WEB | C++

ROADMAP

- Where
- Client
- Server

Serve static Webpage

```
import SimpleHTTPServer
import SocketServer

Handler = SimpleHTTPServer.SimpleHTTPRequestHandler
httpd = SocketServer.TCPServer(("", 8000), Handler)

httpd.serve_forever()
```

WEB SERVICE

WHAT MAKES WEB SERVICES SO POPULAR?

- Minimal first use effort
- Out of the box cross platform support

WEB DEVELOPMENT

ADVANTAGES

- Ecosystem
- Tools
- Frameworks

CHALLENGES

- Javascript (still)
- Browser support

DEVELOPMENT CYCLE

1. Edit .jsx .scss .tx ...
2. Compile to ECMAScript version X and plain css
3. Uglify
4. Bundle

EXAMPLE SERVICE

- Persistence
- Bidirectional communication

VAR.BZ

Trust me, this is "definitely" not a malicious site

WEBSOCKET

3 LIBRARIES

- Beast
- uWebSockets
- IncludeOS

REQUIREMENTS

- request handler
- send
- broadcast

UWEBSOCKETS

SETUP

```
#include <uWS/uWS.h>

uWS::Hub h;
h.onMessage(request_handler);
h.listen(3003);
h.run();
```

REQUEST HANDLER

```
h.onMessage([&h, &poll_data]( //) md fix
  uWS::WebSocket<uWS::SERVER>* ws,
  char* message,
  size_t length,
  uWS::OpCode
) {

}
```

SERIALIZATION

FLATBUFFERS

VALIDATION

```
const auto ok = flatbuffers::Verifier(
  reinterpret_cast<const uint8_t*>(message), length
).VerifyBuffer<Strawpoll::Request>(nullptr);
```

SCHEMA

```
enum RequestType:byte { Poll, Result }

table Request {
  type:RequestType;
  vote:long;
}
```

SEND

```
void sendResponse(
  uWS::WebSocket<uWS::SERVER>* ws,
  FlatBufferRef buffer
) {
  ws->send(
    reinterpret_cast<const char*>(buffer.data),
    buffer.size,
    uWS::OpCode::BINARY
  );
}
```

BROADCAST

```
void broadcastResponse(
  uWS::Hub& h,
  FlatBufferRef buffer
) {
  h.getDefaultGroup<uWS::SERVER>().broadcast(
    reinterpret_cast<const char*>(buffer.data),
    buffer.size,
    uWS::OpCode::BINARY
  );
}
```

REQUEST SWITCH

```
switch(request->type()) {
  case Strawpoll::RequestType_Poll:
    sendResponse(ws, poll_data.poll_response.ref());
    break;
  case Strawpoll::RequestType_Result:
    poll_data.register_vote(
      request->vote(),
      {},
      [&ws](FlatBufferRef br) { sendResponse(ws, br); },
      [&h](FlatBufferRef br) { broadcastResponse(h, br); }
    );
    break;
  default:
    sendResponse(ws, poll_data.invalid_type.ref());
}
```

CLIENT

- WebSocket support
- FlatBuffers support

SETUP

```
setupWebSocketCommunication() {
  this.socket = new WebSocket(this.props.apiUrl);
  this.socket.binaryType = 'arraybuffer';

  this.socket.addEventListener('open', this.fetchPoll);
  this.socket.addEventListener('message',
    this.handleServerResponse
  );
  this.socket.addEventListener('close', this.handleDisconnect);
}
```

RESPONSE SWITCH

```
switch(response.type()) {
  case Strawpoll.ResponseType.Poll:
    this.updatePoll(response.poll());
    break;
  case Strawpoll.ResponseType.Result:
    this.updateResult(response.result());
    break;
  case Strawpoll.ResponseType.Error:
    console.error("Error: ", response.error());
    break;
  default:
    console.error("Invalid response type: ", response.type());
};
```

JS WUT?

SORT NUMBER ARRAY

```
const options = [1, 10, 21, 2].sort();
```

```
[1, 10, 2, 21]
```

```
const options = [1, 10, 21, 2].sort((a, b) => a < b);
```

```
[1, 10, 2, 21] or [1, 2, 10, 21]
```

```
const options = [1, 10, 21, 2].sort((a, b) => a - b);
```

```
[1, 2, 10, 21]
```

IMPLICIT CONVERSIONS

```
([![]]+[][[]])[+!+[]+[+[]]]+(!![]+[])[+[]]+(![]+[])[!+[]+
!+[]+!+[]]+(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+
([][[]]+[])[+!+[]]+(!![]+[])[!+[]+!+[]+!+[]]
```

INCLUDEOS

SETUP

```
#include <net/inet4>
#include <service>
#include <net/ws/websocket.hpp>
#include <net/http/server.hpp>

auto& inet = net::Inet4::stack<0>();
auto server = std::make_unique<http::Server>(inet.tcp());
server->on_request(request_handler);
server->listen(3003);
```

REQUEST HANDLER

```
server->on_request([] (auto req, auto rw)
{
  if(req->method() != http::GET) {
    rw->write_header(http::Not_Found);
    return;
  }
  if(req->uri() == "/ws") {
    auto ws = net::WebSocket::upgrade(*req, *rw);
    handle_ws(std::move(ws));
  }
  else {
    rw->write_header(http::Not_Found);
  }
});
```

MESSAGE LOOP

```
ws->on_read = [ws = ws.get()](auto msg)
{
  ws->write(
    msg->data(),
    msg->size(),
    msg->opcode()
  );
};
```

SEND

```
void sendResponse(
  WebSocket_ptr ws,
  FlatBufferRef buffer
) {
  ws->write(
    reinterpret_cast<const char*>(buffer.data),
    buffer.size,
    net::op_code::BINARY
  );
}
```

BEAST

SETUP

```
#include <boost/beast/core.hpp>
#include <boost/beast/websocket.hpp>
#include <boost/asio/ip/tcp.hpp>
#include <boost/asio/strand.hpp>

boost::asio::io_service ios{1};
listener lis{ios, tcp::endpoint{address, port}};
ios.run();
```

EVENT LOOPS

CONNECTION LOOP

```
tcp::acceptor acceptor{ios, {address, port}};

for(;;)
{
    tcp::socket socket{ios};
    acceptor.accept(socket);

    std::thread{[so = std::move(socket)]()
      { do_session(std::move(so)); }
    }.detach();
}
```

MESSAGE LOOP

```
void do_session(tcp::socket&& socket)
{
  websocket::stream<tcp::socket> ws{std::move(socket)};
  ws.accept();

  for(;;)
  {
    boost::beast::multi_buffer buffer;
    ws.read(buffer);

    ws.text(ws.got_text());
    ws.write(buffer.data());
  }
}
```

ASYNC IO

NON SEQUENTIAL LOOP

goto

CYCLIC CALL GRAPH

```
void foo();

void bar() { foo(); }
void foo() { bar(); }
```

ACCEPTING CONNECTION

```
explicit listener()
{
  if (!acceptor_.is_open()) return;
  do_accept();
}

void do_accept()
{
  acceptor_.async_accept(
    socket_,
    strand_.wrap([this](boost::system::error_code ec)
      { on_accept(ec); }
    )
  );
}
```

NEW SESSION

```
void on_accept(boost::system::error_code ec)
{
  if (ec) fail(ec, "accept");
  else
  {
    sessions_.try_emplace(
      session_id_counter_++,
      std::move(socket_)
    );
  }

  do_accept();
}
```

MESSAGE LOOP | START

```
explicit session(tcp::socket&& socket)
    : ws_{std::move(socket)}
{
  ws_.async_accept(
    strand_.wrap([this](boost::system::error_code ec)
      { on_accept(ec); }
    )
  );
}
```

MESSAGE LOOP | READ

```
void do_read()
{
  buffer_.consume(buffer_.size());

  ws_.async_read(
    buffer_,
    strand_.wrap(
      [this](
        boost::system::error_code ec,
        size_t bytes_transferred
      )
      { on_read(ec, bytes_transferred); }
    )
  );
}
```

MESSAGE LOOP | WRITE

```
void do_write()
{
  ws_.async_write(
    std::array<boost::asio::const_buffer, 1>{{
      std::move(message_queue_.back())
    }},
    strand_.wrap(
      [this](
        boost::system::error_code ec,
        size_t bytes_transferred
      )
      { on_write(ec, bytes_transferred); }
    )
  );
}
```

SESSION EVENT CHAIN

- on_accept->
- do_read->
- on_read->
- do_write->
- on_write->
- do_read->

BROADCAST

```
using session_t = session<on_session_close_t, broadcast_t>;
using sessions_t = std::unordered_map<size_t, session_t>;

void broadcast(FlatBufferRef br)
{
  for (auto& [key, session] : sessions_)
  {
    session.add_message(br);
    session.flush_message_queue();
  }
}
```

QUEUING MECHANISM

```
public:
  void add_message(FlatBufferRef br)
  {
    if (!write_in_process_ || has_voted())
      message_queue_.push_back({ br.data, br.size });
  }

private:
  std::vector<boost::asio::const_buffer> message_queue_;
```

FLUSHING

```
void flush_message_queue()
{
  if (write_in_process_) return;
  if (message_queue_.empty()) return;

  ws_.async_write(
    std::array<boost::asio::const_buffer, 1>{{
      std::move(message_queue_.back())
    }},
    strand_.wrap(after_write)
  );

  write_in_process_ = true;
  message_queue_.pop_back();
}
```

EVENT CHAIN INTEGRITY

```
void after_write(
  boost::system::error_code ec,
  size_t bytes_transferred
)
{
  write_in_process_ = false;
  on_write(ec, bytes_transferred);
}
```

```
void do_read()
{
  if (read_in_process_) return;

  ws_.async_read([...])

  read_in_process_ = true;
}
```

EXIT

```
void on_read(
  boost::system::error_code ec,
  size_t bytes_transferred
) {
  if (ec == websocket::error::closed)
  {
    on_close_(session_id_);
    return;
  }

  [...]
}
```

AVOIDING POINTER INVALIDATION

```
struct ErrorResponses {
  const FlatBufferWrapper invalid_message = make_msg(
    "Invalid request message"
  );
  const FlatBufferWrapper invalid_type = make_msg(
    "Invalid request type"
  );
};

ErrorResponses error_responses{};
```

INITILIZE ONCE | REFILL

```
void inplace_assign(const FlatBufferWrapper& other)
{
  // placeholder error handling
  if (other.buffer_.size() != buffer_.size())
    return;

  std::copy_n(
    other.buffer_.data(),
    other.buffer_.size(),
    buffer_.data()
  );
}
```

EXAMPLE CODE SIZE

- uWebSockets | ~1.6kb
- IncludeOS | ~3kb
- Beast | ~9kb

EXAMPLE COMPILE TIME

- uWebSockets | Debug: 1.4s | Release: 1.8s
- IncludeOS | Default: 2.9s
- Beast | Debug: 8.5s | Release: 17.1s

DOCUMENTATION

THREADING

SSL

SECURITY

PERFORMANCE

100 CONCURRENT CONNECTIONS

MEMORY USAGE | USER SPACE

- uWebSockets: 4.6mb
- Beast: 4.2mb

CONNECTIONS PER MS

- uWebSockets: 7.1
- Beast: 5.3

10.000 CONCURRENT CONNECTIONS

MEMORY USAGE | USER SPACE

- uWebSockets: 6.4mb
- Beast: 49.3mb

CONNECTIONS PER MS

- uWebSockets: 35.0
- Beast: 26.4

BEST LIBRARY?

WRONG QUESTION!

KEY STRENGTH

- Easy | uWebSockets
- Secure | IncludeOS
- Modular | Beast

TAKEAWAY

NICHE?

FUTURE

- Networking Ts
- Executor Ts
- Coroutine Ts

SOURCE

https://github.com/Voultapher/Presentations

CONTACT

- Email: lukas.bergdoll@gmail.com
- GitHub: https://github.com/Voultapher

VOTE RESULT

Q&A