Script Server

- script server의 역할

  script(input, 계산식, output)를 등록해놓고 새로운 태그 생성

  이후 해당 태그로 새로운 알람포인트를 만들거나 모델학습에 사용됨

  개발자보다는 현직에 있는 관리자들이 사용하는 기능

- script서버 .py별 기능

▼ main.py

- 4개의 쓰레드로 구성됨
  - run_api_server

    host, port 지정후 uvicorn run
  - rawvalues_collector

    collect_kafka_script_value함수 사용

    raw_value를 kafka를 통해 수신

    (Kafka는 from_ipcm과 연결되어있음)

    Consumer()객체를 생성해서 while문으로 종료신호 전까지 계속해서 값을 수신
  - script_calculator

    계산결과를 보내기위해 Producer()객체 생성

    customerscriptmanager를 통해 script 결과 계산

    Update되는 값만 producer로 전송

    - 해당 producer에는 to_ipcm에 대한 부분이 없어서 값이 어디로 가는 지 확인 필요
  - cleanup_debug_client

    5초마다 디버그 세션데이터 정리

calc.py에 debugsession_manager사용

▼ api

▼ Script.py

- add_script

  새로운 커스텀 태그 스크립트 추가

  calc_script_manager의 save_cunstom_tag함수 사용

  - save_custom_tag : 새로운 script를 DB에 저장
    (save_input과 save_output이 결합되어있음)

- register_script

  새로운 커스텀 태그 등록

  calc_manager의 create_custom_tag, register_calc_tag함수 사용

- unregister_script

  특정 커스텀 태그 스크립트 등록 해제

  calc_manager의 unregister_calc_tag함수 사용

▼ calc.py

- 함수
  - get_last_data

    최신 데이터 불러오기

  - Convert_df2dict

    Dataframe구조 → dict구조로 변경하기

- api
  - syntax_validation

    코드 구분 검증하기

    - 과정
      1. request불러오기 - _ScriptInfo(CustomCalcScript.py)
      2. custom_script생성(request기반의 데이터들)
      3. get_last_data로 생성된 custom_script에 최신데이터 입력하기
      4. custom_script.calc를 통해 계산

5. 계산이 잘 수행되었는지 메시지출력(오탈자, 사용변수명, output tag명등)

```
result = {
      "message": "Success",
      "details": {
         "static_vars": custom_script.initialization_code,
         "code": custom_script.calculation_code,
         "calc_result": calc_result,
      },
   }
```

- caculate

  계산 수행하기

  - 과정

    1. request불러오기 - request_script_calc(customCalcScript.py)
    2. custom_script생성(request기반의 데이터들)
    3. 현재 시점확인(datetime.datetime.now())
    4. 최신데이터를 넣어서 계산하기
    5. 결과입력

```
script_id = 요청받은 script의 id
script_name = 요청받은 script의 일
input_data = 요청받은 script의 input data
calc_result = 요청받은 script의 계산결과
calc_at = 요청받은 시간
calc_time_taken = 요청받은 script의 계산시간
```

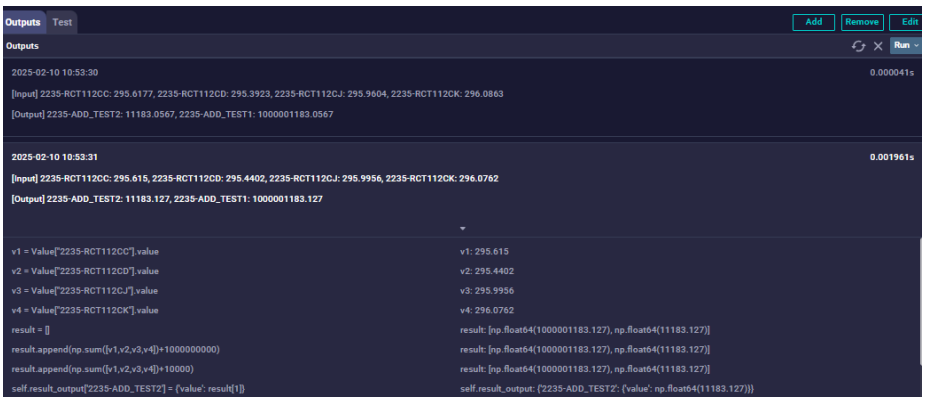
- Debug(calc와 거의 비슷함 - 계산과정만 추가)
  - 과정
    1. Request불러오기

       RequestScript_calc(CustomCalcScript.py)
    2. Custom_script생성(request기반의 데이터들)

3. 현재시점확인
4. 최신데이터 넣어서 결과 확인하기
5. 디버그 로그 찍기
6. 결과 입력

> calc와의 차이점(추가되는 요소)
> *** calc_result를 result_output으로 받음 - 이유찾기 ***
> calc_result_log = 요청받은 script의 로그

- Clac와의 차이점



Debug는 전체 과정, 변수들을 보여주면서 왜 이런 결과가 나왔는지 확인가능

- Excution_time

  계산 테스트 수행 1초동안 많은 계산하고 총 계산시간과 평균계산시간 확인

  - 과정

    Calcscript생성 → 최신데이터 계산 → 계산 소요시간 확인

- Plot

  Script기반으로 input데이터와 outputdata의 Trend확인

  - 과정
    1. Calc_script생성
    2. Influx에서 실시간 데이터 수신하기
    3. 계산 후 결과 출력 (inputdata, output_result)

▼ customcalctag.py

사용자 정의 스크립트 관리

▼ customscript

개별 사용자 정의 스크립트를 나타내는 클래스

- 기능

  스크립트 초기화

  스크립트 코드 컴파일 후 실행

  입출력 태그 데이터 관리

- 과정

  1. db에서 받은 데이터들로 script정보 입력
  2. output_code : 출력코드 생성 (결과tag = tag1)
  3. combine_code : 계산코드와 출력코드 결합 (계산식 = 결과tag)
  4. initial_code실행(최초 한번만 실행됨)
  5. compile_code : combine코드 컴파일하기
  6. 마지막 데이터 구조 생성해두기

- 추가 기능

  - 마지막 데이터 업데이트하기
  - 코드 규칙 검증(사용x)

▼ custom_script_manager

- load_custom_script

  postgreDB에서 script정보 불러온 후 customscript(class)로 script로 만든 후 등록하기

- save_custom_taginput

  tag정보 postgreDB에 저장

```
custom_tag.script_name,
custom_tag.input_tagnames,
```

```
custom_tag.initialization_code,
custom_tag.calculation_code,
```

- save_custom_tagoutput

```
script_id,
output_tag.tagname,
output_tag.tagname,
output_tag.script,
```

- save_custom_tag

  input과 output동시 진행하게 합치기

- register_calc_tag

  custom_script객체에 새로운 script추가하기

- unregister_calc_tag

  custom_script객체에서 특정 script제거하기

- calc_scripts

  값 계산하기

- 과정

  1. load_custom_script로 DB에서 데이터 불러온 후 새로운 script등록
  2. Kafka에서 초신 데이터를 가져와 태그값 업데이트
  3. 등록된 스크립트 실행으로 결과생성
  4. 스크립트 추가제거할때 DB와 객체 모두 적용