Оглавление

[Оглавление 1](#_Toc175664377)

[Основы работы с сетями в C# и .NET 2](#_Toc175664378)

[Введение в сети и протоколы 2](#_Toc175664379)

[Адреса в .NET 2](#_Toc175664380)

[IPAddress 2](#_Toc175664381)

[Встроенные адреса 3](#_Toc175664382)

[Схема адресации 3](#_Toc175664383)

[Конечная точка IPEndPoint 4](#_Toc175664384)

[Адреса Uri 5](#_Toc175664385)

[System.Uri 5](#_Toc175664386)

[UriBuilder 6](#_Toc175664387)

[DNS 7](#_Toc175664388)

[Получение информации о сетевой конфигурации и сетевом трафике 8](#_Toc175664389)

[NetworkInterface и сетевые устройства 8](#_Toc175664390)

[*Получение информации о всех подключениях* 10](#_Toc175664391)

[Мониторинг трафика 12](#_Toc175664392)

[Класс Socket 14](#_Toc175664393)

[Определение сокета 14](#_Toc175664394)

[Свойства и методы сокета 16](#_Toc175664395)

[Закрытие сокета 17](#_Toc175664396)

[Протокол HTTP. Класс HttpClient и HttpListener 19](#_Toc175664397)

[Введение в протокол HTTP 19](#_Toc175664398)

[Методы HTTP 19](#_Toc175664399)

[Статусный код HTTP 20](#_Toc175664400)

[Формат запроса и ответа 21](#_Toc175664401)

[Создание HttpClient 22](#_Toc175664402)

[Создание HttpClient 22](#_Toc175664403)

[Исчерпание сокетов 22](#_Toc175664404)

[HttpClient как статический объект/синглтон 22](#_Toc175664405)

[Создание HttpClient с помощью IHttpClientFactory 23](#_Toc175664406)

[Отправка запросов с помощью HttpClient 24](#_Toc175664407)

[Отправка запроса методом SendAsync 25](#_Toc175664408)

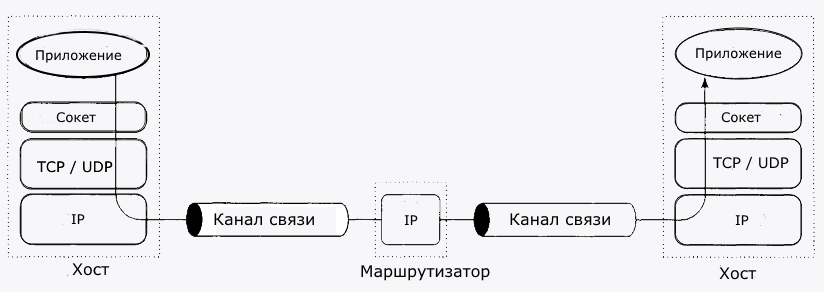
[Получение ответа 25](#_Toc175664409)

[GetAsync() 26](#_Toc175664410)

[GetStringAsync / GetByteArrayAsync / GetStreamAsync 26](#_Toc175664411)

Основы работы с сетями в C# и .NET

## Введение в сети и протоколы



**Основная функциональность фреймворка .NET по работе с сетями содержится в пакете System.Net. Также есть дополнительные пакеты:**

* **System.Net.Http: содержит функциональность по работе с протоколом HTTP**
* **System.Net.NetworkInformation: предоставляет доступ к данным о сетевом трафике и сетевых адресах, а также к прочей информации о хостах сети. Также предоставляет функциональность ping**
* **System.Net.Security: предоставляет сетевые потоки для безопасной связи между хостами**
* **System.Net.Sockets: предоставляет доступ к функциональности сокетов операционной системы**
* **System.Net.WebSockets: предоставляет доступ к реализации инфтерфейса WebSocket**
* **System.Net.Quic: содержит типы, которые реализуют протокол QUIC в соответствии со спецификацией RFC 9000**

## Адреса в .NET

### IPAddress

**В системе классов .NET ip-адрес представлен классом IPAddress из пространства System.Net.**

***Конструкторы:***

**public IPAddress (byte[] address);**

**public IPAddress (long newAddress);**

***Пример:***

**using System.Net;**

**IPAddress ip = new IPAddress(new byte[] { 127, 0, 0, 1 });**

**// или**

**IPAddress ip = new IPAddress(0x0100007F);**

**// 127.0.0.1, где 1 - 01, 0 - 00, 0 - 00, 127 - 7F**

**// расположены наоборот**

**// или через Parse**

**IPAddress ip = IPAddress.Parse("127.0.0.1");**

**// или через TryParse**

**IPAddress.TryParse("127.0.0.1", out IPAddress? ip);**

**Console.WriteLine(ip); // 127.0.0.1**

### Встроенные адреса

**IPAddress предоставляет ряд адресов по умолчанию через ряд статических свойств:**

* **Статическое свойство Loopback: возвращает объект IPAddress для адреса 127.0.0.1**
* **Статическое свойство Any: возвращает объект IPAddress для адреса 0.0.0.0**
* **Статическое свойство Broadcast: возвращает объект IPAddress для адреса 255.255.255.255**

**IPAddress anyIp = IPAddress.Any;**

**IPAddress localIp = IPAddress.Loopback;**

**IPAddress broadcastIp = IPAddress.Broadcast;**

### Схема адресации

**Свойство класса IPAddress AddressFamily может принимать значения:**

* **AppleTalk: адрес AppleTalk**
* **Atm: адрес собственных служб ATM**
* **Banyan: адрес Banyan**
* **Ccitt: адреса протоколов CCITT, таких как протокол X25**
* **Chaos: адрес протоколов MIT CHAOS**
* **Cluster: адрес кластерных продуктов корпорации Майкрософт**
* **ControllerAreaNetwork: сетевой адрес области контроллера**
* **DataKit: адрес протоколов Datakit**
* **DataLink: адрес интерфейса прямого канала передачи данных**
* **DecNet: адрес DECnet**
* **Ecma: адрес ЕСМА (European Computer Manufacturers Association — европейская ассоциация производителей компьютеров)**
* **FireFox: адрес FireFox**
* **HyperChannel: адрес NSC Hyperchannel**
* **Ieee12844: адрес рабочей группы IEEE 12844**
* **ImpLink: адрес ARPANET IMP**
* **InterNetwork: IPv4-адрес**
* **InterNetworkV6: IPv6-адрес**
* **Ipx: IPX- или SPX-адрес**
* **Irda: IrDA-адрес**
* **Iso: адрес протоколов ISO**
* **Lat: LAT-адрес**
* **Max: MAX-адрес**
* **NetBios: адрес NetBios**
* **NetworkDesigners: адрес шлюзовых протоколов Network Designers OSI**
* **NS: адрес протоколов Xerox NS**
* **Osi: адрес протоколов OSI**
* **Packet: адрес пакета нижнего уровня**
* **Pup: адрес протоколов PUP**
* **Sna: адрес IBM SNA**
* **Unix: локальный адрес Unix для узла**
* **Unknown: семейство неизвестных адресов**
* **Unspecified: семейство неуказанных адресов**
* **VoiceView: адрес VoiceView**

***Пример:***

**IPAddress ip = new IPAddress(new byte[] { 127, 0, 0, 1 });**

**Console.WriteLine(ip.AddressFamily); // InterNetwork**

### Конечная точка IPEndPoint

**Конечная точка представляет объединение IP-адреса и порта и в .NET представляет класс IPEndPoint из пространства имен System.Net.**

**Конструкторы:**

**public IPEndPoint (long address, int port);**

**public IPEndPoint (IPAddress address, int port);**

***Пример:***

**using System.Net;**

**IPAddress ip = IPAddress.Parse("127.0.0.1");**

**IPEndPoint endpoint = new IPEndPoint(ip, 8080);**

**// или**

**IPEndPoint endPoint = IPEndPoint.Parse("127.0.0.1:8080");**

**// или**

**IPEndPoint.TryParse("127.0.0.1:8080", out IPEndPoint? endPoint);**

**Console.WriteLine(endpoint); // 127.0.0.1:8080**

**Свойства IPEndPoint предоставляют доступ к информации о конечной точке:**

* **Address: возвращает или устанавливает IP-адрес**
* **AddressFamily: возвращает схему адресации, которую применяет IP-адрес**
* **Port: возвращает или устанавливает номер порта**

**IPEndPoint endpoint = IPEndPoint.Parse("127.0.0.1:8080");**

**Console.WriteLine(endpoint.Address); // 127.0.0.1**

**Console.WriteLine(endpoint.Port); // 8080**

## Адреса Uri

**Uniform Resource Identifier (URI) - Единый идентификатор ресурса**

**Uniform Resource Locator (URL) - Единый локатор ресурсов - подвид URI**

**Общая форма адреса URL:**

**//[authority/]path[?query][#fragment]**

**authority включает домен и порт, а также возможные учетные данные пользователя как логин и пароль:**

**//[access\_credentials][@]host\_domain[:port]**

**Символ @ отделяет учетные данные от домена. Параметр access\_credentials, который описывает учетные данные, задается в виде**

**[user\_id][:][password]**

**Параметр user\_id предоставляет логин, а password - пароль для доступа к ресурсу. Они отделяются друг от друга двоеточием.**

**Компонент query имеет следующую форму:**

**?[parameter1=value2][(;|&)parameter2=value2]...**

**В конце после символа решетки # может указываться фрагмент - необязательная строка для идентификации компонентов внутри URL. Обычно применяется в веб-браузерах для навигации по частям веб-страницы.**

**При отправке сообщения по URI компьютер обращается к DNS (Domain Name System - система доменных имен), которая сопоставляет URI и IP.**

### System.Uri

**Uri uri = new("http://website.com/");**

**// или абсолютный адрес**

**Uri uri = new Uri("http://website.com/", UriKind.Absolute);**

**// относительный адрес**

**Uri uri = new Uri("sharp/net", UriKind.Relative);**

**// абсолютный или относительный**

**Uri uri = new Uri("http://website.com/sharp/net",**

**UriKind.RelativeOrAbsolute);**

**// или**

**Uri.TryCreate(string url,**

**UriCreationOptions options, // new UriCreationOptions()**

**out Uri? newUri);**

**Базовые свойства:**

* **AbsoluteUri: возвращает абсолютный адрес URI**

**https://user:password@www.somesite.com/home/index?q1=v1&q2=v2#fragmentName**

* **Authority: возвращает либо имя хоста в соответствии с системой доменных имен DNS, либо IP-адрес и порт сервера.**

**www.somesite.com**

* **Fragment: возвращает фрагмент адреса URI.**

**#fragmentName**

* **Host: возвращает хост.**

**www.somesite.com**

* **IsAbsoluteUri: возвращает true, если адрес абсолютный.**
* **IsDefaultPort: возвращает true, если адрес URI использует порт по умолчанию для своей схемы.**
* **IsFile: возвращает true, если адрес Uri представляет адрес файла.**
* **IsLoopback: возвращает true, если адрес Uri указывает на локальный хост.**
* **OriginalString: возвращает оригинальную строку адреса URI, которая передана в конструктор Uri.**
* **PathAndQuery: возвращает значения свойств AbsolutePath и Query, разделяя их вопросительным знаком (?).**

**/home/index?q1=v1&q2=v2**

* **Port: возвращает номер порта для текущего адреса URI.**

**443**

* **Query: возвращает строку запроса из текущего адреса URI.**

**?q1=v1&q2=v2**

* **Scheme: возвращает схему текущего адреса URI.**

**https**

* **Segments: возвращает массив сегментов пути для текущего адреса URI. Каждый сегмент представляет часть пути, которая ограничена слешами**

**["/", "home/", "index" ]**

* **UserInfo: возвращает имя и пароль пользователя.**

**user:password**

### UriBuilder

**Также для создания адреса можно использовать System.UriBuilder.**

**Свойства UriBuilder:**

* **Uri: возвращает созданный экземпляр Uri.**

**https://user:password@www.somesite.com/home/index?q1=v1&q2=v2#fragmentName**

* **Fragment: возвращает или задает фрагмент URI.**

**#fragmentName**

* **Host: возвращает или задает имя хоста или IP-адрес сервера.**

**www.somesite.com**

* **Password: возвращает или задает пароль.**

**password**

* **Path: возвращает или задает путь к ресурсу, на который ссылается URI.**

**/home/index**

* **Port: возвращает или задает номер порта URI.**

**443**

* **Query: возвращает или задает строку запроса.**

**?q1=v1&q2=v2**

* **Scheme: возвращает или задает схему URI.**

**https**

* **UserName: возвращает или задает имя пользователя.**

**user**

**public UriBuilder (string? scheme,**

**string? host,**

**int port,**

**string? path,**

**string? extraValue)**

## DNS

**Физичеcки dns - это сеть серверов, в которой размещен каталог, где доменные имена сопоставлены с IP-адресами. При отправке запроса по имени домена, например, при вводе в адресной строке браузере адреса "https://metanit.com" и отправке запроса, вначале идет запрос на один из этих dns-серверов. Однако, если текущий сервер не может разрешить доменное имя, он перенаправит его вверх по иерархии серверов имен в более общий родительский домен. Каждый сервер будет проверять данное доменное имя и искать домен в своем собственном каталоге, где сопоставляются домены и IP-адреса. Если сервер или один из подчиненных серверов смог сопоставить домен с одним из ip-адресов, то сервер просто заменяет доменное имя в запросе на IP-адрес, с которым он сопоставляется, и соответствующим образом пересылает запрос.**

**Для идентификации ip-адресов .NET предоставляет статический класс Dns, который располагается в пространстве имен System.Net.**

**Методы:**

* **GetHostAddresses (string hostNameOrAddress): запрашивает DNS-сервер и возвращает все ip-адреса для определенного имени хоста в виде массива System.Net.IPAddress[]. Данный метод имеет асинхронного двойника в виде метода GetHostAddressesAsync(string hostNameOrAddress)**
* **GetHostEntry (string hostNameOrAddress): запрашивает DNS-сервер и возвращает объект IPHostEntry для определенного имени хоста или ip-адреса. Данный метод имеет асинхронного двойника в виде метода GetHostEntryAsync(string hostNameOrAddress)**
* **GetHostName(): возвращает имя хоста для локального компьютера**

***Пример:***

**using System.Net;**

**var googleEntry = await Dns.GetHostEntryAsync("google.com");**

**// google.com**

**foreach (var ip in googleEntry.AddressList)**

**{**

**// 142.250.74.78**

**}**

**// или**

**var googleIps = await Dns.GetHostAddressesAsync("google.com");**

**foreach (var ip in googleIps)**

**{**

**// 142.250.74.78**

**}**

## Получение информации о сетевой конфигурации и сетевом трафике

### NetworkInterface и сетевые устройства

**Для получения информации о сетевых устройствах/интерфейсах на текущей машине можно использовать абстрактный класс NetworkInterface из пространства имен System.Net.NetworkInformation. Собственно класс NetworkInterface и представляет отдельный сетевой интерефейс компьютера.**

**Свойства NetworkInterface:**

* **Description: возвращает описание сетевого интерфейса**
* **Id: возвращает идентификатор сетевого адаптера**
* **Name: возвращает название сетевого адаптера**
* **NetworkInterfaceType: тип сетевого интерефейса в виде константы перечисления System.Net.NetworkInformation.NetworkInterfaceType**
* **OperationalStatus: возвращает текущий статус операций**
* **Speed: возвращает скорость сетевого адаптера в виде количества битов в секунду**

**Методы NetworkInterface:**

* **GetAllNetworkInterfaces(): возвращает массив объектов NetworkInterface, где каждый элемент представляет сетевой интерфейс на локальной машине (статический метод)**
* **GetIPProperties(): возвращает объект IPInterfaceProperties, который представляет все свойства сетевого интерфейса**
* **GetIPStatistics(): возвращает статистику для текущего сетевого интерфейса в виде объекта , который хранит статистику в свойствах:**
  + **BytesReceived: возвращает количество байтов, полученных интерфейсом.**
  + **BytesSent: возвращает количество байтов, отправленных интерфейсом.**
  + **IncomingPacketsDiscarded: возвращает количество входящих пакетов, которые были удалены.**
  + **IncomingPacketsWithErrors: возвращает количество входящих пакетов с ошибками.**
  + **IncomingUnknownProtocolPackets: возвращает количество входящих пакетов с неизвестным протоколом, которые были получены в интерфейсе.**
  + **NonUnicastPacketsReceived: возвращает количество неодноадресных пакетов, полученных интерфейсом.**
  + **NonUnicastPacketsSent: возвращает количество неодноадресных пакетов, отправленных интерфейсом.**
  + **OutgoingPacketsDiscarded: возвращает количество исходящих пакетов, которые были удалены.**
  + **OutgoingPacketsWithErrors: возвращает количество исходящих пакетов с ошибками.**
  + **OutputQueueLength: возвращает длину очереди вывода.**
  + **UnicastPacketsReceived: возвращает количество одноадресных пакетов, полученных интерфейсом.**
  + **UnicastPacketsSent: возвращает количество одноадресных пакетов, отправленных интерфейсом.**
* **GetIsNetworkAvailable(): возвращает true, если доступно какое-либо сетевое подключение (статический метод)**
* **GetPhysicalAddress(): возвращает физический адрес сетевого интерфейса**

***Пример:***

**using System.Net.NetworkInformation;**

**var adapters = NetworkInterface.GetAllNetworkInterfaces();**

**Console.WriteLine($"Обнаружено {adapters.Length} устройств");**

**foreach (NetworkInterface adapter in adapters)**

**{**

**Console.WriteLine("==================================================");**

**Console.WriteLine();**

**Console.WriteLine($"ID устройства: ------------- {adapter.Id}");**

**Console.WriteLine($"Имя устройства: ------------ {adapter.Name}");**

**Console.WriteLine($"Описание: ------------------ {adapter.Description}");**

**Console.WriteLine($"Тип интерфейса: ----{adapter.NetworkInterfaceType}");**

**Console.WriteLine($"Физический адрес:-- {adapter.GetPhysicalAddress()}");**

**Console.WriteLine($"Статус: ---------------{adapter.OperationalStatus}");**

**Console.WriteLine($"Скорость: ------------------ {adapter.Speed}");**

**IPInterfaceStatistics stats = adapter.GetIPStatistics();**

**Console.WriteLine($"Получено: ----------------- {stats.BytesReceived}");**

**Console.WriteLine($"Отправлено: --------------- {stats.BytesSent}");**

**}**

**/\***

**Output:**

**Обнаружено 6 устройств**

**=====================================================================**

**ID устройства: ------------- {D5CA0C59-7CDF-4B7C-9502-A7296BCF0158}**

**Имя устройства: ------------ vEthernet (WSL)**

**Описание: ------------------ Hyper-V Virtual Ethernet Adapter**

**Тип интерфейса: ------------ Ethernet**

**Физический адрес: ---------- 0D135B218A94**

**Статус: -------------------- Up**

**Скорость: ------------------ 10000000000**

**Получено: ----------------- 8403404**

**Отправлено: --------------- 515439337**

**=====================================================================**

**ID устройства: ------------- {E5786DCA-6A1D-15A1-B418-E19DA6C04076}**

**Имя устройства: ------------ Подключение по локальной сети\* 1**

**Описание: ------------------ Microsoft Wi-Fi Direct Virtual Adapter**

**Тип интерфейса: ------------ Wireless80211**

**Физический адрес: ---------- 0ACC270A7951**

**Статус: -------------------- Down**

**Скорость: ------------------ -1**

**Получено: ----------------- 0**

**Отправлено: --------------- 0**

**\*/**

## Получение информации о всех подключениях

**Класс IPGlobalProperties позволяет получить детальную информацию по сетевому трафику и его конфигурации.**

**Методы:**

* **GetActiveTcpConnections(): возвращает сведения о TCP-подключениях (массив TcpConnectionInformation[]). TcpConnectionInformation имеют 3 свойства:**
  + **LocalEndPoint: локальная конечная точка, через которую текущий компьютер установил TCP-подключение с удаленным хостом**
  + **RemoteEndPoint: адрес удаленного хоста, с которым установлено TCP-подключение**
  + **State: состояние TCP-подключения в виде одной из констант перечисления TcpState:**
  + **Closed: TCP-подключение закрыто**
  + **CloseWait: локальная конечная точка ТСР-подключения ожидает от локального пользователя запрос на разрыв подключения**
  + **Closing: локальная конечная точка ТСР-подключения ожидает подтверждение ранее отправленного запроса на разрыв подключения**
  + **DeleteTcb: удаляется буфер управления передачей (TCB) для ТСР-подключения**
  + **Established: TCP-подключение установлено.**
  + **FinWait1: локальная конечная точка ТСР-подключения ожидает от удаленной конечной точки запрос на разрыв подключения или подтверждение ранее отправленного запроса на разрыв подключения.**
  + **FinWait2: локальная конечная точка ТСР-подключения ожидает от удаленной конечной точки запрос на разрыв подключения.**
  + **LastAck: локальная конечная точка ТСР-подключения ожидает окончательное подтверждение ранее отправленного запроса на разрыв подключения.**
  + **Listen: локальная конечная точка ТСР-подключения прослушивает запросы на подключение**
  + **SynReceived: локальная конечная точка ТСР-подключения отправила и получила запрос на подключение, и ожидает подтверждения.**
  + **SynSent: локальная конечная точка ТСР-подключения отправила удаленной конечной точке заголовок сегмента с установленным управляющим битом синхронизации (SYN) и ожидает соответствующий запрос на подключение.**
  + **TimeWait: локальная конечная точка ТСР-подключения ожидает в течение достаточного времени, чтобы обеспечить получение удаленной точкой подтверждения ее запроса на разрыв подключения.**
  + **Unknown: неизвестное состояние ТСР-подключения .**
* **GetActiveTcpListeners(): возвращает массив адресов TCP-слушателей (массив IPEndPoint[])**
* **GetActiveUdpListeners(): возвращает массив адресов UDP-слушателей (массив IPEndPoint[]).**
* **GetIcmpV4Statistics(): возвращает статистику протокола ICMPv4 (объект IcmpV4Statistics)**
* **GetIcmpV6Statistics(): возвращает статистику протокола ICMPv6 (объект IcmpV6Statistics)**
* **GetIPv4GlobalStatistics(): возвращает статистику протокола IPv4 (объект IPGlobalStatistics)**
* **GetIPv6GlobalStatistics(): возвращает статистику протокола IPv6 (объект IPGlobalStatistics)**
* **GetIPGlobalProperties(): возвращает объект IPGlobalProperties, который предоставляет информацию по сетевой конфигурации и статистику трафика (статический метод)**
* **GetUnicastAddresses() / GetUnicastAddressesAsync(): возвращает таблицу IP-адресов одноадресной рассылки (объект UnicastIPAddressInformationCollection)**

***Пример:***

using System.Net.NetworkInformation;

var ipProps = IPGlobalProperties.GetIPGlobalProperties();

var tcpConnections = ipProps.GetActiveTcpConnections();

Console.WriteLine($"Всего {tcpConnections.Length} активных TCP-подключений");

Console.WriteLine();

foreach (var connection in tcpConnections)

{

Console.WriteLine("=============================================");

Console.WriteLine($"Локальный адрес: {connection.LocalEndPoint.Address}:{connection.LocalEndPoint.Port}");

Console.WriteLine($"Адрес удаленного хоста: {connection.RemoteEndPoint.Address}:{connection.RemoteEndPoint.Port}");

Console.WriteLine($"Состояние подключения: {connection.State}");

}

/\*

Output:

Всего 4 активных TCP-подключений

=============================================

Локальный адрес: 192.168.0.112:52308

Адрес удаленного хоста: 13.69.68.64:443

Состояние подключения: TimeWait

=============================================

Локальный адрес: 192.168.0.112:52310

Адрес удаленного хоста: 23.96.225.71:443

Состояние подключения: Established

=============================================

Локальный адрес: 192.168.0.112:58623

Адрес удаленного хоста: 20.54.37.64:443

Состояние подключения: Established

=============================================

Локальный адрес: 192.168.0.112:58631

Адрес удаленного хоста: 52.166.127.37:9354

Состояние подключения: Established

\*/

### Мониторинг трафика

Методы GetIPv4GlobalStatistics() и GetIPv6GlobalStatistics() класса IPGlobalProperties возвращают объект IPGlobalStatistics, который предоставляет доступ к статистике сетевого трафика с помощью следующих свойств:

* DefaultTtl: возвращает срок жизни (TTL) IP-пакетов.
* ForwardingEnabled: возвращает значение bool, которое указывает, разрешена ли переадресация IP-пакетов.
* NumberOfInterfaces: возвращает количество сетевых интерфейсов.
* NumberOfIPAddresses: возвращает количество IP-адресов, назначенных локальному компьютеру.
* NumberOfRoutes: возвращает количество маршрутов в таблице IP-маршрутизации.
* OutputPacketRequests: возвращает количество исходящих IP-пакетов.
* OutputPacketRoutingDiscards: возвращает количество маршрутов, удаленных из таблицы маршрутизации.
* OutputPacketsDiscarded: возвращает количество отправленных отброшенных IP-пакетов
* OutputPacketsWithNoRoute: возвращает количество IP-пакетов, для которых локальному компьютеру не удалось определить маршрут к адресу назначения.
* PacketFragmentFailures: возвращает количество IP-пакетов, которые не удалось фрагментировать.
* PacketReassembliesRequired: возвращает количество IP-пакетов, для которых требовалась восстановление.
* PacketReassemblyFailures: возвращает количество IP-пакетов, которые не были успешно восстановлены.
* PacketReassemblyTimeout: возвращает максимальное время, в течение которого должны поступить все фрагменты IP-пакета.
* PacketsFragmented: возвращает количество фрагментированных IP-пакетов.
* PacketsReassembled: возвращает количество собранных IP-пакетов.
* ReceivedPackets: возвращает количество полученных IP-пакетов.
* ReceivedPacketsDelivered: возвращает количество доставленных IP-пакетов.
* ReceivedPacketsDiscarded: возвращает количество отброшенных полученных IP-пакетов, которые были удалены.
* ReceivedPacketsForwarded: возвращает количество переадресованных IP-пакетов.
* ReceivedPacketsWithAddressErrors: возвращает количество полученных IP-пакетов с ошибками в адресе.
* ReceivedPacketsWithHeadersErrors: возвращает количество полученных IP-пакетов с ошибками в заголовке.
* ReceivedPacketsWithUnknownProtocol: возвращает количество IP-пакетов с неизвестным протоколом в заголовке, полученных локальным компьютером.

*Пример:*

using System.Net.NetworkInformation;

var ipProps = IPGlobalProperties.GetIPGlobalProperties();

var ipStats = ipProps.GetIPv4GlobalStatistics();

Console.WriteLine($"Входящие пакеты: {ipStats.ReceivedPackets}");

Console.WriteLine($"Исходящие пакеты: {ipStats.OutputPacketRequests}");

Console.WriteLine($"Отброшено входящих пакетов: {ipStats.ReceivedPacketsDiscarded}");

Console.WriteLine($"Отброшено исходящих пакетов: {ipStats.OutputPacketsDiscarded}");

Console.WriteLine($"Ошибки фрагментации: {ipStats.PacketFragmentFailures}");

Console.WriteLine($"Ошибки восстановления пакетов: {ipStats.PacketReassemblyFailures}");

/\*

Output:

Входящие пакеты: 10428590

Исходящие пакеты: 6745275

Отброшено входящих пакетов: 43737

Отброшено исходящих пакетов: 2

Ошибки фрагментации: 0

Ошибки восстановления пакетов: 0

\*/

## Класс Socket

**В основе межсетевых взаимодействий по сетевых протоколам TCP и UDP лежат сокеты. Сокет предоставляет интерфейс доступа к определенному порту определенного хоста. То есть через сокет один хост может обращаться к приложению на другом хосте. В .NET сокеты представлены классом Socket из пространства имен System.NET.Sockets, который предоставляет низкоуровневый интерфейс для приема и отправки сообщений по сети.**

### Определение сокета

**Конструкторы:**

* **Socket(AddressFamily, SocketType, ProtocolType): создает сокет, используя указанные семейство адресов, тип сокета и протокол.**
  + **AddressFamily - перечисление, которое задает схему адресации, которую может использовать сокет. Данное перечисление содержит 33 константы. Наиболее используемые:**
    - **InterNetwork: адрес по протоколу IPv4**
    - **InterNetworkV6: адрес по протоколу IPv6**
    - **Ipx: адрес IPX или SPX**
    - **NetBios: адрес NetBios**
  + **SocketType - перечисление, которое устанавливает тип сокета. Может принимать следующие значения:**
    - **Dgram: сокет будет получать и отправлять дейтаграммы по протоколу Udp. Данный тип сокета работает в связке с типом протокола - Udp и значением AddressFamily.InterNetwork**
    - **Raw: сокет имеет доступ к нижележащему протоколу транспортного уровня и может использовать для передачи сообщений такие протоколы, как ICMP и IGMP**
    - **Rdm: сокет может взаимодействовать с удаленными хостами без установки постоянного подключения. В случае, если отправленные сокетом сообщения невозможно доставить, то сокет получит об этом уведомление**
    - **Seqpacket: обеспечивает надежную двустороннюю передачу данных с установкой постоянного подключения**
    - **Stream: обеспечивает надежную двустороннюю передачу данных с установкой постоянного подключения. Для связи используется протокол TCP, поэтому этот тип сокета используется в паре с типом протокола Tcp и значением AddressFamily.InterNetwork**
    - **Unknown: адрес NetBios**
  + **ProtocolType - перечисление, которое устанавливает тип используемого протокола. Может принимать следующие значения:**
    - **Ggp**
    - **Icmp**
    - **IcmpV6**
    - **Idp**
    - **Igmp**
    - **IP**
    - **IPSecAuthenticationHeader (Заголовок IPv6 AH)**
    - **IPSecEncapsulatingSecurityPayload (Заголовок IPv6 ESP)**
    - **IPv4**
    - **IPv6**
    - **IPv6DestinationOptions (Заголовок IPv6 Destination Options)**
    - **IPv6FragmentHeader (Заголовок IPv6 Fragment)**
    - **IPv6HopByHopOptions (Заголовок IPv6 Hop by Hop Options)**
    - **IPv6NoNextHeader (Заголовок IPv6 No next)**
    - **IPv6RoutingHeader (Заголовок IPv6 Routing)**
    - **Ipx**
    - **ND**
    - **Pup**
    - **Raw**
    - **Spx**
    - **SpxII**
    - **Tcp**
    - **Udp**
    - **Unknown (неизвестный протокол)**
    - **Unspecified (неуказанный протокол)**

**Каждое значение представляет соответствующий протокол, но наиболее используемыми являются Tcp и Udp.**

* **Socket(SafeSocketHandle): создает сокет с помощью дескриптора сокета - объекта SafeSocketHandle.**
* **Socket(SocketInformation): создает сокет, используя структуру SocketInformation.**
* **Socket(SocketType, ProtocolType): создает сокет, используя указанные тип сокета и протокол.**

***Пример:***

**using System.Net.Sockets;**

**// Tcp-протокол**

**Socket socket = new Socket(AddressFamily.InterNetwork,**

**SocketType.Stream,**

**ProtocolType.Tcp);**

**// Udp-протокол**

**Socket socket = new Socket(AddressFamily.InterNetwork,**

**SocketType.Dgram,**

**ProtocolType.Udp);**

### Свойства и методы сокета

**Свойства:**

* **AddressFamily: представляет схему адресации, используемую сокетом, в виде перечисления AddressFamily**
* **Available: возвращает объем данных, полученных от подключенного хоста и доступных для чтения**
* **Connected: возвращает true, если сокет подключен к удаленному хосту**
* **LocalEndPoint: возвращает локальную точку (объект типа EndPoint), по которой запущен сокет и по которой он принимает данные**
* **ProtocolType: возвращает тип протокола в виде значения перечисления ProtocolType**
* **RemoteEndPoint: возвращает адрес удаленного хоста, к которому подключен сокет (объект типа EndPoint)**
* **SocketType: возвращает тип сокета в виде значения перечисления SocketType**

**Методы:**

* **Accept() / AcceptAsync(): создает новый объект Socket для обработки входящего подключения**
* **Bind(): связывает объект Socket с локальной конечной точкой**
* **Close(): закрывает сокет**
* **Connect() / ConnectAsync: устанавливает соединение с удаленным хостом**
* **Listen(): начинает прослушивание входящих запросов**
* **Poll(): определяет состояние сокета**
* **Receive() / ReceiveAsync: получает данные**
* **ReceiveFrom() / ReceiveFromAsync(): получает данные и сохраняет конечную точку, от которой получены данные**
* **Send() / SendAsync(): отправляет данные**
* **SendTo() / SendToAsync(): отправляет данные на определенную конечную точку**
* **Shutdown(): блокирует на сокете прием и/или отправку данных. Принимает в качестве параметра значение из перечисления SocketShutdown:**
  + **Send: блокируется отправка данных**
  + **Receive: блокируется получение данных**
  + **Both: блокируются отправка и получение данных**

**В зависимости от применяемого протокола (TCP, UDP и т.д.) общий принцип работы с сокетами будет немного различаться.**

**При применении протокола, который требует установление соединения, например, TCP, сервер должен вызвать метод Bind для установки точки для прослушивания входящих подключений и затем запустить прослушивание подключений с помощью метода Listen. Далее с помощью метода Accept можно получить входящие запросы на подключение в виде объекта Socket, который используется для взаимодействия с удаленным узла. У полученного объекта Socket вызываются методы Send и Receive соответственно для отправки и получения данных. Если необходимо подключиться к серверу, то вызывается метод Connect. Для обмена данными с сервером также применяются методы Send или Receive.**

**Если применяется протокол, для которого не требуется установление соединения, например, UDP, то после вызова метода Bind не надо вызывать метод Listen. И в этом случае для приема данных используется метод ReceiveFrom() / ReceiveFromAsync(), а для отправки данных - метод SendTo() / SendToAsync().**

### Закрытие сокета

**Socket socket = new Socket(AddressFamily.InterNetwork,**

**SocketType.Stream,**

**ProtocolType.Tcp);**

**// работа с сокетом socket ...**

**socket.Close();**

**// или можно использовать конструкцию using**

**using (Socket socket = new Socket(AddressFamily.InterNetwork,**

**SocketType.Stream,**

**ProtocolType.Tcp))**

**{**

**// работа с сокетом socket ...**

**}**

**При работе с протоколами, ориентированными на установку соединения, например, TCP, Microsoft рекомендует перед методом Close вызывать метод Shutdown, который блокирует примем и отправку данных для сокета и гарантирует, что все данные будут получены и отправлены перед закрытием сокета.**

**Socket socket = new Socket(AddressFamily.InterNetwork,**

**SocketType.Stream,**

**ProtocolType.Tcp);**

**try**

**{**

**// работа с сокетом socket ...**

**socket.Shutdown(SocketShutdown.Both);**

**}**

**catch(Exception e)**

**{**

**...**

**}**

**finally**

**{**

**socket.Close();**

**}**

Протокол HTTP. Класс HttpClient и HttpListener

## Введение в протокол HTTP

**HTTP (Hypertext Transfer Protocol) представляет протокол для запроса ресурсов с веб-сервера. В .NET основной функционал по работе с протоколом HTTP сосредоточен в простанстве имен System.Net.Http, среди основных классов в котором следует отметить класс HttpClient (класс, который позволяет отправлять запросы к веб-ресурсам) и класс HttpListener, который выполняет роль веб-сервера.**

### Методы HTTP

**Когда клиент собирается сделать запрос на сервер, данный клиент должен определить метод ответа сервера на запрос. Метод HTTP описывает действия сервера при обработке запроса. Основные из этих методов:**

* **OPTIONS: возвращает список остальных методов HTTP, которые поддерживает сервер для указанного адреса URL**
* **TRACE: служебный метод, который просто повторяет исходный запрос, полученный сервером. Полезен для идентификации любых изменений, внесенных в запрос объектами в сети, пока запрос находится в пути.**
* **CONNECT: устанавливают туннель TCP/IP между исходным хостом и удаленным хостом.**
* **GET: извлекает копию ресурса по URL-адресу, на который был отправлен HTTP-запрос.**
* **HEAD: так же как и GET, извлекает копию ресурса по URL, только ожидает получения одних заголовков без тела ответа**
* **POST: предназначен для отправки данных в теле запроса для их сохранения в виде нового ресурса на сервере**
* **PUT: предназначен для отправки данных в теле запроса для изменения уже имеющегося ресурса на сервере**
* **PATCH: предназначен для отправки данных в теле запроса для частичного изменения уже имеющегося ресурса на сервере**
* **DELETE: предназначен для удаления ресурса по указанному адресу URL**

**В .NET для представления метода HTTP в пространстве имен System.Net.Http определен класс HttpMethod. Для определения в приложении запроса определенного типа класс предоставляет ряд статических свойств, который возвращают объект HttpMethod, представляющий определенный тип запросов:**

* **HttpMethod.Delete**
* **HttpMethod.Get**
* **HttpMethod.Head**
* **HttpMethod.Options**
* **HttpMethod.Patch**
* **HttpMethod.Post**
* **HttpMethod.Put**
* **HttpMethod.Trace**

**Если используется HTTP-метод, который не покрывается этими свойствами, то можно использовать конструктор, в который передается название метода:**

**HttpMethod customMethod = new HttpMethod("CUSTOM");**

### Статусный код HTTP

**При ответе сервер устанавливает статусный код HTTP, который указывает на статус обработки запроса. По умолчанию это 3-цифровой код, где первая цифра указывает на общий характер ответа, а вторая и третья цифра конкретизируют статус. Есть пять групп статусных кодов:**

1. **1XX: информационные коды, которые указывают, что запрос получен, и его обработка продолжается**
2. **2XX: коды, которые указывают на успешную обработку запроса**
3. **3XX: коды переадресации**
4. **4XX: коды, которые указывают, что запрос содержит ошибки. То есть ошибка возникла на стороне клиента, который отправил запрос**
5. **5XX: коды, которые указывают, что в процессе обработки запроса возникла ошибка на сервере. То есть запрос от клиента корректен, а проблема на стороне сервера**

**В .NET за представление статусных кодов отвечает перечисление System.Net.HttpStatusCode, которое определяет для большинства статусных кодов соответствующие константы:**

* **Continue (100)**
* **SwitchingProtocols (101)**
* **Processing (102)**
* **EarlyHints (103)**
* **OK (200)**
* **Created (201)**
* **Accepted (202)**
* **NonAuthoritativeInformation (203)**
* **NoContent (204)**
* **ResetContent (205)**
* **PartialContent (206)**
* **MultiStatus (207)**
* **AlreadyReported (208)**
* **IMUsed (226)**
* **Ambiguous (300)**
* **MultipleChoices (300)**
* **Moved (301)**
* **MovedPermanently (301)**
* **Found (302)**
* **Redirect (302)**
* **SeeOther (303)**
* **RedirectMethod (303)**
* **NotModified (304)**
* **UseProxy (305)**
* **Unused (306)**
* **TemporaryRedirect (307)**
* **RedirectKeepVerb (307)**
* **PermanentRedirect (308)**
* **BadRequest (400)**
* **Unauthorized (401)**
* **PaymentRequired (402)**
* **Forbidden (403)**
* **NotFound (404)**
* **MethodNotAllowed (405)**
* **NotAcceptable (406)**
* **ProxyAuthenticationRequired (407)**
* **RequestTimeout (408)**
* **Conflict (409)**
* **Gone (410)**
* **LengthRequired (411)**
* **PreconditionFailed (412)**
* **RequestEntityTooLarge (413)**
* **RequestUriTooLong (414)**
* **UnsupportedMediaType (415)**
* **RequestedRangeNotSatisfiable (416)**
* **ExpectationFailed (417)**
* **MisdirectedRequest (421)**
* **UnprocessableEntity (422)**
* **Locked (423)**
* **FailedDependency (424)**
* **UpgradeRequired (426)**
* **PreconditionRequired (428)**
* **TooManyRequests (429)**
* **RequestHeaderFieldsTooLarge (431)**
* **UnavailableForLegalReasons (451)**
* **InternalServerError (500)**
* **NotImplemented (501)**
* **BadGateway (502)**
* **ServiceUnavailable (503)**
* **GatewayTimeout (504)**
* **HttpVersionNotSupported (505)**
* **VariantAlsoNegotiates (506)**
* **InsufficientStorage (507)**
* **LoopDetected (508)**
* **NotExtended (510)**
* **NetworkAuthenticationRequired (511)**

### Формат запроса и ответа

***Пример запроса от клиента:***

**GET /users/id/12 HTTP/1.1**

**// Метод, путь к ресурсу, протокол**

**Accept: application/json**

**Host: www.google.com\r\n**

**Connection: Close\r\n\r\n**

**// серия заголовков**

***Пример ответа от сервера:***

**HTTP/1.1 401 Bad Request**

**// протокол, статусный код**

**// далее идут заголовки с метаданными,**

**// которые указывают, как парсить ответ**

**// опционально тело ответа**

## Создание HttpClient

**Для отправки HTTP-запросов в .NET применяется класс HttpClient из пространства имен System.Net.Http.**

### Создание HttpClient

***Конструкторы:***

**public HttpClient (System.Net.Http.HttpMessageHandler handler);**

**HttpMessageHandler - абстрактный класс, реализующий метод Dispose().**

**public HttpClient (System.Net.Http.HttpMessageHandler handler, bool disposeHandler);**

**Если disposeHandler == true, HttpMessageHandler удаляется вместе с вызовом HttpClient.Dispose(). Если же мы хотим и после удаления HttpClient продолжать использовать объект HttpMessageHandler, то этому параметру следует передать значение false.**

**public HttpClient ();**

**// эквивалентно:**

**// HttpClient(new HttpClientHandler(), true),**

**// где HttpClientHandler - класс-наследник HttpMessageHandler**

### Исчерпание сокетов

**HttpClient нацелен на многоразовое использование в течение всей жизни приложения. Создание отдельного объекта для каждого запроса может привести к исчерпанию количества доступных сокетов и приведет к ошибкам SocketException.**

**Чтобы избежать проблемы нехватки сокетов, Microsoft рекомендует для определения HttpClient один из следующих подходов:**

* **Долговременные экземпляры HttpClient в виде статических объектов или синглтонов, которые существуют в течение всей жизни приложения**
* **Кратковременные экземпляры HttpClient, созданные с помощью фабрики IHttpClientFactory**

### HttpClient как статический объект/синглтон

***Пример:***

**class Program**

**{**

**static HttpClient client = new HttpClient();**

**static async Task Main(string[] args)**

**{**

**// использование HttpClient**

**}**

**}**

**Но в этом случае HttpClient устанавливает записи DNS только при создании подключения. Он не отслеживает срок жизни (TTL), указанный DNS-сервером для определенной dns-записи. Эта запись позволяет определить адрес хоста для создания запроса. Если эти записи DNS постоянно меняются, то клиент HttpClient не будет учитывать эти изменения.**

**Для решения этой проблемы рекомендуется устанавливать параметр SocketsHttpHandler (наследник HttpMessageHandler) со свойством PooledConnectionLifetime, чтобы при замене продключения производился поиск DNS.**

***Пример:***

**class Program**

**{**

**static HttpClient? httpClient;**

**static async Task Main(string[] args)**

**{**

**var socketsHandler = new SocketsHttpHandler**

**{**

**PooledConnectionLifetime = TimeSpan.FromMinutes(2)**

**// когда завершается интервал (2 минуты, в данном случае)**

**// текущее подключение закрывается, и открывается новое**

**};**

**httpClient = new HttpClient(socketsHandler);**

**// использование HttpClient**

**}**

**}**

### Создание HttpClient с помощью IHttpClientFactory

**Для создания объекта HttpClient у фабрики IHttpClientFactory вызывается метод CreateClient. Например, если у нас простое консольное или десктопное приложение, то нам надо добавить через nuget пакеты Microsoft.Extensions.DependencyInjection и Microsoft.Extensions.Http. Если у нас проект веб-приложения, то подобные пакеты уже по умолчанию установлены в проект.**

**Фабрика IHttpClientFactory управляет пулом объектов HttpMessageHandler. Если некоторый объект HttpMessageHandler продолжает существовать в этом пуле, то он может повторно использоваться для создания нового объекта HttpClient. Это позволяет снизить вероятность исчерпания сокетов.**

***Пример:***

**using Microsoft.Extensions.DependencyInjection;**

**// определяем коллекцию сервисов приложения**

**var services = new SeriviceCollection();**

**// используя встроенный механизм Dependency Injection**

**// внедряем сервисы, связанные с HttpClient, в том числе IHttpClientFactory**

**services.AddHttpClient();**

**// создаем провайдер сервисов**

**var serviceProvider = services.BuildServiceProvider();**

**// получаем сервис IHttpClientFactory**

**var httpClientFactory = serviceProvider.GetService<IHttpClientFactory>();**

**// создаем объект HttpClient**

**var httpClient = httpClientFactory?.CreateClient();**

**// использование HttpClient**

## Отправка запросов с помощью HttpClient

**Методы HttpClient для выполнения запросов:**

* **Send() / SendAsync(): отправляет запрос в виде объекта HttpRequestMessage и получает ответ сервера в виде объекта HttpResponseMessage**
* **GetAsync(): отправляет GET-запрос на указанный адрес и получает ответ сервера в виде объекта HttpResponseMessage**
* **GetByteArrayAsync(): отправляет GET-запрос на указанный адрес и возвращает ответ в виде массива байтов (byte[])**
* **GetStreamAsync(): отправляет GET-запрос на указанный адрес и возвращает ответ в виде объекта Stream**
* **GetStringAsync(): отправляет GET-запрос на указанный адрес и возвращает ответ в виде строки**
* **PostAsync(): отправляет POST-запрос на указанный адрес и получает ответ сервера в виде объекта HttpResponseMessage**
* **PutAsync(): отправляет PUT-запрос на указанный адрес и получает ответ сервера в виде объекта HttpResponseMessage**
* **PatchtAsync(): отправляет Patch-запрос на указанный адрес и получает ответ сервера в виде объекта HttpResponseMessage**
* **DeleteAsync(): отправляет DELETE-запрос на указанный адрес и получает ответ сервера в виде объекта HttpResponseMessage**

**С помощью ряда свойств можно настроить базовые настройки отправки запросов:**

* **BaseAddress: возвращает или устанавливает базовый адрес в виде объекта URI, который будет использоваться при отправке запросов.**
* **DefaultProxy: возвращает или устанавливает настройки прокси.**
* **DefaultRequestHeaders: возвращает или устанавливает коллекцию заголовков по умолчанию, которые отправляются на сервер при каждом запросе**
* **DefaultRequestVersion: возвращает или устанавливает версию протокола HTTP по умолчанию, которая применяется при отправке запросов**
* **DefaultVersionPolicy: возвращает или устанавливает политику версии по умолчанию**
* **MaxResponseContentBufferSize: возвращает или устанавливает максимальный размер буфера в байтах при считывании данных ответа**
* **Timeout: возвращает или устанавливает тайм-аут перед отправкой запроса.**

### Отправка запроса методом SendAsync

**public Task<System.Net.Http.HttpResponseMessage> SendAsync (System.Net.Http.HttpRequestMessage request);**

**Класс HttpRequestMessage позволяет настроить данные запроса с помощью следующих свойств:**

* **Content: возвращает или устанавливает содержимое запроса**
* **Method: возвращает или устанавливает метод запроса**
* **RequestUri: возвращает или устанавливает адрес запроса в виде объекта Uri**
* **Version: возвращает или устанавливает версию протокола HTTP**

**Установить метод и адрес запроса также можно с помощью одного из конструкторов класса:**

**static HttpClient httpClient = new HttpClient();**

**static async Task Main()**

**{**

**// определяем данные запроса**

**using HttpRequestMessage request = new HttpRequestMessage(HttpMethod.Get, "https://www.google.com");**

**// выполняем запрос**

**await httpClient.SendAsync(request);**

**}**

### Получение ответа

**После выполнения запроса метод SendAsync возвращает ответ в виде объекта HttpResponseMessage.**

**Свойства HttpRequestMessage:**

* **Content: возвращает или устанавливает содержимое HTTP-ответа. Это свойство представляет объект HttpContent, который позволяет считать данные запроса с помощью одного из методов:**
* **ReadAsStringAsync(): возвращает ответ в виде строки**
* **ReadAsByteArrayAsync(): возвращает ответ в виде массива байт**
* **ReadAsStreemAsync(): возвращает ответ в виде потока - объекта Stream**
* **Headers: возвращает коллекцию заголовков HTTP-ответа (объект HttpResponseHeaders). Каждый заголовок в этой коллекции представлен парой ключ-значения. Для получения конкретного заголовка можно использовать метод GetValues(), в который передается название заголовка:**

**response.Headers.GetValues("Accept")**

* **IsSuccessStatusCode: возвращает true, если запрос HTTP прошел успешно**
* **ReasonPhrase: возвращает сообщение статуса**
* **RequestMessage: возвращает данные запроса, которые связаны с этим ответом**
* **StatusCode: возвращает статусный код ответа**
* **TrailingHeaders: возвращает дополнительные заголовки, включенные в запрос HTTP**
* **Version: возвращает версию использованного протокола HTTP**

***Пример:***

**static HttpClient httpClient = new HttpClient();**

**static async Task Main()**

**{**

**// определяем данные запроса**

**using HttpRequestMessage request = new HttpRequestMessage(**

**HttpMethod.Get,**

**"https:www.google.com");**

**// получаем ответ**

**using HttpResponseMessage response = await httpClient.SendAsync(request);**

**// просматриваем данные ответа:**

**// статус**

**var status = response.StatusCode;**

**// заголовки**

**foreach (var header in response.Headers)**

**{**

**var headerKey = header.Key;**

**foreach (var headerValue in header.Value) ...**

**}**

**// содержимое ответа**

**string content = await response.Content.ReadAsStringAsync();**

**}**

### GetAsync()

**Аналогично GetAsync()/PostAsync()/PatchAsync()/DeleteAsync()**

***Пример:***

**// получаем ответ**

**using HttpResponseMessage response**

**= await httpClient.GetAsync("https://www.google.com");**

**// содержимое ответа в виде html-кода веб-страницы**

**string content = await response.Content.ReadAsStringAsync();**

### GetStringAsync / GetByteArrayAsync / GetStreamAsync

**Когда нам надо получить строку, то гораздо проще выполнить запрос методом GetStringAsync():**

**string content = await httpClient.GetStringAsync("https://www.google.com");**

**Если сервер отправляет двоичные данные, то их можно получить с помощью метода GetByteArrayAsync():**

**byte[] buffer = await httpClient.GetByteArrayAsync("https://www.google.com");**

**И также можно получить поток методом GetStreamAsync():**

**using Stream stream = await httpClient.GetStreamAsync("https://www.google.com");**

**StreamReader reader = new StreamReader(stream);**

**string content = await reader.ReadToEndAsync();// считываем поток в строку**