Яндекс Такси

Итоги встречи в Кёльне

Полухин Антон

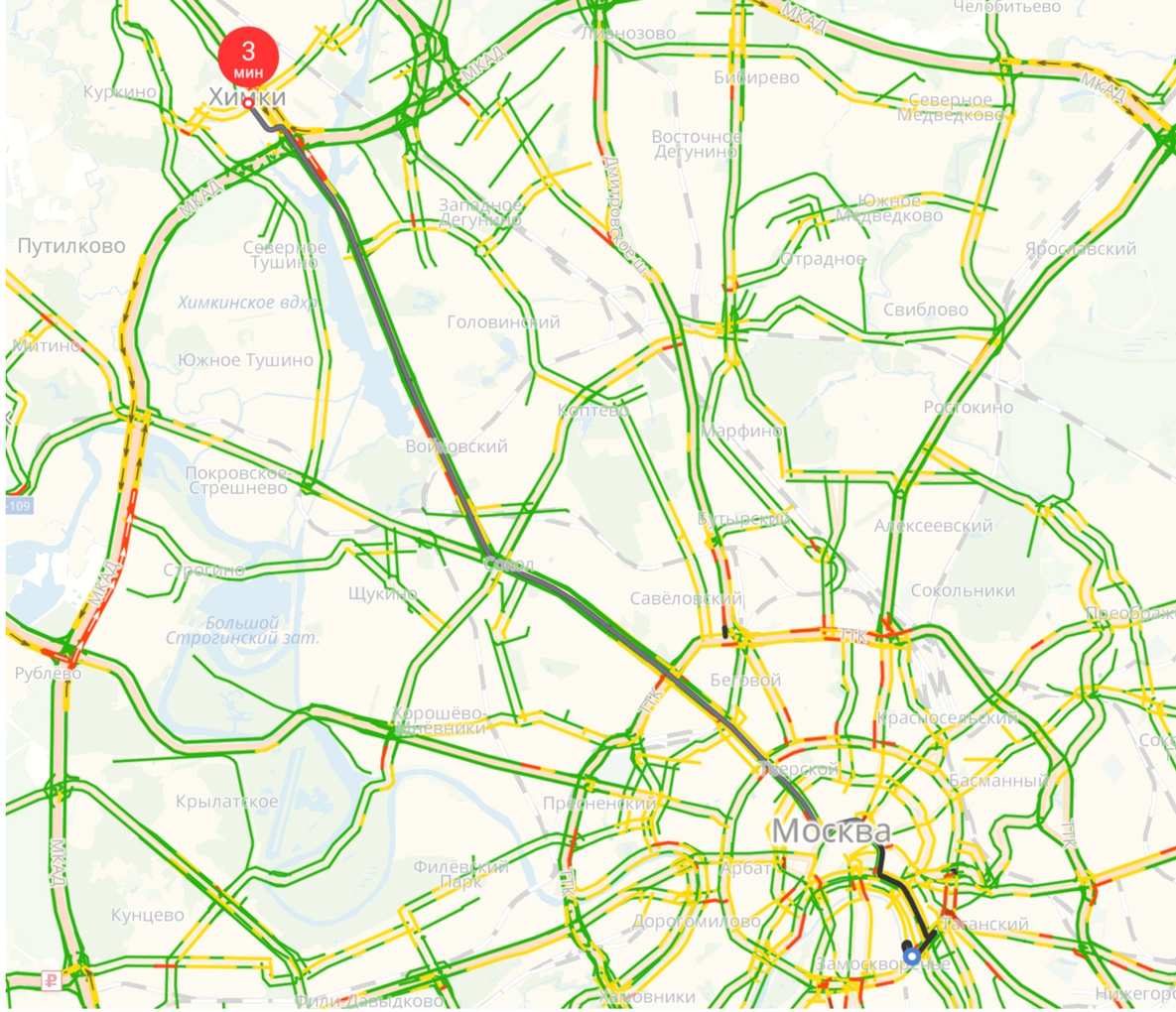
Antony Polukhin

Яндекс Такси


BISTRO

Содержание

- Контракты
- std::format
- std::jthread
- Что дальше?



Контракты?

~~Контракты~~!

P0542R5 vs P1607R0

P0542:

`{default, audit, axiom} * {continuation on, continuation off} * {build modes...}`

P0542R5 vs P1607R0

P0542:

`{default, audit, axiom} * {continuation on, continuation off} * {build modes...}`

P1607:

`{ignore, assume, check_never_continue, check_maybe_continue}`

P1607

```
#if defined(BUILD_LEVEL_DEBUG)
#  define AUDIT_FOO check_maybe_continue
#else
#  define AUDIT_FOO assume
#endif

void foo(int i)
  [[pre AUDIT_FOO: i > 10]]
;
```

P0542

```
void foo(int i)
  [[pre audit: i > 10]]
;
```

P0542

```
void foo(int i)
  [[pre audit: i > 10]] // это в заголовочном файле
;
```

P0542

```
void foo(int i)
  [[pre audit: i > 10]] // библиотека собрана с одним уровнем, ваш проект - с другим
;
```

P0542

```
void foo(int i)
  [[pre audit: i > 10]] // библиотека собрана с одним уровнем, ваш проект - с другим
;                       // conditionally supported
```

P0542

```
void foo(int i) // libfoo
  [[pre audit: i > 10]] // это в заголовочном файле
;

void used_by_foo(int i) // libused_by_foo
  [[pre audit: is_prime(i)]] // это в заголовочном файле другой библиотеки
;
```

P0542

```
void foo(int i)
  [[pre audit: i > 10]] // хочется проверять
;

void used_by_foo(int i)
  [[pre audit: is_prime(i)]] // хочется отключить
;
```

P0542

```
void foo(int i)
  [[pre audit: i > 10]] // хочется проверять
;

void used_by_foo(int i)
  [[pre audit: is_prime(i)]] // хочется отключить
;

// ...
#include <foo>
#include <used_by_foo>
```

P1607

```
void foo(int i)
  [[pre AUDIT_FOO: i > 10]] // хочется проверять
;

void used_by_foo(int i)
  [[pre AUDIT_USED_BY_FOO: is_prime(i)]] // хочется отключить
;

// ...
#include <foo>
#include <used_by_foo>
```

P0542R5 vs P1607R0

P0542:

```
{default, audit, axiom} * {continuation on, continuation off} * {build modes...}
```

P1607:

```
{ignore, assume, check_never_continue, check_maybe_continue}
```

P0542R5 vs P1607R0

P0542:

`{default, audit, axiom} * {continuation on, continuation off} * {build modes...}`

P1607:

`{ignore, assume, check_never_continue, check_maybe_continue}`

P0542R5 vs P1607R0

P0542:

`{default, audit, axiom} * {continuation on, continuation off} * {build modes...}`

P1607:

`{ignore, assume, check_never_continue, check_maybe_continue}`

P0542R5 vs P1607R0

~~P0542:~~

~~`{default, audit, axiom} * {continuation on, continuation off} * {build modes...}`~~

~~P1607:~~

~~`{ignore, assume, check_never_continue, check_maybe_continue}`~~

~~P0542R5 vs P1607R0~~

~~P0542:~~

~~{default, audit, axiom} * {continuation on, continuation off} * {build modes...}~~

~~P1607:~~

~~{ignore, assume, check_never_continue, check_maybe_continue}~~

std::format?

std::format!

std::format customization basic

```
#include <format>

enum color { red, green, blue };
```

std::format customization basic

```
#include <format>

enum color { red, green, blue };

const char* color_names[] = {"red", "green", "blue"};
```

std::format customization basic

```cpp
#include <format>

enum color { red, green, blue };

const char* color_names[] = {"red", "green", "blue"};


std::format("The pixel is {}", color::red);  // The pixel is red

std::format_to(output_it, "{} is the color", color::red);  // red is the color
```

std::format customization basic

```
#include <format>

enum color { red, green, blue };

const char* color_names[] = {"red", "green", "blue"};


namespace std {

template <>

struct formatter<color> : formatter<const char*> {

  auto format(color c, format_context& ctx) {

    return formatter<const char*>::format(color_names[c], ctx);

  }

};

}  // namespace std
```

std::format customization basic

```
#include <format>

enum color { red, green, blue };

const char* color_names[] = {"red", "green", "blue"};


namespace std {

template <>

struct formatter<color> : formatter<const char*> {

  auto format(color c, format_context& ctx) {

    return formatter<const char*>::format(color_names[c], ctx);

  }

};

}  // namespace std
```

std::format с опциями

std::format customization simple

```
struct Answer {};
```

std::format customization simple

```
struct Answer {};


std::format("The Answer is {:d}", Answer{});  // The Answer is 42
std::format_to(output_it, "{:X} is the Answer", Answer{});  // 2A is the Answer
std::format_to(output_it, "{:x} is the Answer", Answer{});  // 2a is the Answer
```

std::format customization simple

```
struct Answer {};

namespace std {

template <>

struct formatter<Answer> {  // Parses dynamic width in the format "{:x}".

  char format_specifier_{'d'};


  auto parse(format_parse_context& parse_ctx);


  auto format(Answer, format_context& format_ctx);

};

}  // namespace std
```

std::format customization simple

```
struct Answer {};

namespace std {

template <>

struct formatter<Answer> {  // Parses dynamic width in the format "{:x}".

  char format_specifier_{'d'};


  auto parse(format_parse_context& parse_ctx);


  auto format(Answer, format_context& format_ctx);

};

}  // namespace std
```

std::format customization simple

```
struct Answer {};

namespace std {

template <>

struct formatter<Answer> {  // Parses dynamic width in the format "{:x}".

  char format_specifier_{'d'};


  auto parse(format_parse_context& parse_ctx);


  auto format(Answer, format_context& format_ctx);

};

}  // namespace std
```

std::format customization simple

```
auto parse(format_parse_context& parse_ctx) {
  auto iter = parse_ctx.begin();
  if (iter != parse_ctx.end()) {
    format_specifier_ = *iter;
    ++iter
  }
  return iter;
}
auto format(Answer, format_context& format_ctx) {
  return format_to(format_ctx.out(), "{:{}}", 42, format_specifier_);
}
```

std::format customization simple

```
auto parse(format_parse_context& parse_ctx) {
  auto iter = parse_ctx.begin();
  if (iter != parse_ctx.end()) {
    format_specifier_ = *iter;
    ++iter
  }
  return iter;
}
auto format(Answer, format_context& format_ctx) {
  return format_to(format_ctx.out(), "{:{}}", 42, format_specifier_);
}
```

std::format customization simple

```
auto parse(format_parse_context& parse_ctx) {
  auto iter = parse_ctx.begin();
  if (iter != parse_ctx.end()) {
    format_specifier_ = *iter;
    ++iter
  }
  return iter;
}
auto format(Answer, format_context& format_ctx) {
  return format_to(format_ctx.out(), "{:{}}", 42, format_specifier_);
}
```

std::format customization simple

```
auto parse(format_parse_context& parse_ctx) {
  auto iter = parse_ctx.begin();
  if (iter != parse_ctx.end()) {
    format_specifier_ = *iter;
    ++iter
  }
  return iter;
}
auto format(Answer, format_context& format_ctx) {
  return format_to(format_ctx.out(), "{:{}}", 42, format_specifier_);
}
```

std::format customization simple

```
auto parse(format_parse_context& parse_ctx) {
  auto iter = parse_ctx.begin();
  if (iter != parse_ctx.end()) {
    format_specifier_ = *iter;
    ++iter
  }
  return iter;
}
auto format(Answer, format_context& format_ctx) {
  return format_to(format_ctx.out(), "{:{}}", 42, format_specifier_);
}
```

std::format customization simple

```
auto parse(format_parse_context& parse_ctx) {
  auto iter = parse_ctx.begin();
  if (iter != parse_ctx.end()) {
    format_specifier_ = *iter;
    ++iter
  }
  return iter;
}
auto format(Answer, format_context& format_ctx) {
  return format_to(format_ctx.out(), "{:{}}", 42, format_specifier_);
}
```

std::format с динамическими опциями

std::format customization simple

```
struct Answer {};


std::format("The Answer is {1:{0}}", 'd', Answer{});  // The Answer is 42
std::format_to(output_it, "{0:{1}} is the Answer", Answer{}, 'X');  // 2A is the Answer
std::format_to(output_it, "{1:{0}} is the Answer", 'x', Answer{});  // 2a is the Answer
```

std::format customization hardcore

```
namespace std {
template <>
struct formatter<Answer> { // Parses dynamic width in the format "{:{<digit>}}"
  int arg_index_ = 0;

  auto parse(format_parse_context& parse_ctx);

  auto format(Answer, format_context& format_ctx);
};
}  // namespace std
```

std::format customization hardcore

```
auto parse(format_parse_context& parse_ctx) {
  auto iter = parse_ctx.begin();
  auto get_char = [&]() { return iter != parse_ctx.end() ? *iter : 0; };
  if (get_char() != '{') return iter;
  ++iter;
  char c = get_char();
  if (!std::isdigit(c) || (++iter, get_char()) != '}')
    throw format_error("invalid format");
  arg_index_ = c - '0';
  return ++iter;
}
```

std::format customization hardcore

```
auto parse(format_parse_context& parse_ctx) {
  auto iter = parse_ctx.begin();
  auto get_char = [&]() { return iter != parse_ctx.end() ? *iter : 0; };
  if (get_char() != '{') return iter;
  ++iter;
  char c = get_char();
  if (!std::isdigit(c) || (++iter, get_char()) != '}')
    throw format_error("invalid format");
  arg_index_ = c - '0';
  return ++iter;
}
```

std::format customization hardcore

```
auto parse(format_parse_context& parse_ctx) {
  auto iter = parse_ctx.begin();
  auto get_char = [&]() { return iter != parse_ctx.end() ? *iter : 0; };
  if (get_char() != '{') return iter;
  ++iter;
  char c = get_char();
  if (!std::isdigit(c) || (++iter, get_char()) != '}')
    throw format_error("invalid format");
  arg_index_ = c - '0';
  return ++iter;
}
```

std::format customization hardcore

```
auto parse(format_parse_context& parse_ctx) {
  auto iter = parse_ctx.begin();
  auto get_char = [&]() { return iter != parse_ctx.end() ? *iter : 0; };
  if (get_char() != '{') return iter;
  ++iter;
  char c = get_char();
  if (!std::isdigit(c) || (++iter, get_char()) != '}')
    throw format_error("invalid format");
  arg_index_ = c - '0';
  return ++iter;
}
```

std::format customization hardcore

```
auto format(Answer, format_context& format_ctx) {
  auto arg = format_ctx.arg(arg_index_);
  char fmt = visit_format_arg(
      [](auto value) -> char {
        if constexpr (!std::is_same_v<char, decltype(value)>)
          throw format_error("format is not a char");
        else return value;
      }, arg);
  return format_to(format_ctx.out(), "{:{}}", 42, fmt);
}
```

std::format customization hardcore

```
auto format(Answer, format_context& format_ctx) {
  auto arg = format_ctx.arg(arg_index_);
  char fmt = visit_format_arg(
      [](auto value) -> char {
        if constexpr (!std::is_same_v<char, decltype(value)>)
          throw format_error("format is not a char");
        else return value;
      }, arg);
  return format_to(format_ctx.out(), "{:{}}", 42, fmt);
}
```

std::format customization hardcore

```
auto format(Answer, format_context& format_ctx) {
  auto arg = format_ctx.arg(arg_index_);
  char fmt = visit_format_arg(
      [](auto value) -> char {
        if constexpr (!std::is_same_v<char, decltype(value)>)
          throw format_error("format is not a char");
        else return value;
      }, arg);
  return format_to(format_ctx.out(), "{:{}}", 42, fmt);
}
```

std::format customization hardcore

```
auto format(Answer, format_context& format_ctx) {
  auto arg = format_ctx.arg(arg_index_);
  char fmt = visit_format_arg(
      [](auto value) -> char {
        if constexpr (!std::is_same_v<char, decltype(value)>)
          throw format_error("format is not a char");
        else return value;
      }, arg);
  return format_to(format_ctx.out(), "{:{}}", 42, fmt);
}
```

std::format customization hardcore

```
auto format(Answer, format_context& format_ctx) {
  auto arg = format_ctx.arg(arg_index_);
  char fmt = visit_format_arg(
      [](auto value) -> char {
        if constexpr (!std::is_same_v<char, decltype(value)>)
          throw format_error("format is not a char");
        else return value;
      }, arg);
  return format_to(format_ctx.out(), "{:{}}", 42, fmt);
}
```

std::format customization hardcore

```
auto format(Answer, format_context& format_ctx) {
  auto arg = format_ctx.arg(arg_index_);
  char fmt = visit_format_arg(
      [](auto value) -> char {
        if constexpr (!std::is_same_v<char, decltype(value)>)
          throw format_error("format is not a char");
        else return value;
      }, arg);
  return format_to(format_ctx.out(), "{:{}}", 42, fmt);
}
```

std::jthread!

std::jthread

```cpp
#include <thread>
void sample() {
  std::jthread t([](std::stop_token st) {
    while (!st.stop_requested()) {
      /* ... */
    }
  });

  /* ... */

  t.request_stop();
}
```

std::jthread

```cpp
#include <thread>
void sample() {
  std::jthread t([](std::stop_token st) {
    while (!st.stop_requested()) {
      /* ... */
    }
  });

  /* ... */

  t.request_stop();
}
```

std::jthread

```
bool ready = false;
std::mutex ready_mutex;
std::condition_variable_any ready_cv;  // нужен именно `_any`!
std::jthread t([&ready, &ready_mutex, &ready_cv](std::stop_token st) {
      std::unique_lock lock{ready_mutex};
      // Ждёт нотификации и ready == true, или stop_token.request_stop(),
      // или jthread.request_stop().
      bool success = ready_cv.wait_until(lock, [&ready] { return ready; }, st);
      /* ... */
  }
});
```

std::jthread

```
bool ready = false;
std::mutex ready_mutex;
std::condition_variable_any ready_cv;  // нужен именно `_any`!
std::jthread t([&ready, &ready_mutex, &ready_cv](std::stop_token st) {
      std::unique_lock lock{ready_mutex};
      // Ждёт нотификации и ready == true, или stop_token.request_stop(),
      // или jthread.request_stop().
      bool success = ready_cv.wait_until(lock, [&ready] { return ready; }, st);
      /* ... */
  }
});
```

std::jthread

```cpp
bool ready = false;
std::mutex ready_mutex;
std::condition_variable_any ready_cv;  // нужен именно `_any`!
std::jthread t([&ready, &ready_mutex, &ready_cv](std::stop_token st) {
      std::unique_lock lock{ready_mutex};
      // Ждёт нотификации и ready == true, или stop_token.request_stop(),
      // или jthread.request_stop().
      bool success = ready_cv.wait_until(lock, [&ready] { return ready; }, st);
      /* ... */
  }
});
```

std::jthread

```
bool ready = false;
std::mutex ready_mutex;
std::condition_variable_any ready_cv;  // нужен именно `_any`!
std::jthread t([&ready, &ready_mutex, &ready_cv](std::stop_token st) {
      std::unique_lock lock{ready_mutex};
      // Ждёт нотификации и ready == true, или stop_token.request_stop(),
      // или jthread.request_stop().
      bool success = ready_cv.wait_until(lock, [&ready] { return ready; }, st);
      /* ... */
  }
});
```

Дальнейшие планы...

Спасибо

Полухин Антон

Эксперт-разработчик C++

antoshkka@gmail.com

antoshkka@yandex-team.ru

https://github.com/apolukhin

https://stdcpp.ru/

РГ21 C++ РОССИЯ