HttpClient

прошлое, настоящее, будущее

Риваль Абдрахманов

Positive Technologies

SpbDotNet, 2019

Содержание

- 1 HttpClient базовая информация
- 2 Неочевидные проблемы
- ③ Интерфейс IHttpClientFactory
- 4 Дополнительные улучшения в .NET Core 2.1
- 5 HttpRequestMessage и HttpResponseMessage
- 6 Новое в .NET Core 3.0

HttpClient - базовая информация

HttpClient Class

 Provides a base class for sending HTTP requests and receiving HTTP responses from a resource identified by a URI;

HttpClient Class

- Provides a base class for sending HTTP requests and receiving HTTP responses from a resource identified by a URI;
- GetAsync(...), PostAsync(...), SendAsync(...) и др.;

HttpClient Class

- Provides a base class for sending HTTP requests and receiving HTTP responses from a resource identified by a URI;
- GetAsync(...), PostAsync(...), SendAsync(...) и др.;
- HttpClient реализует IDisposable.

• Provides a mechanism for releasing unmanaged resources;

- Provides a mechanism for releasing unmanaged resources;
- public void Dispose();

- Provides a mechanism for releasing unmanaged resources;
- public void Dispose();
- Конструкция using(...);

- Provides a mechanism for releasing unmanaged resources;
- public void Dispose();
- Конструкция using(...);
- ullet Диспозиться o диспозь

- Provides a mechanism for releasing unmanaged resources;
- public void Dispose();
- Конструкция using(...);
- ullet Диспозиться o диспозь
- ullet Диспозиться o внимательней

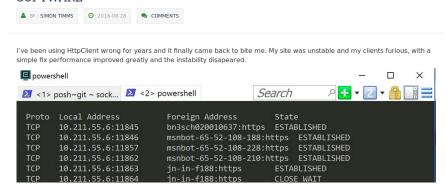
Disposable HttpClient

```
using(var client = new HttpClient())
{
  var response = await client.GetStringAsync(...);
}
```

Неочевидные проблемы

https://aspnetmonsters.com/2016/08/2016-08-27-httpclientwrong/

YOU'RE USING HTTPCLIENT WRONG AND IT IS DESTABILIZING YOUR SOFTWARE



```
for(int i = 0; i < 10; i++)
{
  using (var client = new HttpClient())
  {
    await client.GetStringAsync("https://google.com");
  }
}</pre>
```

Проверяем через netstat:

```
tcp
                  0 rafaelldi-Latitud:41757 lg-in-f147.1e100.:https TIME WAIT
                  0 rafaelldi-Latitud:44779 lg-in-f147.1e100.:https TIME WAIT
tcp
                  0 rafaelldi-Latitud:37367 lg-in-f147.1e100.:https TIME WAIT
tcp
                  0 rafaelldi-Latitud:32979 lg-in-f147.1e100.:https TIME WAIT
tcp
tcp
                  0 rafaelldi-Latitud:38399 lg-in-f104.1e100.:https TIME WAIT
                  0 rafaelldi-Latitud:44257 lg-in-f147.1e100.:https TIME WAIT
tcp
                  0 rafaelldi-Latitud:46173 lg-in-f147.1e100.:https TIME WAIT
tcp
           0
                  0 rafaelldi-Latitud:44449 lq-in-f147.1e100.:https TIME WAIT
tcp
                  0 rafaelldi-Latitud:35151 lg-in-f147.1e100.:https TIME WAIT
tcp
tcp
                  0 rafaelldi-Latitud:46791 lg-in-f147.1e100.:https TIME WAIT
```

- 10 сокетов в состоянии *TIME WAIT*;
- Соединение закрыто с одной стороны, но мы всё ещё ждём доходящие пакеты, чтобы правильно их обработать;
- Приводит к SocketException.

"HttpClient is intended to be instantiated once and re-used throughout the life of an application. Instantiating an HttpClient class for every request will exhaust the number of sockets available under heavy loads."

https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient

Решение проблемы - переиспользование клиента:

```
private static HttpClient Client = new HttpClient();
```

https://byterot.blogspot.com/2016/07/singleton-httpclient-dns.html

Wednesday, 20 July 2016

Singleton HttpClient? Beware of this serious behaviour and how to fix it

If you are consuming a Web API in your server-side code (or .NET client-side app), you are very likely to be using an HttpClient.

HttpClient is a very nice and clean implementation that came as part of Web API and replaced its clunky predecessor WebClient (although only in its HTTP functionality, WebClient can do more than just HTTP).

HttpClient is usually meant to be used with more than just a single request. It conveniently allows for default headers to be set and applied to all requests. Also you can plug in a CookieContainer to allow for all sessions.

- Не учитываются изменения DNS;
- Соединение держится до закрытия сокета.

Решение для .NET Framework:

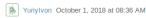
- Класс ServicePointManager;
- ServicePointManager.DnsRefreshTimeout время, которое будет закеширован полученный IP адрес для каждого доменного имени, по умолчанию 2 минуты;
- ServicePoint.ConnectionLeaseTimeout время, которое соединение будет удерживаться открытым, по умолчанию не ограничено;
- ServicePoint.MaxIdleTime время бездействия, после которого соединение будет закрыто.

Пример:

```
ServicePointManager.DnsRefreshTimeout = 60000;
var sp = ServicePointManager.FindServicePoint(
   new Uri("https://google.com"));
sp.ConnectionLeaseTimeout = 60000;
sp.MaxIdleTime = 60000;
```

Лимит одновременных соединений с сервером

https://habr.com/ru/post/424873/



Подводные камни HttpClient в .NET

.NET

Продолжая серию статей о «подводных камнях» не могу обойти стороной System.Net.HttpClient, который очень часто используется на практике, но при этом имеет несколько серьезных проблем, которые могут быть сразу не видны.

Достаточно частая проблема в программировании — то, что разработчики сфокусированы только на функциональных возможностях того или иного компонента, при этом совершенно не учитывают очень важную нефункциональную составляющую, которая может влиять на производительность, масштабируемость, легкость восстановления в случае сбоев, безопасность и т.д. Например, тот же HttpClient — вроде бы и элементарный компонент, но есть несколько вопросов: сколько он создает параллельных соединений к серверу, как долго они живут, как он себя поведет, если DNS имя, к которому обращался ранее, будет переключено на другой IP адрес? Попробуем ответить на эти вопросы в статье.

Лимит одновременных соединений с сервером

- Лимит одновременных соединений с сервером по умолчанию равен 2;
- ServicePointManager.DefaultConnectionLimit;
- Для localhost по умолчанию равен int.MaxValue;
- Только для .NET Framework.

Интерфейс IHttpClientFactory

IHttpClientFactory

IHttpClientFactory

An IHttpClientFactory can be registered and used to configure and create HttpClient instances in an app.

Интерфейс был добавлен в ASP.NET Core 2.1.

Для консольного приложения придётся добавить Microsoft. Extensions. Hosting и Microsoft. Extensions. Http.

IHttpClientFactory - базовое использование

```
services.AddHttpClient();
```

IHttpClientFactory - базовое использование

• Регистрация через метод расширения IServiceCollection:

```
services.AddHttpClient();
```

• Добавление в конструктор с помощью DI:

```
public SomeService(IHttpClientFactory
    clientFactory)
{
    _clientFactory = clientFactory;
}
```

IHttpClientFactory - базовое использование

• Регистрация через метод расширения IServiceCollection:

```
services.AddHttpClient();
```

• Добавление в конструктор с помощью DI:

```
public SomeService(IHttpClientFactory
    clientFactory)
{
    _clientFactory = clientFactory;
}
```

Создание клиента (без using):

```
var client = _clientFactory.CreateClient();
var response = await client.SendAsync(request);
```

Named clients

```
services.AddHttpClient("some-site", c =>
{
   c.BaseAddress =
      new Uri("https://some-site.com/");
   c.DefaultRequestHeaders
      .Add("Accept", "application/json");
});
```

Named clients

```
services.AddHttpClient("some-site", c =>
 ₹
    c.BaseAddress =
      new Uri("https://some-site.com/");
    c.DefaultRequestHeaders
      .Add("Accept", "application/json");
 });

    Добавление в конструктор с помощью DI:

 public SomeService(IHttpClientFactory
     clientFactory)
    _clientFactory = clientFactory;
```

Named clients

```
services.AddHttpClient("some-site", c =>
 ₹
    c.BaseAddress =
      new Uri("https://some-site.com/");
    c.DefaultRequestHeaders
      .Add("Accept", "application/json");
 });

    Добавление в конструктор с помощью DI:

 public SomeService(IHttpClientFactory
     clientFactory)
    _clientFactory = clientFactory;
 var client =
```

```
_clientFactory.CreateClient("some site");
```

Typed clients

```
Класс типизированного клиента:
public class SomeSiteClient
{
  private readonly HttpClient _client;
  public SomeSiteClient(HttpClient client)
    client.BaseAddress =
      new Uri("https://some-site.com/");
    client.DefaultRequestHeaders
      .Add("Accept", "application/json");
    _client = client;
```

Typed clients

```
Kласс типизированного клиента:
public class SomeSiteClient
{
    ...

public async Task < SomeData > GetSomeData()
    {
    var response =
        await _client.GetAsync("/get-some-data");
    ...
```

return result;

Typed clients

• Регистрация типизированного клиента:

```
services.AddHttpClient < SomeSiteClient > ();
```

Typed clients

• Регистрация типизированного клиента:

```
services.AddHttpClient < SomeSiteClient > ();
```

• Добавление в конструктор через DI:

```
public SomeService(SomeSiteClient someSiteClient)
{
    _someSiteClient = someSiteClient;
}
```

Refit

Библиотека Refit для REST API (https://github.com/reactiveui/refit)

```
public interface ISomeSiteClient
{
    [Get("/get-some-data")]
    Task<SomeData > GetSomeData();
}

public class SomeData
{
    public string Message { get; set; }
}
```

Refit

• Регистрация клиента:

```
services
.AddRefitClient < ISomeSiteClient > ()
.ConfigureHttpClient(c => c.BaseAddress =
    new Uri("https://some-site.com"));
```

Refit

• Регистрация клиента:

```
services
   .AddRefitClient < ISomeSiteClient > ()
   .ConfigureHttpClient(c => c.BaseAddress =
        new Uri("https://some-site.com"));
```

Добавление в конструктор через DI:

```
public SomeService(ISomeSiteClient someSiteClient)
{
   _someSiteClient = someSiteClient;
}
```

Дополнительные улучшения в .NET Core 2.1

DelegatingHandler

- DelegatingHandler позволяют создать цепочку обработки исходящих запросов;
- Схоже с middleware в ASP.NET Core:
- Функциональность уже была, но с *IHttpClientFactory* стало проще использовать.

Создание DelegatingHandler

```
public class SomeHandler : DelegatingHandler
  protected override async Task < HttpResponseMessage >
     SendAsync (HttpRequestMessage request,
     CancellationToken cancellationToken)
    var response = await base.SendAsync(request,
       cancellationToken);
    return response;
```

Регистрация DelegatingHandler

```
services.AddHttpClient("some-site")
  //first
  .AddHttpMessageHandler < OutsideHandler > ()
  //second
  .AddHttpMessageHandler < InsideHandler > ()
```

Polly

- Polly популярная библиотека обработки ошибок;
- Содержит различные политики: Retry, Circuit Breaker, Timeout, ...
- Необходимо установить Microsoft. Extensions. Http. Polly;
- Подходит не только для HttpClient.



Добавление политик

Обработка всех ответов со статус кодами 5хх и 408.

```
services.AddHttpClient("some-site")
.AddTransientHttpErrorPolicy(p => p.RetryAsync(3))
.AddTransientHttpErrorPolicy(
   p => p.CircuitBreakerAsync(5,
        TimeSpan.FromSeconds(30)));
```

Настройка внутреннего HttpMessageHandler

```
services.AddHttpClient("some-site")
   .ConfigurePrimaryHttpMessageHandler(() =>
{
    return new SocketsHttpHandler()
    {
        AutomaticDecompression =
            DecompressionMethods.GZip
    };
});
```

Настройка времени жизни HttpMessageHandler

- Для каждого именованного клиента есть свой HttpMessageHandler;
- IHttpClientFactory при создании нового HttpClient будет переиспользовать HttpMessageHandler, если его время жизни не вышло;
- Время жизни по умолчанию 2 минуты.

```
services.AddHttpClient("some-site")
.SetHandlerLifetime(TimeSpan.FromMinutes(5));
```

HttpRequestMessage и HttpResponseMessage

Hовое в .NET Core 3.0