MQTT 使用示例

本文档提供了完整的 MQTT 客户端和服务器示例代码。

目录

简单的同步客户端

```
#include "acl_cpp/lib_acl.hpp"
#include "acl_cpp/mqtt/mqtt_client.hpp"
#include "acl_cpp/mqtt/mqtt_connect.hpp"
#include "acl_cpp/mqtt/mqtt_connack.hpp"
#include "acl_cpp/mqtt/mqtt_publish.hpp"
#include "acl_cpp/mqtt/mqtt_puback.hpp"
#include "acl_cpp/mqtt/mqtt_disconnect.hpp"

int main(void) {
    // 开启日志
    acl::log::stdout_open(true);

    // 创建 MQTT 客户端
    acl::mqtt_client client("127.0.0.1|1883", 10, 30);

    // 打开连接
    if (!client.open()) {
        printf("连接服务器失败\n");
        return 1;
    }
    printf("连接服务器成功\n");

    // 构建并发送 CONNECT 消息
    acl::mqtt_connect conn_msg;
    conn_msg.set_cid("simple_client")
            .set_username("admin")
            .set_passwd("password")
            .set_keep_alive(60);

    if (!client.send(conn_msg)) {
        printf("发送 CONNECT 消息失败\n");
        return 1;
    }
    printf("发送 CONNECT 消息成功\n");

    // 接收 CONNACK 消息
    acl::mqtt_message* msg = client.get_message();
    if (msg == NULL) {
        printf("接收 CONNACK 失败\n");
        return 1;
    }

    if (msg->get_header().get_type() != acl::MQTT_CONNACK) {
        printf("期望 CONNACK, 但收到类型 %d\n", msg-
>get_header().get_type());
        delete msg;
        return 1;
    }

    acl::mqtt_connack* connack = (acl::mqtt_connack*) msg;
    if (connack->get_connack_code() != acl::MQTT_CONNACK_OK) {
        printf("连接被拒绝, 错误码: %d\n", connack->get_connack_code());
        delete msg;
        return 1;
    }
    printf("连接确认成功\n");
    delete msg;

    // 发布一条消息
    acl::mqtt_publish pub_msg;
    pub_msg.set_topic("test/topic")
           .set_pkt_id(1)
           .set_payload(11, "Hello MQTT!");
    pub_msg.get_header().set_qos(acl::MQTT_QOS1);

    if (!client.send(pub_msg)) {
        printf("发送 PUBLISH 消息失败\n");
        return 1;
    }
    printf("发送 PUBLISH 消息成功\n");

    // 接收 PUBACK
    msg = client.get_message();
    if (msg && msg->get_header().get_type() == acl::MQTT_PUBACK) {
        acl::mqtt_puback* puback = (acl::mqtt_puback*) msg;
        printf("收到 PUBACK, 消息ID: %d\n", puback->get_pkt_id());
        delete msg;
    }

    // 发送 DISCONNECT 并关闭连接
    acl::mqtt_disconnect disc;
    client.send(disc);

    printf("完成\n");
    return 0;
}
```

简单的异步客户端

```
#include "acl_cpp/lib_acl.hpp"
#include "acl_cpp/mqtt/mqtt_aclient.hpp"
#include "acl_cpp/mqtt/mqtt_connect.hpp"
#include "acl_cpp/mqtt/mqtt_connack.hpp"
#include "acl_cpp/mqtt/mqtt_publish.hpp"

class simple_mqtt_client : public acl::mqtt_aclient {
public:
    simple_mqtt_client(acl::aio_handle& handle)
        : mqtt_aclient(handle, NULL) {}

protected:
    // 对象销毁
    void destroy() override {
        printf("客户端对象销毁\n");
        delete this;
    }

    // 连接成功
    bool on_open() override {
        printf("连接成功, 发送 CONNECT\n");

        acl::mqtt_connect conn_msg;
        conn_msg.set_cid("async_simple_client")
                .set_keep_alive(60);

        return send(conn_msg);
    }

    // 接收消息
    bool on_body(const acl::mqtt_message& msg) override {
        printf("收到消息, 类型: %d\n", msg.get_header().get_type());

        switch (msg.get_header().get_type()) {
        case acl::MQTT_CONNACK:
            return handle_connack((const acl::mqtt_connack&) msg);
        case acl::MQTT_PUBACK: {
            const acl::mqtt_puback& puback = (const acl::mqtt_puback&)
msg;
            printf("收到 PUBACK, 消息ID: %d\n", puback.get_pkt_id());

            // 任务完成, 关闭连接
            close();
            break;
        }
        default:
            break;
        }

        return true;
    }

    // 连接断开
    void on_disconnect() override {
        printf("连接已断开\n");
    }

private:
    bool handle_connack(const acl::mqtt_connack& connack) {
        if (connack.get_connack_code() != acl::MQTT_CONNACK_OK) {
            printf("连接被拒绝: %d\n", connack.get_connack_code());
            return false;
        }

        printf("连接确认成功，发布消息\n");

        // 发布消息
        acl::mqtt_publish pub_msg;
        pub_msg.set_topic("test/topic")
               .set_pkt_id(1)
               .set_payload(11, "Hello MQTT!");
        pub_msg.get_header().set_qos(acl::MQTT_QOS1);

        return send(pub_msg);
    }
};

int main(void) {
    acl::log::stdout_open(true);

    // 创建 AIO 事件句柄
    acl::aio_handle handle(acl::ENGINE_KERNEL);

    // 创建客户端
    simple_mqtt_client* client = new simple_mqtt_client(handle);

    // 连接服务器
    if (!client->open("127.0.0.1|1883", 10, 30)) {
        printf("启动连接失败\n");
        client->destroy();
        return 1;
    }

    // 事件循环
    while (true) {
        handle.check();
    }

    return 0;
}
```

MQTT 发布者

```
#include "acl_cpp/lib_acl.hpp"
#include "acl_cpp/mqtt/mqtt_client.hpp"
#include "acl_cpp/mqtt/mqtt_connect.hpp"
#include "acl_cpp/mqtt/mqtt_connack.hpp"
#include "acl_cpp/mqtt/mqtt_publish.hpp"
#include "acl_cpp/mqtt/mqtt_puback.hpp"
#include "acl_cpp/mqtt/mqtt_pingreq.hpp"
#include "acl_cpp/mqtt/mqtt_pingresp.hpp"

class MqttPublisher {
public:
    MqttPublisher(const char* addr, const char* client_id)
        : client_(addr, 10, 30)
        , client_id_(client_id)
        , pkt_id_(1)
        , connected_(false) {
    }

    bool connect(const char* username = NULL, const char* password = NULL)
{
        if (!client_.open()) {
            printf("连接服务器失败\n");
            return false;
        }

        // 发送 CONNECT
        acl::mqtt_connect conn_msg;
        conn_msg.set_cid(client_id_.c_str())
                .set_keep_alive(60);

        if (username) {
            conn_msg.set_username(username);
        }
        if (password) {
            conn_msg.set_passwd(password);
        }

        if (!client_.send(conn_msg)) {
            printf("发送 CONNECT 失败\n");
            return false;
        }

        // 接收 CONNACK
        acl::mqtt_message* msg = client_.get_message();
        if (msg == NULL || msg->get_header().get_type() !=
acl::MQTT_CONNACK) {
            printf("接收 CONNACK 失败\n");
            if (msg) delete msg;
            return false;
        }

        acl::mqtt_connack* connack = (acl::mqtt_connack*) msg;
        bool ok = (connack->get_connack_code() == acl::MQTT_CONNACK_OK);
        delete msg;

        if (ok) {
            connected_ = true;
            printf("连接成功\n");
        } else {
            printf("连接被拒绝\n");
        }

        return ok;
    }

    bool publish(const char* topic, const char* payload,
                 acl::mqtt_qos_t qos = acl::MQTT_QOS1) {
        if (!connected_) {
            printf("未连接到服务器\n");
            return false;
        }

        acl::mqtt_publish pub_msg;
        pub_msg.set_topic(topic)
               .set_payload(strlen(payload), payload);
        pub_msg.get_header().set_qos(qos);

        if (qos > acl::MQTT_QOS0) {
            pub_msg.set_pkt_id(pkt_id_++);
            if (pkt_id_ == 0) {
                pkt_id_ = 1;
            }
        }

        if (!client_.send(pub_msg)) {
            printf("发送 PUBLISH 失败\n");
            return false;
        }

        printf("发布消息: %s -> %s (QoS %d)\n", topic, payload, qos);

        // QoS 1 需要等待 PUBACK
        if (qos == acl::MQTT_QOS1) {
            acl::mqtt_message* msg = client_.get_message();
            if (msg && msg->get_header().get_type() == acl::MQTT_PUBACK) {
                printf("收到 PUBACK\n");
                delete msg;
            }
        }

        return true;
    }
    void disconnect() {
        if (connected_) {
            acl::mqtt_disconnect disc;
            client_.send(disc);
            connected_ = false;
        }
    }

private:
    acl::mqtt_client client_;
    std::string client_id_;
    unsigned short pkt_id_;
    bool connected_;
};

int main(void) {
    acl::log::stdout_open(true);

    MqttPublisher publisher("127.0.0.1|1883", "publisher_001");

    if (!publisher.connect("admin", "password")) {
        return 1;
    }

    // 发布多条消息
    for (int i = 0; i < 5; i++) {
        char payload[64];
        snprintf(payload, sizeof(payload), "message_%d", i);
        publisher.publish("test/topic", payload);
        sleep(1);
    }

    publisher.disconnect();
    return 0;
}
```

MQTT 订阅者

```
#include "acl_cpp/lib_acl.hpp"
#include "acl_cpp/mqtt/mqtt_client.hpp"
#include "acl_cpp/mqtt/mqtt_connect.hpp"
#include "acl_cpp/mqtt/mqtt_connack.hpp"
#include "acl_cpp/mqtt/mqtt_subscribe.hpp"
#include "acl_cpp/mqtt/mqtt_suback.hpp"
#include "acl_cpp/mqtt/mqtt_publish.hpp"
#include "acl_cpp/mqtt/mqtt_puback.hpp"

class MqttSubscriber {
public:
    MqttSubscriber(const char* addr, const char* client_id)
```

```
        : client_(addr, 10, 30)
        , client_id_(client_id)
        , pkt_id_(1)
        , connected_(false) {
    }

    bool connect(const char* username = NULL, const char* password = NULL)
{
        if (!client_.open()) {
            printf("连接服务器失败\n");
            return false;
        }

        acl::mqtt_connect conn_msg;
        conn_msg.set_cid(client_id_.c_str())
                .set_keep_alive(60);

        if (username) {
            conn_msg.set_username(username);
        }
        if (password) {
            conn_msg.set_passwd(password);
        }

        if (!client_.send(conn_msg)) {
            printf("发送 CONNECT 失败\n");
            return false;
        }

        acl::mqtt_message* msg = client_.get_message();
        if (msg == NULL || msg->get_header().get_type() !=
acl::MQTT_CONNACK) {
            printf("接收 CONNACK 失败\n");
            if (msg) delete msg;
            return false;
        }

        acl::mqtt_connack* connack = (acl::mqtt_connack*) msg;
        bool ok = (connack->get_connack_code() == acl::MQTT_CONNACK_OK);
        delete msg;

        if (ok) {
            connected_ = true;
            printf("连接成功\n");
        }

        return ok;
    }

    bool subscribe(const char* topic, acl::mqtt_qos_t qos =
acl::MQTT_QOS1) {
        if (!connected_) {
            printf("未连接到服务器\n");
            return false;
        }

        acl::mqtt_subscribe sub_msg;
        sub_msg.set_pkt_id(pkt_id_++)
               .add_topic(topic, qos);

        if (!client_.send(sub_msg)) {
            printf("发送 SUBSCRIBE 失败\n");
            return false;
        }

        printf("订阅主题: %s (QoS %d)\n", topic, qos);

        // 接收 SUBACK
        acl::mqtt_message* msg = client_.get_message();
        if (msg && msg->get_header().get_type() == acl::MQTT_SUBACK) {
            acl::mqtt_suback* suback = (acl::mqtt_suback*) msg;
            printf("订阅确认, 消息ID: %d\n", suback->get_pkt_id());
            delete msg;
            return true;
        }

        if (msg) delete msg;
        return false;
    }

    void loop() {
        printf("开始接收消息...\n");

        while (true) {
            acl::mqtt_message* msg = client_.get_message();
            if (msg == NULL) {
                printf("连接断开\n");
                break;
            }

            if (msg->get_header().get_type() == acl::MQTT_PUBLISH) {
                handle_publish((acl::mqtt_publish*) msg);
            }

            delete msg;
        }
    }

private:
    void handle_publish(acl::mqtt_publish* pub) {
        printf("\n收到消息:\n");
        printf("  主题: %s\n", pub->get_topic());
        printf("  QoS: %d\n", pub->get_header().get_qos());
        printf("  内容: %.*s\n",
               (int)pub->get_payload_len(),
               pub->get_payload().c_str());

        // QoS 1 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);
        }
    }

private:
    acl::mqtt_client client_;
    std::string client_id_;
    unsigned short pkt_id_;
    bool connected_;
};

int main(void) {
    acl::log::stdout_open(true);

    MqttSubscriber subscriber("127.0.0.1|1883", "subscriber_001");

    if (!subscriber.connect("admin", "password")) {
        return 1;
    }

    // 订阅多个主题
    subscriber.subscribe("test/topic");
    subscriber.subscribe("sensor/+");
    subscriber.subscribe("control/#");

    // 进入消息接收循环
    subscriber.loop();

    return 0;
}
```

MQTT 服务器

```
#include "acl_cpp/lib_acl.hpp"
#include "acl_cpp/mqtt/mqtt_client.hpp"
#include "acl_cpp/mqtt/mqtt_connect.hpp"
#include "acl_cpp/mqtt/mqtt_connack.hpp"
#include "acl_cpp/mqtt/mqtt_publish.hpp"
#include "acl_cpp/mqtt/mqtt_puback.hpp"
#include "acl_cpp/mqtt/mqtt_subscribe.hpp"
#include "acl_cpp/mqtt/mqtt_suback.hpp"
#include "acl_cpp/mqtt/mqtt_pingreq.hpp"
#include "acl_cpp/mqtt/mqtt_pingresp.hpp"
#include "acl_cpp/mqtt/mqtt_disconnect.hpp"
#include <map>
#include <set>

// 简单的主题匹配
```

```
bool topic_match(const std::string& filter, const std::string& topic) {
    // 简化实现，仅支持 # 通配符
    if (filter == topic) {
        return true;
    }

    size_t pos = filter.find('#');
    if (pos != std::string::npos) {
        return topic.compare(0, pos, filter, 0, pos) == 0;
    }

    return false;
}

class MqttSession {
public:
    MqttSession(acl::socket_stream* conn)
        : client_(*conn, 10)
        , conn_(conn)
        , authenticated_(false) {
    }

    ~MqttSession() {
        delete conn_;
    }

    void run() {
        printf("新客户端连接\n");

        while (true) {
            acl::mqtt_message* msg = client_.get_message();
            if (msg == NULL) {
                printf("客户端断开连接\n");
                break;
            }

            bool ok = handle_message(msg);
            delete msg;

            if (!ok) {
                break;
            }
        }

        printf("会话结束\n");
    }

private:
    bool handle_message(acl::mqtt_message* msg) {
        switch (msg->get_header().get_type()) {
        case acl::MQTT_CONNECT:
            return handle_connect((acl::mqtt_connect*) msg);
        case acl::MQTT_PUBLISH:
            return handle_publish((acl::mqtt_publish*) msg);
        case acl::MQTT_SUBSCRIBE:
            return handle_subscribe((acl::mqtt_subscribe*) msg);
        case acl::MQTT_PINGREQ:
            return handle_pingreq();
        case acl::MQTT_DISCONNECT:
            printf("收到 DISCONNECT\n");
            return false;
        default:
            printf("收到未处理的消息类型: %d\n", msg-
>get_header().get_type());
            break;
        }
        return true;
    }

    bool handle_connect(acl::mqtt_connect* conn) {
        printf("收到 CONNECT: client_id=%s\n", conn->get_cid());

        client_id_ = conn->get_cid();

        // 简单验证
        if (conn->get_username() && conn->get_passwd()) {
            if (strcmp(conn->get_username(), "admin") == 0 &&
                strcmp(conn->get_passwd(), "password") == 0) {
                authenticated_ = true;
            }
        }

        acl::mqtt_connack connack;
        if (authenticated_) {
            connack.set_connack_code(acl::MQTT_CONNACK_OK);
            printf("认证成功\n");
        } else {
            connack.set_connack_code(acl::MQTT_CONNACK_ERR_AUTH);
            printf("认证失败\n");
        }

        client_.send(connack);
        return authenticated_;
    }

    bool handle_publish(acl::mqtt_publish* pub) {
        printf("收到 PUBLISH: 主题=%s, 内容=%.*s\n",
               pub->get_topic(),
               (int)pub->get_payload_len(),
               pub->get_payload().c_str());

        // QoS 1 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);
        }
        return true;
    }

    bool handle_subscribe(acl::mqtt_subscribe* sub) {
        printf("收到 SUBSCRIBE:\n");

        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("  主题: %s, QoS: %d\n", topics[i].c_str(), qoses[i]);
            subscriptions_.insert(topics[i]);
        }

        // 发送 SUBACK
        acl::mqtt_suback suback;
        suback.set_pkt_id(sub->get_pkt_id())
              .add_topic_qos(qoses);

        client_.send(suback);
        return true;
    }

    bool handle_pingreq() {
        printf("收到 PINGREQ\n");
        acl::mqtt_pingresp resp;
        client_.send(resp);
        return true;
    }

private:
    acl::mqtt_client client_;
    acl::socket_stream* conn_;
    std::string client_id_;
    bool authenticated_;
    std::set<std::string> subscriptions_;
};

int main(void) {
    acl::log::stdout_open(true);

    // 创建服务器 socket
    acl::server_socket server;
    if (!server.open("127.0.0.1|1883")) {
        printf("监听失败\n");
        return 1;
    }

    printf("MQTT 服务器启动, 监听 1883 端口\n");

    // 接受客户端连接
    while (true) {
        acl::socket_stream* conn = server.accept();
        if (conn == NULL) {
            printf("accept 失败\n");
            break;
        }

        // 处理客户端（简化实现，实际应该使用多线程）
        MqttSession session(conn);
        session.run();
    }

    return 0;
}
```

QoS 2 消息处理

```
#include "acl_cpp/lib_acl.hpp"
#include "acl_cpp/mqtt/mqtt_client.hpp"
#include "acl_cpp/mqtt/mqtt_publish.hpp"
#include "acl_cpp/mqtt/mqtt_pubrec.hpp"
#include "acl_cpp/mqtt/mqtt_pubrel.hpp"
#include "acl_cpp/mqtt/mqtt_pubcomp.hpp"

// QoS 2 发送方
bool publish_qos2(acl::mqtt_client& client, const char* topic, const char*
payload) {
    unsigned short pkt_id = 1;

    // 1. 发送 PUBLISH (QoS 2)
    acl::mqtt_publish pub;
    pub.set_topic(topic)
       .set_pkt_id(pkt_id)
       .set_payload(strlen(payload), payload);
    pub.get_header().set_qos(acl::MQTT_QOS2);

    if (!client.send(pub)) {
        printf("发送 PUBLISH 失败\n");
        return false;
    }
    printf("发送 PUBLISH (QoS 2)\n");

    // 2. 等待 PUBREC
    acl::mqtt_message* msg = client.get_message();
    if (msg == NULL || msg->get_header().get_type() != acl::MQTT_PUBREC) {
        printf("未收到 PUBREC\n");
        if (msg) delete msg;
        return false;
    }

    acl::mqtt_pubrec* pubrec = (acl::mqtt_pubrec*) msg;
    printf("收到 PUBREC, 消息ID: %d\n", pubrec->get_pkt_id());
    delete msg;
    // 3. 发送 PUBREL
    acl::mqtt_pubrel pubrel;
    pubrel.set_pkt_id(pkt_id);

    if (!client.send(pubrel)) {
        printf("发送 PUBREL 失败\n");
        return false;
    }
    printf("发送 PUBREL\n");

    // 4. 等待 PUBCOMP
    msg = client.get_message();
    if (msg == NULL || msg->get_header().get_type() != acl::MQTT_PUBCOMP)
{
        printf("未收到 PUBCOMP\n");
        if (msg) delete msg;
        return false;
    }

    acl::mqtt_pubcomp* pubcomp = (acl::mqtt_pubcomp*) msg;
    printf("收到 PUBCOMP, 消息ID: %d\n", pubcomp->get_pkt_id());
    printf("QoS 2 消息发送完成\n");
    delete msg;

    return true;
}

// QoS 2 接收方
bool handle_qos2_publish(acl::mqtt_client& client, acl::mqtt_publish* pub)
{
    unsigned short pkt_id = pub->get_pkt_id();

    printf("收到 PUBLISH (QoS 2): 主题=%s, 内容=%.*s\n",
           pub->get_topic(),
           (int)pub->get_payload_len(),
           pub->get_payload().c_str());

    // 1. 发送 PUBREC
    acl::mqtt_pubrec pubrec;
    pubrec.set_pkt_id(pkt_id);

    if (!client.send(pubrec)) {
        printf("发送 PUBREC 失败\n");
        return false;
    }
    printf("发送 PUBREC\n");

    // 2. 等待 PUBREL
    acl::mqtt_message* msg = client.get_message();
    if (msg == NULL || msg->get_header().get_type() != acl::MQTT_PUBREL) {
        printf("未收到 PUBREL\n");
        if (msg) delete msg;
        return false;
    }
    acl::mqtt_pubrel* pubrel = (acl::mqtt_pubrel*) msg;
    printf("收到 PUBREL, 消息ID: %d\n", pubrel->get_pkt_id());
    delete msg;

    // 3. 发送 PUBCOMP
    acl::mqtt_pubcomp pubcomp;
    pubcomp.set_pkt_id(pkt_id);

    if (!client.send(pubcomp)) {
        printf("发送 PUBCOMP 失败\n");
        return false;
    }
    printf("发送 PUBCOMP\n");
    printf("QoS 2 消息接收完成\n");

    return true;
}
```

心跳机制

```
#include "acl_cpp/lib_acl.hpp"
#include "acl_cpp/mqtt/mqtt_client.hpp"
#include "acl_cpp/mqtt/mqtt_pingreq.hpp"
#include "acl_cpp/mqtt/mqtt_pingresp.hpp"
#include <time.h>

class MqttClientWithHeartbeat {
public:
    MqttClientWithHeartbeat(const char* addr, int keep_alive)
        : client_(addr, 10, 30)
        , keep_alive_(keep_alive)
        , last_ping_(0) {
    }

    bool connect(const char* client_id) {
        if (!client_.open()) {
            return false;
        }

        acl::mqtt_connect conn;
        conn.set_cid(client_id)
            .set_keep_alive(keep_alive_);

        if (!client_.send(conn)) {
            return false;
        }

        acl::mqtt_message* msg = client_.get_message();
        if (msg && msg->get_header().get_type() == acl::MQTT_CONNACK) {
            acl::mqtt_connack* connack = (acl::mqtt_connack*) msg;
            bool ok = (connack->get_connack_code() ==
acl::MQTT_CONNACK_OK);
            delete msg;

            if (ok) {
                last_ping_ = time(NULL);
            }

            return ok;
        }

        if (msg) delete msg;
        return false;
    }

    void loop() {
        while (true) {
            // 检查是否需要发送心跳
            time_t now = time(NULL);
            if (now - last_ping_ >= keep_alive_) {
                if (!send_ping()) {
                    printf("发送心跳失败\n");
                    break;
                }
                last_ping_ = now;
            }

            // 设置较短的超时，以便及时发送心跳
            // 实际应用中应该使用非阻塞或异步方式
            sleep(1);
        }
    }

private:
    bool send_ping() {
        printf("发送心跳\n");

        acl::mqtt_pingreq ping;
        if (!client_.send(ping)) {
            return false;
        }

        acl::mqtt_message* msg = client_.get_message();
        if (msg && msg->get_header().get_type() == acl::MQTT_PINGRESP) {
            printf("收到心跳响应\n");
            delete msg;
            return true;
        }

        if (msg) delete msg;
        return false;
    }

private:
    acl::mqtt_client client_;
    int keep_alive_;
    time_t last_ping_;
};

int main(void) {
    acl::log::stdout_open(true);

    MqttClientWithHeartbeat client("127.0.0.1|1883", 60);

    if (!client.connect("heartbeat_client")) {
        printf("连接失败\n");
        return 1;
    }

    printf("连接成功，开始心跳\n");
    client.loop();

    return 0;
}
```

遗嘱消息

```
#include "acl_cpp/lib_acl.hpp"
#include "acl_cpp/mqtt/mqtt_client.hpp"
#include "acl_cpp/mqtt/mqtt_connect.hpp"

int main(void) {
    acl::log::stdout_open(true);

    acl::mqtt_client client("127.0.0.1|1883", 10, 30);

    if (!client.open()) {
        printf("连接失败\n");
        return 1;
    }

    // 设置遗嘱消息
    acl::mqtt_connect conn;
    conn.set_cid("will_client")
        .set_keep_alive(60)
        .set_will_topic("status/will_client")    // 遗嘱主题
        .set_will_msg("offline")                  // 遗嘱内容
        .set_will_qos(acl::MQTT_QOS1);           // 遗嘱 QoS

    if (!client.send(conn)) {
        printf("发送 CONNECT 失败\n");
        return 1;
    }

    acl::mqtt_message* msg = client.get_message();
    if (msg && 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，已设置遗嘱消息\n");
            printf("  遗嘱主题: status/will_client\n");
            printf("  遗嘱内容: offline\n");

            // 模拟异常断开（不发送 DISCONNECT）
            // 服务器会自动发布遗嘱消息
            printf("模拟异常断开...\n");
            sleep(2);
            // 直接退出，不调用 disconnect()
        }
        delete msg;
    }

    return 0;
}
```

SSL/TLS 加密通信

```
#include "acl_cpp/lib_acl.hpp"
#include "acl_cpp/mqtt/mqtt_aclient.hpp"
#include "acl_cpp/mqtt/mqtt_connect.hpp"

class SecureMqttClient : public acl::mqtt_aclient {
public:
    SecureMqttClient(acl::aio_handle& handle, acl::sslbase_conf* ssl_conf)
        : mqtt_aclient(handle, ssl_conf) {
    }

protected:
    void destroy() override {
        delete this;
    }

    bool on_open() override {
        printf("SSL 连接建立成功\n");

        acl::mqtt_connect conn;
        conn.set_cid("ssl_client")
            .set_username("admin")
            .set_passwd("password")
            .set_keep_alive(60);

        return send(conn);
    }

    bool on_body(const acl::mqtt_message& msg) override {
        printf("收到消息类型: %d\n", msg.get_header().get_type());
        if (msg.get_header().get_type() == acl::MQTT_CONNACK) {
            const acl::mqtt_connack& connack = (const acl::mqtt_connack&)
msg;
            if (connack.get_connack_code() == acl::MQTT_CONNACK_OK) {
                printf("SSL 连接认证成功\n");
            }
        }

        return true;
    }

    void on_disconnect() override {
        printf("SSL 连接断开\n");
    }
};

int main(void) {
    acl::log::stdout_open(true);

    acl::aio_handle handle(acl::ENGINE_KERNEL);

    // 创建 SSL 配置
    acl::openssl_conf* ssl_conf = new acl::openssl_conf(true);

    // 可选：设置 CA 证书
    // ssl_conf->set_ca_path("/path/to/ca.crt");
    // ssl_conf->set_ca_file("/path/to/ca.crt");

    // 可选：设置客户端证书（双向认证）
    // ssl_conf->add_cert("/path/to/client.crt");
    // ssl_conf->set_key("/path/to/client.key");

    SecureMqttClient* client = new SecureMqttClient(handle, ssl_conf);

    // 设置 SNI（Server Name Indication）
    client->set_host("mqtt.example.com");

    // 连接服务器（使用 8883 SSL 端口）
    if (!client->open("mqtt.example.com|8883", 10, 30)) {
        printf("连接失败\n");
        client->destroy();
        delete ssl_conf;
        return 1;
    }

    printf("开始 SSL 连接...\n");

    while (true) {
        handle.check();
    }

    delete ssl_conf;
    return 0;
}
```

编译和运行

编译

```
# 假设已经编译安装了 ACL 库
g++ -o mqtt_client simple_client.cpp \
    -I/path/to/acl/lib_acl_cpp/include \
    -L/path/to/acl/lib \
    -l_acl_cpp -l_acl -l_protocol \
    -lpthread -ldl -lz

# 如果使用 SSL
g++ -o mqtt_ssl_client ssl_client.cpp \
    -I/path/to/acl/lib_acl_cpp/include \
    -L/path/to/acl/lib \
    -l_acl_cpp -l_acl -l_protocol \
    -lpthread -ldl -lz -lssl -lcrypto
```

运行

```
# 运行客户端
./mqtt_client

# 运行服务器
./mqtt_server

# 运行发布者
./mqtt_publisher

# 运行订阅者
./mqtt_subscriber
```

相关文档

- MQTT 同步客户端
- MQTT 异步客户端
- MQTT 消息类型