Особенности создания XS-модулей на языке C++

Владимир Тимофеев

2013

Почему XS?

Потому что мы хотим:

- Увеличить производительность

Почему XS?

Потому что мы хотим:

- Увеличить производительность
- Уменьшить потребление памяти

Почему XS?

Потому что мы хотим:

- Увеличить производительность
- Уменьшить потребление памяти
- Использовать сторонние компоненты

Почему C++ ?

XS – кодогенератор для C. Но C++ ...

- Похож на C

Почему C++ ?

XS – кодогенератор для C. Но C++ ...

- Похож на C
- Сравним по скорости и потреблению памяти

Почему C++ ?

XS – кодогенератор для C. Но C++ ...

- Похож на C
- Сравним по скорости и потреблению памяти
- + Инкапсуляция

Почему C++ ?

XS – кодогенератор для C. Но C++ ...

- Похож на C
- Сравним по скорости и потреблению памяти
- + Инкапсуляция
- + STL, Boost, ...

Пример

☛Makefile.PL | CppStat.pm | CppStat.xs | typemap | perlobject.map

```
1  use ExtUtils::MakeMaker;
3  my $CC = 'g++';
5  WriteMakefile(
       NAME           => 'CppStat',
7      VERSION_FROM => 'CppStat.pm',
       TYPEMAPS       => ['perlobject.map'],
9      CC => $CC,
       LD => '$(CC)',
11 );
```

Пример

Makefile.PL | ☛CppStat.pm | CppStat.xs | typemap | perlobject.map

```
package CppStat;
use strict;
use warnings;

our $VERSION = '0.01';

require XSLoader;
XSLoader::load('CppStat', $VERSION);

1;
```

Пример

Makefile.PL | CppStat.pm | ☛CppStat.xs | typemap | perlobject.map

```
#ifdef __cplusplus
extern "C" {
#endif

#include "EXTERN.h"
#include "perl.h"

#ifdef __cplusplus
}
#endif

#include "XSUB.h"

#include <numeric>
#include <vector>
```

Пример

Makefile.PL | CppStat.pm | ☛CppStat.xs | typemap | perlobject.map

```
class CppStat {
public:
    CppStat() : values() { }
    ~CppStat() { }

    void add(double val) {
        values.push_back(val);
    }
    double avg() const {
        return std::accumulate(
                values.begin(), values.end(), 0.0
            ) / values.size();
    }
private:
    std::vector<double> values;
};
```

Пример

Makefile.PL | CppStat.pm | ☛CppStat.xs | typemap | perlobject.map

```
34 MODULE = CppStat             PACKAGE = CppStat

36 CppStat*
   CppStat::new();
38
   void
40 CppStat::DESTROY();

42 void
   CppStat::add(double val);
44
   double
46 CppStat::avg();
```

Пример

Makefile.PL	CppStat.pm	CppStat.xs	☛typemap	perlobject.map

```
TYPEMAP
CppStat *       O_OBJECT
```

Пример

Makefile.PL	CppStat.pm	CppStat.xs	typemap	☛perlobject.map

Можно поискать в интернете или сгенерировать с помощью модуля ExtUtils::Typemap::ObjectMap на CPAN

CC = g++

```
use ExtUtils::MakeMaker;

my $CC = 'g++';

WriteMakefile(
    NAME           => 'CppStat',
    VERSION_FROM => 'CppStat.pm',
    TYPEMAPS       => ['perlobject.map'],
    CC => $CC,
    LD => '$(CC)',
);
```

- Только GCC
- Все файлы как C++

-x c++

```
use ExtUtils::MakeMaker;
use Config;

my $CCFLAGS = join(' ',$Config{ccflags},'-x c++');

WriteMakefile(
    NAME           => 'CppStat',
    VERSION_FROM   => 'CppStat.pm',
    TYPEMAPS       => ['perlobject.map'],
    LIBS           => ['-lstdc++'],
    CCFLAGS        => $CCFLAGS,
);
```

- Компилятор тот же, что для perl
- Компилятор должен понимать опцию -x (GCC,Clang)
- Все файлы как C++

ExtUtils::CppGuess

```
use ExtUtils::MakeMaker;
use ExtUtils::CppGuess;

my $guess = ExtUtils::CppGuess->new;

WriteMakefile(
    NAME           => 'CppStat',
    VERSION_FROM   => 'CppStat.pm',
    TYPEMAPS       => ['perlobject.map'],
    CONFIGURE_REQUIRES => {
        'ExtUtils::CppGuess' => 0,
    },
    $guess->makemaker_options,
);
```

- Компилятор тот же, что для perl
- GCC,Clang,MSVC
- Все файлы как C++

Module::Build::WithXSpp

Использует ExtUtils::CppGuess, но...

- XS++
- Module::Build

Статистика

- Всего на CPAN: ~ 120
- CC = g++: ~ 50
- ExtUtils::CppGuess: 13
- Module::Build: ~ 15
- Module::Build::WithXSpp: 6

Конфликты имен

```
#ifdef __cplusplus
extern "C" {
#endif

#include "EXTERN.h"
#include "perl.h"

#ifdef __cplusplus
}
#endif

#include "XSUB.h"

#undef do_open
#undef do_close

#include <iostream>
```

Просто и неправильно

MyClass.cc

```
1 void MyClass::method(int arg) {
      std::vector<int> tmp;
3     if (arg < 0) croak("MyClass::method: arg should
          be positive");
      // ...
5 }
```

Просто и неправильно

MyClass.cc

```
1 void MyClass::method(int arg) {
      std::vector<int> tmp;
3     if (arg < 0) croak("MyClass::method: arg should
          be positive");
      // ...
5 }
```

- Утечка памяти

Просто и неправильно

MyClass.cc

```
1 void MyClass::method(int arg) {
      std::vector<int> tmp;
3     if (arg < 0) croak("MyClass::method: arg should
          be positive");
      // ...
5 }
```

- Утечка памяти
- Деструктор для tmp вызван не будет

Просто и неправильно

MyClass.cc

```
1 void MyClass::method(int arg) {
      std::vector<int> tmp;
3     if (arg < 0) croak("MyClass::method: arg should
          be positive");
      // ...
5 }
```

- Утечка памяти
- Деструктор для tmp вызван не будет
- Потому что croak реализован через longjmp

Переходим к C++ исключениям

MyClass.cc

```
#include <stdexcept>
void MyClass::method(int arg) {
    std::vector<int> tmp;
    if (arg < 0) throw std::runtime_error("MyClass
        ::method: arg should be positive");
    // ...
}
```

- Программа упадет
- Исключение надо еще перехватить

Переходим к C++ исключениям

MyModule.xs

```
void
MyClass::method(int arg);
```

Придется переписать и обернуть в try...catch.

Переходим к C++ исключениям

MyModule.xs

```
void
MyClass::method(int arg);
```

Придется переписать и обернуть в try...catch.

```
void
MyClass::method(int arg)
CODE:
    try {
        THIS->method(arg);
    }
    catch (const std::exception& e) {
        croak("Ops: %s", e.what());
    }
    catch (...) {
        croak("Ops: unknown c++ exception");
    }
```

Переходим к C++ исключениям

```
void
MyClass::method(int arg)
CODE:
    try {
        THIS->method(arg);
    }
    catch (const std::exception& e) {
        croak("Ops: %s", e.what());
    }
    catch (...) {
        croak("Ops: unknown c++ exception");
    }
```

Переходим к C++ исключениям

```
void
MyClass::method(int arg)
CODE:
    try {
        THIS->method(arg);
    }
    catch (const std::exception& e) {
        croak("Ops: %s", e.what());
    }
    catch (...) {
        croak("Ops: unknown c++ exception");
    }
```

- И получим опять утечку памяти (хоть и мелкую)

Переходим к C++ исключениям

```
void
MyClass::method(int arg)
CODE:
    try {
        THIS->method(arg);
    }
    catch (const std::exception& e) {
        croak("Ops: %s", e.what());
    }
    catch (...) {
        croak("Ops: unknown c++ exception");
    }
```

- И получим опять утечку памяти (хоть и мелкую)
- Именно в такой код наши вызовы оборачивает ExtUtils::XSpp

Вот так не течет

```
void
MyClass::method(int arg)
CODE:
    char errmsg[1024];
    errmsg[0] = '\0';
    try {
        THIS->method(arg);
    }
    catch (const std::exception& e) {
        snprintf(errmsg, 1024, "Ops: %s", e.what())
            ;
    }
    catch (...) {
        snprintf(errmsg, 1024, "Ops: unknown c++
            exception");
    }
    if (errmsg[0]) {
        croak("%s", errmsg);
    }
```

Хотелось бы писать Makefile.PL так

```
use ExtUtils::MakeMaker;

WriteMakefile(
    NAME           => 'CppStat',
    VERSION_FROM   => 'CppStat.pm',
    TYPEMAPS       => ['perlobject.map'],
    XS_LANG        => 'c++',
);
```

Может быть XS может выглядеть так?

```
  void
2 MyClass::method(int arg)
  CPPCATCH: std
```

Итого

- Писать на C++ можно

Итого

- Писать на C++ можно
- Есть поле для улучшений

Итого

- Писать на C++ можно
- Есть поле для улучшений
- Вопросы?

Итого

- Писать на C++ можно
- Есть поле для улучшений
- Вопросы?
- `mailto:vovkasm@gmail.com`
- Эта презентация доступна на `https://github.com/vovkasm/using-cpp-in-perlxs-slides`