Sphere

Skeleton for PHysical and Engineering REsearch

Ver 2.0.0

フロークラスマニュアル

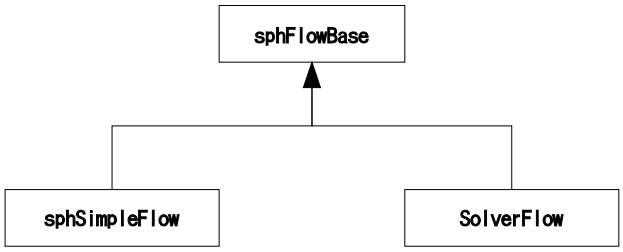
2010 年 8 月 01 日

目次

１．概要

SPHERE フレームワークは複数のソルバ、カプラをサポートしています。
その複数のソルバ、カプラの実行メソッドを呼び出すのがフロークラスです。
フロークラスをソルバに実装することにより柔軟なソルバの実行が可能となります。

１．１　クラス構成

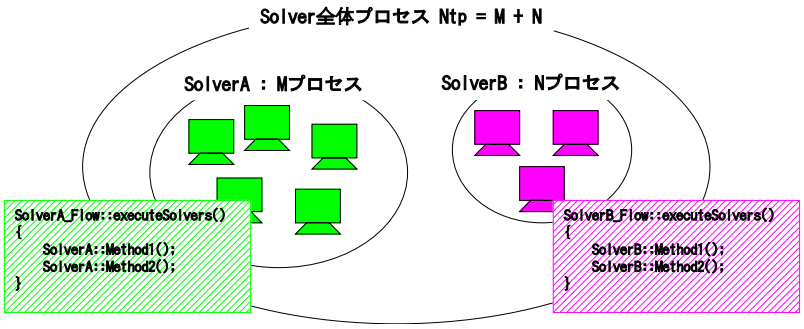


クラス名	説明
sphFlowBase	フロークラスの基底クラスです。
sphSimpleFlow	SPHERE フレームワーク側で提供するフローの実行クラスです。
SolverFlow	ソルバ側で実装するフロークラスです。 sphFlowBase を基底クラスとします。

１．２　フロークラスの生成

ソルバ側で実装するフロークラスは、ソルバ毎に別々のフロークラスを持つことができます。しかし１プロセス（ノード）に１つのフロークラスしか生成できません。

(1) 複数のフロークラス



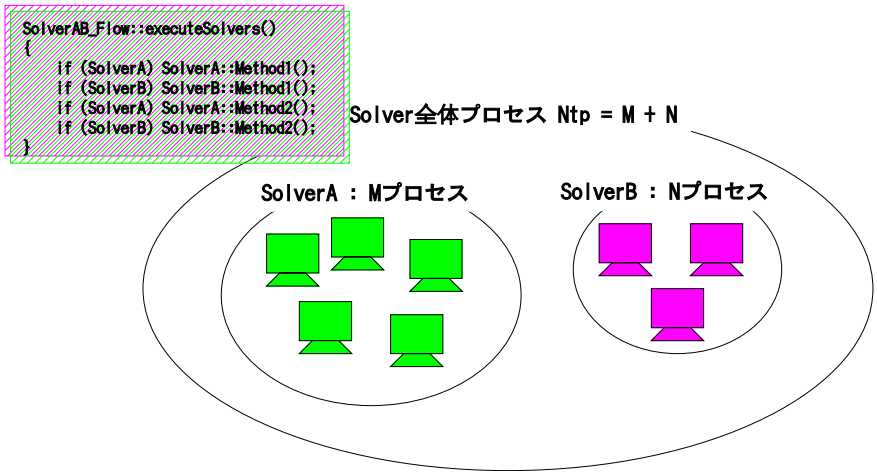
ソルバ全体プロセスが複数のソルバプロセスに重なり無く、１つのプロセスに１つのソルバプロセスが実行する形態です。

この場合、ソルバ毎にフロークラスを実装することができます。

実装されたフロークラスは自プロセスのソルバのメソッドを実装します。

コンフィグレーションにフロー要素をそれぞれのソルバ要素配下に定義する必要があります。

(2) １つのフロークラス



ソルバ全体プロセスが複数のソルバプロセスに重なり無く、１つのプロセスに１つのソルバプロセスが実行する形態です。

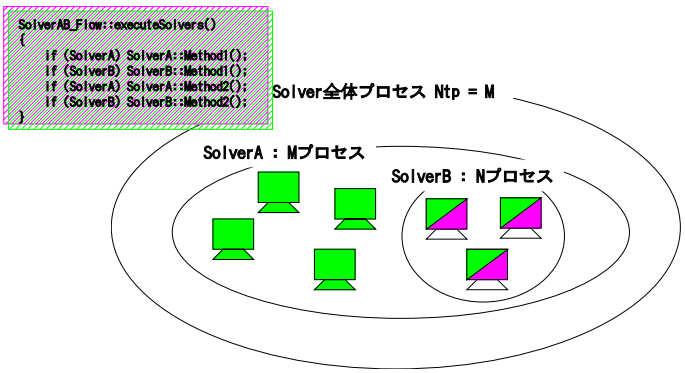
この場合、ソルバー全体で１つのフロークラスを持つことができます。

実装されたフロークラスは複数のソルバのメソッドを実装できますが、呼び出されるのは自プロセスのメソッドのみです。

自プロセスに実行メソッドのソルバが存在するかチェックしなければなりませんが、並列実行形態の変化に柔軟に対応できます。

コンフィグレーションにフロー要素を SphereConfig 要素に定義する必要があります。

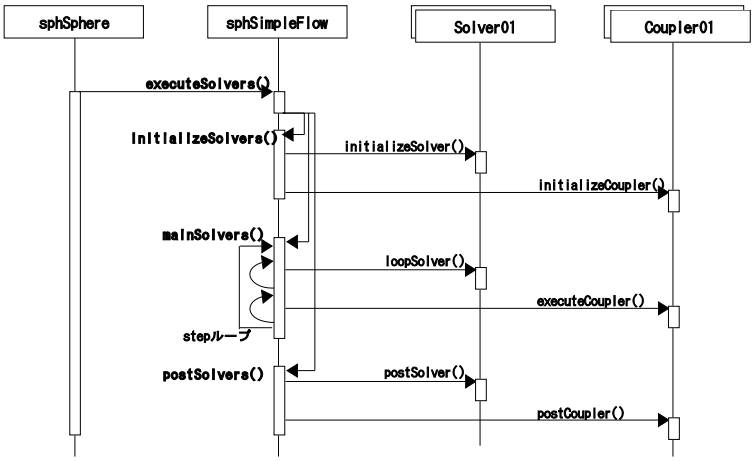
(3) １つだけのフロークラス



ソルバ全体プロセスが複数のソルバプロセスによって重なりを持ち、１つのプロセスに複数のソルバプロセスが実行する形態です。
この場合、ソルバー全体で１つのフロークラスしか持つことはできません。
複数のフロークラスを生成しようとするとエラーとなります。
コンフィグレーションにフロー要素を SphereConfig 要素に定義する必要があります。

１．３　フロークラスの実行シーケンス

デフォルトのフロー制御クラスである sphSimpleFlow クラスを例にソルバ、カプラの実行シーケンスを以下に説明します。



(1) executeSolvers()
SPHERE フレームワークの起動後呼び出されます。
"executeSolvers"メソッドでは以下のフロークラスの３つのメソッドを呼び出します。

a) initializeSolvers()
b) mainSolvers()
c) postSolvers()

(2) initializeSolvers()
自プロセスにインスタンスされているソルバ、カプラの初期化メソッドを順次呼び出します。

(3) mainSolvers()
ソルバのメインループを実行します。
インスタンスされているソルバの"loopSolver"、カプラの"executeCoupler"を順次呼び出します。
コンフィグレーションで定義されたループ回数分繰り返します。

(4) postSolvers()
ソルバ、カプラの破棄メソッドを順次呼び出します。

２．コンフィグレーション

SPHERE ではコンフィグレーションにフロー要素を定義することにより、ソルバのフロークラスをインスタンスします。
フロー要素にはステップ数、時間を定義します。
以下、コンフィグレーションのフロー要素について説明します。
詳細については「コンフィグレーション文法マニュアル」を参照してください。

２．１　SphFlow 要素（プログラムフローの指定）

記述ルール

要素名	SphFlow	
用　途	プログラムフローを指定する。 この要素がない場合、シンプルフローとなる	
属　　性		
属性名	値	種別
label	プログラムフローの識別ラベル	必須
id	プログラムフローの識別番号	必須
class	インスタンスフロークラス名 以下のクラス名は、デフォルトの予約クラス名とする。 "sph_simple_voxel"：SimpleVoxel タイプのデフォルトフローを行う。	必須
内包可能な要素（子要素として取り得る要素）		

要素名	意味	種別
SphStepTime	計算ステップ数を指定する。	任意

フロー要素のフロークラス名、識別ラベル、識別番号によりソルバ側でフロークラスを生成します。

フロー要素は、ソルバ全体で１つのフロークラスを用いる場合は SphereConfig 要素配下に定義します。

ソルバ毎に別々のフロークラスを用いる場合はソルバ要素配下に定義します。

（ソルバ全体で１つのフロークラスの定義）

```
<SphereConfig version="2.0.0">
    <SphFlow label="flow01 " id=1>
        < SphStepTime    max_step="10000"/>
    </SphFlow>
    <SphSolverList>
        <SphSolver id="1" label="SOLVER01" class="SOLVER01 " />
    </SphSolverList>
</SphereConfig>
```

（ソルバ毎にフロークラスの定義）

```
<SphereConfig version="2.0.0">
    <SphSolverList>
        <SphSolver id="1" label="SOLVER01" class="SOLVER01 " >
            <SphFlow label="flow01 " id=1>
                < SphStepTime    max_step="10000"/>
            </SphFlow>
        </SphSolver>
        <SphSolver id="1" label="SOLVER02" class="SOLVER02 " >
            <SphFlow label="flow02 " id=2>
                < SphStepTime    max_step="10000"/>
            </SphFlow>
        </SphSolver>
    </SphSolverList>
</SphereConfig>
```

２．２　SphStepTime 要素（計算ステップ、時間設定）

記述ルール

要素名	SphStepTime	
用　途	計算ステップ数、時刻を定義する。	
属　　性		
属性名	値	種別
max_step	計算ステップ数（整数） デフォルト＝1000	任意
base_step	基準ステップ デフォルト＝0	任意
base_time	基準時間 デフォルト＝0.0	任意
delta_time	１ステップ時間 デフォルト＝0.0	任意
内包可能な要素（子要素として取り得る要素）		
要素名	意味	種別
なし		

３．ソルバのフロークラス実装

３．１　フロークラスの生成クラス・メソッド

フロークラスはソルバ側に実装された sphFlowFactory クラスによって生成されます。

sphFlowBase* sphFlowFactory::factoryFlow(const char* class_name, const char* label, int id) const		
フロークラスの生成を行う		
引数	class_name	フロークラス名
	label	フロー識別ラベル
	id	フロー識別 ID
戻り値	生成カプラ	

フロークラス名、フロー識別ラベル、フロー識別 ID はコンフィグレーションに定義されたインスタンスフロークラス名、識別名（文字列）、識別 ID（整数）です。

フロークラス名によってソルバ側に実装しているフロークラスを判断し、フロークラスの生成を行います。

（使用例）

```
sphFlowBase*
sphFlowFactory::factoryFlow(const char* class_name, const char* label, int id) const
{
  sphFlowBase* flowObj = NULL;
  if (class_name != NULL) {
    if (strcasecmp(class_name, "FLOW01") == 0) flowObj = new sphFlow01(label, id);
  }
  // create default flow
  if (flowObj == NULL) flowObj = sphFlowFactoryBase::factoryFlow(class_name, label, id);
  return flowObj;
}
```

３．２　フロークラスのテンプレート

フロークラスは以下のテンプレートをソルバ側に実装する必要があります。

(1) フロークラスヘッダーファイル（ファイル名：sphFlow01.h）

```
#ifndef _SPHFLOW01_H_
#define _SPHFLOW01_H_
#include "solverbase/sphSimpleFlow.h"

class sphFlow01: public sphSimpleFlow
{
protected:
  virtual int initializeSolvers();
  virtual int mainSolvers();
  virtual int postSolvers();
public:
  sphFlow01();
  sphFlow01(const char* label, int id);
  virtual ~sphFlow01();
  virtual int initializeFlow(const sphElem* config_elem);
```

```
  virtual int initializeFlow(const SklCfg::SklSolverConfig* sklconfig);
  virtual int executeSolvers();
};
#endif /* _SPHFLOW01_H_ */
```

(2) フロークラスソースファイル（ファイル名：sphFlow01.C）

```
#include "sphFlow01.h"

sphFlow01::sphFlow01(): sphSimpleFlow()
{
}
sphFlow01::sphFlow01(const char* label, int id) : sphSimpleFlow(label, id)
{
}

sphFlow01::~sphFlow01()
{
}

int sphFlow01::initializeFlow(const sphElem* config_elem)
{
  int ret = sphSimpleFlow::initializeFlow(config_elem);
  return ret;
}

int sphFlow01::initializeFlow(const SklCfg::SklSolverConfig* sklconfig)
{
  int ret = sphSimpleFlow::initializeFlow(sklconfig);
  return ret;
}

int sphFlow01::executeSolvers()
{
  int ret = sphSimpleFlow::executeSolvers();
  return ret;
```

```
}

int sphFlow01::initializeSolvers()
{
  int ret = sphSimpleFlow::initializeSolvers();
  return ret;
}

int sphFlow01::mainSolvers()
{
  int ret = sphSimpleFlow::mainSolvers();
  return ret;
}

int sphFlow01::postSolvers()
{
  int ret = sphSimpleFlow::postSolvers();
  return ret;
}
```

以下、テンプレートに実装されたメソッドについて説明します。

３．２．１　コンストラクタ

sphFlow01(); sphFlow01(const char* label, int id);		
フロークラスの初期化を行う		
引数	label	フロー識別ラベル
	id	フロー識別 ID
戻り値	なし	

フロークラスの初期化を行います。
コンストラクタにて必ず基底フロークラスのコンストラクタを呼び出してください。

３．２．２　デストラクタ

virtual ~sphFlow01();	
フロークラスの破棄処理を行う	
引数	なし
戻り値	なし

フロークラスの破棄処理を行います。

３．２．３　フロー初期化

virtual int initializeFlow(const sphElem* config_elem); virtual int initializeFlow(const SklCfg::SklSolverConfig* sklconfig);	
フローの初期化を行う	
引数	config_elem　　フロー要素（Sphere V200） sklconfig　　コンフィグレーションオブジェクト（Vsphere）
戻り値	-1:error, 0:forced terminated, 1:normality

フローの初期化を行います。
コンフィグレーションが SPHERE(V200)形式と VSPHERE 形式にて呼び出されるメソッドが異なります。
SPHERE(V200)形式の場合、実行フローのフロー要素が渡されます。
１つのフローにて実行する場合は、SphereConfig 要素配下のフロー要素、ソルバ別のフローにて実行する場合は、ソルバ要素配下のフロー要素となります。

３．２．４　フロー実行

virtual int executeSolvers();	
フローの実行を行う	
引数	なし
戻り値	-1:error, 0:forced terminated, 1:normality

フローの実行を行います。
sphere の起動直後に最初に呼び出されるメソッドです。

３．２．５　ソルバ初期化

virtual int initializeSolvers();	
ソルバ、カプラの初期化メソッドを呼び出します。	
引数	なし
戻り値	-1:error, 0:forced terminated, 1:normality

３．２．６　ソルバ実行

virtual int mainSolvers();	
ソルバ、カプラの実行メソッドを呼び出します。	
引数	なし
戻り値	-1:error, 0:forced terminated, 1:normality

３．２．７　ソルバ破棄

virtual int postSolvers();	
ソルバ、カプラの破棄メソッドを呼び出します。	
引数	なし
戻り値	-1:error, 0:forced terminated, 1:normality

４．ステップクラス

ステップクラスは実行最大ステップ数、現在ステップ数、ステップ時間を管理するクラスです。

コンフィグレーションの SphStepTime 要素から設定値を取得してステップクラスに設定します。（sphFlowBase::initializeFlow(config_elem);を呼び出す必要があります。）

フロークラスからステップクラスを取得メソッドは以下です。

sphStepTime* sphFlowBase::getStepTime() const;	
ステップ・時間を管理しているステップクラスを取得します。	
引数	なし

戻り値	ステップクラス

以下、ステップクラスのメソッドについて説明します。

（ステップクラスメソッド一覧）

No	メソッド名	説明
1	size_t getCurrentStep() const void setCurrentStep(size_t currentStep)	実行中のループステップ数を取得・設定する。
2	size_t getBaseStep() const bool setBaseStep(size_t baseStep)	基準ステップ数を取得・設定する。
3	size_t getTotalStep() const	基準ステップからの実行ステップ数を取得する。
4	size_t getMaxStep() const bool setMaxStep(size_t max_step)	実行最大ステップ数を取得・設定する。
5	double getCurrentTime() const bool setCurrentTime(double currentTime)	現在時間を取得・設定する。
6	double getBaseTime() const bool setBaseTime(double baseTime)	基準時間を取得・設定する。
7	double getDeltaTime() const bool setDeltaTime(double deltaTime)	1 ステップ時間を取得・設定する。
8	bool incrementTime()	現在時間に 1 ステップ時間を加算する。
9	bool chkInterval(size_t interval) const static bool chkInterval(size_t currentStep, size_t interval)	現在ステップ数からインターバルチェックを行う.

４．１　現在ステップ数

size_t getCurrentStep() const void setCurrentStep(size_t currentStep)	
実行中のループステップ数を取得・設定する。	
引数	currentStep　　設定実行ステップ数
戻り値	実行中のループステップ数

実行中のループステップ数を取得・設定します。

４．２　基準ステップ数

size_t getBaseStep() const bool setBaseStep(size_t baseStep)	
基準ステップ数を取得・設定する。	
引数	baseStep　　設定基準ステップ数
戻り値	取得基準ステップ数

基準ステップ数を取得します。
取得・設定する基準ステップ数は、コンフィグレーションの SphStepTime 要素の "base_step"属性値となります。

４．３　基準加算実行ステップ数

size_t getTotalStep() const	
基準ステップからの実行ステップ数を取得する。	
引数	なし
戻り値	実行ステップ数

現在ステップ数に基準ステップ数を加算したステップ数を取得します。

getTotalStep() = getCurrentStep() + getBaseStep()

４．４　実行最大ステップ数

size_t getMaxStep() const bool setMaxStep(size_t max_step)	
実行最大ステップ数を取得・設定する。	
引数	max_step　　設定実行最大ステップ数
戻り値	実行最大ステップ数／成否

実行最大ステップ数を取得・設定します。
取得・設定する実行最大ステップ数は、コンフィグレーションの SphStepTime 要素の

"max_step"属性値となります。

４．５　現在時間

double getCurrentTime() const bool setCurrentTime(double currentTime)	
現在時間を取得・設定する。	
引数	currentTime　　設定現在時間
戻り値	取得現在時間／成否

現在時間を取得・設定します。

４．６　基準時間

double getBaseTime() const bool setBaseTime(double baseTime)	
基準時間を取得・設定する。	
引数	なし
戻り値	成否

基準時間を取得・設定を取得します。
取得・設定する基準時間は、コンフィグレーションの SphStepTime 要素の"base_time"属性値となります。

４．７　１ステップ時間

double getDeltaTime() const bool setDeltaTime(double deltaTime)	
1 ステップ時間を取得・設定する。	
引数	deltaTime　　設定 1 ステップ時間
戻り値	取得 1 ステップ時間／成否

１ステップ時間を取得・設定を取得します。

取得・設定する 1 ステップ時間は、コンフィグレーションの SphStepTime 要素の "delta_time"属性値となります。

４．８　ステップ時間加算

bool incrementTime()	
現在時間に 1 ステップ時間を加算する。	
引数	なし
戻り値	成否

現在時間に 1 ステップ時間を加算します。

４．９　インターバルチェック

bool chkInterval(size_t interval) const static bool chkInterval(size_t currentStep, size_t interval)	
現在ステップ数からインターバルチェックを行う.	
引数	interval　　インターバルステップ数 currentStep　　現在ステップ数
戻り値	true:インターバル間隔

現在ステップ数がインターバルステップ数の間隔であるかチェックを行います。