
Яндекс

Яндекс

Яндекс Такси

Шаблоны C++ и базы данных

Сергей Федоров, ведущий разработчик

План доклада или о чём вот это всё

00 | Зачем писать свой драйвер?

10 | Чтение и запись буферов полей

20 | Работа с записями БД

30 | Работа с набором данных

40 | Как это поможет мне?

50 | Вопросы и ответы

05 | API с «человеческим» лицом

Что потрогаем

00 | Первичный шаблон без определения

10 | SFINAE для специализаций шаблонов

20 | Variadic templates

30 | Fold expressions

40 | Constexpr функции

50 | If constexpr

Зачем это всё?

00

Зачем писать свой драйвер?

Зачем писать свой драйвер?

1. Это красиво
2. Асинхронно
3. Prepared statements
4. Бинарный протокол

Зачем писать свой драйвер?

05

API с «человеческим» лицом

Отправка запросов
Обработка ответов

Исполнение запросов

```
auto trx = cluster.Begin({});
trx.Execute("insert into foobar (foo, bar) values ($1, $2)", 42, "baz");
trx.Commit();
```

Работа с результатами запроса

```
auto trx = cluster.Begin();
auto res = trx.Execute("select foo, bar from foobar where foo = $1", 42);
for (auto row : res) {
  // process row
  for (auto field : row) {
    // process field
  }
}
trx.Commit();
```

Работа с результатами запроса

```
auto trx = cluster.Begin();
auto res = trx.Execute("select foo, bar from foobar where foo = $1", 42);
for (auto row : res) {
  // process row
  auto foo = row[0].As<int>();
  auto bar = row[1].As<std::string>();
}
trx.Commit();
```

Работа с результатами запроса

```
auto trx = cluster.Begin();
auto res = trx.Execute("select foo, bar from foobar where foo = $1", 42);
for (auto row : res) {
  // process row
  auto [foo, bar] = row.As<int, std::string>();
}
trx.Commit();
```

Работа с результатами запроса

```
struct FooBar { int foo; std::string bar; }; // RFC 3092
auto trx = cluster.Begin();
auto res = trx.Execute("select foo, bar from foobar where foo = $1", 42);
for (auto row : res) {
  // process row
  auto foobar = row.As<FooBar>();
}
trx.Commit();
```

Работа с результатами запроса

```
struct FooBar { int foo; std::string bar; }; // RFC 3092
auto trx = cluster.Begin();
auto res = trx.Execute("select foo, bar from foobar where foo = $1", 42);
auto foobars = res.AsContainer<std::vector<FooBar>>();
trx.Commit();
```

Как всё это работает?

10

Чтение и запись буферов отдельных полей

- fold expression и запись
- Замена tag switching на if constexpr
- Определение наличия специализации
- SFINAE для специализаций

11

Запись аргументов запроса

Отправка запроса

```
auto trx = cluster.Begin({});
trx.Execute("insert into foobar (foo, bar) values ($1, $2)", 42, "baz");
trx.Commit();
```

Запись параметров запроса

```cpp
/// Execute statement with arbitrary parameters
/// Suspends coroutine for execution
/// @throws NotInTransaction, SyntaxError, ConstraintViolation,
/// InvalidParameterType
template <typename... Args>
ResultSet Transaction::Execute(const std::string& statement, Args const&... args) {
  detail::QueryParameters params;
  if constexpr (sizeof...(Args) > 0) {
    params.Write(GetConnectionUserTypes(), args...);
  }
  return DoExecute(statement, params, {});
}
```

Запись параметров запроса

```
template <typename... T>
void QueryParameters::Write(const UserTypes& types, const T&... args) {
  (Write(types, args), ...);
}
```

Запись параметров запроса

```
template <typename T>
void QueryParameters::Write(const UserTypes& types, const T& arg) {
  static_assert(io::traits::kIsMappedToPg<T>,
                "Type doesn't have a mapping to Postgres type");
  WriteParamType(types, arg);
  WriteNullable(types, arg, io::traits::IsNullable<T>{});
}

template <typename T>
void QueryParameters::WriteNullable(const UserTypes& types, const T& arg, std::true_type) {
  using NullDetector = io::traits::GetSetNull<T>;
  if (NullDetector::IsNull(arg)) {
    WriteNull();
  } else {
    WriteData(types, arg);
  }
}

template <typename T>
void QueryParameters::WriteNullable(const UserTypes& types, const T& arg, std::false_type) {
  WriteData(types, arg);
}
```

if constexpr

```
template <typename T>
void QueryParameters::Write(const UserTypes& types, const T& arg) {
  static_assert(io::traits::kIsMappedToPg<T>,
                "Type doesn't have a mapping to Postgres type");
  WriteParamType(types, arg);
  if constexpr (io::traits::kIsNullable<T>) {
    using NullDetector = io::traits::GetSetNull<T>;
    if (NullDetector::IsNull(arg)) {
      WriteNull();
      return;
    }
  }
  WriteData(types, arg);
}
```

15

Чтение записи по ~~слогам~~ полям

Чтение полей в переменные

```
auto trx = cluster.Begin({});
auto res = trx.Execute("select foo, bar from foobar where foo >= $1", 42);
for (auto row : res) {
  auto foo = row[0].As<int>();
  auto bar = row[1].As<std::string>();
}
trx.Commit();
```

Чтение полей в переменные

```
auto trx = cluster.Begin({});
auto res = trx.Execute("select foo, bar from foobar where foo >= $1", 42);
for (auto row : res) {
  int foo;
  std::string bar;
  row[0].To(foo);
  row[1].To(bar);
}
trx.Commit();
```

Чтение буфера поля записи

```cpp
/// Read the field's buffer into user-provided variable.
/// @throws FieldValueIsNull If the field is null and the C++ type is
///                          not nullable.
template <typename T>
void Field::To(T&& val) const {
  using ValueType = typename std::decay<T>::type;
  auto fb = GetBuffer();
  if (fb.is_null) {
    if constexpr (io::traits::kIsNullable<ValueType>) {
      using NullSetter = io::traits::GetSetNull<ValueType>;
      NullSetter::SetNull(val);
    } else {
      throw FieldValueIsNull{field_index_};
    }
  } else {
    Read(fb, std::forward<T>(val));
  }
}
```

Чтение буфера поля записи

```
template <typename T>
void Field::Read(const io::FieldBuffer& buffer, T&& val) const {
  using ValueType = typename std::decay<T>::type;
  static_assert(io::traits::kHasAnyParser<ValueType>,
                "Type doesn't have any parsers defined");
  if (buffer.format == io::DataFormat::kTextDataFormat) {
    ReadText(buffer, std::forward<T>(val));
  } else {
    ReadBinary(buffer, std::forward<T>(val));
  }
}
```

Чтение буфера поля записи

```
template <typename T>
void Field::ReadBinary(const io::FieldBuffer& buffer, T&& val) const {
  using ValueType = typename std::decay<T>::type;
  if constexpr (io::traits::kHasBinaryParser<ValueType>) {
    io::ReadBinary(buffer, std::forward<T>(val));
  } else {
    throw NoValueParser{::utils::GetTypeName<T>(),
                        io::DataFormat::kBinaryDataFormat};
  }
}
```

Чтение буфера поля записи

```
template <typename T>
void ReadBinary(const FieldBuffer& buffer, T&& value) {
  using ValueType = std::decay_t<T>;
  static_assert( traits::kHasBinaryParser<ValueType>,
                 "Type doesn't have a binary parser");
  ReadBuffer<DataFormat::kBinaryDataFormat>(buffer, std::forward<T>(value));
}

template <DataFormat F, typename T>
void ReadBuffer(const FieldBuffer& buffer, T&& value) {
  using ValueType = std::decay_t<T>;
  static_assert((traits::kHasParser<ValueType, F>),
                 "Type doesn't have an appropriate parser");
  using BufferReader = typename traits::IO<ValueType, F>::ParserType;
  BufferReader{std::forward<T>(value)}(buffer);
}
```

16

Система «свойств» (traits)

IsNullable

```
template <typename T>
struct IsNullable : std::false_type {};

template <typename T>
constexpr bool kIsNullable = IsNullable<T>::value;

template <typename T>
struct GetSetNull {
  inline static bool IsNull(const T&) { return false; }
  inline static void SetNull(T&) {
    throw TypeCannotBeNull(::utils::GetTypeName<T>());
  }
};
```

IsNullable

```
template <typename T>
struct IsNullable<std::optional<T>> : std::true_type {};

template <typename T>
struct GetSetNull<std::optional<T>> {
  using ValueType = std::optional<T>;
  inline static bool IsNull(const ValueType& v) { return !!v; }
  inline static void SetNull(ValueType& v) { ValueType().swap(v); }
};
```

Рабочие лошадки

```
/// @brief Primary template for Postgre buffer parser.
/// Specialisations must provide call operators that parse FieldBuffer.
template <typename T, DataFormat, typename Enable = ::utils::void_t<>>
struct BufferParser;

/// @brief Primary template for Postgre buffer formatter
/// Specialisations must provide call operators that write to a buffer.
template <typename T, DataFormat, typename Enable = ::utils::void_t<>>
struct BufferFormatter;
```

Рабочие лошадки

```
namespace traits {
template <typename T, DataFormat F, typename Enable = ::utils::void_t<>>
struct Input {
  using type = BufferParser<T, F>;
};

template <typename T, DataFormat F, typename Enable = ::utils::void_t<>>
struct Output {
  using type = BufferFormatter<T, F>;
};

template <typename T, DataFormat F>
struct IO {
  using ParserType = typename Input<T, F>::type;
  using FormatterType = typename Output<T, F>::type;
};
}  // namespace traits
```

Вспомогательные свойства

```
template <typename T, DataFormat Format>
struct HasParser : utils::IsDeclComplete<typename IO<T, Format>::ParserType> {};

template <typename T, DataFormat Format>
struct HasFormatter
    : utils::IsDeclComplete<typename IO<T, Format>::FormatterType> {};

template <typename T, DataFormat F>
struct CustomParserDefined : utils::IsDeclComplete<BufferParser<T, F>> {};

template <typename T>
using CustomBinaryParserDefined =
    CustomParserDefined<T, DataFormat::kBinaryDataFormat>;

template <typename T>
constexpr bool kCustomBinaryParserDefined = CustomBinaryParserDefined<T>::value;
```

Чёрная магия

```
namespace detail {
template <typename T, std::size_t = sizeof(T)>
std::true_type IsCompleteImpl(T*);

std::false_type IsCompleteImpl(...);
}  // namespace detail

template <typename T>
using IsDeclComplete = decltype(detail::IsCompleteImpl(std::declval<T*>()));
```

17

Примеры

Специализации для bool

```
template <>
struct BufferParser<bool, DataFormat::kBinaryDataFormat> {
  bool& value;
  explicit BufferParser(bool& val) : value{val} {}
  void operator()(const FieldBuffer& buf) {
    if (buf.length != 1) {
      throw InvalidInputBufferSize{buf.length, "for boolean type"};
    }
    value = *buf.buffer != 0;
  }
};
template <>
struct BufferFormatter<bool, DataFormat::kBinaryDataFormat> {
  bool value;
  explicit BufferFormatter(bool val) : value(val) {}
  template <typename Buffer>
  void operator()(const UserTypes&, Buffer& buf) const {
    buf.push_back(value ? 1 : 0);
  }
};
```

Специализация ввода/вывода для массивов

```
template <typename Container>
struct ArrayBinaryParser; // contents skipped
template <typename Container>
struct ArrayBinaryFormatter; // contents skipped

namespace traits {
template <typename T>
struct Input<T, DataFormat::kBinaryDataFormat,
             std::enable_if_t<!detail::kCustomBinaryParserDefined<T> &&
                              io::detail::kEnableArrayParser<T>>> {
  using type = io::detail::ArrayBinaryParser<T>;
};

template <typename T>
struct Output<T, DataFormat::kBinaryDataFormat,
              std::enable_if_t<!detail::kCustomBinaryFormatterDefined<T> &&
                               io::detail::kEnableArrayFormatter<T>>> {
  using type = io::detail::ArrayBinaryFormatter<T>;
};
```

20

Работа с записями БД

- Вычисление аргументов шаблонного аргумента шаблона
- fold expressions и чтение

Варианты использования

```
auto [foo, bar] = row.As<int, std::string>(); // 1
row.To(foo, bar); // 2
auto foobar = row.As<FooBar>(); // 3
row.To(foobar); // 4
```

Реализация мечты

```
class Row {
 public:
  template <typename T>
  void To(T&& val) const; // 1
  template <typename T>
  void To(T&& val, RowTag) const; // 2
  template <typename T>
  void To(T&& val, FieldTag) const; // 3

  template <typename... T>
  void To(T&&... val) const; // 4
};
```

Реализация мечты

```
template <typename... T>
void Row::To(T&&... val) const {
  if (sizeof...(T) > Size()) {
    throw InvalidTupleSizeRequested(Size(), sizeof...(T));
  }
  detail::RowDataExtractor<T...>::ExtractValues(*this, std::forward<T>(val)...);
}

template <typename T>
void Row::To(T&& val, RowTag) const {
  using ValueType = std::decay_t<T>;
  static_assert(io::traits::kIsRowType<ValueType>,
                "This type cannot be used as a row type");
  using RowType = io::RowType<ValueType>;
  using TupleType = typename RowType::TupleType;
  detail::TupleDataExtractor<TupleType>::ExtractTuple(
      *this, RowType::GetTuple(std::forward<T>(val)));
}
```

Реализация мечты

```
template <typename IndexTuple, typename... T>
struct RowDataExtractorBase;

template <std::size_t... Indexes, typename... T>
struct RowDataExtractorBase<std::index_sequence<Indexes...>, T...> {
  static void ExtractValues(const Row& row, T&&... val) {
    (row[Indexes].To(std::forward<T>(val)), ...);
  }
  static void ExtractTuple(const Row& row, std::tuple<T...>& val) {
    (row[Indexes].To(std::get<Indexes>(val)), ...);
  }
  static void ExtractTuple(const Row& row, std::tuple<T...>&& val) {
    (row[Indexes].To(std::get<Indexes>(val)), ...);
  }
};
```

Реализация мечты

```
template <typename... T>
struct RowDataExtractor
    : RowDataExtractorBase<std::index_sequence_for<T...>, T...> {};

template <typename T>
struct TupleDataExtractor;
template <typename... T>
struct TupleDataExtractor<std::tuple<T...>>
    : RowDataExtractorBase<std::index_sequence_for<T...>, T...> {};
```

21

Всё — tuple

(но это не точно)

Так откуда же tuple?

```
template <typename T>
void Row::To(T&& val, RowTag) const {
  using ValueType = std::decay_t<T>;
  static_assert(io::traits::kIsRowType<ValueType>,
                "This type cannot be used as a row type");
  using RowType = io::RowType<ValueType>;
  using TupleType = typename RowType::TupleType;
  detail::TupleDataExtractor<TupleType>::ExtractTuple(
      *this, RowType::GetTuple(std::forward<T>(val)));
}
```

RowType

```
template <typename T>
struct RowType : detail::RowTypeImpl<T, traits::kRowCategory<T>> {};
```

std::tuple — тратим меньше сил

```
template <typename T>
struct RowTypeImpl<T, traits::RowCategoryType::kTuple> {
  using ValueType = T;
  using TupleType = T;

  static TupleType& GetTuple(ValueType& v) { return v; }
  static const TupleType& GetTuple(const ValueType& v) { return v; }
};
```

Интрузивный метод

```
class FooClass {
  int foo;
  std::string bar;
public:
  auto Introspect() {
    return std::tie(foo, bar);
  }
};

namespace io {

template <typename T>
struct RowTypeImpl<T, traits::RowCategoryType::kIntrusiveIntrospection> {
  using ValueType = T;
  static auto GetTuple(ValueType& v) { return v.Introspect(); }
};

}  // namespace io
```

Волшебный метод

```cpp
#include <boost/pfr/precise.hpp>

struct FooBar {
  int foo;
  std::string bar;
};

namespace io {

template <typename T>
struct RowTypeImpl<T, traits::RowCategoryType::kAggregate> {
  using ValueType = T;
  static auto GetTuple(ValueType& v) {
    return boost::pfr::structure_tie(v);
  }
};

}  // namespace io
```

https://github.com/apolukhin/magic_get

30

Работа с набором данных

if constexpr для особых случаев

И чтобы совсем красиво

```
struct FooBar { int foo; std::string bar; }; // RFC 3092
auto trx = cluster.Begin();
auto res = trx.Execute("select foo, bar from foobar where foo = $1", 42);
auto foobars = res.AsContainer<std::vector<FooBar>>();
trx.Commit();
```

Особый шаг

```
template <typename Container>
Container ResultSet::AsContainer() const {
  using ValueType = typename Container::value_type;
  Container c;
  if constexpr (io::traits::kCanReserve<Container>) {
    c.reserve(Size());
  }
  auto res = AsSetOf<ValueType>();
  std::copy(res.begin(), res.end(), io::traits::Inserter(c));
  return c;
}
```

40

Как это поможет мне?

Рецепты

Рецепты

Система парсеров	Система «свойств» типов
Tag switching	if constexpr
Variadic recursion	fold expression
Вычислить параметр шаблона	Первичный шаблон без специализации
Особое действие для типа	Свойство и if constexpr
Наличие специализации	Магия

50

Вопросы и, возможно, ответы

Вопросы
Возможно ответы

Яндекс

Яндекс Такси

Спасибо

Сергей Федоров

Ведущий разработчик

 ser-fedorov@yandex-team.ru

 @zmij_r

 https://github.com/zmij