMessage, System.Net.Http.HttpResponseMessage, System.Threading.CancellationToken)
SendRequestAsync(System.Net.Http.HttpRequestMessage, System.Threading.CancellationToken)
ServiceClientActions(Primavera.Hydrogen.Rest.Client.ServiceClient<T>)
ServiceClient
```

Repare-se como os métodos estão relacionados com os métodos HTTP (GET, POST, etc.), os parâmetros e os resultados.

ServiceClientActions

Analisemos um desses métodos para compreender a lógica associada à invocação da API.

Essa lógica é basicamente a seguinte:

```
11 references | Hugo Ribeiro, 119 days ago | 2 authors, 2 changes | 1 incoming change
public async Task<ServiceOperationResult<TResult>> ExecutePostAsync<TRequestBody, TResult>(Uri requestUri, TRequestBody body, HttpStatusCode expectedStatusCode, CancellationToken cancellationToken)
{
    // Validation

    SmartGuard.NotNull(() => requestUri, requestUri);
    SmartGuard.NotNull(() => body, body);

    // Check the cancellation token

    cancellationToken.ThrowIfCancellationRequested();

    // Build the request

    using (HttpRequestMessage request = new HttpRequestMessage(HttpMethod.Post, requestUri)
        .AddJsonContent(body, this.ServiceClient.SerializationOptions))
    {
        // Send the request

        using (HttpResponseMessage response = await this.SendRequestAsync(request, cancellationToken).ConfigureAwait(false))
        {
            // Check the cancellation token

            cancellationToken.ThrowIfCancellationRequested();

            // Error?

            if (response.StatusCode != expectedStatusCode)
            {
                await this.HandleRequestErrorAsync(request, response, cancellationToken).ConfigureAwait(false);
            }

            // Deserialize response

            (string, TResult) deserialized = await this.DeserializeResponseAsync<TResult>(expectedStatusCode, request, response, cancellationToken).ConfigureAwait(false);

            // Result

            return new ServiceOperationResult<TResult>(request, response, deserialized.Item1, deserialized.Item2);
        }
    }
}
```

Preparação e serialização do pedido

Envio do pedido

Tratamento de erros

Desseriazação da resposta (sucesso ou erro)

Resultado

AddJsonContent

A serialização do body do pedido é realizada com System.Text.Json.JsonSerializer (não NewtonSoft!!!):

```
14 references | 5/5 passing | Hugo Lourenço | 1 author, 1 change | 1 incoming change
internal static HttpRequestMessage AddJsonContent(this HttpRequestMessage request, object value, JsonSerializerOptions options)
{
    // Validation

    SmartGuard.NotNull(() => request, request);
    SmartGuard.NotNull(() => value, value);
    SmartGuard.NotNull(() => options, options);

    // Serialize the value

    string payload = JsonSerializer.Serialize(value, options);

    // Add the content to the request

    request.Content = new StringContent(payload, Encoding.UTF8);

    // Set the content type

    request.Content.Headers.ContentType = MediaTypeHeaderValue.Parse("application/json; charset=utf-8");

    // Result

    return request;
}
```

SendRequestAsync

Este método é crucial porque é neste momento que se dá o processo de autorização (quando necessário).

```
12 references | Hugo Ribeiro, 119 days ago | 2 authors, 2 changes | 1 incoming change
public async Task<HttpResponseMessage> SendRequestAsync(HttpRequestMessage request, CancellationToken cancellationToken)
{
    // Validation

    SmartGuard.NotNull(() => request, request);

    // Apply credentials on the request

    if (this.ServiceClient.Credentials != null)
    {
        // Check cancellation token

        cancellationToken.ThrowIfCancellationRequested();

        // Invoke credentials method

        await this.ServiceClient.Credentials.HandleRequestAsync(request, cancellationToken).ConfigureAwait(false);
    }

    // Create a clone of the request message (cannot send the same message twice)
    // Message clone must occur before first "SendAsync" invocation because "SendAsync" disposes the request body.
    // Fix was made to .Net Core https://github.com/dotnet/corefx/pull/19082 but not to .Net Standard

    using (HttpRequestMessage requestClone = await request.CloneAsync().ConfigureAwait(false))
    {
        // Send the request
        // Cannot dispose response here because it is the result and it is used outside

        HttpResponseMessage response = await this.ServiceClient.HttpClient.SendAsync(request, cancellationToken).ConfigureAwait(false);

        // Apply credentials response handling

        bool retry = false;

        if (this.ServiceClient.Credentials != null)
        {
            // Check cancellation token

            cancellationToken.ThrowIfCancellationRequested();

            // Invoke credentials methods

            retry = await this.ServiceClient.Credentials.HandleResponseAsync(requestClone, response, cancellationToken).ConfigureAwait(false);
        }

        // Should retry?

        if (!retry)
        {
            return response;
        }

        // Send the request again

        return await this.ServiceClient.HttpClient.SendAsync(requestClone, cancellationToken).ConfigureAwait(false);
    }
}
```

Quando as credencias existem, o pedido é preparado para autorização pela instância das credenciais (ver adiante).

Envio do pedido

Tratamento da resposta na perspetiva da autorização (para os casos é que é necessário voltar a tentar) (ver adiante).

Resultado ou retry

HandleRequestErrorAsync

Este método trata os pedidos que devolvem códigos de erro/inesperados (ex.: BadRequest, NotFound). Espera-se que venha sempre no body da resposta uma instância de ServiceError (ver Hydrogen.Rest).

```
12 references | Hugo Ribeiro, 34 days ago | 2 authors, 3 changes | 1 incoming change
public async Task HandleRequestErrorAsync(HttpRequestMessage request, HttpResponseMessage response, CancellationToken cancellationToken)
{
    // Validation

    SmartGuard.NotNull(() => request, request);
    SmartGuard.NotNull(() => response, response);

    // Read the response content

    ServiceError error = null;
    string responseContent = null;

    try
    {
        // Make sure the content is defined

        if (response.Content != null)
        {
            responseContent = await response.Content.ReadAsStringAsync().ConfigureAwait(false);

            // Check the cancellation token

            cancellationToken.ThrowIfCancellationRequested();

            // Handle the response content

            if (!string.IsNullOrEmpty(responseContent))
            {
                // Deserialize the service error

                // NOTE:
                // When the body received is not a ServiceError instance, the result of serialization will be UNSPECIFIED
                // This should be ignored.

                error = JsonSerializer.Deserialize<ServiceError>(
                    responseContent,
                    this.ServiceClient.DeserializationOptions);

                if (error != null && error.IsUnspecified)
                {
                    // Is it an error returning by the API versioning middleware?

                    ServiceApiVersioningError apiVersioningError = JsonSerializer.Deserialize<ServiceApiVersioningError>(
                        responseContent,
                        this.ServiceClient.DeserializationOptions);

                    if (apiVersioningError != null && apiVersioningError.IsDefined)
                    {
                        // Translate to a service error

                        error = new ServiceError(
                            ServiceErrorCodes.ApiVersioningFailure,
                            Properties.Resources.RES_Error_ApiVersioningFailure);

                        error.Details.Add(
                            new ServiceErrorDetail(
                                apiVersioningError.Error.Code,
                                apiVersioningError.Error.Message));
                    }
                    else
                    {
                        error = null;
                    }
                }
            }
        }
    }
    catch (JsonException)
    {
        // Ignore, the body is not JSON
    }

    // Check the cancellation token

    cancellationToken.ThrowIfCancellationRequested();

    // Initialize exception

    ServiceException ex;

    if (error == null)
    {
        if (string.IsNullOrEmpty(responseContent))
        {
            ex = new ServiceException(Properties.Resources.RES_Exception_OperationReturnedUnexpectedStatusCode.Format(response.StatusCode));
        }
        else
        {
            ex = new ServiceException(Properties.Resources.RES_Exception_OperationReturnedUnexpectedStatusCodeWithContent.Format(response.StatusCode, responseContent));
        }
    }
    else
    {
        ex = new ServiceException(error.Message, error);
    }

    // Set other exception properties

    ex.Request = new HttpRequestMessageSurrogate(request);
    ex.Response = new HttpResponseMessageSurrogate(response, responseContent);

    // Dispose request

    if (request != null)
    {
```

DeserializeResponseAsync

A desserialização da resposta é simples:

```
1 reference | Hugo Ribeiro, 119 days ago | 2 authors, 2 changes | 1 incoming change
public async Task<(string, TResult)> DeserializeResponseAsync<TResult>(HttpRequestMessage request, HttpResponseMessage response, CancellationToken cancellationToken)
{
    // Validation

    SmartGuard.NotNull(() => request, request);
    SmartGuard.NotNull(() => response, response);

    // Validate content

    if (response.Content == null)
    {
        throw new SerializationException(Properties.Resources.RES_Exception_UnableToDeserializeResponse_ContentIsNull);
    }

    // Read the response content

    string responseContent = await response.Content.ReadAsStringAsync().ConfigureAwait(false);

    // Check the cancellation token

    cancellationToken.ThrowIfCancellationRequested();

    // Deserialize the body

    try
    {
        TResult result = JsonSerializer.Deserialize<TResult>(responseContent, this.ServiceClient.DeserializationOptions);
        return (responseContent, result);
    }
    catch (JsonException ex)
    {
        // Dispose request

        if (request != null)
        {
            request.Dispose();
        }

        // Dispose response

        if (response != null)
        {
            response.Dispose();
        }

        // Throw error

        throw new SerializationException(Properties.Resources.RES_Exception_UnableToDeserializeResponse, responseContent, ex);
    }
}
```

Autorização (HttpBearerChallenge)

O tema da autorização OAuth é um pouco mais complexo.

O ServiceClient suporta um conceito definido no protocolo OAuth, a que se costuma chamar HTTP Bearer Challenge.

https://tools.ietf.org/html/rfc6750#section-3

Basicamente, o comportamento é o seguinte:

- A API deve devolver ao cliente – sempre que não o conseguir autorizar por credenciais inválidas – um header – WWW-Authenticate – que inclua a informação necessária para este corrigir o seu pedido.

O ServiceClient usa exatamente esse mecanismo para suportar o tipo de credenciais AuthenticationCallbackCredentials, que tem, como principal benefício, não ser necessário programar hard-coded o endereço do servidor de identidade nem os scopes necessários para falar com determinada API. A API simplesmente falhará inicialmente com 401 e devolverá no header WWW-Authenticate essa informação.

As API desenvolvidas com o Lithium respeitam esta definição do protocolo OAuth. Outras API – incluindo as dos produtos Elevation – não o fazem. Nesses casos, as credenciais AuthenticationCallbackCredentials não podem ser utilizadas.

Autorização (ServiceClientCredentials)

Como vimos, o service client aceita as credenciais do serviço como uma instância de uma classe abstrata com o nome ServiceClientCredentials. Isso permite depois que o cliente escolha o tipo de credenciais que pretende usar, mas garante o comportamento por defeito que o service client espera (como vimos antes no envio dos pedidos).

A API é a seguinte:

```
57 references | Hugo Ribeiro, 43 days ago | 2 authors, 3 changes | 1 incoming change
public abstract partial class ServiceClientCredentials
{
    Public Fields

    Constructors

    #region Public Methods

    /// <summary>
    /// Allows to initialize the credentials depending on the specified service client.
    /// </summary>
    /// <typeparam name="T">The type of the service client.</typeparam>
    /// <param name="serviceClient">The service client instance that is using these credentials.</param>
    1 reference | Hugo Lourenço | 1 author, 1 change | 1 incoming change
    public virtual void InitializeServiceClient<T>(ServiceClient<T> serviceClient)
        where T : ServiceClient<T>
    {
    }

    /// <summary>
    /// Allows to apply the credentials before sending the request for the first time.
    /// </summary>
    /// <param name="request">The request that is being sent.</param>
    /// <param name="cancellationToken">The cancellation token.</param>
    /// <returns>
    /// A <see cref="Task"/> that represents the asynchronous operation.
    /// </returns>
    8 references | Hugo Ribeiro, 50 days ago | 2 authors, 2 changes | 1 incoming change
    public virtual Task HandleRequestAsync(HttpRequestMessage request, CancellationToken cancellationToken)...

    /// <summary>
    /// Allows to apply the credentials after sending the request based on the failure response received.
    /// </summary>
    /// <param name="request">The request that is being sent.</param>
    /// <param name="response">The response received after sending the request for the first time.</param>
    /// <param name="cancellationToken">The cancellation token.</param>
    /// <returns>
    /// A <see cref="Task{TResult}"/> that represents the asynchronous operation.
    /// A value indicating whether the request should be sent again.
    /// </returns>
    3 references | Hugo Lourenço | 1 author, 1 change | 1 incoming change
    public virtual Task<bool> HandleResponseAsync(HttpRequestMessage request, HttpResponseMessage response, CancellationToken cancellationToken)...

    #endregion
}
```

Invocado quando o service client é inicializado

Invocado antes de enviar o pedido

Invocado depois de receber a resposta

Autorização (AccessTokenCredentials)

Este tipo de credenciais permite passar diretamente um token previamente obtido de alguma forma. É o método mais básico e não faz qualquer tipo de gestão do access token (se for inválido por exemplo).

A implementação é bastante simples:

```
10 references | Hugo Ribeiro, 41 days ago | 2 authors, 4 changes | 1 incoming change
public partial class AccessTokenCredentials : ServiceClientCredentials
{
    #region Public Properties

    /// <summary>
    /// Gets the client application access token, previously obtained from the authority server.
    /// </summary>
    2 references | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
    public virtual string AccessToken
    {
        get;
    }

    #endregion

    Constructors

    #region Public Methods

    #region Overrides

    /// <inheritdoc />
    8 references | Hugo Ribeiro, 50 days ago | 2 authors, 2 changes | 1 incoming change
    public override Task HandleRequestAsync(HttpRequestMessage request, CancellationToken cancellationToken)
    {
        // Validation

        SmartGuard.NotNull(() => request, request);

        // Set the access token in the authorization header

        request.Headers.Authorization = new AuthenticationHeaderValue(OidcConstants.AuthenticationSchemes.Bearer, this.AccessToken);

        // Set the custom headers

        request.Headers.AddOrReplaceWithoutValidation(CustomHeaderNames.CredentialsKind, "access-token");

        // Default behavior

        return base.HandleRequestAsync(request, cancellationToken);
    }

    #endregion

    #endregion
}
```

Repare-se que apenas o método HandleRequestAsync é overriden. Os restantes ficam com o comportamento por defeito (nulo).

Autorização (ClientCredentials)

Estas credenciais permitem definir o client id, o client secret e o scope. A instância tratará de obter o access token quando for necessário, garantindo caching e a renovação do token.

Neste caso, são tratados tanto o pedido como a resposta.

```
/// <inheritdoc />
8 references | Hugo Ribeiro, 50 days ago | 2 authors, 2 changes | 1 incoming change
public override async Task HandleRequestAsync(HttpRequestMessage request, CancellationToken cancellationToken)
{
    // Validation

    SmartGuard.NotNull(() => request, request);

    // Set the custom headers

    request.Headers.AddOrReplaceWithoutValidation(CustomHeaderNames.CredentialsKind, "client-credentials");
    request.Headers.AddOrReplaceWithoutValidation(CustomHeaderNames.ClientId, this.ClientId);

    // Check if the access token was already retrieved

    if (!string.IsNullOrEmpty(this.AccessToken))
    {
        // Is it not expired?

        if (!JwtSecurityTokenDecoder.HasExpired(this.AccessToken))
        {
            // Set the access token in the authorization header

            request.Headers.Authorization = new AuthenticationHeaderValue(OidcConstants.AuthenticationSchemes.Bearer, this.AccessToken);

            // Default behavior

            await base.HandleRequestAsync(request, cancellationToken).ConfigureAwait(false);

            // Return

            return;
        }
    }

    // Otherwise, retrieve a new access token

    await this.RetrieveAndSetAccessTokenAsync(request, cancellationToken).ConfigureAwait(false);

    // Default behavior

    await base.HandleRequestAsync(request, cancellationToken).ConfigureAwait(false);
}
```

```
/// <inheritdoc />
3 references | Hugo Lourenço | 1 author, 1 change | 1 incoming change
public override async Task<bool> HandleResponseAsync(HttpRequestMessage request, HttpResponseMessage response, CancellationToken cancellationToken)
{
    // Validation

    SmartGuard.NotNull(() => request, request);
    SmartGuard.NotNull(() => response, response);

    // If the status code is unauthorized it means the access token failed
    // We should clear it to ensure that future requests try to get a new one
    // If there is a WWW-Authenticate header indicating that the token has expired
    // we should try the request again

    if (response.StatusCode == HttpStatusCode.Unauthorized)
    {
        this.AccessToken = null;

        if (response.Headers.WwwAuthenticate != null && response.Headers.WwwAuthenticate.Count > 0)
        {
            string header = response.Headers.WwwAuthenticate.ElementAt(0).ToString();

            if (HttpBearerChallenge.IsTokenExpiredBearerChallenge(response.RequestMessage.Method, response.RequestMessage.RequestUri, header))
            {
                await this.RetrieveAndSetAccessTokenAsync(request, cancellationToken).ConfigureAwait(false);
                return true;
            }
        }
    }

    // Default result

    return false;
}
```

TokenClient

Esta é a classe do Hydrogen que deve ser utilizada para obter tokens de qualquer servidor OAuth.

É usada por ClientCredentials como se segue:

```
2 references | Hugo Lourenço | 1 author, 1 change | 1 incoming change
private async Task<string> RetrieveAccessTokenCoreAsync(HttpMessageHandler handler, CancellationToken cancellationToken)
{
    // Debug

    System.Diagnostics.Debug.WriteLine("Retrieving access token (ClientCredentials)...");

    // Default result

    string result = null;

    // Retrieve a new access token using the token client

    ClientCredentialsTokenRequest tokenRequest = new ClientCredentialsTokenRequest()
    {
        Address = this.AuthorityServer.AbsoluteUri,
        AddressIsAuthority = true,
        ClientCredentialStyle = this.CredentialStyle,
        ClientId = this.ClientId,
        ClientSecret = this.ClientSecret,
        Scope = this.Scope
    };

    if (this.TokenCache != null)
    {
        tokenRequest.Cache = this.TokenCache;
    }

    using (TokenClient tokenClient = GetTokenClient(handler))
    {
        TokenResponse tokenResponse = await tokenClient.GetTokenAsync(tokenRequest, cancellationToken).ConfigureAwait(false);
        if (tokenResponse == null || tokenResponse.IsError)
        {
            string message = Properties.Resources.RES_Exception_AuthorityServer_TokenEndpointReturnedError.Format(tokenResponse.Error);
            throw new AuthorityServerException(message, tokenResponse.Exception);
        }

        result = tokenResponse.AccessToken;
    }

    // Result

    return result;
}
```

Autorização (AuthenticationCallbackCredentials)

Estas credenciais permitem suportar o tal cenário do bearer challenge. Primeiro é realizado um pedido sem credenciais, para obter o header WWW-Authenticate, que depois é usado para obter o token (os parâmetros para o obter ficam nos argumentos da função).

```
/// <inheritdoc />
8 references | Hugo Ribeiro, 50 days ago | 2 authors, 2 changes | 1 incoming change
public override async Task HandleRequestAsync(HttpRequestMessage request, CancellationToken cancellationToken)
{
    // Validation

    SmartGuard.NotNull(() => request, request);

    // Set the custom headers

    request.Headers.AddOrReplaceWithoutValidation(CustomHeaderNames.CredentialsKind, "authentication-callback");
    request.Headers.AddOrReplaceWithoutValidation(CustomHeaderNames.Challenge, "true");

    // Before sending the request we try to get the access token from cache

    string accessToken = await this.RetrieveAccessTokenFromHttpBearerChallengeCacheAsync(request).ConfigureAwait(false);
    if (!string.IsNullOrEmpty(accessToken))
    {
        // Set the access token in the authorization header

        request.Headers.Authorization = new AuthenticationHeaderValue(OidcConstants.AuthenticationSchemes.Bearer, accessToken);
    }

    // Default behavior

    await base.HandleRequestAsync(request, cancellationToken).ConfigureAwait(false);
}
```

```
/// <inheritdoc />
3 references | Hugo Ribeiro, 50 days ago | 2 authors, 2 changes | 1 incoming change
public override async Task<bool> HandleResponseAsync(HttpRequestMessage request, HttpResponseMessage response, CancellationToken cancellationToken)
{
    // Validation

    SmartGuard.NotNull(() => request, request);
    SmartGuard.NotNull(() => response, response);

    // If the status code is unauthorized we look for a bearer challenge to try to retrieve the access token
    // It the access token is available we retry the request

    if (response.StatusCode == HttpStatusCode.Unauthorized)
    {
        string accessToken = await this.RetrieveAccessTokenAsync(response).ConfigureAwait(false);
        if (!string.IsNullOrEmpty(accessToken))
        {
            if (request != null && request.Headers != null)
            {
                // Set the access token in the authorization header

                request.Headers.Authorization = new AuthenticationHeaderValue(OidcConstants.AuthenticationSchemes.Bearer, accessToken);

                // Set the custom headers

                request.Headers.AddOrReplaceWithoutValidation(CustomHeaderNames.Challenge, "false");
            }

            // Retry

            return true;
        }
    }

    // Default result

    return false;
}
```

Autorização (AuthenticationCallbackCredentials)

O cliente deve passar um delegate para obter o token. É usado assim:

```
1 reference | Hugo Lourenço | 1 author, 1 change | 1 incoming change
private async Task<string> RetrieveAccessTokenAsync(HttpResponseMessage response)
{
    // Get the WWW-Authenticate header value

    if (response.Headers.WwwAuthenticate != null && response.Headers.WwwAuthenticate.Count > 0)
    {
        string header = response.Headers.WwwAuthenticate.ElementAt(0).ToString();

        // Is it a bearer challenge?

        if (HttpBearerChallenge.IsBearerChallenge(header))
        {
            // Is it valid?

            if (HttpBearerChallenge.IsValidBearerChallenge(response.RequestMessage.Method, response.RequestMessage.RequestUri, header))
            {
                // Create the challenge

                HttpBearerChallenge challenge = new HttpBearerChallenge(response.RequestMessage.Method, response.RequestMessage.RequestUri, header, true);

                // Add to cache

                HttpBearerChallengeCache.Instance().SetChallenge(response.RequestMessage.Method, response.RequestMessage.RequestUri, challenge);

                // Invoke the callback

                return await this.OnAuthenticate(new AuthenticationCallbackArgs(challenge.AuthorizationUri, challenge.ResourceUri, challenge.Audience)).ConfigureAwait(false);
            }

            // The response does not contain a valid bearer challenge

            throw new HttpBearerChallengeException(
                HttpBearerChallengeErrorCode.InvalidBearerChallenge,
                Properties.Resources.RES_Exception_HttpBearerChallenge_InvalidChallenge
                    .Format(header));
        }

        // The response does not contain the bearer challenge

        throw new HttpBearerChallengeException(
            HttpBearerChallengeErrorCode.BearerChallengeNotFound,
            Properties.Resources.RES_Exception_HttpBearerChallenge_ChallengeNotFound
                .Format(header));
    }

    // The response does not contain the WWW-Authenticate header

    throw new HttpBearerChallengeException(
        HttpBearerChallengeErrorCode.WWWAuthenticateHeaderNotFound,
        Properties.Resources.RES_Exception_HttpBearerChallenge_WWWAuthenticateNotFound);
}
```

Autorização (NoCredentials)

Por fim, temos este tipo especial de credenciais que corresponde a não definir credenciais nenhumas. É um caso marginal.

Será útil para invocar serviços que não usam OAuth ou para invocar endpoints não protegidos de serviços com OAuth.

Serialização de pedidos e respostas

Como vimos antes, a serialização dos pedidos e das respostas é feito com o JsonSerializer e não com o NewtonSoft (que deve desaparecer das dependências dos nossos projetos todos ASAP).

Isto implica limitações mas suporta todas as funcionalidades requeridas pelos modelos que podem ser construídos com o SDK Lithium.

No ServiceClient são usadas opções por defeito, que podem ser modificadas (nas implementações concretas, não pelo cliente).

```
/// <summary>
/// Gets or sets the options that will be used in serialization.
/// </summary>
22 references | 10/10 passing | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
public virtual JsonSerializerOptions SerializationOptions
{
    get;
    protected set;
}

/// <summary>
/// Gets or sets the options that will be used in deserialization.
/// </summary>
20 references | 10/10 passing | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
public virtual JsonSerializerOptions DeserializationOptions
{
    get;
    protected set;
}
```

```
1 reference | Hugo Ribeiro, 137 days ago | 2 authors, 2 changes | 1 incoming change
protected virtual void InitializeOptions()
{
    // Serialization and deserialization options

    JsonSerializerOptions serializationOptions = new JsonSerializerOptions()
    {
        WriteIndented = true,
        IgnoreNullValues = true,
        IgnoreReadOnlyProperties = true,
        PropertyNamingPolicy = JsonNamingPolicy.CamelCase
    };

    serializationOptions.Converters.Add(new Iso8601TimeSpanConverter());
    serializationOptions.Converters.Add(new Iso8601TimeSpanNullableConverter());

    this.SerializationOptions = serializationOptions;

    JsonSerializerOptions deserializationOptions = new JsonSerializerOptions()
    {
        IgnoreNullValues = true,
        IgnoreReadOnlyProperties = true,
        PropertyNamingPolicy = JsonNamingPolicy.CamelCase
    };

    deserializationOptions.Converters.Add(new Iso8601TimeSpanConverter());
    deserializationOptions.Converters.Add(new Iso8601TimeSpanNullableConverter());

    this.DeserializationOptions = deserializationOptions;
}
```

Tratamento de resultados

Para um ServiceClient, uma operação da API pode:

- Retornar o HTTP Status Code esperado e é sucesso. A operação devolverá uma ServiceOperationResult ou ServiceOperationResult<T>
- Retornar um HTTP Status Code inesperado e é um erro. Será lançada uma exceção do tipo ServiceException.

Isto induz diretamente a forma correta de invocar as operações na client lib. Só existe (ainda, pelo menos) uma forma correta de o fazer. Este é o aspeto em que se têm cometido mais erros na utilização das client libs.

O "código tipo" é verboso e, até um certo ponto, um pouco contraintuitivo porque é lançada uma exceção mesmo para status codes que, não sendo o esperado, de sucesso, não são completamente inesperados (ex.: NotFound).

Este modelo não é uma invenção nossa. Foi adaptado das livrarias do Azure (TableStorage, BlobStorage, KeyVault, etc.).

Tem por base a ideia que uma operação só tem 1 único resultado esperado, todos os outros devem ser alvo de tratamento explícito.

Outro aspeto importante a considerar é que é possível que ocorram outros erros, esses completamente inesperados (ex.: não haver memória disponível) e, esses lançarão, exceções completamente diferentes.

Código como este está completamente errado:

```
using MyServiceClient client = new MyServiceCLient(...)
try
{
   ServiceOperationResult result = client.CallSomethingAsync().ConfigureAwait(false);
   (...)
}
catch (Exception ex)
{
   (...)
}
```

Tratamento de resultados (sucesso)

Quando a operação devolve o resultado esperado, é retornada uma instância de ServiceOperationResult (se o resultado não tiver body) ou ServiceOperationResult<T> (se tiver).

```
/// <summary>
/// Describes the response of a service operation with a response body.
/// </summary>
/// <typeparam name="T">The type of the response body.</typeparam>
/// <seealso cref="HttpOperationResponse{T}" />
99+ references | Hugo Ribeiro, 41 days ago | 2 authors, 3 changes | 1 incoming change
public partial class ServiceOperationResult<T> : HttpOperationResponse<T>
{
    Constructors
}
```

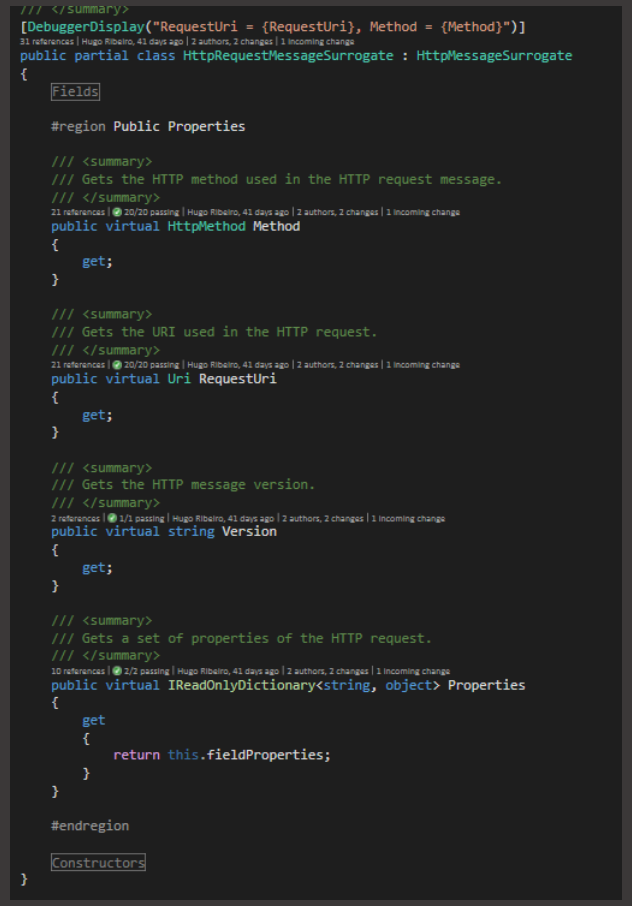
```
/// <summary>
/// Describes the response of a service operation with a response body.
/// </summary>
/// <typeparam name="T">The type of the response body.</typeparam>
/// <seealso cref="HttpOperationResponse" />
/// <seealso cref="IHttpOperationResponse{T}" />
[DebuggerDisplay("Body = {Body}")]
8 references | Hugo Ribeiro, 41 days ago | 2 authors, 3 changes | 1 incoming change
public partial class HttpOperationResponse<T> : HttpOperationResponse, IHttpOperationResponse<T>
{
    #region Public Properties

    /// <inheritdoc />
    57 references | 34/34 passing | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
    public virtual T Body
    {
        get;
    }

    #endregion

    Constructors
}
```

```
/// <summary>
/// Describes the response of a service operation without a response body.
/// </summary>
12 references | Hugo Ribeiro, 41 days ago | 2 authors, 3 changes | 1 incoming change
public partial class HttpOperationResponse : IHttpOperationResponse
{
    #region Public Properties

    /// <inheritdoc />
    95 references | 18/18 passing | Hugo Ribeiro, 41 days ago | 2 authors, 3 changes | 1 incoming change
    public virtual HttpRequestMessageSurrogate Request
    {
        get;
    }

    /// <inheritdoc />
    61 references | 19/19 passing | Hugo Ribeiro, 41 days ago | 2 authors, 3 changes | 1 incoming change
    public virtual HttpResponseMessageSurrogate Response
    {
        get;
    }

    #endregion

    Constructors
}
```

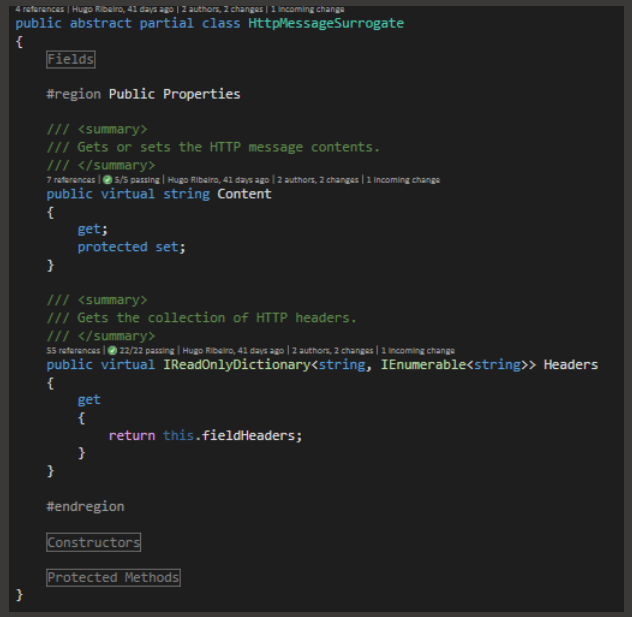
HttpRequestMessageSurrogate

Este tipo disponibiliza uma representação (clone) em memória da HttpRequestMessage efetivamente utilizada no pedido (porque esta última é disposable).

Esta é outra das vantagens do modelo ServiceOperationResult/ServiceException. O código de tratamento dos resultados pode depender do request feito e da response obtida.



```
/// </summary>
[DebuggerDisplay("RequestUri = {RequestUri}, Method = {Method}")]
31 references | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
public partial class HttpRequestMessageSurrogate : HttpMessageSurrogate
{
    Fields

    #region Public Properties

    /// <summary>
    /// Gets the HTTP method used in the HTTP request message.
    /// </summary>
    21 references | 20/20 passing | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
    public virtual HttpMethod Method
    {
        get;
    }

    /// <summary>
    /// Gets the URI used in the HTTP request.
    /// </summary>
    21 references | 20/20 passing | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
    public virtual Uri RequestUri
    {
        get;
    }

    /// <summary>
    /// Gets the HTTP message version.
    /// </summary>
    2 references | 1/1 passing | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
    public virtual string Version
    {
        get;
    }

    /// <summary>
    /// Gets a set of properties of the HTTP request.
    /// </summary>
    10 references | 2/2 passing | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
    public virtual IReadOnlyDictionary<string, object> Properties
    {
        get
        {
            return this.fieldProperties;
        }
    }

    #endregion

    Constructors
}
```



```
4 references | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
public abstract partial class HttpMessageSurrogate
{
    Fields

    #region Public Properties

    /// <summary>
    /// Gets or sets the HTTP message contents.
    /// </summary>
    7 references | 5/5 passing | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
    public virtual string Content
    {
        get;
        protected set;
    }

    /// <summary>
    /// Gets the collection of HTTP headers.
    /// </summary>
    55 references | 22/22 passing | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
    public virtual IReadOnlyDictionary<string, IEnumerable<string>> Headers
    {
        get
        {
            return this.fieldHeaders;
        }
    }

    #endregion

    Constructors

    Protected Methods
}
```

HttpResponseMessageSurrogate

Este tipo disponibiliza a representação (clone) em memória da HttpResponseMessage:

```
/// <summary>
/// Defines a surrogate of <see cref="HttpResponseMessage"/> instances.
/// The surrogate allow accessing most of the properties after the instances are disposed.
/// </summary>
[DebuggerDisplay("StatusCode = {StatusCode}, ReasonPhrase = {ReasonPhrase}")]
31 references | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
public partial class HttpResponseMessageSurrogate : HttpMessageSurrogate
{
    #region Public Properties

    /// <summary>
    /// Gets a value indicating whether the HTTP response was successful.
    /// </summary>
    21 references | 20/20 passing | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
    public virtual bool IsSuccessStatusCode
    {
        get;
    }

    /// <summary>
    /// Gets the status code of the HTTP response.
    /// </summary>
    36 references | 34/34 passing | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
    public virtual HttpStatusCode StatusCode
    {
        get;
    }

    /// <summary>
    /// Gets the reason phrase, typically sent along with the status code.
    /// </summary>
    3 references | 2/2 passing | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
    public virtual string ReasonPhrase
    {
        get;
    }

    /// <summary>
    /// Gets the HTTP message version.
    /// </summary>
    2 references | 1/1 passing | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
    public virtual string Version
    {
        get;
    }

    #endregion

    Constructors
}
```

Tratamento de resultados (insucesso)

Quando a operação devolve um resultado inesperado, é lançada uma exceção do tipo ServiceException.

```
/// <summary>
/// Describes an error returned in the HTTP response of a REST service.
/// </summary>
/// <seealso cref="RestException" />
99+ references | Hugo Ribeiro, 40 days ago | 2 authors, 3 changes | 1 incoming change
public partial class ServiceException : RestException
{
    #region Public Properties

    /// <summary>
    /// Gets information about the HTTP request that raised the error.
    /// </summary>
    1 reference | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
    public virtual HttpRequestMessageSurrogate Request
    {
        get;
        internal set;
    }

    /// <summary>
    /// Gets information about the HTTP response that returned the error.
    /// </summary>
    27 references | 14/14 passing | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
    public virtual HttpResponseMessageSurrogate Response
    {
        get;
        internal set;
    }

    /// <summary>
    /// Gets the response object that describes the error.
    /// </summary>
    41 references | 13/13 passing | Hugo Ribeiro, 41 days ago | 2 authors, 2 changes | 1 incoming change
    public virtual ServiceError Body
    {
        get;
        internal set;
    }

    #endregion

    Constructors
}
```

Note-se que o body é sempre um ServiceError.

Padrão de invocação de operações

Dos pontos anteriores derivam, então, a forma correta de invocar determinada operação de um serviço. Há N exemplos disso mesmo no código dos projetos Client.Console dos micro serviços. Por exemplo:

```
using MyServiceClient client = new MyServiceClient(...);
try
{
  ServiceOperationResult<MyModel> result = await client.GetMyModelAsync(...).ConfigureAwait(false);
  (...)
}
catch (ServiceException ex)
{
  if (ex.Response.StatusCode == HttpStatusCode.NotFound)
  {
    (...)
  }
  else if (ex.Response.StatusCode == HttpStatusCode.BadRequest)
  {
    if (ex.Body.Code.EqualsNoCase(“IdRequired”)) then
    {
      (...)
    }
    else
    {
      (...)
    }
  }
  else
  {
    (...)
  }
}
```