IoT

Modbus InfluxDB

Table of contents

1 IoT

1.1 IoT

- -
- - PLC
- -
- -
- -

1.2 Python IoT

: - - - - -

: - `pyserial` - - `pymodbus` - - `influxdb-client` - DB - `paho-mqtt` - IoT - `asyncio` - I/O

2

2.1 PySerial

```
import serial
import time

#
ser = serial.Serial('/dev/ttyUSB0', baudrate=9600, timeout=1)

#
ser.write(b'READ_SENSORS\n')

#
response = ser.readline()
data = response.decode('utf-8').strip()

#
if ':' in data:
    temp, humidity = data.split(',')
    temp_value = float(temp.split(':')[1])
    humidity_value = float(humidity.split(':')[1])

ser.close()
```

2.2 Arduino

```
class ArduinoInterface:
    def __init__(self, port='/dev/ttyACM0'):
        self.ser = serial.Serial(port, 9600, timeout=2)
        time.sleep(2)  # Arduino

    def read_sensors(self):
        self.ser.write(b'GET_DATA\n')
        response = self.ser.readline()

        try:
            data = json.loads(response.decode())
            return {
                'temperature': data['temp'],
                'humidity': data['hum'],
                'timestamp': datetime.now()
            }
        except:
            return None

    def control_led(self, pin, state):
        command = f"LED,{pin},{'ON' if state else 'OFF'}\n"
        self.ser.write(command.encode())
```

2.3

2.3.1

```
def safe_serial_read(ser, timeout=5):
    try:
        ser.timeout = timeout
        data = ser.readline()
        if data:
            return data.decode('utf-8').strip()
    except serial.SerialException as e:
        logger.error(f"    : {e}")
    except UnicodeDecodeError:
        logger.warning("     ")
    return None
```

2.3.2

```
def reconnect_serial(port, baudrate, max_attempts=5):
    for attempt in range(max_attempts):
        try:
            ser = serial.Serial(port, baudrate, timeout=1)
            return ser
        except serial.SerialException:
            time.sleep(2 ** attempt)  #
    raise ConnectionError("    ")
```

3 Modbus

3.1 Modbus

Modbus - - - TCP/IP - PLC

: - - (R/W) - - (RO) - - (RO) - - (R/W)

3.2 PyModbus

```python
from pymodbus.client.sync import ModbusTcpClient

class ModbusController:
    def __init__(self, host='192.168.1.100', port=502):
        self.client = ModbusTcpClient(host, port=port)
        self.connected = self.client.connect()

    def read_sensors(self):
        #
        result = self.client.read_holding_registers(0, 3, unit=1)

        if result.isError():
            return None

        return {
            'temperature': result.registers[0] / 100.0,  # °C
            'pressure': result.registers[1] / 10.0,     # bar
            'flow_rate': result.registers[2]            # L/min
        }

    def control_pump(self, pump_id, state):
        #
        self.client.write_coil(pump_id, state, unit=1)
```

3.3

```
class IndustrialSystem:
    def __init__(self):
        self.plc = ModbusController('192.168.1.100')
        self.setpoints = {'temperature': 25.0, 'pressure': 2.5}

    def monitor_process(self):
        sensors = self.plc.read_sensors()

        #
        if sensors['temperature'] > self.setpoints['temperature'] + 2:
            self.plc.control_pump(0, False)  #
        elif sensors['temperature'] < self.setpoints['temperature'] - 2:
            self.plc.control_pump(0, True)   #

        #
        if sensors['pressure'] > 3.0:
            self.plc.control_pump(1, False)  #
            self.send_alert("   ")

        return sensors
```

4 InfluxDB

4.1

- -
- -
- - Flux
- -
- **Grafana** -

4.2 InfluxDB

```python
from influxdb_client import InfluxDBClient, Point
from influxdb_client.client.write_api import SYNCHRONOUS

class IoTDataManager:
    def __init__(self, url, token, org, bucket):
        self.client = InfluxDBClient(url=url, token=token, org=org)
        self.write_api = self.client.write_api(write_options=SYNCHRONOUS)
        self.query_api = self.client.query_api()
        self.bucket = bucket
        self.org = org

    def write_sensor_data(self, device_id, location, measurements):
        points = []
        for field, value in measurements.items():
            point = Point("sensors") \
                .tag("device_id", device_id) \
                .tag("location", location) \
                .field(field, value)
            points.append(point)

        self.write_api.write(bucket=self.bucket, org=self.org, record=points)
```

4.3

```
def get_device_stats(self, device_id, hours=24):
    query = f'''
    from(bucket: "{self.bucket}")
      |> range(start: -{hours}h)
      |> filter(fn: (r) => r.device_id == "{device_id}")
      |> group(columns: ["_field"])
      |> aggregateWindow(every: 1h, fn: mean, createEmpty: false)
      |> yield(name: "hourly_average")
    '''

    result = self.query_api.query(org=self.org, query=query)

    stats = {}
    for table in result:
        for record in table.records:
            field = record.get_field()
            value = record.get_value()
            time = record.get_time()

            if field not in stats:
                stats[field] = []
            stats[field].append({'time': time, 'value': value})

    return stats
```

5 IoT

5.1

```
class IoTMonitoringSystem:
    def __init__(self):
        self.devices = {}
        self.data_queue = queue.Queue()
        self.influx_client = IoTDataManager(...)
        self.running = False

    def add_device(self, device_id, device_type, **config):
        if device_type == 'serial':
            device = ArduinoInterface(config['port'])
        elif device_type == 'modbus':
            device = ModbusController(config['host'])

        self.devices[device_id] = {
            'interface': device,
            'config': config,
            'last_reading': None
        }

    def collect_data(self):
        while self.running:
            for device_id, device_info in self.devices.items():
                try:
                    data = device_info['interface'].read_sensors()
                    if data:
                        self.data_queue.put((device_id, data))
                except Exception as e:
                    logger.error(f"{device_id}     : {e}")

            time.sleep(5)  # 5
```

5.2

```
def process_data(self):
    while self.running:
        try:
            device_id, data = self.data_queue.get(timeout=1)

            #
            data['device_id'] = device_id
            data['timestamp'] = datetime.now()

            #
            if self.validate_data(data):
                # InfluxDB
                self.influx_client.write_sensor_data(
                    device_id=device_id,
                    location=self.devices[device_id]['config']['location'],
                    measurements=data
                )

                #
                self.check_alerts(device_id, data)

            self.data_queue.task_done()

        except queue.Empty:
            continue
        except Exception as e:
            logger.error(f"    : {e}")
```

6

6.1

```python
class SmartGreenhouse:
    def __init__(self):
        self.arduino = ArduinoInterface('/dev/ttyACM0')
        self.data_manager = IoTDataManager(...)
        self.optimal_conditions = {
            'temperature': (20, 28),  #  ,
            'humidity': (60, 80),
            'soil_moisture': (30, 70)
        }

    def monitor_and_control(self):
        #
        sensors = self.arduino.read_sensors()

        #
        self.data_manager.write_sensor_data(
            'greenhouse_001', 'facility_a', sensors
        )

        #
        if sensors['temperature'] > 28:
            self.arduino.control_fan(True)

        if sensors['soil_moisture'] < 30:
            self.arduino.control_irrigation(True)

        return sensors
```

6.2

```
class FactoryMonitoring:
    def __init__(self):
        self.plc = ModbusController('192.168.1.100')
        self.data_manager = IoTDataManager(...)
        self.production_line = {
            'target_rate': 1000,  # /
            'efficiency_threshold': 0.85
        }

    def monitor_production(self):
        #
        data = self.plc.read_production_counters()

        #
        current_rate = data['units_produced'] / data['runtime_hours']
        efficiency = current_rate / self.production_line['target_rate']

        #
        metrics = {
            'production_rate': current_rate,
            'efficiency': efficiency,
            'downtime': data['downtime_minutes'],
            'quality_score': data['quality_percentage']
        }

        self.data_manager.write_equipment_status('line_001', metrics)

        #
        if efficiency < 0.85:
            self.send_alert(f"  : {efficiency:.2%}")
```

7

7.1

7.1.1

```
def retry_on_failure(max_retries=3, delay=1.0):
    def decorator(func):
        @functools.wraps(func)
        def wrapper(*args, **kwargs):
            for attempt in range(max_retries):
                try:
                    return func(*args, **kwargs)
                except Exception as e:
                    if attempt == max_retries - 1:
                        raise e
                    time.sleep(delay)
            return wrapper
        return decorator

@retry_on_failure(max_retries=5, delay=2.0)
def read_device_data(device):
    return device.read_sensors()
```

7.1.2

```
class CircuitBreaker:
    def __init__(self, failure_threshold=5, timeout=60):
        self.failure_threshold = failure_threshold
        self.timeout = timeout
        self.failure_count = 0
        self.last_failure_time = None
        self.state = 'CLOSED'  # CLOSED, OPEN, HALF_OPEN
    def call(self, func, *args, **kwargs):
        if self.state == 'OPEN':
            if time.time() - self.last_failure_time > self.timeout:
                self.state = 'HALF_OPEN'
            else:
                raise Exception("         ")

        try:
            result = func(*args, **kwargs)
            self.on_success()
            return result
        except Exception as e:
            self.on_failure()
            raise e
```

7.2

- - Modbus TCP   TLS
- - API
- - IoT
- -
- -
- -

8

8.1

: - PySerial - Modbus - InfluxDB - I/O -

: - - - - -

8.2

- -
- -
- -
- - HVAC
- -
- -

8.3

1. - Arduino Raspberry Pi PLC
2. - Grafana Web
3. -
4. -
5. - MQTT OPC-UA CAN
6. -

8.4

Python

IoT

8.5

:

:

|

8.6

Python

IoT