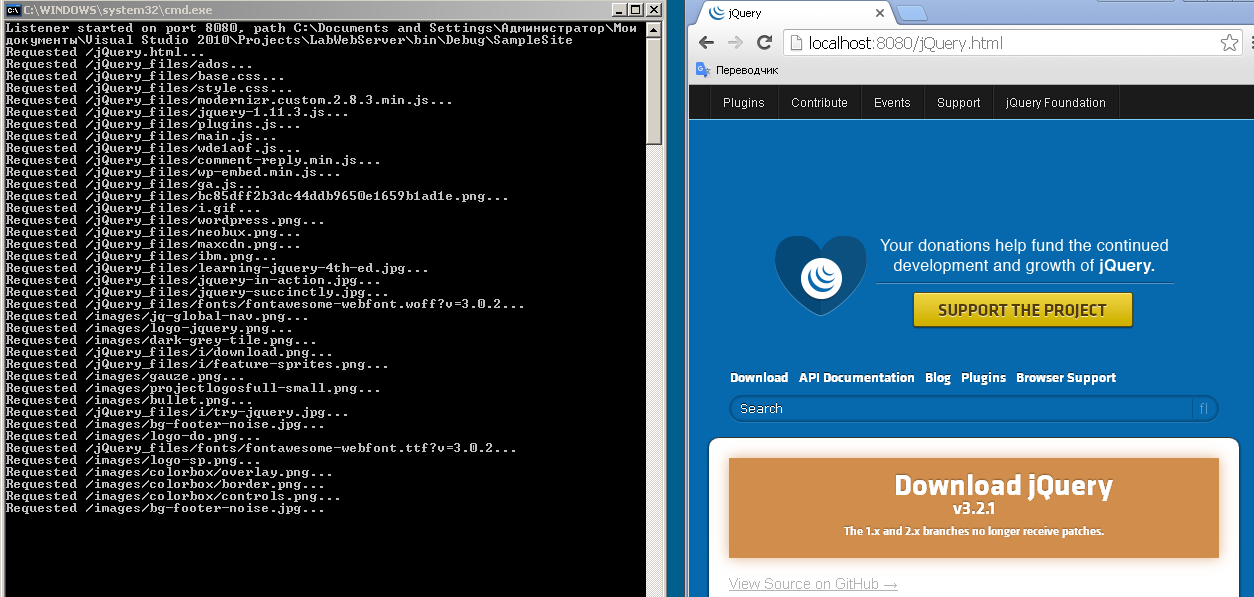
# Лабораторная работа №5: разработка HTTP веб-сервера

## Цели

1. Опробовать сетевую передачу данных по протоколу TCP/IP.
2. Реализовать на практике события .NET.

## Задание

Написать простейший HTTP веб-сервер, способный отдавать статические файлы и, опционально, обрабатывать прочие запросы на получение данных. Проверка работоспособности веб-сервера будет выполняться через веб-браузер. Пример работы веб-сервера изображён ниже (слева консоль веб-сервера, справа результат запроса тестовой страницы в веб-браузере):



Веб-сервер должен обрабатывать только запросы на выдачу файлов (запросы GET). Если запрашиваемый файл найден, то веб-сервер должен вернуть его вместе с необходимыми заголовками HTTP. Описание протокола HTTP и все необходимые заголовки перечислены во вспомогательных материалах.

Выполнение перечисленных выше требований оценивается как «удовлетворительно». Для получения более высоких оценок необходимо дополнительно:

* Для получения оценки «Хорошо» в веб-сервере должно быть реализовано событие, подписываясь на которое вызывающий код сможет самостоятельно обрабатывать запрос вместо веб-сервера. Если пользовательский код обработал запрос, то серверу его обрабатывать уже не надо.
* Для получения оценки «Отлично» размер класс веб-сервера должен быть менее 250 строк.

### Требования к программе

Запрещено применение готовых библиотек для работы HTTP. Программа должна самостоятельно получать данные по TCP/IP, разбирать их и отдавать ответ.

Веб-сервер должен быть реализован в виде отдельного класса. Класс должен содержать только один открытый метод Start для запуска веб-сервера.

При выполнении задания на оценку «хорошо» веб-сервер должен содержать одно событие CustomRequest, содержащее как минимум строку запроса.

В качестве параметров (параметров конструктора или в виде свойств) веб-сервер должен получать как минимум номер TCP-порта и путь к каталогу, файлы из которого он будет отдавать.

## Демонстрация работы

1. Запустить веб-сервер.
2. Открыть в браузер и перейти в нём на тестовую страницу. Она должна отобразиться корректно.

## Вспомогательные материалы

### Протокол HTTP

Протокол HTTP (HyperText Transfer Protocol) представляет собой протокол для передачи произвольных данных, построенный по схеме запрос-ответ. Каждый запрос и ответ представляют собой последовательность строк, разделённых символами перевода строки (CRLF=”\r\n”). Конец запроса кодируется как пустая строка (или, другими словами, как два подряд идущих перевода строки).

Каждое HTTP-сообщение состоит из трёх частей, которые передаются в указанном порядке:

* Стартовая строка (англ. Starting line) — определяет тип сообщения;
* Заголовки (англ. Headers) — характеризуют тело сообщения, параметры передачи и прочие сведения;
* Тело сообщения (англ. Message Body) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

Протокол определяет множество заголовков, но в рамках данной лабораторной работы все заголовки запросов можно игнорировать: обрабатывать только стартовую строку, пропуская остаток запроса вплоть до его завершения (до пустой строки). Пример HTTP-запроса:

*GET /wiki/SomePage.html HTTP/1.1*

*Host: ru.wikipedia.org*

*User-Agent: Mozilla/5.0 (X11; U; Linux i686; ru; rv:1.9b5) Gecko/2008050509 Firefox/3.0b5*

*Accept: text/html*

*Connection: close*

В данном примере клиент (скорее всего веб-браузер) запрашивает (GET) ресурс «/wiki/SomePage.html». Веб-сервер должен проверить первое слово запроса: в рамках лабораторной работы необходимо обрабатывать только запросы, начинающиеся с «GET ».

В ответ сервер должен вернуть стартовую строку с кодом возврата и, если запрашиваемый файл найден, содержимое файла. Пример стартовой строки ответа:

*HTTP/1.1 200 OK*

Здесь:

HTTP/1.1 – это фиксированная строка с версией протокола;

200 – код ошибки (перечень основных кодов перечислен ниже);

OK – текстовое описание ошибки.

После стартовой строки необходимо вернуть заголовки ответа. В рамках лабораторной работы достаточно вернуть следующие заголовки:

* Content-Type – MIME-тип содержимого (текстовый файл, HTML, картинка, или что-то еще – достаточный для выполнения работы список типов перечислен ниже);
* Content-Length – длина содержимого в байтах.

Пример ответа сервера на запрос:

*HTTP/1.1 200 OK*

*Content-Type: text/html*

*Content-Length: 16*

*<html>hi!</html>*

Основные коды ошибок HTTP с типовым текстовым описанием:

* 200 – успешное завершение запроса (обычно называется «OK»);
* 400 – некорректный запрос (обычно называется «Bad request»);
* 404 – страница не найдена (обычно называется «Not found»);
* 500 – внутренняя ошибка сервера (обычно называется «Internal error»).

Некоторые общепринятые MIME-типы (можно определить по расширению файла):

|  |  |  |
| --- | --- | --- |
| **Тип файла** | **Расширение** | **MIME** |
| HTML | .html или .htm | text/html |
| стили CSS | .css | text/css |
| скрипт JavaScript | .js | text/javascript |
| изображение JPEG | .jpg или .jpeg | image/jpeg |
| изображение PNG | .png | image/png |
| изображение GIF | .gif | image/gif |
| неизвестное содержимое |  | application/unknown |

### Полезные методы и структуры .NET

Для прослушивания подключений и передачи данных по TCP/IP рекомендуется использование классов System.Net.Sockets.TcpListener и System.Net.Sockets.TcpClient.

Пример программы, считывающей HTTP запрос и возвращающий в ответ всегда успех:

// Запуск прослушивания на порту 777

var listener = new TcpListener ( IPAddress.Any, 777 );

listener.Start ( );

while ( true ) {

// Блокировка потока пока кто-нибудь не подключится

var client = listener.AcceptTcpClient ( );

Console.WriteLine ( "Accepted from {0}", client.Client.RemoteEndPoint );

var stream = client.GetStream ( );

using ( var streamReader = new StreamReader ( stream ) ) {

while ( true ) {

// Чтение запроса до конца

var line = streamReader.ReadLine ( );

if ( string.IsNullOrEmpty ( line ) )

break;

Console.WriteLine ( line );

}

// Ответ

string response = "HTTP/1.1 200 OK\r\n\r\n";

byte[] responseBytes = Encoding.ASCII.GetBytes ( response );

stream.Write ( responseBytes, 0, responseBytes.Length );

}

}

Коды HTTP перечислены в System.Net.HttpStatusCode (например, HttpStatusCode.OK).

Проверка наличия файла на диске и запрос его размера можно реализовать с помощью класса System.IO.FileInfo