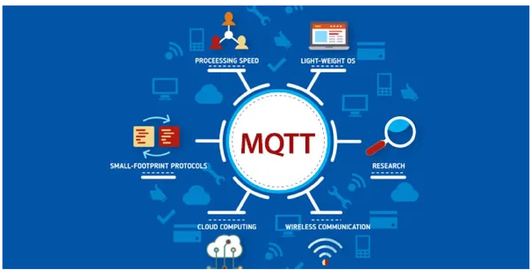
.net下优秀的MQTT框架MQTTnet使用方法，物联网通讯必备类库

作者：微信公众号：【架构师老卢】 11-15 15:35



概述： MQTTnet 是一个用于.NET的高性能MQTT类库，实现了MQTT协议的各个层级，包括连接、会话、发布/订阅、QoS（服务质量）等。

MQTTnet 是一个高性能的MQTT类库，支持.NET Core和.NET Framework。

MQTTnet 原理：

MQTTnet 是一个用于.NET的高性能MQTT类库，实现了MQTT协议的各个层级，包括连接、会话、发布/订阅、QoS（服务质量）等。其原理涉及以下关键概念：

1. **MqttClient：** MqttClient 是MQTTnet库中表示客户端的主要类。它负责与MQTT服务器建立连接，并处理消息的发布和订阅。
2. **MqttServer：** MqttServer 则表示MQTT服务器，负责接受客户端的连接，管理连接状态，并转发消息到相应的订阅者。
3. **消息处理：** MQTT消息分为发布消息和订阅消息。发布消息由客户端发送到服务器，然后由服务器广播给所有订阅者。
4. **QoS（服务质量）：** MQTT支持不同级别的服务质量，包括0、1和2。MQTTnet允许你根据需要选择适当的QoS级别。
5. **异步通信：** MQTTnet广泛使用异步编程模型，允许并发处理多个连接，提高性能。

MQTTnet 优点：

1. **高性能：** MQTTnet被设计为高性能的MQTT库，适用于处理大量的消息和连接。
2. **跨平台：** 支持.NET Core和.NET Framework，使其可以在不同的操作系统上运行。
3. **灵活性：** 提供了许多配置选项，允许你根据应用程序的需求进行调整。
4. **WebSocket支持：** 支持通过WebSocket协议进行通信，适用于Web应用程序。
5. **活跃社区：** MQTTnet有一个活跃的社区，提供了文档、示例和支持。

使用方法（服务端、客户端、WEB端）：

下面是一个简单的示例，演示如何在.NET Core中使用MQTTnet创建一个基本的MQTT服务端和客户端。请注意，这个示例只是为了演示基本概念，实际应用中可能需要更多的配置和错误处理。

服务端示例：

```
using System;
using MQTTnet;
using MQTTnet.Server;

class Program
{
    static async System.Threading.Tasks.Task Main(string[] args)
    {
        // 创建服务端配置
        var optionsBuilder = new MqttServerOptionsBuilder()
            .WithDefaultEndpointPort(1883)
            .WithConnectionValidator(c =>
            {
                Console.WriteLine($"Client connected: {c.ClientId}");
                // 可以在这里添加连接验证逻辑
            });

        // 创建MQTT服务器实例
        var mqttServer = new MqttFactory().CreateMqttServer();

        // 处理连接成功事件
        mqttServer.ClientConnectedHandler = new MqttServerClientConnectedHandlerDelegate(e =>
        {
            Console.WriteLine($"Client connected: {e.ClientId}");
        });

        // 处理连接断开事件
        mqttServer.ClientDisconnectedHandler = new MqttServerClientDisconnectedHandlerDelegate(e =>
        {
            Console.WriteLine($"Client disconnected: {e.ClientId}");
        });

        // 处理接收到消息事件
        mqttServer.ApplicationMessageReceivedHandler = new MqttApplicationMessageReceivedHandlerDelegate(e =>
        {
            Console.WriteLine($"Received message from client {e.ClientId}: {e.ApplicationMessage.Payload}");
        });

        // 启动MQTT服务器
        await mqttServer.StartAsync(optionsBuilder.Build());

        Console.WriteLine("MQTT Server已启动。按任意键退出。");
        Console.ReadLine();

        // 停止MQTT服务器
        await mqttServer.StopAsync();
    }
}
```

客户端示例：

```
using System;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using MQTTnet;
using MQTTnet.Client;
using MQTTnet.Client.Options;

class Program
{
    static async Task Main(string[] args)
    {
        // 创建客户端配置
        var options = new MqttClientOptionsBuilder()
            .WithTcpServer("localhost", 1883)
            .WithClientId("Client1") // 客户端ID
            .Build();

        // 创建MQTT客户端实例
        var mqttClient = new MqttFactory().CreateMqttClient();

        // 处理连接成功事件
        mqttClient.UseConnectedHandler(e =>
        {
            Console.WriteLine("Connected to MQTT Broker");
        });

        // 处理连接断开事件
        mqttClient.UseDisconnectedHandler(e =>
        {
            Console.WriteLine("Disconnected from MQTT Broker");
        });

        // 处理接收到消息事件
        mqttClient.UseApplicationMessageReceivedHandler(e =>
        {
            Console.WriteLine($"Received message: {e.ApplicationMessage.Payload}");
        });

        // 连接到MQTT服务器
        await mqttClient.ConnectAsync(options, CancellationToken.None);

        // 发布消息
        var message = new MqttApplicationMessageBuilder()
            .WithTopic("topic/test")
            .WithPayload("Hello, MQTT!")
            .WithExactlyOnceQoS()
            .WithRetainFlag()
            .Build();

        await mqttClient.PublishAsync(message, CancellationToken.None);

        Console.WriteLine("Message published. Press any key to exit.");
        Console.ReadLine();

        // 断开与MQTT服务器的连接
        await mqttClient.DisconnectAsync();
    }
}
```

Web端示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="https://cdnjs.cloudflare.com/ajax/libs/mqtt/4.0.0/mqtt.min.js"></script>
    <title>MQTT Web Client</title>
</head>
<body>
    <h1>MQTT Web Client</h1>

    <script>
        // 连接到MQTT服务器
        const client = mqtt.connect('mqtt://your-mqtt-broker-url');

        // 当连接成功时的处理逻辑
        client.on('connect', function () {
            console.log('Connected to MQTT Broker');

            // 订阅主题
            client.subscribe('topic/test', function (err) {
                if (!err) {
                    console.log('Subscribed to topic/test');
                }
            });

            // 发布消息
            client.publish('topic/test', 'Hello, MQTT!');
        });

        // 当接收到消息时的处理逻辑
        client.on('message', function (topic, message) {
            console.log('Received message:', message.toString());
        });

        // 处理连接断开事件
        client.on('close', function () {
            console.log('Connection closed');
        });

        // 处理错误事件
        client.on('error', function (err) {
            console.error('Error:', err);
        });
    </script>
</body>
</html>
```

以上代码中增加了连接断开事件处理（`UseDisconnectedHandler`、Web端的`close`事件）和错误事件处理（Web端的`error`事件）。这些事件处理可以根据实际需求进一步扩展。