# OpenFOAM® ソースコード構造入門

2015年6月13日

オープンCAE勉強会@富山

富山県立大学　中川慎二

# 注意

- OpenFOAMユーザーガイド，プログラマーズガイド，OpenFOAM Wiki，CFD Online，その他多くの情報を参考にしています。開発者，情報発信者の皆様に深い謝意を表します。
- この講習内容は，講師の個人的な経験（主に，卒研生等とのコードリーディング）から得た知識を共有するものです。この内容の正確性を保証することはできません。この情報を使用したことによって問題が生じた場合，その責任は負いかねますので，予めご了承ください。

一部のクラス図の作成に，astah* community（無償版）を利用しています。
株式会社チェンジビジョン 社製　http://astah.change-vision.com/ja/

# 概要

- OpenFOAM の利用者を対象とし，OpenFOAMのソースコードの読み方の基本の基本を学びます。
- OpenFOAMのソースコードが，どのように整理されているのかを学びます。
- 実際にOpenFOAMのソースコードを見ながら，OpenFOAMのソースコードの特徴，ソースコード解読初心者が躓きやすい点などについても，解説します。

# 目次

- OpenFOAMとC++
  - クラス
  - ソルバ
  - ライブラリ
- OpenFOAMインストールとファイルの場所
- OpenFOAMソースコードの構造
  - ソルバ
  - ライブラリ
- OpenFOAMソースコードの調べ方

# OpenFOAMとC++

# OpenFOAMのソースコード

- C++ 言語
- オブジェクト指向プログラミング
  - カプセル化 (振る舞いの隠蔽とデータ隠蔽)
  - インヘリタンス (継承) -- クラスベースの言語
  - ポリモーフィズム (多態性、多相性) -- 型付きの言語
    - オーバーロード（多重定義）　同じ名前で引数の異なる関数
    - オーバーライド　親クラスの関数を子クラスで上書き
- ジェネリックプログラミング
  - データ型に依存しないコード。Templateを利用。
- OpenFOAM とは、 CFD に必要な機能を追加した C++言語 である。　といっても過言ではない…

# C++ クラスとは

- 部品（オブジェクト）の設計図
- 様々な値（状態，属性）と，それを操作するための機能（function,メソッド， 関数）を含む
- クラスは、値と関数の集まりである
- この設計図（クラス）に基づいて，プログラム実行時に，部品（オブジェクト）が作られる
- オブジェクト生成時にはコンストラクタが働く

一般コード例
```
int     n(7);
int     i = 10;
```
型名　変数名（初期値）

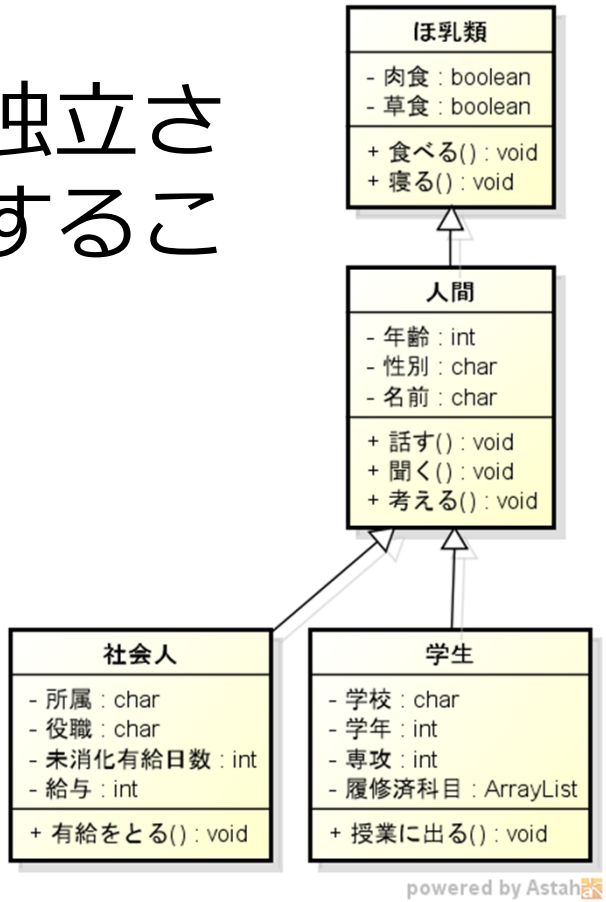
OpenFOAMコード例
```
dimensionedScalar     DT( x, y);
```
クラス名　　オブジェクト名（初期値）

# C++ クラスの継承

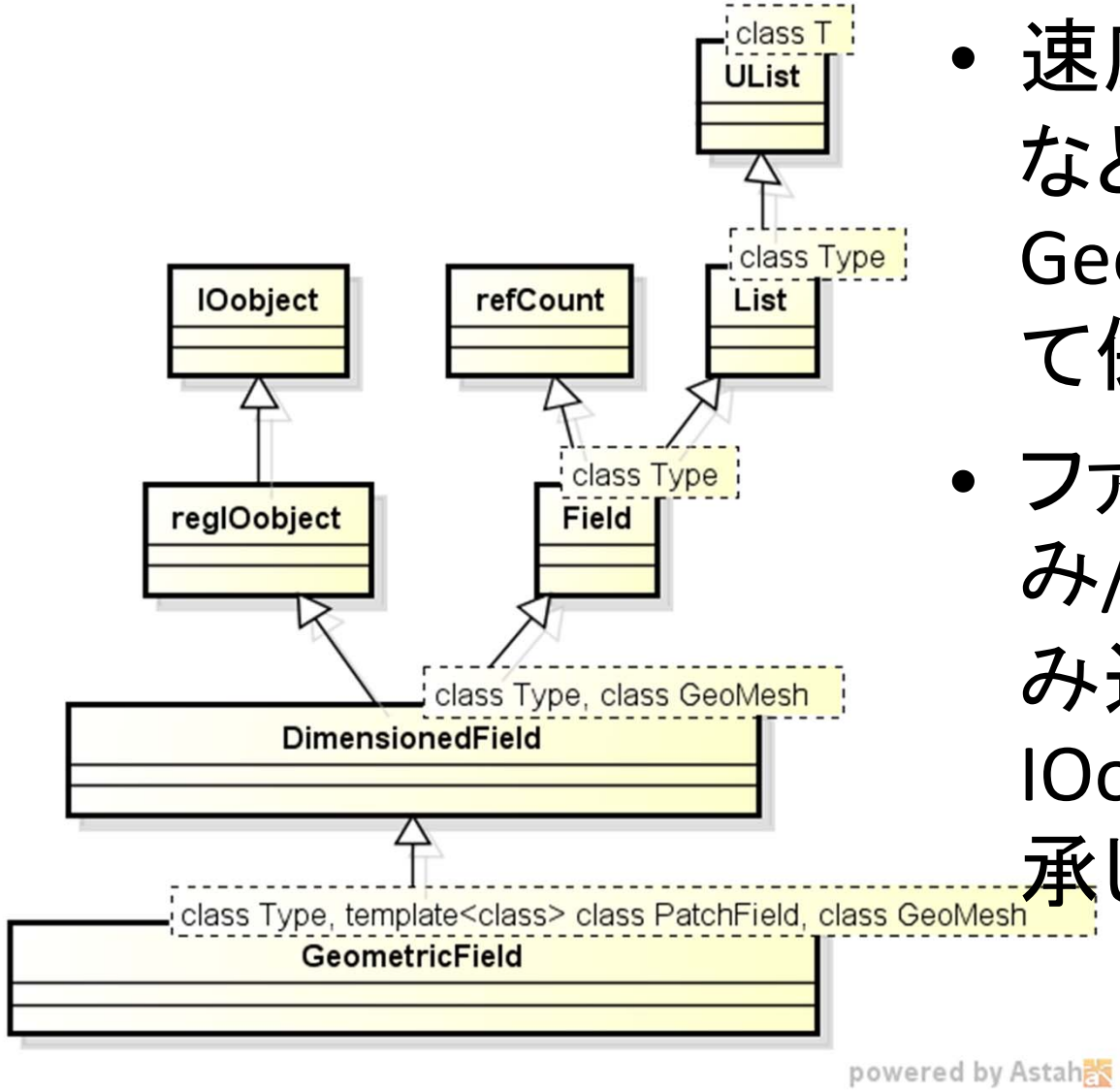
- 複数のクラスに共通する仕組みは，抽象化した独立クラス(親クラス)とする
- 複数のクラス(子クラス)が，独立させたクラス(親クラス)を継承することで，共通の機能を実現する

例

- 親クラス＝人間
- 子クラス＝学生，社会人



# GeometricFieldクラスの継承関係




- 速度U，圧力p，温度Tなどのデータは，GeometricField型として保存されている。
- ファイルへの書き込み/ファイルからの読み込みなどは，IOobjectクラスから継承している。

# ソルバ

- 特定の問題を解くために，OpenFOAMのコードを組み合わせたプログラム。
- ソルバのソースコードは，とてもシンプル。難しい作業は，部品に任せる。
- まとまった作業は別ファイルに記述し，includeすることで，読みやすさを保つ。
- 様々な部品（オブジェクト）が，協調しながら，目的を果たす。
- 部品どうしは，適切な独立性を持っている。
- ソルバのソースコードは，**ソルバ名.C** となっている。

# ライブラリ

- 特定に機能を実現するのに必要な部品を集めたもの。
- 関連する複数のクラスから，1つのライブラリを作成する。
- 例えば，imcompressibleTransportModelsライブラリの中に，viscosityModelクラスや，BirdCarreauクラス etc. が含まれている。
- srcディレクトリ内で，Makeディレクトリが存在すれば，ライブラリ。

# OpenFOAMインストール方法とファイルの格納場所

# インストール方法

- Ubuntu Deb Pack
  - Ubuntu OS のパッケージ管理システムを使う。バージョンの組み合わせに制限あり。
  - 最も簡単・早い方法。コンパイル不要。
- Source Pack
  - 汎用的方法。ソースコードをコンパイルする。
- Git Repository
  - 汎用的方法。ソースコードの最新版を取得してコンパイルする。
  - リリース後のバグフィックスが受けられる。
  - バージョンコントロールシステム GIT を利用する。

# インストール方法とファイルの場所

## 最新版 2.4.0 の場合

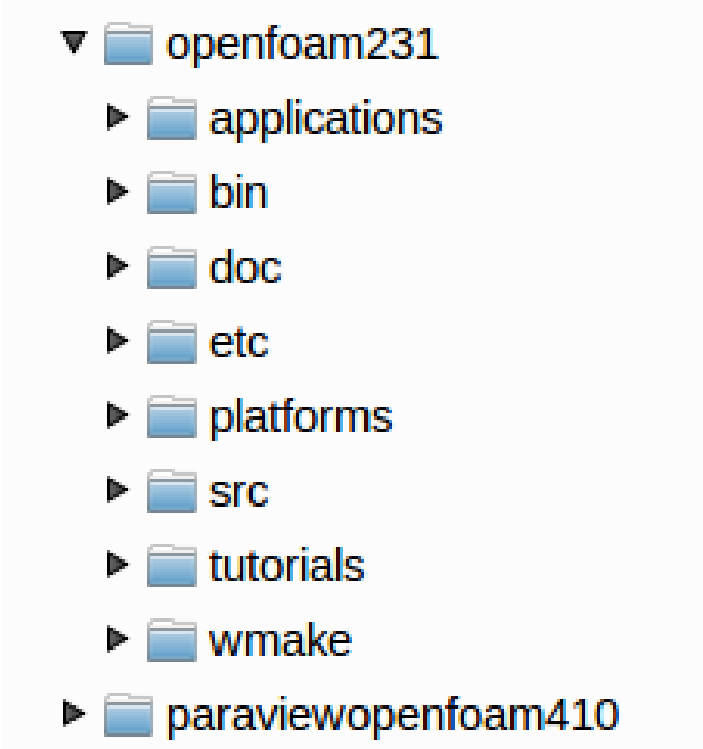
- Ubuntu Deb Pack
  - 標準インストール先　/opt/openfoam240
  - 標準作業ディレクトリ　$HOME/OpenFOAM/*user*-2.4.0
- Source Pack
  - 標準インストール先　$HOME/OpenFOAM/OpenFOAM-2.4.0
  - 標準作業ディレクトリ　$HOME/OpenFOAM/*user*-2.4.0
- Git Repository
  - 標準インストール先　$HOME/OpenFOAM/OpenFOAM-2.4.x
  - 標準作業ディレクトリ　$HOME/OpenFOAM/*user*-2.4.x

# そもそも，インストールとは？

- ファイルの配置
  - コンピュータが理解できる言葉で書かれた実行ファイル
- 実行に必要な情報のマシンへの通知
  - どこに，どのファイルがあるのか。
    → パスの設定
  - どのような条件・設定で動作するのか。
    → 環境変数の設定
  - OpenFOAMでは，インストールディレクトリ/etc/bashrcファイルを読み込むことで設定している

# OpenFOAMソースコードの構造

# ソースコード　ディレクトリ構造



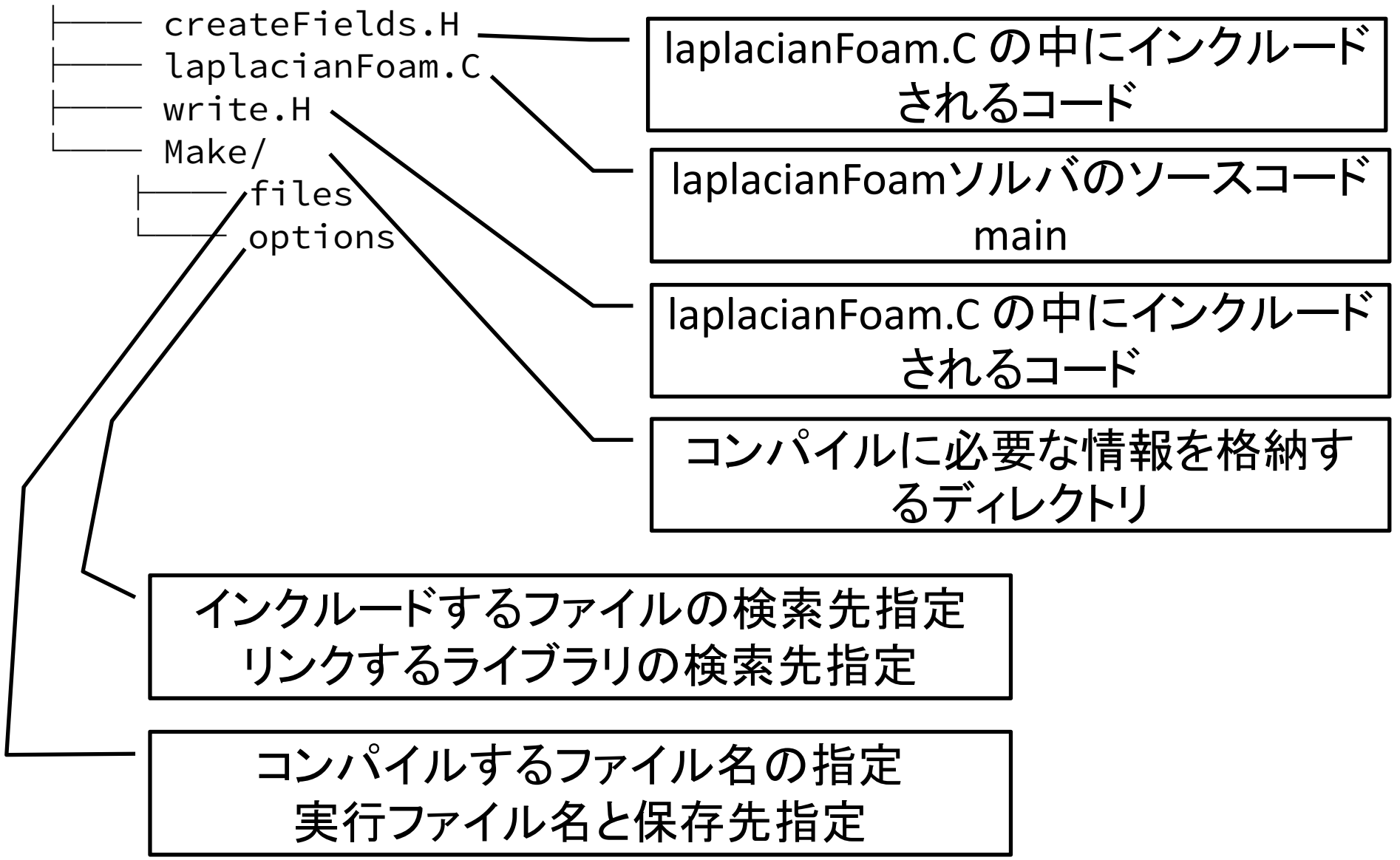

OpenFOAMソースコードの主な要素

- **src**: the core OpenFOAM source code
- **applications**: collections of library functionality wrapped up into applications, such as solvers and utilities
- **tutorials**: a suite of test cases that highlight a broad cross-section of OpenFOAM's capabilities
- **doc**: supporting documentation

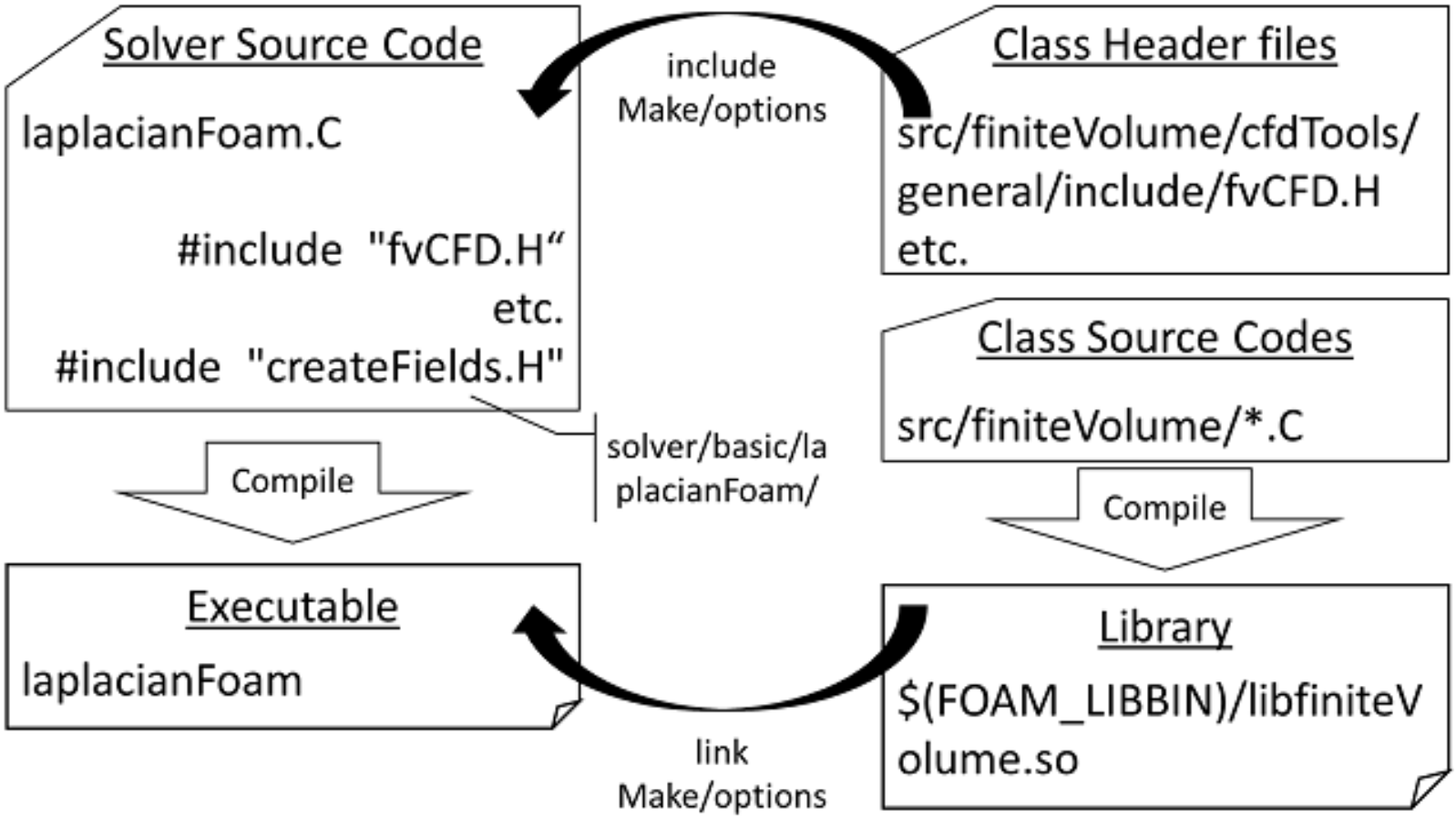
# applications/solvers 例

```
├── basic
│   ├── laplacianFoam
│   ├── potentialFoam
│   └── scalarTransportFoam
├── compressible
│   ├── rhoCentralFoam
│   ├── rhoPimpleFoam
│   ├── rhoSimpleFoam
│   └── sonicFoam
├── heatTransfer
│   ├── buoyantBoussinesqPimpleFoam
│   ├── buoyantBoussinesqSimpleFoam
│   ├── buoyantPimpleFoam
│   ├── buoyantSimpleFoam
│   ├── chtMultiRegionFoam
│   └── thermoFoam
├── incompressible
│   ├── adjointShapeOptimizationFoam
│   ├── boundaryFoam
│   ├── icoFoam
│   ├── nonNewtonianIcoFoam
│   ├── pimpleFoam
│   ├── pisoFoam
│   ├── shallowWaterFoam
│   └── simpleFoam
├── lagrangian
│   ├── coalChemistryFoam
│   ├── DPMFoam
│   ├── icoUncoupledKinematicParcelFoam
│   ├── reactingParcelFilmFoam
│   ├── reactingParcelFoam
│   ├── sprayFoam
│   └── uncoupledKinematicParcelFoam
└── multiphase
    ├── cavitatingFoam
    ├── compressibleInterFoam
    ├── compressibleMultiphaseInterFoam
    ├── driftFluxFoam
    ├── interFoam
    ├── interPhaseChangeFoam
    ├── multiphaseEulerFoam
    ├── multiphaseInterFoam
    ├── potentialFreeSurfaceFoam
    ├── twoLiquidMixingFoam
    └── twoPhaseEulerFoam
```

# applications/solvers/basic/laplacian



# ソルバのコンパイル　　ライブラリのコンパイル



# src/transportModels 例

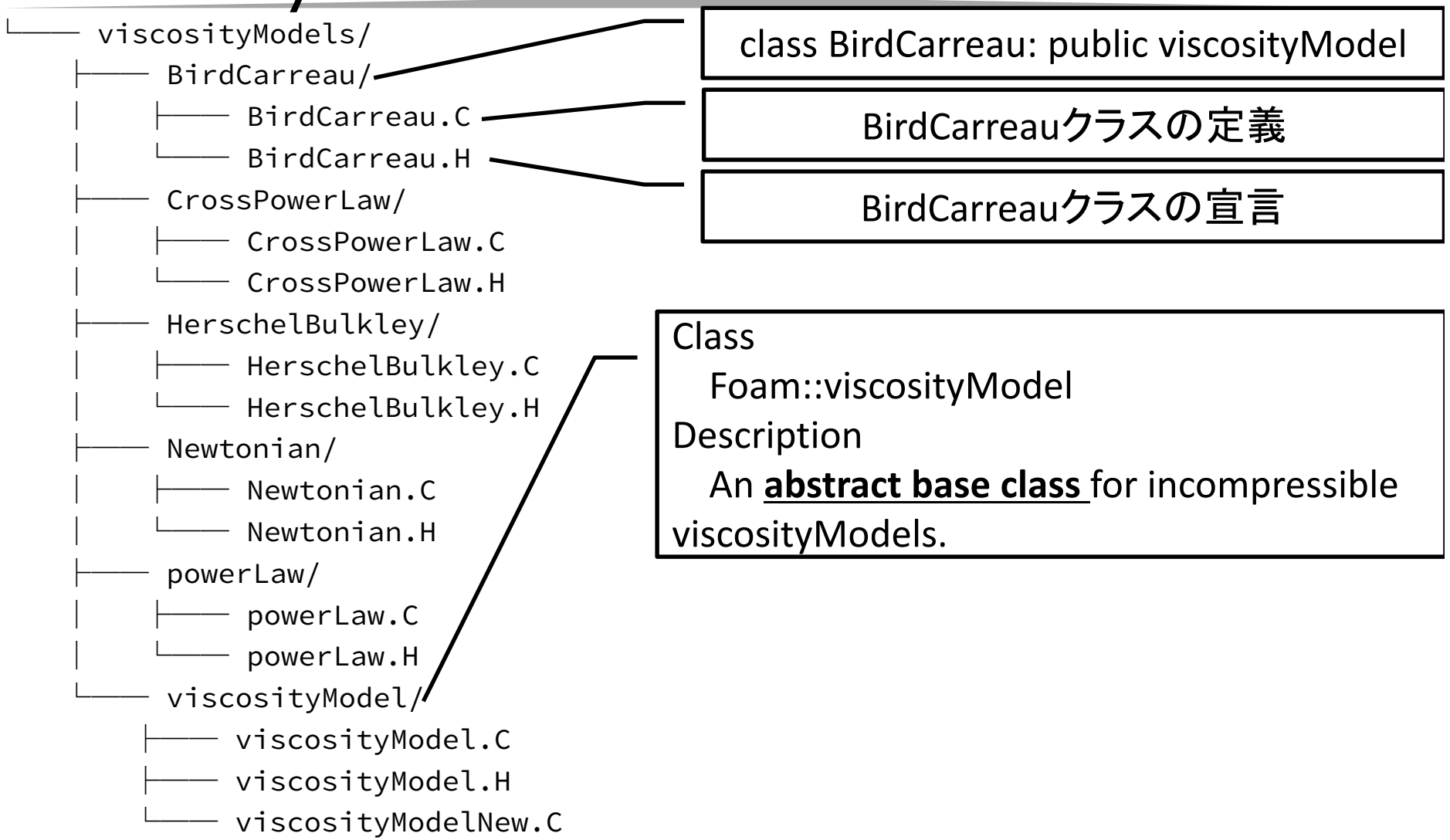
```
├── Allwmake*
├── compressible/
│   ├── compressibleTransportModel/
│   ├── lnInclude/
│   └── Make/
├── immiscibleIncompressibleTwoPhaseMixture/
│   ├── immiscibleIncompressibleTwoPhaseMixture.C
│   ├── immiscibleIncompressibleTwoPhaseMixture.H
│   ├── lnInclude/
│   └── Make/
├── incompressible/
│   ├── incompressibleTwoPhaseMixture/
│   ├── lnInclude/
│   ├── Make/
│   ├── singlePhaseTransportModel/
│   ├── transportModel/
│   └── viscosityModels/
├── interfaceProperties/
│   ├── interfaceCompression/
│   ├── interfaceProperties.C
│   ├── interfaceProperties.H
│   ├── lnInclude/
│   └── Make/
├── twoPhaseMixture/
│   ├── lnInclude/
│   ├── Make/
│   └── twoPhaseMixture/
└── twoPhaseProperties/
    ├── alphaContactAngle/
    ├── alphaFixedPressure/
    ├── lnInclude/
    └── Make/
```

Makeディレクトリが存在
コンパイルする単位：ライブラリ

incompressibleTransportModelsライブラリのディレクトリ(Make/files参照)

viscosityMedelクラスと，その派生クラスが格納される

# src/transportModels/ incompressible/ viscosityModels



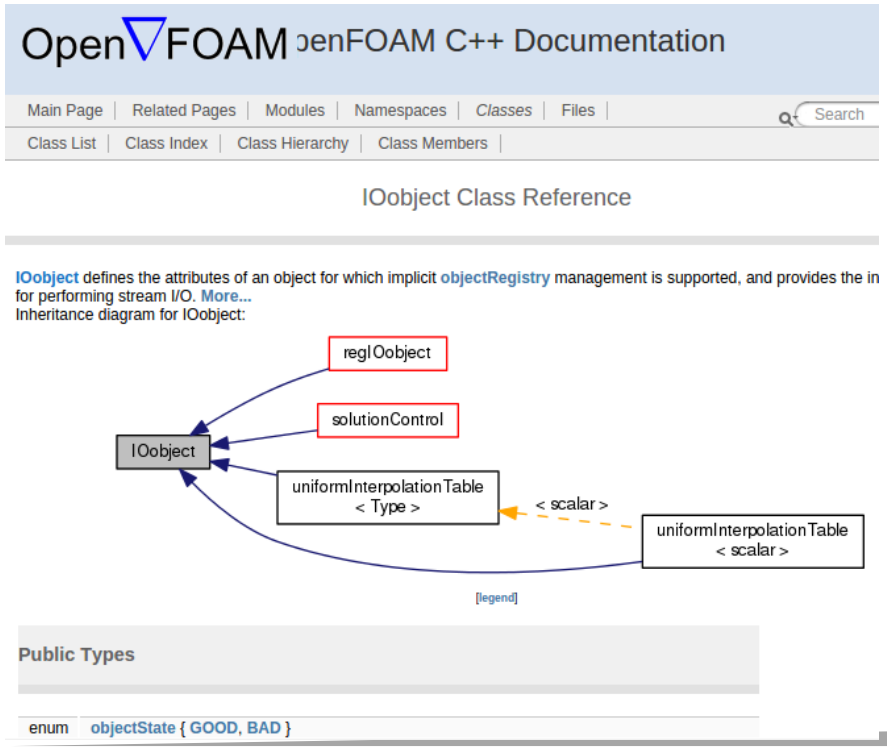
# OpenFOAMソースコードの調べ方

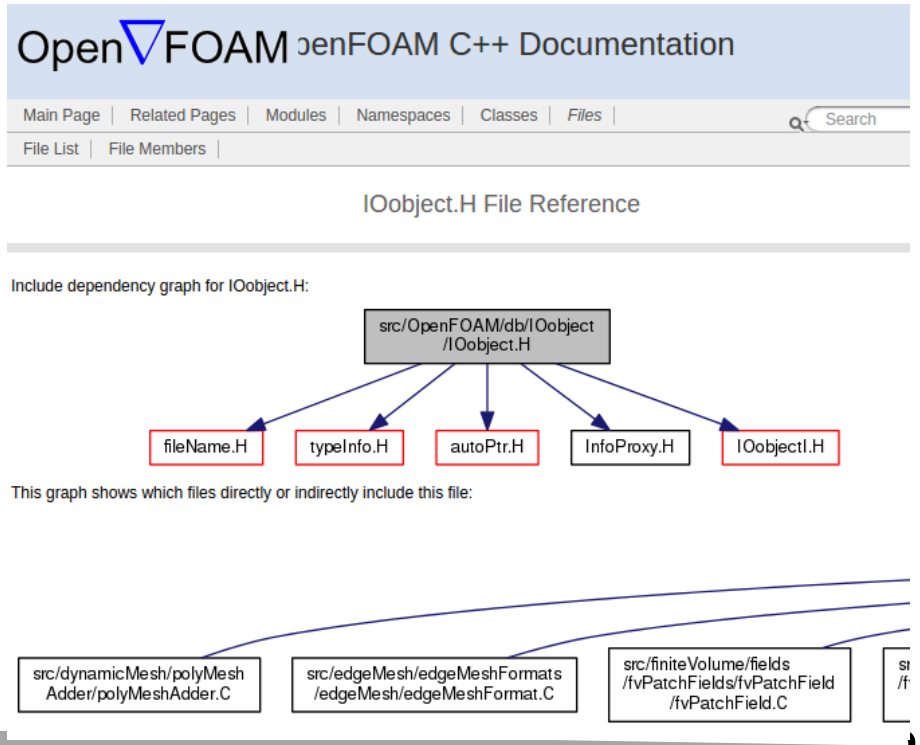
# 調べ方

- OpenFOAM C++ Source Guide
  - コード自体から，Doxygen を使って生成
  - www.openfoam.org/docs/cpp
    - （注意）上記サイトは最新リリース版のみ
  - クラス説明，コード説明，継承関係図など
  - ローカルマシン上でも，Doxygenを使って，同じ情報を作成できる
    - 注意：大量のファイルが生成され，2GB以上の容量が必要。
      - sudo apt-get install doxygen graphviz
      - cd $WM_PROJECT_DIR/doc/Doxygen
      - ./Allwmake

# OpenFOAM C++ SourceGuide 注意

- Class Reference と File Reference では，表示される内容が異なる
- クラスの全容を知りたい時には， Class Reference を参照するとよい。



# ソースコードを読み解くために

- 変数のタイプ（クラス）を意識
  - volScalarField, dimensionedScalar？など
- まずは，宣言(*.H)を見て，流れをつかむ。
  - 関数は，何(クラス)を受け取り，何を返すか？
- Slow and steady wins the race
  - 少しずつ，理解を深める
  - 小さな部分の積み重ね
  - 繰り返す，繰り返す，繰り返す
- 基礎を学習
  - 現象・式とソースの両方を学ぶ
- 一般的なデザインパターンの理解を深める

# 調べ方

- デバッガを使用する。
  - HowTo debugging

    http://openfoamwiki.net/index.php/HowTo_debugging

- debugSwitchesを利用して，実行時に追加メッセージを表示させる。

- Linuxのfindコマンドを使って，ファイルを探す。

# デバッグ情報

- 実行時にプログラムの動作状況を把握するため，標準出力へ，各種情報を出力する。
- どの情報を出力する/しないは，DebugSwitches でコントロールすることが可能である。
- システムの設定は，OpenFOAM/OpenFOAM-2.3.x/etc/controlDict 内に記述してある。
  - $HOME/.OpenFOAM/$WM_PROJECT_VERSIONに，上記controlDictを複製し，修正することで，ユーザ毎に設定を変更することができる。
  - OpenFOAM 2.2.0 以降では，ケースのcontrolDict内にDebugSwitchesを記載することで，個別に設定することも可。

    http://www.openfoam.org/version2.2.0/runtime-control.php
- foamDebugSwitchesユーティリティを実行すると，登録されているDebugSwitchesが表示される。（全てが有効な情報を書き出すとは限らない。）

# 参考資料

- OpenFOAM ProgrammersGuide, UserGuide
- ソースコード
- Imperial College 博士論文など
  - Hrvoje Jasak, Henrik Rusche, Franjo Juretic などなど
  - http://powerlab.fsb.hr/ped/kturbo/OpenFOAM/docs/
- PENGUINITIS!
  - http://www.geocities.jp/penguinitis2002/index.html
- http://openfoamwiki.net/
- http://www.cfd-online.com

http://eddy.pu-toyama.ac.jp/bbtb56fof-126/#_126

# solve ( A-B); なのか,
# solve ( A==B); なのか？

laplacianFoam.C では,

```
solve
(
    fvm::ddt(T) - fvm::laplacian(DT, T)
);
```

と表記してありますが、

```
solve
(
    fvm::ddt(T) == fvm::laplacian(DT, T)
);
```

としても良いのですか？

上記2つの表記は，どちらでも良いです。同じ意味になります。

　これには，演算子”==”のオーバーロードが関係しています。

　ソースコードを見ながら確認します。

　演算子は，どのクラスに対しての演算子なのか，に注意して調べる必要があります。

　今の質問では，fvm::ddt(T) と fvm::laplacian(DT, T) との演算です。両者のクラスは何になるでしょう？

fvmLaplacian.Cを見る。

fvm::laplacian(DT, T) の戻り値＝ tmp<fvMatrix<Type> > 型

このTypeは，laplacianメソッドに渡したＴと同じ。

```
238 template<class Type, class GType>
239 tmp<fvMatrix<Type> >
240 laplacian
241 (
242     const GeometricField<GType, fvPatchField, volMesh>& gamma,
243     const GeometricField<Type, fvPatchField, volMesh>& vf
244 )
```

　ということは，fvm::ddt(T) == fvm::laplacian(DT, T) では，tmp<fvMatrix<Type> >に対しての演算子”==”を確認する必要があります。

fvMatrix.H にGlobal operators としてoperator==が記載されています。

```
615 template<class Type>
616 tmp<fvMatrix<Type> > operator==
617 (
618     const tmp<fvMatrix<Type> >&,
619     const tmp<fvMatrix<Type> >&
620 );
```

fvMatrix.C に定義がある。

演算子“==”の左側 tmp<fvMatrix<Type> > をtAという名前で，右側 tmp<fvMatrix<Type> > をtBという名前で受取る。

演算結果として，return の後にあるもの（tA - tB）が戻される。

つまり，左辺ー右辺の行列（tmp<fvMatrix<Type> > 型）を戻すことになる。

```
1495 template<class Type>
1496 Foam::tmp<Foam::fvMatrix<Type> > Foam::operator==
1497 (
1498     const tmp<fvMatrix<Type> >& tA,
1499     const tmp<fvMatrix<Type> >& tB
1500 )
1501 {
1502     checkMethod(tA(), tB(), "==");
1503     return (tA - tB);
1504 }
```

なお，ここで使った演算子”-”自体も，fvMatrixクラス用にオーバーロードしたものが使われています。オリジナルのコードで使われているのと同じです。こちらは直感的にわかるので，疑問に感じないかもしれません。

（参考）
オペレータ(演算子)のオーバーロードについて。
http://homepage2.nifty.com/well/Operator.html
http://www.geocities.jp/ky_webid/cpp/language/017.html
http://www.geocities.jp/penguinitis2002/study/OpenFOAM/OpenFOAM-cpp_primer.html