# Boostライブラリー周の旅

## Ver 1.54.0 ～ 1.58.0

高橋 晶(Akira Takahashi)
faithandbrave@longgate.co.jp

2015/05/30 Boost.勉強会 #17 東京

# はじめに

この発表は、以下のような方を対象にして、

- Boostに興味はあるけど、触ったことがない
- バージョンアップについていけなくなった
- Boostの全容を知りたい

Boost 1.58.0時点での、なるべく全てのライブラリの
概要を知ってもらうためのものです。

# Boostとは

- 標準ライブラリに満足できなかった人たちが作っている、C++の準標準ライブラリ。
- Boostから標準ライブラリに、多くの機能が採用されている
- 普段のプログラミング全般で使える基本的なものから、専門的なものまで、いろいろなライブラリがある。
- ライセンスはBoost Software License 1.0
  - 無償で商用利用可能
  - 著作権表記の必要なし
  - ソースコードの改変自由

# 本日紹介するライブラリ

- Log
- TTI
- Type Erasure
- Predef
- Align
- Type Index
- Endian
- Sort

# 質問は随時受け付けます

- この発表では、8ライブラリの紹介をします。
- 1ライブラリに付き、(ほぼ)ひとつのサンプルコードで解説する、というスタイルです。
- 5〜10分ほど発表時間が余るようにしてあるので、
  - 「ここがよくわからなかった」
  - 「こんなこともできる？」
  - 「ここはどうなっている」
  - といった疑問があれば、随時質問してください。

# Log

ロギングライブラリ。ログレベルの設定、フォーマット、ファイルサイズや日付によるローテーションなど。

```
using namespace boost::log;
add_file_log("log.txt");

// infoレベル以上を出力し、それ以外は捨てる
core::get()->set_filter(
   trivial::severity >= trivial::info
);

BOOST_LOG_TRIVIAL(debug) << "デバッグメッセージ";
BOOST_LOG_TRIVIAL(info)  << "情報メッセージ";
BOOST_LOG_TRIVIAL(error) << "エラーメッセージ";
BOOST_LOG_TRIVIAL(fatal) << "致命的なエラーメッセージ";
```

# TTI (Type Traits Introspection)

型がどんな情報を持っているかコンパイル時に調べるライブラリ。

```
// メンバ関数mf()を持っているか判定するメタ関数を生成する
BOOST_TTI_HAS_MEMBER_FUNCTION(mf)

struct X {
  int mf(int, int);
};

型Xが、intを2つ受け取り、intを返すメンバ関数mf()を持っているか
constexpr bool b = has_member_function_mf<
  X,
  int,
  boost::mpl::vector<int, int>
>::value;
```

# Type Erasure 1/2

コンセプトで実行時多相性を表現するライブラリ。

「できること」を列挙して設定したany型に、それが可能な

あらゆる型のオブジェクトを実行時に代入して使用できる。

```
using namespace boost::type_erasure;
any<
  boost::mpl::vector<
    copy_constructible<>, // コピー構築できること
    incrementable<>,      // インクリメントできること
    ostreamable<>         // ストリーム出力できること
  >
> x(10); // int値をコピー(要件を満たしている型ならなんでもOK)
++x;
std::cout << x << std::endl; // 「11」が出力される
```

# Type Erasure 2/2

コンセプトは、自分で定義できる。

```
// 1引数をとるpush_back()メンバ関数を持っていること、
// というコンセプトを定義する。
BOOST_TYPE_ERASURE_MEMBER((has_push_back), push_back, 1)

// intを引数にとり、voidを返すpush_backを持っている、
// コンテナへの参照を受け取って操作する
void append_many(any<has_push_back<void(int)>, _self&> container)
{
  for(int i = 0; i < 10; ++i)
    container.push_back(i);
}
```

# Predef

コンパイラ、アーキテクチャ、OS、標準ライブラリなどの情報を取得するライブラリ。
これらのマクロは必ず定義されるので、if文で使ってもいい。

```
// GCCかどうか
#if BOOST_COMP_GNUC
  #if BOOST_COMP_GNUC >= BOOST_VERSION_NUMBER(4,0,0)
    const char* the_compiler = "バージョン4以上のGNU GCC,"
  #else
    const char* the_compiler = "バージョン4未満のGNU GCC"
  #endif
#else
  const char* the_compiler = "GNU GCCではない"
#endif
```

# Align

アラインメント関係の情報を取得するメタ関数や
メモリアロケータなど。

```
//  16バイトアラインメントでメモリ確保するアロケータを、
// vectorで使用する
std::vector<
  char,
  boost::alignment::aligned_allocator<char, 16>
> v(100);
```

# Type Index

std::type_info／std::type_indexのBoost版。

RTTIの無効化や、型のデマングル名などに対応している。

```
using boost::typeindex::type_id;
std::cout << type_id<int>().raw_name() << std::endl;
std::cout << type_id<int>().pretty_name() << std::endl;
```

```
i
int
```

(GCCの場合)

# Endian

エンディアン操作のライブラリ。
エンディアン指定の算術型や、エンディアン変換の関数など。

```
using namespace boost::endian;

little_uint8_t a; // リトルエンディアンの8ビット符号なし整数型
big_uint8_t b;    // ビッグエンディアンの(以下略)

std::uint8_t x = 0;
std::ifstream file(…);
file.read(&x, sizeof(uint8_t));

x = big_to_native(x); // ビッグエンディアンのデータを、
                      // その環境のエンディアンに変換。
```

# Sort

範囲を並び替えるアルゴリズムのライブラリ。

巨大な配列をソートする際にstd::sort()よりも高速になる、

spreadsort()が含まれている。

```
using namespace boost::sort::spreadsort;

std::vector<int> v; // N >= 10,000
spreadsort(v.begin(), v.end());
```

この関数に指定できるのは、以下の型の範囲のみ：

- 整数型
- 浮動小数点数型
- 文字列型

# 最近のBoost事情 1/2

- 開発のリポジトリが、SubversionからGitHubに移行しました。github.com/boostorg
  - これは、ユーザーからの貢献を受け付けやすくするのが主な目的です。
- pull requestすれば取り込んでもらえます。

# 最近のBoost事情 2/2

- CMT (Community Maintenance Team)という制度が始まりました。
  - Boostの各ライブラリは、だいたい作者が一人でメンテナンスしています。
    作者が失踪して、メンテナンスされない時期がたまにあります。
  - CMTに合意したライブラリは、何人かのBoost開発者が適時メンテナンスします。
    (pull requestを代わりに受け付けたり)

# 本日の紹介はここまで

- 今回の「Boostライブラリー周の旅」では、
  Boost 1.54.0から1.58.0までの更新を紹介しました。

- 発表では差分のみを紹介していますが、
  これまで紹介したものをマージした資料も公開しています。

- 今回Boostに興味を持たれた方は、そちらのマージした資料もぜひご覧ください。