test2

Sam Hobbs

September 4, 2019

Contents

1 mongodb/pathloss.service.cpp

```cpp
#include <iostream>
#include <exception>
#include <chrono>
#include <future>
#include <softroles/propagation/pathloss.hpp>

#include <bsoncxx/builder/basic/document.hpp>
#include <bsoncxx/builder/basic/kvp.hpp>
#include <bsoncxx/json.hpp>
#include <bsoncxx/oid.hpp>
#include <bsoncxx/string/to_string.hpp>
#include <mongocxx/change_stream.hpp>
#include <mongocxx/client.hpp>
#include <mongocxx/instance.hpp>
#include <mongocxx/pool.hpp>
#include <mongocxx/uri.hpp>

using::bsoncxx::builder::basic::kvp;
using::bsoncxx::builder::basic::make_document;

namespace {

std::string get_server_version(const mongocxx::client& client) {
  bsoncxx::builder::basic::document server_status{};
  server_status.append(bsoncxx::builder::basic::kvp("serverStatus", 1));
  bsoncxx::document::value output = client["test"].run_command(server_status.extract());

  return bsoncxx::string::to_string(output.view()["version"].get_utf8().value);
}

}

void func(mongocxx::collection collection, const bsoncxx::document::element& doc) {
  auto id = doc["_id"].get_oid().value.to_string();
  std::string error;
  float freq, dist;
  try {
    freq = std::stof(static_cast<std::string>(doc["args"]["freq"].get_utf8().value));
    dist = std::stof(static_cast<std::string>(doc["args"]["dist"].get_utf8().value));
  } catch(const std::exception& e) {
    error = e.what();
    collection.update_one(make_document(kvp("_id", bsoncxx::oid(id))),
                          make_document(kvp("$set", make_document(kvp("error",
                              make_document(kvp("stage", "arg␣parse"), kvp("message",
                              error)))))));
  }
  try {
    auto start = std::chrono::high_resolution_clock::now();
    auto result = softroles::propagation::pathloss(freq, dist);
    auto stop = std::chrono::high_resolution_clock::now();
    auto duration = static_cast<int>(std::chrono::duration_cast<std::chrono::microseconds
        >(stop-start).count());
    collection.update_one(make_document(kvp("_id", bsoncxx::oid(id))),
                          make_document(kvp("$set", make_document(kvp("output",result),
                              kvp("duration",duration)))));
  } catch(const std::exception& e) {
    error = e.what();
    collection.update_one(make_document(kvp("_id", bsoncxx::oid(id))),
                          make_document(kvp("$set", make_document(kvp("error",
                              make_document(kvp("stage", "calculate"), kvp("message",
                              error)))))));
  }
```

```
}

int main(int argc, char **argv) {
  try {
    mongocxx::instance inst {};
    mongocxx::pool pool{mongocxx::uri{}};

    auto entry = pool.acquire();

    if (get_server_version(*entry) < "3.6") {
      std::cerr << "Change␣streams␣are␣only␣supported␣on␣Mongo␣versions␣>=␣3.6." << std::
          endl;
      // CXX-1548: Should return EXIT_FAILURE, but Travis is currently running Mongo 3.4
      return EXIT_SUCCESS;
    }

    mongocxx::options::change_stream options;
    const std::chrono::milliseconds await_time{1000};
    options.max_await_time(await_time);
    options.full_document("updateLookup");

    auto collection = (*entry)["modules"]["propagation"];
    mongocxx::change_stream stream = collection.watch(options);

    while (true) {
      for (const auto& event : stream) {
        //std::cout << bsoncxx::to_json(event) << std::endl;
        auto operation = event["operationType"].get_utf8().value;
        if(operation.compare("insert") == 0) {
          auto doc = event["fullDocument"];
          auto function = doc["function"].get_utf8().value;
          //std::cout << collection << std::endl;
          if(function.compare("pathloss") == 0) {
            std::future<void> async = std::async(func, collection, doc);
          }
        }
      }
    }

  } catch (const std::exception& exception) {
    std::cerr << "Caught␣exception␣\"" << exception.what() << "\"" << std::endl;
  } catch (...) {
    std::cerr << "Caught␣unknown␣exception␣type" << std::endl;
  }
  return EXIT_SUCCESS;
}
```

2 mongodb/radiohorizon.service.cpp

```cpp
#include <iostream>
#include <exception>
#include <chrono>
#include <future>
#include <softroles/propagation/radiohorizon.hpp>

#include <bsoncxx/builder/basic/document.hpp>
#include <bsoncxx/builder/basic/kvp.hpp>
#include <bsoncxx/json.hpp>
#include <bsoncxx/oid.hpp>
#include <bsoncxx/string/to_string.hpp>
#include <mongocxx/change_stream.hpp>
#include <mongocxx/client.hpp>
#include <mongocxx/instance.hpp>
#include <mongocxx/pool.hpp>
#include <mongocxx/uri.hpp>

using::bsoncxx::builder::basic::kvp;
using::bsoncxx::builder::basic::make_document;

namespace {

std::string get_server_version(const mongocxx::client& client) {
  bsoncxx::builder::basic::document server_status{};
  server_status.append(bsoncxx::builder::basic::kvp("serverStatus", 1));
  bsoncxx::document::value output = client["test"].run_command(server_status.extract());

  return bsoncxx::string::to_string(output.view()["version"].get_utf8().value);
}

}

void func(mongocxx::collection collection, const bsoncxx::document::element& doc) {
  auto id = doc["_id"].get_oid().value.to_string();
  std::string error;
  float height;
  try {
    height = std::stof(static_cast<std::string>(doc["args"]["height"].get_utf8().value));
  } catch(const std::exception& e) {
    error = e.what();
    collection.update_one(make_document(kvp("_id", bsoncxx::oid(id))),
                          make_document(kvp("$set", make_document(kvp("error",
                              make_document(kvp("stage", "arg␣parse"), kvp("message",
                              error)))))));
  }
  try {
    auto start = std::chrono::high_resolution_clock::now();
    auto result = softroles::propagation::radiohorizon(height);
    auto stop = std::chrono::high_resolution_clock::now();
    auto duration = static_cast<int>(std::chrono::duration_cast<std::chrono::microseconds
        >(stop-start).count());
    collection.update_one(make_document(kvp("_id", bsoncxx::oid(id))),
                          make_document(kvp("$set", make_document(kvp("output",result),
                              kvp("duration",duration)))));
  } catch(const std::exception& e) {
    error = e.what();
    collection.update_one(make_document(kvp("_id", bsoncxx::oid(id))),
                          make_document(kvp("$set", make_document(kvp("error",
                              make_document(kvp("stage", "calculate"), kvp("message",
                              error)))))));
  }

}
```

```
int main(int argc, char **argv) {
  try {
    mongocxx::instance inst {};
    mongocxx::pool pool{mongocxx::uri{}};

    auto entry = pool.acquire();

    if (get_server_version(*entry) < "3.6") {
      std::cerr << "Change␣streams␣are␣only␣supported␣on␣Mongo␣versions␣>=␣3.6." << std::
          endl;
      // CXX-1548: Should return EXIT_FAILURE, but Travis is currently running Mongo 3.4
      return EXIT_SUCCESS;
    }

    mongocxx::options::change_stream options;
    const std::chrono::milliseconds await_time{1000};
    options.max_await_time(await_time);
    options.full_document("updateLookup");

    auto collection = (*entry)["modules"]["propagation"];
    mongocxx::change_stream stream = collection.watch(options);

    while (true) {
      for (const auto& event : stream) {
        //std::cout << bsoncxx::to_json(event) << std::endl;
        auto operation = event["operationType"].get_utf8().value;
        if(operation.compare("insert") == 0) {
          auto doc = event["fullDocument"];
          auto function = doc["function"].get_utf8().value;
          //std::cout << collection << std::endl;
          if(function.compare("radiohorizon") == 0) {
            std::future<void> async = std::async(func, collection, doc);
          }
        }
      }
    }

  } catch (const std::exception& exception) {
    std::cerr << "Caught␣exception␣\"" << exception.what() << "\"" << std::endl;
  } catch (...) {
    std::cerr << "Caught␣unknown␣exception␣type" << std::endl;
  }
  return EXIT_SUCCESS;
}
```

3 pathloss.bin.cpp

```cpp
#include <exception>
#include <iostream>
#include <args.hxx>
#include <softroles/propagation/pathloss.hpp>

int main(int argc, char **argv) {
  args::ArgumentParser parser("radio wave propagation path loss", "");
  args::HelpFlag help(parser, "help", "displays this help menu", {'h', "help"});
  args::Group group1(parser, "flag arguments:", args::Group::Validators::AllOrNone);
  args::ValueFlag<float> freq1(group1, "freq", "frequency in MHz", {'f', "freq"});
  args::ValueFlag<float> dist1(group1, "dist", "distance in km", {'d', "dist"});
  args::Group group2(parser, "positional arguments:", args::Group::Validators::AllOrNone)
      ;
  args::Positional<float> freq2(group2, "freq", "");
  args::Positional<float> dist2(group2, "dist", "");
  try
  {
    parser.ParseCLI(argc, argv);
  }
  catch (args::Help)
  {
    std::cout << parser;
    return 0;
  }
  catch (args::ParseError e)
  {
    std::cerr << e.what() << std::endl;
    std::cerr << parser;
    return 1;
  }
  catch (args::ValidationError e)
  {
    std::cerr << "Usage:" << e.what() << std::endl;
    std::cerr << parser;
    return 1;
  }

  if(freq2 && dist2) {
    std::cout << softroles::propagation::pathloss(args::get(freq2), args::get(dist2)) <<
        std::endl;
  }
  else if(freq1 && dist1) {
    std::cout << softroles::propagation::pathloss(args::get(freq1), args::get(dist1)) <<
        std::endl;
  }
  else {
    std::cerr << parser;
  }
  return 0;
}
```

4 pathloss.cpp

```cpp
#include <exception>
#include <math.h>
#include "../include/pathloss.hpp"

float softroles::propagation::pathloss(float f, float d) {
  try {
    return 32.44 + 20*std::log10(f*d);
  }
  catch(std::exception& e) {
    throw;
  }
}
```

5 pathloss.service.cpp

```cpp
#include <exception>
#include <iostream>
#include <string>
#include <propagation/include/pathloss.hpp>
#include <rapidjson/document.h>

int main(int argc, char **argv) {
  static const char* kTypeNames[] =
  { "Null", "False", "True", "Object", "Array", "String", "Number" };
  std::string input;
  rapidjson::Document document;
  while (true) {
    std::getline(std::cin, input);
    document.Parse(input.c_str());
    std::cout << document.IsObject() << "\n" << std::flush;
    if(document.IsObject() &&
        document.HasMember("freq") && document["freq"].IsNumber() &&
        document.HasMember("dist") && document["dist"].IsNumber()) {
      std::cout << softroles::propagation::pathloss(
                  document["freq"].GetDouble(),
                  document["dist"].GetDouble()) << std::endl;
      //for (auto& m : document.GetObject())
      //printf("Type of member %s is %s\n",
      //     m.name.GetString(), kTypeNames[m.value.GetType()]);
    }
  }

  return 0;
}
```

6 radiohorizon.bin.cpp

```cpp
#include <exception>
#include <iostream>
#include <args.hxx>
#include <softroles/propagation/radiohorizon.hpp>

int main(int argc, char **argv) {
  args::ArgumentParser parser("horizon␣distance␣for␣radio␣waves.", "");
  args::HelpFlag help(parser, "help", "Displays␣this␣help␣menu", {'h', "help"});
  args::Group group1(parser, "flag␣arguments:", args::Group::Validators::AllOrNone);
  args::ValueFlag<float> height1(group1, "height", "antenna␣height␣in␣[m]", {'h', "height
      "});
  args::Group group2(parser, "positional␣arguments:", args::Group::Validators::AllOrNone)
      ;
  args::Positional<float> height2(group2, "height", "");
  try
  {
    parser.ParseCLI(argc, argv);
  }
  catch (args::Help)
  {
    std::cout << parser;
    return 0;
  }
  catch (args::ParseError e)
  {
    std::cerr << e.what() << std::endl;
    std::cerr << parser;
    return 1;
  }
  catch (args::ValidationError e)
  {
    std::cerr << "Usage:" << e.what() << std::endl;
    std::cerr << parser;
    return 1;
  }

  if(height1) {
    std::cout << softroles::propagation::radiohorizon(args::get(height1)) << std::endl;
  }
  else if(height2) {
    std::cout << softroles::propagation::radiohorizon(args::get(height2)) << std::endl;
  }
  else {
    std::cerr << parser;
  }
  return 0;
}
```

7 radiohorizon.cpp

```cpp
#include <exception>
#include <math.h>
#include "../include/radiohorizon.hpp"

float softroles::propagation::radiohorizon(float h) {
  try {
    return 4.12 * std::sqrt(h);
  }
  catch(std::exception& e) {
    throw;
  }
}
```