Макросы в Скале

Евгений Бурмако
École Polytechnique Fédérale de Lausanne

О чем речь?

Доклад в двух предложениях:

- Зачем люди используют кодогенерацию?
- Что в этом плане можно сделать в Скале?

Копипаста

Копипаста это плохо

Абстракция

- Все мы умеем абстрагироваться от частностей и обобщать повторяющийся код
- Все мы знаем, что иногда это не работает

Примеры

- Геттеры-сеттеры-конструкторы
- Публикация и подписка на события
- Переносчики данных
- Доменно-специфические языки

Что делать?

- Мучительно писать горы однотипного кода
- Колбасить кодогенераторы
- Поменять язык программирования

Кодогенерация

- Плагины к билд тулу, генерирующие текст
- Текстовые макросы (C/C++)
- Синтаксические макросы (Lisp, Nemerle, Scala)
- Постпроцессоры байткода
- Рантаймовая кодогенерация

Макросы в Скале 2.10.x

Что умеют?

Превращать вызовы специально объявленных функций в произвольный код.

Было:

```
assert(2 + 2 == 4, "сообщение об ошибке")
```

Стало:

```
if (2 + 2 != 4) raise("сообщение об ошибке")
```

Как это работает?

- Макросы = функции, которые вызывает компилятор
- В эти функции передается код программы
- Эти функции возвращают код
- Сгенерированный код компилируется как написанный вручную
- Детали: http://scalamacros.org/documentation.html

Пример

```
assert(2 + 2 == 4, "сообщение об ошибке")

def assert(cond: Boolean, msg: Any) =
  macro impl

def impl(c: Ctx)(cond: c.Tree, msg: c.Tree) =
  If(Select(cond, TermName("$unary_bang")),
    Apply(Ident(TermName("raise")), List(msg)),
    Literal(Constant(())))
```

Сильные стороны

- Исполнение произвольного кода (макросы - это обычные скальные функции)
- Генерация произвольного кода (вызовы методов, объявления локальных классов/функций)
- Интроспекция (код компилируемой программы, список мемберов классов, загрузка аннотаций)
- Общение с тайпчекером (тайпчек, поиск имплиситов, проверка сабтайпинга)
- http://www.scala-lang.org/api/current/index.html#scala.reflect.macros.Context

Временные сложности

- Приходится работать на самом низком уровне (деревья для представления кода, внутренние структуры данных компилятора)
- API для общения макросов и компилятора пока что не отличается стабильностью
- Проблемы с инфраструктурой (IDE, отладка, инкрементальная компиляция, раздельная компиляция)

Кто уже использует макросы?

Slick

```
val coffees = Queryable[Coffee]
val result = coffees.map(c => (c.name, c.price))
```

- Вроде бы обычные filter и map на самом деле являются макросами
- Эти макросы ничего не делают с аргументами во время компиляции, а просто сохраняют эти функции на будущее (реификация)
- В рантайме сохраненные функции транслируются в SQL (или во что еще угодно)

Play

Скала 2.9:

```
case class Person(name: String)
implicit val personReads = (
  (__ \ 'name).reads[String]
)(Person)
```

Скала 2.10:

```
case class Person(name: String)
implicit val personReads = Json.reads[Person]
```

Скала 2.11:

```
case class Person(name: String)
```

SBT

Было:

```
myTask <<= (aTask, bTask) map { (a, b) =>
  f(a, b)
}
```

Стало:

```
myTask := f(a.value, b.value)
```

- Аналогично можно создавать свои нотации и в других случаях, перегружая разнообразные языковые конструкции (точку с запятой, условия, циклы, объявления и так далее)

Sqltyped

```
scala> val q = sql("select name from person")
scala> q() map (_ get name)
res0: List[String] = List("Martin", "Jason")

scala> q() map (_ get salary)
<console>:24: error: No such key salary
```

- Сиквел-запросы, валидируемые и типизируемые во время компиляции
- Да, макросы натурально берут и стучатся в базу данных в компайл-тайме

Declosurify

```
for (i <- 0 until 100000000 optimized) { ... }
```

- Циклы на функциях высшего порядка это удобно
- Но Скала пока что не умеет эффективно компилировать такие функции
- При помощи макроса можно взять красиво записанный цикл и преобразовать его в некрасивую, но быструю форму - бенчмарки сыты, глаза целы

Макросы в будущей Скале

Прогресс не стоит на месте

С момента кодофриза Скалы 2.10.0 мы работали над новыми фишечками не покладая рук

Что нас интересует?

- Как сделать написание макросов гуманным?
- Какие еще возможности дать нашим пользователям?
- Как скрестить возможности макросов и тайпчекера?

Квазицитаты

Было:

- `Apply(Ident(TermName("future")), List(body))`
- `case ClassDef(name, _, Template(_, _, _ :: defs) => ...`

Стало:

- `q"future { $body } "`
- `case q"class $name { ..${_ :: defs} }" => ...`

Тайп макросы

```
type H2Db(url: String) = macro impl

object Db extends H2Db("coffees”)
println(Db.Coffees.all)
```

- Уже как несколько месяцев макросы умеют генерировать публично видимые определения (классы, трейты, объекты)
- Очевидное применение – строго типизированная работа с данными, но есть и более веселые вещи

Макро аннотации

```
@Serializable
class Foo(val x: Int, val y: String)
```

- Совсем недавно я научился добавлять мемберы в произвольные классы в компилируемой программе.
- Открывающиеся возможности очень широки:
  - Кастомные кейс классы
  - Удобная сериализация
  - Более удобное реактивное программирование

Эзотерика

Макросы + тайпчекер:

- Имплисит макросы
- Нетипизированные макросы
- Макросы, влияющие на вывод типов
- Вычисления на типах при помощи макросов

Macro paradise

- Кодовое название форка Скалы, в котором тусуются штучки из этого раздела доклада
- Некоторые из фич в парадайзе уже заинтересовали наших продакшен ребят
- Заюзать парадайз очень легко – надо всего лишь поменять scalaOrganization в SBT билде

Заключение

Заключение

- Макросы = легковесные и портабельные плагины к компилятору Скалы
- Они уже с нами в продакшен релизе Скалы 2.10.0
- Такие популярные библиотеки как Slick, SBT и Play используют макросы
- Мы не стоим на месте и постоянно развиваем концепцию макросов в Скале

Вопросы и ответы

Евгений Бурмако
École Polytechnique Fédérale de Lausanne

eugene.burmako@epfl.ch
http://xeno-by.livejournal.com