pybind11を使ってpythonで利用可能なc++とeigenライブラリの開発環境をDockerで作ってほしい

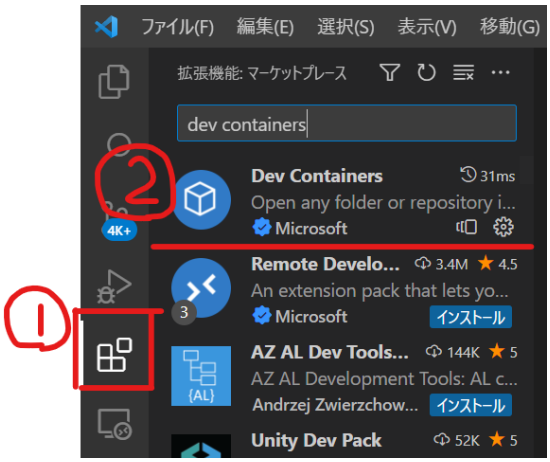
pybind11を使ってc++の共有ライブラリを作成し、Pythonから実行できる環境を提供する。
開発においてはVSCodeからPython, c++それぞれブレークポイントを指定してデバッグをする事ができる。

Requirement

Windowsローカルに下記アプリケーションがインストールされている事

- WSL
- Docker Desktop
- VSCode

VSCodeの拡張機能DevContainersがインストールされている事



Installation

1. zipファイルをローカルに展開(下記フォルダパスは一例)

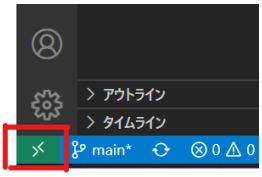
```
C:\FEMPython
```

2. WSLでDockerfileをbuild、runする

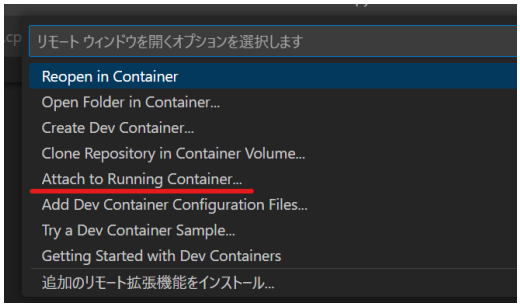
```
cd /mnt/c/FEMPython
docker build -t fem .
docker run --name fem -dit fem
```

3. VSCodeでRemoteログインする

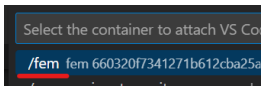
- VSCodeの左下の緑色のボタンをクリック



- 出てきたメニューからAttach to Running Container...をクリック

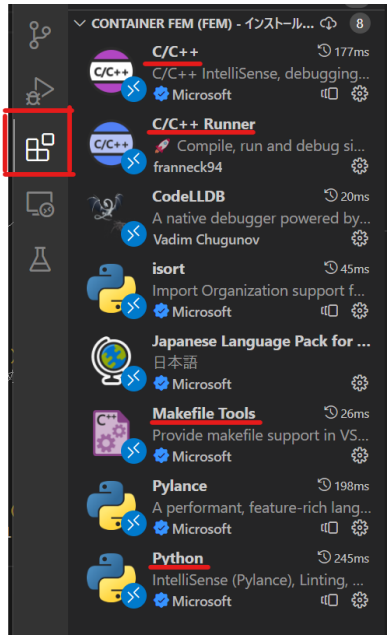


- /fem を選択してコンテナ内にVSCodeで入る



4. 拡張機能で必要な機能をDevContainerにインストール

- Python
- C/C++
- C/C++ Runner
- Makefile Tools

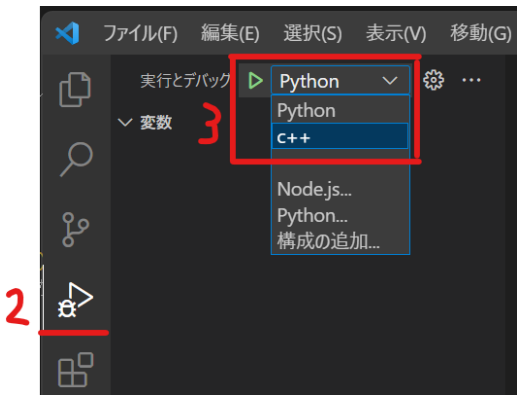


Usage

デバッグ方法

1. /FEMPython/tests/test_001.py を開く
2. 実行とデバッグをクリック

3. Pythonもしくはc++を選択してデバッグ実行(F5を押した場合は直近選ばれていたデバッガが選択される)
4. 指定したデバッガに対応するPythonコードもしくはc++コード内のブレークポイントでコード実行を一時停止できる



Note

各ファイルの説明

- Dockerfile
  - PythonとEigenをインストール
  - Pythonの仮想環境構築は今回見送り(複雑になるため)
  - Eigenはgit cloneのみ。コンパイル時に組み込むためbuildは不要

- cppファイル
  - FileIO.cpp
    - readFemModelメソッドを定義
    - Eigenも読み込んでおり、プログラム内で使用可能(実際に呼び出しはしていない)
    - pybind11形式でreadFemModel関数とFemDataModelクラスを参照ファイル化する
  - FemDataModel.h FemDataModel.cpp
    - readFemModelのreturnとして返すクラスを定義。メソッドはかなり簡素化+private変数に文字列を設定、読み出せる関数を追加。

- Makefile
  - SRCSにコンパイルする(#includeでつながっている)cppファイルを全て定義の上pybind11の共有ライブラリ形式(.so)でコンパイル
  - コンパイルのキーワードに-gが入っていることでデバッグが可能になる。

- Pythonファイル
  - pybind11でコンパイルした参照ファイルを同じフォルダ内に配置することでimport FileIOでコンパイルしたメソッド、クラスを呼び出すことができる。
  - Pythonで読み込んだテキストをreadFemModelメソッドの引数に指定して、返り値となるFemDataModelクラスのインスタンスのprivate変数に格納して返している。
  - Python上ではFileIO.cppで指定したメソッド(インスタンス内変数の読み出しなど)を使用することができる。

- /FEMPython/.vscode (デバッグ設定が記述されており重要)

    - launch.json
        - Pythonのデバッガにc++のデバッガを組み込む形になっている。
        - pybindで生成される共有ライブラリ形式(.so)ファイルは実行ファイルではないので通常のコンパイルはできない。
        - pythonとの混合デバッグモードにより、ブレークポイントを指定したデバッグが可能になる。
        - **pythonコードからしかデバッグできないので注意！**
    - tasks.json
        - デバッグを行う際に事前に行う処理などを記載しておける
        - ここでは下記の処理をlaunch.jsonの"preLaunchTask"(デバッグ前に実施する処理)に指定してある
            - 共有ライブラリ形式(.so)ファイルのコンパイル
            - /FEMPython/testsフォルダへのリネーム＆移動

プログラム拡張時のメモ

- cppファイルを共有ライブラリ(.so)に追加する際はMakefileのSRCSにファイル名を追加する事で、コンパイル、リンクができるようになる

```
# 使用するcppファイルはSRCS追加
SRCS = FileIO.cpp
SRCS += FemDataModel.cpp
# SRCS += AAA.cpp  # cppを追加するときはここで追加していく

```

- Python側で呼び出したいクラスがある場合は使用するメソッドを全てFileIO.cppファイルに書き込んでおく。ここに記載がないとc++側で定義していてもPython側で呼び出せない。

```
PYBIND11_MODULE(FileIO, m){

    // 関数のbind
    m.def("readFemModel", &readFemModel);

    // クラスのbind
    pybind11::class_<FemDataModel>(m, "FemDataModel")
            .def(pybind11::init())
            .def("init", &FemDataModel::init)
            .def("clear", &FemDataModel::clear)
            .def("read_string", &FemDataModel::readString)
            .def("print_string", &FemDataModel::printString);

}

```

- 共有ライブラリ名はコンパイルするファイル名と同じにする必要がある(この場合はFileIO.cpp) 変更する場合はMakefileのFileIOの記載を変更してコンパイル

```

FileIO: $(SRCS)
    g++ -O3 -Wall -shared -std=c++20 -fPIC `python3 -m pybind11 --includes` -g $(SRCS) -o FileIO`python3-config --exte
# -gが重要であり、デバッグ用のビルドが可能にするオプション。これを入れることでブレークポイントでのデバッグができる。
```

- 今回は仮想環境に直接インストールしたPythonを使用したが、仮想環境を作成する場合はMakefileにPythonパスの指定が必要(今回未検証)
  参考URL：https://qiita.com/exy81/items/e309df7e33d4ff20a91a

ubuntu on Docker版 (2017/11/12追記)
5行目が少しだけ変わっています

```
clang++ -std=c++14 \
  -shared \
  -I<git cloneしたディレクトリ>/pybind11/include \
  -L/<Anacondaインストールディレクトリ>/lib/ \
  `/<Anacondaインストールディレクトリ>/bin/python3-config --cflags --ldflags` -fPIC \
  -o mylibs.so \
  mylibs.cpp
```

オプションについて説明すると
-shared : ライブラリ作成
-I : pybind11のインクルードパスを指定
-L : python用のライブラリパスを指定 (python3-configが相対パスしか出力しないため必要となる)
python3-config : python向けライブラリをコンパイルするときのオプションを出力するコマンド
-o : ライブラリのファイル名指定

Author

- 作成者 : Konihey