MQTT 消息类型

概述

ACL MQTT 模块为 MQTT 3.1.1 协议的所有消息类型提供了对应的 C++ 类。所有消息类都继承自 `mqtt_message` 基类，支持消息的序列化和反序列化。

消息类型枚举

```
typedef enum {
    MQTT_RESERVED_MIN   = 0,   // 保留
    MQTT_CONNECT        = 1,   // 客户端请求连接
    MQTT_CONNACK        = 2,   // 连接确认
    MQTT_PUBLISH        = 3,   // 发布消息
    MQTT_PUBACK         = 4,   // 发布确认 (QoS 1)
    MQTT_PUBREC         = 5,   // 发布接收 (QoS 2)
    MQTT_PUBREL         = 6,   // 发布释放 (QoS 2)
    MQTT_PUBCOMP        = 7,   // 发布完成 (QoS 2)
    MQTT_SUBSCRIBE      = 8,   // 订阅主题
    MQTT_SUBACK         = 9,   // 订阅确认
    MQTT_UNSUBSCRIBE    = 10,  // 取消订阅
    MQTT_UNSUBACK       = 11,  // 取消订阅确认
    MQTT_PINGREQ        = 12,  // 心跳请求
    MQTT_PINGRESP       = 13,  // 心跳响应
    MQTT_DISCONNECT     = 14,  // 断开连接
    MQTT_RESERVED_MAX   = 15,  // 保留
} mqtt_type_t;
```

QoS 级别

```
typedef enum {
    MQTT_QOS0 = 0x0,  // 最多一次交付
    MQTT_QOS1 = 0x1,  // 至少一次交付
    MQTT_QOS2 = 0x2,  // 恰好一次交付
} mqtt_qos_t;
```

mqtt_message - 消息基类

所有 MQTT 消息类型的基类。

主要方法

```
class mqtt_message {
public:
    // 构造函数
```

```
    explicit mqtt_message(mqtt_type_t type);
    explicit mqtt_message(const mqtt_header& header);

    virtual ~mqtt_message();

    // 序列化为字节流
    virtual bool to_string(string& out) = 0;

    // 从字节流反序列化
    virtual int update(const char* data, int dlen) = 0;

    // 检查是否解析完成
    virtual bool finished() const { return false; }

    // 获取消息头
    mqtt_header& get_header();
    const mqtt_header& get_header() const;

    // 创建消息对象
    static mqtt_message* create_message(const mqtt_header& header);
};
```

连接相关消息

mqtt_connect - 连接请求

客户端向服务器发起连接请求。

方法

```
class mqtt_connect : public mqtt_message {
public:
    mqtt_connect();
    mqtt_connect(const mqtt_header& header);

    // 设置方法
    void set_keep_alive(unsigned short keep_alive);
    void set_cid(const char* cid);
    void set_username(const char* name);
    void set_passwd(const char* passwd);
    void set_will_qos(mqtt_qos_t qos);
    void set_will_topic(const char* topic);
    void set_will_msg(const char* msg);
    void clean_session();

    // 获取方法
    unsigned short get_keep_alive() const;
    const char* get_cid() const;
    const char* get_username() const;
    const char* get_passwd() const;
    mqtt_qos_t get_will_qos() const;
```

```
    const char* get_will_topic() const;
    const char* get_will_msg() const;
    bool session_cleaned() const;
};
```

使用示例

```
// 创建连接请求
acl::mqtt_connect conn;
conn.set_cid("client_001")              // 客户端 ID
    .set_username("admin")               // 用户名
    .set_passwd("123456")                // 密码
    .set_keep_alive(60);                 // 心跳间隔（秒）

// 设置遗嘱消息
conn.set_will_topic("status/client_001")
    .set_will_msg("offline")
    .set_will_qos(acl::MQTT_QOS1);

// 清除会话
conn.clean_session();

// 发送
client.send(conn);
```

mqtt_connack - 连接确认

服务器对客户端连接请求的响应。

连接状态码

```
enum {
    MQTT_CONNACK_OK       = 0x00,  // 连接成功
    MQTT_CONNACK_ERR_VER  = 0x01,  // 不可接受的协议版本
    MQTT_CONNACK_ERR_CID  = 0x02,  // 客户端标识符被拒绝
    MQTT_CONNACK_ERR_SVR  = 0x03,  // 服务器不可用
    MQTT_CONNACK_ERR_AUTH = 0x04,  // 用户名或密码错误
    MQTT_CONNACK_ERR_DENY = 0x05,  // 未授权
};
```

方法

```
class mqtt_connack : public mqtt_message {
public:
    mqtt_connack();
    mqtt_connack(const mqtt_header& header);
```

```
    // 设置会话控制
    mqtt_connack& set_session(bool on);

    // 设置连接返回码
    mqtt_connack& set_connack_code(unsigned char code);

    // 获取会话状态
    bool get_session() const;

    // 获取连接返回码
    unsigned char get_connack_code() const;
};
```

使用示例

```
// 接收 CONNACK
acl::mqtt_message* msg = client.get_message();
if (msg->get_header().get_type() == acl::MQTT_CONNACK) {
    acl::mqtt_connack* connack = (acl::mqtt_connack*) msg;

    if (connack->get_connack_code() == acl::MQTT_CONNACK_OK) {
        printf("连接成功\n");
        if (connack->get_session()) {
            printf("会话已恢复\n");
        }
    } else {
        printf("连接失败: %d\n", connack->get_connack_code());
    }
}
delete msg;
```

发布相关消息

mqtt_publish - 发布消息

发布消息到指定主题。

方法

```
class mqtt_publish : public mqtt_message {
public:
    mqtt_publish();
    mqtt_publish(const mqtt_header& header);

    // 设置主题
    mqtt_publish& set_topic(const char* topic);
```

```
    // 设置消息 ID（QoS > 0 时必须）
    mqtt_publish& set_pkt_id(unsigned short id);

    // 设置载荷
    mqtt_publish& set_payload(unsigned len, const char* data = NULL);

    // 获取主题
    const char* get_topic() const;

    // 获取消息 ID
    unsigned short get_pkt_id() const;

    // 获取载荷长度
    unsigned get_payload_len() const;

    // 获取载荷
    const string& get_payload() const;
};
```

使用示例

```
// 发布 QoS 0 消息
acl::mqtt_publish pub0;
pub0.set_topic("sensor/temperature")
    .set_payload(4, "25.5");
pub0.get_header().set_qos(acl::MQTT_QOS0);
client.send(pub0);

// 发布 QoS 1 消息
acl::mqtt_publish pub1;
pub1.set_topic("sensor/humidity")
    .set_pkt_id(1)  // QoS > 0 需要设置消息 ID
    .set_payload(4, "60.2");
pub1.get_header().set_qos(acl::MQTT_QOS1);
client.send(pub1);

// 接收 PUBLISH
acl::mqtt_message* msg = client.get_message();
if (msg->get_header().get_type() == acl::MQTT_PUBLISH) {
    acl::mqtt_publish* pub = (acl::mqtt_publish*) msg;
    printf("主题: %s\n", pub->get_topic());
    printf("内容: %.*s\n",
           (int)pub->get_payload_len(),
           pub->get_payload().c_str());
    printf("QoS: %d\n", pub->get_header().get_qos());

    // QoS 1 需要发送 PUBACK
    if (pub->get_header().get_qos() == acl::MQTT_QOS1) {
        acl::mqtt_puback puback;
        puback.set_pkt_id(pub->get_pkt_id());
        client.send(puback);
```

```
    }
}
delete msg;
```

mqtt_puback - 发布确认（QoS 1）

QoS 1 消息的确认。

方法

```
class mqtt_puback : public mqtt_ack {
public:
    mqtt_puback();
    mqtt_puback(const mqtt_header& header);

    void set_pkt_id(unsigned short id);
    unsigned short get_pkt_id() const;
};
```

使用示例

```
// 发送 PUBACK
acl::mqtt_puback puback;
puback.set_pkt_id(message_id);
client.send(puback);
```

mqtt_pubrec - 发布接收（QoS 2）

QoS 2 消息的第一步确认。

```
class mqtt_pubrec : public mqtt_ack {
public:
    mqtt_pubrec();
    mqtt_pubrec(const mqtt_header& header);
};
```

mqtt_pubrel - 发布释放（QoS 2）

QoS 2 消息的第二步。

```
class mqtt_pubrel : public mqtt_ack {
public:
    mqtt_pubrel();
```

```
    mqtt_pubrel(const mqtt_header& header);
};
```

mqtt_pubcomp - 发布完成（QoS 2）

QoS 2 消息的最后确认。

```
class mqtt_pubcomp : public mqtt_ack {
public:
    mqtt_pubcomp();
    mqtt_pubcomp(const mqtt_header& header);
};
```

QoS 2 完整流程示例

```
// 发送方
acl::mqtt_publish pub;
pub.set_topic("important/data")
   .set_pkt_id(1)
   .set_payload(10, "critical");
pub.get_header().set_qos(acl::MQTT_QOS2);
client.send(pub);

// 等待 PUBREC
acl::mqtt_message* msg = client.get_message();
if (msg->get_header().get_type() == acl::MQTT_PUBREC) {
    acl::mqtt_pubrec* pubrec = (acl::mqtt_pubrec*) msg;

    // 发送 PUBREL
    acl::mqtt_pubrel pubrel;
    pubrel.set_pkt_id(pubrec->get_pkt_id());
    client.send(pubrel);
}
delete msg;

// 等待 PUBCOMP
msg = client.get_message();
if (msg->get_header().get_type() == acl::MQTT_PUBCOMP) {
    printf("QoS 2 消息发送完成\n");
}
delete msg;
```

订阅相关消息

mqtt_subscribe - 订阅主题

订阅一个或多个主题。

方法

```
class mqtt_subscribe : public mqtt_message {
public:
    mqtt_subscribe();
    mqtt_subscribe(const mqtt_header& header);

    // 设置消息 ID
    mqtt_subscribe& set_pkt_id(unsigned short id);

    // 添加主题
    mqtt_subscribe& add_topic(const char* topic, mqtt_qos_t qos);

    // 获取消息 ID
    unsigned short get_pkt_id() const;

    // 获取所有主题
    const std::vector<std::string>& get_topics() const;

    // 获取所有 QoS
    const std::vector<mqtt_qos_t>& get_qoses() const;
};
```

使用示例

```
// 订阅多个主题
acl::mqtt_subscribe sub;
sub.set_pkt_id(1)
   .add_topic("sensor/temperature", acl::MQTT_QOS1)
   .add_topic("sensor/humidity", acl::MQTT_QOS1)
   .add_topic("sensor/pressure", acl::MQTT_QOS2);

client.send(sub);

// 解析订阅请求
const std::vector<std::string>& topics = sub.get_topics();
const std::vector<acl::mqtt_qos_t>& qoses = sub.get_qoses();

for (size_t i = 0; i < topics.size(); i++) {
    printf("订阅: %s (QoS %d)\n", topics[i].c_str(), qoses[i]);
}
```

主题通配符

MQTT 支持两种通配符：

- `+` - 单级通配符，匹配一个层级

    - `sensor/+/temperature` 匹配 `sensor/room1/temperature`、`sensor/room2/temperature`

- `#` - 多级通配符，匹配多个层级（只能在末尾）
    - `sensor/#` 匹配 `sensor/temperature`、`sensor/room1/temperature`

```
acl::mqtt_subscribe sub;
sub.set_pkt_id(1)
   .add_topic("sensor/+/temperature", acl::MQTT_QOS1)  // 单级通配符
   .add_topic("control/#", acl::MQTT_QOS1);             // 多级通配符

client.send(sub);
```

mqtt_suback - 订阅确认

服务器对订阅请求的确认。

方法

```
class mqtt_suback : public mqtt_message {
public:
    mqtt_suback();
    mqtt_suback(const mqtt_header& header);

    // 设置消息 ID
    mqtt_suback& set_pkt_id(unsigned short id);

    // 添加主题 QoS
    mqtt_suback& add_topic_qos(mqtt_qos_t qos);
    mqtt_suback& add_topic_qos(const std::vector<mqtt_qos_t>& qoses);

    // 获取消息 ID
    unsigned short get_pkt_id() const;

    // 获取所有 QoS
    const std::vector<mqtt_qos_t>& get_qoses() const;
};
```

使用示例

```
// 接收 SUBACK
acl::mqtt_message* msg = client.get_message();
if (msg->get_header().get_type() == acl::MQTT_SUBACK) {
    acl::mqtt_suback* suback = (acl::mqtt_suback*) msg;
    printf("订阅成功，消息ID: %d\n", suback->get_pkt_id());
```

```
    const std::vector<acl::mqtt_qos_t>& qoses = suback->get_qoses();
    for (size_t i = 0; i < qoses.size(); i++) {
        if (qoses[i] == 0x80) {
            printf("主题 %lu 订阅失败\n", i);
        } else {
            printf("主题 %lu 订阅成功, QoS: %d\n", i, qoses[i]);
        }
    }
}
delete msg;
```

mqtt_unsubscribe - 取消订阅

取消订阅一个或多个主题。

方法

```
class mqtt_unsubscribe : public mqtt_message {
public:
    mqtt_unsubscribe();
    mqtt_unsubscribe(const mqtt_header& header);

    // 设置消息 ID
    mqtt_unsubscribe& set_pkt_id(unsigned short id);

    // 添加主题
    mqtt_unsubscribe& add_topic(const char* topic);

    // 获取消息 ID
    unsigned short get_pkt_id() const;

    // 获取所有主题
    const std::vector<std::string>& get_topics() const;
};
```

使用示例

```
// 取消订阅
acl::mqtt_unsubscribe unsub;
unsub.set_pkt_id(2)
     .add_topic("sensor/temperature")
     .add_topic("sensor/humidity");

client.send(unsub);
```

mqtt_unsuback - 取消订阅确认

服务器对取消订阅请求的确认。

```
class mqtt_unsuback : public mqtt_ack {
public:
    mqtt_unsuback();
    mqtt_unsuback(const mqtt_header& header);

    void set_pkt_id(unsigned short id);
    unsigned short get_pkt_id() const;
};
```

心跳消息

mqtt_pingreq - 心跳请求

客户端发送心跳请求以保持连接。

```
class mqtt_pingreq : public mqtt_message {
public:
    mqtt_pingreq();
    mqtt_pingreq(const mqtt_header& header);
};
```

使用示例

```
// 发送心跳
acl::mqtt_pingreq ping;
client.send(ping);
```

mqtt_pingresp - 心跳响应

服务器对心跳请求的响应。

```
class mqtt_pingresp : public mqtt_message {
public:
    mqtt_pingresp();
    mqtt_pingresp(const mqtt_header& header);
};
```

心跳机制

```

```

```
// 客户端心跳示例
time_t last_ping = time(NULL);
int keep_alive = 60;  // 秒

while (true) {
    time_t now = time(NULL);
    if (now - last_ping >= keep_alive) {
        acl::mqtt_pingreq ping;
        if (client.send(ping)) {
            last_ping = now;
        }
    }

    // 接收消息...
}
```

断开连接

mqtt_disconnect - 断开连接

客户端主动断开连接。

```
class mqtt_disconnect : public mqtt_message {
public:
    mqtt_disconnect();
    mqtt_disconnect(const mqtt_header& header);
};
```

使用示例

```
// 优雅断开连接
acl::mqtt_disconnect disc;
client.send(disc);

// 关闭连接
client.close();
```

消息 ID 管理

需要确认的消息必须设置有效的消息 ID（1-65535）。

```
class MessageIdManager {
public:
    MessageIdManager() : next_id_(1) {}
```

```
    unsigned short get_next_id() {
        unsigned short id = next_id_++;
        if (next_id_ == 0 || next_id_ > 65535) {
            next_id_ = 1;
        }
        return id;
    }

private:
    unsigned short next_id_;
};

// 使用示例
MessageIdManager id_mgr;

acl::mqtt_publish pub;
pub.set_pkt_id(id_mgr.get_next_id())
   .set_topic("test/topic")
   .set_payload(4, "data");
```

工具函数

```
// 获取消息类型描述
const char* mqtt_type_desc(mqtt_type_t type);

// 获取 QoS 描述
const char* mqtt_qos_desc(mqtt_qos_t qos);

// 使用示例
printf("消息类型: %s\n", acl::mqtt_type_desc(acl::MQTT_PUBLISH));
printf("QoS: %s\n", acl::mqtt_qos_desc(acl::MQTT_QOS1));
```

完整示例：消息处理框架

```
class MqttMessageHandler {
public:
    void handle(acl::mqtt_message* msg) {
        if (msg == NULL) {
            return;
        }

        switch (msg->get_header().get_type()) {
        case acl::MQTT_CONNECT:
            handle_connect((acl::mqtt_connect*)msg);
            break;
        case acl::MQTT_CONNACK:
            handle_connack((acl::mqtt_connack*)msg);
            break;
        case acl::MQTT_PUBLISH:
```

```
                handle_publish((acl::mqtt_publish*)msg);
                break;
            case acl::MQTT_PUBACK:
                handle_puback((acl::mqtt_puback*)msg);
                break;
            case acl::MQTT_SUBSCRIBE:
                handle_subscribe((acl::mqtt_subscribe*)msg);
                break;
            case acl::MQTT_SUBACK:
                handle_suback((acl::mqtt_suback*)msg);
                break;
            case acl::MQTT_PINGREQ:
                handle_pingreq((acl::mqtt_pingreq*)msg);
                break;
            case acl::MQTT_PINGRESP:
                handle_pingresp((acl::mqtt_pingresp*)msg);
                break;
            case acl::MQTT_DISCONNECT:
                handle_disconnect((acl::mqtt_disconnect*)msg);
                break;
            default:
                printf("未知消息类型: %d\n", msg->get_header().get_type());
                break;
            }

            delete msg;
        }

    private:
        void handle_connect(acl::mqtt_connect* msg) { /* ... */ }
        void handle_connack(acl::mqtt_connack* msg) { /* ... */ }
        void handle_publish(acl::mqtt_publish* msg) { /* ... */ }
        void handle_puback(acl::mqtt_puback* msg) { /* ... */ }
        void handle_subscribe(acl::mqtt_subscribe* msg) { /* ... */ }
        void handle_suback(acl::mqtt_suback* msg) { /* ... */ }
        void handle_pingreq(acl::mqtt_pingreq* msg) { /* ... */ }
        void handle_pingresp(acl::mqtt_pingresp* msg) { /* ... */ }
        void handle_disconnect(acl::mqtt_disconnect* msg) { /* ... */ }
    };
```

相关文档

- MQTT 同步客户端
- MQTT 异步客户端
- 使用示例