
Яндекс

Яндекс Почта

OZO — асинхронная типобезопасная header-only библиотека клиент PostgreSQL для C++17

#RuPostgres: Масштабирование приложений на PostgreSQL

Немного о себе

- Существуем с 2002-го года
- 9 миллионов уникальных пользователей в день
- 110 миллионов отправляемых и принимаемых сообщений в день

OZO

Что такое OZO?

Что такое OZO?

- OZO - основанная на Boost.Asio и libpq header-only асинхронная типобезопасная C++17 библиотека для взаимодействия с PostgreSQL.

Существующие решения для C++

- **libpqxx**
- **sqlpp11**
- **libpqmxx**
- **Pgfe**
- **…**

Принципы построения архитектуры OZO

- Поддержка различных вариантов асинхронности.
- Стабильность за счёт максимальной типизации.
- Производительность за счёт compile-time вычислений.
- Расширяемость за счёт концепций.
- Проработанная система ошибок.
- Покрытие unit-тестами.

Основные концепции в OZO

- **Connection** - соединение и его свойства
- **ConnectionProvider** - выдаёт соединение для исполнения запроса
- **ConnectionSource** - стратегия получения соединение
- **Query** - предоставляет текст и параметры запроса

OZO

Простой пример

“Hello OZO!”

```
boost::asio::io_context io;
auto work = boost::asio::make_work_guard(io);
std::thread t{[&]{ for(;;) io.run();}};

const auto conn_src = ozo::make_connection_info(“host=…,port=…“);
const auto conn_provider = ozo::make_connector(conn_src, io);

std::vector<std::string> res;
ozo::request(conn_provider,
    “SELECT ‘Hello OZO!’“_SQL,
    std::back_inserter(res), boost::asio::use_future);

work.reset();
assert(res.size() == 1 && res.front() == “Hello OZO!“);
```

OZO

Пример по-сложнее

Пример по-сложнее

- Получим несколько полей из таблицы.
- Передадим параметр в запросе.
- Воспользуемся пулом соединений.

Простая табличка

```
CREATE TABLE users_info(
    id          bigint  NOT NULL,
    name        text,
    amount      bigint  NOT NULL
);
```

Как получить id и name из user_info

```
const auto conn_src = ozo::make_connection_info(“host=…,port=…“);
const auto conn_pool = ozo::make_connection_pool(conn_src);
const auto conn_provider = ozo::make_connector(conn_pool, io);

const auto query = "SELECT id, name FROM users_info WHERE amount>="_SQL
+ std::int64_t(25);

ozo::rows_of<std::int64_t, std::optional<std::string>> rows;

ozo::request(conn_provider, query, ozo::into(rows), use_future).get();

for(auto& row: rows) {
    std::cout << std::get<0>(row)
              << '\t' << std::get<1>(row) << std::endl;
}
```

Создаём ConnectionProvider

```
const auto conn_src = ozo::make_connection_info(“host=…,port=…“);
const auto conn_pool = ozo::make_connection_pool(conn_src);
const auto conn_provider = ozo::make_connector(conn_pool, io);

const auto query = "SELECT id, name FROM users_info WHERE amount>="_SQL
+ std::int64_t(25);

ozo::rows_of<std::int64_t, std::optional<std::string>> rows;

ozo::request(conn_provider, query, ozo::into(rows), use_future).get();

for(auto& row: rows) {
    std::cout << std::get<0>(row)
              << '\t' << std::get<1>(row) << std::endl;
}
```

Простой запрос с параметром

```
const auto conn_src = ozo::make_connection_info(“host=…,port=…“);
const auto conn_pool = ozo::make_connection_pool(conn_src);
const auto conn_provider = ozo::make_connector(conn_pool, io);

const auto query = "SELECT id, name FROM users_info WHERE amount>="_SQL
+ std::int64_t(25);

ozo::rows_of<std::int64_t, std::optional<std::string>> rows;

ozo::request(conn_provider, query, ozo::into(rows), use_future).get();

for(auto& row: rows) {
    std::cout << std::get<0>(row)
              << '\t' << std::get<1>(row) << std::endl;
}
```

Простой запрос с параметром

```
const auto conn_src = ozo::make_connection_info("host=…,port=…");
const auto conn_pool = ozo::make_connection_pool(conn_src);
const auto conn_provider = ozo::make_connector(conn_pool, io);

const auto query = "SELECT id, name FROM users_info WHERE amount>="_SQL
+ std::int64_t(25);

ozo::rows_of<std::int64_t, std::optional<std::string>> rows;

ozo::request(conn_provider, query, ozo::into(rows), use_future).get();

for(auto& row: rows) {
    std::cout << std::get<0>(row)
              << '\t' << std::get<1>(row) << std::endl;
}
```

Будет передан как бинарный параметр

Контейнер для результата запроса

```
const auto conn_src = ozo::make_connection_info("host=…,port=…");
const auto conn_pool = ozo::make_connection_pool(conn_src);
const auto conn_provider = ozo::make_connector(conn_pool, io);

const auto query = "SELECT id, name FROM users_info WHERE amount>="_SQL
+ std::int64_t(25);

ozo::rows_of<std::int64_t, std::optional<std::string>> rows;

ozo::request(conn_provider, query, ozo::into(rows), use_future).get();

for(auto& row: rows) {
    std::cout << std::get<0>(row)
              << '\t' << std::get<1>(row) << std::endl;
}
```

Контейнер для результата запроса

```
const auto conn_src = ozo::make_connection_info("host=…,port=…");
const auto conn_pool = ozo::make_connection_pool(conn_src);
const auto conn_provider = ozo::make_connector(conn_pool, io);

const auto query = "SELECT id, name FROM users_info WHERE amount>="_SQL
+ std::int64_t(42);

using ozo::rows_of = std::vector<std::tuple<…>>;

ozo::rows_of<std::int64_t, std::optional<std::string>> rows;

ozo::request(conn_provider, query, ozo::into(rows), use_future).get();

for(auto& row: rows) {
    std::cout << std::get<0>(row)
              << '\t' << std::get<1>(row) << std::endl;
}
```

Контейнер для результата запроса

```
const auto conn_src = ozo::make_connection_info("host=…,port=…");
const auto conn_pool = ozo::make_connection_pool(conn_src);
const auto conn_provider = ozo::make_connector(conn_pool, io);

const auto query = "SELECT id, name FROM users_info WHERE amount>="_SQL
+ std::int64_t(25);

ozo::rows_of<std::int64_t, std::optional<std::string>> rows;

ozo::request(conn_provider, query, ozo::into(rows), use_future).get();

for(auto& row: rows) {
    std::cout << std::get<0>(row)
              << '\t' << std::get<1>(row) << std::endl;
}
```

Порядок параметров важен!

Контейнер для результата запроса

```
const auto conn_src = ozo::make_connection_info("host=…,port=…");
const auto conn_pool = ozo::make_connection_pool(conn_src);
const auto conn_provider = ozo::make_connector(conn_pool, io);

const auto query = "SELECT id, name FROM users_info WHERE amount>=" SQL
+ std::int64_t(25);

ozo::rows_of<std::int64_t, std::optional<std::string>> rows;

ozo::request(conn_provider, query, ozo::into(rows), use_future).get();

for(auto& row: rows) {
    std::cout << std::get<0>(row)
              << '\t' << std::get<1>(row) << std::endl;
}
```

Поле “name” может быть NULL

Отправляем запрос в базу и ждём ответ

```
const auto conn_src = ozo::make_connection_info("host=…,port=…");
const auto conn_pool = ozo::make_connection_pool(conn_src);
const auto conn_provider = ozo::make_connector(conn_pool, io);

const auto query = "SELECT id, name FROM users_info WHERE amount>="_SQL
+ std::int64_t(25);

ozo::rows_of<std::int64_t, std::optional<std::string>> rows;

ozo::request(conn_provider, query, ozo::into(rows), use_future).get();

for(auto& row: rows) {
    std::cout << std::get<0>(row)
              << '\t' << std::get<1>(row) << std::endl;
}
```

Обрабатываем результат

```
const auto conn_src = ozo::make_connection_info("host=…,port=…");
const auto conn_pool = ozo::make_connection_pool(conn_src);
const auto conn_provider = ozo::make_connector(conn_pool, io);

const auto query = "SELECT id, name FROM users_info WHERE amount>="_SQL
+ std::int64_t(25);

ozo::rows_of<std::int64_t, std::optional<std::string>> rows;

ozo::request(conn_provider, query, ozo::into(rows), use_future).get();

for(auto& row: rows) {
    std::cout << std::get<0>(row)
              << '\t' << std::get<1>(row) << std::endl;
}
```

OZO

Несколько запросов в одном соединении

Несколько запросов в одном соединении

```
auto conn = ozo::request(conn_provider, query1, ozo::into(rows1),
use_future).get();

// . . .

ozo::request(conn, query2, ozo::into(rows2), use_future).get();
```

OZO

Как не следить за порядком полей в запросе

Контейнер для результата запроса

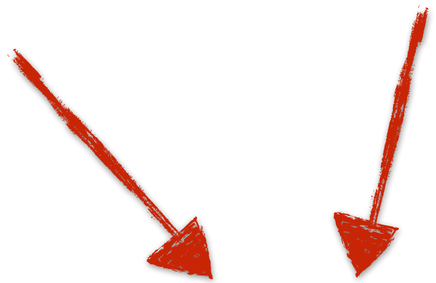
```
const auto conn_src = ozo::make_connection_info("host=…,port=…");
const auto conn_pool = ozo::make_connection_pool(conn_src);
const auto conn_provider = ozo::make_connector(conn_pool, io);

const auto query = "SELECT id, name FROM users_info WHERE amount>="_SQL
+ std::int64_t(25);

ozo::rows_of<std::int64_t, std::optional<std::string>> rows;

ozo::request(conn_provider, query, ozo::into(rows), use_future).get();

for(auto& row: rows) {
    std::cout << std::get<0>(row)
              << '\t' << std::get<1>(row) << std::endl;
}
```

Можно адаптировать структуру Boost.Fusion

```
BOOST_FUSION_DEFINE_STRUCT(
    (), row_type,
    (std::optional<std::string>>, name)
    (std::int64_t, id)
);
```

Имена полей в структуре

```
BOOST_FUSION_DEFINE_STRUCT(
    (), row_type,
    (std::optional<std::string>>, name)
    (std::int64_t, id)
);
```

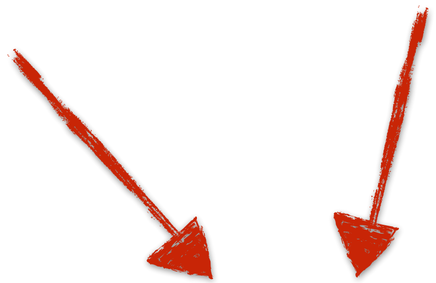


Порядок параметров не важен!

```
const auto query = "SELECT id, name FROM users_info WHERE amount>="_SQL
+ std::int64_t(25);
```

Имена полей в структуре

```
BOOST_FUSION_DEFINE_STRUCT(
    (), row_type,
    (std::optional<std::string>>, name)
    (std::int64_t, id)
);
```



Но теперь важно имя поля! :)

```
const auto query = "SELECT id, name FROM users_info WHERE amount>="_SQL
+ std::int64_t(25);
```

Контейнер для результата запроса

```
const auto conn_src = ozo::make_connection_info(“host=…,port=…“);
const auto conn_pool = ozo::make_connection_pool(conn_src);
const auto conn_provider = ozo::make_connector(conn_pool, io);

const auto query = "SELECT id, name FROM users_info WHERE amount>="_SQL
+ std::int64_t(25);

std::vector<row_type> rows;

ozo::request(conn_provider, query, ozo::into(rows), use_future).get();

for(auto& row: rows) {
    std::cout << row.id << ‘\t' << row.name << std::endl;
}
```

OZO

Как не ждать

Поддерживаемые модели асинхронного кода

- Callback.
- Boost.Coroutine.
- std::future.
- Всё что поддерживает boost::asio::async_completion.

Callback

```
auto rows = std::make_shared<std::vector<rows_type>>();

ozo::request(conn_provider, query, ozo::into(*rows),
         [rows](ozo::error_code ec, auto conn) {
    if (!ec) {
        std::cout << "id" << '\t' << "name" << std::endl;
        for (auto& row: res) {
            std::cout << row.id << '\t' << row.name << std::endl;
        }
    }

});
```

Boost.Coroutine

```
boost::asio::spawn(io, [&io](auto yield) {
    auto rows = std::vector<rows_type>();

    ozo::request(conn_provider, query, ozo::into(rows), yield);

    std::cout << "id" << '\t' << "name" << std::endl;
    for (auto& row: res) {
        std::cout << row.id << ‘\t' << row.name << std::endl;
    }
});
```

std::future

```
auto rows = std::vector<rows_type>();

auto conn = ozo::request(conn_provider, query, ozo::into(rows),
                                               boost::asio::use_future)

// . . . Do something else . . .

conn.get();

for(auto& row: rows) {
    std::cout << std::get<0>(row)
              << '\t' << std::get<1>(row) << std::endl;
}
```

OZO

Как обработать ошибки

Источники информации об ошибке

error_code - не имеет контекста, поэтому Connection предоставляет:

〉 Сообщение об ошибке от libpq.

〉 Дополнительный контекст ошибки от OZO.

Callback

```
auto rows = std::make_shared<std::vector<rows_type>>();

ozo::request(conn_provider, query, ozo::into(*rows),
        [rows](ozo::error_code ec, auto conn) {
    if (ec) {
        std::cerr << ec.message()
                  << " | " << ozo::error_message(conn)
                  << " | " << ozo::get_error_context(conn)
                  << std::endl;
        return;
    }
    //...
});
```

Boost.Coroutines

```
boost::asio::spawn(io, [&io, provider](auto yield) {
    auto rows = std::vector<rows_type>();

    ozo::error_code ec;
    auto conn = ozo::request(provider, query, ozo::into(rows),
                                                    yield[ec]);

    if (ec) {
        std::cerr << ec.message()
                  << " | " << ozo::error_message(conn)
                  << " | " << ozo::get_error_context(conn)
                  << std::endl;
    }
});
```

std::future

```
std::vector<row_type> rows;

try {
    ozo::request(provider, query, ozo::into(rows),
                                  boost::asio::use_future).get();
} catch (const ozo::system_error& e) {
    std::cerr << e.what() << std::endl;
}
```

std::future

```
std::vector<row_type> rows;

auto conn = ozo::get_connection(provider).get();
try {
    ozo::request(conn, query, ozo::into(rows),
                                    boost::asio::use_future).get();
} catch (const ozo::system_error& e) {
    std::cerr << e.what()
              << " | " << ozo::error_message(conn)
              << " | " << ozo::get_error_context(conn)
              << std::endl;
}
```

OZO

Запросы и параметры

Концепция запроса

- Для типа T отвечающего концепции должны быть определены функции:

```
QueryText get_query_text(std::declval<const T&>());

HanaTuple get_query_params(std::declval<const T&>());
```

Статический запрос с помощью query_builder

```
std::string word;
...
const auto query = (
     "SELECT number "_SQL +
       "FROM numerals "_SQL +
      "WHERE word = "_SQL + word + "::text"_SQL
).build();
```

Динамический запроса из файла

```
struct number_by_numeral_word {
    static constexpr auto name = "number by numeral word"_s;
    using parameters_type = std::tuple<std::string>;
};

struct numeral_word_by_number { /*...*/ };

const auto repo = ozo::make_query_repository<
    number_by_numeral_word,
    numeral_word_by_number
>();

const auto query = repo.make_query<number_by_numeral_word>(
                                        std::string("second"));
```

Динамический запроса из файла

```
struct number_by_numeral_word {
    static constexpr auto name = "number by numeral word"_s;
    using parameters_type = std::tuple<std::string>;
};

struct numeral_word_by_number { /*...*/ };

const auto repo = ozo::make_query_repository<
    number_by_numeral_word,
    numeral_word_by_number
>();

const auto query = repo.make_query<number_by_numeral_word>(
                                        std::string("second"));
```

Файл конфигурации запросов

```
-- name: number by numeral word
SELECT number
  FROM numerals
 WHERE word = :word::text
-- name: numeral word by number
SELECT word
  FROM numerals
 WHERE number = :number::bigint
```

Динамический запроса из файла

```
struct number_by_numeral_word {
    static constexpr auto name = "number by numeral word"_s;
    using parameters_type = std::tuple<std::string>;
};

struct numeral_word_by_number { /*...*/ };

const auto repo = ozo::make_query_repository<
    number_by_numeral_word,
    numeral_word_by_number
>();

const auto query = repo.make_query<number_by_numeral_word>(
                                        std::string("second"));
```

Динамический запроса из файла

```
struct number_by_numeral_word {
    static constexpr auto name = "number by numeral word"_s;
    using parameters_type = std::tuple<std::string>;
};

struct numeral_word_by_number { /*...*/ };

const auto repo = ozo::make_query_repository<
    number_by_numeral_word,
    numeral_word_by_number
>();

const auto query = repo.make_query<number_by_numeral_word>(
                                        std::string("second"));
```

Преобразуется в бинарное представление

```
const auto conn_src = ozo::make_connection_info(“host=…,port=…“);
const auto conn_pool = ozo::make_connection_pool(conn_src);
const auto conn_provider = ozo::make_connector(conn_pool, io);

const auto query = "SELECT id, name FROM users_info WHERE amount>="_SQL
+ std::int64_t(25);

ozo::rows_of<std::int64_t, std::optional<std::string>> rows;

ozo::request(conn_provider, query, ozo::into(rows), use_future).get();

for(auto& row: rows) {
    std::cout << std::get<0>(row)
              << '\t' << std::get<1>(row) << std::endl;
}
```

OZO

Система типов и интроспекция

Система типов

〉 Поддерживает основные встроенные типы через стандартные C++ типы.

〉 Поддерживает расширение встроенных типов за счёт стандартных и пользовательских типов.

〉 Поддерживает расширение пользовательскими типами.

Расширение системы типов

```
OZO_PG_DEFINE_TYPE_AND_ARRAY(
    boost::uuids::uuid, "uuid", UUIDOID, 2951, bytes<16>
)
```

Расширение системы типов

```
OZO_PG_DEFINE_TYPE_AND_ARRAY(
    boost::uuids::uuid, "uuid", UUIDOID, 2951, bytes<16>
)
```

Тип в C++

Расширение системы типов

```
OZO_PG_DEFINE_TYPE_AND_ARRAY(
    boost::uuids::uuid, "uuid", UUIDOID, 2951, bytes<16>
)
```

Имя типа в PostgreSQL

Расширение системы типов

```
OZO_PG_DEFINE_TYPE_AND_ARRAY(
    boost::uuids::uuid, "uuid" UUIDOID, 2951, bytes<16>
)
```

OID типа в PostgreSQL:

OID - для встроенного типа

`ozo::null_oid` - для пользовательского

Расширение системы типов

```
OZO_PG_DEFINE_TYPE_AND_ARRAY(
    boost::uuids::uuid, "uuid", UUIDOID, 2951, bytes<16>
)
```

OID массива типа в PostgreSQL:

OID - для встроенного типа

`ozo::null_oid` - для пользовательского

Расширение системы типов

```
OZO_PG_DEFINE_TYPE_AND_ARRAY(
    boost::uuids::uuid, "uuid", UUIDOID, 2951, bytes<16>
)
```

Размер типа:

`bytes<N>` - в байтах для типов с фиксированным размером

`dynamic_size` - для типов с динамическим размером

Кастомизация сереализации типа

- Нужно переопределить шаблон по-умолчанию

```
template <typename In, typename = std::void_t<>>
struct send_impl{
    template <typename M>
    static ostream& apply(ostream& out, const oid_map_t<M>&,
                                                const In& in) {

        return write(out, in);
    }
};
```

Кастомизация десереализации типа

- Нужно переопределить шаблон по-умолчанию

```
template <typename Out, typename = std::void_t<>>
struct recv_impl{
    template <typename M>
    static istream& apply(istream& in, int32_t size,
                                       const oid_map_t<M>&, Out& out) {

        if constexpr (is_dynamic_size<Out>::value) {
            out.resize(size);
        }
        return read(in, out);
    }
};
```

OZO

Что предоставляет библиотека

OZO для пользователя

- Асинхронный API через callback, std::future, Boost.Coroutine и пр.
- Интроспекция типов с поддержкой Boost.Fusion, Boost.Hana.
- Type-safe бинарный протокол.
- Система кодов ошибок построенная на error_code/error_category.
- Compile-time и run-time конструкторы запросов.
- Таймауты, транзакции, поддержка ретраев.
- Compile-time конфигурируемая статистика.
- Расширяемость.

OZO для контрибьютера

- Unit-тесты.
- Интеграционные тесты.
- Общее покрытие тестами порядка 98%.
- Docker-образы для сборки и запуска тестов.

Ближайшие планы

- **Документация** - работаем над ней и будем улучшать.
- **Prepare statement** - будем делать.
- **Интроспекция композитов** - в работе.
- **Фреймворк для ретраев** - будем делать.
- **Система сбора статистики** - будем делать.
- **Построчное получение результата** - будем делать.
- **Поддержка нотификаций** - будем делать.

Контакты

- Сергей Хандриков
- **Email**: thed636@yandex-team.ru
- **GitHub**: https://github.com/yandex/ozo
- **Документация**: https://yandex.github.io/ozo/