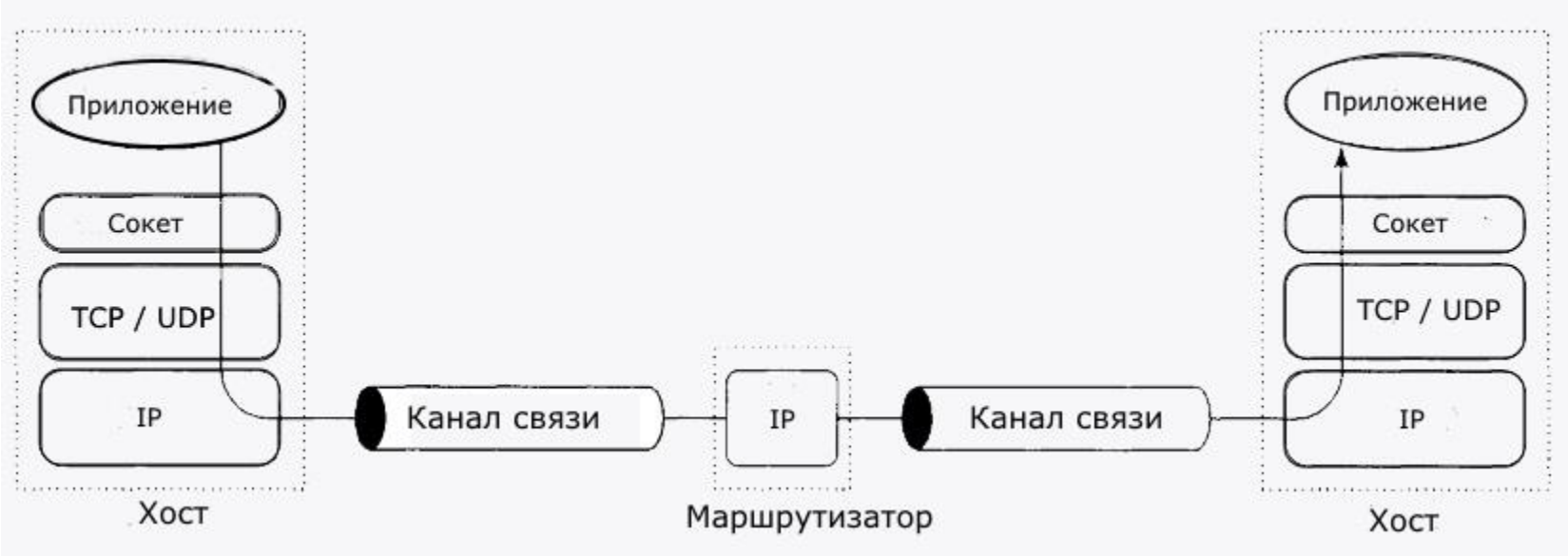
Основы работы с сетями в C# и .NET

ОДНИМ ИЗ РАСПРОСТРАНЕННЫХ ПРИЛОЖЕНИЙ, КОТОРОЕ ИСПОЛЬЗУЕТ ПЕРЕДАЧУ ПО СЕТИ, ЯВЛЯЕТСЯ

Вся сеть состоит из отдельных элементов – хостов, которые представляют собой компьютеры и другие подключенные устройства.

Вся сеть состоит из отдельных элементов – хостов, которые представляют собой компьютеры и другие подключенные устройства.

Существует множество различных протоколов. Протоколы, которые используются для передачи данных по сети, составляют семейство протоколов TCP/IP. Основные из них: Internet Protocol (IP), Transmission Control Protocol (TCP) и User Datagram Protocol (UDP).



Выше IP располагается транспортный уровень, который образуют протоколы TCP и UDP.

TCP позволяет отследить потерю пакетов и их дублирование при передаче.
UDP подобного не позволяет сделать и нацелен на простую передачу данных.

Сокеты – интерфейс для создания сетевых приложений, который опирается на встроенные возможности операционной системы

В зависимости от используемого протокола различают два вида сокетов:
1) потоковые сокеты (stream socket) и
2) дейтаграммные сокеты (datagram socket).

Потоковые сокеты используют протокол TCP, дейтаграммные - протокол UDP.

В итоге, когда приложение посылает сообщение приложению, запущенному на другом хосте, то приложение обращается к сокетам для передачи данных на уровень **TCP / UDP**

ДАЛЕЕ С ЭТОГО ТРАНСПОРТНОГО УРОВНЯ ДАННЫЕ ПЕРЕДАЮТСЯ СЕТЕВОМУ УРОВНЮ - УРОВНЮ ПРОТОКОЛА IP.

Чтобы уникально определять хосты в сети каждый хост имеет адрес. Существует несколько различных протоколов адресов. В настоящее время наиболее распространен протокол IPv4, который предполагает представление адреса в виде 32-битного числа

В последнее время набирает оборот использование адресов протокола IPv6, которые представляют собой 128-битное значение.

Однако такие адреса очень сложно запомнить, поэтому в реальности чаще оперируют доменами.

Домены представляют специальные названия, используемые для интернет-адресов

Например, есть доменное имя **"www.microsoft.com"**, ему соответствует адрес в

формате IPv4 **2.23.143.150**

Например, есть доменное имя **"www.microsoft.com"**, ему соответствует адрес в

формате IPv4 **2.23.143.150**

при отправке или передаче данных по доменному имени, компьютер еще обращается к службам Domain Name System (DNS), который выполняют сопоставление между интернет-адресами в формате IPv4 или IPv6 и доменными названиями.

Ключевыми компонентами сетевого взаимодействия являются клиент и сервер.

В реальности, как правило, при создании приложений не потребуется СЛИШКОМ глубокого знания всех протоколов и нюансов их работы

В реальности, как правило, при создании приложений не потребуется СЛИШКОМ глубокого знания всех протоколов и нюансов их работы

Основная функциональность фреймворка .NET по работе с сетями содержится в пакете **System.Net**.

1) System.Net.Http
2) System.Net.NetworkInformation
3) System.Net.Security
4) System.Net.Sockets
5) System.Net.WebSockets
6) System.Net.Quic

В качестве уникального адреса выступает IP-адрес

37.120.16.63 IPv4

2001:0db8:85a3:0000:0000:8a2e:0370:7334 IPv6

В системе классов .NET ip-адрес представлен классом IPAddress из пространства System.Net. Для определения ip-адреса можно применять ряд конструкторов этого класса. Некоторые из них:

```
public IPAddress (byte[] address);
public IPAddress (long newAddress);
```

```
using System.Net;

IPAddress localIp = new IPAddress(new byte[] { 127, 0, 0, 1 });
Console.WriteLine(localIp); // 127.0.0.1

IPAddress someIp = new IPAddress(0x0100007F);
Console.WriteLine(someIp); // 127.0.0.1
```

$$7F_{16} = 7 \cdot 16^1 + 15 \cdot 16^0 = 112 + 15 = 127_{10}$$

IPAddress.Parse и IPAddress.TryParse

Еще один и более удобный способ представляет метод Parse(), который представляет адрес в формате IPv4 или IPv6 в виде строки и возвращает объект IPAddress:

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

IPAddress.Parse и IPAddress.TryParse

Однако строка может содержать ошибки, и вполне возможно, что конвертация в IPAddress пройдет неудачно. В этом случае возникнет исключение. Чтобы этого избежать, мы можем использовать метод TryParse():

```
using System.Net;

IPAddress.TryParse("127.0.0.1rrr", out IPAddress? ip);
Console.WriteLine(ip?.ToString());
```

Встроенные адреса

Также класс 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

Среди свойств IPAddress следует отметить свойство AddressFamily, которое указывает на схему адресации.

Схема адресации

AppleTalk: адрес AppleTalk
Osi: адрес протоколов OSI
NS: адрес протоколов Xerox NS
 и.т.д

```
using System.Net;

IPAddress localIp = new IPAddress(new byte[] { 127, 0, 0, 1 });
Console.WriteLine(localIp.AddressFamily); // InterNetwork
```

Конечная точка IPEndpoint

Каждое сетевое приложение, которое прослушивает входящие запросы, запускается на определенном порту. Одномоментно один порт может прослушиваться только одним приложением.

Порт представляет 2-х байтное значение от 0 до 65535

Конечная точка IPEndpoint

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

Конечная точка IPEndpoint

Для создания конечной точки может применяться один из конструкторов класса:

```
public IPEndPoint (long address, int port);
public IPEndPoint (IPAddress address, int port);
```

Конечная точка IPEndpoint

В первом случае для установки ip-адреса применяется значение long, а во втором случае - объект IPAddress. Второй параметр во всех случаях представляет порт. Пример создания:

```
using System.Net;

IPAddress ip = IPAddress.Parse("127.0.0.1");
IPEndPoint endpoint = new IPEndPoint(ip, 8080);
Console.WriteLine(endpoint);        // 127.0.0.1:8080
```

Конечная точка IPEndpoint

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) предоставляет строку, которая глобально и уникально идентифицирует ресурс в сети

АДРЕСА URI

Подвидом URI является Uniform Resource Locator (URL) – универсальный стандарт для определения размещения ресурсов в сети.

АДРЕСА URI

Рассмотрим из каких частей состоит URL. Общая форма адреса URL выглядит следующим образом:

```
scheme:[//authority/]path[?query][#fragment]
```

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

```
//[access_credentials][@]host_domain[:port]
```

АДРЕСА URI

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 :

DNS

DNS представляет распределенную децентрализованную систему для получении информации о доменах.

DNS

Для идентификации ip-адресов .NET предоставляет статический класс Dns, который располагается в пространстве имен System.Net. В частности, мы можем использовать следующие методы данного класса:

DNS

1) GetHostAddresses
2) GetHostEntry
3) GetHostName()

DNS

Например, получим данные по доменному имени "google.com":

```
using System.Net;

var googleEntry = await Dns.GetHostEntryAsync("google.com");
Console.WriteLine(googleEntry.HostName);
foreach (var ip in googleEntry.AddressList)
{
    Console.WriteLine(ip);
}
```

DNS

Например, получим данные по доменному имени "google.com":

```
using System.Net;

var googleEntry = await Dns.GetHostEntryAsync("google.com");
Console.WriteLine(googleEntry.HostName);
foreach (var ip in googleEntry.AddressList)
{
    Console.WriteLine(ip);
}
```

Получение информации о сетевой конфигурации и сетевом трафике

Для получения информации о сетевых устройствах/интерфейсах на текущей машине можно использовать абстрактный класс **NetworkInterface.**

```
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}");
}
```

Получение информации о всех подключениях

Еще один класс - IPGlobalProperties позволяет получить детальную информацию по сетевому трафику и его конфигурации с помощью ряда методов

Мониторинг трафика

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

Мониторинг трафика

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

КЛАСС SOCKET

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 интерфейс для приема и отправки сообщений по сети.

Свойства сокета

свойства класса позволяют получить информацию о сокете.

Закрытие сокета

```
Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
// работа с сокетом socket
// ..............
socket.Close();
```

Закрытие сокета

```
Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

try
{
    socket.Shutdown(SocketShutdown.Both);
}
catch(Exception ex)
{
    Console.WriteLine(ex.Message);
}
finally
{
    socket.Close();
}
```