Содержание

Предисловие

Конкурентное и параллельное программирование постепенно превращается из узкоспециализированной дисциплины, интересной только специалистам, занимающимся разработкой ядра операционной системы или высокопроизводительными вычислениями, в комплекс знаний, которыми должен обладать каждый профессиональный программист. По мере превращения параллельных и распределенных вычислений в норму большинство приложений будет создаваться конкурентными – для увеличения производительности или обработки асинхронных событий.

Но пока большинство разработчиков не готово к этой революции. Возможно, уже давно они изучали традиционную модель параллельных вычислений, основанную на потоках и блокировках, но эта модель не способна обеспечить высокую надежность и приемлемую производительность в приложениях с массовым параллелизмом. В действительности потоки и блокировки сложны в использовании, а пользоваться ими правильно еще сложнее. Чтобы добиться успеха, необходимо использовать абстракции конкурентного выполнения, находящиеся на более высоком уровне и допускающие объединение.

15 лет назад я работал над предшественником Scala – экспериментальным языком Funnel со встроенной семантикой конкуренции. Все понятия программирования были реализованы в этом языке как синтаксический сахар поверх функциональных сетей, объектно-ориентированного варианта исчисления соединений процессов (join calculus). Даже притом, что исчисление соединений процессов считается замечательной теорией, после нескольких экспериментов мы поняли, что проблема конкуренции более многогранна, из-за чего ее трудно выразить единственной формулировкой. Нет общего рецепта, решающего все проблемы конкуренции; правильное решение во многом зависит от конечной цели. Хотите реализовать асинхронные вычисления, запускаемые в ответ на события или при получении потоков значений? Или получить автономные объекты, изолированные сущности, взаимодействующие друг с другом посредством сообщений? Или определить транзакции поверх изменяемого хранилища? Или, может быть, главной целью организации параллельных вычислений является увеличение производительности? Для каждой из этих задач существует своя абстракция, решающая свои проблемы: отложенные вычисления, реактивные потоки данных, акторы, транзакционная память или коллекции с поддержкой параллельных операций.

Это привело нас к языку Scala и к данной книге. Из-за большого количества полезных абстракций конкуренции идея встроить их все в язык программирования не показалась нам захватывающей. Главной целью создания Scala было упрощение определения высокоуровневых абстракций в пользовательском коде и библиотеках. Благодаря этому любой сможет определять модули, реализующие разные аспекты конкурентного программирования. Все эти модули могут быть основаны на низкоуровневых примитивах, предоставляемых операционной системой. По прошествии времени можно уверенно сказать, что этот подход оправдал себя. Современный язык Scala имеет несколько мощных и элегантных библиотек для

конкурентного программирования. Эта книга познакомит вас с некоторыми из них, объяснит область применения каждой и представит прикладные шаблоны.

Трудно найти более авторитетного специалиста, чем автор этой книги. Александр Прокопец (Aleksandar Prokopec) участвовал в развитии нескольких наиболее популярных библиотек поддержки конкурентного и параллельного программирования в Scala. Он также предложил некоторые из особенно сложных структур данных и алгоритмов. В лице этой книги он создал простое и понятное руководство и одновременно надежный справочник для области, в которой он работает. Я уверен, что книгу «Конкурентное программирование на Scala» должен прочитать каждый, кто пишет конкурентные и параллельные программы на Scala. Я также предполагаю, что она заинтересует всех, кто просто желает узнать больше об этой интереснейшей и быстро развивающейся области компьютерных вычислений.

Мартин Одерски (*Martin Odersky*),
профессор федеральной
политехнической школы Лозанны (EPFL),
создатель Scala

Об авторе

Александр Прокопец (Aleksandar Prokopec) – исследователь в области конкурентного и распределенного программирования. Имеет степень доктора компьютерных наук, полученную в федеральной политехнической школе Лозанны (École Polytechnique Fédérale de Lausanne), Швейцария. Работал в Google и в настоящее время работает ведущим исследователем в Oracle Labs.

Став членом команды разработчиков Scala в EPFL, Александр активно участвовал в создании и развитии этого языка программирования и занимался разработкой абстракций конкуренции, параллельной обработки данных и конкурентных структур данных для Scala. Он создал библиотеку параллельных коллекций для Scala, предназначенную для высокоуровневого программирования алгоритмов параллельной обработки данных, и участвовал в рабочих группах по разработке таких конкурентных библиотек для Scala, как Futures/Promises и ScalaSTM. Александр – главный автор реактивной модели программирования для распределенных вычислений.

Благодарности

Прежде всего я хотел бы сказать спасибо моим рецензентам: Самире Ташарофи (Samira Tasharofi), Лукасу Рицу (Lukas Rytz), Доминику Грюнцу (Dominik Gruntz), Михаэлю Шинцу (Michel Schinz), Чжень Ли (Zhen Li) и Владимиру Костюкову (Vladimir Kostyukov), – за их ценные отзывы и комментарии. Также я хочу сказать спасибо редакторам из издательства Packt: Кевину Колако (Kevin Colaco), Шрути Катти (Sruthi Kutty), Капилу Хемнани (Kapil Hemnani), Вайбхаву Павару (Vaibhav Pawar) и Себастьяну Родригесу (Sebastian Rodrigues), – за помощь в работе над этой книгой. Мне было очень приятно работать с этими людьми.

Библиотеки поддержки конкуренции, описываемые в этой книге, не увидели бы свет без усилий множества людей. Многие прямо или косвенно способствовали их развитию. Эти люди – настоящие герои, и я хочу поблагодарить их за великолепную реализацию поддержки конкурентного программирования на Scala. Трудно было не упустить кого-то, но я старался, чтобы этого не случилось. Если кто-то почувствует себя обойденным вниманием, напишите мне, и я упомяну о вас в следующем издании этой книги.

Безусловно, я должен сказать слова благодарности в адрес Мартина Одерски (Martin Odersky) за создание языка программирования Scala, послужившего основой для библиотек конкурентного программирования, описанных в этой книге. Отдельное спасибо ему, всем членам команды разработки Scala из EPFL и всем разработчикам из Typesafe, приложившим максимум усилий, чтобы сделать Scala одним лучших языков программирования общего назначения.

Большинство библиотек конкуренции для Scala так или иначе опирается на работу Дуга Ли (Doug Lea). Его библиотека Fork/Join легла в основу реализации акторов Akka, Scala Parallel Collections, библиотеки Futures/Promises, а многие конкурентные структуры данных в JDK, описываемые в этой книге, были разработаны им самим. Еще больше библиотеки конкуренции для Scala было создано с использованием его советов.

Библиотека Futures/Promises для Scala первоначально была спроектирована Филиппом Халлером (Philipp Haller), Хизером Миллером (Heather Miller), Воджином Джовановичем (Vojin Jovanović) и мною из EPFL, Виктором Клангом (Viktor Klang) и Ролендом Куном (Roland Kuhn) из команды Akka, и Мариусом Эриксеном (Marius Eriksen) из Twitter, при содействии Хавока Пеннингтона (Havoc Pennington), Рича Догерти (Rich Dougherty), Джейсона Заугга (Jason Zaugg), Дуга Ли (Doug Lea) и многих других.

Хотя я был основным автором библиотеки Scala Parallel Collections, вклад в ее развитие внесло много разных людей, в том числе: Фил Багвелл (Phil Bagwell), Сартин Одерски (Martin Odersky), Тиарк Ромпф (Tiark Rompf), Дуг Ли (Doug Lea) и Натан Бронсон (Nathan Bronson). Позже Дмитрий Петрашко (Dmitry Petrashko) и я приступили к работе над улучшенной версией библиотеки параллельных и стандартных операций с коллекциями, оптимизированных с использованием Scala Macros. Юджин Бурмако (Eugene Burmako) и Денис Шабалин (Denys Shabalin) – одни из основных разработчиков проекта Scala Macros.

Работу над проектом Rx начинали Эрик Мейер (Erik Meijer), Уэс Дайер (Wes Dyer) и другие члены команды. Первоначально реализованный для .NET фреймворк Rx позднее был перенесен на многие другие языки, включая Java, Scala, Groovy, JavaScript и PHP, и получил широкое распространение благодаря вкладу, который внесли Бен Кристенсен (Ben Christensen), Сэмюэль Грюттер (Samuel Grütter), Шиксьенг Жу (Shixiong Zhu), Донна Малайери (Donna Malayeri) и многие другие.

Натан Бронсон (Nathan Bronson) – один из основных разработчиков проекта ScalaSTM, реализация по умолчанию которого основана на проекте CCSTM Натана. Программный интерфейс ScalaSTM был спроектирован группой экспертов, в состав которой вошли: Натан Бронсон (Nathan Bronson), Джонас Бонер (Jonas Bonér), Гай Корланд (Guy Korland), Кришна Шанкар (Krishna Sankar), Даниель Спивак (Daniel Spiewak) и Питер Вентьер (Peter Veentjer).

Первая версия библиотеки акторов для Scala была разработана Филиппом Халлером (Philipp Haller), черпавшим вдохновение из модели акторов в Erlang. Эта версия библиотеки побудила Джонаса Бонера (Jonas Bonér) начать разработку фреймворка акторов Akka. Со временем в развитии проекта Akka приняли участие многие специалисты, и в их числе: Виктор Кланг (Viktor Klang), Хенрик Энгстрём (Henrik Engström), Питер Влагтер (Peter Vlugter), Роланд Кун (Roland Kuhn), Патрик Нордуолл (Patrik Nordwall), Бъерн Антонссон (Björn Antonsson), Рич Догерти (Rich Dougherty), Йоханс Рудольф (Johannes Rudolph), Матиас Дениц (Mathias Doenitz), Филипп Халлер (Philipp Haller) и многие другие.

Наконец, я хочу выразить благодарность всему сообществу Scala за их вклад и за то, что сделали Scala замечательным языком программирования.

О технических рецензентах

Викаш Шарма (Vikash Sharma) – программист и активный сторонник программного обеспечения с открытым исходным кодом, живущий в Индии. Стремится к простоте, и это помогает ему писать ясный и простой в сопровождении код. Создал видеокурс по языку Scala. Работает младшим консультантом в Infosys, а также как разработчик на Scala.

Невозможно выразить благодарность, которую я испытываю к моей семье за ее поддержку, маме, папе и брату. Я очень ценю, что вы всегда оказывались рядом, когда ваша поддержка была нужнее всего. Особое спасибо Виджаю Атикешвану (Vijay Athikesavan), что передал мне свои знания и научил программированию.

Доминик Грюнц (Dominik Gruntz) – получил степень доктора в Высшей технической школе в Цюрихе (ETH Zürich) и работает профессором на факультете информатики в Университете прикладных наук в северо-западной Швейцарии (FHNW) с 2000 года. Кроме собственных исследований ведет курс о конкурентном программировании. Всего несколько лет тому назад цель курса состояла в том, чтобы убедить студентов, что разработка правильно работающих конкурентных программ – слишком сложная наука для простых смертных (и Доминику регулярно удавалось достичь этой цели). Но ситуация изменилась с появлением в Java и Scala высокоуровневых фреймворков поддержки конкурентных операций, и эта книга – «Конкурентное программирование на Scala» – является отличным источником знаний для всех программистов, желающих узнать, как писать правильные, читаемые и эффективные конкурентные программы. Эта книга – идеальное пособие для курса о конкурентном программировании.

Спасибо за возможность поддержать этот проект в роли технического рецензента.

Чжень Ли (Zhen Li) – влюбилась в компьютеры еще в начальной школе, когда впервые познакомилась с языком Logo. После получения квалификации «инженер-программист» в университете Фудань, в Шанхае (Китай) и закончив обучение на факультете информатики в университетском колледже Дублина (Ирландия), поступила в аспирантуру университета штата Джорджия, США. Для своих исследований выбрала психологию поведения программистов во время обучения и, в частности, их способы представления конкурентных программ. Основываясь на результатах исследований, она старается разработать эффективные приемы программирования и парадигмы обучения, способные помочь программистам проще понять суть конкурентных программ.

Чжень Ли имеет практический опыт преподавания студентам старших курсов по различным темам, связанным с информатикой, включая системное и сетевое программирование, моделирование и имитацию, а также взаимодействия человек–машина. Ее главный вклад в преподавание программирования компьютеров заключался в написании учебных планов и проведении курсов с различными языками программирования и способами организации конкурентных вычисле-

ний, которые побуждают учащихся активно постигать философию проектирования программного обеспечения и изучать все стороны конкурентного программирования.

Чжень Ли также имеет большой опыт в области промышленных инноваций. За последние 10 лет она успела поработать во многих IT-компаниях, включая Oracle, Microsoft и Google, где участвовала в разработке передовых продуктов, платформ и инфраструктур для предприятий, и облачных бизнес-технологий. Чжень Ли увлечена программированием и обучением. Вы можете связаться с ней по адресу: janeli@uga.edu.

Лукас Риц (Lukas Rytz) – специалист по компиляторам, работает в команде Scala в Typesafe. Получил степень доктора в EPFL в 2013 году и был рекомендован Мартином Одерски (Martin Odersky), изобретателем языка Scala.

Михаэль Шинц (Michel Schinz) – лектор в EPFL.

Самира Ташарофи (Samira Tasharofi) – получили степень доктора в области программной инженерии в Иллинойском университете в Урбана-Шампейне, США. Проводила исследования в разных областях, таких как тестирование конкурентных программ и, в частности, программ, основанных на использовании акторов, шаблонов параллельного программирования и верификации компонентных систем.

Самира рецензировала такие книги, как «Actors in Scala», «Parallel Programming with Microsoft .NET: Design Patterns for Decomposition and Coordination on Multicore Architectures (Patterns and Practices)» и «Parallel Programming with Microsoft Visual C++: Design Patterns for Decomposition and Coordination on Multicore Architectures (Patterns and Practices)». Рецензировала также научно-исследовательские работы на конференциях по программированию, включая ASE, AGERE, SPLASH, FSE и FSEN. Была членом подготовительного комитета 4-го международного семинара по программированию с использованием акторов, агентов и децентрализованных средств управления (AGERE 2014) и 6-й международной конференции по основам программной инженерии (FSEN 2015).

Хочу сказать спасибо моим мужу и маме за их неиссякаемую любовь и поддержку.

Посвящается Саше (Sasha).
Она единственный, пожалуй, доктор физической химии, прочитавший эту книгу.

Вступление

Конкуренция повсюду. С появлением на рынке многоядерных процессоров на разработчиков обрушился лавинообразный спрос на конкурентные программы. Конкурентное программирование, которое прежде использовалось для выражения асинхронных операций в программах и компьютерных системах и считалось академической дисциплиной, в настоящее время превратилось в распространенную методологию. Как результат с невероятной скоростью стали развиваться перспективные фреймворки и библиотеки для поддержки конкурентного программирования. В последние годы конкурентное программирование переживает свой ренессанс.

С ростом уровня абстракций конкуренции в современных языках и фреймворках становится все важнее знать, как и когда их использовать. Хорошее понимание особенностей работы классических примитивов конкуренции и синхронизации, таких как потоки выполнения, блокировки и мониторы, больше не является обязательным условием. Высокоуровневые фреймворки, решающие многие традиционные проблемы конкуренции и ориентированные на решение конкретных задач, постепенно завоевывают мир конкурентного программирования.

Эта книга описывает высокоуровневые приемы конкурентного программирования на языке Scala. В ней представлены подробные объяснения разных тем, связанных с конкуренцией, и охватывается базовая теория конкурентного программирования. Одновременно в ней описываются современные фреймворки поддержки конкурентного программирования, их семантика и приемы использования. Цель книги – познакомить вас с важнейшими абстракциями конкурентного программирования и в то же время показать, как они работают в реальном коде.

Мы верим, что, прочитав эту книгу, вы получите прочное понимание теории конкурентного программирования и приобретете полезные практические навыки, необходимые для создания правильных и эффективных конкурентных программ. Эти навыки станут первым шагом на пути к превращению вас в эксперта по конкурентному программированию.

Мы надеемся, что чтение этой книги доставит вам столько уже удовольствия, сколько получили мы, когда работали над ней.

О чем рассказывается в книге

Эта книга организована как последовательность глав, описывающих разные аспекты конкурентного программирования. Книга охватывает основные программные интерфейсы (API) конкурентного программирования в библиотеке времени выполнения Scala, знакомит с примитивами конкуренции и дает обширный обзор высокоуровневых абстракций.

Глава 1 «*Введение*» разъясняет необходимость конкурентного программирования и дает некоторое философское обоснование. Также она охватывает основы языка программирования Scala, необходимые для понимания остальной части книги.

Глава 2 «*Конкуренция в JVM и модель памяти в Java*» рассказывает об основах конкурентного программирования. В этой главе вы узнаете, как работать с потоками выполнения и защищать доступ к общей памяти, а также познакомитесь с моделью памяти в Java.

Глава 3 «*Традиционные строительные блоки конкурентных программ*» представляет классические инструменты реализации конкуренции, такие как пулы потоков, атомарные переменные и коллекции с поддержкой параллельного доступа, и демонстрирует примеры их использования с учетом особенностей языка Scala. Основное внимание в этой книге уделяется современным высокоуровневым фреймворкам конкурентного программирования. Поэтому данная глава, представляющая обзор традиционных приемов конкурентного программирования, не погружается слишком глубоко в их детали.

Глава 4 «*Асинхронное программирование с объектами Future и Promise*» – это первая глава, посвященная мезанизмам поддержки конкуренции в Scala. Эта глава знакомит с программным интерфейсом объектов `Future` и `Promise` и показывает, как правильно пользоваться ими для реализации асинхронных программ.

Глава 5 «*Параллельные коллекции данных*» описывает множество параллельных коллекций в Scala. В этой главе вы узнаете, как распараллелить операции с коллекцией, когда это возможно, и как оценить прирост производительности от такого распараллеливания.

Глава 6 «*Конкурентное программирование с Reactive Extensions*» научит приемам событийного и асинхронного программирования с использованием фреймворка Reactive Extensions. Здесь вы увидите взаимосвязь операций с потоками событий и коллекциями, узнаете, как передавать события между потоками выполнения и как проектировать реактивные пользовательские интерфейсы с использованием потоков событий.

Глава 7 «*Программная транзакционная память*» знакомит с библиотекой ScalaSTM для транзакционного программирования, реализующей безопасную и понятную модель общей памяти. В этой главе вы узнаете, как защитить доступ к общим данным с помощью масштабируемых транзакций и в то же время снизить риск взаимоблокировки и состояния гонки.

Глава 8 «*Акторы*» описывает модель программирования с акторами, реализованную в фреймворке Akka. В этой главе вы узнаете, как создавать распределенные приложения, выполняющиеся на нескольких компьютерах, которые опираются на механизм обмена сообщениями.

Глава 9 «*Конкуренция на практике*» сравнивает разные библиотеки поддержки конкуренции, представленные в предыдущих главах. В этой главе вы узнаете, как выбрать правильную абстракцию конкуренции для решения той или иной задачи и как комбинировать разные абстракции при проектировании больших конкурентных приложений.

Глава 10 «*Реакторы*» представляет модель программирования «Реактор», целью которой является усовершенствование структуры конкурентных и распределенных программ. Эта новая модель позволяет выделить шаблоны конкурентного и распределенного программирования в модульные компоненты, называемые протоколами.

Мы рекомендуем читать главы по порядку, сверху вниз, но в целом в этом нет необходимости. Если вы хорошо знакомы с темами, обсуждаемыми в главе 2 «Кон-

куренция в JVM и модель памяти в Java», вы можете пропустить ее и сразу перейти к другим главам. Из всех глав на содержимое предыдущих глав опираются только глава 9 «Конкуренция на практике», где дается практический обзор предыдущих тем, и глава 10 «Реакторы», при чтении которой полезно иметь представление о том, как действуют акторы и потоки событий.

Что потребуется для работы с книгой

В этом разделе описываются некоторые требования, которые должны быть выполнены, чтобы вы смогли читать и понимать эту книгу. Мы расскажем, как установить библиотеку Java Development Kit, необходимую для запуска программ на Scala, и покажем, как использовать Simple Build Tool для запуска разных примеров.

Мы не требуем наличия интегрированной среды разработки. Вам выбирать, какие программы использовать для написания кода – Vim, Emacs, Sublime Text, Eclipse, IntelliJ IDEA, Notepad++ или какие-то другие.

Установка JDK

Программы на Scala не компилируются непосредственно в машинный код, поэтому не могут запускаться как выполняемые файлы на разных аппаратных платформах. Компилятор Scala производит промежуточный код в особом формате, который называют байт-кодом Java. Чтобы вы могли запустить этот промежуточный код, на вашем компьютере должна быть установлена виртуальная машина Java. В этом разделе мы расскажем, как загрузить и установить библиотеку Java Development Kit, включающую виртуальную машину Java и другие полезные инструменты.

Существует несколько разных реализаций JDK от разных производителей. Мы рекомендуем использовать дистрибутив Oracle JDK. Чтобы загрузить и установить Java Development Kit, выполните следующие шаги:

1. Откройте в веб-браузере следующую страницу: www.oracle.com/technetwork/java/javase/downloads/index.html.
2. Если эта страница не открывается, перейдите на главную страницу какой-нибудь поисковой системы и введите слова: **JDK Download** (JDK скачать).
3. Найдите в результатах ссылку для загрузки Java SE на сайте Oracle, загрузите соответствующую версию JDK 7 для своей операционной системы: Windows, Linux или Mac OS X; 32- или 64-разрядную.
4. Если вы пользуетесь операционной системой Windows, просто запустите программу установки. В Mac OS X откройте dmg-архив, чтобы установить JDK. Наконец, в Linux распакуйте архив в каталог по своему выбору, например XYZ, и добавьте подкаталог bin в переменную PATH:

   ```
   export PATH=XYZ/bin:$PATH
   ```

5. Теперь должна появиться возможность запустить в терминале команды `java` и `javac`. Введите команду `javac` и убедитесь, что она доступна (далее в книге вам не придется запускать ее непосредственно, но данный шаг помогает убедиться, что она доступна).

Может так получиться, что в вашей системе уже имеется установленная версия JDK. Чтобы проверить это, просто попробуйте выполнить команду `javac`, как в последнем шаге в описании выше.

Установка и использование SBT

Simple Build Tool (SBT) – это инструмент командной строки, предназначенный для сборки проектов на Scala. Его задача – компиляция кода на Scala, управление зависимостями, непрерывная компиляция и тестирование, развертывание и многое другое. На протяжении всей книги мы будем использовать SBT для управления зависимостями наших проектов и запуска примеров кода.

Чтобы установить SBT, выполните следующие инструкции:

1. Перейдите на страницу http://www.scala-sbt.org/.
2. Загрузите дистрибутив для своей платформы. Если вы используете Windows, загрузите файл `msi`. Если вы пользуетесь Linux или OS X, загрузите архив `zip` или `tgz`.
3. Установите SBT. В Windows просто запустите программу установки. В Linux или OS X распакуйте содержимое архива в домашний каталог.

Теперь SBT готов к использованию. Выполните следующие шаги, чтобы создать новый проект SBT:

1. В Windows запустите программу Command Prompt (Командная строка), а в Linux или OS X откройте окно терминала.
2. Создайте пустой каталог с именем `scala-concurrency-examples`:

   ```
   $ mkdir scala-concurrency-examples
   ```

3. Перейдите в каталог `scala-concurrency-examples`:

   ```
   $ cd scala-concurrency-examples
   ```

4. Создайте каталог для наших примеров:

   ```
   $ mkdir src/main/scala/org/learningconcurrency/
   ```

5. Теперь откройте редактор и создайте в нем файл с именем `build.sbt`. Этот файл определяет различные свойства проекта. Создайте его в корневом каталоге проекта (`scala-concurrency-examples`). Добавьте в файл следующие строки (имейте в виду, что пустые строки являются обязательными):

   ```
   name := "concurrency-examples"

   version := "1.0"

   scalaVersion := "2.11.1"
   ```

6. Наконец, вернитесь в окно терминала и запустите SBT в корневом каталоге проекта:

   ```
   $ sbt
   ```

7. SBT запустит интерактивную оболочку, которую можно использовать для ввода различных команд сборки.

Теперь вы можете начинать писать программы на Scala. Откройте редактор и создайте в каталоге `src/main/scala/org/learningconcurrency` файл с именем `HelloWorld.scala`. Добавьте в него следующие строки:

```
package org.learningconcurrency

object HelloWorld extends App {
  println("Hello, world!")
}
```

Теперь вернитесь в окно терминала с запущенной интерактивной оболочкой SBT и запустите программу следующей командой:

```
> run
```

Эта программа должна вывести в терминал:

```
Hello, world!
```

Этих шагов достаточно для большинства примеров в данной книге. Иногда мы будем полагаться на внешние библиотеки, запуская примеры. Эти библиотеки автоматически обнаруживаются инструментом SBT в стандартных репозиториях. Для некоторых библиотек нам понадобится указать дополнительные репозитории, поэтому добавим следующие строки в фал `build.sbt`:

```
resolvers ++= Seq(
  "Sonatype OSS Snapshots" at
    "https://oss.sonatype.org/content/repositories/snapshots",
  "Sonatype OSS Releases" at
    "https://oss.sonatype.org/content/repositories/releases",
  "Typesafe Repository" at
    "http://repo.typesafe.com/typesafe/releases/"
)
```

Теперь, включив все необходимые репозитории, можно добавить несколько библиотек. Добавив следующую строку в файл `build.sbt`, мы получим доступ к библиотеке Apache Commons IO:

```
libraryDependencies += "commons-io" % "commons-io" % "2.4"
```

После изменения файла `build.sbt` его нужно перезагрузить во всех запущенных экземплярах SBT. Для этого в интерактивной оболочке SBT введите следующую команду:

```
> reload
```

Это позволит SBT обнаружить любые изменения в определениях в файле сборки и загрузить необходимые дополнительные программные пакеты.

Разные библиотеки Scala находятся в разных пространствах имен, которые называют пакетами. Чтобы получить доступ к содержимому определенного пакета, мы используем инструкцию `import`. Когда в примерах мы впервые используем некоторую библиотеку, мы всегда показываем набор необходимых инструкций `import`, но потом мы просто опускаем их ради экономии места.

По той же причине мы не повторяем объявление пакета в примерах кода. Мы всегда предполагаем, что все примеры в каждой конкретной главе принадлежат одному пакету. Например, все примеры в главе 2 «*Конкуренция в JVM и модель памяти в Java*» находятся в пакете `org.learningconcurrency.ch2`. Файлы с исходным кодом примеров из этой главы начинаются со следующих строк:

```
package org.learningconcurrency
package ch2
```

Наконец, в этой книге рассказывается о конкуренции и асинхронном выполнении. Многие примеры запускают параллельные вычисления, которые могут

продолжаться после завершения основной программы. Чтобы гарантировать, что такие параллельные вычисления всегда будут завершаться вместе с главной программой, когда выполняются в одной оболочке SBT, добавим следующую строку в файл `build.sbt`:

```
fork := false
```

В примерах, когда процессы должны выполняться под управлением разных экземпляров JVM, мы будет ясно указывать на это и давать четкие инструкции.

Использование Eclipse, IntelliJ IDEA и других IDE

Преимущество использования интегрированной среды разработки (Integrated Development Environment, IDE), такой как Eclipse или IntelliJ IDEA, заключается в возможности писать, компилировать и автоматически запускать программы на Scala. В этом случае нет необходимости устанавливать SBT, как описывалось в предыдущем разделе. Хотя мы советуем опробовать примеры из книги с помощью SBT, вы свободно можете использовать для этой цели свою IDE.

Однако мы должны предупредить, что такие редакторы, как Eclipse и IntelliJ IDEA, запускают программы в отдельном процессе JVM. Как отмечалось в предыдущем разделе, некоторые параллельные вычисления продолжают выполняться после завершения основной программы. Чтобы гарантировать завершение параллельных вычислений вместе с самой программой, вам может понадобиться добавить инструкцию `sleep` в конец главной программы с целью задержать ее. В большинстве примеров в этой книге инструкция `sleep` уже добавлена, но в некоторых программах вам может потребоваться добавить ее самим.

Кому адресована книга

Эта книга адресована в первую очередь разработчикам, умеющим писать последовательные программы на Scala и желающим научиться писать конкурентные программы. Предполагается, что у вас уже есть опыт программирования на языке Scala. На протяжении всей книги мы стремились использовать только наиболее простые особенности языка, чтобы показать, как писать конкурентный код, поэтому даже при наличии элементарных знаний вы не должны испытывать проблем в изучении предлагаемых в книге тем.

Однако нельзя сказать, что книга адресована исключительно разработчикам на Scala. Даже если вы имеете опыт программирования на Java, в .NET или на любом другом языке, вы наверняка найдете эту книгу полезной. Понимание основ объектно-ориентированного или функционального программирования является достаточным условием для чтения этой книги.

Наконец, эта книга является отличным введением в современное конкурентное программирование вообще. Даже если вы обладаете навыками реализации многопоточных вычислений и знаете, как устроена модель конкуренции в JVM, вы все равно узнаете здесь много нового о современных высокоуровневых средствах поддержки конкуренции. Многие библиотеки, представленные в этой книге, только начинают прокладывать свой путь к главенствующим языкам программирования, и некоторые из них являются по-настоящему передовыми технологиями.

Соглашения

В этой книге вы обнаружите несколько стилей оформления текста, которые разделяют различные виды информации. Ниже приводятся примеры этих стилей и поясняется их значение.

Элементы программного кода в тексте, имена таблиц в базах данных, имена папок и файлов, расширения файлов, пути к каталогам в файловой системе, фиктивные адреса URL, ввод пользователя и учетные записи в Twitter оформляются так: «Следующие инструкции читают ссылку и передают ее в функцию `BeautifulSoup`».

Блоки кода оформляются следующим образом:

```
package org
package object learningconcurrency {
  def log(msg: String): Unit =
    println(s"${Thread.currentThread.getName}: $msg")
}
```

Когда потребуется привлечь ваше внимание к определенному фрагменту в блоке программного кода, он будет выделяться жирным:

```
object ThreadsMain extends App {
  val t: Thread = Thread.currentThread
  val name = t.getName
  println(s"I am the thread $name")
}
```

Ввод или вывод в командной строке будет оформляться так:

```
$ mkdir scala-concurrency-examples
```

Новые термины и **важные слова** будут выделены жирным. Текст, отображаемый на экране, например в меню или в диалогах, будет оформляться так: «Чтобы загрузить новые модули, выберите пункт меню Files | Settings | Project Name | Project Interpreter (Файлы | Настройки | Имя проекта | Интерпретатор проекта)».

Так оформляются предупреждения и важные примечания.

Так оформляются советы и рекомендации.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.