Boostライブラリー周の旅

Ver 1.61.0

高橋 晶(Akira Takahashi)
faithandbrave@gmail.com

2016/07/23 Boost.勉強会 #20 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 1.61.0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。
この資料は、バージョン間の差分を紹介するものですが、
これまでの資料をマージしたバージョンも公開します。

バージョンアップの詳細

- boostjpサイトでは、Boostのリリースノートを翻訳したものを公開しています
- http://boostjp.github.io/document/version.html
  この階層以下に、各バージョンのリリースノートがあります
- 本家のリリースノートよりもくわしく書いています
  - チケットタイトルを直訳するのではなく、問題と修正の内容を確認して書いています
  - 新機能が入った際には、ドキュメントから概要とサンプルコードを持ってきたりもしています
  - 本家より優れたものを作ることが目的ではなく、日本語で情報を届けるのに適したやり方として情報補完をしています

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

- Compute
- DLL
- Hana
- Metaparse

質問は随時受け付けます

- この発表では、4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。
- 発表時間に余裕があるので、随時質問してください。

Compute

OpenCLをベースとした、マルチコアCPUやGPGPUを扱うライブラリ。boost::compute名前空間にあるデータ構造とアルゴリズムが、デバイス用のもの。ホストのデータ構造とやりとりできる。設計はThrust風。

```
// デバイスのセットアップ…

std::vector<int> host_data = { 1, 3, 5, 7, 9 };
compute::vector<int> device_vector(5, context);

// ホスト環境からデバイス環境にデータをコピー
compute::copy(
  host_data.begin(),
  host_data.end(),
  device_vector.begin(), queue
);
```

DLL

DLLやSOといった形式になっているC++ライブラリを扱うライブラリ。
インポート、エクスポート、関数呼び出しなどの機能をサポートする

```
// インポート側
auto cpp11_func = dll::import<int(std::string&&)>(
                  path_to_shared_library, "i_am_a_cpp11_function");
// 関数呼び出し
cpp11_func("hello");
```

```
// エクスポート側
#define API extern "C" BOOST_SYMBOL_EXPORT
namespace some_namespace {
    API int i_am_a_cpp11_function(std::string&& param) noexcept;
}
```

Hana 1/2

C++14時代のメタプログラミングライブラリ。
FusionとMPLを合わせたもの。
値のシーケンスと型のシーケンス両方への操作を一様にできる。

```
// 値のシーケンスを扱う例
auto xs = hana::make<hana::tuple_tag>(1, 3.14, "hello");

// タプルの全ての要素を文字列に変換
auto ys = hana::transform(xs, [](auto x) {
  return boost::lexical_cast<std::string>(x);
});

hana::for_each(ys, [](const std::string& x) {
  std::cout << x << std::endl;
});
```

Hana 2/2

C++14時代のメタプログラミングライブラリ。
FusionとMPLを合わせたもの。
値のシーケンスと型のシーケンス両方への操作を一様にできる。

```
// 型のシーケンスを扱う例
auto xs = hana::tuple_t<int, double, std::string>;

// 算術型のみ抽出する
auto ys = hana::filter(xs, [](auto x) {
  return hana::traits::is_arithmetic(x);
});

static_assert(hana::to_tuple(ys) == hana::tuple_t<int, double>);
```

Metaparse 1/3

コンパイル時の構文解析ライブラリ。
コンパイル時文字列クラスも付いている。
printfのフォーマットや正規表現などをコンパイル時に検証するために
使用できる。

```
// コンパイル時文字列
using hello1 = string<'H', 'e', 'l', 'l', 'o'>;
using hello2 = BOOST_METAPARSE_STRING("Hello");

static_assert(
  std::is_same_v<hello1, hello2>::type::value
);
```

Metaparse 2/3

コンパイル時の構文解析ライブラリ。
コンパイル時文字列クラスも付いている。
printfのフォーマットや正規表現などをコンパイル時に検証するために
使用できる。

```
// コンパイル時に、文字列を整数に変換
using digit_value = transform<digit, util::digit_to_int<>>;
static_assert(
  get_result<
    digit_value::apply<BOOST_METAPARSE_STRING("0"), start>
  >::type::value == 0
);
```

Metaparse 3/3

コンパイル時の構文解析ライブラリ。
コンパイル時文字列クラスも付いている。
printfのフォーマットや正規表現などをコンパイル時に検証するために
使用できる。

```
// 演算式を解析
using expr = sequence<token<int_>, token<lit_c<'+'>>, token<int_>>;
using parser = build_parser<expr>;

static_assert(
  !is_error<
    parser::apply<BOOST_METAPARSE_STRING("11 + 2")>
  >::type::value
);
```

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 1.61.0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。