FrontISTR の解析制御ファイルにデータブロックを追加する方法

2013 年 10 月 9 日
東大奥田研 D1　稲垣和久

1. はじめに

本稿では FrontISTR の解析制御ファイルに新規種類のデータブロック (!で始まるヘッダー行とそれに続くデータ行) を追加する方法を説明する。対象とする FrontISTR のバージョンは 3.4c である。

本稿の構成は次の通りである。はじめに第 2 節で FrontISTR における解析制御ファイルの構文解析について説明する。ここでは構文解析サブルーチンの呼出し位置・処理内容と、主要なデータ処理サブルーチン群について述べる。その後に第 3 節で具体的なヘッダーの追加手順を説明する。また本稿の末尾では、補足として FrontISTR の入力ファイル処理全体についての概要説明を行った。

2. 解析制御ファイルの構文解析について

i. 構文解析サブルーチンの呼出し位置

解析制御ファイルの構文解析を行うサブルーチンの名称は fstr_setup である。図 1 に fstr_setup の呼出し位置を示す。FrontISTR プログラム開始から fstr_setup の呼出しまでは、全て src/main フォルダにある fistr_main.f90 に記述されている。コードを見るとプログラム開始から fstr_init→fstr_init_condition→fstr_setup の順で call されていることが確認できる。

src/main/fistr_main.f90

```
 18  program fstr_main
 58  call hecmw_init
 65  call hecmw_get_mesh   !メッシュ構造体の setup
 71  call fstr_init
122    subroutine fstr_init
       ～構造体の初期化など～
168    call fstr_init_condition
265      subroutine fstr_init_condition
271      call hecmw_ctrl_get_control_file  !解析制御ファイル名の取得
277      call fstr_setup
282      write(*,*) 'fstr_setup: OK'       !完了メッセージの表示
```

図 1. fstr_setup の呼出し位置

ii. fstr_setup の処理

fstr_setup は/src/common フォルダにある fstr_setup.f90 に定義されている。その処理内容を図 2 に示す。

src/common/fstr_setup.f90

```
53   subroutine fstr_setup
97   P%MESH   => hecMESH
...  ...                          !type(fstr_param_pack)::P にポインタ割付
104  P%FREQ   => fstrFREQ
108  c_solution = 0;
...  ...                          !各種カウンタの初期化
117  c_fload    = 0;
119  ctrl = fstr_ctrl_open        !解析制御ファイルの open

127  do !１回目の読み込み
128    rcode = fstr_ctrl_get_c_h_name !次のヘッダーを取得
129    if( header_name == '!VERSION' ) then
131    else if( header_name == '!SOLUTION' ) then
...    ...
265    else if( header_name == '!END' ) then
266      exit
267    end if          !END が見つかるか次のヘッダーが見つからないと exit
270    if( fstr_ctrl_seek_next_header(ctrl) == 0) exit
271  end do
       ～構造体の初期化など(1)～
313  rcode = fstr_ctrl_rewind    !先頭から読み直し

322  do !２回目の読み込み
323    rcode = fstr_ctrl_get_c_h_name !次のヘッダーを取得
325    if( header_name == '!ORIENTATION' ) then
334    else if( header_name == '!CONTACT' ) then
...    ...
576    else if( header_name == '!END' ) then
577      exit
578    end if          !END が見つかるか次のヘッダーが見つからないと exit
580    if( fstr_ctrl_seek_next_header(ctrl) == 0) exit
581  end do
       ～構造体の初期化など(2)～
673  end subroutine fstr_setup
```

図 2. fstr_setup の処理

はじめに L97 から L124 にかけて、読み込みのための各種初期化処理が行われる。fstr_param_pack は hecMESH や fstrSOLID といったデータ構造体のポインタをまとめた構造体であり、各データブロックの解析を行うサブルーチンにこれらの構造体を引き渡すために使われる。fstr_setup 内では変数名「P」で 1 つだけ定義されている。また、「c_」で始まる integer 型のデータは、各データブロックの出現回数をカウントするものである。fstr_ctrl_open は解析制御ファイルを open する関数である。

初期化処理が終わると 1 回目の解析ループに入る（L127～L271）。このループでは「次のヘッダーを探索」「見つかったヘッダーの種類で分岐してデータブロックの解析を実行」を繰り返し、「!END ヘッダーが見つかる」か「次のヘッダーが見つからない」と終了する。1 回目のループで大部分のヘッダーの処理が行われるが、!STEP や!MATERIAL など一部のヘッダーについては出現回数のカウントのみ行う[1]。

1 回目の解析ループ終了後、ヘッダー出現回数のカウントのみ行われたデータについてメモリの確保が実行される（L271～L313）。関連するカウンタ類をクリアしたのち再び解析制御ファイルを冒頭から読み直し、2 回目のループが実行される（L313～L581）。

iii. データ処理サブルーチン群

個々のデータブロック処理はサブルーチン fstr_setup_*（*にはヘッダーの名称が入る）の中で行われる。fstr_setup_*は fstr_setup.f90 の fstr_setup 以降で定義されている。処理の内容はデータブロックの種類によって様々なので、ここでは共通する項目についてのみ説明を行う。

(1) fstr_ctrl_get_*（*にはヘッダーの名称が入る）

実際にデータブロックを分析し、構造体に格納する役割を果たすサブルーチンである[2]。ごく簡単な処理の場合は fstr_setup_*で処理が済まされ、定義されない場合もある。このサブルーチンはデータブロックが属するカテゴリに応じて異なるファイルに定義されている。

(a) 共通：fstr_ctrl_common.f90　…STEP、SOLVER、CONTACT など
(b) 静解析：fstr_ctrl_static.f90　…STATIC、BOUNDARY、CLOAD など
(c) 動解析：fstr_ctrl_dynamic.f90　…VELOCITY、ACCELERATION など
(d) 固有値解析：fstr_ctrl_eigen.f90　…EIGEN
(e) 周波数応答解析：fstr_ctrl_freq.f90　…FLOAD
(f) 熱伝導解析：fstr_ctrl_heat.f90　…HEAT、FIXTEMP、CFLUX など
(g) 材料：fstr_ctrl_material.f90　…MATERIAL、ELASTICITY、PLASTICITY など
(h) その他：fstr_ctrl_modifier.f90　…MPC など

いずれのファイルも src/common の下にある。新たにデータブロックを作成する際は、対応する fstr_ctrl_get_*を適切なカテゴリのファイルに定義することが望ましい。

(2) fstr_ctrl_get_param_ex

ヘッダー行に記述されるオプションの内容を解析し、値の取得を行う関数である。定義は src/common 内にある fstr_ctrl_util.c（c 言語のファイル）に記述されており、fstr_ctrl_util_f.inc を介

[1]カウントのみ行うヘッダーには、「任意個定義可能なデータ」で「専用のデータ構造体が定義されていて初期化処理が必要」なものが該当しているように思います（!STEP、!MATERIAL、!CONTACT など）。!BOUDARY や!CLOAD など、単なる integer/real 配列に格納されるものは、可変長配列を使って 1 回目のループで処理してしまうようです。

[2] fstr_setup_*と fstr_ctrl_get_*について、どのようなポリシーで機能が区分けされているか確証が持てないのですが、見た感じでは格納領域の確保が fstr_setup_*の仕事、確保された領域へのデータ格納が fstr_ctrl_get_*の仕事のようです。

して include されている。その仕様は次の表 1 の通りである。

```
int fstr_ctrl_get_param_ex( fstr_ctrl_data* ctrl, const char* param_name,
    const char* value_list, int necessity, char type, void* val );
```

表 1. fstr_ctrl_get_param_ex の仕様

変数型	変数名	説明
fstr_ctrl_data*	ctrl	fstr_control_data のポインタ（データブロックのポインタ）
const char*	param_name	パラメータの名称
const char*	value_list	パラメータの取りうる値のリスト（type='P'の時のみ有効）
int	necessity	パラメータの必要性（1：必須、0：必須でない）
char	type	変数タイプ（'I'：整数、'C'or'S'：文字列、'R'：実数、 'P'：パターン、'E'：存在）
void*	val	パラメータの格納先
int	return	0：成功、1：パラメータが無い、2：値が無い 3：型変換に失敗、4：不正な値の範囲

この関数の引数を様々に変えることで、多様な型のオプションが読み込まれるようになる。以下に実際の使用例を示す。

【使用例 1】!STEP の MAXITER オプション

```
fstr_ctrl_get_param_ex( ctrl, 'MAXITER ', '# ', 0, 'I', steps%max_iter )
```

…指定は必須でなく、読み取った整数値を integer 型変数 steps%max_iter に格納する。

【使用例 2】!SOLVER の METHOD オプション

```
character(72) :: mlist = '1,2,3,4,101,CG,BiCGSTAB,GMRES,GPBiCG,...,MUMPS'
fstr_ctrl_get_param_ex( ctrl, 'METHOD ', mlist, 1, 'P', method )
```

…指定は必須、mlist の中で一致するパターンの ID（登場順。'1'なら 1、'CG'なら 6）を integer 型変数 method に格納する。

【使用例 3】!WRITE の VISUAL オプション

```
fstr_ctrl_get_param_ex( ctrl, 'VISUAL ', '# ', 0, 'E', visual )
```

…指定は必須でなく、ヘッダー行に'VISUAL'オプションが存在すれば 1 を、しなければ 0 を integer 型変数 visual に格納する。

(3) fstr_ctrl_get_data_line_n

データ行の行数を返す関数。fstr_ctrl_util.c に定義。

(4) fstr_ctrl_get_data_ex

指定した行のデータを取得する。fstr_ctrl_util.c に定義。仕様は表 2 の通り。

フォーマットは読み込み行のデータの型と個数を指定する文字列であり、'i'を整数、'r'を実数、's'を文字列として、これらを並べて定義する。フォーマットは大文字も使用可。データはカンマ区切りで与える。具体的には次の例のように指定する。

【指定例 1】'rrr '：3 つの実数。たとえば「0.1, 0.5, -0.0」

【指定例 2】'ir ' : 1 つの整数と 1 つの実数。たとえば「10, 0.0」
【指定例 3】'SIIR ' : 1 つの文字列、2 つの整数、1 つの実数。たとえば「GRP1, 1, 1, 10.0」

```
int fstr_ctrl_get_data_ex ( fstr_ctrl_data* ctrl, int line_no, const char* format, ...);
```

表 2. fstr_ctrl_get_data_ex の仕様

変数型	変数名	説明
fstr_ctrl_data*	ctrl	fstr_control_data のポインタ（データブロックのポインタ）
int	line_no	取得する行番号
const char*	format	フォーマット
int	return	0：成功、-1：失敗

格納先となる変数は、指定したフォーマットに従って必要な個数を与える。使用例は次の通り。
【使用例 1】!SOLVER の 1 行目を読み込み

```
fstr_ctrl_get_data_ex( ctrl, 1, 'iii ', nier, iterpremax, nrest )
```

1 行目（integer×3）を読みこみ、それぞれの値を nier、interpremax、nrest に格納。
この関数は、データ行の形式が行ごとに異なる場合に用いる。

(5) fstr_ctrl_get_data_array_ex

テーブル形式のデータを一括で取得する。fstr_ctrl_util.c に定義。仕様は表 3 の通り。

```
int fstr_ctrl_get_data_array_ex ( fstr_ctrl_data* ctrl, const char* format, ...);
```

表 3. fstr_ctrl_get_data_array_ex の仕様

変数型	変数名	説明
fstr_ctrl_data*	ctrl	fstr_control_data のポインタ（データブロックのポインタ）
const char*	format	フォーマット
int	return	0：成功、-1：失敗

フォーマットの定義は fstr_ctrl_get_data_ex と同じである。各行のデータは全て同じフォーマットに従っているものとして扱われ、格納先（データ型に応じた配列）に一括で代入される。使用例は次の通り。

【使用例 1】!ELASTIC, DEPENDENCY=1（温度依存弾性材）のデータ行読み込み

```
data_fmt = 'RRR '
fstr_ctrl_get_data_array_ex( ctrl, data_fmt, fval(1,:), fval(2,:), fval(3,:) )
```

各行とも real×3 のデータ型であり、第 1 変数（ヤング率）を fval の第 1 行に、第 2 変数（ポアソン比）を fval の第 2 行に、第 3 変数（温度）を fval の第 3 行に一括で格納する。

3. データブロックの追加手順

これまで説明した内容をもとに、本節では具体的なデータ格納手順について説明する。

i. 変数の追加

はじめに新たに入力するデータの格納先を用意する。既存の変数にデータをセットするだけであればこの作業は必要ない。新規に作成する変数の格納先は、fstr_param_pack で統合されている構造体の中から適切なものを選択し、その配下に作成するのが妥当と考えられる[3]。詳細は割愛するが、これらの構造体はディレクトリ src/lib 内の m_fstr.f90、m_out.f90、m_step.f90 などに定義されているので、ここに定義を追加する[4]。

ii. fstr_setup の編集および fstr_setup_* の作成

次に fstr_setup を新しい種類のヘッダーに対応させる。１回目の解析ループ（fstr_setup.f90・L127～L271）に、新たなヘッダーに対応した else if 文を追加する。さらに、新しいヘッダーに対応した読み込み関数 fstr_setup_*を m_fstr_setup モジュール内に定義し、先ほど追加した else if 文の中で call する。追加例を図 3 に示す。必要であればカウンタを用意し、実際の読み込み処理を２回目のループに回す。

fstr_setup_*の実装については、よほど単純な処理であればこのサブルーチン内で完結させてよいが、そうでない場合は他のデータブロックに倣って fstr_ctrl_get_*と処理を分離して書くと良いと思われる。

```
  …
  else if( header_name == '!NEW_HEADER' ) then          !else if 文の追加
    call fstr_setup_newheader( ctrl, c_newheader, P )
  else if( header_name == '!END' ) then
    exit
  …


subroutine fstr_setup_newheader( ctrl, c_newheader,  P )  !subroutine の定義
```

図 3. 読み込み処理の定義＆呼出し例（!NEW_HEADER）

iii. fstr_ctrl_get_* の作成

最後に読み込みの fstr_ctrl_get_*を定義する。定義を行うファイルはデータブロックの属するカテゴリに応じて適切に選択する（2.(iii)(1) 参照）。2.(iii)(2) ～2.(iii)(5) で説明した関数を必要に応じて組み合わせ、読み込み処理を作成する。この部分の実装は作成したいデータブロックの仕様によって様々であるので、以下では簡単な例を１つだけ挙げる。複雑なデータブロックの実装方法は、他のデータブロックの fstr_ctrl_get_* 関数の実装を参照のこと。

実装例を図 4 に、入力データ例を図 5 に示す。ここでは fstrSOLID の下に integer 型の変数 new_int を新たに定義したとして、!NEW_HEADER のデータ行で入力した値を new_int に設定する処理を行っている。非常に簡単な処理で fstr_ctrl_get_* 関数を作成するほどでもないが、参考のため敢えて分離した場合の実装例を記載した。

---

[3] m_fstr_setup モジュールが use しているモジュール内のグローバル変数も fstr_setup からアクセス可能ですので、ここに新たな変数を作ることも可能なのですが（実際にそういった変数も存在します）、変数の管理上好ましくないと思います。

[4] 初期化処理・終了処理の追加も忘れずに。なお、hecMESH、hecMAT は（!SOLVER の設定を除いて）解析制御ファイルで設定を行うものではないと思いますので、ここでは除いて考えています。

```
～fstr_setup.f90～
subroutine fstr_setup_newheader( ctrl, c_newheader,  P )  !subroutine の定義
  integer(kind=kint), intent(in)         :: ctrl
  integer(kind=kint), intent(in)         :: c_newheader
  type(fstr_param_pack), intent(inout)   :: P

  integer(kind=kint) :: rcode !成否判定
  rcode = fstr_ctrl_get_NEWHEADER( ctrl, P%SOLID%new_int )
  if( rcode /= 0 ) call fstr_ctrl_err_stop !読み込み失敗のときにエラーを返す

end subroutine

～fstr_ctrl_*.f90～
function fstr_ctrl_get_NEWHEADER( ctrl, new_int)  !subroutine の定義
  integer(kind=kint)  :: ctrl
  integer(kind=kint)  :: new_int
  integer(kind=kint)  :: fstr_ctrl_get_NEWHEADER

  !データ行（integer の値一つ）を new_int に格納
  fstr_ctrl_get_NEWHEADER  = fstr_ctrl_get_data_ex( ctrl, 1, 'I' , new_int )
end function
```

図 4. !NEW_HEADER 読み込み処理の実装例

```
～解析制御ファイル：*.cnt～
!NEW_HEADER
10
```

図 5. !NEW_HEADER 入力データ例

【補足】FrontISTR の入力ファイル処理

本節では FrontISTR の入力ファイル処理の全体について、概要を説明する。

はじめに図 6 に FrontISTR のシステム構成と入力の概念図を示す。FrontISTR のプログラムは、メッシュデータ処理、行列処理、通信等の機能を持つミドルウェア hecmw の上に、各種解析を行うアプリ部分が乗った 2 層構造になっている。解析を実行する際には全体制御ファイル、メッシュファイル（単一領域もしくは全体領域)、解析制御ファイルの 3 つを入力する必要があるが、内部的には全体制御ファイルとメッシュファイルをミドルウェアが、解析制御ファイルをアプリ部分が処理する作りになっている。なお、ここで言う「解析制御ファイルを処理するアプリ側のルーチン」が、本稿で説明を行った fstr_setup である。

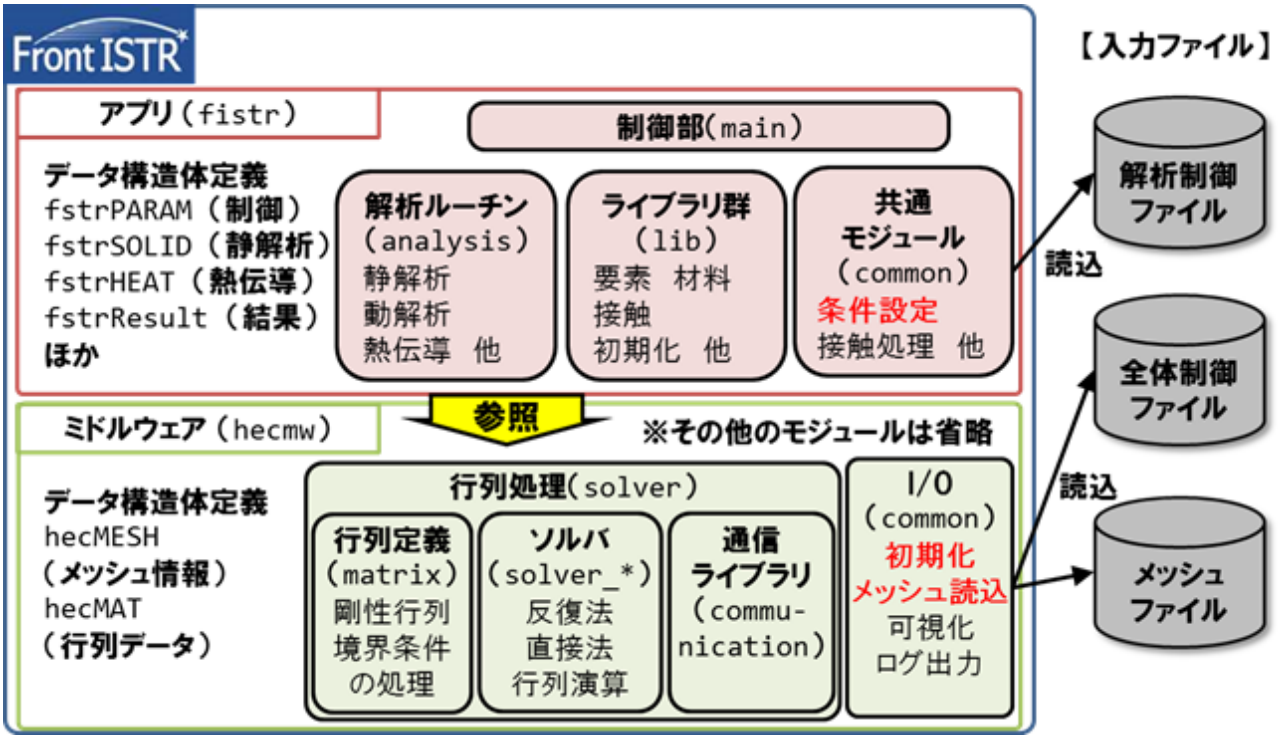

図 6. FrontISTR のシステム構成と入力の概念図

次に、FrontISTR プログラムでの入力ファイル処理の流れを図 7 に示す。

プログラムが始まると、まずミドルウェアの初期化関数 hecmw_init を call する[5]。ここで全体制御ファイルの読み込みを行い、メッシュファイルや解析制御ファイルの名前などを取得する。

続けてミドルウェアのメッシュ読み込み関数 hecmw_get_mesh を call する。内部では全体制御ファイルでの指定に応じて単一領域メッシュの読み込み関数 get_entire_mesh か分散領域メッシュの読み込み関数 hecmw_get_dist_mesh のいずれかが call され、メッシュデータ構造体 hecMESH がセットアップされる[6]。

その後、アプリ側のセットアップサブルーチン fistr_init を call する。内部では各種解析構造体ポインタの初期化後、fstr_init_file という（一見関係ありそうな）名前のサブルーチンが call されるが、これはログファイル類の初期化を行う関数であり入力には関係ない。続けて hecmw_mat_con が call され、ここで hecMESH から行列構造体 hecMAT のコネクティビティ情報が作成される。その後 fstr_setup が call され、解析制御ファイルの設定に応じて各種解析構造体（fstrSOLID など、fstr****名の構造体）がセットアップされる。最後に hecMAT_init で hecMAT のデータ格納領域（hecMAT%D など）が allocate される。

以上が入力ファイル処理の概要である。以降はセットアップされた解析条件に応じて、対応する解析ルーチンが開始される。

---

[5] ミドルウェア関数の呼出しは c - fortran インターフェースを介して行われるため、中身までコードをたどるのは少し複雑です。またミドルウェアでは随所でグローバル変数が定義されており、処理を追うことにも苦労するかもしれません…。

[6] ちなみにこの後に call されている hecmw2fstr_mesh_conv は、要素（主に 4 面体 2 次要素）のコネクティビティを hecmw 用から fistr 用に変換する処理であり、大筋には関係ありません。FrontISTR はミドルウェアの要素ライブラリを使わず、自前で要素ライブラリを定義しているのですが、これらの要素コネクティビティが異なるため必要になっています。

fistr1/src/main/main.f90

```
18  program fstr_main
58    call hecmw_init  !全体制御ファイル hecmw_ctrl.dat の読込み
      hecmw1/src/common/hecmw_util_f.f90
      461  subroutine hecmw_init
      462    character(len=HECMW_FILENAME_LEN):: ctrlfile = "hecmw_ctrl.dat"
      463    call hecmw_init_ex(ctrlfile)  !ここでメッシュファイル名を取得
             …（以下省略）
65    call hecmw_get_mesh( name_ID , hecMESH ) !メッシュファイルの読込み
      hecmw1/src/common/hecmw_io.f90
       37  subroutine hecmw_get_mesh(name_ID, mesh) !読み込み＆mesh 構造体へ格納
             …（以下省略）
71    call fstr_init
      122  subroutine fstr_init
      136    call fstr_init_file                    !ログファイル類初期化
      157    call hecmw_mat_con(hecMESH, hecMAT)    !hecMAT のコネクティビティ作成
      168    call fstr_init_condition
             265  subroutine fstr_init_condition
             277    call fstr_setup                  !解析制御ファイルの読込み
      175    call hecMAT_init( hecMAT )             !hecMAT の領域確保
77    ! =============== ANALYSIS ====================  !解析ルーチンの開始
```

図 7. 入力ファイル処理の流れ

以上