Принципы работы в сети

№2

Исключения

- При работе с классами WebRequest и WebResponse могут возникать исключения. Непосредственно к работе с сетью относится класс исключений **WebException**. Его ключевым свойством является свойство **Status**, которое позволяет получит тип

Свойство статус

- ConnectFailure: невозможно подключиться к ресурсу
- ConnectionClosed: подключение было преждевременно закрыто
- NameResolutionFailure: служба DNS не может сопоставить имя хоста с ip-адресом
- RequestCanceled: запрос был отменен
- Timeout: ответ не был получен в течение определенного времени
- UnknownError: возникло исключение неизвестного типа

Класс Socket свойства

- AddressFamily: возвращает все адреса, используемые сокетом. Данное свойство представляет одно из значений, определенных в одноименном перечислении **AddressFamily**. Перечисление содержит 18 различных значений, наиболее используемые:
- InterNetwork: адрес по протоколу IPv4
- InterNetworkV6: адрес по протоколу IPv6
- Ipx: адрес IPX или SPX
- NetBios: адрес NetBios

Класс Socket свойства

- Connected: возвращает true, если сокет подключен к удаленному хосту
- ProtocolType. Есть следующие возможные значения перечисления:
  - IPv4
  - IPv6
  - Tcp
  - Udp

    и т.д.
- RemoteEndPoint: возвращает адрес удаленного хоста, к которому подключен сокет

Класс Socket свойства

SocketType: возвращает тип сокета. Представляет одно из значений из перечисления **SocketType**:

- Dgram: сокет будет получать и отправлять дейтаграммы по протоколу Udp. Данный тип сокета работает в связке с типом протокола - Udp и значением AddressFamily.InterNetwork
- 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.

Класс Socket создание

- Для создания объекта сокета можно использовать один из его конструкторов. Например, сокет, использующий протокол Tcp:

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

Класс Socket методы

- Accept(): создает новый объект Socket для обработки входящего подключения
- Bind(): связывает объект Socket с локальной конечной точкой
- Close(): закрывает сокет
- Connect(): устанавливает соединение с удаленным хостом
- Listen(): начинает прослушивание входящих запросов
- Poll(): определяет состояние сокета
- Receive(): получает данные
- Send(): отправляет данные
- Shutdown(): блокирует на сокете прием и отправку данных

Протокол TCP

- Для работы с протоколом TCP в .NET предназначены классы **TcpClient** и **TcpListener**. Эти классы строятся поверх класса**System.Net.Sockets.Socket**.

TCP – клиент. TcpClient

- Connect() - для подключения к серверу TCP
- GetStream() для взаимодействия с сервером который возвращает объект **NetworkStream.**
- Write() для отправки данных
- Read() для чтения данных
- Close() закрытие потока

```
try
{
    TcpClient client = new TcpClient();
    client.Connect(server, port);

    byte[] data = new byte[256];
    StringBuilder response = new StringBuilder();
    NetworkStream stream = client.GetStream();

    do
    {
        int bytes = stream.Read(data, 0, data.Length);
        response.Append(Encoding.UTF8.GetString(data, 0, bytes));
    }
    while (stream.DataAvailable); // пока данные есть в потоке

    Console.WriteLine(response.ToString());
```

TCP-сервер. Класс TcpListener

- Класс TcpListener прослушивает входящие подключения по определенному порту.
- Для запуска и остановки сервера:
- Start()
- Stop()

TcpListener

- Когда к серверу обращается клиент, то мы можем использовать один из двух методов **AcceptSocket** или **AcceptTcpClient** для получения соответственно объекта **Socket** или **TcpClient**

```
TcpListener server=null;
try
{
    IPAddress localAddr = IPAddress.Parse("127.0.0.1");
    server = new TcpListener(localAddr, port);

    // запуск слушателя
    server.Start();

    while (true)
    {
        Console.WriteLine("Ожидание подключений... ");

        // получаем входящее подключение
        TcpClient client = server.AcceptTcpClient();
        Console.WriteLine("Подключен клиент. Выполнение запроса...");

        // получаем сетевой поток для чтения и записи
        NetworkStream stream = client.GetStream();
```

```
            // сообщение для отправки клиенту
            string response = "Привет мир";
            // преобразуем сообщение в массив байтов
            byte[] data = Encoding.UTF8.GetBytes(response);

            // отправка сообщения
            stream.Write(data, 0, data.Length);
            Console.WriteLine("Отправлено сообщение: {0}", response);
            // закрываем подключение
            client.Close();
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
    finally
    {
        if (server != null)
            server.Stop();
    }
```

NetworkStream

- Он отличается от других классов потоков тем, что он не является буферизованным и не поддерживает перемещение в произвольную позицию с помощью метода Seek. Расположен в пространстве имен System.Net.Sockets

NetworkStream

- Свойство DataAvailable: возвращает значение true, если в потоке есть данные. Если их нет, возвращается false.
- Метод Read(byte[] buffer, int offset, int length);: считывает данные из потока в массив buffer, начиная со смещения offset. Количество считываемых из потока данных указывается в параметре length
- Метод Write(byte[] buffer, int offset, int length): отправляет данные из массива buffer, начиная со смещения offset, в поток. Количество отправляемых из массива данных указывается в параметре length
- Метод Close(): закрывает поток

```
listener = new TcpListener(IPAddress.Parse("127.0.0.1"), PORT);
listener.Start();
Console.WriteLine("Ожидание подключений...");

while (true)
{
    TcpClient client = listener.AcceptTcpClient();
    NetworkStream stream = client.GetStream();

    StreamReader reader = new StreamReader(stream);
    // считываем строку из потока
    string message = reader.ReadLine();
    Console.WriteLine("Получено: " + message);

    // отправляем ответ
    StreamWriter writer = new StreamWriter(stream);
    message = message.ToUpper();
    Console.WriteLine("Отправлено: " + message);
    writer.WriteLine(message);
```

```
Console.Write("Введите сообщение: ");
string message = Console.ReadLine();
client = new TcpClient(ADDRESS, PORT);
NetworkStream stream = client.GetStream();

// отправляем сообщение
StreamWriter writer = new StreamWriter(stream);
writer.WriteLine(message);
writer.Flush();

// BinaryReader reader = new BinaryReader(new BufferedStream(stream));
StreamReader reader = new StreamReader(stream);
message = reader.ReadLine();
Console.WriteLine("Получен ответ: " + message);
```

Работа с электронной почтой

- Для отправки почты в среде интернет используется протокол SMTP (Simple Mail Transfer Protocol). Данный протокол указывает, как почтовые сервера взаимодействуют при передаче электронной почты.

Работа с электронной почтой

- Для работы с протоколом SMTP и отправки электронной почты в .NET предназначен класс **SmtpClient** из пространства имен**System.Net.Mail**.

SmtpClient

- Host: smtp-сервер, с которого производится отправление почты. Например, smtp.yandex.ru
- Port: порт, используемый smp-сервером. Если не указан, то по умолчанию используется 25 порт.
- Credentials: аутентификационные данные отправителя
- EnableSsl: указывает, будет ли использоваться протокол SSL при отправке

MailMessage – сообщение

- Attachments: содержит все прикрепления к письму
- Body: непосредственно текст письма
- From: адрес отправителя. Представляет объект MailAddress
- To: адрес получателя. Также представляет объект MailAddress
- Subject: определяет тему письма
- IsBodyHtml: указывает, представляет ли письмо содержимое с кодом html

```
// отправитель - устанавливаем адрес и отображаемое в письме имя
MailAddress from = new MailAddress("somemail@gmail.com", "Tom");
// кому отправляем
MailAddress to = new MailAddress("somemail@yandex.ru");
// создаем объект сообщения
MailMessage m = new MailMessage(from, to);
// тема письма
m.Subject = "Тест";
// текст письма
m.Body = "<h2>Письмо-тест работы smtp-клиента</h2>";
// письмо представляет код html
m.IsBodyHtml = true;
// адрес smtp-сервера и порт, с которого будем отправлять письмо
SmtpClient smtp = new SmtpClient("smtp.gmail.com", 587);
// логин и пароль
smtp.Credentials = new NetworkCredential("somemail@gmail.com", "mypassword");
smtp.EnableSsl = true;
smtp.Send(m);
```