Содержание

От издательства

При подготовке данной книги к печати, ее рукописи были переданы для обсуждения членам сообщества ru-scala (http://ru-scala.livejournal.com/), принявшим самое деятельное участие в ее улучшении.

Особую благодарность издательство выражает Руслану Шевченко, Виталию Моральяну, Андрею Легкому и Ивану Федорову. Спасибо вам, друзья! Вашу помощь переоценить невозможно!

Предисловие

Когда я встретил Кея Хорстманна (Cay Horstmann) несколько лет тому назад, он сказал, что необходимо написать хорошую вводную книгу, описывающую язык Scala. Как раз перед этим вышла моя собственная книга, поэтому я, разумеется, спросил его, что в ней не так. Он ответил, что книга замечательная, но слишком большая – его студентам просто не хватает терпения прочитать все восемьсот страниц книги «Programming in Scala». Мне не оставалось ничего иного, как признать его правоту. И он вознамерился исправить ситуацию, написав книгу «Scala для нетерпеливых».

Я очень рад, что его книга наконец вышла, потому что она полностью соответствует своему названию. Она представляет собой весьма практичное введение в язык программирования Scala, описывает, в частности, чем этот язык отличается от Java, как преодолевать некоторые типичные проблемы, возникающие при его изучении, и как писать хороший программный код на языке Scala.

Scala – чрезвычайно выразительный и гибкий язык программирования. Он позволяет разработчикам библиотек использовать весьма сложные, высокоуровневые абстракции, чтобы пользователи этих библиотек, в свою очередь, могли легко и просто выражать свои мысли. В зависимости от того, с каким кодом вы столкнетесь, он может казаться очень простым или очень сложным.

Год назад я попытался дать некоторые разъяснения, определив ряд уровней для языка Scala и его стандартной библиотеки. Всего было выделено по три уровня для прикладных программистов и для создателей библиотек. Начальные уровни были просты в изучении, и их было вполне достаточно, чтобы начать писать программы. Знания, получаемые на средних уровнях, позволяют писать более выразительные и более функциональные программы, а библиотеки более гибкие в использовании. Освоив высшие уровни, программисты становятся экспертами, способными решать специализированные задачи. В то время я писал:

Я надеюсь, что это поможет начинающим решить, в каком порядке изучать темы, а учителям и авторам книг подскажет, в каком порядке представлять материал.

Книга Кея стала первой, где эта идея была воплощена в жизнь. Каждая глава помечена значком, обозначающим ее уровень, который сообщает читателю, насколько простой или сложной она является и на кого ориентирована – на разработчиков библиотек или прикладных программистов.

Как можно догадаться, первые главы представляют собой быстрое введение в основные возможности языка Scala. Но книга не останавливается на этом. Она также охватывает множество концепций «среднего» уровня и, наконец, доходит до описания весьма сложных тем, которые обычно не рассматриваются во вводных книгах, таких как создание парсер-комбинаторов или использование ограниченных продолжений. Метки, обозначающие уровень, могут служить руководством при выборе глав для чтения. И Кею с успехом удалось просто и доходчиво рассказать даже о самых сложных понятиях.

Мне настолько понравилась идея книги «Scala для нетерпеливых», что я предложил Кею и его редактору Грегу Доенчу (Greg Doench) выложить первую часть книги в свободный доступ на вебсайте Typesafe[1]. Они любезно согласились с моим предложением, за что я очень благодарен им. Теперь любой желающий может быстро обратиться к самому лучшему, на мой взгляд, компактному введению в язык Scala.

Мартин Одерски (Martin Odersky)
Январь 2012

---

[1] http://typesafe.com/resources/free-books. – *Прим. перев.*

Вступление

Развитие языков Java и C++ существенно замедлилось, и программисты, стремящиеся использовать самые современные технологии, обратили свои взоры на другие языки. Scala – весьма привлекательный выбор. Я считаю, что это самый интересный вариант для программистов, стремящихся вырваться за рамки Java или C++. Scala имеет выразительный синтаксис, который выглядит весьма свежо после приевшихся шаблонов Java. Программы на этом языке выполняются под управлением виртуальной машины Java, что открывает доступ к огромному количеству библиотек и инструментов. Он поддерживает функциональный стиль программирования, не отказываясь при этом от объектно-ориентированного стиля, давая возможность осваивать новые парадигмы постепенно. Интерпретатор дает возможность быстро опробовать свои идеи, что превращает изучение Scala в весьма увлекательное занятие. Наконец, язык Scala является статически типизированным языком, что дает компилятору возможность находить ошибки, а вам не тратить время на их поиск в работающей программе.

Я написал эту книгу для *нетерпеливых* читателей, желающих приступить к программированию на языке Scala немедленно. Я полагаю, что вы знакомы с Java, C# или C++, и потому не буду утруждать себя объяснением, что такое переменные, циклы или классы. Я не буду терпеливо перечислять все особенности языка, я не буду читать лекции о превосходстве одной парадигмы над другой, и я не заставлю вас продираться сквозь длинные искусственные примеры. Вместо этого вы будете получать необходимую информацию небольшими порциями, чтобы ее можно было быстро прочитать и вернуться к ней при необходимости.

Scala – сложный язык, но вы сможете эффективно использовать его, даже не зная всех его тонкостей. Мартин Одерски (Martin Odersky), создатель языка Scala, определил уровни владения языком для прикладных программистов и разработчиков библиотек, как показано в табл. П.1.

Таблица П.1. Уровни владения языком Scala

Прикладные программисты	Разработчики библиотек	Общий уровень владения языком
Начальный A1		Начальный
Переходный A2	Простой L1	Переходный
Эксперт A3	Сложный L2	Сложный
	Эксперт L3	Эксперт

Каждая глава (а иногда и отдельные разделы) помечены специальным значком, обозначающим уровень владения языком, необходимым для ее чтения. Главы следуют по возрастанию уровня сложности A1, L1, A2, L2, A3, L3. Даже если вы не планируете создавать собственные библиотеки, знание инструментов Scala, которыми пользуются разработчики библиотек, поможет вам эффективнее использовать чужие библиотеки.

Надеюсь, вам понравится изучать язык Scala с помощью этой книги. Если вы обнаружите ошибки или у вас появятся предложения по улучшению, посетите сайт http://horstmann.com/scala и оставьте свой комментарий. На этом сайте вы также найдете ссылку на файл архива, содержащего все примеры программного кода из этой книги.

Я очень благодарен Дмитрию Кирсанову (Dmitry Kirsanov) и Алине Кирсановой (Alina Kirsanova), превратившим мою рукопись в формате XHTML в замечательную книгу, позволив мне сконцентрироваться на содержимом и не отвлекаться на оформление. Любой автор скажет, насколько это здорово!

Книгу рецензировали: Адриан Кумиски (Adrian Cumiskey), Майк Дэвис (Mike Davis), Роб Диккенс (Rob Dickens), Даниэль Собрал (Daniel Sobral), Крейг Татарин (Craig Tataryn), Дэвид Уоленд (David Walend) и Уильям Уилер (William Wheeler). Спасибо вам за ваши комментарии и предложения!

Наконец, как всегда, хочу выразить признательность моему редактору Грегу Доенчу (Greg Doench) за то, что подал идею написать эту книгу, и за его поддержку в процессе работы.

Кей Хорстманн (Cay Horstmann)
Сан-Франциско, 2012

Об авторе

Кей Хорстманн (Cay S. Horstmann) – основной автор книги «Core Java™, Volumes I & II, Eighth Edition» (Sun Microsystems Press, 2008)[1], а также десятков других книг для профессиональных программистов и студентов факультетов информатики. Он является профессором информатики университета в Сан-Хосе и обладателем звания Java Champion.

---

[1] Кей С. Хорстманн, Г. Корнелл. Java 2. Библиотека профессионала. Основы. – Т. 1. – Вильямс, 2008. – ISBN: 978-5-8459-1378-4; Кей С. Хорстманн, Г. Корнелл. Java 2. Библиотека профессионала. Тонкости программирования. – Т. 2. – Вильямс, 2008. – ISBN: 978-5-8459-1482-8. – *Прим. перев.*

Глава 1. Основы

Темы, рассматриваемые в этой главе A1

- 1.1. Интерпретатор Scala.
- 1.2. Объявление значений и переменных.
- 1.3. Часто используемые типы.
- 1.4. Арифметика и перегрузка операторов.
- 1.5. Вызов функций и методов.
- 1.6. Метод apply.
- 1.7. Scaladoc.
- Упражнения.

В этой главе вы узнаете, как использовать язык Scala в качестве мощного карманного калькулятора, выполняя арифметические операции над числами в интерактивном режиме. Попутно здесь будет представлено множество важных понятий и идиом языка Scala. Вы также узнаете, как просматривать документацию Scaladoc.

В этом введении рассматриваются следующие основные темы:

- использование интерпретатора Scala;
- определение переменных с помощью объявлений `var` и `val`;
- числовые типы;
- использование операторов и функций;
- навигация по документации Scaladoc.

1.1. Интерпретатор Scala

Чтобы приступить к работе с интерпретатором Scala, необходимо:

- установить язык Scala;
- добавить путь к каталогу `scala/bin` в переменную окружения `PATH`;
- открыть окно терминала в своей операционной системе;
- ввести команду `scala` и нажать клавишу **Enter**.

Совет. Не любите пользоваться командной оболочкой? Другие способы запуска интерпретатора описываются на странице http://horstmann.com/scala/install.

Вводите следующие команды, завершая ввод нажатием клавиши **Enter**. Каждый раз интерпретатор будет выводить ответ. Например, если ввести `8 * 5 + 2` (как показано ниже), интерпретатор выведет ответ `42`.

```
scala> 8 * 5 + 2
res0: Int = 42
```

Результату назначается имя `res0`. Его можно использовать в последующих вычислениях:

```
scala> 0.5 * res0
res1: Double = 21.0
scala> "Hello, " + res0
res2: java.lang.String = Hello, 42
```

Как видите, интерпретатор также отображает тип результата – `Int`, `Double` и `java.lang.String` в примерах выше.

Вы можете вызывать методы. При некоторых способах запуска есть возможность использовать клавишу **Tab** для автоматического дополнения имен методов. Попробуйте ввести `res2.to` и нажать клавишу **Tab**. Если интерпретатор предложит варианты выбора, такие как:

```
toCharArray toLowerCase toString toUpperCase
```

это означает, что функция автоматического дополнения работает. Введите `U` и снова нажмите клавишу табуляции. Теперь вы должны получить единственный вариант:

```
res2.toUpperCase
```

Нажмите клавишу **Enter**, и на экране появится ответ. (Если функция автоматического дополнения не работает, придется ввести имя метода вручную.)

Попробуйте также понажимать клавиши со стрелками ↑ и ↓. В большинстве реализаций вы увидите прежде выполнявшиеся команды и сможете редактировать их. С помощью клавиш ←, → и **Del** измените последнюю команду на

```
res2.toLowerCase
```

Как видите, интерпретатор Scala способен прочитать выражение, вычислить его, вывести результат и прочитать следующее выражение. Такой порядок действий называется *цикл чтения–вычисления–вывода* (read-eval-print loop, REPL).

Строго говоря, программа `scala` *не является* интерпретатором. За кулисами она быстро компилирует введенные вами команды в байт-код и выполняет его в виртуальной машине Java. По этой причине большинство программистов на Scala предпочитают называть этот цикл «the REPL».

> **Совет.** Цикл REPL – ваш друг и помощник. Немедленная обратная связь поощряет эксперименты, и вы будете чувствовать себя увереннее, имея возможность немедленно получать результаты.

При этом очень хорошо иметь постоянно открытое окно редактора, чтобы можно было копировать в него удачные фрагменты кода для использования в будущем. Кроме того, экспериментируя с более сложными примерами, их можно сначала компоновать в редакторе, а затем копировать в окно REPL.

1.2. Объявление значений и переменных

Вместо имен `res0`, `res1` и т. д. можно определить собственные имена:

```
scala> val answer = 8 * 5 + 2
answer: Int = 42
```

Их можно использовать в последующих выражениях:

```
scala> 0.5 * answer
res3: Double = 21.0
```

Значение, объявленное с помощью `val`, в действительности является константой – ее значение нельзя изменить:

```
scala> answer = 0
<console>:6: error: reassignment to val
```

Чтобы объявить переменную, значение которой может изменяться, следует использовать ключевое слово `var`:

```
var counter = 0
counter = 1 // OK, переменные могут изменяться
```

В языке Scala предпочтительнее использовать `val`, если в дальнейшем не предполагается изменять значение. Самое удивительное для программистов на Java или C++, что в большинстве программ не требуется много переменных `var`.

Обратите внимание на отсутствие необходимости явно объявлять тип значения или переменной. Тип автоматически определяется из типа инициализирующего выражения. (Объявление значения или переменной без инициализации является ошибкой.)

Однако при необходимости тип можно объявить явно. Например:

```
val greeting: String = null
val greeting: Any = "Hello"
```

> **Примечание.** В языке Scala тип переменной или функции всегда указывается после имени этой переменной или функции. Это упрощает чтение объявлений сложных типов.

Так как мне часто приходится переключаться между языками Scala и Java, я замечаю, что мои пальцы автоматически набирают такие Java-объявления, как `String greeting`, поэтому мне приходится исправлять их на `greeting: String`. Это немного раздражает, но когда я работаю со сложными программами на языке Scala, мне нравится, что не нужно расшифровывать объявления в стиле языка C.

> **Примечание.** Возможно, кто-то уже заметил отсутствие точек с запятой после объявлений переменных и инструкций присваивания. В языке Scala точки с запятой необходимы, только если в одной строке присутствует несколько инструкций.

В одном объявлении можно объявить сразу несколько значений или переменных:

```
val xmax, ymax = 100 // xmax и ymax получат значение 100
var greeting, message: String = null
  // обе переменные, greeting и message, - строки со значением null
```

1.3. Часто используемые типы

Вы уже видели некоторые типы данных из языка Scala, такие как `Int` и `Double`. Как и в языке Java, в Scala имеются семь числовых типов: `Byte`, `Char`, `Short`, `Int`, `Long`, `Float` и `Double`, – и один логический тип `Boolean`. Однако, в отличие от Java, все эти типы являются *классами*. В Scala нет никакой разницы между простыми типами и классами. Вы можете вызывать методы чисел, например:

```
1.toString() // Вернет строку "1"
```

или еще интереснее:

```
1.to(10)     // Вернет Range(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

(Класс `Range` будет рассматриваться в главе 13, а пока просто считайте его коллекцией чисел.)

В языке Scala нет необходимости использовать типы-обертки (wrapper types). Эту работу берет на себя компилятор, преобразуя простые типы в обертки и обратно. Например, создав массив чисел типа `Int`, в виртуальной машине вы получите массив `int[]`.

Как было показано в разделе 1.1 «Интерпретатор Scala», в случае со строками Scala опирается на класс `java.lang.String`. Однако расширяет этот класс более чем сотней дополнительных операций в классе `StringOps`. Например, метод `intersect` возвращает символы, общие для двух строк:

```
"Hello".intersect("World") // Вернет "lo"
```

В этом примере объект `"Hello"` типа `java.lang.String` неявно преобразуется в объект типа `StringOps`, и затем вызывается метод `intersect` класса `StringOps`.

Поэтому не забудьте заглянуть в описание класса `StringOps`, когда будете пользоваться документацией для Scala (см. раздел 1.7 «Scaladoc»).

Существуют также аналогичные классы `RichInt`, `RichDouble`, `RichChar` и т. д. Каждый из них имеет небольшой набор удобных методов для расширения на своих бедных родственников – `Int`, `Double` или `Char`. Метод `to`, представленный выше, в действительности является методом класса `RichInt`. В выражении

```
1.to(10)
```

значение 1 типа Int сначала преобразуется в объект типа RichInt, а затем вызывается метод to этого объекта.

Наконец, существуют классы BigInt и BigDecimal для вычислений с произвольной (но конечной) точностью. Они опираются на классы java.math.BigInteger и java.math.BigDecimal, но, как будет показано в следующем разделе, они намного удобнее, потому что допускают использование с обычными математическими операторами.

> **Примечание.** Для преобразований между числовыми типами в Scala используются методы, а не операция приведения типа. Например, 99.44.toInt вернет 99, а 99.toChar – 'c'. Разумеется, как и в языке Java, метод toString преобразует любой объект в строку.

Преобразовать строку цифр в число можно с помощью методов toInt и toDouble. Например, "99.44".toDouble вернет 99.44.

1.4. Арифметика и перегрузка операторов

Арифметические операторы в языке Scala действуют так же, как в Java или в C++:

```
val answer = 8 * 5 + 2
```

Операторы + - * / % выполняют свою обычную работу, как и поразрядные операторы & | ^ >> <<. Есть лишь один необычный аспект: эти операторы в действительности являются методами. Например,

```
a + b
```

это сокращенная форма записи

```
a.+(b)
```

Здесь + является именем метода. В языке Scala отсутствует глупое предубеждение против неалфавитно-цифровых символов в именах методов. Вы можете объявлять методы, содержащие почти любые символы в именах. Например, класс BigInt определяет метод с именем /%, который возвращает частное и остаток от деления.

В общем случае следующая форма записи

```
a method b
```

является сокращением от

```
a.method(b)
```

где `method` – это метод с двумя параметрами (один неявный и один – явный). Например, вместо

```
1.to(10)
```

можно записать

```
1 to 10
```

Используйте любую форму, которая будет проще для восприятия. Начинающие программисты на Scala по привычке придерживаются синтаксиса языка Java, и это хорошо. Разумеется, даже самые прожженные Java-программисты предпочтут использовать `a + b` вместо `a.+(b)`.

Между языком Scala, с одной стороны, и Java или C++ – с другой, существует одна весьма заметная разница. В языке Scala отсутствуют операторы `++` и `--`. Вместо них используются выражения `+=1` и `-=1`:

```
counter+=1 // Увеличит значение переменной counter, так как в Scala нет ++
```

Некоторые спрашивают, существуют ли какие-либо глубинные причины, объясняющие отсутствие оператора `++` в языке Scala. (Обратите внимание, что нельзя просто реализовать метод с именем `++`. Поскольку класс `Int` является неизменяемым, такой метод не сможет изменить целочисленное значение.) Разработчики языка Scala решили не вводить еще одно специальное правило, только чтобы сэкономить на нажатиях клавиш.

При работе с объектами типа `BigInt` и `BigDecimal` можно использовать обычные математические операторы:

```
val x: BigInt = 1234567890
x * x * x // Вернет 1881676371789154860897069000
```

Так намного лучше, чем в Java, где вы были бы вынуждены вызывать `x.multiply(x).multiply(x)`.

Примечание. В Java не поддерживается возможность перегрузки операторов, и разработчики Java утверждают, что это – благо, потому что препятствует появлению совершенно сумасшедших операторов, таких как `!@$&*`, которые могут сделать программу совершенно нечитаемой. Конечно, это глупо – программу точно так же можно сделать нечитаемой, выбирая сумасшедшие имена методов, такие как `qxywz`. Язык Scala позволяет определять операторы, полагаясь на ваше благоразумие при использовании этой возможности.

1.5. Вызов функций и методов

В дополнение к методам в языке Scala поддерживаются функции. В Scala имеются такие математические функции, как `min` или `pow`, пользоваться которыми намного проще, чем в Java, – для этого не требуется вызывать *статические методы* класса.

```
sqrt(2)     // Вернет 1.4142135623730951
pow(2, 4)   // Вернет 16.0
min(3, Pi) // Вернет 3.0
```

Математические функции определены в пакете `scala.math`. Их можно импортировать инструкцией

```
import scala.math._ // Символ _ в Scala - "групповой" символ, аналог * в Java
```

Примечание. При использовании пакета, имя которого начинается с префикса `scala.`, этот префикс можно опустить. Например, инструкция `import math._` эквивалентна инструкции `import scala.math._`, а вызов `math.sqrt(2)` эквивалентен вызову `scala.math.sqrt(2)`.

Инструкция `import` подробнее будет обсуждаться в главе 7. А пока просто используйте инструкцию `import packageName._`, когда вам потребуется импортировать какой-либо пакет.

В языке Scala отсутствуют статические методы, но в нем есть похожая особенность – *объекты-одиночки* (singleton objects), которые подробно будут рассматриваться в главе 6. Часто классы имеют *объекты-компаньоны* (companion object), чьи методы играют роль статических методов в Java. Например, объект-компаньон `BigInt` для класса `BigInt` имеет метод `probablePrime`, который генерирует случайное простое число с заданным количеством битов:

```
BigInt.probablePrime(100, scala.util.Random)
```

Попробуйте выполнить эту команду в REPL – вы получите число вида 10394479804912002754865402407l3. Обратите внимание, что вызов BigInt.probablePrime напоминает вызов статического метода в Java.

> **Примечание.** Здесь Random – это объект-одиночка, генератор случайных чисел, определенный в пакете scala.util. Это одна из ситуаций, когда объекты-одиночки оказываются предпочтительнее классов. В Java часто встречается ошибка, когда для каждого случайного числа создается новый объект java.util.Random.

Вызовы методов без аргументов в языке Scala часто записываются без использования круглых скобок. Например, в API класса StringOps имеется метод distinct без (), возвращающий уникальные символы в строке. Его можно вызвать как

```
"Hello".distinct
```

Скобки обычно отсутствуют у методов без параметров, которые не изменяют объект. Подробнее об этом будет рассказываться в главе 5.

1.6. Метод apply

В языке принято использовать синтаксис, напоминающий вызовы функций. Например, если s – это строка, тогда выражение s(i) вернет *i*-й символ строки. (В C++ та же самая операция выполняется как s[i]; в Java – как s.charAt(i).) Попробуйте выполнить в REPL следующую команду:

```
"Hello"(4) // Вернет 'o'
```

Ее можно считать перегруженной формой оператора (). Однако в действительности она реализована как метод с именем apply. Например, в описании класса StringOps можно найти метод

```
def apply(n: Int): Char
```

То есть выражение "Hello"(4) фактически является краткой формой записи

```
"Hello".apply(4)
```

Заглянув в описание объекта-компаньона `BigInt`, можно увидеть методы `apply`, позволяющие преобразовывать строки или числа в объекты `BigInt`. Например, вызов

```
BigInt("1234567890")
```

является краткой формой записи

```
BigInt.apply("1234567890")
```

и возвращает новый объект `BigInt`, при этом *нет необходимости использовать ключевое слово new*. Например:

```
BigInt("1234567890") * BigInt("112358111321")
```

Использование метода `apply` объекта-компаньона является в языке Scala типичной идиомой конструирования объектов. Например, вызов `Array(1, 4, 9, 16)` вернет массив, созданный методом `apply` объекта-компаньона `Array`.

1.7. Scaladoc

Для исследования Java API программисты на Java пользуются системой генерации документации Javadoc. В Scala есть аналогичный инструмент – Scaladoc (рис. 1.1).

Навигация по документации в Scaladoc немного сложнее, чем в Javadoc. Классы в языке Scala обычно имеют намного больше вспомогательных методов, чем Java-классы. Некоторые методы используют пока незнакомые вам возможности. Наконец, описания некоторых особенностей рассказывают о том, как они реализованы, а не как ими пользоваться. (Разработчики языка Scala продолжают работать над улучшением внешнего вида Scaladoc, поэтому в будущем этот инструмент, возможно, станет более доступным для начинающих программистов.)

Ниже приводится несколько советов, касающихся навигации в Scaladoc, для тех, кто приступает к изучению языка.

Существует возможность работать с электронной документацией на сайте www.scala-lang.org/api, но лучше загрузить копию на странице www.scala-lang.org/downloads#api и установить ее локально.

В отличие от Javadoc, где список классов приводится в алфавитном порядке, классы в Scaladoc отсортированы по именам пакетов.