Which Boost features overlap with C++11?

I put my C++ skills on the shelf several years ago and it seems now, when I need them again, the landscape has changed.

We have got C++11 now, and my understanding is that it overlaps many Boost features.

Is there some summary where those overlaps lie, which Boost libraries going to become legacy, recommendation of which C++11 features to use instead of boost ones and which better not?

c++ boost c++11

edited Jun 26 '16 at 23:10
jotik
7,481 6 27 80

asked Jan 13 '12 at 13:58
user377178
1,083 3 9 10

4 Boost was one of the first to implement the TR1 library. Since that's now in the standard, you should prefer the standard version. Boost.Lambda is also sort of replaced by actual lambdas now. – Kerrek SB Jan 13 '12 at 14:00

6 The Wikipedia article on C++11 has a good summary of most changes. – Some programmer dude Jan 13 '12 at 14:03

2 Answers

Replaceable by C++11 language features or libraries

- Foreach → range-based for
- Functional/Forward → Perfect forwarding (with rvalue references, variadic templates and std::forward)
- In Place Factory, Typed In Place Factory → Perfect forwarding (at least for the documented use cases)
- Lambda → Lambda expression (in non-polymorphic cases)
- Local function → Lambda expression
- Min-Max → std::minmax, std::minmax_element
- Ratio → std::ratio
- Static Assert → static_assert
- Thread → <thread>, etc (but check this question).
- Typeof → auto, decltype
- Value initialized → List-initialization (§8.5.4/3)
- Math/Special Functions → `<cmath>`, see the list below
    - gamma function (tgamma), log gamma function (lgamma)
    - error functions (erf, erfc)
    - `log1p`, `expm1`
    - `cbrt`, `hypot`
    - `acosh`, `asinh`, `atanh`

TR1 (they are marked in the documentation if those are TR1 libraries)

- Array → std::array
- Bind → std::bind
- Enable If → std::enable_if
- Function → std::function
- Member Function → std::mem_fn
- Random → <random>
- Ref → std::ref, std::cref
- Regex → <regex>
- Result Of → std::result_of

- Swap (swapping arrays) → std::swap
- Tuple → std::tuple
- Type Traits → <type_traits>
- Unordered → <unordered_set>, <unordered_map>

Features back-ported from C++11:

- Atomic ← std::atomic
- Chrono ← <chrono> (see below)
- Move ← Rvalue references

Replaceable by C++17 language features:

- String_ref → std::string_view
- Filesystem → <filesystem> (Filesystem TS)
- Optional → std::optional (Library Fundamentals TS v1)
- Any → std::any (Library Fundamentals TS v1)
- Math/Special Functions → `<cmath>` (Special Math IS), see the list below
  - beta function
  - (normal / associated / spherical) Legendre polynomials
  - (normal / associated) Legendre polynomials
  - Hermite polynomials
  - Bessel (J / Y / I / K) functions (Y is called Neumann function in C++)
  - spherical Bessel (j / y) functions
  - (incomplete / complete) elliptic integrals of (first / second / third kind)
  - Riemann zeta function
  - exponential integral Ei
- Variant → std::variant (P0088R2)

The standard team is still working on it:

- Math Common Factor → std::experimetal::gcd, lcm (Library Fundamentals TS v2)
- Concept check → Concepts TS
- Range → Range TS
- Asio → Networking TS (sockets and timers only)
- Multiprecision → Numerics TS
- Coroutine/Coroutine2 → Coroutines TS

A large part of MPL can be trimmed down or removed using variadic templates. Some common use cases of Lexical cast can be replaced by std::to_string and std::sto*X*.

Some Boost libraries are related to C++11 but also have some more extensions, e.g. Boost.Functional/Hash contains hash_combine and related functions not found in C++11, Boost.Chrono has I/O and rounding and many other clocks, etc. so you may still want to take a look at the boost ones before really dismissing them.

edited May 23 '17 at 12:02
Community ♦
1 1

answered Jan 13 '12 at 14:55
kennytm
372k 70 850 879

1 Add to the list Boost.Chrono, Boost.Exception, and Boost.Swap. – ildjarn Jan 13 '12 at 17:59

9 Note that Boost.Lambda (or rather, Boost.Phoenix' lambdas), are still useful for polymorphic lambdas. – Xeo Jan 13 '12 at 18:00

2 Nice list, although I do not believe `std::unique_ptr` is part of TR1 (since it requires move semantics) – Nemo Jan 13 '12 at 18:07

1 @ildjarn: Boost.Chrono provides much more functions than <chrono>. Boost.Exception — only N2179 is relevant. – kennytm Jan 13 '12 at 18:41

2 @Nemo: Yes. Only std::tr1::shared_ptr is part of TR1, and const std::unique_ptr replaces the use cases of boost::scoped_ptr and boost::scoped_array – kennytm Jan 13 '12 at 18:43

Actually, I don't think the boost libraries are going to become legacy.

Yes, you should be able to use `std::type_traits`, `regex`, `shared_ptr`, `unique_ptr`, `tuple<>`, `std::tie`, `std::begin` instead of Boost Typetraits/Utility, Boost Smartpointer, Boost Tuple, Boost Range libraries, but there should in practice be no real need to 'switch' unless you are moving more of your code to c++11.

Also, in my experience, the `std` versions of most of these are somewhat less featureful. E.g. AFAICT the standard does *not* have

- Perl5 regular expressions
- call_traits
- Certain regex interface members (such as `bool boost::basic_regex<>::empty()`) and othe interface differences
  - this bites more since the Boost interface is exactly matched with Boost Xpressive
  - and it plays much more nicely with Boost String Algorithms Obviously, the latter don't have *standard* counterparts (yet?)
- Many things relating to TMP (Boost Fusion)
- Lazy, expression template-based lambdas; they have inevitable benefits in that they *can* be polymorphic *today*, as opposed to C++11. Therefore they can often be more succinct:

  ```
  std::vector<int> v = {1,2,-9,3};

  for (auto i : v | filtered(_arg1 >=0))
      std::cout << i << "\n";

  // or:
  boost::for_each(v, std::cout << _arg1);
  ```

  Most definitely, this still has some appeal over C++11 lambdas (with trailing return types, explicit capturing and declared parameters).

Also, there is a BIG role for Boost, precisely in facilitating path-wise migration from C++03 to C++11 and integrating C++11 and C++03 codebases. I'm particularly thinking of

- Boost Auto (BOOST_AUTO)
- Boost Utility (`boost::result_of<>` and related)
- Boost Foreach (BOOST_FOREACH)
- Don't forget: Boost Move - which makes it possible to write classes with move semantics with a syntax that will compile equally well on C++03 compilers with Boost 1_48+ and C++11 compilers.

Just my $0.02

answered Jan 13 '12 at 15:05

sehe
252k 31 303 420