Строки, строки, строки и initializer_list

Полухин Антон
Эксперт разработчик C++

о чём поговорим

Back to basics: std::string_view, std::string. 01

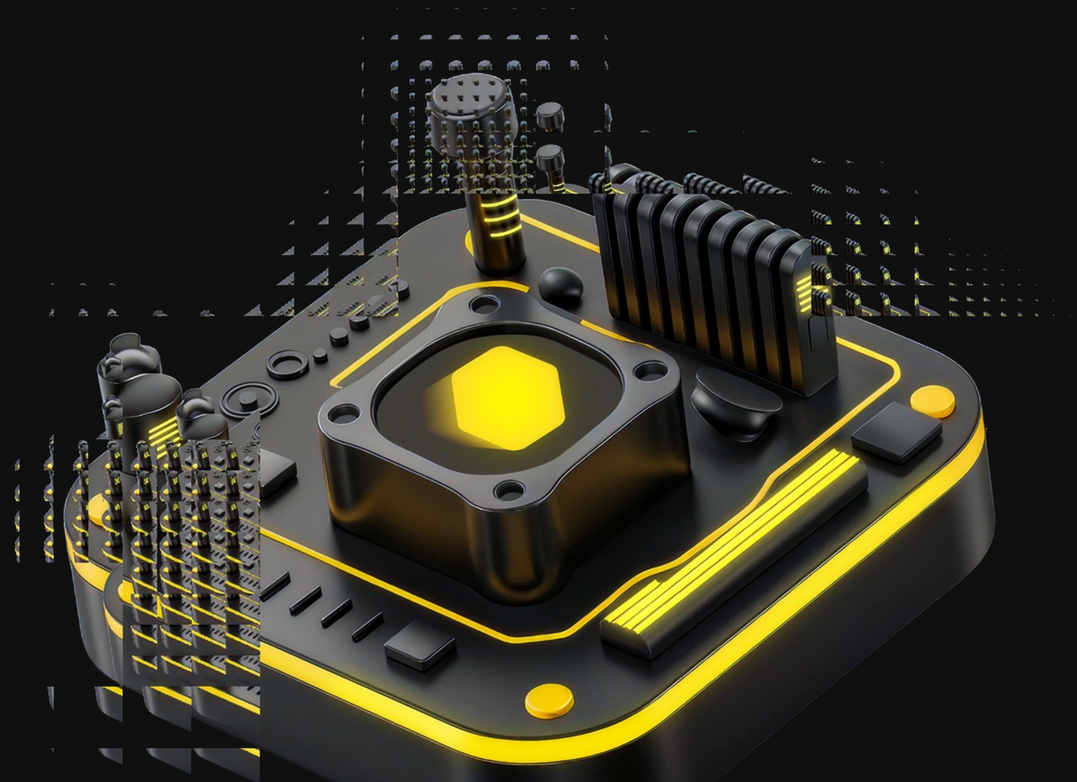
Нуль терминированность строки и как с ней не напортачитьН��t Н�5�+ 02

constinit, почему это важно и что делать с std::string, который в него не может 03

Бессмертные constinit нуль-терминированные строки 04

Обходим ограничения std::initializer_list 05

string vs string_view



Основы std::string и SSO

```
std::string hello = "Hello";
hello.append(" world");
```

Передача std::string

```
void do_something(const std::string& s);


auto sample() {
    std::string hello = "Hello";
    hello.append(" world");
    // ...
    do_something(hello);
```

Неявное конструирование std::string

```cpp
void do_something(const std::string& s);


auto sample() {
    std::string hello = "Hello";
    hello.append(" world");
    // ...
    do_something(hello);
    do_something("Wubba lubba dub dub!!!");  // Не влезает в SSO
```

Неявное конструирование std::string

```
void do_something(const std::string& s);


auto sample() {
    std::string hello = "Hello";
    hello.append(" world");
    // ...
    do_something(hello);
    do_something("Wubba lubba dub dub!!!");  // Не влезает в SSO
    do_something(
        std::string{"Wubba lubba dub dub!!!"}  // Не влезает в SSO
    );
}
```

std::string vs std::string_view

```
void do_something(const std::string& s);
```

std::string vs std::string_view

```
void do_something2(std::string_view s);
```

std::string_view

```
void do_something2(std::string_view s);

int sample2() {
    std::string hello = "Hello";
    hello.append(" world");
    do_something2(hello);
    do_something2("Wubba lubba dub dub!!!");
```

Дешёвые операции std::string_view

```
void do_something2(std::string_view s);

int sample2() {
    std::string hello = "Hello";
    hello.append(" world");
    do_something2(hello);
    do_something2("Wubba lubba dub dub!!!");

    std::string_view wor = hello;
    wor.remove_prefix(6);  // Нет аллокаций и копирований!
    wor.remove_suffix(2);  // Нет аллокаций и копирований!
```

\0 и std::string_view

```
void do_something2(std::string_view s);

int sample2() {
    std::string hello = "Hello";
    hello.append(" world");
    do_something2(hello);
    do_something2("Wubba lubba dub dub!!!");

    std::string_view wor = hello;
    wor.remove_prefix(6);  // Нет аллокаций и копирований!
    wor.remove_suffix(2);  // Нет аллокаций и копирований!

    assert(wor == "wor");  // ... но нет и нуль терминирования
    assert(std::strlen(wor.data()) != wor.size());
}
```

О важности c_str()

```
void parse_xml(const char*);


void do_something(const std::string& s) {
    parse_xml(s.c_str());  // Надёжно
}
```

О важности c_str()

```
void parse_xml(const char*);


void do_something2(std::string_view s) {
    parse_xml(s.c_str());  // Не скомпилируется, ура!
}
```

О важности c_str()

```
void parse_xml(const char*);


void do_something2(std::string_view s) {
    parse_xml(s.data());  // Упадёт! :(
}
```

\0 и эффективная передача параметра

```cpp
void do_something3(const char* s) { /* ... */ }

void do_something3(const std::string& s) { do_something3(s.c_str()); }
```

\0 и возвращаемое значение

```
class Something {
public:
    std::string_view foo() const;
    std::string_view bar() const;
    std::string_view buz() const;
    // ...
};
```

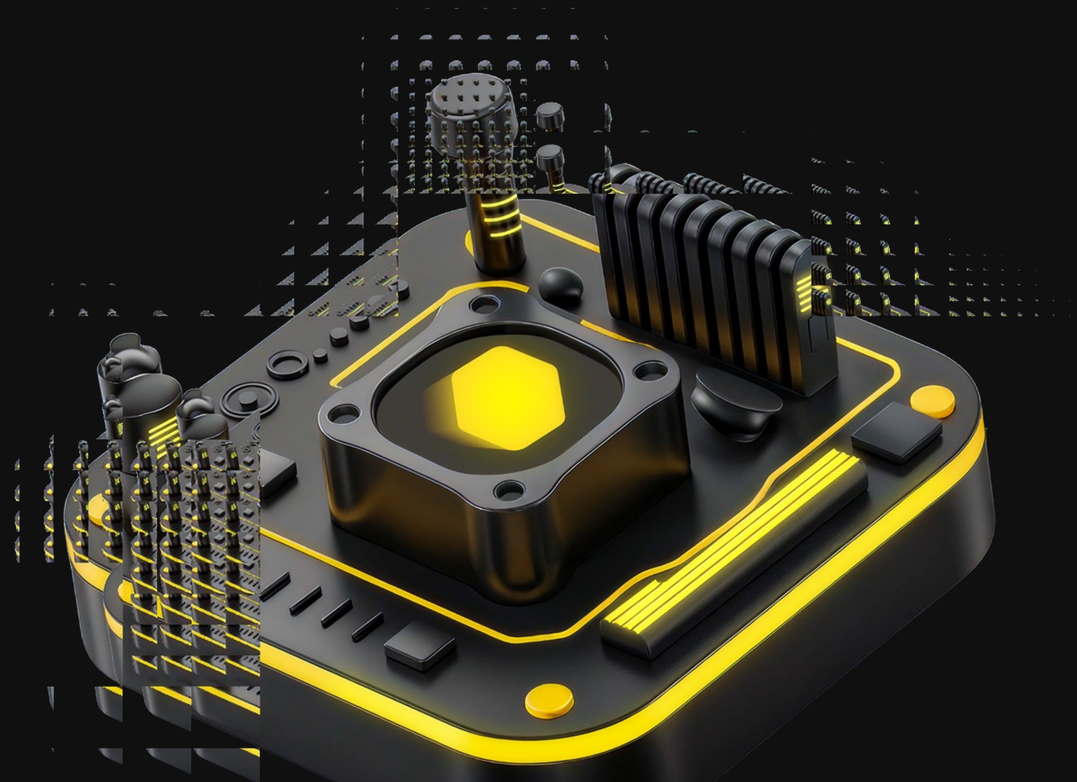
\0 и возвращаемое значение

```
class Something {
public:
    std::string_view foo() const;
    std::string_view bar() const;
    std::string_view buz() const;
    // ...
};


auto sample3() {
    parse_xml(Something{}.bar().data());  // Хлипкая конструкция
}
```

\0 и возвращаемое значение

```
class Something {
public:
    std::string_view foo() const;
    const char* bar() const;
    std::string_view buz() const;
    // ...
};


auto sample3() {
    parse_xml(Something{}.bar());  // Надо деаллоцировать? nullptr?
}
```

std::zstring_view
std::cstring_view



zstring_view

```
class zstring_view : public std::string_view {

public:

    zstring_view() = delete;
    zstring_view(const zstring_view& str) = default;

    constexpr zstring_view(const char* str) noexcept : std::string_view{str} {}
    zstring_view(const std::string& str) noexcept : std::string_view{str} {}

    zstring_view& operator=(std::string_view) = delete;
    zstring_view& operator=(const zstring_view&) = default;

    constexpr const char* c_str() const noexcept { return std::string_view::data(); }

};
```

zstring_view

```
class zstring_view : public std::string_view {
public:
    zstring_view() = delete;
    zstring_view(const zstring_view& str) = default;

    constexpr zstring_view(const char* str) noexcept : std::string_view{str} {}
    zstring_view(const std::string& str) noexcept : std::string_view{str} {}

    zstring_view& operator=(std::string_view) = delete;
    zstring_view& operator=(const zstring_view&) = default;

    constexpr const char* c_str() const noexcept { return std::string_view::data(); }

};
```

zstring_view

```
class zstring_view : public std::string_view {
public:
    zstring_view() = delete;
    zstring_view(const zstring_view& str) = default;

    constexpr zstring_view(const char* str) noexcept : std::string_view{str} {}
    zstring_view(const std::string& str) noexcept : std::string_view{str} {}

    zstring_view& operator=(std::string_view) = delete;
    zstring_view& operator=(const zstring_view&) = default;

    constexpr const char* c_str() const noexcept { return std::string_view::data(); }

};
```

zstring_view

```
class zstring_view : public std::string_view {

public:

    zstring_view() = delete;
    zstring_view(const zstring_view& str) = default;

    constexpr zstring_view(const char* str) noexcept : std::string_view{str} {}
    zstring_view(const std::string& str) noexcept : std::string_view{str} {}

    zstring_view& operator=(std::string_view) = delete;
    zstring_view& operator=(const zstring_view&) = default;

    constexpr const char* c_str() const noexcept { return std::string_view::data(); }

};
```

zstring_view::c_str()

```
class zstring_view : public std::string_view {
public:
    zstring_view() = delete;
    zstring_view(const zstring_view& str) = default;

    constexpr zstring_view(const char* str) noexcept : std::string_view{str} {}
    zstring_view(const std::string& str) noexcept : std::string_view{str} {}

    zstring_view& operator=(std::string_view) = delete;
    zstring_view& operator=(const zstring_view&) = default;
    constexpr const char* c_str() const noexcept { return std::string_view::data(); }
};
```

std::string vs std::string_view

```
void parse_xml(const char*);


void do_something2(std::string_view s) {
    parse_xml(s.c_str());  // Не скомпилируется, ура!
}
```

std::zstring_view

```
void parse_xml(const char*);


void do_something2(std::zstring_view s) {
    parse_xml(s.c_str());  // Работает, ура!
}
```

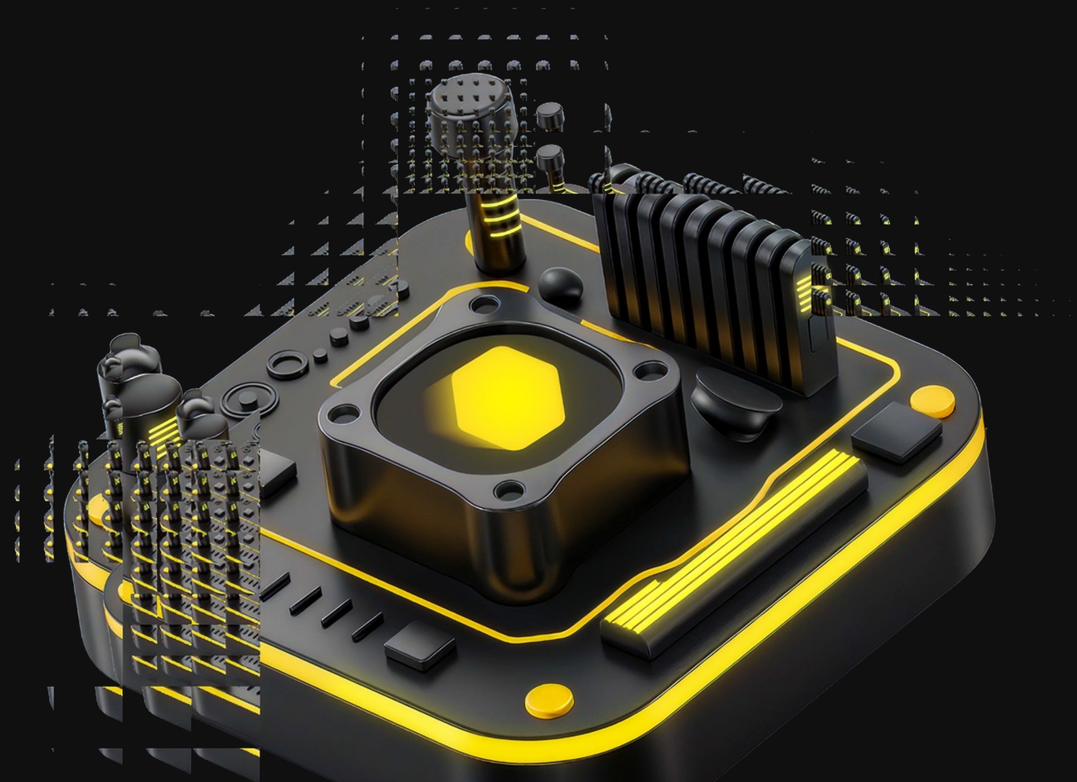
std::string_view

```
class Something {
public:
    std::string_view foo() const;
    std::string_view bar() const;
    std::string_view buz() const;
    // ...
};


auto sample3() {
    parse_xml(Something{}.bar().data());  // Хлипкая конструкция
}
```

std::zstring_view

```
class Something {
public:
    std::string_view foo() const;
    std::zstring_view bar() const;
    std::string_view buz() const;
    // ...
};


auto sample3() {
    parse_xml(Something{}.bar().c_str());  // Надёжно!!
}
```

constinit std::string



constinit std::string

```
extern const std::string kSomeConstant;
```

Проблемы std::string

```
extern const std::string kSomeConstant;


// в первом .cpp файле
const std::string kSomeConstant = R"~(
    Не выходи из комнаты; считай, что тебя продуло.
    Что интересней на свете стены и стула?
    Зачем выходить оттуда, куда вернешься вечером
    таким же, каким ты был, тем более — изувеченным?)~";
```

Проблемы std::string

```cpp
extern const std::string kSomeConstant;


// в первом .cpp файле
const std::string kSomeConstant = R"~(
    Не выходи из комнаты; считай, что тебя продуло.
    Что интересней на свете стены и стула?
    Зачем выходить оттуда, куда вернешься вечером
    таким же, каким ты был, тем более — изувеченным?)~";


// во втором .cpp файле
const std::string kSomeConstantWithAuthor = kSomeConstant + "\n\t И. Бродский";
constexpr std::string_view kSomeConstant2 = kSomeConstant;
```

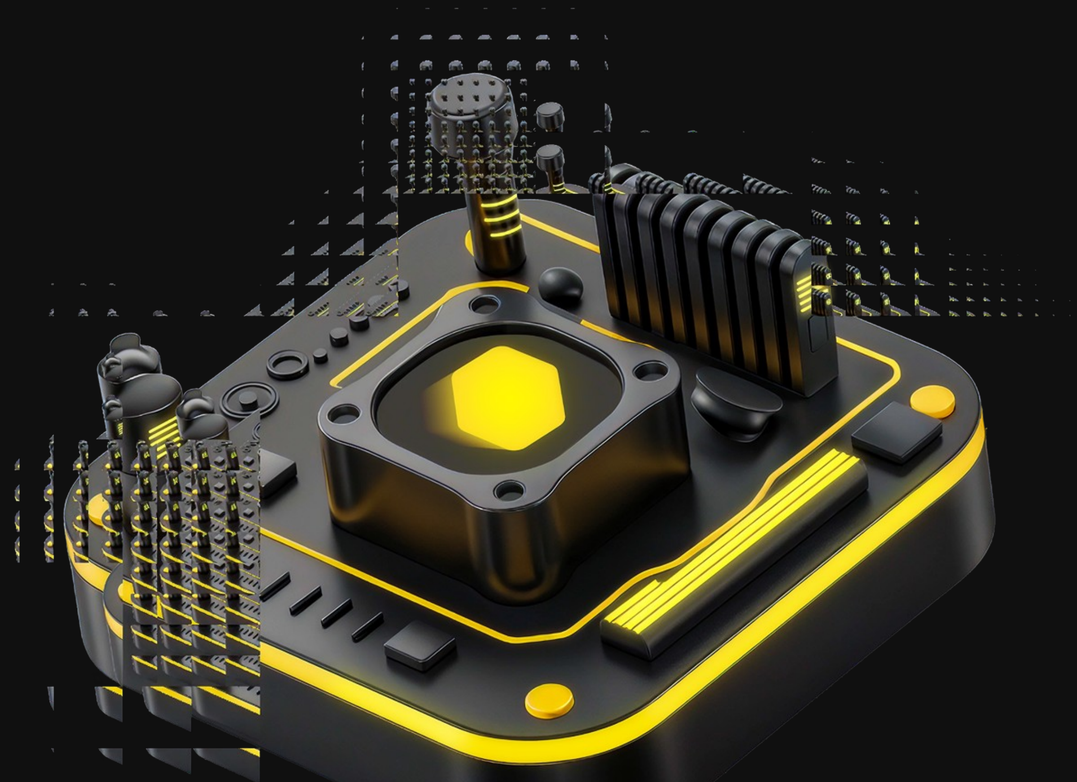
Непонятный lifetime

```
class Something {
public:
    std::string_view foo() const;
    std::zstring_view bar() const;
    std::string_view buz() const;
};
```

Непонятный lifetime

```
class Something {
public:
    std::string_view foo() const;
    std::zstring_view bar() const;
    std::string_view buz() const;
};


auto sample4() {
    auto s = Something{}.buz();
    // ...
    return s;
}
```

utils::StringLiteral



utils::StringLiteral

```
class StringLiteral : public zstring_view {
public:
    StringLiteral() = delete;


    consteval StringLiteral(const char* literal) noexcept: zstring_view{literal} {}
};
```

utils::StringLiteral

```
class StringLiteral : public zstring_view {
public:
    StringLiteral() = delete;


    consteval StringLiteral(const char* literal) noexcept: zstring_view{literal} {}
};
```

utils::StringLiteral

```
class StringLiteral : public zstring_view {
public:
    StringLiteral() = delete;


#if defined(__clang__) && __clang_major__ < 18
    constexpr
#else
    consteval
#endif
        StringLiteral(const char* literal) noexcept: zstring_view{literal} {}
};
```

utils::StringLiteral

```
class Something {
public:
    std::string_view foo() const;
    std::zstring_view bar() const;
    utils::StringLiteral buz() const;
};


utils::StringLiteral sample4() {
    utils::StringLiteral s = Something{}.buz();
    // ...
    return s;
}
```

utils::StringLiteral

```
class Something {
public:
    std::string_view foo() const;
    std::zstring_view bar() const;
    utils::StringLiteral buz() const;
};


utils::StringLiteral sample4() {
    utils::StringLiteral s = Something{}.buz();
    // ...
    return s;
}
```

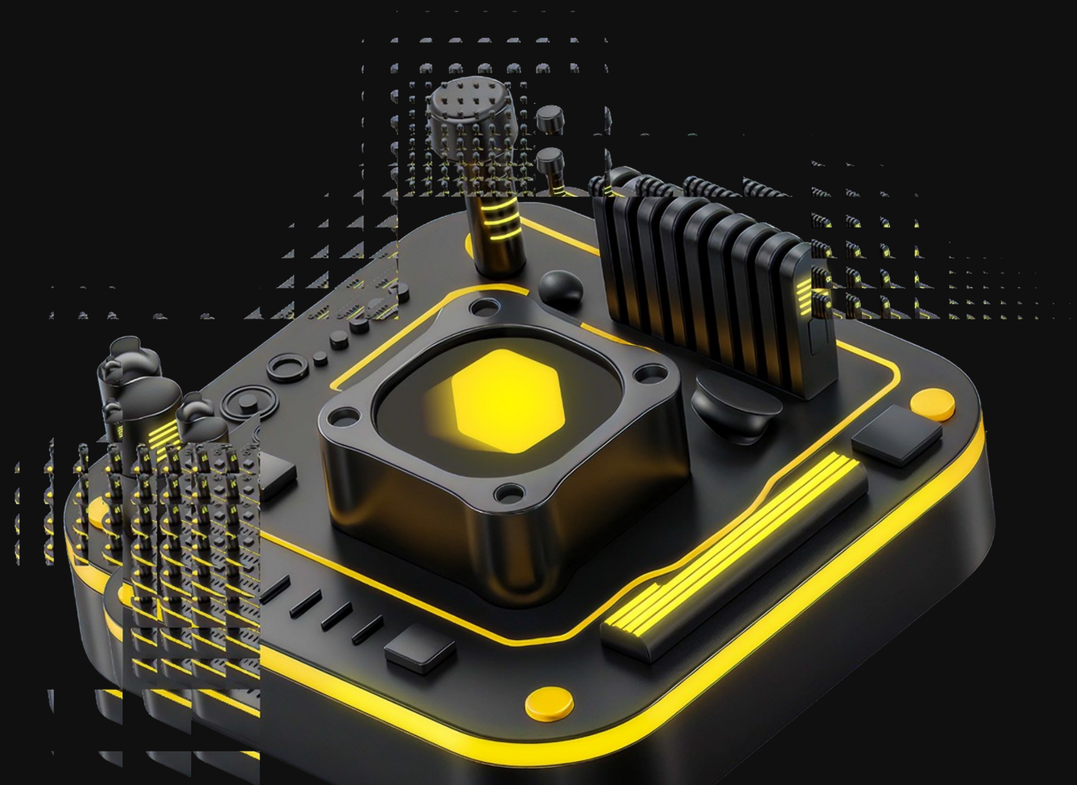
utils::StringLiteral

```
void some_async(const char* literal);
```

utils::StringLiteral

```
void some_async(utils::StringLiteral x);
```

std::initializer_list



std::initializer_list

```
logging::LogExtra log_extra{
    {tracing::kHttpMetaType, meta_type},
    {tracing::kType, kTracingTypeRequest},
    {"request_body_length", request.RequestBody().length()},
    {kTracingBody, handler_.GetRequestBodyForLoggingChecked(request, context)},
    {kTracingUri, handler_.GetUrlForLoggingChecked(request, context)},
    {tracing::kHttpMethod, request.GetMethodStr()},
};
```

std::initializer_list

```
class LogExtra final {
public:
    using Value = std::variant<
        std::string,
        long long,
        double,
        JsonString>;
    using Pair = std::pair<std::string, Value>;
    using InitializerList = std::initializer_list<Pair>;

    LogExtra(InitializerList initial);
```

std::initializer_list

```
class LogExtra final {
public:
    using Value = std::variant<
        std::string,
        long long,
        double,
        JsonString>;
    using Pair = std::pair<std::string, Value>;
    using InitializerList = std::initializer_list<Pair>;

    LogExtra(InitializerList initial);
```

std::initializer_list

```
logging::LogExtra log_extra{
    {tracing::kHttpMetaType, meta_type},
    {tracing::kType, kTracingTypeRequest},
    {"request_body_length", request.RequestBody().length()},
    {kTracingBody, handler_.GetRequestBodyForLoggingChecked(request, context)},
    {kTracingUri, handler_.GetUrlForLoggingChecked(request, context)},
    {tracing::kHttpMethod, request.GetMethodStr()},
};
```

std::initializer_list

```
logging::LogExtra log_extra{
    {tracing::kHttpMetaType, meta_type},
    {tracing::kType, kTracingTypeRequest},
    {"request_body_length", request.RequestBody().length()},
    {kTracingBody, handler_.GetRequestBodyForLoggingChecked(request, context)},
    {kTracingUri, handler_.GetUrlForLoggingChecked(request, context)},
    {tracing::kHttpMethod, request.GetMethodStr()},
};
```

std::initializer_list

```
class LogExtra final {
public:
    using Value = std::variant<
        std::string,
        long long,
        double,
        JsonString>;
    using Pair = std::pair<std::string, Value>;
    using InitializerList = std::initializer_list<Pair>;

    LogExtra(InitializerList initial);
```

std::initializer_list

```
class LogExtra final {
public:
    using ValueView = std::variant<
        std::string_view,
        long long,
        double,
        impl::JsonStringViewForInitializerList>;
    using Pair = std::pair<std::string_view, ValueView>;
    using InitializerList = std::initializer_list<Pair>;

    LogExtra(InitializerList initial);
```

Наколеночный view

```
struct JsonStringViewForInitializerList {
    JsonStringViewForInitializerList() = delete;
    /*implicit*/ JsonStringViewForInitializerList(const JsonString& json) noexcept
        : json_str{&json} {}
    /*implicit*/ JsonStringViewForInitializerList(const formats::json::Value& json)
noexcept
        : json_value{&json} {}


    const JsonString* const json_str{nullptr};
    const formats::json::Value* const json_value{nullptr};
};
```

Наколеночный view

```
struct JsonStringViewForInitializerList {
    JsonStringViewForInitializerList() = delete;
    /*implicit*/ JsonStringViewForInitializerList(const JsonString& json) noexcept
        : json_str{&json} {}
    /*implicit*/ JsonStringViewForInitializerList(const formats::json::Value& json)
noexcept
        : json_value{&json} {}


    const JsonString* const json_str{nullptr};
    const formats::json::Value* const json_value{nullptr};
};
```

Выводы

Выводы

01 std::string* хороши

Выводы

01 std::string* хороши, но

Выводы

01 std::string* хороши, но

02 std::zstring_view тоже нужен

Выводы

01 std::string* хороши, но

02 std::zstring_view тоже нужен

03 c_str() для \0 строк

Выводы

01 std::string* хороши, но

02 std::zstring_view тоже нужен

03 c_str() для \0 строк

04 Нужен StringLiteral

Выводы

01 std::string* хороши, но

02 std::zstring_view тоже нужен

03 c_str() для \0 строк

04 Нужен StringLiteral

05 initializer_list сломан, но...

Выводы

01 std::string* хороши, но

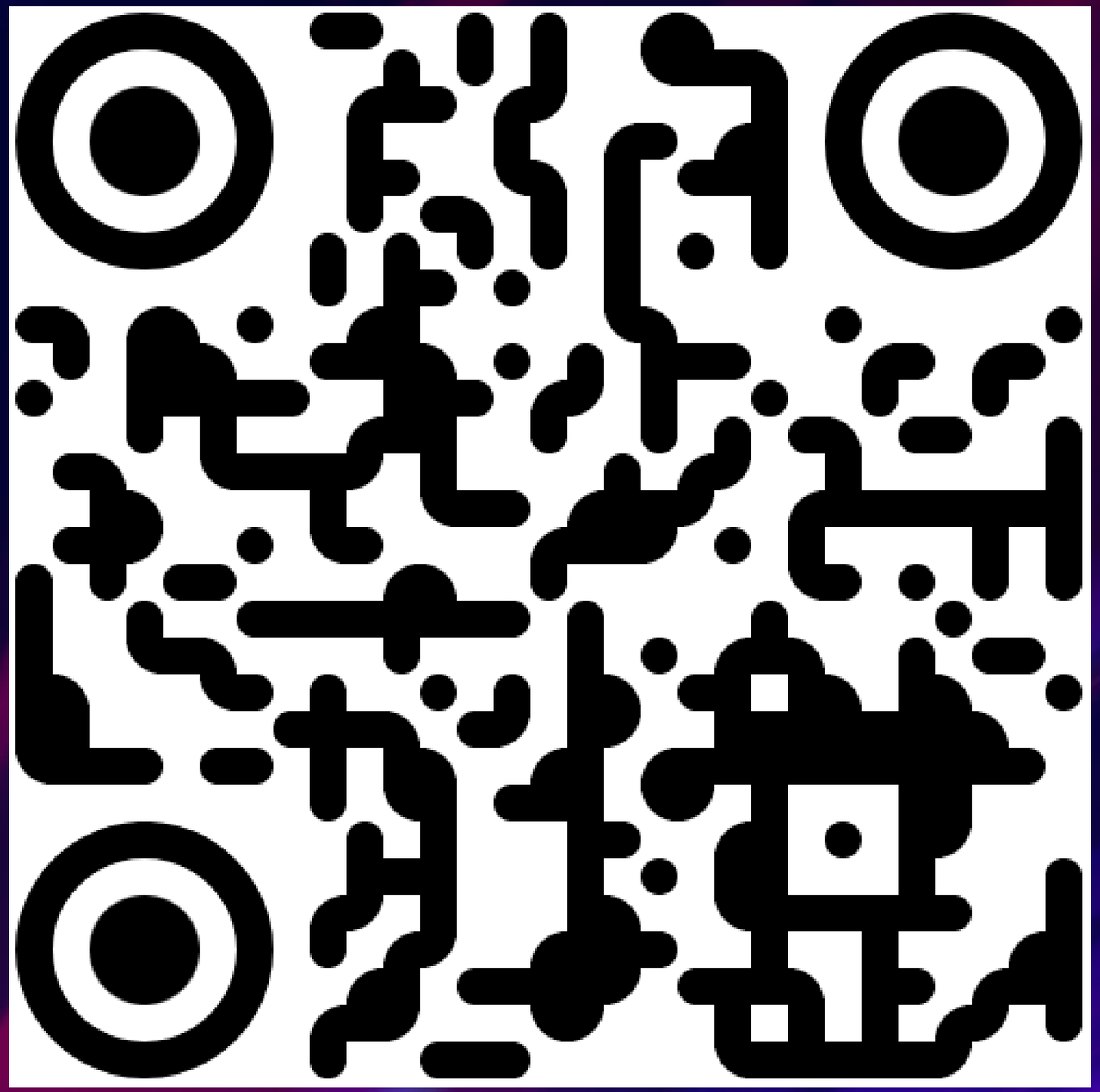
02 std::zstring_view тоже нужен

03 c_str() для \0 строк

04 Нужен StringLiteral

05 initializer_list сломан, но...

06 ... view исправляют ситуацию



БУДУЩЕЕ
В НАШИХ
РУКАХ