Интерактивная отладка программ на языке R в интегрированной среде разработки IntelliJ IDEA

Прошев Семён Александрович

научный руководитель: Тузова Е.А.

СПб АУ НОЦНТ РАН

11 июня 2016 г.

Введение

R

- интерпретируемый
- отличительная черта — векторизация
- встроенные возможности по генерации графиков на основе пользовательских данных
- область применения — машинное обучение, статистика

Актуальность

RStudio

- точки останова — статические

IntelliJ

- система плагинов
- широкие возможности работы с текстовыми файлами (vcs, поиск, ...)

Экосистема

- CLion, IntelliJ IDEA, PyCharm
- привычный процесс программирования

Цель и задачи

Реализовать интерактивную отладку программ на языке R в IDE IntelliJ IDEA

- Разработка визуального отладчика
  - анализ возможностей языка по отладке
  - разработка способа взаимодействия с интерпретатором
  - реализация отладчика
- Интеграция графического вывода
  - анализ возможных способов перехвата графического вывода
  - разработка способа поддержки графиков в IDE
  - интеграция выбранного решения

Подходы к отладке программ

- низкоуровневые команды, вводимые в консоли
- API, сообщающее отладочную информацию

Возможности языка R по отладке

Режим *Browser*

- переключает интерпретатор в пошаговое исполнение
- после каждого шага — результат текущей инструкции и местоположение следующей

Команды

- запуск текущей инструкции и переход к следующей — *n*
- список переменных в текущем окружении — *ls*
- тип каждой из них — *typeof*
- значение — имя переменной
- пометка функции как отлаживаемой — *debug*

Взаимодействие с интерпретатором

- запуск интерпретатора как отдельного процесса
- взаимодействие посредством потоков ввода-вывода
  - маркер конца ответа
  - определение типа сообщения

Требования со стороны платформы

- установка, отключение или удаление точки останова
- путь выполнения программы
- остановка процесса отладки и завершение исполнения программы

Требования со стороны платформы

XDebugProcess

- *stepOver* – исполнение текущего выражения и переход к следующему
- *stepInto* – вход внутрь функции и начало ее пошагового исполнения
- *stepOut* – окончание отладки функции
- *resume* – продолжение исполнения до следующей точки останова
- *runToPosition* – продолжение исполнения до указанного места

Отладка

- запуск и настройка интерпретатора
- посылка запускаемого кода в качестве отдельной функции и ее запуск
- отправка инструкций интерпретатору
- обновление отладочной информации
- проверка позиции на соответствие точке останова

Графический вывод

Встроенные способы генерации графиков на основе пользовательских данных

- вывод на экран
- сохранение на диск в одном из форматов: PDF, SVG, PNG, JPEG, TIFF и др.
- не сохраняют график до тех пор, пока не началась отрисовка нового графика

Интеграция графиков

- Возможность встраивать собственные устройства графического вывода
  - C/C++
  - Rinternals.h
  - GraphicsEngine.h
  - GraphicsDevice.h
- R — язык с автоматическим управлением памятью

Интеграция графиков

DevDesc хранит функции, обрабатывающие графические события

- отрисовка линии, круга, прямоугольника
- активация/деактивация устройства
- закрытие устройства

GEDevDesc

- создается на основе *DevDesc*
- используется интерпретатором
- хранит историю графических событий

Интеграция графиков

- создание фонового png-устройства
- закрытие фонового устройства после каждого ”рисования”
  - влечет за собой сохранение графика на диск
- создание нового фонового устройства при следующем ”рисовании”
  - требует копирования истории с ”главного” устройства
- плагин следит за изменениями в директории графиков и отображает их

Результаты

- динамические точки останова
- отображение типов и значений переменных
- отладка функций (в т.ч. вложенных и переопределенных)
- интерпретация выражений во время отладки
  - watch
  - evaluate expression
  - ”условные” точки останова
- отладка циклов
- поддержка графиков и пользовательского ввода

Спасибо за внимание!

- **Отладчик и отображение графиков**
  - *https://github.com/ktisha/TheRPlugin*
- **Устройство графического вывода**
  - *https://github.com/sproshev/TheRPlugin_Device*

Типы сообщений

- PLUS — интерпретатор ожидает продолжения ввода
- EMPTY — пустой ответ
- DEBUGGING_IN — начало отладки функции
- DEBUG_AT — информация о следующей инструкции
- START_TRACE_BRACE, START_TRACE_UNBRACE — вход в функцию, помеченную командой trace
- CONTINUE_TRACE — вход в ту же функцию, из которой только что вышли
- EXITING_FROM — окончание отладки функции
- RECURSIVE_EXITING_FROM — окончание отладок нескольких функций
- RESPONSE — непустой ответ

Отладка функции

```
> foo <- function(x) {
+     x + 2
+ }

> foo(c(1:5))
debugging in: foo(c(1:5))
debug at #1: {
    x + 2
}

> n
debug at #2: x + 2

> n
exiting from: foo(c(1:5))
[1] 3 4 5 6 7
```

Отладка однострочной функции

```
1 > foo <- function(x) x + 2
2
3 > foo(c(1:5))
4 debugging in: foo(c(1:5))
5 debug: x + 2
6
7 > n
8 exiting from: foo(c(1:5))
9 [1] 3 4 5 6 7
```

Отладка и трассировка однострочной функции

```
> foo(c(1:5))
debugging in: foo(c(1:5))
debug: {
    .doTrace(foo_enter(), "on entry")
    x + 2
}

> n
debug: .doTrace(foo_enter(), "on entry")

> n
Tracing foo(c(1:5)) on entry
[1] "foo"
debug: x + 2

> n
exiting from: foo(c(1:5))
[1] 3 4 5 6 7
```

Отладка функции высшего порядка

```
> sapply(c(1:2), foo)
debugging in: FUN(X[[i]], ...)
debug: x + 2

> n
exiting from: FUN(X[[i]], ...)
debugging in: FUN(X[[i]], ...)
debug: x + 2

> n
exiting from: FUN(X[[i]], ...)
[1] 3 4
```

Нюансы

- поведение интерпретатора зависит от окружения
- команда trace меняет код функции
- отладка циклов на верхнем уровне
- отладка однострочных функций и циклов

Исполнение выражения

```
1  SEXP createExpressionSexp(const std::string &str,
2                            ScopeProtector *protector) {
3    return createExpressionSexp(createSexp(str,protector),
4                                protector);
5  }
6
7  SEXP evaluateExpression(SEXP exprSexp,
8                          ScopeProtector *protector) {
9    SEXP result = Rf_eval(VECTOR_ELT(exprSexp, 0),
10                         R_GlobalEnv);
11
12   protector->add(result);
13
14   return result;
15 }
```

Исполнение выражения

```
1  SEXP createSexp(const std::string &str,
2                  ScopeProtector *protector) {
3    SEXP result = Rf_allocVector(STRSXP, 1);
4    protector->add(result);
5    SET_STRING_ELT(result, 0, Rf_mkChar(str.c_str()));
6    return result;
7  }
8
9  SEXP createExpressionSexp(SEXP strSexp,
10                           ScopeProtector *protector) {
11   ParseStatus status;
12   SEXP result = R_ParseVector(strSexp, 1, &status,
13                               R_NilValue);
14   protector->add(result);
15   return result;
16 }
```