Вы всё ещё пишете код руками?

Тогда мы идём к вам!

Сергей Садовников

Обо мне

Сергей Садовников

Руководитель отдела разработки Finteza в компании “MetaQuotes Ltd.”

corehard.by

И снова сериализация

```
enum class Enum
{
        Item1,
        Item2
};

struct SimpleStruct
{
    int intField;
    std::string strField;
    Enum enumField;
};

struct ComplexStruct
{
    int intField;
    SimpleStruct innerStruct;
    std::vector<SimpleStruct> structVector;
    std::set<int> intSet;
};
```

И снова сериализация

```
enum class Enum
{
        Item1,
        Item2
};

struct SimpleStruct
{
    int intField;
    std::string strField;
    Enum enumField;
};

struct ComplexStruct
{
    int intField;
    SimpleStruct innerStruct;
    std::vector<SimpleStruct> structVector;
    std::set<int> intSet;
};
```

И снова сериализация

```
enum class Enum
{
        Item1,
        Item2
};

struct SimpleStruct
{
   int intField;
   std::string strField;
   Enum enumField;
};

struct ComplexStruct
{
   int intField;
   SimpleStruct innerStruct;
   std::vector<SimpleStruct> structVector;
   std::set<int> intSet;
};
```

0. И снова сериализация - всё руками

```
void JsonSerialize(rapidjson::Value& node,
                   const SimpleStruct& structValue,
                   rapidjson::Document::AllocatorType& allocator)
{
    using namespace rapidjson;

    node.SetObject();
    node.AddMember("intField", structValue.intField, allocator);
    node.AddMember("strField",
                   StringRef(structValue.strField.data(),
                             structValue.strField.size()),
                   allocator);
    node.AddMember(
        "enumField",
        JsonSerialize(structValue.enumField, allocator).Move(),
        allocator);
}
```

0. И снова сериализация - всё руками

```
rapidjson::Value JsonSerialize(
                Enum value,
                rapidjson::Document::AllocatorType& allocator)
{
    rapidjson::Value node;
    switch (value)
    {
    case Enum::Item1:
        node.SetString("Item1", allocator);
        break;
    case Enum::Item2:
        node.SetString("Item2", allocator);
        break;
    }

    return node;
}
```

0. И снова сериализация - всё руками

```
void JsonDeserialize(const rapidjson::Value& node,
                     SimpleStruct& structValue)
{
    using namespace rapidjson;

    if (node.HasMember("intField"))
        structValue.intField = node["intField"].GetInt();
    if (node.HasMember("strField"))
        structValue.strField = node["strField"].GetString();
    if (node.HasMember("enumField"))
    {
        const Value& value = node["enumField"];
        JsonDeserialize(value, structValue.enumField);
    }
}
```

0. И снова сериализация - всё руками

```
void JsonDeserialize(const rapidjson::Value& node, Enum& value)
{
    static std::pair<const char*, Enum> items[] = {
        { "Item1", Enum::Item1 }, { "Item2", Enum::Item2 }
    };

    const char* itemName = node.GetString();

    auto p = std::lower_bound(
      begin(items), end(items), itemName, [](auto&& i, auto&& v) {
          return strcmp(i.first, v) < 0;
      });

    if (p == end(items) || strcmp(p->first, itemName) != 0)
        throw std::invalid_argument(
          std::string("Bad Enum enum item name: ") + itemName);

    value = p->second;
}
```

0. И снова сериализация - всё руками

- Всё приходится писать руками
- Поддерживать в консистентном состоянии
- Следить за всем глазками

0. И снова сериализация - всё руками

И ничего не забывать!

И снова сериализация

Как сделать жизнь проще?

И снова сериализация

Макросы!

1. И снова сериализация - макросы

```
FL_STRINGIZED_ENUM(Enum,
        FL_ENUM_ENTRY(Item1)
        FL_ENUM_ENTRY(Item2)
)
```

1. И снова сериализация - макросы

```
FL_STRINGIZED_ENUM(Enum,
        FL_ENUM_ENTRY(Item1)
        FL_ENUM_ENTRY(Item2)
)
```

```
enum class Enum
{
        Item1,
        Item2
};
```

1. И снова сериализация - макросы

```
FL_STRINGIZED_ENUM(Enum,
        FL_ENUM_ENTRY(Item1)
        FL_ENUM_ENTRY(Item2)
)
```

```
enum class Enum
{
        Item1,
        Item2
};
```

```
rapidjson::Value JsonSerialize(
        Enum value,
        rapidjson::Document::AllocatorType& allocator)
{
   //...
}

void JsonDeserialize(const rapidjson::Value& node,
        Enum& value)
{
   //...
}
```

1. И снова сериализация - макросы

```
#define FL_ENUM_EMPTY_NAME BOOST_PP_EMPTY()

#define FL_DECLARE_ENUM_ENTRY_IMPL(EntryName, EntryVal, _) EntryName EntryVal

#define FL_DECLARE_ENUM_ENTRY(_1, _2, Entry) FL_DECLARE_ENUM_ENTRY_IMPL Entry,

#define FL_DECLARE_ENUM(EnumName, S) enum EnumName {
BOOST_PP_SEQ_FOR_EACH(FL_DECLARE_ENUM_ENTRY, _, S) };

#define FL_MAKE_STRING_ENUM_NAME(Entry) Entry
#define FL_MAKE_WSTRING_ENUM_NAME(Entry) L##Entry

#define FL_DECLARE_ENUM2STRING_ENTRY_IMPL(STRING_EXPANDER, EntryId, EntryName)
\
  case EntryId:                                                                    \
    result = STRING_EXPANDER(EntryName);
\
    break;

#define FL_DECLARE_STRING2ENUM_ENTRY_IMPL(STRING_EXPANDER, EntryId, EntryName)
result[STRING_EXPANDER(EntryName)] = EntryId;

#define FL_DECLARE_ENUM2STRING_ENTRY(_, STRING_EXPANDER, Entry)
\
  FL_DECLARE_ENUM2STRING_ENTRY_IMPL(STRING_EXPANDER, BOOST_PP_TUPLE_ELEM(3, 0, Entry),
BOOST_PP_TUPLE_ELEM(3, 2, Entry))
#define FL_DECLARE_STRING2ENUM_ENTRY(_, STRING_EXPANDER, Entry)
\
  FL_DECLARE_STRING2ENUM_ENTRY_IMPL(STRING_EXPANDER, BOOST_PP_TUPLE_ELEM(3, 0, Entry),
BOOST_PP_TUPLE_ELEM(3, 2, Entry))
```

```
#define FL_DECLARE_ENUM_STRINGS(EnumName, MACRO_NAME, S)
  inline EnumName##_StringMap_CharType const* EnumName##ToString(EnumName e)
  {                                                                                \
    EnumName##_StringMap_CharType const* result = NULL;
    switch (e)                                                                     \
    {                                                                              \
      BOOST_PP_SEQ_FOR_EACH(FL_DECLARE_ENUM2STRING_ENTRY, MACRO_NAME, S)
    }                                                                              \
    return result;                                                                 \
  }                                                                                \
  inline EnumName StringTo##EnumName(EnumName##_StringMap_CharType const* str)
  {                                                                                \
    static std::map<std::basic_string<EnumName##_StringMap_CharType>, EnumName> strings_map =
      []() -> std::map<std::basic_string<EnumName##_StringMap_CharType>, EnumName>
    {                                                                              \
      std::map<std::basic_string<EnumName##_StringMap_CharType>, EnumName> result;
      BOOST_PP_SEQ_FOR_EACH(FL_DECLARE_STRING2ENUM_ENTRY, MACRO_NAME, S)
      return result;                                                               \
    }                                                                              \
    ();                                                                            \
                                                                                   \
    return flex_lib::detail::FindEnumEntryForString(strings_map, str);
  }

#define FL_STRINGIZED_ENUM(EnumName, EnumItems)
  typedef char EnumName##_StringMap_CharType;
  FL_DECLARE_ENUM(EnumName, EnumItems)
  FL_DECLARE_ENUM_STRINGS(EnumName, FL_MAKE_STRING_ENUM_NAME, EnumItems)
#define FL_WSTRINGIZED_ENUM(EnumName, EnumItems)
  typedef wchar_t EnumName##_StringMap_CharType;
  FL_DECLARE_ENUM(EnumName, EnumItems)
  FL_DECLARE_ENUM_STRINGS(EnumName, FL_MAKE_WSTRING_ENUM_NAME, EnumItems)
#define FL_ENUM_ENTRY(EnumEntry) ((EnumEntry, BOOST_PP_EMPTY(), #EnumEntry))
#define FL_ENUM_NAMED_ENTRY(EnumEntry, EnumEntryName) ((EnumEntry, BOOST_PP_EMPTY(),
EnumEntryName))
#define FL_ENUM_SPEC_ENTRY(EnumEntry, Id) ((EnumEntry, = Id, #EnumEntry))
```

1. И снова сериализация - макросы

Что делать с информацией о типах?

И снова сериализация

Макросы + шаблоны!

2. И снова сериализация - макросы и шаблоны

```
namespace hana = boost::hana;

struct SimpleStruct
{
    BOOST_HANA_DEFINE_STRUCT(SimpleStruct,
    (int, intField),
    (std::string, strField)
    );
};
```

2. И снова сериализация - макросы и шаблоны

```
namespace hana = boost::hana;

struct SimpleStruct
{
    BOOST_HANA_DEFINE_STRUCT(SimpleStruct,
    (int, intField),
    (std::string, strField)
    );
};
```

2. И снова сериализация - макросы и шаблоны

```
template<typename T>
void SerializeToStream(const T& value, std::ostream& os)
{
    hana::for_each(hana::members(value), [&](auto member) {os << member << std::endl;});
}

template<typename T>
void DeserializeFromStream(std::istream& is, T& value)
{
    hana::for_each(hana::keys(value), [&](auto key)
    {
        auto& member = hana::at_key(value, key);
        is >> member;
    });
}
```

2. И снова сериализация - макросы и шаблоны

```
template<typename T>
void SerializeToStream(const T& value, std::ostream& os)
{
   hana::for_each(hana::members(value), [&](auto member) {os << member << std::endl;});
}

template<typename T>
void DeserializeFromStream(std::istream& is, T& value)
{
   hana::for_each(hana::keys(value), [&](auto key)
   {
      auto& member = hana::at_key(value, key);
      is >> member;
   });
}
```

2. И снова сериализация - макросы и шаблоны

```
2007  //////////////////////////////////////////////////////////////////////////////
2008  // BOOST_HANA_DEFINE_STRUCT
2009  //////////////////////////////////////////////////////////////////////////////
2010  #define BOOST_HANA_DEFINE_STRUCT(...) \
2011      BOOST_HANA_DEFINE_STRUCT_IMPL(BOOST_HANA_PP_NARG(__VA_ARGS__), __VA_ARGS__)
2012
2013  #ifdef BOOST_HANA_WORKAROUND_MSVC_PREPROCESSOR_616033
2014  #define BOOST_HANA_DEFINE_STRUCT_IMPL(N, ...) BOOST_HANA_DEFINE_STRUCT_IMPL_I(N, __VA_ARGS__)
2015  #define BOOST_HANA_DEFINE_STRUCT_IMPL_I(N, ...) \
2016      BOOST_HANA_PP_CONCAT(BOOST_HANA_PP_CONCAT(BOOST_HANA_DEFINE_STRUCT_IMPL_, N)(__VA_ARGS__),)
2017  #else
2018  #define BOOST_HANA_DEFINE_STRUCT_IMPL(N, ...) \
2019      BOOST_HANA_PP_CONCAT(BOOST_HANA_DEFINE_STRUCT_IMPL_, N)(__VA_ARGS__)
2020  #endif
2021
2022
2023  #define BOOST_HANA_DEFINE_STRUCT_IMPL_1(TYPE )       \
2024           \
2025                                                                    \
2026    struct hana_accessors_impl {                                    \
2027      static constexpr auto apply() {                               \
2028        struct member_names {                                       \
```

boost/hana/detail/struct_macros.hpp (Boost 1.71)

2. И снова сериализация - макросы и шаблоны

```
2783  #define BOOST_HANA_DEFINE_STRUCT_IMPL_41(TYPE , m1, m2, m3, m4, m5, m6, m7, m8, m9, m10, m11, m12, m13, m14,
2784    BOOST_HANA_PP_DROP_BACK m1 BOOST_HANA_PP_BACK m1; BOOST_HANA_PP_DROP_BACK m2 BOOST_HANA_PP_BACK m2; BOOST_
2785                                                                                                          \
2786    struct hana_accessors_impl {                                                                          \
2787      static constexpr auto apply() {                                                                     \
2788        struct member_names {                                                                             \
2789          static constexpr auto get() {                                                                   \
2790              return ::boost::hana::make_tuple(                                                           \
2791                BOOST_HANA_PP_STRINGIZE(BOOST_HANA_PP_BACK m1), BOOST_HANA_PP_STRINGIZE(BOOST_HANA_PP_BACK m2)
2792              );                                                                                          \
2793          }                                                                                               \
2794        };                                                                                                \
2795        return ::boost::hana::make_tuple(                                                                 \
2796          ::boost::hana::make_pair(::boost::hana::struct_detail::prepare_member_name<0, member_names>(), ::boo
2797        );                                                                                                \
2798      }                                                                                                   \
2799    }                                                                                                     \
2800  /**/
2801
2802
2803  #endif // !BOOST_HANA_DETAIL_STRUCT_MACROS_HPP
```

boost/hana/detail/struct_macros.hpp (Boost 1.71)

И снова сериализация

Свой собственный парсер!

3. И снова сериализация - свой парсер

- Qt MOC
- Unreal Header Tool (UHT)

3. И снова сериализация - свой парсер

```python
import sys

source = open(sys.argv[1], 'r')

current_type = ""
attributes = {}
types = {}

for line in source:
    words = line.split()
    # skip empty lines
    if len(words) < 1:
        continue

    # start a type - also skips forward declarations
    if words[0] in {"struct", "class"} and current_type == "" and len(words) > 1 and ";" not in line:
        if words[1] in ["HAPI", "HA_EMPTY_BASE"] and len(words) > 2:
            current_type = words[2]
        else:
            current_type = words[1]
        types[current_type] = {"attributes" : attributes, "fields" : []}
        # reset attributes for the following fields
        attributes = {}
        continue
    # end a type
```

https://github.com/onqtam/cmake-reflection-template/blob/master/scripts/parse_source.py

3. И снова сериализация - свой парсер

```
class Foo {
    FRIENDS_OF_TYPE(Foo); // friend declarations of the generated functions

    FIELD int                   a;
    FIELD float                 b = 42.f;
    FIELD std::string           c {""};
    FIELD std::map<int, int> field_with_spaces_in_its_type;
    FIELD int
                                field_on_the_next_line_of_its_type;
};

#include <gen/my_type.h.inl>
```

http://onqtam.com/programming/2017-09-02-simple-cpp-reflection-with-cmake/

3. И снова сериализация - свой парсер

```
any serialize(const Foo& src) {
    map<string, any> out;
    out["a"] = serialize(src.a);
    out["b"] = serialize(src.b);
    out["c"] = serialize(src.c);
    out["field_with_spaces_in_its_type"] = serialize(src.field_with_spaces_in_its_type
    out["field_on_the_next_line_of_its_type"] = serialize(src.field_on_the_next_line_o
    return out;
}
void deserialize(const any& src, Foo& dst) {
    const auto& data = any_cast<const map<string, any>&>(src);
    if(data.count("a")) { deserialize(data.at("a"), dst.a); }
    if(data.count("b")) { deserialize(data.at("b"), dst.b); }
    if(data.count("c")) { deserialize(data.at("c"), dst.c); }
    if(data.count("field_with_spaces_in_its_type")) { deserialize(data.at("field_with_
    if(data.count("field_on_the_next_line_of_its_type")) { deserialize(data.at("field_
}
```

http://ongtam.com/programming/2017-09-02-simple-cpp-reflection-with-cmake/

3. И снова сериализация - свой парсер

- Ограниченность применения
- Сложность поддержки и развития
- Сложность интеграции
- Позволяет вводить новые элементы в язык

И снова сериализация

DSL!

4. И снова сериализация - DSL

```
enum class Enum
{
        Item1,
        Item2
};

struct SimpleStruct
{
    int intField;
    std::string strField;
    Enum enumField;
};

struct ComplexStruct
{
    int intField;
    SimpleStruct innerStruct;
    std::vector<SimpleStruct> structVector;
    std::set<int> intSet;
};
```

4. И снова сериализация - DSL

```
enum Enum { Item1 = 0; Item2 = 1; }
```

```
struct SimpleStruct
{
    int intField;
    std::string strField;
    Enum enumField;
};

struct ComplexStruct
{
    int intField;
    SimpleStruct innerStruct;
    std::vector<SimpleStruct> structVector;
    std::set<int> intSet;
};
```

4. И снова сериализация - DSL

```
enum Enum { Item1 = 0; Item2 = 1; }
```

```
message SimpleStruct {
  sint32 intField = 1;
  string strField = 2;
  Enum enumField = 3;
}
message ComplexStruct {
  sint32 intField = 1;
  SimpleStruct innerStruct = 2;
  repeated
     SimpleStruct structVector = 3;
  repeated
     sint32 intSet = 4;
}
```

4. И снова сериализация - DSL

```
syntax = "proto3";
package serialization_test;

enum Enum { Item1 = 0; Item2 = 1; }


message SimpleStruct {
  sint32 intField = 1;
  string strField = 2;
  Enum enumField = 3;
}
message ComplexStruct {
  sint32 intField = 1;
  SimpleStruct innerStruct = 2;
  repeated
     SimpleStruct structVector = 3;
  repeated
     sint32 intSet = 4;
}
```

4. И снова сериализация - DSL

- Универсальность!
- Интеграция с разными языками и платформами!
- Стабильность реализации!

4. И снова сериализация - DSL

```
std::ostringstream os;
serialization_test::SimpleStruct pbValue;
pbValue.set_intfield(100500);
pbValue.set_strfield("Hello World!");
pbValue.set_enumfield(pb_Item1);
pbValue.SerializeToOstream(&os);
```

```
serialization_test::SimpleStruct pbValue;
 if (!pbValue.ParseFromIstream(&is))
   return false;

 auto intField = pbValue.intfield();
 auto strField = pbValue.strfield();
 auto enumField = pbValue.enumfield();
```

4. И снова сериализация - DSL

```
std::ostringstream os;
serialization_test::SimpleStruct pbValue;
pbValue.set_intfield(100500);
pbValue.set_strfield("Hello World!");
pbValue.set_enumfield(pb_Item1);
pbValue.SerializeToOstream(&os);
```

```
serialization_test::SimpleStruct pbValue;
 if (!pbValue.ParseFromIstream(&is))
   return false;

 auto intField = pbValue.intfield();
 auto strField = pbValue.strfield();
 auto enumField = pbValue.enumfield();
```

4. И снова сериализация - DSL

- Навязывают свой стиль и подход
- Требуют библиотек поддержки
- Возможности ограничены компилятором DSL и языком

И снова сериализация

В C++ до сих пор нет рефлексии!

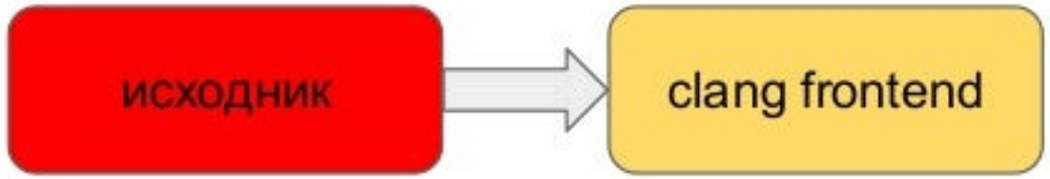
И снова сериализация

5. clang в генерации исходного кода

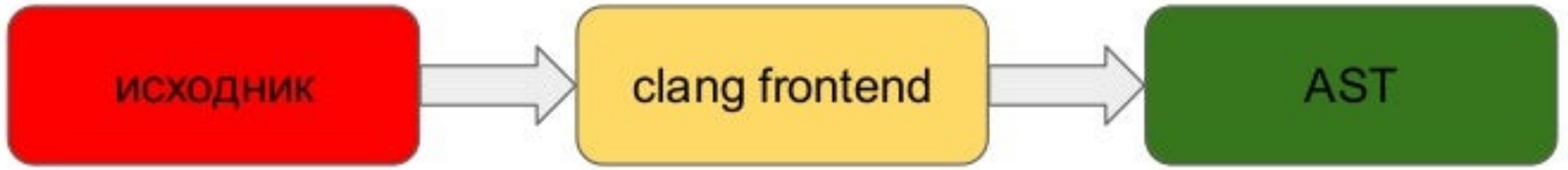
- libclang
- clangTooling

5. clang в генерации исходного кода

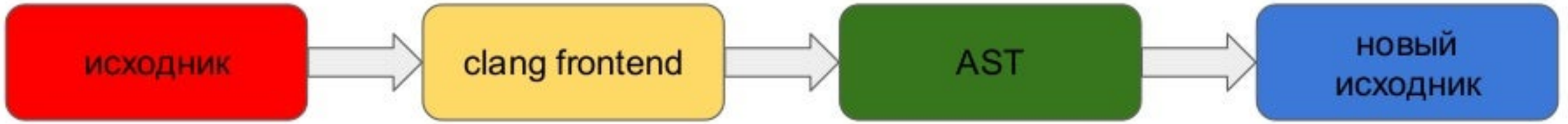
5. clang в генерации исходного кода



5. clang в генерации исходного кода



5. clang в генерации исходного кода



5. clang в генерации исходного кода

- Полноценный C++-компилятор со всеми его возможностями
- Доступ к коду на уровне AST

clang в генерации исходного кода

- Нетривиальное API
- Нестабильное API на уровне C++-библиотек

Решения на базе clang

- cppast
- tinyrefl
- автопрограммист

Решения на базе clang

- cppast (https://github.com/foonathan/cppast)
- tinyrefl (https://github.com/Manu343726/tinyrefl)
- автопрограммист (https://github.com/flexferrum/autoprogrammer)

Решения на базе clang

- Обёртки на clang API
- Предоставляют доступ к clang AST в более удобном виде

Решения на базе clang

cppast

6. cppast

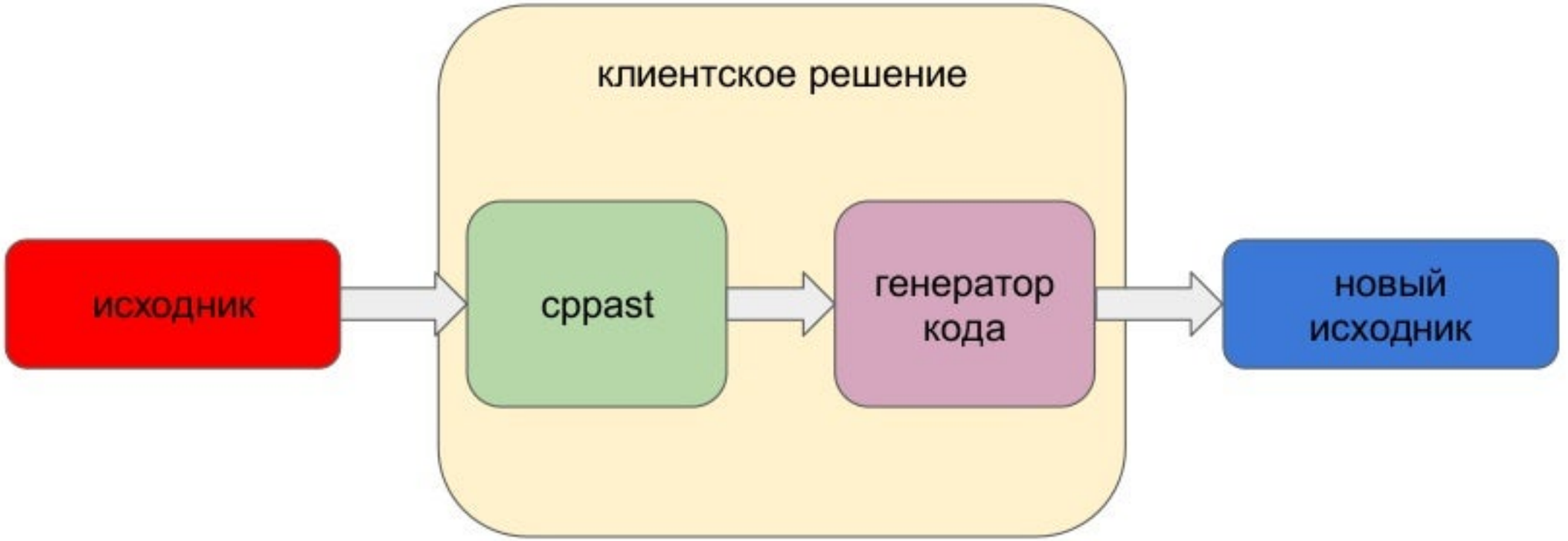
- Автор: **Jonathan Müller** (foonathan)
- Профиль: https://github.com/foonathan
- Лицензия: MIT

Обёртка над libclang, представляющая AST в виде набора C++-классов

6. cppast

- Библиотека
- Основана на **libclang** и инкапсулирует его
- Предоставляет основные классы для работы с AST: информация о сущностях, типах, выражениях и т. п.
- Предоставляет доступ к doxygen-комментариям в коде
- Поддерживает кастомные атрибуты в формате C++11

6. cppast



6. cppast

```
using parser_t = cppast::simple_file_parser<cppast::libclang_parser>;
cppast::cpp_entity_index index;
parser_t                 parser{type_safe::ref(index)};
parser_t::config      config;
config.set_flags(cppast::cpp_standard::cpp_14);
try
{
    auto file = parser.parse(filepath, config);
    if(file.has_value())
        visit_ast_and_generate(file.value());
    else
        std::cerr << "error parsing input file\n";
}
catch(const cppast::libclang_error& error)
{
    std::cerr << "[error] " << error.what() << "\n";
}
```

6. cppast

```
using parser_t = cppast::simple_file_parser<cppast::libclang_parser>;
cppast::cpp_entity_index index;
parser_t              parser{type_safe::ref(index)};
parser_t::config      config;
config.set_flags(cppast::cpp_standard::cpp_14);
try
{
    auto file = parser.parse(filepath, config);
    if(file.has_value())
        visit_ast_and_generate(file.value());
    else
        std::cerr << "error parsing input file\n";
}
catch(const cppast::libclang_error& error)
{
    std::cerr << "[error] " << error.what() << "\n";
}
```

6. cppast

```
using parser_t = cppast::simple_file_parser<cppast::libclang_parser>;
cppast::cpp_entity_index index;
parser_t                 parser{type_safe::ref(index)};
parser_t::config     config;
config.set_flags(cppast::cpp_standard::cpp_14);
try
{
    auto file = parser.parse(filepath, config);
    if(file.has_value())
        visit_ast_and_generate(file.value());
    else
        std::cerr << "error parsing input file\n";
}
catch(const cppast::libclang_error& error)
{
    std::cerr << "[error] " << error.what() << "\n";
}
```

6. cppast

```
using parser_t = cppast::simple_file_parser<cppast::libclang_parser>;
cppast::cpp_entity_index index;
parser_t                 parser{type_safe::ref(index)};
parser_t::config      config;
config.set_flags(cppast::cpp_standard::cpp_14);
try
{
    auto file = parser.parse(filepath, config);
    if(file.has_value())
        visit_ast_and_generate(file.value());
    else
        std::cerr << "error parsing input file\n";
}
catch(const cppast::libclang_error& error)
{
    std::cerr << "[error] " << error.what() << "\n";
}
```

6. cppast

```
using parser_t = cppast::simple_file_parser<cppast::libclang_parser>;
cppast::cpp_entity_index index;
parser_t                 parser{type_safe::ref(index)};
parser_t::config      config;
config.set_flags(cppast::cpp_standard::cpp_14);
try
{
    auto file = parser.parse(filepath, config);
    if(file.has_value())
        visit_ast_and_generate(file.value());
    else
        std::cerr << "error parsing input file\n";
}
catch(const cppast::libclang_error& error)
{
    std::cerr << "[error] " << error.what() << "\n";
}
```

6. cppast

```
void visit_ast_and_generate(const cppast::cpp_file& file)
{
cppast::visit(file,
[](const cppast::cpp_entity& e) {
    // ...
},
[](const cppast::cpp_entity& e, const cppast::visitor_info& info) {
    if (e.kind() == cppast::cpp_entity_kind::class_t && !info.is_old_entity())
    {
    // ...
    }
    else if (e.kind() == cppast::cpp_entity_kind::namespace_t)
    {
    // ...
    }
});
}
```

6. cppast

```
void visit_ast_and_generate(const cppast::cpp_file& file)
{
cppast::visit(file,
[](const cppast::cpp_entity& e) {
    // ...
},
[](const cppast::cpp_entity& e, const cppast::visitor_info& info) {
    if (e.kind() == cppast::cpp_entity_kind::class_t && !info.is_old_entity())
    {
    // ...
    }
    else if (e.kind() == cppast::cpp_entity_kind::namespace_t)
    {
    // ...
    }
});
}
```

6. cppast

```
void visit_ast_and_generate(const cppast::cpp_file& file)
{
cppast::visit(file,
[](const cppast::cpp_entity& e) {
    // ...
},
[](const cppast::cpp_entity& e, const cppast::visitor_info& info) {
    if (e.kind() == cppast::cpp_entity_kind::class_t && !info.is_old_entity())
    {
    // ...
    }
    else if (e.kind() == cppast::cpp_entity_kind::namespace_t)
    {
    // ...
    }
});
}
```

6. cppast

```
void visit_ast_and_generate(const cppast::cpp_file& file)
{
[](const cppast::cpp_entity& e) {
    // ...
},
[](const cppast::cpp_entity& e, const cppast::visitor_info& info) {
    if (e.kind() == cppast::cpp_entity_kind::class_t && !info.is_old_entity())
    {
    // ...
    }
    else if (e.kind() == cppast::cpp_entity_kind::namespace_t)
    {
    // ...
    }
});
}
```

6. cppast

```
void visit_ast_and_generate(const cppast::cpp_file& file)
{
[](const cppast::cpp_entity& e) {
    // ...
},
[](const cppast::cpp_entity& e, const cppast::visitor_info& info) {
    if (e.kind() == cppast::cpp_entity_kind::class_t && !info.is_old_entity())
    {
    // ...
    }
    else if (e.kind() == cppast::cpp_entity_kind::namespace_t)
    {
    // ...
    }
});
}
```

6. cppast

```
void visit_ast_and_generate(const cppast::cpp_file& file)
{
[](const cppast::cpp_entity& e) {
    // ...
},
[](const cppast::cpp_entity& e, const cppast::visitor_info& info) {
    if (e.kind() == cppast::cpp_entity_kind::class_t && !info.is_old_entity())
    {
    // ...
    }
    else if (e.kind() == cppast::cpp_entity_kind::namespace_t)
    {
    // ...
    }
});
}
```

6. cppast

```
[](const cppast::cpp_entity& e) {
    return (!cppast::is_templated(e)
            && e.kind() == cppast::cpp_entity_kind::class_t
            && cppast::is_definition(e)
           || e.kind() == cppast::cpp_entity_kind::namespace_t;
},
```

6. cppast

```
[](const cppast::cpp_entity& e) {
    return (!cppast::is_templated(e)
            && e.kind() == cppast::cpp_entity_kind::class_t
            && cppast::is_definition(e)
           || e.kind() == cppast::cpp_entity_kind::namespace_t;
},
```

6. cppast

```
[](const cppast::cpp_entity& e) {
    return (!cppast::is_templated(e)
            && e.kind() == cppast::cpp_entity_kind::class_t
            && cppast::is_definition(e)
           || e.kind() == cppast::cpp_entity_kind::namespace_t;
},
```

6. cppast

```
[](const cppast::cpp_entity& e) {
    return (!cppast::is_templated(e)
            && e.kind() == cppast::cpp_entity_kind::class_t
            && cppast::is_definition(e)
           || e.kind() == cppast::cpp_entity_kind::namespace_t;
},
```

6. cppast

```
[](const cppast::cpp_entity& e) {
    return (!cppast::is_templated(e)
            && e.kind() == cppast::cpp_entity_kind::class_t
            && cppast::is_definition(e)
            && cppast::has_attribute(e, "generate::serialize"))
           || e.kind() == cppast::cpp_entity_kind::namespace_t;
},
```

6. cppast

```
if (e.kind() == cppast::cpp_entity_kind::class_t && !info.is_old_entity())
{
    auto& class_ = static_cast<const cppast::cpp_class&>(e);
    std::cout << "void serialize(const serializer& s, const " << class_.name() << "& obj) {\n";
    for (auto& base : class_.bases())
        std::cout << " serialize(s, static_cast<const " << base.name() << "&>(obj));\n";

    for (auto& member : class_)
    {
        if (member.kind() == cppast::cpp_entity_kind::member_variable_t)
            generate_serialize_member(std::cout,
                                      static_cast<
                                          const cppast::cpp_member_variable&>(
                                          member));
    }

    std::cout << "}\n\n";
}
```

6. cppast

```
if (e.kind() == cppast::cpp_entity_kind::class_t && !info.is_old_entity())
{
    auto& class_ = static_cast<const cppast::cpp_class&>(e);
    std::cout << "void serialize(const serializer& s, const " << class_.name() << "& obj) {\n";
    for (auto& base : class_.bases())
        std::cout << "  serialize(s, static_cast<const " << base.name() << "&>(obj));\n";

    for (auto& member : class_)
    {
        if (member.kind() == cppast::cpp_entity_kind::member_variable_t)
            generate_serialize_member(std::cout,
                                      static_cast<
                                          const cppast::cpp_member_variable&>(
                                          member));
    }

    std::cout << "}\n\n";
}
```

6. cppast

```
if (e.kind() == cppast::cpp_entity_kind::class_t && !info.is_old_entity())
{
    auto& class_ = static_cast<const cppast::cpp_class&>(e);
    std::cout << "void serialize(const serializer& s, const " << class_.name() << "& obj) {\n";
    for (auto& base : class_.bases())
        std::cout << " serialize(s, static_cast<const " << base.name() << "&>(obj));\n";

    for (auto& member : class_)
    {
        if (member.kind() == cppast::cpp_entity_kind::member_variable_t)
            generate_serialize_member(std::cout,
                                      static_cast<
                                          const cppast::cpp_member_variable&>(
                                          member));
    }

    std::cout << "}\n\n";
}
```

6. cppast

```
if (e.kind() == cppast::cpp_entity_kind::class_t && !info.is_old_entity())
{
    auto& class_ = static_cast<const cppast::cpp_class&>(e);
    std::cout << "void serialize(const serializer& s, const " << class_.name() << "& obj) {\n";
    for (auto& base : class_.bases())
        std::cout << "  serialize(s, static_cast<const " << base.name() << "&>(obj));\n";

    for (auto& member : class_)
    {
        if (member.kind() == cppast::cpp_entity_kind::member_variable_t)
            generate_serialize_member(std::cout,
                                      static_cast<
                                          const cppast::cpp_member_variable&>(
                                          member));
    }

    std::cout << "}\n\n";
}
```

6. cppast

```
void generate_serialize_member(std::ostream& out, const cppast::cpp_member_variable& member)
{
    auto& type = cppast::remove_cv(member.type());
    if (type.kind() == cppast::cpp_type_kind::builtin_t)
    {
        out << "  s.serialize(obj." << member.name() << ");\n";
    }
    else if (type.kind() == cppast::cpp_type_kind::user_defined_t)
    {
        out << "  serialize(s, obj." << member.name() << ");\n";
    }
    else if (is_c_string(type))
    {
        out << "  s.serialize_string(obj." << member.name() << ");\n";
    }
    else
        throw std::invalid_argument("cannot serialize member " + member.name());
}
```

6. cppast

```
void generate_serialize_member(std::ostream& out, const cppast::cpp_member_variable& member)
{
    auto& type = cppast::remove_cv(member.type());
    if (type.kind() == cppast::cpp_type_kind::builtin_t)
    {
        out << "  s.serialize(obj." << member.name() << ");\n";
    }
    else if (type.kind() == cppast::cpp_type_kind::user_defined_t)
    {
        out << "  serialize(s, obj." << member.name() << ");\n";
    }
    else if (is_c_string(type))
    {
        out << "  s.serialize_string(obj." << member.name() << ");\n";
    }
    else
        throw std::invalid_argument("cannot serialize member " + member.name());
}
```

6. cppast

```
void generate_serialize_member(std::ostream& out, const cppast::cpp_member_variable& member)
{
    auto& type = cppast::remove_cv(member.type());
    if (type.kind() == cppast::cpp_type_kind::builtin_t)
    {
        out << "  s.serialize(obj." << member.name() << ");\n";
    }
    else if (type.kind() == cppast::cpp_type_kind::user_defined_t)
    {
        out << "  serialize(s, obj." << member.name() << ");\n";
    }
    else if (is_c_string(type))
    {
        out << "  s.serialize_string(obj." << member.name() << ");\n";
    }
    else
        throw std::invalid_argument("cannot serialize member " + member.name());
}
```

6. cppast

```
void generate_serialize_member(std::ostream& out, const cppast::cpp_member_variable& member)
{
    auto& type = cppast::remove_cv(member.type());
    if (type.kind() == cppast::cpp_type_kind::builtin_t)
    {
        out << "  s.serialize(obj." << member.name() << ");\n";
    }
    else if (type.kind() == cppast::cpp_type_kind::user_defined_t)
    {
        out << "  serialize(s, obj." << member.name() << ");\n";
    }
    else if (is_c_string(type))
    {
        out << "  s.serialize_string(obj." << member.name() << ");\n";
    }
    else
        throw std::invalid_argument("cannot serialize member " + member.name());
}
```

6. cppast

```
void generate_serialize_member(std::ostream& out, const cppast::cpp_member_variable& member)
{
    auto& type = cppast::remove_cv(member.type());
    if (type.kind() == cppast::cpp_type_kind::builtin_t)
    {
        out << "  s.serialize(obj." << member.name() << ");\n";
    }
    else if (type.kind() == cppast::cpp_type_kind::user_defined_t)
    {
        out << "  serialize(s, obj." << member.name() << ");\n";
    }
    else if (is_c_string(type))
    {
        out << "  s.serialize_string(obj." << member.name() << ");\n";
    }
    else
        throw std::invalid_argument("cannot serialize member " + member.name());
}
```

6. cppast

- Мощная, но достаточно низкоуровневая библиотека
- Можно использовать в качестве базы для своих утилит

Решения на базе clang

tinyrefl

7. tinyrefl

- Автор: **Manu Sánchez** (Manu343726)
- Профиль: https://github.com/Manu343726
- Лицензия: MIT

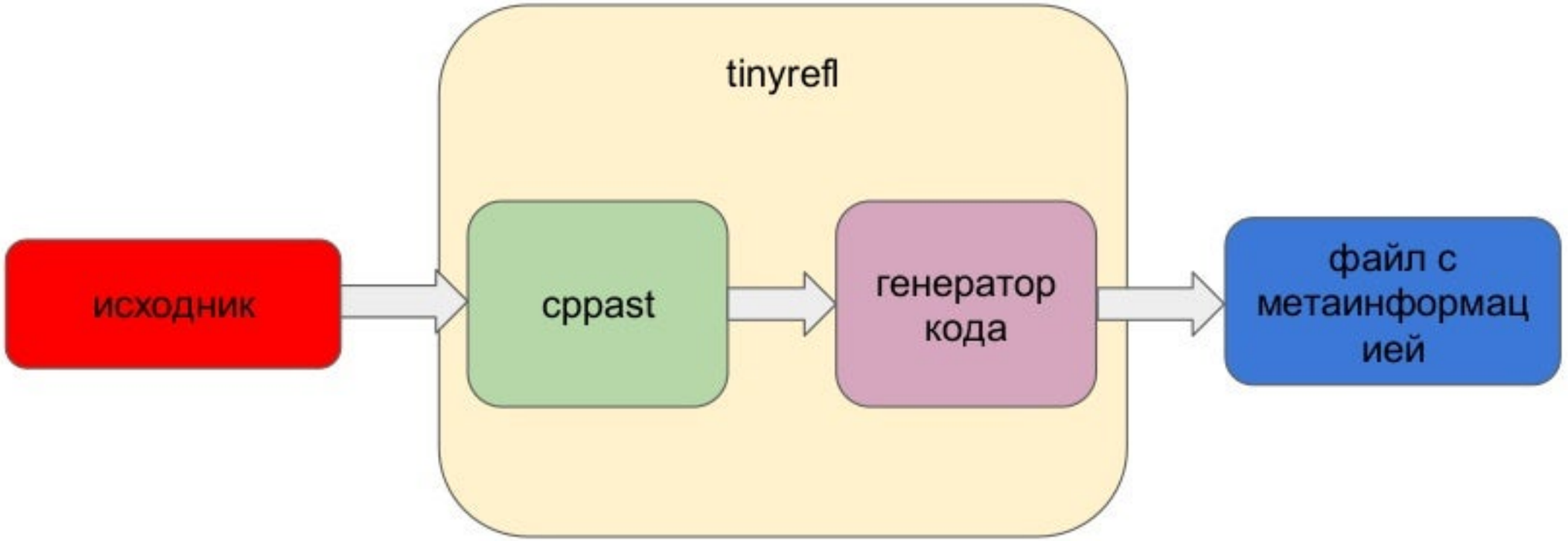
Один из вариантов реализации static reflection для C++.

Основана на **cppast**.

7. tinyrefl

- Утилита + библиотека поддержки
- Законченное решение
- Основана на **cppast**
- Эмулирует static reflection для C++
- Имеет интеграцию с **boost.hana** и **boost.fusion**
- Поддерживает кастомные атрибуты
- Расширяема

7. tinyrefl



7. tinyrefl

```
struct SimpleStruct
{
    int intField;
    std::string strField;
    Enum enumField;
};
```

7. tinyrefl

```
struct [[serializable]] SimpleStruct
{
    int intField;
    std::string strField;
    Enum enumField;
};
```

7. tinyrefl

```
template<typename Class>
auto JsonSerialize(rapidjson::Value& node,
                   Class&& structValue,
                   rapidjson::Document::AllocatorType& allocator) -> std::enable_if_t<
        tinyrefl::has_metadata<std::decay_t<Class>>() &&
        tinyrefl::has_attribute<std::decay_t<Class>>("serializable")>
{
    tinyrefl::visit_member_variables(structValue, [&node, &allocator](const auto& name, const auto& var) {
        rapidjson::Value innerNode;
        JsonSerialize(innerNode, var, allocator);
        node.AddMember(name.c_str(), innerNode.Move(), allocator);
    });
    return equal;
}
```

7. tinyrefl

```
template<typename Class>
auto JsonSerialize(rapidjson::Value& node,
                   Class&& structValue,
                   rapidjson::Document::AllocatorType& allocator) -> std::enable_if_t<
      tinyrefl::has_metadata<std::decay_t<Class>>() &&
      tinyrefl::has_attribute<std::decay_t<Class>>("serializable")>
{
   tinyrefl::visit_member_variables(structValue, [&node, &allocator](const auto& name, const auto& var) {
      rapidjson::Value innerNode;
      JsonSerialize(innerNode, var, allocator);
      node.AddMember(name.c_str(), innerNode.Move(), allocator);
   });
   return equal;
}
```

7. tinyrefl

```
template<typename Class>
auto JsonSerialize(rapidjson::Value& node,
                   Class&& structValue,
                   rapidjson::Document::AllocatorType& allocator) -> std::enable_if_t<
       tinyrefl::has_metadata<std::decay_t<Class>>() &&
       tinyrefl::has_attribute<std::decay_t<Class>>("serializable")>
{
    tinyrefl::visit_member_variables(structValue, [&node, &allocator](const auto& name, const auto& var) {
        rapidjson::Value innerNode;
        JsonSerialize(innerNode, var, allocator);
        node.AddMember(name.c_str(), innerNode.Move(), allocator);
    });
    return equal;
}
```

7. tinyrefl

```
template<typename Class>
auto JsonSerialize(rapidjson::Value& node,
                   Class&& structValue,
                   rapidjson::Document::AllocatorType& allocator) -> std::enable_if_t<
       tinyrefl::has_metadata<std::decay_t<Class>>() &&
       tinyrefl::has_attribute<std::decay_t<Class>>("serializable")>
{
    tinyrefl::visit_member_variables(structValue, [&node, &allocator](const auto& name, const auto& var) {
        rapidjson::Value innerNode;
        JsonSerialize(innerNode, var, allocator);
        node.AddMember(name.c_str(), innerNode.Move(), allocator);
    });
    return equal;
}
```

7. tinyrefl

```
#ifndef TINYREFL_GENERATED_FILE_10403593754518508410_INCLUDED
#define TINYREFL_GENERATED_FILE_10403593754518508410_INCLUDED

TINYREFL_REGISTER_FILE(TINYREFL_FILE((TINYREFL_STRING(test_structs.h));))
TINYREFL_REGISTER_CLASS(TINYREFL_CLASS((TINYREFL_STRING(SimpleStruct)),
                                       (TINYREFL_STRING(SimpleStruct))
                                       ))
TINYREFL_REGISTER_MEMBER(TINYREFL_MEMBER((TINYREFL_STRING(intField)),
                                       (TINYREFL_STRING(SimpleStruct::intField)),
                                       (TINYREFL_STRING(int)),
                                       (&SimpleStruct::intField)
                                       ))
TINYREFL_REGISTER_MEMBER(TINYREFL_MEMBER((TINYREFL_STRING(strField)),
                                       (TINYREFL_STRING(SimpleStruct::strField)),
                                       (TINYREFL_STRING(std::string)),
                                       (&SimpleStruct::strField)
                                       ))
TINYREFL_REGISTER_MEMBER(TINYREFL_MEMBER((TINYREFL_STRING(enumField)),
                                       (TINYREFL_STRING(SimpleStruct::enumField)),
                                       (TINYREFL_STRING(Enum)),
                                       (&SimpleStruct::enumField)
                                       ))

#endif // TINYREFL_GENERATED_FILE_10403593754518508410_INCLUDED
```

7. tinyrefl

```
#ifndef TINYREFL_GENERATED_FILE_10403593754518508410_INCLUDED
#define TINYREFL_GENERATED_FILE_10403593754518508410_INCLUDED

TINYREFL_REGISTER_FILE(TINYREFL_FILE((TINYREFL_STRING(test_structs.h));))
TINYREFL_REGISTER_CLASS(TINYREFL_CLASS((TINYREFL_STRING(SimpleStruct)),
                                       (TINYREFL_STRING(SimpleStruct))
                                       ))
TINYREFL_REGISTER_MEMBER(TINYREFL_MEMBER((TINYREFL_STRING(intField)),
                                         (TINYREFL_STRING(SimpleStruct::intField)),
                                         (TINYREFL_STRING(int)),
                                         (&SimpleStruct::intField)
                                         ))
TINYREFL_REGISTER_MEMBER(TINYREFL_MEMBER((TINYREFL_STRING(strField)),
                                         (TINYREFL_STRING(SimpleStruct::strField)),
                                         (TINYREFL_STRING(std::string)),
                                         (&SimpleStruct::strField)
                                         ))
TINYREFL_REGISTER_MEMBER(TINYREFL_MEMBER((TINYREFL_STRING(enumField)),
                                         (TINYREFL_STRING(SimpleStruct::enumField)),
                                         (TINYREFL_STRING(Enum)),
                                         (&SimpleStruct::enumField)
                                         ))

#endif // TINYREFL_GENERATED_FILE_10403593754518508410_INCLUDED
```

7. tinyrefl

```
#ifndef TINYREFL_GENERATED_FILE_10403593754518508410_INCLUDED
#define TINYREFL_GENERATED_FILE_10403593754518508410_INCLUDED

TINYREFL_REGISTER_FILE(TINYREFL_FILE((TINYREFL_STRING(test_structs.h));))
TINYREFL_REGISTER_CLASS(TINYREFL_CLASS((TINYREFL_STRING(SimpleStruct)),
                                       (TINYREFL_STRING(SimpleStruct))
                                       ))
TINYREFL_REGISTER_MEMBER(TINYREFL_MEMBER((TINYREFL_STRING(intField)),
                                         (TINYREFL_STRING(SimpleStruct::intField)),
                                         (TINYREFL_STRING(int)),
                                         (&SimpleStruct::intField)
                                         ))
TINYREFL_REGISTER_MEMBER(TINYREFL_MEMBER((TINYREFL_STRING(strField)),
                                         (TINYREFL_STRING(SimpleStruct::strField)),
                                         (TINYREFL_STRING(std::string)),
                                         (&SimpleStruct::strField)
                                         ))
TINYREFL_REGISTER_MEMBER(TINYREFL_MEMBER((TINYREFL_STRING(enumField)),
                                         (TINYREFL_STRING(SimpleStruct::enumField)),
                                         (TINYREFL_STRING(Enum)),
                                         (&SimpleStruct::enumField)
                                         ))

#endif // TINYREFL_GENERATED_FILE_10403593754518508410_INCLUDED
```

7. tinyrefl

- По заголовочному файлу генерирует заголовочный файл с разметкой сущностей в исходном файле
- С помощью API позволяет анализировать эту разметку в compile time
- Позволяет переопределять макросы с разметкой

Решения на базе clang

Автопрограммист

8. Автопрограммист

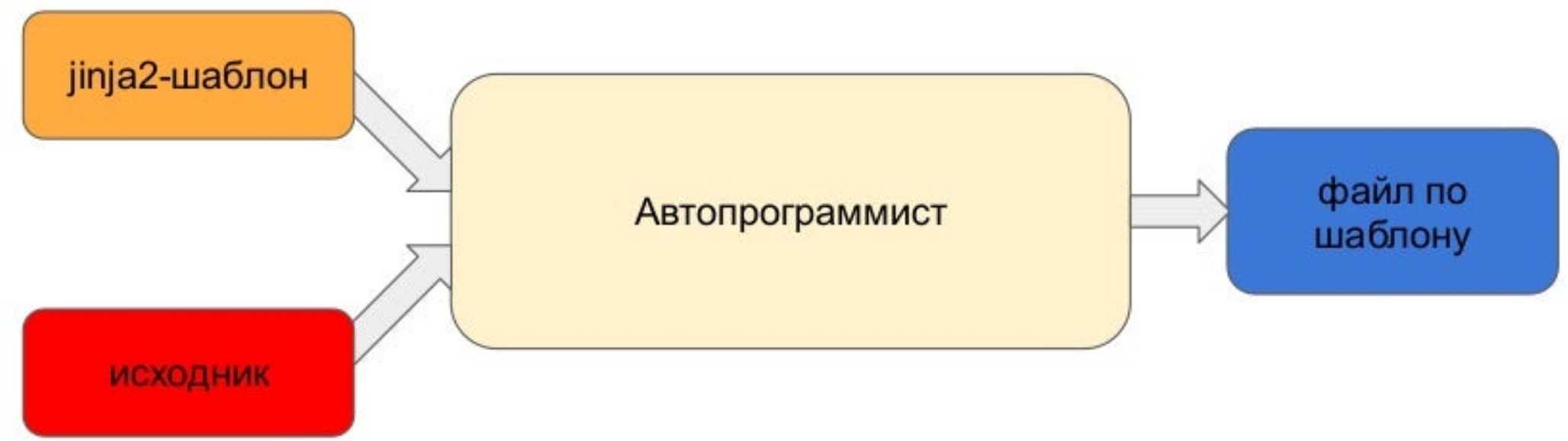
- Автор: **Сергей Садовников** (flexferrum)
- Профиль: https://github.com/flexferrum
- Лицензия: MIT

Генератор исходных текстов на основе clang tooling и шаблонов Jinja2

8. Автопрограммист

- Утилита
- Законченное решение
- Основана на **clang libTooling**
- Проецирует clang AST в Jinja2-шаблоны
- Расширяема

8. Автопрограммист



8. Автопрограммист

```
if (e.kind() == cppast::cpp_entity_kind::class_t && !info.is_old_entity())
{
    auto& class_ = static_cast<const cppast::cpp_class&>(e);
    std::cout << "void serialize(const serializer& s, const " << class_.name() << "& obj) {\n";
    for (auto& base : class_.bases())
        std::cout << " serialize(s, static_cast<const " << base.name() << "&>(obj));\n";

    for (auto& member : class_)
    {
        if (member.kind() == cppast::cpp_entity_kind::member_variable_t)
            generate_serialize_member(std::cout,
                                      static_cast<
                                          const cppast::cpp_member_variable&>(
                                          member));
    }

    std::cout << "}\n\n";
}
```

8. Автопрограммист

```
{% for ns in [rootNamespace] recursive %}
{% for s in ns.classes %}
void JsonSerialize(rapidjson::Value& node, const {{ s.fullQualifiedName }}& structValue, rapidjson::Document::AllocatorType&
allocator)
{
	using namespace rapidjson;

	node.SetObject();
	{% for m in s.members %}
	{
		Value value;
		{{ common.ProcessTypedMember(m.type, 'Serialize', 'structValue.' + m.name) }}
		node.AddMember("{{ m.name }}", value.Move(), allocator);
	}
	{% endfor %}
}
{% endfor %}
{{loop(ns.innerNamespaces)}}
{% endfor %}
```

8. Автопрограммист

```
{% for ns in [rootNamespace] recursive %}
{% for s in ns.classes %}
void JsonSerialize(rapidjson::Value& node, const {{ s.fullQualifiedName }}& structValue, rapidjson::Document::AllocatorType&
allocator)
{
	using namespace rapidjson;

	node.SetObject();
	{% for m in s.members %}
	{
		Value value;
		{{ common.ProcessTypedMember(m.type, 'Serialize', 'structValue.' + m.name) }}
		node.AddMember("{{ m.name }}", value.Move(), allocator);
	}
	{% endfor %}
}
{% endfor %}
{{loop(ns.innerNamespaces)}}
{% endfor %}
```

8. Автопрограммист

```
{% for ns in [rootNamespace] recursive %}
{% for s in ns.classes %}
void JsonSerialize(rapidjson::Value& node, const {{ s.fullQualifiedName }}& structValue, rapidjson::Document::AllocatorType&
allocator)
{
	using namespace rapidjson;

	node.SetObject();
	{% for m in s.members %}
	{
		Value value;
		{{ common.ProcessTypedMember(m.type, 'Serialize', 'structValue.' + m.name) }}
		node.AddMember("{{ m.name }}", value.Move(), allocator);
	}
	{% endfor %}
}
{% endfor %}
{{loop(ns.innerNamespaces)}}
{% endfor %}
```

8. Автопрограммист

```
{% for ns in [rootNamespace] recursive %}
{% for s in ns.classes %}
void JsonSerialize(rapidjson::Value& node, const {{ s.fullQualifiedName }}& structValue, rapidjson::Document::AllocatorType&
allocator)
{
	using namespace rapidjson;

	node.SetObject();
	{% for m in s.members %}
	{
		Value value;
		{{ common.ProcessTypedMember(m.type, 'Serialize', 'structValue.' + m.name) }}
		node.AddMember("{{ m.name }}", value.Move(), allocator);
	}
	{% endfor %}
}
{% endfor %}
{{loop(ns.innerNamespaces)}}
{% endfor %}
```

8. Автопрограммист

```
{% for ns in [rootNamespace] recursive %}
{% for s in ns.classes %}
void JsonSerialize(rapidjson::Value& node, const {{ s.fullQualifiedName }}& structValue,
rapidjson::Document::AllocatorType& allocator)
{
	using namespace rapidjson;

	node.SetObject();
	{% for m in s.members %}
	{
		Value value;
		{{ common.ProcessTypedMember(m.type, 'Serialize', 'structValue.' + m.name) }}
		node.AddMember("{{ m.name }}", value.Move(), allocator);
	}
	{% endfor %}
}
{% endfor %}
{{loop(ns.innerNamespaces)}}
{% endfor %}
```

8. Автопрограммист

```
{% for ns in [rootNamespace] recursive %}
{% for s in ns.classes %}
void JsonSerialize(rapidjson::Value& node, const {{ s.fullQualifiedName }}& structValue, rapidjson::Document::AllocatorType&
allocator)
{
	using namespace rapidjson;

	node.SetObject();
	{% for m in s.members %}
	{
		Value value;
		{{ common.ProcessTypedMember(m.type, 'Serialize', 'structValue.' + m.name) }}
		node.AddMember("{{ m.name }}", value.Move(), allocator);
	}
	{% endfor %}
}
{% endfor %}
{{loop(ns.innerNamespaces)}}
{% endfor %}
```

8. Автопрограммист

```
{% for ns in [rootNamespace] recursive %}
{% for s in ns.classes %}
void JsonSerialize(rapidjson::Value& node, const {{ s.fullQualifiedName }}& structValue, rapidjson::Document::AllocatorType&
allocator)
{
	using namespace rapidjson;

	node.SetObject();
	{% for m in s.members %}
	{
		Value value;
		{{ common.ProcessTypedMember(m.type, 'Serialize', 'structValue.' + m.name) }}
		node.AddMember("{{ m.name }}", value.Move(), allocator);
	}
	{% endfor %}
}
{% endfor %}
{{loop(ns.innerNamespaces)}}
{% endfor %}
```

8. Автопрограммист

```
void generate_serialize_member(std::ostream& out, const cppast::cpp_member_variable& member)
{
    auto& type = cppast::remove_cv(member.type());
    if (type.kind() == cppast::cpp_type_kind::builtin_t)
    {
        out << "  s.serialize(obj." << member.name() << ");\n";
    }
    else if (type.kind() == cppast::cpp_type_kind::user_defined_t)
    {
        out << "  serialize(s, obj." << member.name() << ");\n";
    }
    else if (is_c_string(type))
    {
        out << "  s.serialize_string(obj." << member.name() << ");\n";
    }
    else
        throw std::invalid_argument("cannot serialize member " + member.name());
}
```

8. Автопрограммист

```
{% macro ProcessTypedMember(type, macroPrefix, extraVal) %}
        {% if type.type.isNoType %}
        {% elif type.type.isBuiltinType %}
        {{ extraVal | applymacro(macro=macroPrefix + 'BuiltinType', type, varargs[0], varargs[1], varargs[2] ) }}
        {% elif type.type.isRecordType %}
        {{ extraVal | applymacro(macro=macroPrefix + 'RecordType', type, varargs[0], varargs[1], varargs[2] ) }}
        {% elif type.type.isTemplateType %}
        {{ extraVal | applymacro(macro=macroPrefix + 'TemplateType', type, varargs[0], varargs[1], varargs[2] ) }}
        {% elif type.type.isWellKnownType %}
        {{ extraVal | applymacro(macro=macroPrefix + 'WellKnownType', type, varargs[0], varargs[1], varargs[2] ) }}
        {% elif type.type.isArrayType %}
        {{ extraVal | applymacro(macro=macroPrefix + 'ArrayType', type, type.type.dims, type.type.itemType, varargs[0], varargs[1],
varargs[2] ) }}
        {% elif type.type.isEnumType %}
        {{ extraVal | applymacro(macro=macroPrefix + 'EnumType', type, varargs[0], varargs[1], varargs[2] ) }}
        {% else %}
        {{ extraVal | applymacro(macro=macroPrefix + 'GenericType', type, varargs[0], varargs[1], varargs[2] ) }}
        {% endif %}
{% endmacro %}
```

8. Автопрограммист

```
{% macro ProcessTypedMember(type, macroPrefix, extraVal) %}
        {% if type.type.isNoType %}
        {% elif type.type.isBuiltinType %}
        {{ extraVal | applymacro(macro=macroPrefix + 'BuiltinType', type, varargs[0], varargs[1], varargs[2] ) }}
        {% elif type.type.isRecordType %}
        {{ extraVal | applymacro(macro=macroPrefix + 'RecordType', type, varargs[0], varargs[1], varargs[2] ) }}
        {% elif type.type.isTemplateType %}
        {{ extraVal | applymacro(macro=macroPrefix + 'TemplateType', type, varargs[0], varargs[1], varargs[2] ) }}
        {% elif type.type.isWellKnownType %}
        {{ extraVal | applymacro(macro=macroPrefix + 'WellKnownType', type, varargs[0], varargs[1], varargs[2] ) }}
        {% elif type.type.isArrayType %}
        {{ extraVal | applymacro(macro=macroPrefix + 'ArrayType', type, type.type.dims, type.type.itemType, varargs[0], varargs[1],
varargs[2] ) }}
        {% elif type.type.isEnumType %}
        {{ extraVal | applymacro(macro=macroPrefix + 'EnumType', type, varargs[0], varargs[1], varargs[2] ) }}
        {% else %}
        {{ extraVal | applymacro(macro=macroPrefix + 'GenericType', type, varargs[0], varargs[1], varargs[2] ) }}
        {% endif %}
{% endmacro %}
```

8. Автопрограммист

```
{% macro ProcessTypedMember(type, macroPrefix, extraVal) %}
        {% if type.type.isNoType %}
        {% elif type.type.isBuiltinType %}
        {{ extraVal | applymacro(macro=macroPrefix + 'BuiltinType', type, varargs[0], varargs[1], varargs[2] ) }}
        {% elif type.type.isRecordType %}
        {{ extraVal | applymacro(macro=macroPrefix + 'RecordType', type, varargs[0], varargs[1], varargs[2] ) }}
        {% elif type.type.isTemplateType %}
        {{ extraVal | applymacro(macro=macroPrefix + 'TemplateType', type, varargs[0], varargs[1], varargs[2] ) }}
        {% elif type.type.isWellKnownType %}
        {{ extraVal | applymacro(macro=macroPrefix + 'WellKnownType', type, varargs[0], varargs[1], varargs[2] ) }}
        {% elif type.type.isArrayType %}
        {{ extraVal | applymacro(macro=macroPrefix + 'ArrayType', type, type.type.dims, type.type.itemType, varargs[0], varargs[1],
varargs[2] ) }}
        {% elif type.type.isEnumType %}
        {{ extraVal | applymacro(macro=macroPrefix + 'EnumType', type, varargs[0], varargs[1], varargs[2] ) }}
        {% else %}
        {{ extraVal | applymacro(macro=macroPrefix + 'GenericType', type, varargs[0], varargs[1], varargs[2] ) }}
        {% endif %}
{% endmacro %}
```

8. Автопрограммист

```
{% macro SerializeBuiltinType(valueRef, typeInfo) %}
   value = {{ valueRef }};
{% endmacro %}
```

8. Автопрограммист

```
{% macro SerializeBuiltinType(valueRef, typeInfo) %}
  value = {{ valueRef }};
{% endmacro %}
```

8. Автопрограммист

```
{% macro SerializeWellKnownType(valueRef, typeInfo) %}
	{% if typeInfo.type.type == 'StdString' %}
		const auto& str = {{ valueRef }};
		value.SetString(StringRef(str.data(), str.size()));
	{% elif typeInfo.type.type == 'StdVector' or typeInfo.type.type == 'StdArray' or typeInfo.type.type == 'StdList' or
typeInfo.type.type == 'StdSet' %}
		value.SetArray();
		for (auto& v : {{ valueRef }})
		{
			auto& array = value;
			Value value;
			{{ common.ProcessTypedMember(typeInfo.type.arguments[0], 'Serialize', 'v') }}
			array.PushBack(value.Move(), allocator);
		}
	{% else %}
	// Make handler for well-known type {{ typeInfo.type.type }}
	{% endif %}
{% endmacro %}
```

8. Автопрограммист

```
{% macro SerializeWellKnownType(valueRef, typeInfo) %}
	{% if typeInfo.type.type == 'StdString' %}
		const auto& str = {{ valueRef }};
		value.SetString(StringRef(str.data(), str.size()));
	{% elif typeInfo.type.type == 'StdVector' or typeInfo.type.type == 'StdArray' or typeInfo.type.type == 'StdList' or
typeInfo.type.type == 'StdSet' %}
		value.SetArray();
		for (auto& v : {{ valueRef }})
		{
			auto& array = value;
			Value value;
			{{ common.ProcessTypedMember(typeInfo.type.arguments[0], 'Serialize', 'v') }}
			array.PushBack(value.Move(), allocator);
		}
	{% else %}
	// Make handler for well-known type {{ typeInfo.type.type }}
	{% endif %}
{% endmacro %}
```

8. Автопрограммист

```
{% macro SerializeWellKnownType(valueRef, typeInfo) %}
	{% if typeInfo.type.type == 'StdString' %}
		const auto& str = {{ valueRef }};
		value.SetString(StringRef(str.data(), str.size()));
	{% elif typeInfo.type.type == 'StdVector' or typeInfo.type.type == 'StdArray' or typeInfo.type.type == 'StdList' or typeInfo.type.type == 'StdSet' %}
		value.SetArray();
		for (auto& v : {{ valueRef }})
		{
			auto& array = value;
			Value value;
			{{ common.ProcessTypedMember(typeInfo.type.arguments[0], 'Serialize', 'v') }}
			array.PushBack(value.Move(), allocator);
		}
	{% else %}
	// Make handler for well-known type {{ typeInfo.type.type }}
	{% endif %}
{% endmacro %}
```

8. Автопрограммист

```
void JsonSerialize(rapidjson::Value &node, const SimpleStruct &structValue, rapidjson::Document::AllocatorType &allocator) {
  using namespace rapidjson;
  node.SetObject();
  {
    Value value;
    value = structValue.intField;
    node.AddMember("intField", value.Move(), allocator);
  }
  {
    Value value;
    const auto &str = structValue.strField;
    value.SetString(StringRef(str.data(), str.size()));
    node.AddMember("strField", value.Move(), allocator);
  }
  {
    Value value;
    JsonSerialize(value, structValue.enumField, allocator);
    node.AddMember("enumField", value.Move(), allocator);
  }
}
```

8. Автопрограммист

- Полностью интерпретируемы
- Мощный python-подобный язык
- Глубоко интегрированы в ядро автопрограммиста
- Позволяют генерировать любой текстовый файл

8. Автопрограммист

```
{% import "common_macros.j2tpl" as common with context %}
syntax = "proto3";
{% if env.package_name %}package {{ env.package_name }};{% endif %}

{% for ns in [rootNamespace] recursive %}
{% for s in ns.classes %}
message {{ MakeProtobufName(s.fullQualifiedName) }} {
{% for m in s.members %}
{% set member_decl = common.ProcessTypedMember(m.type, 'ProtobufMember', m.name) | trim %}
{% if member_decl != '' %}{{ member_decl }} = {{ loop.index }};{% endif %}
{% endfor +%}
}
{% endfor %}
{{ loop(ns.innerNamespaces) }}
{% endfor %}
```

8. Автопрограммист

```
syntax = "proto3";
package serialization_test;

message SimpleStruct {
  sint32 intField = 1;
  string strField = 2;
  Enum enumField = 3;
}
```

8. Автопрограммист

```
{% import "common_macros.j2tpl" as common with context %}
syntax = "proto3";
{% if env.package_name %}package {{ env.package_name }};{% endif %}
{% for ns in [rootNamespace] recursive %}

{% for s in ns.classes %}
message {{ MakeProtobufName(s.fullQualifiedName) }} {
{% for m in s.members %}
{% set member_decl = common.ProcessTypedMember(m.type, 'ProtobufMember', m.name) | trim %}
{% if member_decl != '' %}{{ member_decl }} = {{ loop.index }};{% endif %}
{% endfor +%}
}
{% endfor %}
{{loop(ns.innerNamespaces)}}
{% endfor %}
```

8. Автопрограммист

```
{% for ns in [rootNamespace] recursive %}
{% for s in ns.classes %}
template<typename T>
void {{ common.Underscorize(s.fullQualifiedName) }}ToProtoBuf(const {{ s.fullQualifiedName }}& value, T& pbValue)
{
   {% for m in s.members %}
      {{ common.ProcessTypedMember(m.type, 'Serialize', 'value.' + m.name, m.name) }}
   {% endfor +%}
}

void SerializeToStream(const {{ s.fullQualifiedName }}& value, std::ostream& os)
{
   {{ MakeProtobufName(s.fullQualifiedName) }} pbValue;
   {{ common.Underscorize(s.fullQualifiedName) }}ToProtoBuf(value, pbValue);
   pbValue.SerializeToOstream(&os);
}
{% endfor %}
{{loop(ns.innerNamespaces)}}
{% endfor %}
```

8. Автопрограммист

```
template <typename T>
void SimpleStructToProtoBuf(const SimpleStruct &value, T &pbValue) {
  pbValue.set_intfield(value.intField);

  pbValue.set_strfield(value.strField);

  pbValue.set_enumfield(EnumToProtoBuf(value.enumField));
}

void SerializeToStream(const SimpleStruct &value, std::ostream &os) {
  serialization_test::SimpleStruct pbValue;
  SimpleStructToProtoBuf(value, pbValue);
  pbValue.SerializeToOstream(&os);
}
```

8. Автопрограммист

```
template <typename T>
void SimpleStructToProtoBuf(const SimpleStruct &value, T &pbValue) {
  pbValue.set_intfield(value.intField);

  pbValue.set_strfield(value.strField);

  pbValue.set_enumfield(EnumToProtoBuf(value.enumField));
}

void SerializeToStream(const SimpleStruct &value, std::ostream &os) {
  serialization_test::SimpleStruct pbValue;
  SimpleStructToProtoBuf(value, pbValue);
  pbValue.SerializeToOstream(&os);
}
```

8. Автопрограммист

```
{% macro SerializeBuiltinType(memberRef, typeInfo, memberName) %}
    pbValue.set_{{ memberName | lower }}({{ memberRef }});
{% endmacro %}
```

8. Автопрограммист

```
{% macro SerializeBuiltinType(memberRef, typeInfo, memberName) %}
    pbValue.set_{{ memberName | lower }}({{ memberRef }});
{% endmacro %}
```

8. Автопрограммист

```
macro(GenerateFile OUTPUT_FILE_NAME TEMPLATE_NAME)
add_custom_command(OUTPUT ${CODEGEN_DIR}/generated/${OUTPUT_FILE_NAME}
   fl-codegen -gen-serialization -ohdr ${CODEGEN_DIR}/generated/${OUTPUT_FILE_NAME} -input
${CMAKE_CURRENT_SOURCE_DIR}/test_structs.h -tpl-dir ${CMAKE_CURRENT_SOURCE_DIR}/templates -tpl
${TEMPLATE_NAME} ${ARGV2} ${ARGV3} -- clang-cl -std=c++14 -x c++ ${CMAKE_CXX_FLAGS}
   MAIN_DEPENDENCY ${CMAKE_CURRENT_SOURCE_DIR}/test_structs.h
   DEPENDS ${CODEGEN_BIN_NAME} ${CMAKE_CURRENT_SOURCE_DIR}/templates/${TEMPLATE_NAME}
   COMMENT "Generating serialization converters for ${CMAKE_CURRENT_SOURCE_DIR}/test_structs.h with template
${TEMPLATE_NAME}"
   )

   list(APPEND GENERATED_FILES ${CODEGEN_DIR}/generated/${OUTPUT_FILE_NAME})
endmacro()

GenerateFile(json_serialization.h json_serialization.h.j2tpl)
GenerateFile(protobuf_serialization.proto protobuf_serialization.proto.j2tpl "-tpl-param" "package_name=serialization_test")
GenerateFile(protobuf_serialization.cpp protobuf_serialization.cpp.j2tpl "-tpl-param" "package_name=serialization_test")
GenerateFile(protobuf_serialization.h protobuf_serialization.h.j2tpl)
```

8. Автопрограммист

```
macro(GenerateFile OUTPUT_FILE_NAME TEMPLATE_NAME)
add_custom_command(OUTPUT ${CODEGEN_DIR}/generated/${OUTPUT_FILE_NAME}
   fl-codegen -gen-serialization -ohdr ${CODEGEN_DIR}/generated/${OUTPUT_FILE_NAME} -input
${CMAKE_CURRENT_SOURCE_DIR}/test_structs.h -tpl-dir ${CMAKE_CURRENT_SOURCE_DIR}/templates -tpl
${TEMPLATE_NAME} ${ARGV2} ${ARGV3} -- clang-cl -std=c++14 -x c++ ${CMAKE_CXX_FLAGS}
   MAIN_DEPENDENCY ${CMAKE_CURRENT_SOURCE_DIR}/test_structs.h
   DEPENDS ${CODEGEN_BIN_NAME} ${CMAKE_CURRENT_SOURCE_DIR}/templates/${TEMPLATE_NAME}
   COMMENT "Generating serialization converters for ${CMAKE_CURRENT_SOURCE_DIR}/test_structs.h with template
${TEMPLATE_NAME}"
   )

   list(APPEND GENERATED_FILES ${CODEGEN_DIR}/generated/${OUTPUT_FILE_NAME})
endmacro()

GenerateFile(json_serialization.h json_serialization.h.j2tpl)
GenerateFile(protobuf_serialization.proto protobuf_serialization.proto.j2tpl "-tpl-param" "package_name=serialization_test")
GenerateFile(protobuf_serialization.cpp protobuf_serialization.cpp.j2tpl "-tpl-param" "package_name=serialization_test")
GenerateFile(protobuf_serialization.h protobuf_serialization.h.j2tpl)
```

8. Автопрограммист

```
macro(GenerateFile OUTPUT_FILE_NAME TEMPLATE_NAME)
add_custom_command(OUTPUT ${CODEGEN_DIR}/generated/${OUTPUT_FILE_NAME}
  fl-codegen -gen-serialization -ohdr ${CODEGEN_DIR}/generated/${OUTPUT_FILE_NAME} -input
${CMAKE_CURRENT_SOURCE_DIR}/test_structs.h -tpl-dir ${CMAKE_CURRENT_SOURCE_DIR}/templates -tpl
${TEMPLATE_NAME} ${ARGV2} ${ARGV3} -- clang-cl -std=c++14 -x c++ ${CMAKE_CXX_FLAGS}
  MAIN_DEPENDENCY ${CMAKE_CURRENT_SOURCE_DIR}/test_structs.h
  DEPENDS ${CODEGEN_BIN_NAME} ${CMAKE_CURRENT_SOURCE_DIR}/templates/${TEMPLATE_NAME}
  COMMENT "Generating serialization converters for ${CMAKE_CURRENT_SOURCE_DIR}/test_structs.h with template
${TEMPLATE_NAME}"
  )

  list(APPEND GENERATED_FILES ${CODEGEN_DIR}/generated/${OUTPUT_FILE_NAME})
endmacro()

GenerateFile(json_serialization.h json_serialization.h.j2tpl)
GenerateFile(protobuf_serialization.proto protobuf_serialization.proto.j2tpl "-tpl-param" "package_name=serialization_test")
GenerateFile(protobuf_serialization.cpp protobuf_serialization.cpp.j2tpl "-tpl-param" "package_name=serialization_test")
GenerateFile(protobuf_serialization.h protobuf_serialization.h.j2tpl)
```

8. Автопрограммист

```
macro(GenerateFile OUTPUT_FILE_NAME TEMPLATE_NAME)
add_custom_command(OUTPUT ${CODEGEN_DIR}/generated/${OUTPUT_FILE_NAME}
  fl-codegen -gen-serialization -ohdr ${CODEGEN_DIR}/generated/${OUTPUT_FILE_NAME} -input
${CMAKE_CURRENT_SOURCE_DIR}/test_structs.h -tpl-dir ${CMAKE_CURRENT_SOURCE_DIR}/templates -tpl
${TEMPLATE_NAME} ${ARGV2} ${ARGV3} -- clang-cl -std=c++14 -x c++ ${CMAKE_CXX_FLAGS}
  MAIN_DEPENDENCY ${CMAKE_CURRENT_SOURCE_DIR}/test_structs.h
  DEPENDS ${CODEGEN_BIN_NAME} ${CMAKE_CURRENT_SOURCE_DIR}/templates/${TEMPLATE_NAME}
  COMMENT "Generating serialization converters for ${CMAKE_CURRENT_SOURCE_DIR}/test_structs.h with template
${TEMPLATE_NAME}"
  )

  list(APPEND GENERATED_FILES ${CODEGEN_DIR}/generated/${OUTPUT_FILE_NAME})
endmacro()

GenerateFile(json_serialization.h json_serialization.h.j2tpl)
GenerateFile(protobuf_serialization.proto protobuf_serialization.proto.j2tpl "-tpl-param"
"package_name=serialization_test")
GenerateFile(protobuf_serialization.cpp protobuf_serialization.cpp.j2tpl "-tpl-param" "package_name=serialization_test")
GenerateFile(protobuf_serialization.h protobuf_serialization.h.j2tpl)
```

8. Автопрограммист

- Не требует библиотек поддержки, если это не нужно генерируемому коду
- Логика генерации вынесена за пределы утилиты
- Требуется знание нотации шаблонов Jinja2

Резюме

- Сериализация
- Верификаторы и конверторы конфигураций
- RPC (включая Object RPC)
- ORM
- Биндинги к python и другим скриптовым языкам
- Заготовки для автотестов и mock’и
- Что угодно ещё!

Резюме

- Что можно генерировать - **нужно** генерировать!
- Мощные средства для анализа C++ - относительно легко доступны
- Выбор средства определяется задачей

Резюме

```
if (e.kind() == cppast::cpp_entity_kind::class_t && !info.is_old_entity())
{
    auto& class_ = static_cast<const cppast::cpp_class&>(e);
    std::cout << "void serialize(const serializer& s, const " << class_.name() << "& obj) {\n";
    for (auto& base : class_.bases())
        std::cout << " serialize(s, static_cast<const " << base.name() << "&>(obj));\n";

    for (auto& member : class_)
    {
        if (member.kind() == cppast::cpp_entity_kind::member_variable_t)
            generate_serialize_member(std::cout,
                                      static_cast<
                                          const cppast::cpp_member_variable&>(
                                          member));
    }

    std::cout << "}\n\n";
}
```

Резюме

```
if (e.kind() == cppast::cpp_entity_kind::class_t && !info.is_old_entity())
{
    autc
    std::
    for (
      st
    for (
    {
      if
    }
    std::
}
```

```
template<typename Class>
auto JsonSerialize(rapidjson::Value& node,
                   Class&& structValue,
                   rapidjson::Document::AllocatorType& allocator) -> std::enable_if_t<
    tinyrefl::has_metadata<std::decay_t<Class>>() &&
    tinyrefl::has_attribute<std::decay_t<Class>>("serializable")>
{
    tinyrefl::visit_member_variables(structValue, [&node, &allocator](const auto& name, const auto& var) {
        rapidjson::Value innerNode;
        JsonSerialize(innerNode, var, allocator);
        node.AddMember(name.c_str(), innerNode.Move(), allocator);
    });
    return equal;
}
```

Резюме

```
if (e.kind() == cppast::cpp_entity_kind::class_t && !info.is_old_entity())
{
    autc
    std::
    for (
        st
    for (
    {
        if
    }
    std::
}
```

```
template<typename Class>
auto JsonSerialize(rapidjson::Value& node
    {% macro SerializeWellKnownType(valueRef, typeInfo) %}
    {% for s in ns.classes %}
    void JsonSerialize(rapidjson::Value& node, const {{ s.fullQualifiedName }}& structValue, rapidjson::...
        allocator)
    {
        using namespace rapidjson;

        node.SetObject();
        {% for m in s.members %}
        {
            Value value;
            {{ common.ProcessTypedMember(m.type, 'Serialize', 'structValue.' + m.name) }}
            node.AddMember("{{ m.name }}", value.Move(), allocator);
        }
        {% endfor %}
    }
{ tiny
    r
    J
    r
});
    retu
}
```

Полезные ссылки

- cppast: http://bit.ly/34wJAFB
- tinyrefl: http://bit.ly/37Spgk6
- автопрограммист: http://bit.ly/2OtYHK9
- презентация про использование tinyrefl для генерации автотестов: http://bit.ly/2OTE0GI
- презентация по автопрограммисту: http://bit.ly/2rA6xZK
- примеры генераторов сериализаторов на базе автопрограммиста: http://bit.ly/2stj8hS

Спасибо!

Сергей Садовников

e-mail: flexferrum@gmail.com twitter: @flex_ferrum