Если вы видите что-то необычное, просто сообщите мне.

mqtt

- How to use MQTT in Golang

How to use MQTT in Golang

Golang is a statically strongly typed, compiled, concurrent, and garbage-collecting programming language developed by Google. Go is expressive, clean, and efficient. Its concurrency mechanism makes it easy to write programs that maximize the use of multicore and network machines, and its innovative type system enables flexible and modular program construction. Go compiles quickly to machine code, but with the convenience of garbage collection and the power of runtime reflection. It is a fast, statically typed, compiled language, like a dynamically typed, interpreted language.

MQTT is a kind of lightweight IoT messaging protocol based on the publish/subscribe model, which can provide real-time and reliable messaging service for IoT devices, only using very little code and bandwidth. It is suitable for devices with limited hardware resources and the network environment with limited bandwidth. Therefore, MQTT protocol is widely used in IoT, mobile internet, IoV, electricity power, and other industries.

This article mainly introduces how to use paho.mqtt.golang client library in the Golang project, and implement the connection, subscription and messaging between the client and the MQTT broker.

Project initialization

This project is based on go 1.13.12 to develop and test.

go version go version go1.13.12 darwin/amd64 This project uses paho.mqtt.golang as MQTT client library, install:

go get github.com/eclipse/paho.mqtt.golang

Use of Go MQTT

This article will use the free public MQTT broker provided by EMQX. This service is based on the MQTT IoT cloud platform of EMQX to create. The access information of the server is as follows:

- Broker: broker.emqx.io

- TCP Port: 1883
- Websocket Port: 8083 Connect to the MQTT broker

```
package main

import (
    "fmt"
    mqtt "github.com/eclipse/paho.mqtt.golang"
    "time"
)

var messagePubHandler mqtt.MessageHandler = func(client mqtt.Client, msg mqtt.Message) {
    fmt.Printf("Received message: %s from topic: %s\n", msg.Payload(), msg.Topic())
}

var connectHandler mqtt.OnConnectHandler = func(client mqtt.Client) {
    fmt.Println("Connected")
}

var connectLostHandler mqtt.ConnectionLostHandler = func(client mqtt.Client, err error) {
    fmt.Printf("Connect lost: %v", err)
}

func main() {
    var broker = "broker.emqx.io"
    var port = 1883
    opts := mqtt.NewClientOptions()
    opts.AddBroker(fmt.Sprintf("tcp://%s:%d", broker, port))
    opts.SetClientID("go_mqtt_client")
    opts.SetUsername("emqx")
    opts.SetPassword("public")
    opts.SetDefaultPublishHandler(messagePubHandler)
    opts.OnConnect = connectHandler
    opts.OnConnectionLost = connectLostHandler
    client := mqtt.NewClient(opts)
    if token := client.Connect(); token.Wait() && token.Error() != nil {
        panic(token.Error())
  }
}
```

- ClientOptions: used to set broker, port, client id, username, password and other options.
- `messagePubHandler`: global MQTT pub message processing
- `connectHandler`: callback for the connection
- `connectLostHandler`: callback for connection loss If you want to use the TSL connection, you can use the following settings:

```
func NewTlsConfig() *tls.Config {
    certpool := x509.NewCertPool()
    ca, err := ioutil.ReadFile("ca.pem")
    if err != nil {
        log.Fatalln(err.Error())
    }
    certpool.AppendCertsFromPEM(ca)
    // Import client certificate/key pair
    clientKeyPair, err := tls.LoadX509KeyPair("client-crt.pem", "client-key.pem")
    if err != nil {
        panic(err)
    }
    return &tls.Config{
        RootCAs: certpool,
        ClientAuth: tls.NoClientCert,
        ClientCAs: nil,
        InsecureSkipVerify: true,
        Certificates: []tls.Certificate{clientKeyPair},
    }
}
```

If the client certificate is not set, it can be set as follows:

```
func NewTlsConfig() *tls.Config {
    certpool := x509.NewCertPool()
    ca, err := ioutil.ReadFile("ca.pem")
    if err != nil {
        log.Fatalln(err.Error())
    }
    certpool.AppendCertsFromPEM(ca)
    return &tls.Config{
        RootCAs: certpool,
}
```

Then set TLS

```
var broker = "broker.emqx.io"
var port = 8883
opts := mqtt.NewClientOptions()
opts.AddBroker(fmt.Sprintf("ssl://%s:%d", broker, port))
tlsConfig := NewTlsConfig()
opts.SetTLSConfig(tlsConfig)
// other options
Subscription
func sub(client mqtt.Client) {
    topic := "topic/test"
    token := client.Subscribe(topic, 1, nil)
    token.Wait()
    fmt.Printf("Subscribed to topic %s", topic)
}
```

Publish messages

```
func publish(client mqtt.Client) {
    num := 10
    for i := 0; i < num; i++ {
        text := fmt.Sprintf("Message %d", i)
        token := client.Publish("topic/test", 0, false, text)
        token.Wait()
        time.Sleep(time.Second)
    }
}
```

Test We use the following code for testing.

package main

import ("fmt" mqtt "github.com/eclipse/paho.mqtt.golang" "log" "time")

var messagePubHandler mqtt.MessageHandler = func(client mqtt.Client, msg mqtt.Message) { fmt.Printf("Received message: %s from topic: %s\n", msg.Payload(), msg.Topic()) }

var connectHandler mqtt.OnConnectHandler = func(client mqtt.Client) { fmt.Println("Connected") }

var connectLostHandler mqtt.ConnectionLostHandler = func(client mqtt.Client, err error) { fmt.Printf("Connect lost: %v", err) }

func main() { var broker = "broker.emqx.io" var port = 1883 opts := mqtt.NewClientOptions() opts.AddBroker(fmt.Sprintf("tcp://%s:%d", broker, port)) opts.SetClientID("go_mqtt_client") opts.SetUsername("emqx") opts.SetPassword("public") opts.SetDefaultPublishHandler(messagePubHandler) opts.OnConnect = connectHandler opts.OnConnectionLost = connectLostHandler client := mqtt.NewClient(opts) if token := client.Connect(); token.Wait() && token.Error() != nil { panic(token.Error()) }

```
sub(client)
publish(client)


client.Disconnect(250)
```

}

func publish(client mqtt.Client) { num := 10 for i := 0; i < num; i++ { text := fmt.Sprintf("Message %d", i) token := client.Publish("topic/test", 0, false, text) token.Wait() time.Sleep(time.Second) } }

func sub(client mqtt.Client) { topic := "topic/test" token := client.Subscribe(topic, 1, nil) token.Wait() fmt.Printf("Subscribed to topic: %s", topic) } Run code, we can see that the MQTT connection and subscription are successfully, and we can successfully receive the message of subscribing topic. gomqtt.png

Summary So far, we have completed using the paho.mqtt.golang client to connect to the public MQTT broker and have implemented the connection, message publishing and subscription between the test client and the MQTT broker.

Next, you can check out The Easy-to-understand Guide to MQTT Protocol series of articles provided by EMQ to learn about MQTT protocol features, explore more advanced applications of MQTT, and get started with MQTT application and service development.