Reflection в C++ и коты

Василий Немков

Немного о себе

Что такое Reflection?

- Reflection (рефлексия) - способность программы наблюдать и менять свою структуру.

Что такое Reflection?

```
class Cat:
      class Color(Enum):
            White = 0
            Red = 1
            Gray = 2
            Black = 3
      class Sex(Enum):
            Female = 0
            Male = 1
      def __init__(self, name, birthday, furColor, sex):
            self.name = name
            self.birthday = birthday
            self.furColor = furColor
            self.sex = sex

cat = Cat('Bonch', datetime.date(2010, 9, 10), Cat.Color.Red, Cat.Sex.Female)
```

Что такое Reflection?

```
>>> dir(cat)
['Color', 'Sex', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__',
'__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'birthday',
'furColor', 'name', 'sex']

>>> dir(Cat)
['Color', 'Sex', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__',
'__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__']
```

Что такое Reflection?

```
>>> dir(cat)
['Color', 'Sex', … , 'birthday', 'furColor', 'name', 'sex']

>>> dir(Cat)
['Color', 'Sex', … , '__str__', '__subclasshook__', '__weakref__']
```

Что такое Reflection?

```
>>> dir(cat)
['Color', 'Sex', … , 'birthday', 'furColor', 'name', 'sex']

>>> dir(Cat)
['Color', 'Sex', … , '__str__', '__subclasshook__', '__weakref__']

>>> dir(1)
['__abs__', '__add__', '__and__', … , 'numerator', 'real', 'to_bytes']

>>> dir(len)
['__call__', '__class__', '__delattr__', … , '__text_signature__']

>>> dir(math)
['__doc__', '__loader__', '__name__', … , 'tan', 'tanh', 'trunc']
```

Что такое Reflection?

- Reflection (рефлексия) - способность программы наблюдать и менять свою структуру во время исполнения
- Языки в которых поддерживается reflection: Java, C# (.Net), Python, D...
- Compile-time reflection vs run-time reflection

Варианты использования reflection:

- Сериализация
- Интеграция с плагинами и скриптами
- Операторы сравнения
- Изменение кода в процессе выполнения

CMS - Cat Management System

```
class Cat {
  enum FurColor {
    White = 0,
    Red = 1,
    Gray = 2,
    Black = 4
  };
  enum Sex {
    Female = 0,
    Male = 1
  };

  std::string name;
  std::time_t birthday;
  std::bitset<8> furColor;
  Sex sex;
};
```

Сериализация

- Что такое сериализация?
- Зачем её использовать?
- Форматы: JSON, XML, protobuf и т.д.

Рефлексия вручную

```
void serialize(Cat const& cat, Json::Value* dest)
{
  (*dest)["name"] = cat.name;
  (*dest)["birthday"] = cat.birthday;
  (*dest)["furColor"] = cat.furColor;
  (*dest)["sex"] = cat.sex;
}

void deserialize(Json::Value const& source, Cat* cat)
{
  cat->name = source["name"].asString();
  cat->birthday = source["birthday"].asInt64();
  cat->furColor = source["furColor"].asUInt();
  cat->sex = static_cast<Cat::Sex>(source["sex"].asUInt());
}
```

Рефлексия вручную

Плюсы

- Простота
- Полный контроль над тем, как сериализуется\десериализуется объект

Минусы

- Много однотипного кода, приходится писать каждый раз для всех классов.
- Хрупкость - легко забыть добавить обработку нового атрибута.

Рефлексия с помощью Qt

- Meta-object system
- Meta-object compiler
- Run-time reflection

Рефлексия с помощью Qt

```
class qtCat : public QObject {
  Q_OBJECT
public:
  explicit qtCat(QObject *parent = 0);

  enum FurColor {White = 0, Red = 1, Gray = 2, Black = 4};
  Q_DECLARE_FLAGS(FurColorFlag, FurColor)
  Q_FLAG(FurColorFlag)

  enum Sex {Female = 0, Male = 1};
  Q_ENUM(Sex)

  Q_PROPERTY(QString name)
  Q_PROPERTY(QDateTime birthday)
  Q_PROPERTY(FurColorFlag furColor)
  Q_PROPERTY(Sex sex)
};
```

Рефлексия с помощью Qt

```
void serialize(QObject const& source, QJsonObject* json) {
  forEachProperty(source, [json](const QObject& source, const QMetaProperty& property) {
    const char* name = property.name();
    const QVariant& value = property.read(&source);
    (*json)[name] = QJsonValue::fromVariant(value);
  });
}
```

```
void deserialize(const QJsonObject& json, QObject* dest) {
  forEachProperty(*dest, [&json](QObject& dest, const QMetaProperty& property) {
    const char* name = property.name();
    property.write(&dest, json[name].toVariant());
  });
}
```

Рефлексия с помощью Qt

```
template <typename T, typename Functor>
void forEachProperty(T& object, Functor functor) {
  const QMetaObject* metaobject = object.metaObject();

  for (int i = 0; i < metaobject->propertyCount(); ++i) {
    const QMetaProperty& property = metaobject->property(i);

    if (property.enclosingMetaObject() != metaobject || !property.isWritable())
      continue;
    functor(object, property);
  }
}
```

Рефлексия с помощью Qt

Плюсы

- Широкая поддержка платформ и компиляторов.
- Работает уже сейчас!
- Один код для сериализации всех наследников QObject

Минусы

- Необходимо декорировать перечисления и свойства.
- Дополнительный шаг при сборке.
- Run-time only!
- Qt

Рефлексия с помощью Boost

- Boost.Fusion
- BOOST_FUSION_ADAPT_STRUCT
- BOOST_FUSION_DEFINE_STRUCT
- Fusion + MPL для доступа к полям.

Рефлексия с помощью Boost

```
class Cat {
  enum FurColor {/*...*/};
  enum Sex {/*...*/};
  std::string name;
  std::time_t birthday;
  std::bitset<8> furColor;
  Sex sex;
};

BOOST_FUSION_ADAPT_STRUCT(
  Cat,
  name,
  birthday,
  furColor,
  sex
)
```

```
enum FurColor {/*...*/};
enum Sex {/*...*/};

BOOST_FUSION_DEFINE_STRUCT_INLINE(
  Cat,
  (std::string, name)
  (std::time_t, birthday)
  (std::bitset<8>, furColor)
  (Sex, sex)
)
```

Рефлексия с помощью Boost

```
template<typename T, typename N>
void serialize(const T& object, Json::Value* json)
{
  char const* name = boost::fusion::extension::struct_member_name<T, N::value>::call();
  ToJson(boost::fusion::at<N>(object), &(*json)[name]);
}

template<typename T, typename N>
void deserialize(const Json::Value& json, T* object)
{
  char const* name = boost::fusion::extension::struct_member_name<T, N::value>::call();
  if (!json.isMember(name))
    return;

  FromJson(json[name], boost::fusion::at<N>(object));
}
```

Рефлексия с помощью Boost

Плюсы

- Широкая поддержка платформ и компиляторов.
- Работает уже сейчас!
- Compile-time

Минусы

- Необходимо декорировать перечисления и свойства.
- Шаблонная магия
- Boost

Рефлексия в C++11/14

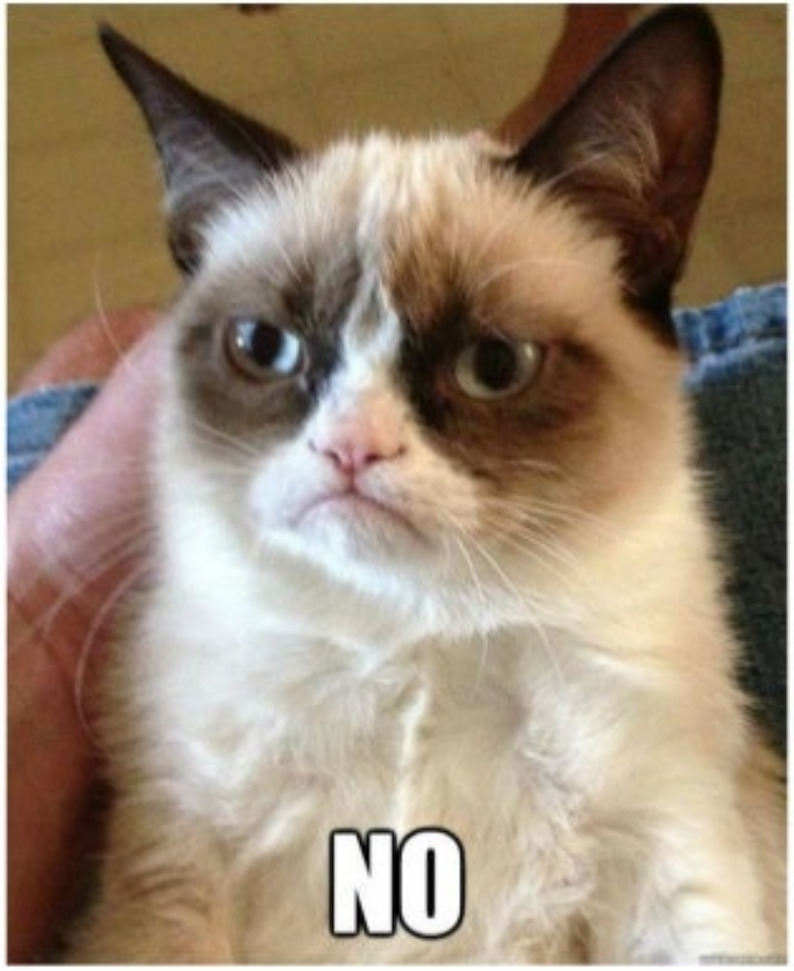
```cpp
#include <type_traits>

struct B {};
struct D : public B {};
std::is_pod<B>::value;                  // true
std::is_class<B>::value;                // true
std::is_abstract<B>::value;             // false
std::is_base_of<B, D>::value;           // true

int a[10][3];
std::extent<decltype(a)>::value;    // 10
std::extent<decltype(a), 1>::value; // 3
```

Рефлексия в C++17 ?

Рефлексия в C++17 ?



Рефлексия в C++??

- SG7 - Study group 7 - Reflection
- N4428 (std::class_traits, std::enum_traits)
- P0255 (`typedef<T,C>`, `typename<T,C>`, `typeid<T,C>`)
- P0194 + P0385 (reflexpr)

Рефлексия в C++?? / N4428

- std::enum_traits
- std::class_traits
- get<N> для доступа к элементам
- Нет возможности получить содержимое namespace

Рефлексия в C++?? / P0255

- typedef<T, C>
- typename<T, C>
- typeid<T, C>
- Дополнительные type traits (is_local, is_private, …)

Рефлексия в C++?? / P0255

```
template <typename T, ArithmeticMember MemberPtr>
void field_to_json(std::ostream& out, const T& v, MemberPtr member ) {
    out << std::quoted(typeid<MemberPtr>... ) << ':' << v.*member;
}

template <typename T, StringMember MemberPtr>
void field_to_json(std::ostream& out, const T& v, MemberPtr member ) {
    out << std::quoted(typeid<MemberPtr>...) << ':' << std::quoted( v.*member );
}

template <typename T, typename MemberPtr>
void field_to_json(std::ostream& out, const T& v, MemberPtr member ) {
    out <<  std::quoted(typeid<MemberPtr>...) << ':';
    class_to_json(out, v.*member);
}
```

Рефлексия в C++?? / P0255

```
template <typename T, typename _first, typename ..._rest>
void class_to_json_helper(std::ostream& out, const T& v, const _first& first, const _rest&
... rest) {
    our << '{';
    field_to_json( out, v, first );
    (((out << ','), field_to_json( out, v, rest )),...);
    return our << '}';
}

template <typename T>
void class_to_json(std::ostream& out, const T& v)
{
    class_to_json_helper( out, v, typedef< T, IsMemObjPointer >... );
};
```

Рефлексия в C++?? / P0194 + P0385

- operator reflexpr()
- Патч к clang
- Библиотека mirror

```
svn co http://llvm.org/svn/llvm-project/llvm/trunk llvm
git clone https://github.com/matus-chochlik/clang.git
cd clang
git checkout reflexpr
cd ../../..
mkdir build-clang
cd build-clang
cmake -G "Unix Makefiles" ../llvm -DCMAKE_BUILD_TYPE=Release
make
make check-clang
```

Рефлексия в C++?? / P0194 + P0385

```
// compose rapidjson document
rapidjson::Document doc;
to_rapidjson(doc, cat);

// write to stdout
rapidjson::OStreamWrapper osw(std::cout);
rapidjson::PrettyWriter<rapidjson::OStreamWrapper> writer(osw);
doc.Accept(writer);
```

Ссылки

Qt

- http://doc.qt.io/qt-5/metaobjects.html
- http://doc.qt.io/qt-5/properties.html
- http://doc.qt.io/qt-5/qobject.html#Q_FLAG

Boost

- http://boost.org/libs/fusion/
- http://jrruethe.github.io/blog/2015/05/21/boost-fusion-json-serializer/
- http://coliru.stacked-crooked.com/a/6cc59d82c2dbba24

C++

- http://en.cppreference.com/w/cpp/header/type_traits
- SG7: https://groups.google.com/a/isocpp.org/forum/#!forum/reflection
- http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4428.pdf
- http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0255r0.pdf

P0194

- http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0194r2.html
- http://clang.llvm.org/get_started.html
- https://github.com/matus-chochlik/mirror
- http://matus-chochlik.github.io/mirror/doc/html/

Вопросы?