B4Mesh : Http node interface

The component - *b4mesh::http* - described hereunder use an existing `boost::io_context` to provide a simple but yet extensive asynchronous http endpoints collections.

Dependencies

This components relies on nothing else but the GreenIT SDK.

In details :

- Threads::Threads
- Boost::system
- Boost::asio
- Boost::beast

Among theses dependencies, those who are dynamic/static libraries are linked automatically the INTERFACE consumer,
and header-only libraries (Boost::beast) are wrapped into INTERFACE library (boost_beast).

Integration

Using CMake, like any other INTERFACE library.
The library name is `b4mesh_http`

```
add_subdirectory(path/to/b4mesh_http)

# Use `b4mesh_http` ...
add_executable(my_binary
    ${CMAKE_CURRENT_SOURCE_DIR}/sources/source.cpp
)
set_target_properties(my_binary
    PROPERTIES
        CXX_STANDARD 17
)
add_dependencies(my_binary b4mesh_http)
target_link_libraries(my_binary
    PRIVATE
        b4mesh_http
)
```

Also, it fully supports CMake's `FetchContent` dependency management.

```
# As Git repository
FetchContent_Declare(
  b4mesh_http-dependency_management
  GIT_REPOSITORY #<path_to_repo.git>
```

```
  GIT_TAG        #<release tag>
)
FetchContent_MakeAvailable(b4mesh_http-dependency_management)

# use `b4mesh_http` target ...
```

As tarball

```
# As Git repository
FetchContent_Declare(
  b4mesh_http-dependency_management
  URL       path/to/b4mesh_http.tar.gz
  URL_HASH  #<url's hash>
)
FetchContent_MakeAvailable(b4mesh_http-dependency_management)

# use `b4mesh_http` target ...
```

Interface

Each endpoint is generated using an initializer-list of :

```
struct /*unspecified : endpoint_argument */ {
    std::string /*unspecified : uri [protocol://ip:port/path/to/ressource]*/;
    std::initializer_list<boost::beast::http::verb> /* unspecified : methods */;
    /* unspecified ->
std::convertible_to<std::function<std::string(std::string_view)>> */
};

// Using structure-binding :
auto [
    uri_as_std_string,
    allowed_methods,
    behavior_callback
    ] = /*unspecified : endpoint_argument*/ {};
```

Thus, the implementation might looks like :

```
using first_argument_type = boost::io_context &;
using second_argument_type = std::initializer_list</*unspecified : see
specifications above*/>;

auto b4mesh:http::add_enpoints(
    first_argument_type first_argument,
    second_argument_type second_argument
)
{ /* implementation logic */ }
```

Synthax example :

```
const auto      threads_count = 1;
net::io_context io_service{threads_count};

using method = boost::beast::http::verb;
auto listeners = b4mesh:http::add_enpoints(
    io_service,
    {
        {
            "0.0.0.0:4242/benchmark",
            { method::put, method::post},
            [](std::string_view request_datas)
                -> std::string
            {
                return {};
            }
        },
        {
            "0.0.0.0:4242/error",
            { method::get },
            [](std::string_view request_datas)
                -> std::string
            {
                std::cout << "error : [GET] received : [" << request_datas <<
"]\n";
                throw std::runtime_error{"test error\n"};
            }
        },
        {
            "0.0.0.0:4242/add_transaction",
            { method::put, method::post},
            [](std::string_view request_datas)
                -> std::string
            {
                std::cout << "add_transaction : [PUT, POST] received : [" <<
request_datas << "]\n";
                return "ok from /add_transaction\n";
            }
        }
    }
);
```

Return type

Calling `b4mesh:http::add_enpoints` will return an unspecified collection of type `listeners_type` *(see below)*, which `std::size(/*listeners_type : value*/)` is less-or-equal `std::size()` result of the second argument *(methods)*.

Size guarantee

There is a strong guarantee that `std::size(/*listeners_type : value*/)` is equal to the number of unique pair of `{ ip, port }` in `add_http_endpoint`'s second argument elements.

For instance :

```
auto listeners = b4mesh:http::add_enpoints(
    io_service,
    {
        {
            "0.0.0.0:4242/benchmark",
            { method::put, method::post},
            [](std::string_view request_datas){ return std::string{};}
        },
        {
            "0.0.0.0:4242/error",
            { method::get },
            [](std::string_view request_datas){ return std::string{};}
        }
    }
);

const auto listeners_size = std::size(listeners);   // size is 1
                                                    // the only endpoint is
[0.0.0.0:4242]
```

Types guarantees

```
auto listeners = b4mesh:http::add_enpoints(
    io_service,
    {}
);
using listeners_type = decltype(listeners);
using listener_type = decltype(listeners)::value_type; //
std::shared_ptr</*unspecified*/>
using endpoint_type = decltype(listeners)::value_type::element_type;
```

You are guarantee that `listeners_type` match the following named requirements :

- MoveAssignable
- Destructible
- Swappable
- Container
- AllocatorAwareContainer
- SequenceContainer
- ContiguousContainer
- ReversibleContainer

`listeners_type::value_type` is a `std::shared_ptr</*unspecified*/>`

Behavior & error management

Creation

- Attempting to create an endpoint using an illed-formed uri results in an `std::invalid_argument` exception.
- Attempting to create an endpoint an empty methods list results in an `std::invalid_argument` exception

Behavior is undefined if the lifetime of `decltype(add_endpoints())` value is shorter than the `boost::io_context` value used to create it.

Request processing

When received, a request is processed threw an internal router which redirect it to its matching behavior_callback.

Such `behavior_callback` is designed the following simple way :

- Request body is passed as argument (as `std::string_view`)
- Return (as moveable `std::string`) is sent back to the request's emitter, as response
- Any thrown exception will result in a server_error, which's body is equal to `error : '<msg>'`, where `msg` is the return of `std::exception::what()`.
- Any thrown value which does not satisfy `std::derived_from<std::exception>` concept results in the same behavior described previously, with `msg` set to `"unknown error"`;

Destruction / end-of-lifetime

As mentionned before, the behavior is undefined when the lifetime of `decltype(add_endpoints())` value is shorter than the `boost::io_context` value used to create it.

Otherwise, `decltype(add_endpoints())` value destruction will clean any allocated ressources.

Limitations

Currently, no mime-type are provided as both `behavior_callback` argument and return type.

> Easy to implement though

Performances / Benchmarks

Using `Apache-Bench`.

Payload :

```
{ "transaction": [
    { "payload": "0123456789"}
]}
```

Nb core *(send)*	**Nb core *(receive)***	**req/sec**	**time per request**	**command**
1	1	3234.60 [#/sec]	0.309 [ms]	ab -t 1000 -c 1 -T 'application/json' -u add_transaction.payload.json http://0.0.0.0:4242/benchmark
4	1	4171.57 [#/sec]	0.240 [ms]	ab -t 1000 -c 4 -T 'application/json' -u add_transaction.payload.json http://0.0.0.0:4242/benchmark
4	4	8116.32 [#/sec]	0.123 [ms]	ab -t 1000 -c 4 -T 'application/json' -u add_transaction.payload.json http://0.0.0.0:4242/benchmark