itcppcon20

An hit and run

from header-only to static linking

@fiorentinoing

2020-06-13 · Como, IT

about

- C++ header-only library are easy to use
    - git clone or copy&paste
- drawbacks
    - target size bloat
    - compilation time

Provide a static linking way to use your lib

not addressing

If the current implementation of your header-only template library is **only** about variadic template to manage context-aware user-space's lambda function, then this talk will **not** help you.

addressing

Libs about template completely instantiable in the **library-space** (i.e. all *configurations* are known in the library-space).

This brief presentation shows how to move all possibile instantiation from user-space *compilation time* to the library-space one.

Here is some code

Here is the main.cpp

```cpp
#include <lib/lib.hpp>

int main()
{
    using namespace network;
    bind<flags::IPv4>("127.0.0.1", 4242);
    bind<flags::IPv6>("::1", 4242);
    return 0;
}
```

The header only version of lib.hpp looks like this:

```cpp
namespace network {
    namespace flags {
        struct IPv4{};
        struct IPv6{};
    } // namespace flags

    template <typename I>
    inline void bind(char const *ip, unsigned port) {
        std::puts(__PRETTY_FUNCTION__);
    }
} // namespace network
```

The static lib version of lib.h looks like this:

```
namespace network {
    namespace flags {
        struct IPv4{};
        struct IPv6{};
    } // namespace flags

    template <typename I>
    void bind(char const *ip, unsigned port);

    extern template void bind<flags::IPv4>(char const *ip, unsigned port);
    extern template void bind<flags::IPv6>(char const *ip, unsigned port);
} // namespace network
```

Note the extern keyword.

And the cpp looks like this:

```cpp
#include "lib.h"

namespace network {
    template <typename I>
    void bind(char const *ip, unsigned port) {
        std::puts(__PRETTY_FUNCTION__);
    }

    template void bind<flags::IPv4>(char const *ip, unsigned port);
    template void bind<flags::IPv6>(char const *ip, unsigned port);
} // namespace network
```

Note the missing inline and extern keywords.

Static linking

You can provide your clients with a static lib with **all**, some or *none* configuration available at linking time.

Depending on the amount of library-space explicitly instantiable code, you'll have a lot of **benefits** from this technique.

Performance (1)

Header-only tests build takes:

```
$> cmake -D BUILD_UVW_LIBS=OFF
$> make uv uv_a gtest gtest_main gmock gmock_main -j4
$> time make -j4

real    0m49,253s
user    2m57,287s
sys     0m8,493s
```

Performance first (2)

Static libs user-space compilation time takes:

```
$> cmake -D BUILD_UVW_LIBS=ON
$> make uv uv_a gtest gtest_main gmock gmock_main uvw-static uvw-shared -j4
$> time make -j4

real    0m15,699s
user    0m55,183s
sys     0m3,416s
```

3.13x speedup (or -68%)

The end

@fiorentinoing