Разработка макрорасширения для языка Kotlin. Обзор предметной области

Косолапов Семен Александрович

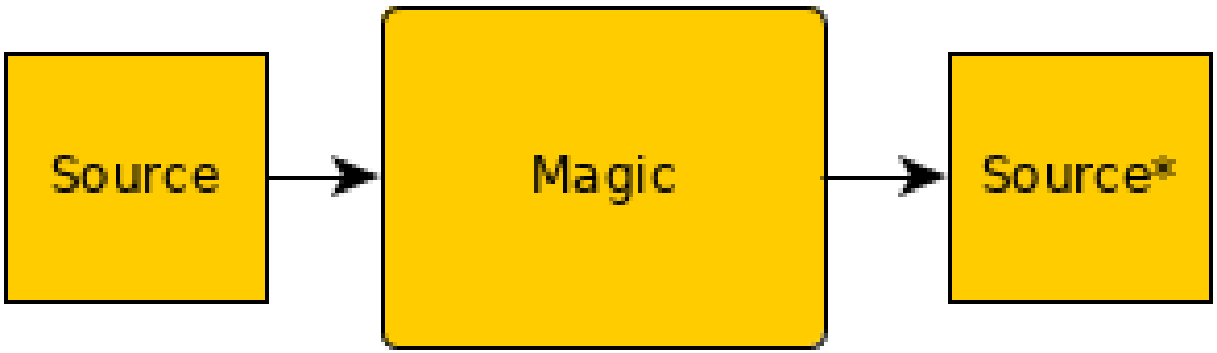
8 декабря 2016 г.

Что такое макросы и с чем их едят?

- Преобразователи исходного кода программ
- Используют специальный язык макрорасширений, элементы которого размещаются вместе с основной программой

Как это работает?

Макрорасширения являются средством метапрограммирования, то есть средством написания программ, порождающих другие программы



Для чего они нужны?

- Исключение дублирования исходного кода
- Шаблонизация кода
- Создание удобных синтаксических конструкций
- Создание DSL
- Генерация built-in данных
- ...

А почему для этого не использовать основной язык программирования?

- Это не всегда удобно программисту
- Может дольше выполняться (например, инлайнинг не всегда работает)
- А иногда это в принципе невозможно

Как появились макрорасширения?

- Изначально – в ассемблере для описания нескольких микроинструкций одной макроинструкцией
- К концу 50-х годов 20 века они приобрели тот вид, в котором используются сейчас
- Работают как простая текстовая подстановка

А для языков высокого уровня?

- Первым был язык Lisp (1958 г.)
- Первый чисто функциональный язык программирования
- Программы как данные: гомоиконность
- Очень простой синтаксис, или отсутствие его как такового
- Удобен для экспериментов
- Первая реализация макросов – в 1963 году

Тогда что с императивными языками?

- Наиболее известный – препроцессор языка C
- Просто текстовая подстановка
- Макросы очень широко используются в языке C

Но там же столько проблем...

- Да, и эти проблемы могут сильно осложнить жизнь программисту
- Никакой проверки типов и даже «целостности» параметров
- Вызов макроса синтаксически выглядит так же, как и вызов функции
- Нет никакой обратной связи от основного языка к языку макрорасширения

Больше скобочек!

- плохой вариант
  #define discriminant(a, b, c) b*b - 4*a*c
- хороший вариант
  #define discriminant(a, b, c) ((b)*(b) - 4*(a)*(c))

Есть и другой выход - гигиена

Переменные, объявленные внутри макроопределения, никогда не будут иметь конфликты с переменными, объявленными в области, где будет происходить раскрытие макро

- Впервые подход реализован для языка Scheme
- Можем обезопасить код от непреднамеренного изменения значений переменных
- Шаг к возможности определения рекурсивных макро
- Есть различные подходы к реализации

А если нужно внутри макроса обратиться к внешним объектам?

- Для этого используется квазицитирование
- При раскрытии макро должен знать, где находится, или хотя бы список объектов, к которым можно обратиться
- Варианты использования неочевидны

Macro-by-example

- А что, если определять семантику аргументов вызова макро?
- Pattern-matching
- Для каждого варианта – своя реализация
- Сопоставлять можно количество аргументов и тип элемента AST
- Регулярные выражения для повторений

Macro-by-example: пример на Scheme

```
(declare-syntax and
    [(and) true]
    [(and e) e]
    [(and e1 e2 ...) (if e1 (and e2 ...) false)])
```

Macro-by-example: пример на Rust

```
macro_rules! create_function {
    ($func_name:ident) => (
        fn $func_name() {
            println!("You called {:?}()",
                     stringify!($func_name))
        }
    )
}

create_function!(foo); // -> You called "foo"()
create_function!(bar); // -> You called "bar"()
```

Macro-by-example: пример на Rust

```
macro_rules! find_min {
    ($x:expr) => ($x);
    ($x:expr, $($y:expr),+) => (
        std::cmp::min($x, find_min!($($y),+))
    )
}

fn main() {
    println!("{}", find_min!(1u32));
    // -> 1
    println!("{}", find_min!(1u32 + 2 , 2u32));
    // -> 2
    println!("{}", find_min!(5u32, 2u32 * 3, 4u32));
    // -> 4
}
```

Что требуется?

Разработать препроцессор макроопределений, расширяющих язык Kotlin.

- На входе - программа на расширенном языке Kotlin с конструкциями, позволяющими определять и применять макро
- На выходе - программа на «чистом» языке Kotlin

Используемые принципы

- Macro-by-example
- Hygienic macros

Как это сделать?

- Использование кода компилятора языка Kotlin
- Основная проблема в том, что внутри AST макроса будет AST исходной программы
- ...Если что-то пойдёт не так, то придётся определять синтаксис с нуля

Что должно получиться?

```
macro define_hello_world {
    (function:ident) -> {
            fun $function() {
                println("Hello, world!")
            }
    }
}

fun someFun() {
    define_hello_world!(hw_in_fun)
    hw_in_fun() // -> "Hello, world!"
}
```

Спасибо за внимание!