厦門大學

软件学院

物联网技术导论实验三

班　　级　软工三班

学　　院　信息学院

专　　业　软件工程

年　　级　2021 级

学　　号　32420212202930

姓　　名　陈澄

1　实验背景

物联网作为一种连接各种物理设备和传感器的技术，其关键在于实现设备间的高效通信和数据传输。MQTT 作为一种轻量级的、基于发布/订阅模式的消息传输协议，被广泛应用于物联网系统中。本实验旨在探究在校园网环境下，MQTT 通信架构的性能表现，并与 UDP 协议进行比较。

2　实验内容

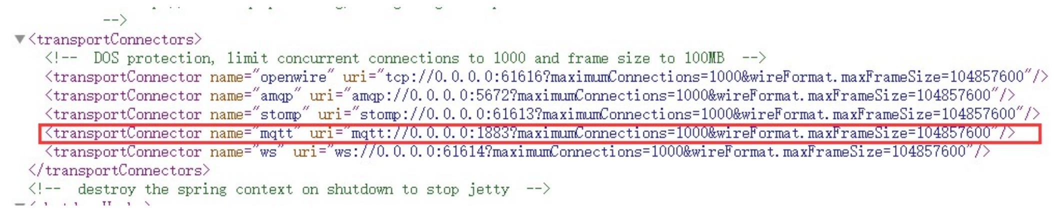
1、实现 MQTT 通信架构，搭建 MQTT 服务器，前端传感器发送数据

(1)使用 ActiveMQ 所支持的 MQTT 通信架构

Apache ActiveMQ

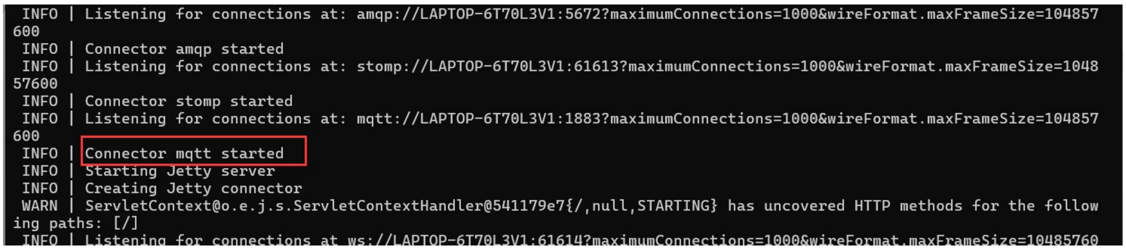
Details of “classic” ActiveMQ’s support for MQTT are available here.

(2)进行 ActiveMQ 配置文件配置

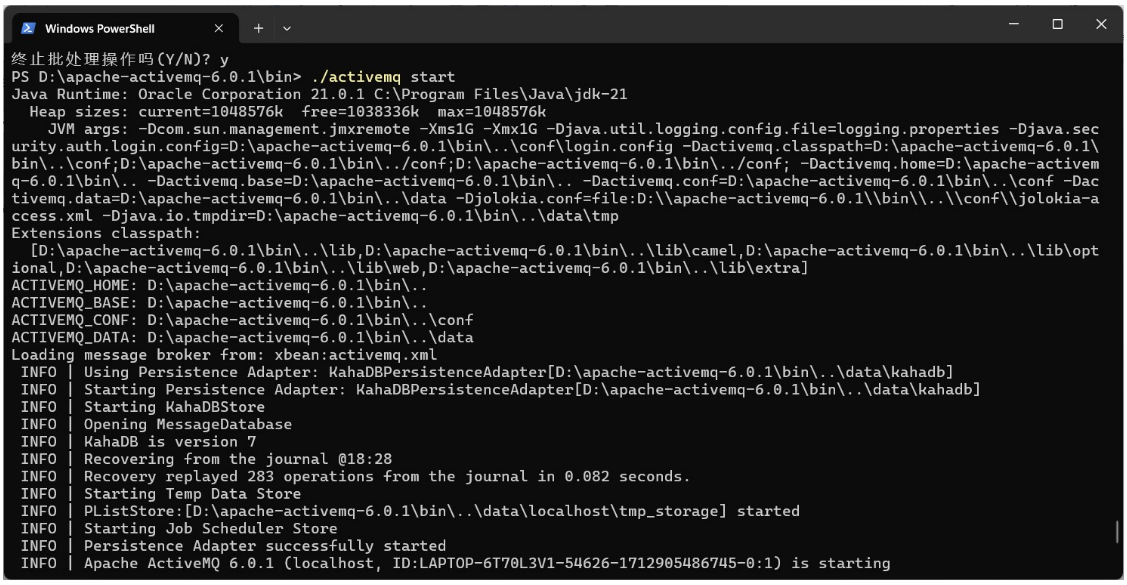


```
      -->
▼<transportConnectors>
   <!--  DOS protection, limit concurrent connections to 1000 and frame size to 100MB  -->
   <transportConnector name="openwire" uri="tcp://0.0.0.0:61616?maximumConnections=1000&wireFormat.maxFrameSize=104857600"/>
   <transportConnector name="amqp" uri="amqp://0.0.0.0:5672?maximumConnections=1000&wireFormat.maxFrameSize=104857600"/>
   <transportConnector name="stomp" uri="stomp://0.0.0.0:61613?maximumConnections=1000&wireFormat.maxFrameSize=104857600"/>
   <transportConnector name="mqtt" uri="mqtt://0.0.0.0:1883?maximumConnections=1000&wireFormat.maxFrameSize=104857600"/>
   <transportConnector name="ws" uri="ws://0.0.0.0:61614?maximumConnections=1000&wireFormat.maxFrameSize=104857600"/>
 </transportConnectors>
 <!--  destroy the spring context on shutdown to stop jetty  -->
```

(3)进入 ActiveMQ 启动 ActiveMQ 服务，确保 MQTT 连接器已经启动并监听指定的端口



```
 INFO | Listening for connections at: amqp://LAPTOP-6T70L3V1:5672?maximumConnections=1000&wireFormat.maxFrameSize=104857
600
 INFO | Connector amqp started
 INFO | Listening for connections at: stomp://LAPTOP-6T70L3V1:61613?maximumConnections=1000&wireFormat.maxFrameSize=1048
57600
 INFO | Connector stomp started
 INFO | Listening for connections at: mqtt://LAPTOP-6T70L3V1:1883?maximumConnections=1000&wireFormat.maxFrameSize=104857
600
 INFO | Connector mqtt started
 INFO | Starting Jetty server
 INFO | Creating Jetty connector
 WARN | ServletContext@o.e.j.s.ServletContextHandler@541179e7{/,null,STARTING} has uncovered HTTP methods for the follow
ing paths: [/]
 INFO | Listening for connections at ws://LAPTOP-6T70L3V1:61614?maximumConnections=1000&wireFormat.maxFrameSize=10485760
```

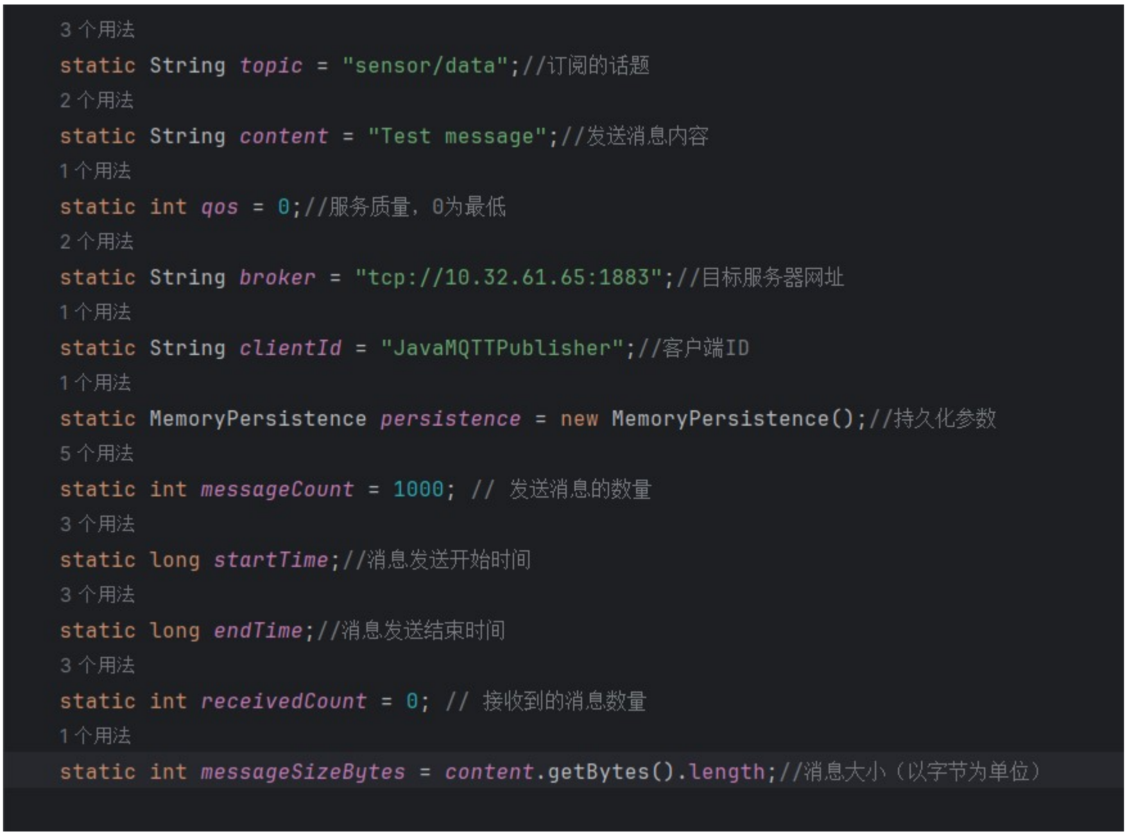


```
终止批处理操作吗(Y/N)? y
PS D:\apache-activemq-6.0.1\bin> ./activemq start
Java Runtime: Oracle Corporation 21.0.1 C:\Program Files\Java\jdk-21
  Heap sizes: current=1048576k  free=1038336k  max=1048576k
    JVM args: -Dcom.sun.management.jmxremote -Xms1G -Xmx1G -Djava.util.logging.config.file=logging.properties -Djava.sec
urity.auth.login.config=D:\apache-activemq-6.0.1\bin\..\conf\login.config -Dactivemq.classpath=D:\apache-activemq-6.0.1\
bin\..\conf;D:\apache-activemq-6.0.1\bin\../conf;D:\apache-activemq-6.0.1\bin\../conf; -Dactivemq.home=D:\apache-activem
q-6.0.1\bin\.. -Dactivemq.base=D:\apache-activemq-6.0.1\bin\.. -Dactivemq.conf=D:\apache-activemq-6.0.1\bin\..\conf -Dac
tivemq.data=D:\apache-activemq-6.0.1\bin\..\data -Djolokia.conf=file:D:\\apache-activemq-6.0.1\\bin\\..\\conf\\jolokia-a
ccess.xml -Djava.io.tmpdir=D:\apache-activemq-6.0.1\bin\..\data\tmp
Extensions classpath:
  [D:\apache-activemq-6.0.1\bin\..\lib,D:\apache-activemq-6.0.1\bin\..\lib\camel,D:\apache-activemq-6.0.1\bin\..\lib\opt
ional,D:\apache-activemq-6.0.1\bin\..\lib\web,D:\apache-activemq-6.0.1\bin\..\lib\extra]
ACTIVEMQ_HOME: D:\apache-activemq-6.0.1\bin\..
ACTIVEMQ_BASE: D:\apache-activemq-6.0.1\bin\..
ACTIVEMQ_CONF: D:\apache-activemq-6.0.1\bin\..\conf
ACTIVEMQ_DATA: D:\apache-activemq-6.0.1\bin\..\data
Loading message broker from: xbean:activemq.xml
 INFO | Using Persistence Adapter: KahaDBPersistenceAdapter[D:\apache-activemq-6.0.1\bin\..\data\kahadb]
 INFO | Starting Persistence Adapter: KahaDBPersistenceAdapter[D:\apache-activemq-6.0.1\bin\..\data\kahadb]
 INFO | Starting KahaDBStore
 INFO | Opening MessageDatabase
 INFO | KahaDB is version 7
 INFO | Recovering from the journal @18:28
 INFO | Recovery replayed 283 operations from the journal in 0.082 seconds.
 INFO | Starting Temp Data Store
 INFO | PListStore:[D:\apache-activemq-6.0.1\bin\..\data\localhost\tmp_storage] started
 INFO | Starting Job Scheduler Store
 INFO | Persistence Adapter successfully started
 INFO | Apache ActiveMQ 6.0.1 (localhost, ID:LAPTOP-6T70L3V1-54626-1712905486745-0:1) is starting
```

至此，服务器已经成功搭建。

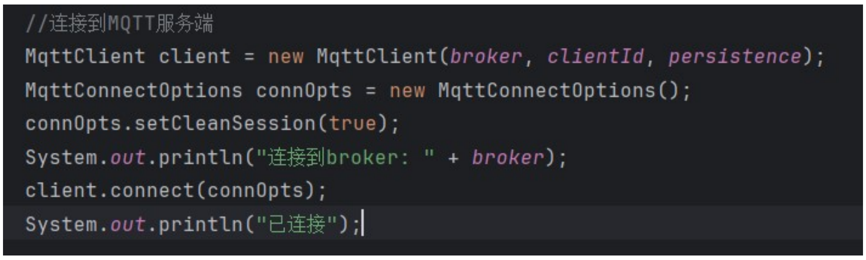
(4)编写前段程序，模拟前段发送传感器数据

基础参数如下：



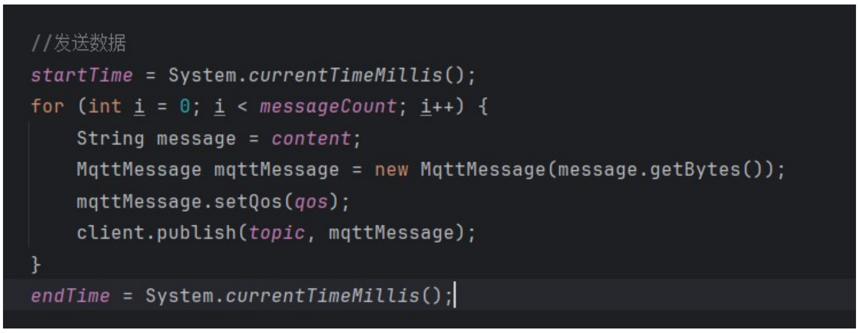
```
3 个用法
static String topic = "sensor/data";//订阅的话题
2 个用法
static String content = "Test message";//发送消息内容
1 个用法
static int qos = 0;//服务质量，0为最低
2 个用法
static String broker = "tcp://10.32.61.65:1883";//目标服务器网址
1 个用法
static String clientId = "JavaMQTTPublisher";//客户端ID
1 个用法
static MemoryPersistence persistence = new MemoryPersistence();//持久化参数
5 个用法
static int messageCount = 1000; // 发送消息的数量
3 个用法
static long startTime;//消息发送开始时间
3 个用法
static long endTime;//消息发送结束时间
3 个用法
static int receivedCount = 0; // 接收到的消息数量
1 个用法
static int messageSizeBytes = content.getBytes().length;//消息大小（以字节为单位）
```

调用 MqttClient 中的 connect 方法连接到服务端



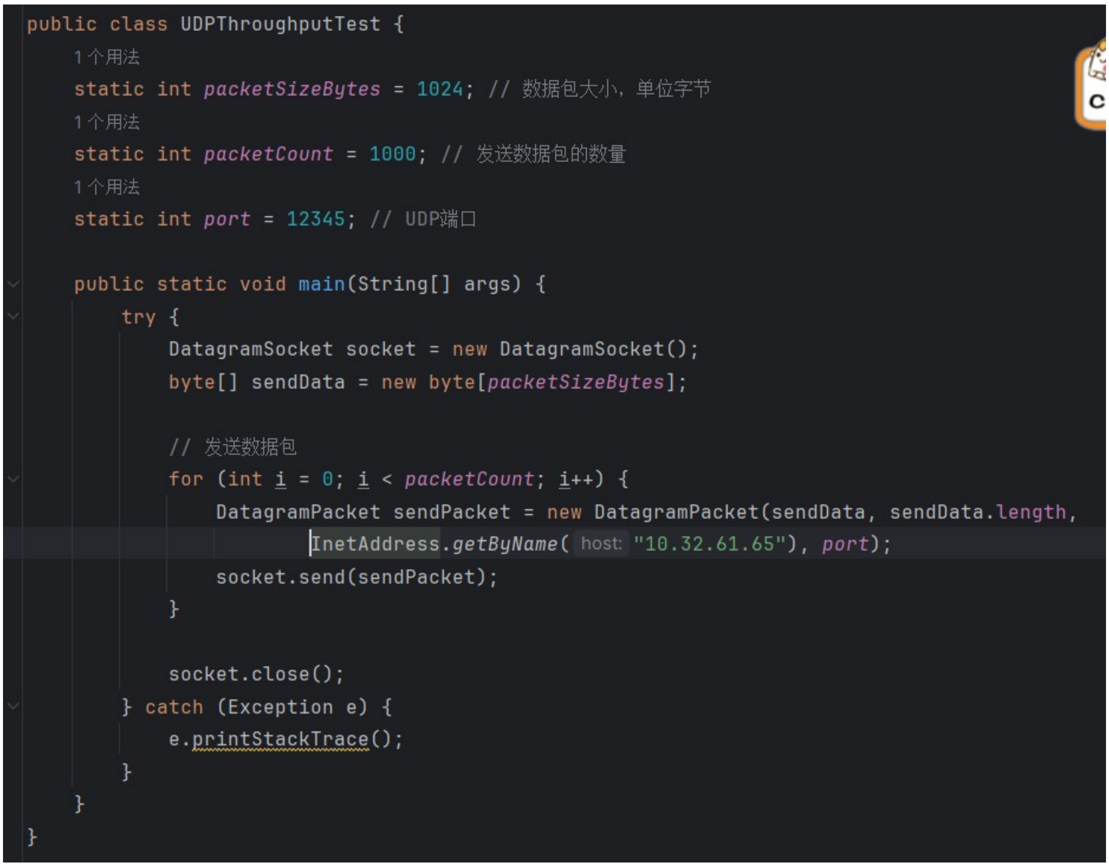
```java
//连接到MQTT服务端
MqttClient client = new MqttClient(broker, clientId, persistence);
MqttConnectOptions connOpts = new MqttConnectOptions();
connOpts.setCleanSession(true);
System.out.println("连接到broker: " + broker);
client.connect(connOpts);
System.out.println("已连接");
```

数据发送，并记录开始时间和结束时间



```java
//发送数据
startTime = System.currentTimeMillis();
for (int i = 0; i < messageCount; i++) {
    String message = content;
    MqttMessage mqttMessage = new MqttMessage(message.getBytes());
    mqttMessage.setQos(qos);
    client.publish(topic, mqttMessage);
}
endTime = System.currentTimeMillis();
```

UTP 报文的发送

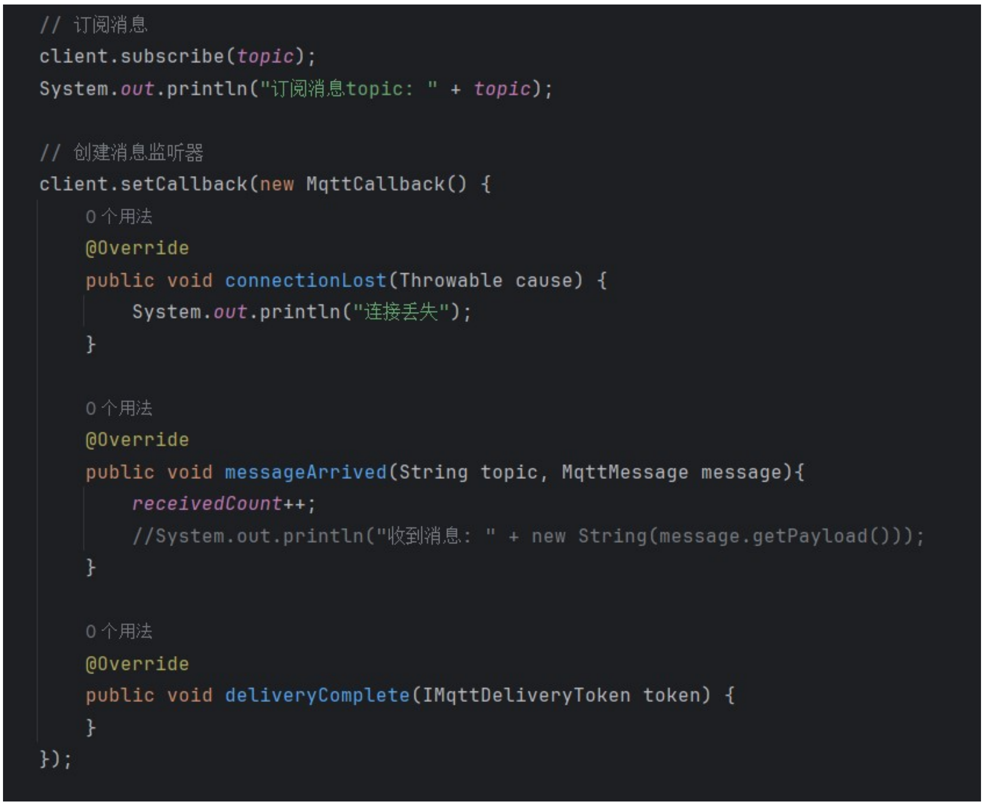


```java
public class UDPThroughputTest {
    1个用法
    static int packetSizeBytes = 1024; // 数据包大小，单位字节
    1个用法
    static int packetCount = 1000; // 发送数据包的数量
    1个用法
    static int port = 12345; // UDP端口

    public static void main(String[] args) {
        try {
            DatagramSocket socket = new DatagramSocket();
            byte[] sendData = new byte[packetSizeBytes];

            // 发送数据包
            for (int i = 0; i < packetCount; i++) {
                DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
                        InetAddress.getByName( host: "10.32.61.65"), port);
                socket.send(sendPacket);
            }

            socket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

2、测试MQTT和UDP收发的吞吐量、丢包率（校园网环境下）

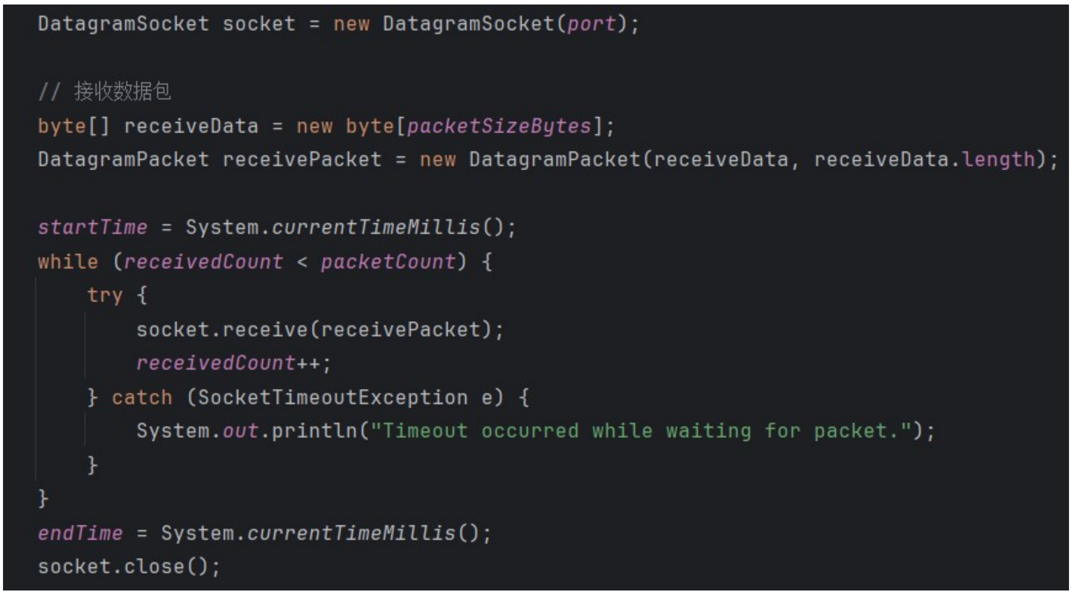
(1)接收消息

MQTT：



```
// 订阅消息
client.subscribe(topic);
System.out.println("订阅消息topic: " + topic);

// 创建消息监听器
client.setCallback(new MqttCallback() {
    0 个用法
    @Override
    public void connectionLost(Throwable cause) {
        System.out.println("连接丢失");
    }

    0 个用法
    @Override
    public void messageArrived(String topic, MqttMessage message){
        receivedCount++;
        //System.out.println("收到消息: " + new String(message.getPayload()));
    }

    0 个用法
    @Override
    public void deliveryComplete(IMqttDeliveryToken token) {
    }
});
```

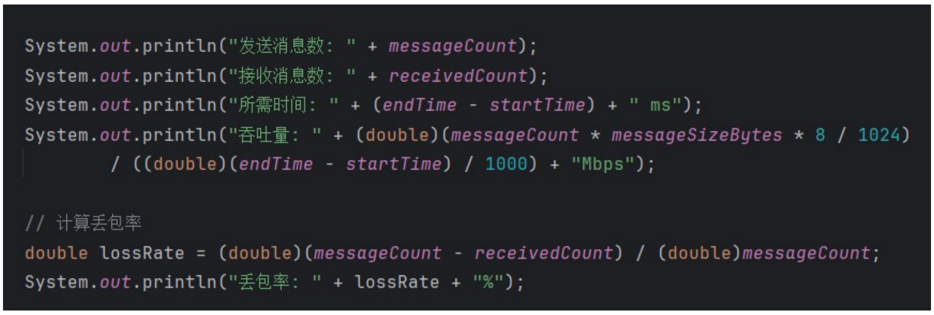
UTP报文：



```
DatagramSocket socket = new DatagramSocket(port);

// 接收数据包
byte[] receiveData = new byte[packetSizeBytes];
DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

startTime = System.currentTimeMillis();
while (receivedCount < packetCount) {
    try {
        socket.receive(receivePacket);
        receivedCount++;
    } catch (SocketTimeoutException e) {
        System.out.println("Timeout occurred while waiting for packet.");
    }
}
endTime = System.currentTimeMillis();
socket.close();
```

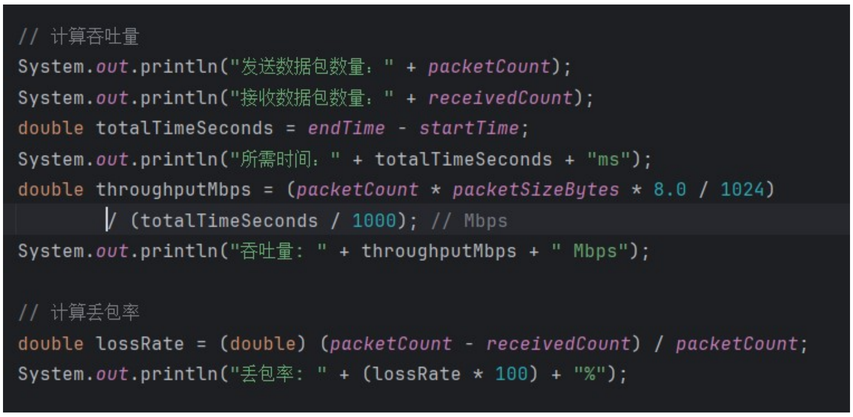
(2)计算吞吐量和丢包率

MQTT：



```
System.out.println("发送消息数: " + messageCount);
System.out.println("接收消息数: " + receivedCount);
System.out.println("所需时间: " + (endTime - startTime) + " ms");
System.out.println("吞吐量: " + (double)(messageCount * messageSizeBytes * 8 / 1024)
        / ((double)(endTime - startTime) / 1000) + "Mbps");

// 计算丢包率
double lossRate = (double)(messageCount - receivedCount) / (double)messageCount;
System.out.println("丢包率: " + lossRate + "%");
```

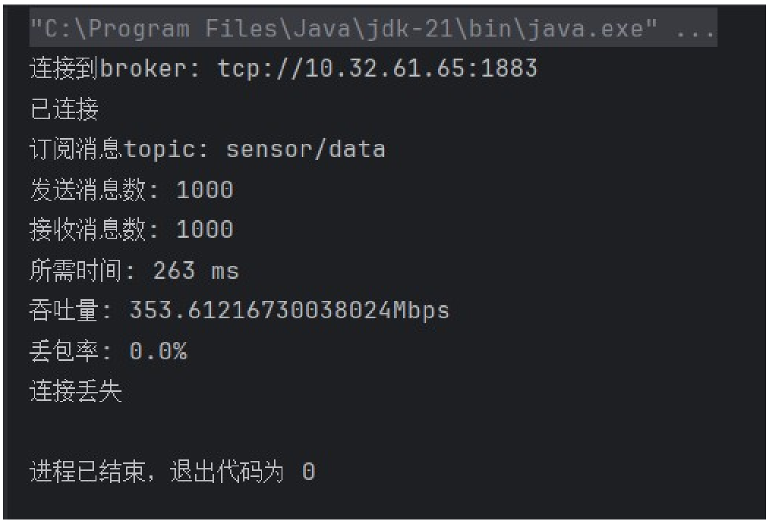
UTP：



```
// 计算吞吐量
System.out.println("发送数据包数量：" + packetCount);
System.out.println("接收数据包数量：" + receivedCount);
double totalTimeSeconds = endTime - startTime;
System.out.println("所需时间：" + totalTimeSeconds + "ms");
double throughputMbps = (packetCount * packetSizeBytes * 8.0 / 1024)
        / (totalTimeSeconds / 1000); // Mbps
System.out.println("吞吐量: " + throughputMbps + " Mbps");

// 计算丢包率
double lossRate = (double) (packetCount - receivedCount) / packetCount;
System.out.println("丢包率: " + (lossRate * 100) + "%");
```

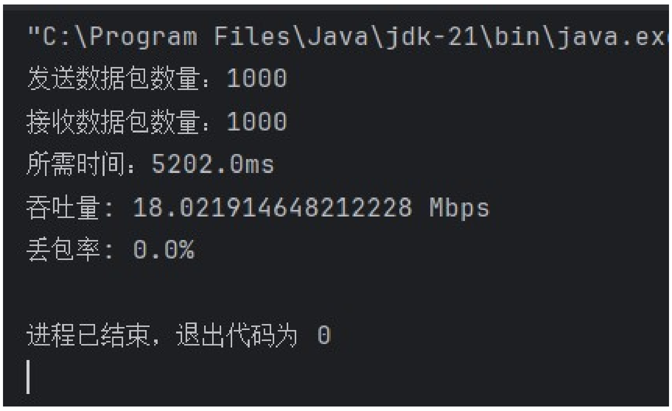
3 实验结果

1. MQTT 吞吐量以及丢包率：



```
"C:\Program Files\Java\jdk-21\bin\java.exe" ...
连接到broker: tcp://10.32.61.65:1883
已连接
订阅消息topic: sensor/data
发送消息数: 1000
接收消息数: 1000
所需时间: 263 ms
吞吐量: 353.61216730038024Mbps
丢包率: 0.0%
连接丢失

进程已结束，退出代码为 0
```

2. UTP 吞吐量以及丢包率:



```
"C:\Program Files\Java\jdk-21\bin\java.ex
发送数据包数量：1000
接收数据包数量：1000
所需时间：5202.0ms
吞吐量: 18.021914648212228 Mbps
丢包率: 0.0%

进程已结束，退出代码为 0
```

4 我的体会

通过本次实验，我对物联网中MQTT通信架构的性能表现有了更深入的了解，并且对其与UDP协议在校园网环境下的比较有了一定的认识。我深刻体会到MQTT协议的轻量级和高效性。相比于传统的HTTP协议，MQTT协议具有更小的通信开销，更适合在资源受限的物联网设备上运行。这使得在校园网环境下，MQTT通信能够更快速地实现数据传输，提高了系统的响应速度。