Сетевое программирование сокетов

Сокет

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.

Виды сокетов

TCP-сокеты: обеспечивают надёжную передачу данных с помощью протокола TCP

UDP-сокеты: используют протокол UDP, который обеспечивает быструю передачу данных, но не гарантирует их доставку или порядок доставки.

UNIX-сокеты: используются для взаимодействия процессов внутри одной операционной системы.

Где используются

- Коммуникация между клиентом и сервером
- Многопользовательские игры
- Распределённые вычисления
- Автоматизация
- Чаты и мессенджеры

Работа с сокетами

1) Создание сокета

2) Привязка к порту

3) Прослушивание соединений

4) Подключение клиента

5) Обмен данными

6) Закрытие соединения

C#

Для работы с сокетами в C# используется пространство имен System.Net.Sockets. Ключевыми классами являются Socket, TcpClient и UdpClient.

Определение сокета

Конструктор:

Socket(AddressFamily, SocketType, ProtocolType): создает сокет, используя указанные семейство адресов, тип сокета и протокол.

Пример

Сокет, который использует протокол Tcp:

```
using System.Net.Sockets;

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

Пример

Сокет, который использует протокол Udp:

```
using System.Net.Sockets;

Socket socket = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp);
```

Свойства

AddressFamily: представляет схему адресации, используемую сокетом

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

Свойства

ProtocolType: возвращает тип протокола

RemoteEndPoint: возвращает адрес удаленного хоста, к которому подключен сокет

SocketType: возвращает тип сокета

Методы

Bind(): связывает объект Socket с локальной конечной точкой

Close(): закрывает сокет

Connect(): устанавливает соединение с удаленным хостом

Listen(): начинает прослушивание входящих запросов

Методы

Receive() / ReceiveAsync: получает данные

Send() / SendAsync(): отправляет данные

Shutdown(): блокирует на сокете прием и/или отправку данных

Методы

Receive() / ReceiveAsync: получает данные

Send() / SendAsync(): отправляет данные

Shutdown(): блокирует на сокете прием и/или отправку данных

Закрытие сокета

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

// работа с сокетом socket

// .............

socket.Close();
```

1. Создание сервера

Для создания серверного приложения необходимо открыть сокет, привязать его к определенному порту и начать прослушивание входящих соединений.

Создание и настройка сокета

```
// Создаем новый сокет для прослушивания входящих соединений
Socket listener = new Socket(SocketType.Stream, ProtocolType.Tcp);
// Привязываем сокет к локальному IP-адресу и порту
IPEndPoint localEndPoint = new IPEndPoint(IPAddress.Any, 5000); // Порт 5000
 listener.Bind(localEndPoint);
```

Прослушка

```
// Начинаем слушать входящие соединения
listener.Listen(10); // Максимум 10 подключений в очереди
Console.WriteLine("Ожидание подключения...");
```

```
while (true) {
        // Принимаем новое соединение
        Socket handler = listener.Accept();
        // Получаем данные от клиента
        byte[] buffer = new byte[1024];
        int bytesReceived = handler.Receive(buffer);
        string data = Encoding.UTF8.GetString(buffer, 0, bytesReceived);
        Console.WriteLine($"Получено сообщение: {data}");
        // Отправляем ответ клиенту
        string response = "Сообщение получено!";
        byte[] responseBuffer = Encoding.UTF8.GetBytes(response);
        handler.Send(responseBuffer);
        // Закрываем соединение
        handler.Shutdown(SocketShutdown.Both);
        handler.Close();
    }
```

2. Создание клиента

Клиентское приложение должно установить соединение с сервером и отправить ему данные.

Создание и настройка сокета

```
// Создаем новый сокет для связи с сервером
Socket clientSocket = new Socket(SocketType.Stream,
ProtocolType.Tcp);
// Устанавливаем соединение с сервером
IPEndPoint serverEndPoint = new
IPEndPoint(IPAddress.Loopback, 5000);
// Локальный хост, порт 5000
clientSocket.Connect(serverEndPoint);
```

Отправка сообщения

```
// Отправляем сообщение серверу
string message = "Hello from client!";
byte[] sendBuffer = Encoding.UTF8.GetBytes(message);
clientSocket.Send(sendBuffer);
```

Получение сообщения

```
// Получаем ответ от сервера
byte[] receiveBuffer = new byte[1024];
int bytesReceived = clientSocket.Receive(receiveBuffer);
string response = Encoding.UTF8.GetString(receiveBuffer, 0,
bytesReceived);
Console.WriteLine($"Ответ от сервера: {response}");
```

Закрытие соединения

```
// Закрываем соединение
clientSocket.Shutdown(SocketShutdown.Both);
clientSocket.Close();
```

Важные моменты

Порт: Порт определяет точку входа для сетевых запросов. Сервер и клиент должны использовать одинаковый порт для успешной коммуникации.

Протоколы: Чаще всего используются протоколы TCP (для надежной передачи данных) и UDP (для быстрой передачи без гарантии доставки).

Важные моменты

Закрытие сокетов: После завершения работы с соединением важно закрыть сокеты, чтобы избежать утечек ресурсов.

Задание

Изменить код таким образом, чтоб появилась возможность подключение нескольких клиентов.