HttpClient

прошлое, настоящее, будущее

Риваль Абдрахманов

Positive Technologies

SpbDotNet, 2019

Содержание

HttpClient - базовая информация

HttpClient Class

- Базовый класс для отправки HTTP-запросов и получения HTTP-ответов;

HttpClient Class

- Базовый класс для отправки HTTP-запросов и получения HTTP-ответов;
- *GetAsync*(...), *PostAsync*(...), *SendAsync*(...) и др.;

HttpClient Class

- Базовый класс для отправки HTTP-запросов и получения HTTP-ответов;
- *GetAsync*(...), *PostAsync*(...), *SendAsync*(...) и др.;
- *HttpClient* реализует *IDisposable*.

IDisposable Interface

- Предоставляет механизм для освобождения неуправляемых ресурсов;

IDisposable Interface

- Предоставляет механизм для освобождения неуправляемых ресурсов;
- *public void Dispose*();

IDisposable Interface

- Предоставляет механизм для освобождения неуправляемых ресурсов;
- *public void Dispose*();
- Конструкция *using*(...);

IDisposable Interface

- Предоставляет механизм для освобождения неуправляемых ресурсов;
- *public void Dispose*();
- Конструкция *using*(...);
- Диспозиться $\rightarrow$ диспозь

IDisposable Interface

- Provides a mechanism for releasing unmanaged resources;
- *public void Dispose*();
- Конструкция *using*(...);
- ~~Диспозиться → диспозь~~
- Диспозиться → будь внимательней

Disposable HttpClient

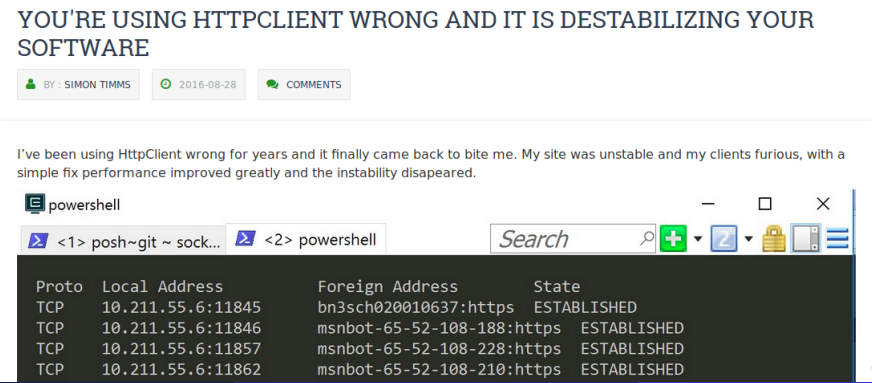
Мотивация:

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

Неочевидные проблемы

Проблема socket exhaustion

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



Проблема socket exhaustion

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

Проблема socket exhaustion

Проверяем через netstat:

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

Проблема socket exhaustion

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

Проблема socket exhaustion

“HttpClient предназначен для однократного создания экземпляра и повторного использования в течение всего жизненного цикла приложения.”

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

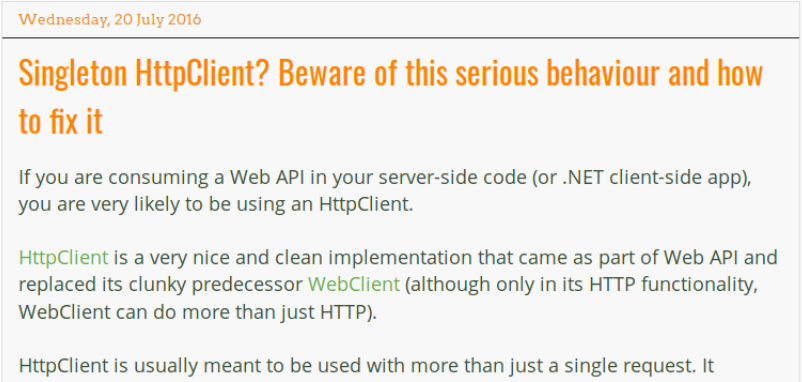
Проблема socket exhaustion

Решение проблемы - переиспользование клиента:

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

Проблема кеширования DNS

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

Проблема кеширования DNS

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

Проблема кеширования DNS

Решение для .NET Framework:

- Класс *ServicePointManager*;

Проблема кеширования DNS

Решение для .NET Framework:

- Класс *ServicePointManager*;
- *ServicePointManager.DnsRefreshTimeout* (2 мин);

Проблема кеширования DNS

Решение для .NET Framework:

- Класс *ServicePointManager*;
- *ServicePointManager.DnsRefreshTimeout* (2 мин);
- *ServicePoint.ConnectionLeaseTimeout* (не ограничено);

Проблема кеширования DNS

Решение для .NET Framework:

- Класс *ServicePointManager*;
- *ServicePointManager.DnsRefreshTimeout* (2 мин);
- *ServicePoint.ConnectionLeaseTimeout* (не ограничено);
- *ServicePoint.MaxIdleTime* (100 сек).

Проблема кеширования DNS

Пример:

```
ServicePointManager.DnsRefreshTimeout = 60000;

var sp = ServicePointManager
  .FindServicePoint(new Uri("http://site.com"));
sp.ConnectionLeaseTimeout = 60000;
sp.MaxIdleTime = 60000;
```

Лимит одновременных соединений

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

YuriyIvon October 1, 2018 at 08:36 AM

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

Лимит одновременных соединений

- Лимит по умолчанию равен 2;

Лимит одновременных соединений

- Лимит по умолчанию равен 2;
- *ServicePointManager.DefaultConnectionLimit*;

Лимит одновременных соединений

- Лимит по умолчанию равен 2;
- *ServicePointManager.DefaultConnectionLimit*;
- Для *localhost* по умолчанию равен *int.MaxValue*;

Лимит одновременных соединений

- Лимит по умолчанию равен 2;
- *ServicePointManager.DefaultConnectionLimit*;
- Для *localhost* по умолчанию равен *int.MaxValue*;
- Только для .NET Framework.

Выводы

- Нельзя на каждый запрос переоткрывать соединение;
- Нельзя постоянно держать соединение открытым;
- Вручную управлять сложно.

Интерфейс IHttpClientFactory

IHttpClientFactory

- Позволяет создавать и конфигурировать *HttpClient*;

.

IHttpClientFactory

- Позволяет создавать и конфигурировать *HttpClient*;
- Был добавлен в ASP.NET Core 2.1;

.

IHttpClientFactory

- Позволяет создавать и конфигурировать *HttpClient*;
- Был добавлен в ASP.NET Core 2.1;
- Для консольного приложения необходимо добавить *Microsoft.Extensions.Hosting* и *Microsoft.Extensions.Http*

.

IHttpClientFactory

Регистрация через метод расширения *IServiceCollection*:

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

IHttpClientFactory

Добавление в конструктор с помощью DI:

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

IHttpClientFactory

Создание клиента:

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

Named clients

Регистрация через метод расширения *IServiceCollection*:

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

Named clients

Добавление в конструктор с помощью DI:

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

Named clients

Создание клиента:

```
var client =
   _clientFactory.CreateClient("some-site");
```

Typed clients

Класс типизированного клиента:

```
public class SomeSiteClient
{
    ...
}
```

Typed clients

Класс типизированного клиента:

```
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

Класс типизированного клиента:

```
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

Добавление в конструктор через DI:

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

Refit

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

Refit

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

Refit

Регистрация клиента:

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

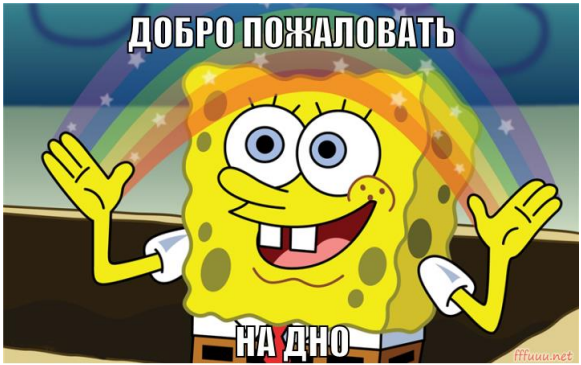
Refit

Добавление в конструктор через DI:

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

Создание HttpClient

Посмотрим глубже, как происходит создание *HttpClient*



Создание HttpClient

```
public class HttpClient : HttpMessageInvoker

public class HttpMessageInvoker : IDisposable
```

Конструкторы HttpMessageInvoker

```
public HttpMessageInvoker(HttpMessageHandler
  handler, bool disposeHandler)

public HttpMessageInvoker(HttpMessageHandler
  handler) : this(handler, true)
```

Конструкторы HttpClient

```
public HttpClient(HttpMessageHandler handler,
  bool disposeHandler) : base(handler,
  disposeHandler)

public HttpClient(HttpMessageHandler handler) :
  this(handler, true)

public HttpClient() : this((HttpMessageHandler)
  new HttpClientHandler())
```

Конструкторы HttpClient

- 3 конструктора;
- Есть возможность передать *HttpMessageHandler*;
- В стандартном конструкторе *disposeHandler* равен *true*.

Dispose HttpClient

```
protected override void Dispose(bool disposing)
{
  if (disposing && !this._disposed)
  {
    this._disposed = true;
    this._pendingRequestsCts.Cancel();
    this._pendingRequestsCts.Dispose();
  }
  base.Dispose(disposing);
}
```

Dispose HttpMessageInvoker

```
public void Dispose()
{
  this.Dispose(true);
  GC.SuppressFinalize((object) this);
}
```

Dispose HttpMessageInvoker

```
protected virtual void Dispose(bool disposing)
{
  if (!disposing || this._disposed)
    return;
  this._disposed = true;
  if (!this._disposeHandler)
    return;
  this._handler.Dispose();
}
```

Dispose HttpClient

- Отменяются все повисшие запросы $\Rightarrow$ могут отменяться чужие запросы;
- Флаг *disposed* выставляется в *true*;
- *Dispose* вызывается у *HttpMessageHandler* только в случае *disposeHandler* = *true*.

IHttpClientFactory

```
public static HttpClient CreateClient(
  this IHttpClientFactory factory)
{
  return factory
    .CreateClient(DefaultName);
}
```

DefaultHttpClientFactory

```
public HttpClient CreateClient(string name)
{
  HttpClient httpClient = new
    HttpClient(this.CreateHandler(name), false);
  return httpClient;
}
```

DefaultHttpClientFactory

```
public HttpMessageHandler CreateHandler(
  string name)
{
  ActiveHandlerTrackingEntry entry =
    this._activeHandlers.GetOrAdd(name,
      this._entryFactory).Value;
  this.StartHandlerEntryTimer(entry);
  return (HttpMessageHandler) entry.Handler;
}
```

Создание с помощью HttpClientFactory

- *Dispose* не имеет эффекта;
- *HttpClientFactory* переиспользует *Handler*;
- *HttpClientFactory* выставляет таймер для *Handler*.

Выводы

- Named clients, Typed client, Refit;
- Переиспользуют *HttpMessageHandler*;
- Закрывают *HttpMessageHandler* спустя время;
- *HttpClient* не *IDisposable*.

Дополнительные улучшения в .NET Core 2.1

DelegatingHandler

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

Создание DelegatingHandler

```
public class SomeHandler : DelegatingHandler

override SendAsync(...)

var response = await base.SendAsync(request,
   cancellationToken);
```

Регистрация DelegatingHandler

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

Polly

Библиотека Polly для обработки ошибок (https://github.com/App-vNext/Polly)

Polly

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

Добавление политик

Обработка всех ответов со статус кодами 5xx 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 внутреннего HttpMessageHandler

- *AllowAutoRedirect*;
- *AutomaticDecompression*;
- *MaxAutomaticRedirections*;
- *MaxResponseHeadersLength*;
- . . .

Время жизни HttpMessageHandler

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

Выводы

- Добавились методы для удобной настройки *HttpClient*.

HttpRequestMessage и HttpResponseMessage

HttpRequestMessage

Представляет сообщение HTTP-запроса.

- *HttpMethod method*;
- *Uri requestUri*;
- *HttpRequestHeaders headers*;
- *Version version*, значение по умолчанию *Version*$(2, 0)$;
- *HttpContent content*, который является *IDisposable*;

Диспозить или нет?

Зависит от *HttpContent*. Чаще всего это *StringContent* $\Rightarrow$ можно не диспозить.

HttpResponseMessage

Представляет ответное сообщение HTTP.

- *HttpStatusCode statusCode* (есть проверка $value > 0$ и $value < 999$);
- *HttpResponseHeaders headers*;
- *string reasonPhrase*
- *HttpRequestMessage requestMessage*
- *Version version*, значение по умолчанию $Version(1, 1)$;
- *HttpContent content*, который является *IDisposable*;

Диспозить или нет?

- Для *GetAsync* и *SendAsync*;
- *HttpCompletionOption*: *ResponseContentRead*, *ResponseHeadersRead*;
- Если *ResponseContentRead*, то данные сохраняются в *MemoryStream* $\Rightarrow$ можно без диспоза;
- Иначе стоит диспозить.

Антипаттерн чтения в строку

```
var response = await
   client.GetAsync("/get-some-data");
var str = await
   response.Content.ReadAsStringAsync();
var someData =
   JsonConvert.DeserializeObject<SomeData>(str);
return someData;
```

Десериализуем из stream

```
var srz = new JsonSerializer();
var response = await
  client.GetAsync("/get-some-data");
var stream = await
  response.Content.ReadAsStreamAsync();
using (var sr = new StreamReader(stream))
using (var jsonReader = new JsonTextReader(sr))
{
  return srz.Deserialize<SomeData>(jsonReader);
}
```

Десериализуем из stream в .net core 3.0

```
var response = await
   client.GetAsync("/get-some-data");
var stream = await
   response.Content.ReadAsStreamAsync();
var someDate = await
   JsonSerializer.DeserializeAsync<SomeData>(stream
return someDate;
```

Выводы

- *HttpRequestMessage* и *HttpResponseMessage* иногда можно не диспозить;
- Лучше не использовать промежуточную строку при десериализации.

Новое в .NET Core 3.0

Поддержка HTTP/2

```
using (var request =
  new HttpRequestMessage(HttpMethod.Get, "/")
    { Version = new Version(2, 0) })
```

Поддержка HTTP/2

```
var client = new HttpClient()
{
  BaseAddress = new Uri("https://localhost:80"),
  DefaultRequestVersion = new Version(2, 0)
};
```

Регистрация gRPC Client

Схожий шаблон с *HttpClient*:

```
services.AddGrpcClient<GreeterClient>(options =>
  {
    options.BaseAddress = new
      Uri("https://localhost:5001");
  });
```

Ссылки

- https://github.com/rafaelldi/SpbDotNet-HttpClient;
- You're using HttpClient wrong and it is destabilizing your software;
- Singleton HttpClient? Beware of this serious behaviour and how to fix it;
- Beware of the .NET HttpClient;
- Подводные камни HttpClient в .NET;
- Make HTTP requests using IHttpClientFactory in ASP.NET Core.

Контакты

- Email rival.abdrakhamnov@gmail.com;
- Блог https://arcadeprogramming.com/.