TextParserの紹介

HPCI戦略分野４「次世代ものづくり」
東大生研　革新的シミュレーション研究センター
川鍋友宏
tkawanab@iis.u-tokyo.ac.jp

ToC

- TextParserのご紹介
  - TextParserとは？
  - 基本の記述法
  - 基本のAPIセット
  - 京でのインストール・ビルド・動作デモ
  - MPIでの利用方法
  - まとめ

TextParserとは？

- YAML的な記述方式のテキストファイルを読み・書きするライブラリプログラム
  - 初期化パラメータファイルなどの記述に利用
- オープンソース
  - 修正BSDライセンス
  - githubで公開中
    https://github.com/avr-aics-riken/TextParser

TextParser書式　サンプル

```
/* c,c++言語のようなコメントが書けます */

// 要素のグルーピングが出来ます
group1{
    param_str = "string"        // 右辺が文字列型の場合はダブルクォートで囲みます
    param_int = 1234
    param_float = 1.234e-9      // 指数表現も可能
}

group2{
    int_vec = ( 1, 3, 5, 7, 9, 11, 13 )     // 整数ベクトル
    days = ("Sun","Mon","Tue","Wed","Thu","Fri","Sat")  // 文字列ベクトル

    // 階層的なグルーピングが出来ます
    group3{
        // ラベル(左辺)に"[@]"を使うことで配列的な表現も可能
        planet[@] = "Mercury"
        planet[@] = "Venus"
        planet[@] = "Earth"
    }
}
```

TextParserの特徴

- 書式が簡単
  - 学習が容易
  - テキストエディタでの見通しが良い
    - 初見で直感的にデータ構造が把握可能
- ただし…
  - あまり複雑なことはできません
  - スキーマ定義/検証もできません
    - XMLにおけるDTD/XML Schemaのようなものはありません

なぜYAMLを使わないの？

- （意外と）仕様が大きいですよね…
  - 学習コストがかかります
- ライブラリのビルドが大変ですよね…
  - cmakeが必要、とか
  - yaml-cpp-0.5.x（最新版）はBoostが必要、とか…
- 構造表現が面倒ですよね…
  - インデントで階層構造を表現
    - インデントは空白スペース文字のみ利用可能
    - （ブロックスタイルの場合）
  - うっかりtabを打つとあとで悩むことに

なので、TextParserは

- 仕様が小さいです
  - シミュレータのパラメタファイルを記述するのが主目的
  - 基本の書式は10分あれば憶えられます
- ビルドが簡単です
  - 標準ライブラリ(libstdc++)だけでビルド出来ます
  - `./configure | make | make install` でOK
- 構造表現が簡単で見通しが良いです
  - {…}で囲んでグルーピング、ネストも出来る
  - インデントは自由です、tabもspaceも使えます
  - 記述法としてはYAMLよりjsonに近いかも

その他の特徴

- 開発言語：C++ (C++98)
- C, Fortran90用APIも提供
- 京を含むLinux,Windows,Mac OSXで利用可能
- singletonモデル（後述）
- MPI版もあります（後述）

- HPCI戦略分野４（東大生研）＋AICS可視化技術研究チームによる共同開発

書式の基本

- 構成要素
  1. ラベル
     - ノード
       - 階層構造の非末端ラベル
     - リーフ
       - 階層構造末端ラベル
  2. 値
     - 文字列
     - 数値（整数、実数）
- 例

```
root_node_label {
    mid_node_label {
        leaf_label = “value”
    }
}
```

ラベルの基本

- 利用可能文字
  - [a-zA-Z0-9_-]
- 同階層内ノード・リーフに同一のラベルは付けられない
- パース処理時には大文字/小文字の区別はしない

- エラーの例

```
foo {
    bar {
        baz = 1.0
        BAZ = 2.0 // error!
    }
}
```

値の基本

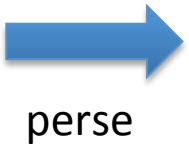
1. 文字列
   - 二重引用符(")で囲む
   - 利用可能文字は`[a-zA-Z0-9_-]`
   - 文字列比較で大文字/小文字の区別はしない
2. 整数
3. 実数
   - 指数表現が可能
     - `1.0e10`
     - `1.23E-10`
     - `.3D-4`

ラベル:配列形式ラベル

- ラベル文字列末尾に[@]をつける
  - 同一階層に重複したラベル文字列が書ける
- パースすると@が配列添字に置き換わる
  - プログラムからデータにアクセスする時は実添字を指定する
- 例

```
foo {
    bar[@] {
        baz[@] = 1.0
        baz[@] = 2.0
    }
    bar[@] {
        baz[@] = 3.0
        baz[@] = 4.0
    }
}
```

ファイル上での記述




```
foo {
    bar[0] {
        baz[0] = 1.0
        baz[1] = 2.0
    }
    bar[1] {
        baz[0] = 3.0
        baz[1] = 4.0
    }
}
```

オンメモリの情報

値：ベクトル

- 文字列もしくは数値の順序付けされたセット
- 例

```
(1, 2, 3, 4)
(-0.1E-5, 0.0, +0.1E-5 )
(“one”, “two”, “three” )
```

- ベクトル要素は全て同じ型の必要あり
- ベクトルのネストは出来ない

値：Numerical Limits

- limits.hで定義されているNumerical Limitsの各値が利用可能

```
CHAR_MIN, CHAR_MAX, SHORT_MIN,
SHORT_MAX, INT_MIN, INT_MAX,
LONG_MIN, LONG_MAX, LONGLONG_MIN,
LONGLONG_MAX, FLOAT_MIN, FLOAT_MAX,
DOUBLE_MIN, DOUBLE_MAX
```

- 例

```
int_val = INT_MAX
float_vec = ( 0.0, 3.14, FLOAT_MAX )
```

値：依存関係付き値

- C言語ライクな三項演算子による条件式により値を制御
- 書式

  ラベル = @dep(条件式)? 値1:値2

- 条件式の左辺はラベル、右辺はその取りうる値
- 条件式で利用できる論理演算子

  ==, !=, &&, ||

- パース時に条件式が評価され、値が決定
- 例

```
b = @dep( “a”==1 ) ? 1 : 2
flag3 = @dep((“flag1”==1)&&(“flag2”==“on”))? “on”:”off”
int_vec = @dep( “type”==“odd” )? (1,3,5,7) : (2,4,6,8)
```

書式の基本について、おわり

- 詳しい仕様はgithubのドキュメントをご覧ください

https://github.com/avr-aics-riken/TextParser

APIの基本(C++)

- TextParserインスタンスの取得

```
static TextParser*
TextParser::get_instance_singleton();
```

  - シングルトンインスタンスを取得
    - インスタンス未生成なら内部的にnewする
  - シングルトンパターンとは
    - プロセス内にそのクラスのインスタンスが１つしか生成されないことを保証する仕組み
  - ユーザプログラム側でインスタンスを保持する必要無し
    - TextPaserAPIを利用するシーンでこのメソッドを毎回呼べばよい

APIの基本(C++)

- ファイル読み込み

```
TextParserError
TextParser::read(const std::string& file);
```

　　file　入力ファイル名

　　戻り値　エラーコード(0:no error)

– ユーザプログラム側で最初に１度実行
– TextParser形式ファイルを読み込み、パースする

API(C++)

- 値の取得

```
TextParserError
TestParser::getValue(
        const std::string& label,
        std::string& value
);
```

label ラベルパス
value 値 （返却引数）
戻り値 エラーコード (0: no error)

– 実際の値の型に関わらず、文字列型として返却

APIの基本(C++)

- APIでのラベルパス指定方法
- 例：

```
foo {
   bar {
        baz = “string_value”
   }
}
```

  - bazのラベルパスは
    “/foo/bar/baz”
    と指定します

APIの基本(C++)

- 値の型を取得

```
TextParserValueType
TestParser::getType(
        const std::string& label,
        int *error
);
```

label ラベルパス
error エラーコード (0: no error)
戻り値 型のenum

APIの基本(C++)

- 返却値の型変換API

```
char TextParser::convertChar(const std::string value, int *error);
short TextParser::convertShort(const std::string value, int *error);
int TextParser::convertInt(const std::string value, int *error);
long TextParser::convertLong(const std::string value, int *error);
long long TextParser::convertLongLong(const std::string value, int
*error);
float TextParser::convertFloat(const std::string value, int *error);
double TextParser::convertDouble(const std::string value, int *error);
```

などを使って前述の`getValue()`, `getType()`で得た値の型変換を行います

APIの基本(C++)

- ベクトル型の要素分解API

```
TextParserError
TextParser::splitVector(
          const std::string& vector_value,
          std::vector<std::string>& velem
);
```

  - `vector_value` ベクトル型パラメータの値（文字列）
  - `velem` 各要素の値（文字列）
  - 戻り値 エラーコード（`0: no error`）

  – 文字列型として分解されるので、各要素を型変換APIを利用して変換

APIの基本(C++)

- 「値の型は予め分かっている」
  - そんな場合は、以下のAPI群を利用

```
bool getInspectedValue(const std::string label, int &ct );
bool getInspectedValue(const std::string label, float &ct );
bool getInspectedValue(const std::string label, double &ct );
bool getInspectedValue(const std::string label, std::string &ct );
```

`label`	ラベルパス
`ct`	変数返却用(出力引数)

API(C++)

- 「ベクトル値の型は予め分かっている」
  – そんな場合は、以下のAPI群を利用

```
bool getInspectedVector(const std::string label,
                        int *vec, const int nvec );
bool getInspectedVector(const std::string label,
                        float *vec, const int nvec );
bool getInspectedVector(const std::string label,
                        double *vec, const int nvec );
bool getInspectedVector(const std::string label,
                        string *vec, const int nvec );
```

label	ラベルパス
vec	ベクトル格納配列ポインタ(出力引数)
nvec	ベクトルサイズ

APIの基本(C言語)

- C++APIをラップして実装

```
TP_HANDLE tp_getInstanceSingleton();

int tp_read(TP_HANDLE tp_hand,char* file);

int tp_getValue(TP_HANDLE tp_hand,char* label,char* value);
int tp_getType(TP_HANDLE tp_hand,char* label, int *type);

char tp_convertChar(TP_HANDLE tp_hand,char* value, int *error);
short tp_convertShort(TP_HANDLE tp_hand,char* value, int *error);
int tp_convertInt(TP_HANDLE tp_hand,char* value, int *error);
long tp_convertLong(TP_HANDLE tp_hand,char* value, int *error);
long long tp_convertLongLong(TP_HANDLE tp_hand,char* value, int *error);
float tp_convertFloat(TP_HANDLE tp_hand,char* value, int *error);
double tp_convertDouble(TP_HANDLE tp_hand,char* value, int *error);
```

など。

- 現状、`getInspected`～系のAPIはありません。（ご要望あれば追加します）

APIの基本(FORTRAN90)

- C言語APIをラップして実装

```
integer TP_GET_INSTANCE_SINGLETON(Integer*8 ptr)

INTEGER TP_READ(INTEGER*8 ptr,CHARACTER(len=*) file)

INTEGER TP_GET_VALUE(INTEGER*8 ptr,CHARACTER(len=*) label,CHARACTER(len=*)
value)
INTEGER TP_GET_TYPE(INTEGER*8 ptr,CHARACTER(len=*) label,INTEGER type)

INTEGER*1 TP_CONVERT_CHAR(INTEGER*8 ptr,CHARACTER(len=*) value, INTEGER error)
INTEGER*1 TP_CONVERT_INT1(INTEGER*8 ptr,CHARACTER(len=*) value, INTEGER error)
```

– などなど（C言語APIと同様の機能セット）

MPI環境での利用

- configureオプション `-enable-mpi`を指定してビルド

```
TextParserError
TextParser::read(const std::string& file);
```

  - rank0がファイルを読み込み、他ランクへMPIで情報配信
  - 初期化ファイルをrank0にのみステージングすればよい

```
TextParserError
TextParser::read_local(const std::string& file);
```

  - 各ランクで独自にファイルを読み込む
  - ランクごとに異なるパラメタセットが持てる

- c言語、Fortran90APIでも同様

デモ

- TextParserを京でgit cloneしてビルドします
  - githubからTextParserリポジトリをcloneする

```
$git clone https://github.com/avr-aics-riken/TextParser
```

    - gitは/opt/local/binにあります
  - ビルド
    - configureオプション(INATALL.txt参照)

```
--prefix=$HOME/my_libs/ \
--host=sparc64-unknown-linux-gnu \
CXX=mpiFCCpx \
CXXFLAGS="-Kfast"
```

デモ
例題データ(sample.tp)

```
//sample.tp
// a sample parameter file of TextParser
root{
  foo{
    bar{
      int_flag=1
      str_flag="off"
    }
  }
  baz[@]{
    msg=@dep("/root/foo/bar/int_flag"==1)?
                "int_flag_is_1":"int_flag_is_not_1"
  }
  baz[@]{
    msg=@dep("/root/foo/bar/str_flag"=="on")?
                "str_flag_is_on":"str_flag_is_not_on"
  }
  baz[@]{
    msg=@dep(("/root/foo/bar/int_flag"==1)&&("/root/foo/bar/str_flag"=="on"))?
                "Both_are_on":"Both_are_not_on"
  }
}
```

デモ
プログラム（tpdemo.cpp 抜粋）


```
// MPI初期化
MPI_Init(&argc,&argv);
int my_rank=0;
MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);

// TestParserインスタンス取得
TextParser *tp=TextParser::get_instance_singleton();

// sample.tpファイルロード
TextParserError err=tp->read( "./sample.tp" );

// /root/baz[@]/msgの各値を表示
string msg1, msg2, msg3;
bool b1=tp->getInspectedValue( "/root/baz[0]/msg", msg1 );
bool b2=tp->getInspectedValue( "/root/baz[1]/msg", msg2 );
bool b3=tp->getInspectedValue( "/root/baz[2]/msg", msg3 );

cout << "rank:" << my_rank << ", msg1:" << msg1 << endl;
cout << "rank:" << my_rank << ", msg2:" << msg2 << endl;
cout << "rank:" << my_rank << ", msg3:" << msg3 << endl;

// あとしまつ
MPI_Finalize();
```

デモ

- テストプログラムを京でコンパイル・リンク
  - TextParserは$HOME/my_libsにインストールした前提

```
$ mpiFCCpx tpdemo.cpp \
  -I$HOME/my_libs/include \
  -L$HOME/my_libs/lib -lTPmpi
```

デモ
ジョブスクリプト（抜粋）

```
#!/bin/sh -x
#PJM --rsc-list "rscgrp=small"
#PJM --rsc-list "node=8"
#PJM --rsc-list "elapse=00:10:00"
#PJM --mpi "use-rankdir"
#PJM --stg-transfiles all
#PJM --stgin "rank=* ./a.out %r:./"
#PJM --stgin "rank=0 ./sample.tp 0:./"

. /work/system/Env_base
mpiexec ./a.out
```

rank0のみに sample.tpをステージ・イン

デモ
実行結果

```
klogin6$ cat job.sh.o2376020
Env_base: K-1.2.0-15
rank:0, msg1:int_flag_is_1
rank:0, msg2:str_flag_is_not_on
rank:0, msg3:Both_are_not_on
rank:2, msg1:int_flag_is_1
rank:2, msg2:str_flag_is_not_on
rank:2, msg3:Both_are_not_on
rank:6, msg1:int_flag_is_1
rank:6, msg2:str_flag_is_not_on
rank:6, msg3:Both_are_not_on
rank:3, msg1:int_flag_is_1
rank:3, msg2:str_flag_is_not_on
rank:3, msg3:Both_are_not_on
  .
  .
  .
```

rank0がsample.tpを読み込み、他ランクに情報配信するので、全ランクが同一の情報を保持していることがわかる

その他API

- 動的なパラメタの編集・追加・削除
- パラメタセットのファイル出力
- 通常の`new()`も可能
  - 複数のTextParserインスタンスを生成可能
- などなど

- 詳しくはドキュメントをご覧ください

まとめ

- パラメタファイルパーサライブラリ、TextParserのご紹介
  - コンパクトなパラメタ記述仕様
  - 軽量、高ポータビリティなライブラリ
  - オープンソース
  - githubで配布
    https://github.com/avr-aics-riken/TextParser
  - 沢山の方々にご利用いただければ幸いです
  - API追加リクエスト、改善ご提案、フィードバックを歓迎します