α-Кеплер

Евгений Бурмако

EPFL, LAMP

14 января 2012

α-Кеплер

Всем привет! Меня зовут Евгений Бурмако, с прошлой осени я работаю в Scala Team и параллельно учусь в аспирантуре EPFL на кафедре Мартина Одерского.

Сегодня мы обсудим прогресс проекта “Кеплер”, в рамках которого реализуются **макросы** и **квазицитаты** - средства метапрограммирования времени компиляции для Скалы. В процессе общения мы реализуем одну штуку, о которой я мечтал со времен знакомства с Немерле. Также мы поговорим о том, какие незапланированные применения макросов нашлись в процессе работы над проектом.

В предыдущем выступлении я рассматривал макросы с теоретической точки зрения, а сегодня будет, в основном, практика. Поэтому перед тем, как продолжить чтение, может быть полезно просмотреть слайды прошлого рассказа.

План

Начнем издалека

Одним из интересных событий этой осени в мире Скалы стало обсуждение интерполяции строк.

Эта функциональность встречается во многих скриптовых языках (bash, Perl, Ruby) и предоставляет возможность встраивать в строки переменные из лексического окружения.

```
scala> val world = "world"
world: String = world

scala> println(s"hello ${world}!")
hello world!
```

Буква s, стоящая прямо перед строкой - не опечатка, а обозначение того, что строка интерполируется (в целях обратной совместимости доллары в обычных строках не будут интерпретироваться специальным образом).

Реализация интерполятора

Несмотря на свою простоту, интерполяция не поддается библиотечной реализации, так как она оперирует лексическим окружением, которое в явном виде недоступно.

В рамках смелого научного эксперимента нам придется открыть `doTypedApply` (функцию, которая типизирует применение методов) и вставить проверку на метод интерполяции. Внутри компилятора у нас есть полный доступ ко всей семантической информации о программе, чем мы и воспользуемся.

В тайпере возможно получить строковый литерал, который требуется проинтерполировать, и прямо на месте распарсить этот литерал. После этого, пройдясь по дереву контекстов, несложно составить словарь видимых переменных и превратить вызов метода `s` в обычную конкатенацию строк.

Если очень интересно, детали реализации можно посмотреть у меня в репозитории, а пока что пойдем дальше.

Использование интерполятора

Пока что все было довольно несложно. Мы встроились в тайпер и заменили вызов функции-маркера на вручную собранное дерево. Звучит страшнее, чем оно есть на самом деле. Через несколько слайдов мы будем заниматься тем же самым в режиме лайв =)

Единственный вопрос в том, как использовать наш патч к компилятору. Интерполяция выглядит более-менее серьезно, но вот трюки, специфичные для проекта, в апстрим явно не примут, а таскать за собой кастомную сборку компилятора неудобно по многим причинам.

Общепринятым решением в данной ситуации является написание плагина к компилятору (например, CPS в Скале реализован именно через плагин), а это очень близко к философии макросов. Мы почти на месте.

Что такое макросы?

Макросы - специального вида плагины к компилятору, которые автоматически загружаются из classpath и преобразуют заданные элементы программы (вызовы функций, ссылки на типы, объявления классов и методов).

По аналогии с рассмотренным выше примером макросы:

- Выполняются во время компиляции
- В отличие от традиционных плагинов прозрачно загружаются компилятором, не требуя оборачивания в `Plugin` и `Component`
- Работают с деревьями выражений, которые соответствуют коду компилируемой программы
- Имеют доступ к внутренним сервисам компилятора (рефлексия ранее скомпилированного кода, вывод и проверка типов, лексические окружения и так далее)

Какие бывают макросы?

Макро-функции получают на вход AST аргументов, раскрываются в AST и инлайнятся в точку вызова.

```
macro def printf(format: String, params: Any*) = ...
printf("Value = %d", 123 + 877)
```

Макро-типы представляют собой типы, члены которых генерируются во время компиляции:

```
macro class MySqlDb(connString: String) = ...
object MyDb extends MySqlDb("Database=Foo;")
```

Макро-аннотации выполняют пост-обработку объявлений методов и типов:

```
macro annotation Serializable(implicit ctx: Context) = ...
@Serializable case class Person(name: String)
```

Отклик

За несколько месяцев жизни проекта “Кеплер” нашлось немало практических задач, которые сильно упрощаются при наличии макросов.

Уже есть желающие использовать макросы для реализации языка запросов в O/RM, я слышал о планах использовать макро-аннотации в линзах, есть идеи насчет применения макро-типов для генерации бойлерплейта, необходимого для вычислений на типах. Буквально позавчера Мартин реализовал прототип оптимизатора, который использует макросы для ускорения `Range.foreach`.

Наша лаборатория выиграла грант Швейцарской комиссии по технологиям и инновациям на разработку усовершенствования Скалы, в основе которого лежат рефлексия и макросы.

Статус

На сегодняшний момент мы переосмыслили публичный интерфейс к компилятору и его структурам данных и воплотили в жизнь прототип компилятора, который реализует макро-функции. Мы довольны тем, как выглядит прототип, и будем дальше двигаться в этом направлении.

В активной разработке находятся квазицитаты - доменно-специфический язык для создания и декомпозиции абстрактных синтаксических деревьев. Скорее всего, они будут реализованы как обобщение строковой интерполяции, но здесь гораздо меньше ясности, чем с макросами.

Мы опубликовали сайт scalamacros.org, на котором собраны материалы по нашему проекту. Особый интерес, на наш взгляд, составляет раздел “Use Cases”, в котором рассматриваются области применения макросов.

План

Демонстрация

(В оффлайновой версии слайдов тут будет транскрипт нашей сессии кодинга)

План

Квазицитаты

В процессе демонстрации был задан вопрос. "Собирать деревья вручную неудобно. Но ведь парсер компилятора умеет преобразовывать код на Скале в деревья. Почему бы не создавать деревья, пользуясь их строковым представлением?"

Ответ на этот вопрос положительный - действительно, можно упростить создание AST с помощью языковой фичи, известной под названием "квазицитирование".

Я хотел обойти эту тему стороной потому, что дизайн квазицитат (в отличие от макросов) даже близко не завершен, поэтому финальная версия может радикально отличаться от текущего прототипа.

Однако предоставляемый квазицитатами функционал действительно очень важен, поэтому мы немного поговорим на эту тему на следующих слайдах. Более подробно теория разобрана в предыдущем выступлении.

Что умеют квазицитаты?

Квазицитаты превращают строки с кодом в эквивалентные синтаксические деревья:

```
val two = scala"2"
Literal(Constant(2))
```

Маленькие квазицитаты можно вставлять в большие при помощи сплайсинга:

```
val four = scala"$two + $two"
Apply(Select(two, newTermName("$plus")), List(two))
```

Квазицитаты можно использовать слева от стрелочки при паттерн-матчинге:

```
four match { case "2 + $x" => println(showRaw(x)) }
Literal(Constant(2))
```

Обобщенные квазицитаты

Механизм квазицитирования можно обобщить для встраивания совершенно чужеродных доменно-специфических языков.

Реализовав парсинг, сплайсинг (и, возможно, паттерн-матчинг) для DSL, программист обеспечивает тесную интеграцию языка с Scala. Для использования провайдера необходимо просто задать идентификатор провайдера перед строчкой с квазицитатой.

Провайдер квазицитаты выполняется во время компиляции, поэтому он может использовать семантическую информацию о программе (таблицы символов, сервис проверки типов, и т.д.), а также выполнять прекомпиляцию (аналогично тому, как мы это делали для регексов).

Интересные способы применения квазицитат для создания доменно-специфических языков рассматриваются в работе Why It's Nice to be Quoted: Quasiquoting for Haskell.

План

Резюме

- Механизм макросов предоставляет прозрачные точки расширения компилятора. С помощью макросов можно анализировать и оптимизировать код, генерировать бойлерплейт и многое другое.
- Макросы в Скале - это уже реальность. В ближайшем будущем мы выпустим бета-версию и представим пропоузал по добавлению макросов в 2.10.
- Интересное направление исследования - квазицитаты. Они не только упрощают работу с макросами, но и позволяют интегрировать в Скалу внешние DSL. Эта фича находится на более ранней стадии по сравнению с макросами, но мы постараемся успеть допилить ее к 2.10.
- По шагам нашей демонстрации уже сейчас можно поэкспериментировать со своими собственными макросами. Если что-то не получится - напишите мне на eugene.burmako@epfl.ch, разберемся вместе.

Что дальше?

Наш следующий шаг - стабилизация рефлексии (которая используется макросами для доступа к деревьям и компилятору) и выпуск бета-версии макросов.

Вместе с бета-версией мы опубликуем документы в рамках Scala Improvement Process, в которых предложим включить макросы и, возможно, квазицитаты в версию 2.10 (которая выйдет в первой половине года).

За новостями можно следить по следующим направлениям: новости на scalamacros.org (официальные объявления), посты в моем ЖЖ (дизайн-заметки и майлстоуны), мессаги в моем твиттере (с этим все понятно :)).

Наконец, за развитием проекта можно наблюдать на гитхабе: https://github.com/scalamacros/kepler.Основная разработка ведется в `topic/macros`, экспериментальные вещи (вроде квазицитат) имеют собственные бранчи.

Вопросы и ответы

eugene.burmako@epfl.ch