Инлайн оптимизация для компилятора Kotlin в JavaScript

Руководитель: Башоров З. А.
Выполнил: Цветков А. Д.

СПбАУ НОЦНТ РАН, 2014

Kotlin

Статически-типизированный

Объектно-ориентированный

Компилируется в Java-байткод и JavaScript

Разрабатывается в JetBrains

Inline-функции

Пример

```
# Тело функции подставляется в место вызова
```

```kotlin
// Kotlin
fun sum(a: Int, b: Int) {
  return a + b
}

val sum = sum(1, 2)
```

```
# Без подстановки
```

```javascript
// JavaScript
var sum = sum(1, 2)
```

Inline-функции

Пример

```
# Тело функции подставляется в место вызова
```

```kotlin
// Kotlin
inline fun sum(a: Int, b: Int) {
  return a + b
}

val sum = sum(1, 2)
```

```
# С подстановкой
```

```javascript
// JavaScript
var sum = 1 + 2
```

Мотивация

- Оптимизация производительности
- После подстановки возможны дополнительные оптимизации (dead code elimination, constant folding, etc..)
- Уменьшение числа генерируемых классов (JVM)
- Необходимо для работы будущей функциональности (type-dependent functions, non-local returns)

Цели

- Исследовать подходы к реализации inline функций
- Реализовать прототип inline функций для js-backend

Задачи

Изучить устройство Kotlin компилятора (js-backend)

Реализовать возможность писать JavaScript код в Kotlin коде

Исследовать работу inline подстановки в проектах GWT, Closure Compiler

Реализовать прототип inline подстановки для Kotlin

JavaScript-in-Kotlin

Мотивация

- Для инлайна функций нескольких модулей нужно уметь восстанавливать JavaScript AST из уже сгенерированного кода

- Бывает полезно пробросить новое API среды исполнения (браузера, Node.js)

- Бывает полезно иметь возможность написать хак, использующий JavaScript напрямую

JavaScript-in-Kotlin

Варианты реализации

В виде комментария (как в GWT)

- Нужно модифицировать frontend (комментарии не сохраняются в AST)

```
// Kotlin
fun sum(a, b): Int = /* jsCode
  return a + b
*/
```

JavaScript-in-Kotlin

Варианты реализации

В виде комментария (как в GWT)

- Нужно модифицировать frontend (комментарии не сохраняются в AST)

В виде псевдо-функции (intrinsic-функции)

+ Не нужно модифицировать frontend

+ Немного проще в реализации

```
// Kotlin
import js

fun sum(a, b): Int = jsCode("""
  return a + b
""")
```

JavaScript-in-Kotlin

Варианты реализации

В виде комментария (как в GWT)

- Нужно модифицировать frontend (комментарии не сохраняются в AST)

В виде псевдо-функции (intrinsic-функции)

+ Не нужно модифицировать frontend

+ Немного проще в реализации

Первый вариант добавляет зависимость frontend от целевой платформы, поэтому был выбран второй вариант

JavaScript-in-Kotlin

Пример 1

```
// Kotlin
class Connection {
  public native abstract fun send(m: String);
}

fun connect(host: String): Connection = jsCode("""
  return new WebSocket(host);
""")
```

JavaScript-in-Kotlin

Пример 2

```
// Kotlin
fun globalLog(host: String): Unit = jsCode("""
  window.onerror = function(msg, url, line) {
    $.post(host, {
      data: { error: msg + ' at ' + url }
    })
  }
""")

fun main() {
  globalLog("/remote/log")
}
```

JavaScript-in-Kotlin

Реазультаты

Реализована возможность писать JavaScript код в Kotlin коде

Написаны тесты

Inline функции

Задачи

Исследовать, как работают реализации в аналогичных проектах (в частности GWT, Closure Compiler)

Выбрать подход к реализации inline-функций

Реализовать прототип

Inline функции

Подходы к реализации inline-функций

Интеграция кода GWT

Интеграция кода Closure Compiler

Реализация прототипа “с нуля”

Inline функции

Подходы к реализации inline-функций

Интеграция кода GWT

+ Похожий AST и API

- Большое количество кода (> 2000 loc)

Inline функции

Подходы к реализации inline-функций

- Интеграция кода GWT
  - Был реализован прототип, использующий GWT
  - Инлайнер оказался недостаточно функциональным
  - Доведение “до ума” по количеству изменений сопоставимо с реализацией “с нуля”
  - Часть кода из прототипа GWT была переиспользована

Inline функции

Подходы к реализации inline-функций

Интеграция кода Closure Compiler

+ Очень продвинутый инлайнер

- Совсем другой AST и API, что затрудняет поддержку в будущем

Inline функции

Подходы к реализации inline-функций

- Интеграция кода Closure Compiler
  - Из-за отличий в AST и API прямая интеграция инлайнера CC затруднительна
  - Возможна конвертация AST (Kotlin → Closure Compiler → Kotlin), но это затрудняет поддержку

Inline функции

Подходы к реализации inline-функций

Реализация прототипа “с нуля”

  # Быстрее можно получить начальный результат

  # Сложность обеспечения работы во всех случаях

Из-за недостаточной функциональности инлайнера GWT и потенциальных сложностей в поддержке CC был выбран вариант реализации с нуля

Inline функции

Пример 1

```kotlin
// Kotlin
inline fun sum(a: Int, b: Int): Int {
  return a + b
}

fun box() {
  val sum = sum(1, 2)
}
```

Inline функции

Пример 1

```
// JavaScript
// ...
box: function () {
  {
    var result_inlined;
    var a = 1;
    var b = 2;
    result_inlined = a + b;
  }
  var sum = result_inlined;
}
// ...
```

Inline функции

Пример 2

```
// Kotlin
inline fun abs(a: Int): Int {
  if (a < 0) {
    return -a
  }

  return a
}

fun box() {
  val abs = abs(-1)
}
```

Inline функции

Пример 2

```
// JavaScript
// ...
box: function () {
  break_inlined: {
    var result_inlined;
    var a = -1;
    if (a < 0) {
      result_inlined = -a;
      break break_inlined;
    }
    result_inlined = a;
    break break_inlined;
  }
  var abs = result_inlined;
}
// ...
```

Inline функции

Результаты

- Исследованы инлайнеры GWT, Closure Compiler
- Реализован прототип
- Написаны тесты

Inline функции

Дальнейшая работа

Поддержка inline функций из разных модулей

Поддержка лямбда выражений с замыканием

Спасибо за внимание!

Ссылки

```
# kotlin.jetbrains.org

# github.com/typeinference/kotlin/tree/inline3

# github.com/typeinference/kotlin/tree/js-code
```