
임베디드 시스템

jCoAP Open Source 실습 3

In-Hyeok Kang

M23522@hallym.ac.kr

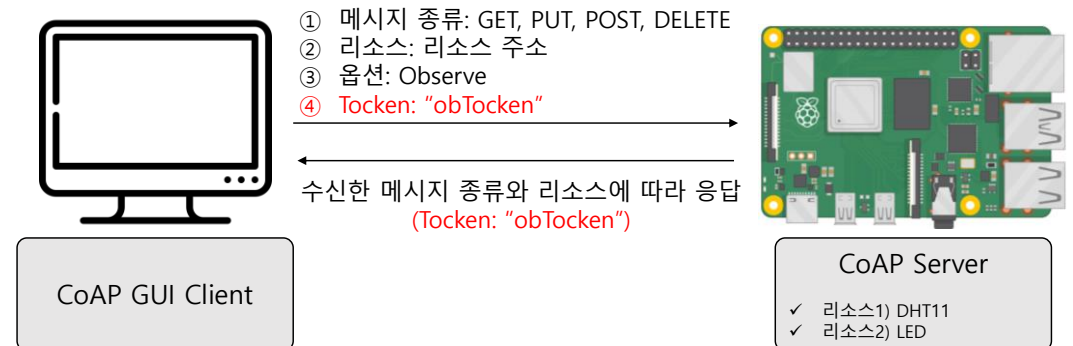
연구실: 공학관 1321호

Contents

1. CoAP
2. DHT11 및 LED 연결
3. CoAP Server & Client 실행 및 결과

CoAP

- CoAP (Constrained Application Protocol)
 - CoAP는 사물인터넷과 같이 대역폭이 제한된 통신 환경에 최적화되어 개발된 Representational State Transfer (REST) 기반의 경량 메시지 전송 프로토콜임
 - CPU, 메모리, 통신 Bandwidth 등이 제한된(Constrained) 기기를 위한 Application Protocol
 - CoAP는 RESTful 기반의 프로토콜이므로 기존의 HTTP (Hypertext Transfer Protocol) 웹 프로토콜과의 연동이 쉬움
 - 자원 관리를 위해 HTTP와 동일하게 **GET, PUT, POST, DELETE**의 메소드를 사용함
 - 자원 발견(Resource Discovery), 멀티캐스트 지원, 비동기 트랜잭션 요청 및 응답 등을 지원함
 - **Sever - Client 구조**



CoAP

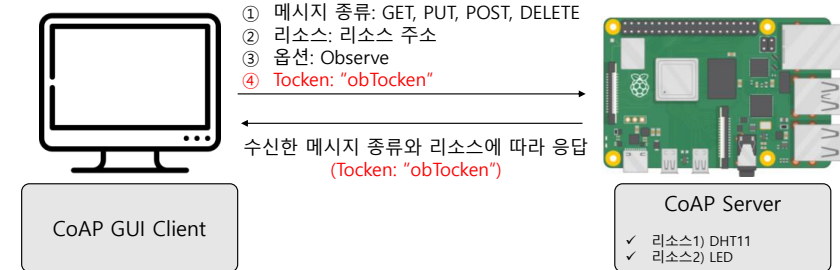
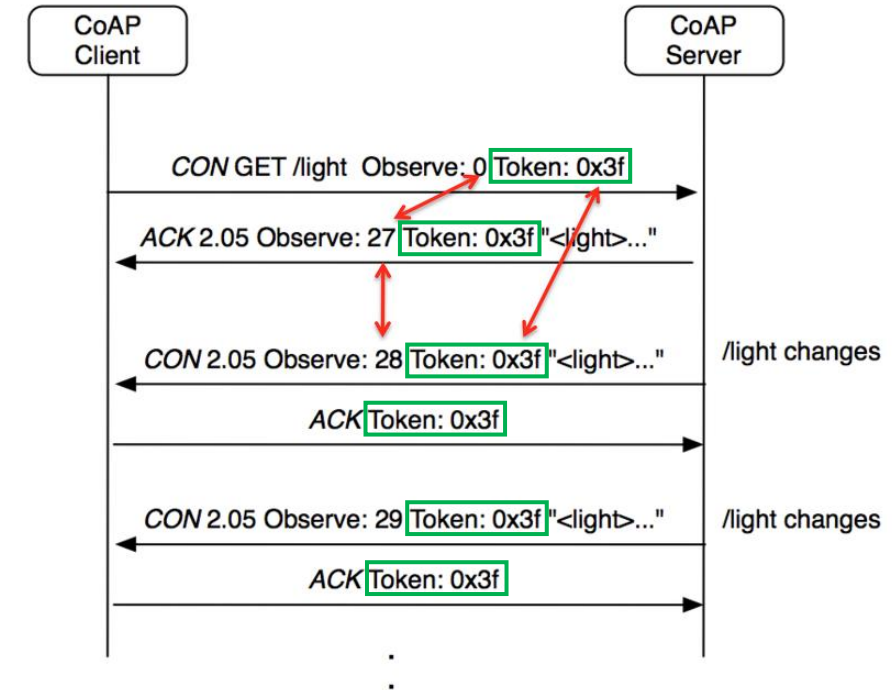
- CoAP (Constrained Application Protocol)

- 리소스/자원(Resource)

- ✓ 센서, 액추에이터, 사용자 정보 등 CoAP Client가 사용할 수 있는 자원임
- ✓ 일반적으로 각각의 센서 또는 액추에이터가 하나의 리소스로 정의됨
- ✓ 사용자 이름, 서버의 상태 등 다양한 형태의 리소스가 존재할 수 있음

- Observe option

- ✓ 리소스의 상태가 변경될 때마다 알림을 받고자 할 때 사용
- ✓ Observer가 여러 리소스에 관심이 있는 경우, 모든 리소스에 대해 개별적인 등록 필요



Request

- CoAP Server의 리소스 조회 → GET 요청
- CoAP Server의 리소스 변경 → PUT 요청
- **Observe GET (리소스 값이 변경될 때만 리소스 값 수신)**

Response

- 다수의 리소스 관리
- CoAP Client의 요청에 따른 응답 수행
- CoAP Client가 Observe를 요청한 메시지의 종류 및 리소스에 대해 주기적으로 리소스 값 전송

CoAP

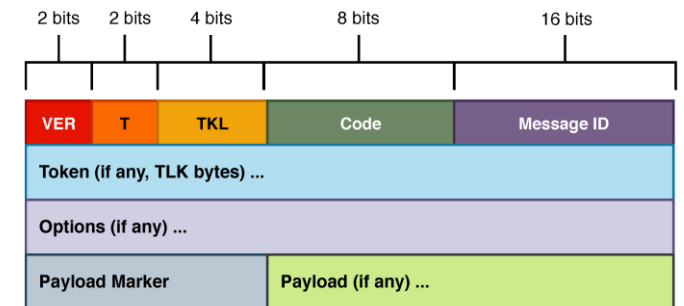
- CoAP (Constrained Application Protocol)

- Message Format

- ✓ CoAP의 메시지 포맷은 Binary 형식으로 인코딩 되어 전송되며, 4-Byte의 고정 헤더, 8-Byte 길이의 Token과 Options, Payload Marker, Payload로 구성됨

4-byte
고정 헤더

- ✓ **Version (Ver):** CoAP의 버전 넘버
 - ✓ **Type (T):** 메시지 타입 – Confirmable (0), Non-confirmable (1), Acknowledgement (2), Reset (3)
 - ✓ **Token Length (TKL):** 토큰 필드의 길이
 - ✓ **Code:** 요청/응답 코드 8-bit unsigned integer – 3-bit: class, 5-bit: detail
Class 0: 요청, Class 2: 성공적인 응답, Class 4: 클라이언트 에러 응답, Class 5: 서버 에러 응답
 - ✓ **Message ID:** 메시지의 중복 확인 및 Acknowledgement/Reset 메시지를 Confirmable/Non-confirmable 유형의 메시지와 일치시키는 데 사용됨
 - ✓ **Token (optional):** Request와 response를 연관시키는 데 사용됨
 - ✓ **Options (optional):** Content-Format, ETag, Location-Path 등의 다양한 옵션 존재
 - ✓ **Payload Marker:** Token과 Options의 끝을 나타냄, 0xFF의 값을 가짐
 - ✓ **Payload:** 전송되는 데이터



< CoAP Message Format >

DHT11 및 LED 연결

• 2. 구성품 연결

- [LED 센서]

✓ [점프 와이어(F/F)]로 연결

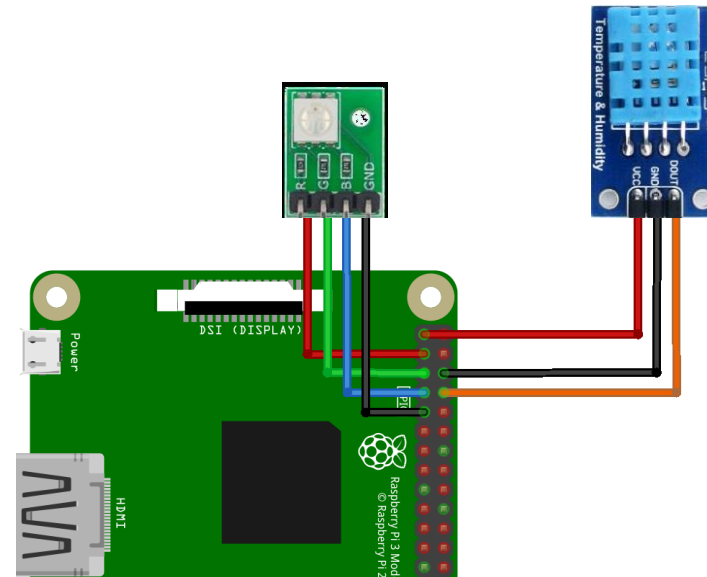
LED 센서 모듈	GPIO Pins
R	8
G	9
B	7
VCC	Ground

- [DHT11 온습도 센서]

✓ [점프 와이어(F/F)]로 연결

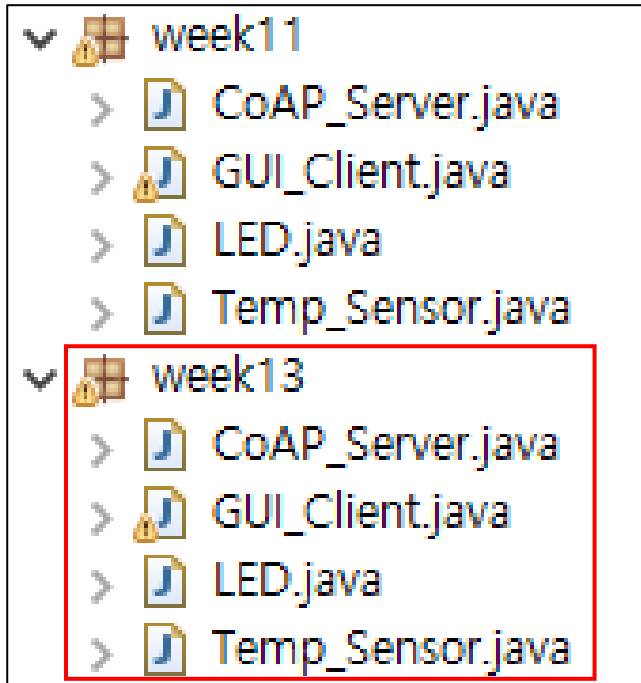
DHT11 온습도 센서	GPIO Pins
DOUT	15
GND	Ground
VCC	3.3 VDC

GPIO#	NAME		NAME	GPIO#
	3.3 VDC Power	1	2	5.0 VDC Power
8	GPIO 8 SDA1 (I2C)	3	4	5.0 VDC Power
9	GPIO 9 SCL1 (I2C)	5	6	Ground
7	GPIO 7 GPCLK0	7	8	GPIO 15 TxD (UART) 15
	Ground	9	10	GPIO 16 RxD (UART) 16
0	GPIO 0	11	12	GPIO 1 PCM_CLK/PWM0 1
2	GPIO 2	13	14	Ground
3	GPIO 3	15	16	GPIO 4 4



CoAP Server & Client 실행 및 결과

- week13 패키지 생성 → **week11 패키지 내 모든 .java 파일을 week13 패키지로 복사**
 - **week11의 observe:** 5초마다 주기적으로 온도 값 출력
 - **week13의 observe:** 온도 값 변경 시에만 온도 값 출력 및 LED 색상 변경



CoAP Server & Client 실행 및 결과

- 1. Temp_Sensor.java 소스 코드

- (a): 온도 센서의 데이터를 읽고, 이전 데이터와 비교하여 변동되었을 때만 구독자들에게 변경된 정보를 알림

- ✓ 온도 센서(DHT11)에서 데이터 읽기
 - ✓ dht.getData(15)를 사용
 - ✓ 온도 데이터를 저장할 변수를 추가로 생성
- ✓ 예외 처리
 - ✓ 온도가 변하지 않았을 경우 처리
 - ✓ Temperature has not changed 문구 출력
 - ✓ 온도 측정 오류 처리(-99.0일 경우)
 - ✓ Temperature Measurement Error 문구 출력
 - ✓ 온도가 변했을 경우 처리
 - ✓ changed() 메서드를 통해
 - 구독하고 있는 클라이언트들에게 변경 온도 전송
 - ✓ 다음 비교를 위해 value에 새로운 온도 값을 저장

```
@Override
public synchronized CoapData get(List<CoapMediaType> mediaTypesAccepted) {
    float[] sensing_data = dht.getData(15);
    this.value = Float.toString(sensing_data[1]);
    return new CoapData(Encoder.StringToByte(this.value), CoapMediaType.text_plain);
}
```

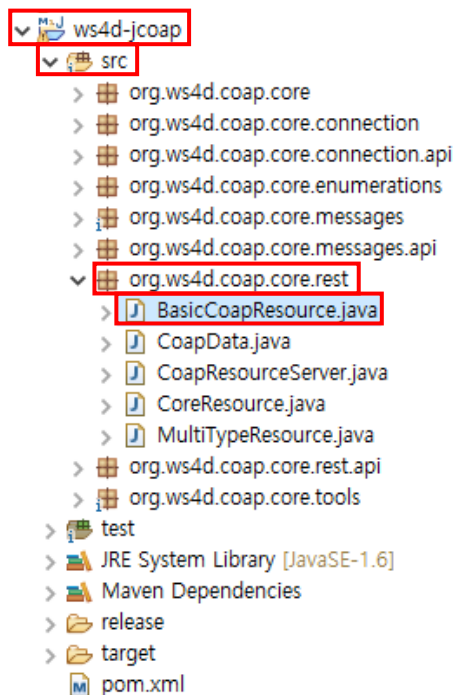
```
//week13
public synchronized void optional_changed() {
```

(a)

```
}
```


CoAP Server & Client 실행 및 결과

- 2. BasicCoapResource.java 소스 코드
 - changed() 메소드 아래 changed(String data) 메소드 추가



```
public synchronized void changed(String data) {  
    if (this.serverListener != null) {  
        this.serverListener.resourceChanged(this);  
    }  
    this.observeSequenceNumber++;  
    if (this.observeSequenceNumber > 0xFFFF) {  
        this.observeSequenceNumber = 0;  
    }  
  
    // notify all observers  
    for (CoapRequest obsRequest : this.observer.values()) {  
        CoapServerChannel channel = (CoapServerChannel) obsRequest.getChannel();  
        CoapResponse response;  
        if (this.reliableNotification == null) {  
            response = channel.createNotification(obsRequest, CoapResponseCode.Content_205,  
                this.observeSequenceNumber);  
        } else {  
            response = channel.createNotification(obsRequest, CoapResponseCode.Content_205,  
                this.observeSequenceNumber, this.reliableNotification);  
        }  
        response.setPayload(new CoapData(Encoder.StringToByte(data), CoapMediaType.text_plain));  
        channel.sendNotification(response);  
    }  
}
```

changed(String data) 메소드 추가

changed() 메소드의 해당 부분을 수정해 오버로딩

CoAP Server & Client 실행 및 결과

- 3. CoAP_Server.java 소스 코드

```
// run the server
try {
    this.resourceServer.start();
} catch (Exception e) {
    e.printStackTrace();
}

//week13
while (true) {
    try {
        Thread.sleep(3000);
        temp_sensor.optional_changed();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

while loop 내
온도 센싱 주기 및 호출 함수 수정

CoAP Server & Client 실행 및 결과

- 4. GUI_Client.java 소스 코드

btn_observe가 맞는지 확인

```
btn_observe.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        // TODO Auto-generated method stub  
        String path = path_text.getText();  
        String payload = payload_text.getText();  
        CoapRequest request = clientChannel.createRequest(CoapRequestCode.GET, path, true);  
        request.setToken(Encoder.StringToByte("obToken")); // Token 값 설정  
        request.setObserveOption(0); // 시작 sequence number 설정 Token 설정  
        displayRequest(request);  
        clientChannel.sendMessage(request);  
    }  
});  
  
payload_label.setBounds(20, 570, 350, 30);  
payload_text.setBounds(20, 600, 440, 30);  
payload_text.setFont(new Font("arian", Font.BOLD, 15));
```

CoAP Server & Client 실행 및 결과

- 4. GUI_Client.java 소스 코드

```
if (GUI_Client.exitAfterResponse) {
    display_text.append("===END===");
    System.exit(0);
}
display_text.append(System.LineSeparator());
display_text.append("*");
display_text.append(System.LineSeparator());

// LED 제어 (Observe에 대한 Response인지 확인 후, 온도값에 따라 LED 리소스에 대한 PUT 요청 전송)
if (Encoder.ByteString(response.getToken()).equals("obToken")) {
    float temp = Float.parseFloat(Encoder.ByteString(response.getPayload()));
    this.control_led(temp);
}
}
```

onResponse() 메소드 안의
맨 하단에 LED 제어를 위한 조건문 추가


```
public void control_led(float temp) {
    CoapRequest request = clientChannel.createRequest(CoapRequestCode.PUT, "/led", true);
    if (temp > 24.0) {
        request.setPayload(new CoapData("red", CoapMediaType.text_plain));
    } else {
        request.setPayload(new CoapData("green", CoapMediaType.text_plain));
    }
    displayRequest(request);
    clientChannel.sendMessage(request);
}
```

control_led 메소드 추가
: 온도 값의 변화에 따라 LED 색상 변경

CoAP Server & Client 실행 및 결과

- 5. JAR 파일 생성 후 XFTP를 통해 Raspberry Pi로 전송
- 6. Raspberry Pi에서 JAR 파일 실행(CoAP Server 실행)
 - `sudo java -jar server.jar`

```
pi@raspberrypi:~ $ sudo java -jar Server.jar
===Run Test Server ===
ERROR StatusLogger No log4j2 configuration file found. Using default configuration: logging only errors to the console. Set system property 'log4j2.debug' to show Log4j2 internal initialization logging.
```

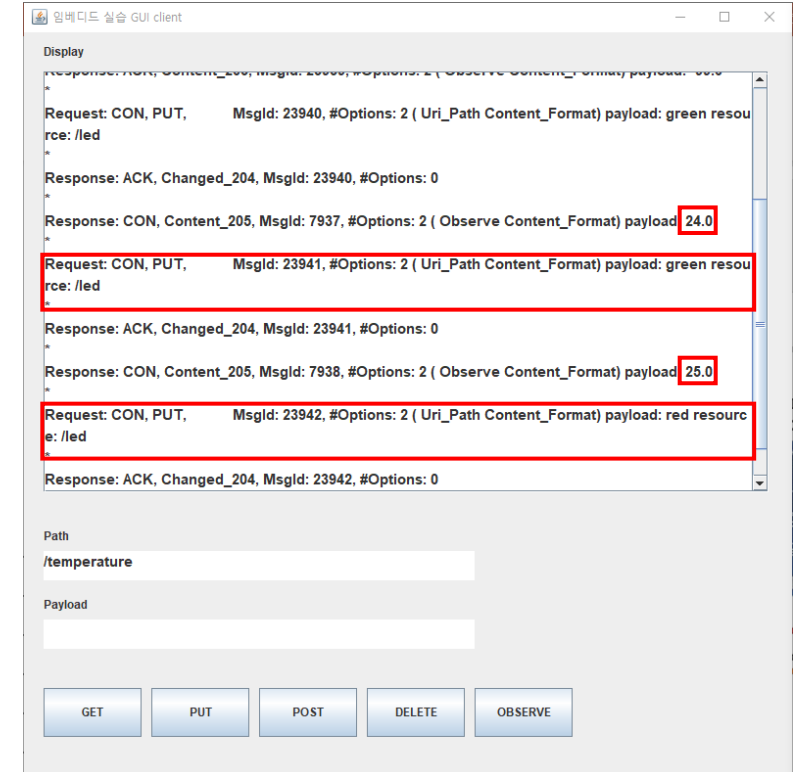


CoAP Server & Client 실행 및 결과

- 7. Eclipse에서 GUI_Client 실행(CoAP Client 실행)
 - Path: /.well-known/core 입력 → **GET** 버튼 클릭 → Display 창에서 등록된 리소스 확인
 - Path: /temperature 입력 → **OBSERVE** 버튼 클릭 → Display 창을 통해 온도가 변화할 때 온도 값이 출력되고 LED 색이 변화하는지 확인해 볼 것

온도 값 출력: 24°C 이하
LED 색상: Green

온도 값 변경: 24 → 25°C 이상
LED 색상: Green → Red





감사합니다

Thank You

