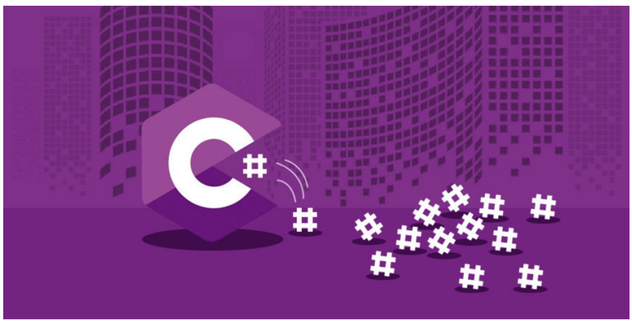
C# HttpClient全攻略：GET、POST、文件传输与授权设置一网打尽

作者：微信公众号：【架构师老卢】 12-28 8:10 354



概述：本文详解了在C#中使用HttpClient发送HTTP请求的全面指南，包括GET、POST请求、文件上传和下载，以及设置Authorization、cookie等操作，为实现网络操作提供了清晰的方法和示例。

1. 说明

`HttpClient`是C#中用于发送HTTP请求的类，基于HttpClient的异步模型，可以实现GET、POST请求，处理响应数据，上传和下载文件，以及设置Authorization、cookie等。

2. 方法说明

2.1 发送GET请求

```
using (HttpClient client = new HttpClient())
{
    HttpResponseMessage response = await client.GetAsync("https://example.com/api/resource");
    // 处理响应
}
```

2.2 发送POST请求

```
using (HttpClient client = new HttpClient())
{
    var content = new StringContent("data to send", Encoding.UTF8, "application/json");
    HttpResponseMessage response = await client.PostAsync("https://example.com/api/resource", content);
    // 处理响应
}
```

2.3 上传文件

```
using (HttpClient client = new HttpClient())
{
    var content = new MultipartFormDataContent();
    content.Add(new ByteArrayContent(File.ReadAllBytes("file.txt")), "file", "file.txt");
    HttpResponseMessage response = await client.PostAsync("https://example.com/api/upload", content);
    // 处理响应
}
```

2.4 下载文件

```
using (HttpClient client = new HttpClient())
{
    byte[] fileData = await client.GetByteArrayAsync("https://example.com/api/download");
    File.WriteAllBytes("downloaded_file.txt", fileData);
}
```

2.5 设置Authorization

```
using (HttpClient client = new HttpClient())
{
    client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", "your_access_token");
    HttpResponseMessage response = await client.GetAsync("https://example.com/api/resource");
    // 处理响应
}
```

2.6 设置Cookie

```
using (HttpClient client = new HttpClient())
{
    var cookieContainer = new CookieContainer();
    var handler = new HttpClientHandler { CookieContainer = cookieContainer };
    using (var httpClient = new HttpClient(handler))
    {
        HttpResponseMessage response = await client.GetAsync("https://example.com/api/resource");
        // 处理响应
    }
}
```

3. 步骤说明

3.1 发送GET请求

1. 创建`HttpClient`实例。
2. 使用`GetAsync`方法发送GET请求。
3. 处理`HttpResponseMessage`以获取响应数据。

3.2 发送POST请求

1. 创建`HttpClient`实例。
2. 创建`StringContent`或`FormUrlEncodedContent`作为请求内容。
3. 使用`PostAsync`方法发送POST请求。
4. 处理`HttpResponseMessage`以获取响应数据。

3.3 上传文件

1. 创建`HttpClient`实例。
2. 创建`MultipartFormDataContent`，添加`ByteArrayContent`作为文件内容。
3. 使用`PostAsync`方法发送包含文件的请求。
4. 处理`HttpResponseMessage`以获取响应数据。

3.4 下载文件

1. 创建`HttpClient`实例。
2. 使用`GetByteArrayAsync`方法获取文件的字节数组。
3. 使用`File.WriteAllBytes`保存文件。

3.5 设置Authorization

1. 创建`HttpClient`实例。
2. 设置`DefaultRequestHeaders.Authorization`属性为`AuthenticationHeaderValue`。

3.6 设置Cookie

1. 创建`HttpClient`实例。
2. 创建`CookieContainer`和`HttpClientHandler`实例。
3. 在`HttpClientHandler`中设置`CookieContainer`。
4. 使用包含`HttpClientHandler`的`HttpClient`实例发送请求。

4. 实例源代码

```
// 示例代码请根据实际情况修改URL和文件路径

// 发送GET请求
using (HttpClient client = new HttpClient())
{
    HttpResponseMessage response = await client.GetAsync("https://example.com/api/resource");
    // 处理响应
}

// 发送POST请求
using (HttpClient client = new HttpClient())
{
    var content = new StringContent("data to send", Encoding.UTF8, "application/json");
    HttpResponseMessage response = await client.PostAsync("https://example.com/api/resource", content);
    // 处理响应
}

// 上传文件
using (HttpClient client = new HttpClient())
{
    var content = new MultipartFormDataContent();
    content.Add(new ByteArrayContent(File.ReadAllBytes("file.txt")), "file", "file.txt");
    HttpResponseMessage response = await client.PostAsync("https://example.com/api/upload", content);
    // 处理响应
}

// 下载文件
using (HttpClient client = new HttpClient())
{
    byte[] fileData = await client.GetByteArrayAsync("https://example.com/api/download");
    File.WriteAllBytes("downloaded_file.txt", fileData);
}

// 设置Authorization
using (HttpClient client = new HttpClient())
{
    client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", "your_access_token");
    HttpResponseMessage response = await client.GetAsync("https://example.com/api/resource");
    // 处理响应
}

// 设置Cookie
using (HttpClient client = new HttpClient())
{
    var cookieContainer = new CookieContainer();
    var handler = new HttpClientHandler { CookieContainer = cookieContainer };
    using (var httpClient = new HttpClient(handler))
    {
        HttpResponseMessage response = await client.GetAsync("https://example.com/api/resource");
        // 处理响应
    }
}
```

5. 注意事项及建议

- 使用`using`语句确保及时释放`HttpClient`资源。
- 异步方法应避免在同步上下文中使用，以提高性能。

- 注意处理HTTP响应状态码和异常，确保请求的正确执行。
- 文件上传和下载时，考虑使用流式传输以节省内存。

通过`HttpClient`，我们能够在C#中方便地进行HTTP请求操作，包括GET、POST请求、文件上传和下载，以及设置Authorization、cookie等。确保使用异步操作和适当的异常处理，能够保证请求的稳定性和性能。