HttpClient в C# - шпаргалка

1. Инициализация HttpClient

```
// Создание экземпляра HttpClient
HttpClient client = new HttpClient();
```

С использованием using:

```
private async void btnSendRequest_Click(object sender, EventArgs e)
{

    using (var client = _httpClientFactory.CreateClient())
    {
        var response = await client.GetAsync("https://api.example.com/data");
        // Обработка ответа
    }
}
```

HttpClient следует создавать **один раз и переиспользовать**, чтобы избежать исчерпания ресурсов (например, портов).
Рекомендуемый способ:

```
namespace MyProject
{
    public partial class Form1 : Form
    {
        static readonly HttpClient client = new HttpClient();

        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

2. Метод GetAsync

Загружает HTTP-ответ в виде **HttpResponseMessage**.
Основные свойства HttpResponseMessage:

- **StatusCode** – возвращает код состояния HTTP-ответа (например, 200, 404, 500).
- **IsSuccessStatusCode** – true, если код состояния в диапазоне 200-299.
- **Content** – содержит тело ответа (HttpContent).
- **Headers** – коллекция заголовков HTTP-ответа.
- **ReasonPhrase** – строковое описание статуса (например, "OK", "Not Found").
- **Version** – версия HTTP-протокола.

```
private async void button1_Click(object sender, EventArgs e)
{
    // Отправка асинхронного GET-запроса по указанному URL
    HttpResponseMessage response = await client.GetAsync("https://ya.kz");

    // Проверка, успешен ли статус ответа (код 200-299)
    if (response.IsSuccessStatusCode)
    {
        // Вывод статуса ответа (например, "OK" для кода 200) в richTextBox1
        richTextBox1.Text = response.StatusCode.ToString() + "\n";

        // Чтение тела ответа (содержимого) в виде строки
        string responseBody = await response.Content.ReadAsStringAsync();

        // Добавление содержимого ответа в richTextBox1
        richTextBox1.Text += responseBody;
    }
}
```

Код с обработкой исключений:

```
private async void button1_Click(object sender, EventArgs e)
{
    try
    {
        // Отправка асинхронного GET-запроса по указанному URL
        HttpResponseMessage response = await client.GetAsync("https://ya.kz");

        // Проверка, успешен ли статус ответа (код 200-299)
        if (response.IsSuccessStatusCode)
        {
            // Вывод статуса ответа (например, "OK" для кода 200) в richTextBox1
            richTextBox1.Text = response.StatusCode.ToString() + "\n";

            // Чтение тела ответа (содержимого) в виде строки
            string responseBody = await response.Content.ReadAsStringAsync();

            // Добавление содержимого ответа в richTextBox1
            richTextBox1.Text += responseBody;
        }
        else
        {
            // Если статус ответа не успешен, выводим сообщение об ошибке
            richTextBox1.Text = "Ошибка: " + response.StatusCode.ToString();
        }
    }
    catch (HttpRequestException ex)
    {
        // Обработка ошибок, связанных с HTTP-запросами (например, проблемы с сетью)
        richTextBox1.Text = "Ошибка при выполнении запроса: " + ex.Message;
    }
    catch (Exception ex)
    {
        // Обработка всех остальных исключений
        richTextBox1.Text = "Произошла ошибка: " + ex.Message;
    }
}
```

3. Метод GetStringAsync

```
private async void button1_Click(object sender, EventArgs e)
{
    string responseBody = await client.GetStringAsync("https://ya.kz");
    richTextBox1.Text = responseBody;
}
```

Пример работы с json:

```
# Используем библиотеку для работы с json
# using System.Text.Json;

private async void button2_Click(object sender, EventArgs e)
{
    // Токен для доступа к API Superhero API
    string token = "8c4dcfa7fb56328af902aa915b46e2fb";

    // ID героя, информацию о котором мы хотим получить
    string heroId = "732";

    // Формируем URL для запроса, подставляя токен и ID героя
    string url = $"https://superheroapi.com/api/{token}/{heroId}";

    // Отправляем асинхронный GET-запрос по сформированному URL
    HttpResponseMessage response = await client.GetAsync(url);

    // Читаем ответ от сервера в виде строки (JSON)
    string jsonResponse = await response.Content.ReadAsStringAsync();

    // Парсим JSON-ответ в объект JsonDocument для дальнейшей работы
    using JsonDocument doc = JsonDocument.Parse(jsonResponse);

    // Получаем корневой элемент JSON-документа
    JsonElement root = doc.RootElement;

    // Извлекаем значение поля "id" из корневого элемента
    string id = root.GetProperty("id").GetString();

    // Извлекаем значение поля "name" из корневого элемента
    string name = root.GetProperty("name").GetString();

    // Извлекаем значение поля "intelligence" из вложенного объекта "powerstats"
    string intelligence = root.GetProperty("powerstats").GetProperty("intelligence").GetString();
}
```

4. Метод GetByteArrayAsync

Загружает содержимое HTTP-ответа в виде массива байтов.
Пример. Скачивание картинки.

```
private async void button1_Click(object sender, EventArgs e)
{
    // URL изображения, которое нужно загрузить
    string url = @"https://http.cat/images/200.jpg";

    try
    {
        // Асинхронно загружаем данные изображения в виде массива байтов
        byte[] imageData = await client.GetByteArrayAsync(url);

        // Создаем поток в памяти (MemoryStream) из массива байтов
        using MemoryStream ms = new MemoryStream(imageData);

        // Загружаем изображение из потока и устанавливаем его в pictureBox1
        pictureBox1.Image = Image.FromStream(ms);
    }
    catch (Exception ex)
    {
        // Если произошла ошибка, выводим сообщение с описанием проблемы
        MessageBox.Show("Ошибка загрузки: " + ex.Message);
    }
}
```

Скачивание файла.

```
private async void button1_Click(object sender, EventArgs e)
{
    // URL файла, который нужно скачать
    string url = @"https://github.com/thonny/thonny/releases/download/v4.1.7/thonny-4.1.7.exe";

    // Локальный путь, куда будет сохранен файл
    string filePath = "thonny.exe";

    try
    {
        // Асинхронно загружаем файл в виде массива байтов
        byte[] fileBytes = await client.GetByteArrayAsync(url);

        // Асинхронно записываем массив байтов в файл по указанному пути
        await File.WriteAllBytesAsync(filePath, fileBytes);

        // Выводим сообщение об успешной загрузке и размере файла в richTextBox1
        richTextBox1.Text = $"Файл {filePath} успешно скачан, размер {fileBytes.Length} байт";
    }
    catch (Exception ex)
    {
        // Если произошла ошибка, выводим сообщение с описанием проблемы в richTextBox1
        richTextBox1.Text = "Ошибка загрузки: " + ex.Message;
    }
}
```

5. Метод GetStreamAsync.

Метод GetStreamAsync является асинхронным методом, предоставляемым классом HttpClient в C#. Он используется для отправки HTTP GET-запроса и получения ответа в виде потока (Stream). Этот метод особенно полезен, когда необходимо работать с большими объемами данных или когда точный размер ответа заранее неизвестен.
Пример. Загрузка файла.

```
private async void button1_Click(object sender, EventArgs e)
{
    string url = @"https://github.com/thonny/thonny/releases/download/v4.1.7/thonny-4.1.7.exe";
    string filePath = "thonny.exe";

    try
    {
        // Открываем поток для чтения данных с сервера
        using (Stream stream = await client.GetStreamAsync(url))
        // Открываем поток для записи данных в файл
        using (FileStream fileStream = new FileStream(filePath, FileMode.Create, FileAccess.Write))
        {
            // Копируем данные из потока HTTP-ответа в файл
            await stream.CopyToAsync(fileStream);
        }

        MessageBox.Show("Файл успешно скачан и сохранен.");
    }
    catch (Exception ex)
    {
        MessageBox.Show("Ошибка загрузки: " + ex.Message);
    }
}
```

6. PostAsync (Отправка данных)

Метод PostAsync используется для отправки данных на сервер (например, при создании ресурса).
Пример отправки POST-запроса на сайт https://httpbin.org/

```
private async void button1_Click(object sender, EventArgs e)
{
    // Создание словаря с данными, которые будут отправлены в запросе
    var postData = new Dictionary<string, string>
    {
        { "name", "Nikita" },   // Имя пользователя
        { "age", "20" },          // Возраст пользователя
        { "city", "Almaty" }    // Город пользователя
    };

    // Кодирование данных в формат application/x-www-form-urlencoded
    var content = new FormUrlEncodedContent(postData);

    try
    {
        // Отправка асинхронного POST-запроса на указанный URL
        HttpResponseMessage response = await client.PostAsync("https://httpbin.org/post", content);

        // Проверка, успешно ли выполнен запрос (статус код 200-299)
        if (response.IsSuccessStatusCode)
        {
            // Чтение содержимого ответа в виде строки
            string responseBody = await response.Content.ReadAsStringAsync();

            // Вывод полученного ответа в элемент richTextBox1
            richTextBox1.Text = $"Response received successfully:\n{responseBody}";
        }
        else
        {
            // Вывод сообщения об ошибке в случае неудачного запроса
            richTextBox1.Text = $"Error: {response.StatusCode}";
        }
    }
    catch (Exception ex)
    {
        // Обработка исключений, если произошла ошибка во время выполнения запроса
        richTextBox1.Text = $"Exception: {ex.Message}";
    }
}
```

Разбор:
postData – это Dictionary<string, string>, где ключи и значения представляют собой пары параметров запроса.
FormUrlEncodedContent – это специальный класс в System.Net.Http, который преобразует переданный словарь в URL-кодированную строку вида:
name=John+Doe&age=30&city=New+York
& разделяет параметры (name=..., age=..., city=...).
Пробелы заменяются на + или %20.
Этот объект передается в HttpClient.PostAsync(), чтобы отправить данные в теле запроса с заголовком Content-Type: application/x-www-form-urlencoded.

Основные свойства HttpClient

BaseAddress — базовый URI для запросов

```
client.BaseAddress = new Uri("https://api.example.com/");
```

DefaultRequestHeaders — коллекция HTTP-заголовков, которые отправляются с каждым запросом

```
client.DefaultRequestHeaders.Add("User-Agent", "My App");
client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", "token");
```

Timeout — максимальное время ожидания ответа

```
client.Timeout = TimeSpan.FromSeconds(30);
```

MaxResponseContentBufferSize — максимальный размер буфера для ответа

```
client.MaxResponseContentBufferSize = 1024 * 1024; // 1 MB
```