

# IoT 개론 및 실습

2018 1학기

강승우

## 3. 사물 인터넷 통신

### 3-2. 오픈 소스 서버/클라이언트를 이용한 MQTT 통신

#### 학습 목표

- 오픈 소스 MQTT 서버와 클라이언트를 사용하고 그 방식을 이해할 수 있다
- 이를 바탕으로 MQTT 통신 프로그램을 작성할 수 있다

## 2) Paho MQTT 클라이언트

- Paho 프로젝트
  - 이클립스 재단의 오픈 소스 MQTT와 MQTT-SN 프로토콜 클라이언트를 개발하고 제공
  - C/C++, Java, Javascript, Python, Android 등을 포함하여 다양한 언어로 구현된 클라이언트 이용 가능
  - <http://www.eclipse.org/paho/>

# Paho Python Client

# Paho Python Client

- Paho Python Client
  - Python 2.7.x or 3.x 상에서 MQTT v3.1 and v3.1.1을 지원하는 클라이언트
- 다운로드 및 설치
  - <https://github.com/eclipse/paho.mqtt.python>
  - pip tool을 이용한 설치  
pip install paho-mqtt

Windows에서는 Python 설치 폴더 아래의 Scripts 폴더에 pip 실행 파일이 있음

Windows에서 Python 3.x 버전을 사용하는데 encoding 관련 에러가 발생하여 설치를 할 수 없을 때는 Python 2.7.x 버전을 이용하면 에러 없이 설치됨

# Usage and API

- Client 클래스
  - Client 객체를 생성하여 브로커에 연결하고, subscribe 혹은 publish를 수행할 수 있음
  - Constructor / reinitialize
  - Connect / reconnect / disconnect
  - Network loop
  - Publish
  - Subscribe / Unsubscribe
  - Callbacks

# Constructor

## • Client() 생성자

- Client(client\_id="", clean\_session=True, userdata=None, protocol=MQTTv31) 3.1
- client\_id
  - 브로커에 연결할 때 사용되는 유일한 클라이언트 id
  - 길이가 0이거나 None이면 임의로 생성됨. 이 경우 clean\_session 값은 True여야 함
- clean\_session clean\_session True
  - 클라이언트 타입을 결정하는 boolean 변수
  - True이면, 브로커는 이 클라이언트가 연결 종료될 경우 모든 관련 정보를 삭제함
  - False이면, 이 클라이언트는 durable client가 되고 구독 관련된 정보와 큐에 쌓인 메시지는 클라이언트가 연결 종료되더라도 계속 유지됨
- userdata
  - 콜백 함수의 userdata 매개변수로 전달되는 사용자 정의 데이터
  - user\_data\_set() 함수로 나중에 업데이트 될 수 있음
- protocol
  - 이 클라이언트에서 사용할 MQTT 프로토콜 버전
  - MQTTv31 or MQTTv311

# Constructor

```
import paho.mqtt.client as mqtt  
mqttc = mqtt.Client()
```

가



# Reinitialize

- `reinitialise()` 함수
  - 클라이언트 객체를 바로 생성한 것처럼 다시 초기 상태로 만든다
  - `Client()` 생성자와 같은 매개변수를 받음
  - `reinitialise(client_id="", clean_session=True, userdata=None)`

```
import paho.mqtt.client as mqtt

mqttc = mqtt.Client()
...
mqttc.reinitialise()
```

# Connect

connect

- connect() 함수
  - 클라이언트를 브로커에 연결한다
  - connect(host, port=1883, keepalive=60, bind\_address="")
- host
  - 브로커의 호스트 네임 혹은 IP 주소
- port
  - 네트워크 포트 번호. Defaults to 1883.
- keepalive
  - 브로커와 통신 사이에 허용되는 최대 주기
  - 이 시간 동안 메시지 교환이 없으면 브로커로 ping 메시지를 전송하게 됨
- bind\_address
  - 이 클라이언트를 연결할 로컬 네트워크 인터페이스의 IP 주소 (복수개의 인터페이스가 존재하는 경우)

# Connect

- Connect 요청에 대한 응답으로 클라이언트가 CONNACK 메시지를 브로커로부터 받으면 on\_connect() 콜백이 생성됨
- Connect 함수 예제

```
import paho.mqtt.client as mqtt  
  
mqttc = mqtt.Client()  
mqttc.connect("iot.eclipse.org")
```


localhost,

“localhost”  
“127.0.0.1”

가 .

# Reconnect, disconnect

- `reconnect()` 함수

- 이전 `connect()`에서 제공된 정보를 이용하여 브로커에 다시 연결함
  - 이 함수를 부르기 전에 `connect()` 함수를 호출한 적이 있어야 함
- 

- `disconnect()` 함수

- 브로커와의 연결을 클린 종료함
- 클라이언트가 `disconnect` 메시지를 전송하면, `on_disconnect()` 콜백이 생성됨

# Network loop

- 클라이언트가 동작하는 과정에서 네트워크로 데이터를 정상적으로 주고 받기 위해서 호출해야 하는 함수
  - 호출이 되지 않는 경우, incoming 네트워크 데이터가 처리가 되지 않거나, outgoing 네트워크 데이터가 제시간에 전송되지 않을 수 있음
- loop()
  - 네트워크 이벤트를 처리하기 위해서 정기적으로 호출해야 함
- loop\_start() / loop\_stop() pub, sub loop\_start .
  - loop\_start()를 connect() 함수 호출 이전 혹은 이후에 한번 호출하면 loop()를 자동으로 호출하는 백그라운드 스레드를 실행함
    - 메인 스레드는 다른 작업을 할 수 있음
  - loop\_stop()을 호출하면 백그라운드 스레드 종료
- loop\_forever()
  - Network loop를 위한 블로킹 함수로 disconnect() 호출 시까지 반환되지 않음

# Network loop

```
import paho.mqtt.client as mqtt

mqttc = mqtt.Client()
mqttc.connect("iot.eclipse.org")
mqttc.loop_start()

while True:
    temperature = 30
    mqttc.publish("paho/temperature", temperature)
```

# Publish

- publish() 함수
  - 브로커로 메시지를 전송함
    - 브로커를 통해 매칭되는 토픽을 구독하는 클라이언트에게 메시지가 전달되게 됨
  - publish(topic, payload=None, qos=0, retain=False)
  - topic
    - 메시지가 publish 되는 토픽
  - payload
    - 전송할 메시지
    - int, float 값을 넣는 경우 해당 숫자를 표현하는 string 값으로 변환됨
  - qos
    - 메시지 전송에 사용할 QoS 값 (0, 1, 2)
  - retain
    - True이면, 메시지는 해당 토픽에 대해서 “last known good”/retained 메시지로 설정됨

가

=

가

All rights reserved.

강승우

retain true

=가

# Publish

- 메시지가 브로커로 전송이 되면, `on_publish()` 콜백이 생성됨



# Subscribe / Unsubscribe

- subscribe() 함수
  - 하나 혹은 그 이상의 토픽에 구독 요청함
  - subscribe(topic, qos=0)
  - topic
    - 구독 요청할 토픽 (string 값으로 표현)
  - qos
    - 구독 요청에 대한 QoS 값 (기본 0)

- 구독 요청에 대해서 브로커가 SUBACK를 보내면, on\_subscribe() 콜백이 생성됨

- unsubscribe() 함수
  - 구독 요청을 해제함
  - unsubscribe(topic)
  - topic
    - Unsubscribe 할 토픽
    - 하나의 string 값, 혹은 string 값의 리스트로 표현

가 unsubscribe  
topic unsub

- 구독 해제에 대해서 브로커가 UNSUBACK를 보내면, on\_unsubscribe() 콜백이 생성됨

# Subscribe / Unsubscribe

- 하나의 토픽을 구독 요청하는 경우

- `subscribe("my/topic", 2)`
  - String, integer 값
- `subscribe(("my/topic", 1))`
  - String, integer의 튜플 이용

- 한번에 여러 개의 토픽을 구독 요청하는 경우

- `subscribe([("my/topic", 0), ("another/topic", 2)])`
  - String, integer 튜플의 리스트

list

qos

..

# Callbacks

- `on_connect()` 함수
  - `callback` 가 , `connect` 가 .
  - 브로커로부터 `connect` 요청에 대한 Ack를 받으면 호출되는 콜백 함수
  - `on_connect(client, userdata, flags, rc)`
    - `connect` `on_connect`가 .
    - `client`
      - 클라이언트 객체
    - `userdata`
      - `Client()` 혹은 `user_data_set()` 호출로 설정되는 사용자 데이터
    - `flags`
      - 브로커가 전송하는 응답 플래그
    - `rc`
      - 연결 결과

# Callbacks

- on\_connect() 함수 예제

```
import paho.mqtt.client as mqtt
def on_connect(client, userdata, rc):
    print("Connection returned result: "+connack_string(rc))

mqttc = mqtt.Client()
mqttc.on_connect = on_connect
mqttc.connect("iot.eclipse.org")
```

# Callbacks

- on\_disconnect() 함수
  - on\_disconnect(client, userdata, rc)

```
def on_disconnect(client, userdata, rc):  
    if rc != 0:  
        print("Unexpected disconnection.")
```

```
mqttc = mqtt.Client()  
mqttc.on_connect = on_connect  
mqttc.on_disconnect = on_disconnect  
...
```

# Callbacks

- `on_publish()`
  - `publish()` 호출로 전송된 메시지가 브로커에게 전송이 되면 호출되는 콜백 함수
- `on_subscribe()`
  - 구독 요청에 대해 브로커가 응답을 하면 호출되는 콜백 함수
- `on_unsubscribe()`
  - 구독 해제에 대해 브로커가 응답을 하면 호출되는 콜백 함수

# Callbacks

- `on_message()`
  - 클라이언트가 구독하는 토픽에 대한 메시지를 받았을 때 호출되는 콜백 함수

```
def on_message(client, userdata, message):  
    print("Received message : " + str(message.payload) + "  
on topic: " + message.topic + " with QoS " +  
str(message.qos))  
  
mqttc.on_message = on_message  
...
```

# Paho Python Client documentation

- 기타 자세한 내용은 아래 다큐먼트를 참고
  - <http://www.eclipse.org/paho/clients/python/docs/>



# Publish client example code

```
import paho.mqtt.client as mqtt

mqttc = mqtt.Client()

# YOU NEED TO CHANGE THE IP ADDRESS OR HOST NAME
mqttc.connect("192.168.0.23")
#mqttc.connect("localhost")

mqttc.loop_start()

mqttc.publish("hello/world", "Hello")

mqttc.loop_stop()
```

# Publish client example code

```
import paho.mqtt.client as mqtt
import random
import time

def getMsg():
    msg = str(random.randrange(20, 36))
    return msg

mqttc = mqtt.Client()
# YOU NEED TO CHANGE THE IP ADDRESS OR HOST NAME
mqttc.connect("192.168.0.23")
#mqttc.connect("localhost")
mqttc.loop_start()
```

```
try:
    while True:
        t = getMsg()
        (result, m_id) = mqttc.publish("room309/temperature", t)
        time.sleep(1)

except KeyboardInterrupt:
    print("Finished!")
    mqttc.loop_stop()
    mqttc.disconnect()
```

# Subscribe client example code

```
import paho.mqtt.client as mqtt

def on_connect(client, userdata, rc):
    print("connected with result code " + str(rc))
    client.subscribe("room309/temperature")
    client.subscribe("room309/humidity")

def on_message(client, userdata, msg):
    print("Topic: " + msg.topic + " Message: " +
          str(msg.payload))

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
```

```
# YOU NEED TO CHANGE THE IP ADDRESS OR HOST NAME
client.connect("192.168.0.23", 1883, 60)
#client.connect("localhost")

try:
    client.loop_forever()
except KeyboardInterrupt:
    print("Finished!")
    client.unsubscribe(["room309/temperature",
                       "room309/humidity"])
    client.disconnect()
```


# Paho Java Client


# Paho Java 클라이언트

- JVM이나 안드로이드와 같은 Java 호환 플랫폼에서 동작하는 애플리케이션을 개발하기 위해 Java로 쓰여진 MQTT 클라이언트 라이브러리
  - Paho Java library 다운로드
    - JAR 파일 다운로드 jar
      - <https://repo.eclipse.org/content/repositories/paho-releases/org/eclipse/paho/org.eclipse.paho.client.mqttv3/>
- 두 가지의 응용 프로그램 프로그래밍 인터페이스를 제공
  - MqttAsyncClient: 실행 결과가 등록된 콜백(callbacks)함수를 통해 전달되는 완전한 비동기 API 제공
  - MqttClient: 동기 호출(synchronous call)을 통해 기능이 수행되는 API

## Eclipse 환경에서 Paho library 사용 프로젝트 만들기

- Eclipse에서 Paho 프로그래밍을 위한 새로운 Java Project를 생성
  - Eclipse에서 File → New → Java Project를 선택 (만약 New 선택 후 "Java Project"가 보이지 않으면 Others를 선택한 후 Java Project를 선택한다.)
  - Project name을 설정한 후 <Finish button>을 클릭
    - "PahoPublishTest"라는 이름의 프로젝트를 생성

 **New Java Project** — □ ×

**Create a Java Project**  
Create a Java project in the workspace or in an external location. 

**Project name:**

☒ **Use default location**  
**Location:**

**JRE**

☒ **Use an execution environment JRE:**  ▼

☐ **Use a project specific JRE:**  ▼

☐ **Use default JRE (currently 'jre1.8.0\_66')** [Configure JREs...](#)


**Project layout**


☐ **Use project folder as root for sources and class files**

☒ **Create separate folders for sources and class files** [Configure default...](#)

**Working sets**

☐ **Add project to working sets**  
**Working sets:**  ▼

 The default compiler compliance level for the current workspace is 1.6. The new project will use a project specific compiler compliance level of 1.8.



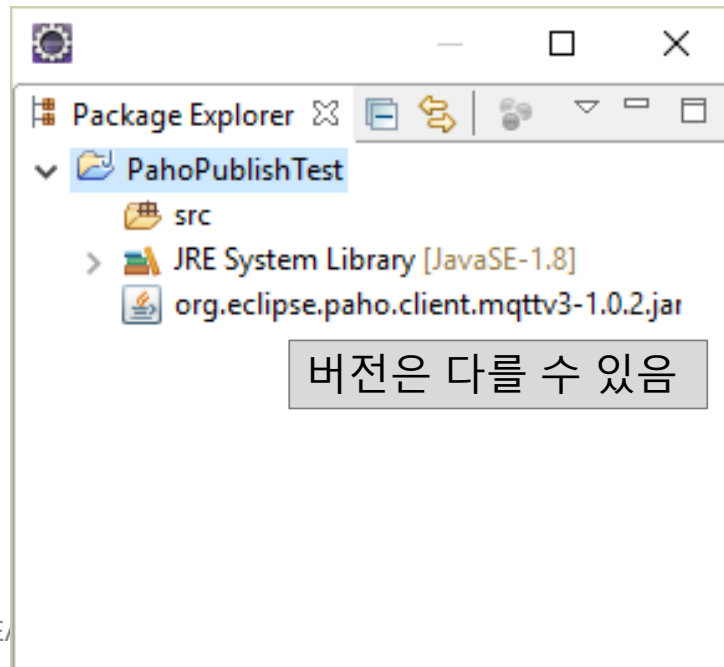
IoT 개론 및 실습

All rights reserved.

강승우

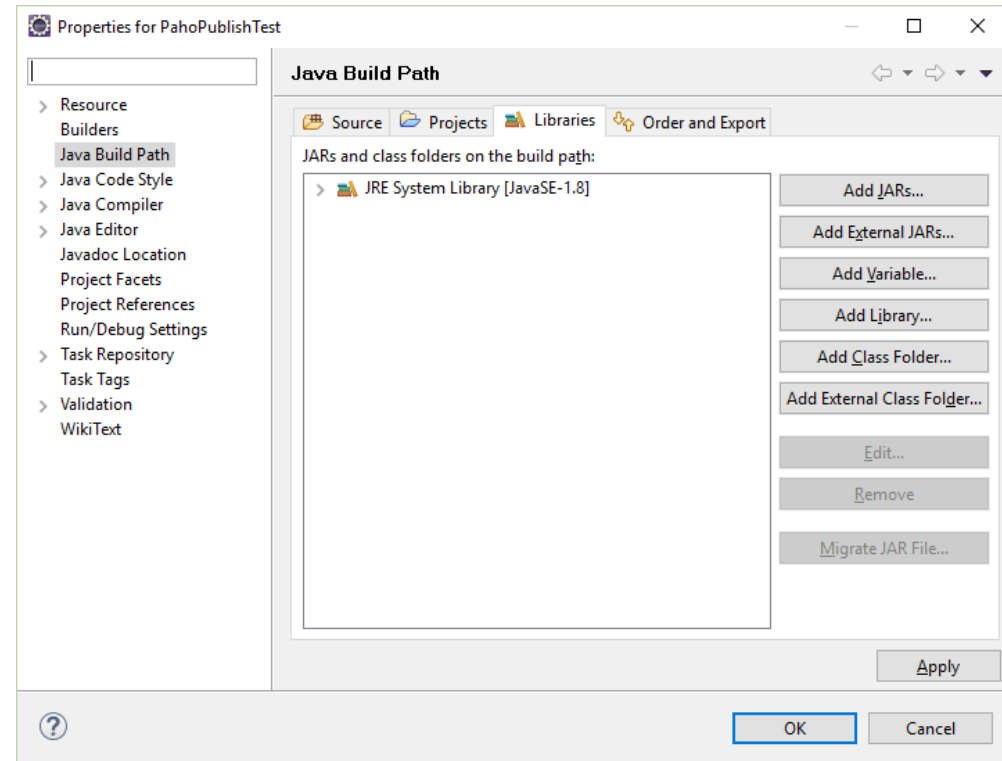
- Paho library jar 파일 복사

- 다운로드 받은 Paho Java Client jar 파일을 생성된 프로젝트에 복사
- 윈도우 탐색기에서 org.eclipse.paho.client.mqttv3-1.0.2.jar 파일을 복사, Eclipse의 Package explorer에서 해당 프로젝트를 선택한 후 붙여 넣기



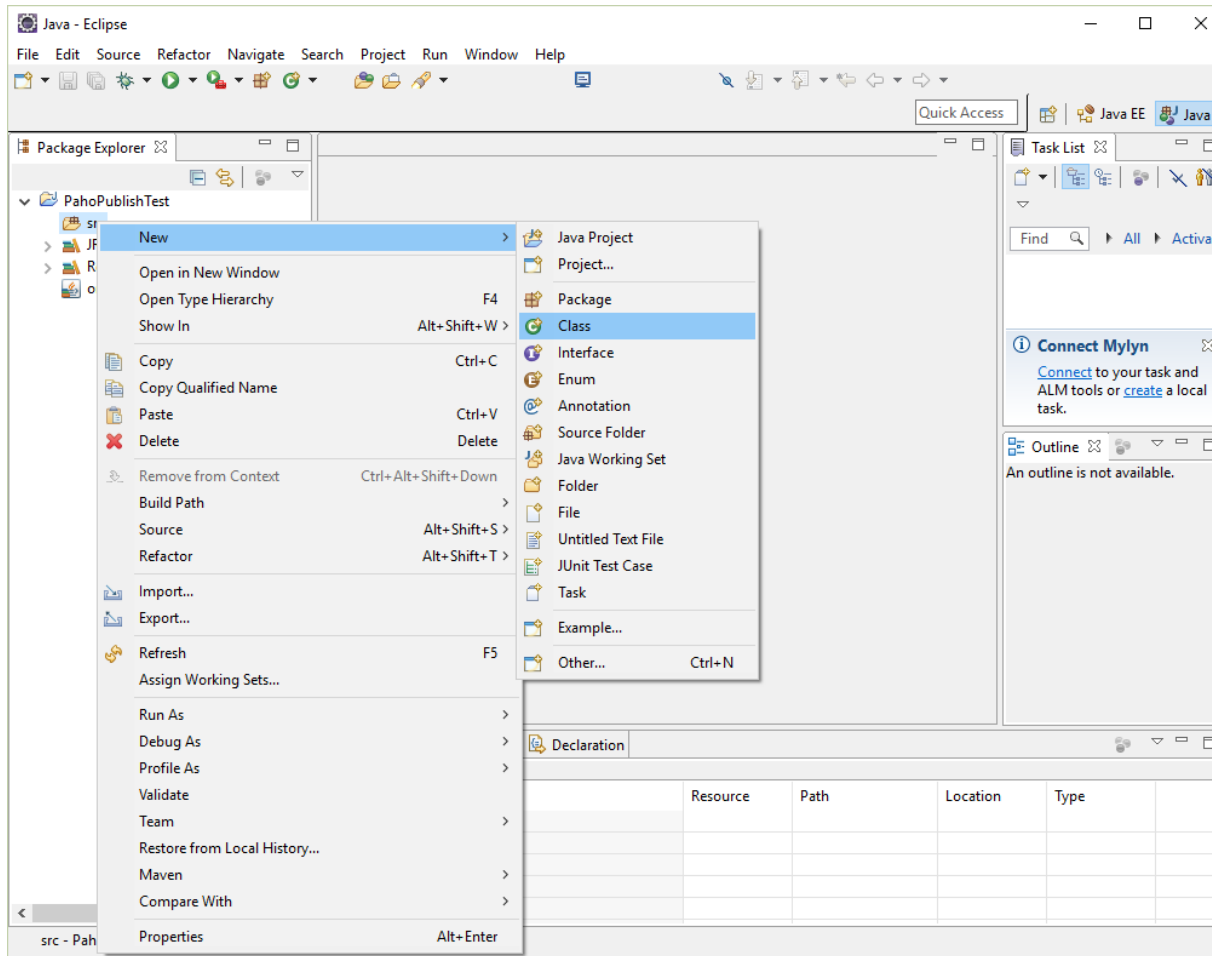


- 해당 jar 파일을 프로젝트에 추가
  - Package Explorer에서 해당 프로젝트를 선택한 후 마우스 오른쪽 버튼을 눌러 <Properties>을 클릭
  - Java Build Path를 선택한 후 Libraries 탭을 선택
  - <Add JARs> 버튼을 클릭
  - 해당 프로젝트를 선택하여 관련 파일을 확인한 후 Paho jar 파일을 선택하고 <OK> 버튼을 클릭



# Paho 클라이언트 라이브러리를 이용한 발행(publish) 테스트

- 자바 클래스 소스 파일 생성



**New Java Class**

**Java Class**

⚠ The use of the default package is discouraged.

Source folder: PahoPublishTest/src Browse...

Package: (default) Browse...

☐ Enclosing type: Browse...

Name: PahoPublishTestMain

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add...  
Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

? Finish Cancel

- Name필드에 PahoPublishTestMain을 입력한 후 <Finish> 버튼을 클릭

```

import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;

public class PahoPublishTestMain {

    public static void main(String[] args) {

        String topic    = "temp/random";
        String content   = "23.0";
        int qos         = 2;
        String broker    = "tcp://test.mosquitto.org:1883";
        String clientId  = "JavaSample";
        MemoryPersistence persistence = new MemoryPersistence();

```

```

try {
    MqttClient sampleClient = new MqttClient(broker, clientId,
persistence);

    MqttConnectOptions connOpts = new MqttConnectOptions();
connOpts.setCleanSession(true);   가   가

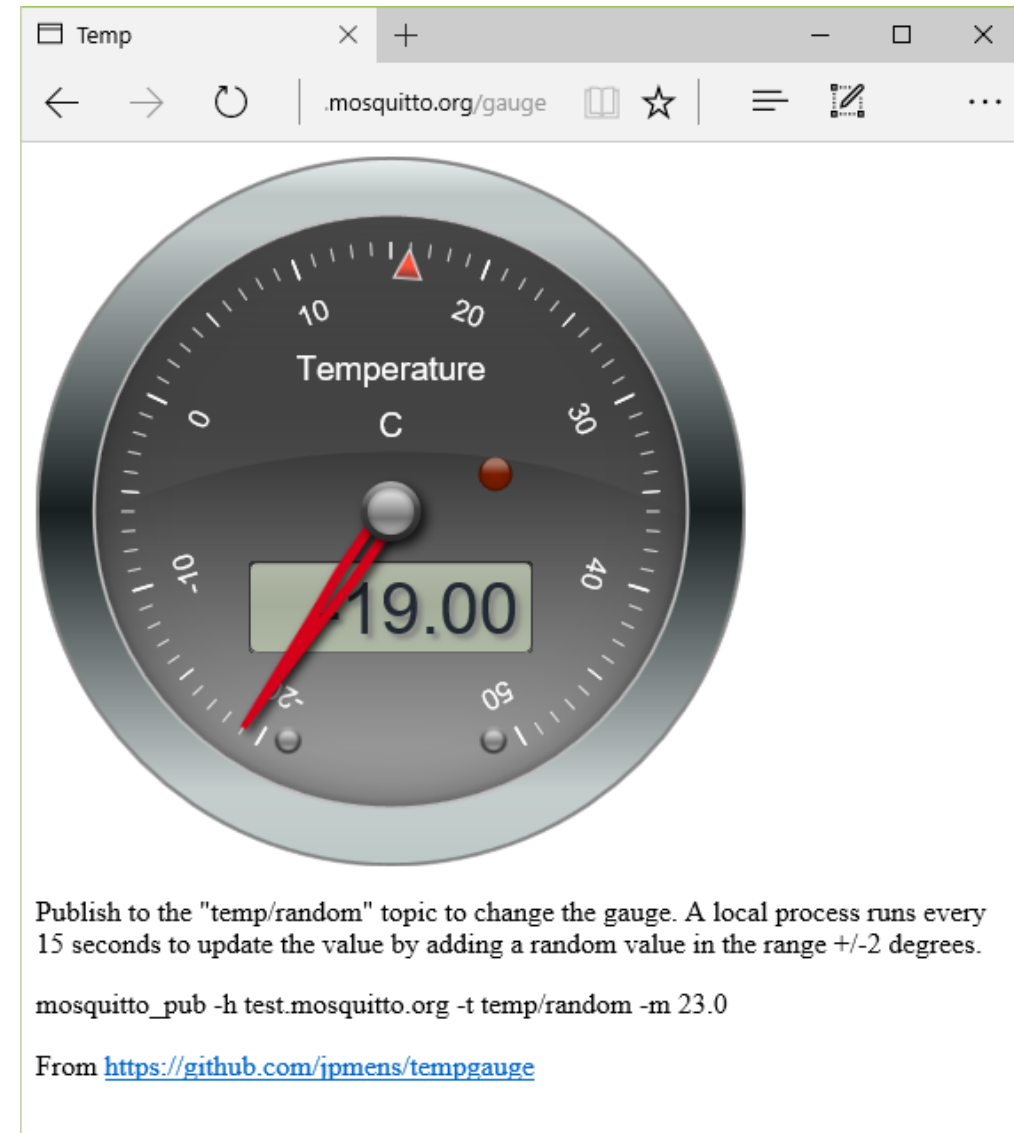
    System.out.println("Connecting to broker: "+broker);
    sampleClient.connect(connOpts);
    System.out.println("Connected");

    System.out.println("Publishing message: "+content);
    MqttMessage message = new MqttMessage(content.getBytes());
    message.setQos(qos);
    sampleClient.publish(topic, message);   ,
    System.out.println("Message published");

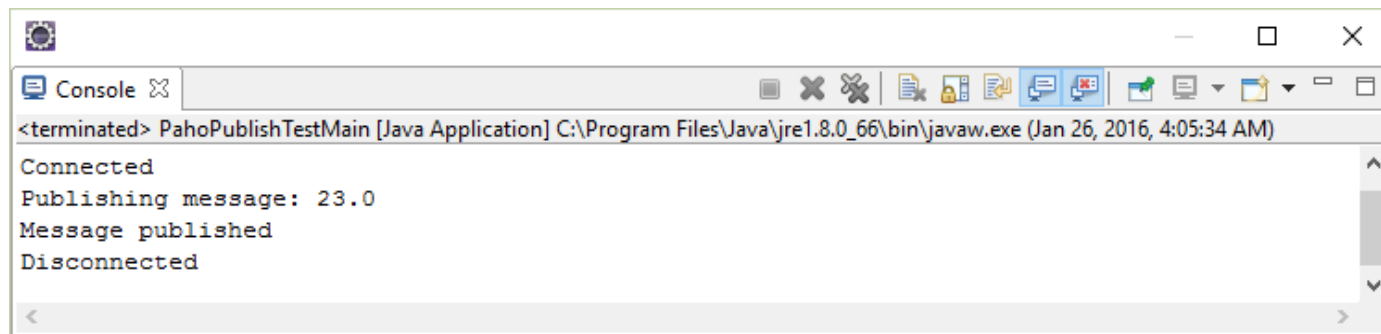
```

```
sampleClient.disconnect();  
System.out.println("Disconnected");  
System.exit(0);  
} catch(MqttException me) {  
    System.out.println("reason "+me.getReasonCode());  
    System.out.println("msg "+me.getMessage());  
    System.out.println("loc "+me.getLocalizedMessage());  
    System.out.println("cause "+me.getCause());  
    System.out.println("excep "+me);  
    me.printStackTrace();  
}  
}  
}
```

- 브라우저에서 다음 링크에 접속
  - <http://test.mosquitto.org/gauge/>



- Eclipse의 PahoPublishTestMain.java 실행
  - PahoPublishTestMain.java 편집기 위에서 마우스 우 클릭 후 “Run As”  
→ “Java Application”을 선택하거나, Ctrl + F11을 입력
  - 실행이 정상적으로 진행된다면, 인터넷 브라우저 창의 온도계의 온도가 23도로 바뀌는 것을 확인할 수 있음
- 정상적으로 실행되는 경우 Eclipse의 콘솔 창에 다음과 같은 메시지가 출력



```
<terminated> PahoPublishTestMain [Java Application] C:\Program Files\Java\jre1.8.0_66\bin\javaw.exe (Jan 26, 2016, 4:05:34 AM)
Connected
Publishing message: 23.0
Message published
Disconnected
```

## Paho 클라이언트 라이브러리를 이용한 구독(subscribe) 테스트

- 앞의 예제와 마찬가지로 Eclipse 프로젝트 생성
  - class 이름: PahoSubscribeTest
- 주의
  - Paho 클라이언트 라이브러리는 각 Java Project마다 따로 설정해야 함
  - Paho 클라이언트 라이브러리 Jar 파일을 새로운 프로젝트에 복사한 후 프로젝트 설정에서 추가



```

import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;

public class PahoSubscribeTest implements MqttCallback {
    /**
     * MqttCallbaxk  imple
     * @param args
     */

    public static void main(String[] args) {
        PahoSubscribeTest test = new PahoSubscribeTest();
        test.runClient();
    }

```

```

public void runClient() {
    String topic    = "Temp/#";
    int qos        = 2;
    String broker   = "tcp://test.mosquitto.org:1883";
    String clientId = "JavaSample";
    MemoryPersistence persistence = new MemoryPersistence();

    try {
        MqttClient sampleClient = new MqttClient(broker, clientId,
        persistence);
        sampleClient.setCallback(this);
        MqttConnectOptions connOpts = new MqttConnectOptions();
        connOpts.setCleanSession(true);

        System.out.println("Connecting to broker: "+broker);
        sampleClient.connect(connOpts);
        System.out.println("Connected");
    }
}

```

```

sampleClient.subscribe(topic, qos);
System.out.println("Subscribing message");

Thread.sleep(5000);

sampleClient.disconnect();
System.out.println("Disconnected");
System.exit(0);
} catch (InterruptedException ie) {
    ie.printStackTrace();
} catch (MqttException me) {
    System.out.println("reason "+me.getReasonCode());
    System.out.println("msg "+me.getMessage());
    System.out.println("loc "+me.getLocalizedMessage());
    System.out.println("cause "+me.getCause());
    System.out.println("excep "+me);
    me.printStackTrace();
}
}

```

```

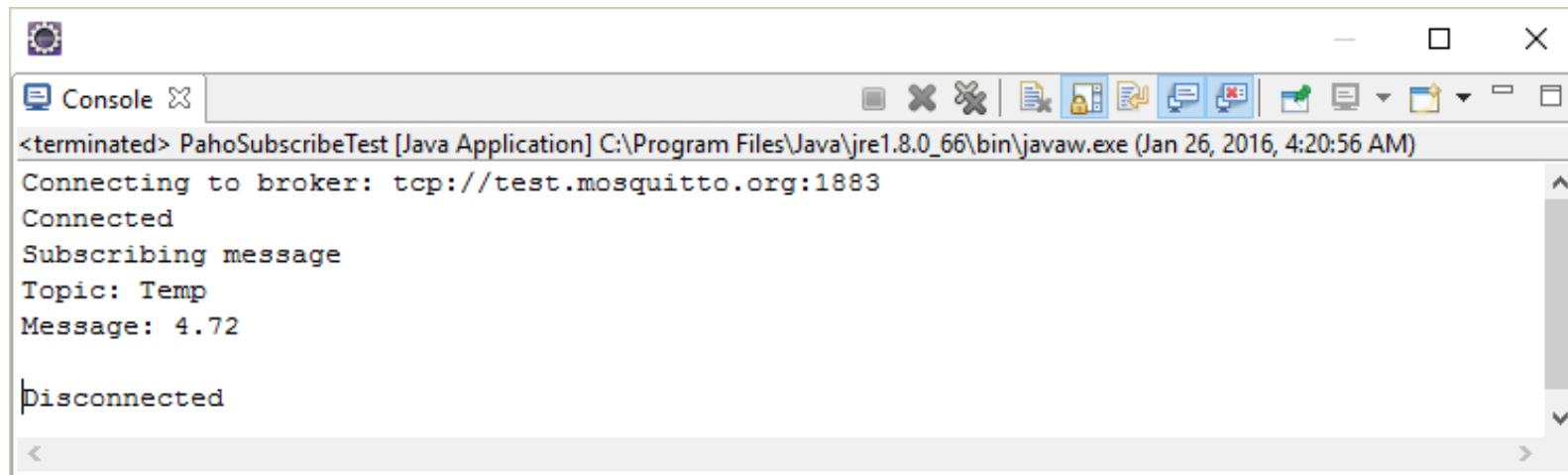
@Override
public void connectionLost(Throwable t) {
    System.out.println("Connection lost!");
}

@Override
public void deliveryComplete(IMqttDeliveryToken token) {
}

@Override
public void messageArrived(String topic, MqttMessage
message) throws Exception {
    System.out.println("Topic: " + topic);
    System.out.println("Message: " + new
String(message.getPayload()));
    System.out.println();
}
}

```

- Eclipse의 PahoSubscribeTest.java 실행
  - 정상적으로 실행되는 Eclipse의 콘솔 창에서 구독된 메시지를 출력하다가 5초 후에 종료됨
    - 메시지는 서버에서 제공하는 데이터에 따라 달라질 수 있음



```
<terminated> PahoSubscribeTest [Java Application] C:\Program Files\Java\jre1.8.0_66\bin\javaw.exe (Jan 26, 2016, 4:20:56 AM)
Connecting to broker: tcp://test.mosquitto.org:1883
Connected
Subscribing message
Topic: Temp
Message: 4.72
Disconnected
```

# Paho 자바 클라이언트 사용법

- 클라이언트 객체 생성
  - 모든 MQTT 동작은 MqttClient 객체를 통해 이루어짐
  - MQTT 브로커에 접속하거나, 발행/구독 동작을 실행하기 전에, MqttClient 인스턴스를 생성한다

```
MqttClient client = new MqttClient(  
    "tcp://broker.mqttdashboard.com:1883", // URI  
    MqttClient.generateClientId(), // ClientId  
    new MemoryPersistence() // Persistence  
);
```

3

## 3가 가

- URI 지정
  - 프로토콜을 항상 명시 (여기서는 'tcp')
  - 포트 번호도 항상 같이 입력
  - 만약 로컬 머신에서 Mosquitto를 이용한 Mqtt 브로커를 이용할 경우, Mosquitto를 실행한 후 URI에 "tcp://127.0.0.1:1883"을 입력
  - 만약 다른 포트 번호를 사용한다면 해당 포트 번호로 설정
- MQTT 클라이언트 ID
  - MQTT 브로커 입장에서 항상 유일무이해야 한다
  - 만약 따로 관리하지 않고 임시로 접속할 경우에는 MqttClient에서 제공하는 generateClientId() 함수를 사용
- Persistence
  - 발행/구독 메시지가 전송 중에 지정된 QoS를 적용하기 위한 경우 사용한다.
  - 해당 기능을 사용하지 않을 경우 in-memory persistence를 사용하는 new MemoryPersistence()를 지정
  - 만약 persistence 변수를 생략하면 파일 기반의 persistence가 기본으로 적용

- MQTT 브로커 접속

- MqttClient 객체를 생성한 후, 지정된 MQTT 브로커에 접속
- connect() 함수 사용

```
client.connect();
```

- isConnected() 함수

- MqttClient 객체가 MQTT 브로커에 이미 접속되어 있는지 확인
- 만약 이미 접속되어 있으면 true를 반환

MQTT  
/

- 특정 버전의 MQTT 브로커 접속

- 만약 특정한 MQTT 브로커 버전에 접속하기 위해서는 MqttConnectionOptions 를 사용
- 기본적으로 Paho는 MQTT 3.1.1버전에 접속을 시도하고 실패할 경우 MQTT 3.1버전에 접속을 시도
- 특정 MQTT 버전을 지정하기 위해서는 MqttConnectionOptions 객체를 생성한 후 해당 버전을 명시
- connect() 함수를 이용할 때 MqttConnectionOptions 객체를 같이 넘겨줌

```
MqttConnectOptions options = new MqttConnectOptions();  
  
options.setMqttVersion(MqttConnectOptions.MQTT_VERSION_3_1);  
// options.setMqttVersion(MqttConnectOptions.MQTT_VERSION_3_1_1);  
  
client.connect(options);
```

- 사용자이름 / 비밀번호 설정
  - 만약 MQTT서버에서 특정 사용자이름을 사용할 경우 MqttConnectOptions 객체를 이용하여 설정
  - 주의할 점
    - 사용자 이름은 String 형태로 입력
    - 비밀번호는 char 배열 형태로 전달

```
MqttConnectOptions options = new MqttConnectOptions();  
options.setUserName("username");  
options.setPassword("password".toCharArray());  
client.connect(options);
```



- 발행(publish)

- MQTT 브로커를 통해 MQTT 메시지를 발행하기 위해서는 publish() 함수를 사용
- Paho 라이브러리를 통해 한 문장 명령어 (one-line command)로 간단하게 수행할 수 있음

```
client.publish(  
    "topic", // topic  
    "payload".getBytes(UTF_8), // payload  
    2, // QoS  
    false); // retained?
```

Topic

MQTT 메시지 토픽

payload

MQTT 메시지 내용

MQTT 메시지 내용은 byte 배열 형태로 입력

QoS

메시지 전달시에 적용하는 QoS 값을 의미

0, 1, 2 중에 하나의 값을 선택

Retained 가 ,

해당 메시지를 서버에서 보유하고 있을지 여부를 표현  
만약 해당 메시지를 브로커에서 보유하게 하기 위해서는  
retained 값을 true로 설정

- 구독(subscribe)

- MQTT 메시지를 받기 위해서, 클라이언트는 MQTT 메시지 토픽과 QoS 레벨을 지정
- 구독을 통해서 받아오는 MQTT 메시지를 처리하기 위해서는 먼저 `MqttCallback`을 등록
  - 해당 콜백 함수들은 MQTT 브로커를 통해 메시지를 전달받으면 실행
  - 안전한 메시지 구독을 보장하기 위해서는 클라이언트가 MQTT 브로커에 접속하기 전에 콜백 함수를 설정해야 함
  - 만약 접속 후에 설정할 경우 특정 메시지의 수신을 놓칠 수도 있음
- `subscribe()` 함수 이용
  - `subscribe()` 함수는 MQTT 메시지 토픽과 QoS 레벨을 입력 값으로 받음
  - 만약 여러 메시지 토픽에 대해서 구독을 신청할 경우, 메시지 토픽들과 각 토픽들의 QoS 값을 각각 String 배열, integer 배열로 설정한 후 `subscribe()` 함수에 인자로 전달
  - 콜백 함수에서 메시지 topic을 확인하여 구분 후 처리

```
client.setCallback(new MqttCallback() {  
  
    @Override  
    public void connectionLost(Throwable cause) {  
        //Called when the client lost the connection to the broker  
    }  
  
    @Override  
    public void messageArrived(String topic, MqttMessage message) throws Exception {  
        System.out.println(topic + ": " + Arrays.toString(message.getPayload()));  
    }  
  
    @Override  
    public void deliveryComplete(IMqttDeliveryToken token) {  
        //Called when a outgoing publish is complete  
    }  
});  
  
client.connect();  
client.subscribe("#", 1);
```

- 구독 취소(Unsubscribe)

- 특정 토픽에 대한 구독 취소는 unsubscribe() 함수 이용
- 여러 토픽을 한 번에 취소하기 위해서는, 해당 토픽들을 String 배열 형태로 unsubscribe() 함수에 전달

```
client.unsubscribe("#");
```

- 접속 종료

- MQTT 브로커와 접속을 종료하기 위해서는 `disconnect()` 함수를 사용
- MQTT DISCONNECT 메시지를 보내기 전에 지연된 동작이 있을 경우 `disconnect()` 함수는 동작이 끝날 때까지 대기
  - 최대 30초까지 동작들이 완료되기를 기다리고 그 후에도 동작이 끝나지 않으면 강제 종료
  - 만약 특정 타임아웃을 지정하고 싶을 경우 `milliseconds` 단위의 타임아웃 시간을 `disconnect()` 함수의 인자로 전달

```
client.disconnect();
```

# 학습 정리

- 라즈베리 파이에서 MQTT 서버와 클라이언트를 구동하여 테스트 해보자
  - 파이썬 클라이언트 라이브러리를 이용하여 라즈베리 파이에서 수집한 센서 데이터를 MQTT 서버에 발행(publish)하는 프로그램을 작성한다
  - 센서 데이터를 구독(subscribe)하는 프로그램을 작성한다
  - 라즈베리 파이에서 발행 클라이언트와 구독 클라이언트를 실행하여 센서 데이터 전송이 잘 이루어지는지 확인한다