

Оглавление

Предисловие	11
Благодарности	12
О книге	13
Кому адресована эта книга	13
Содержание книги	13
Соглашения об оформлении программного кода	15
Требования к программному обеспечению	15
Автор в сети.....	15
Об авторах	16
Об иллюстрации на обложке.....	16
Глава 1. Введение в Akka	18
1.1. Что такое Akka?.....	22
1.2. Актеры: краткий обзор	22
1.3. Два подхода к масштабированию: подготовка примера	24
1.4. Традиционное масштабирование	26
1.4.1. Традиционный подход к масштабированию и хранению: переместить все в базу данных	26
1.4.2. Традиционное масштабирование и интерактивная работа: опрос.....	29
1.4.3. Традиционное масштабирование и интерактивная работа: обработка ошибок	31
1.5. Масштабирование с Akka.....	32
1.5.1. Подход к масштабированию и хранению с Akka: отправка и прием сообщений	33
1.5.2. Масштабирование с Akka и интерактивная работа: отправка сообщений	36
1.5.3. Масштабирование с Akka и отказы: асинхронное разделение	37
1.5.4. Подход Akka: отправка и получение сообщений	37
1.6. Актеры: универсальная модель программирования	39
1.6.1. Модель асинхронного выполнения.....	40
1.6.2. Операции с актерами.....	41
1.7. Актеры Akka	44
1.7.1 ActorSystem	45
1.7.2. ActorRef, почтовый ящик и актер	47
1.7.3. Диспетчеры	47
1.7.4. Актеры и сеть	49
1.8. В заключение	49
Глава 2. Подготовка и запуск	51
2.1. Клонирование, сборка и интерфейс тестирования	52

2.1.1. Сборка с помощью sbt.....	53
2.1.2. Забегая вперед: REST-сервер GoTicks.com.....	54
2.2. Исследование акторов в приложении.....	59
2.2.1. Структура приложения	59
2.2.2. Актор, осуществляющий продажу: TicketSeller.....	64
2.2.3. Актор VoxOffice	65
2.2.4. Актор RestApi.....	67
2.3. Вперед, в облако	70
2.3.1. Создание приложения в облаке Heroku	71
2.3.2. Развертывание и запуск в Heroku	72
2.4. В заключение	73
Глава 3. Разработка с акторами через тестирование.....	75
3.1. Тестирование акторов.....	76
3.2. Односторонние взаимодействия	78
3.2.1. Примеры SilentActor	79
3.2.2. Пример SendingActor	84
3.2.3. Пример SideEffectingActor	89
3.3. Двусторонние взаимодействия.....	92
3.4. В заключение	93
Глава 4. Отказоустойчивость.....	95
4.1. Что такое отказоустойчивость.....	95
4.1.1. Простые объекты и исключения	98
4.1.2. И пусть падает	103
4.2. Жизненный цикл актора	107
4.2.1. Событие start	107
4.2.2. Событие stop.....	108
4.2.3. Событие restart	109
4.2.4. Объединяем фрагменты жизненного цикла вместе.....	111
4.2.5. Мониторинг жизненного цикла	113
4.3. Супервизор	114
4.3.1. Иерархия супервизора	114
4.3.2. Предопределенные стратегии	117
4.3.3. Собственные стратегии.....	118
4.4. В заключение	124
Глава 5. Объекты Future.....	125
5.1. Примеры использования объектов Future	126
5.2. Объекты Future не блокируют выполнение потока	131
5.2.1. Объекты Promise – это обещания.....	135
5.3. Обработка ошибок в объектах Future	138
5.4. Комбинирование объектов Future	143
5.5. Объединение объектов Future с акторами	152

5.6. В заключение	153
Глава 6. Первое распределенное приложение	155
6.1. Горизонтальное масштабирование	156
6.1.1. Общая терминология	156
6.1.2. Причины использования модели распределенного программирования	158
6.2. Горизонтальное масштабирование и удаленные взаимодействия	159
6.2.1. Реорганизация приложения GoTicks.com	161
6.2.2. Удаленные взаимодействия в REPL	161
6.2.3. Удаленный поиск.....	167
6.2.4. Удаленное развертывание	175
6.2.5. Тестирование с multi-JVM.....	180
6.3. В заключение	186
Глава 7. Настройка, журналирование и развертывание	188
7.1. Настройка.....	188
7.1.1. Попытка настройки Akka	189
7.1.2. Использование значений по умолчанию.....	192
7.1.3. Настройка Akka	195
7.1.4. Настройка для нескольких систем.....	196
7.2. Журналирование	199
7.2.1. Журналирование в приложении Akka	199
7.2.2. Использование журналирования	201
7.2.3. Управление журналированием из Akka	202
7.3. Развертывание приложений на основе акторов	204
7.4. В заключение	208
Глава 8. Шаблоны структуризации акторов.....	210
8.1. Конвейеры и фильтры	211
8.1.1. Шаблон: конвейеры и фильтры.....	211
8.1.2. Конвейеры и фильтры в Akka	212
8.2. Параллельная обработка дроблением с последующим объединением результатов	216
8.2.1. Область применения	216
8.2.2. Распараллеливание задач в Akka	218
8.2.3. Реализация компонента дробления с использованием списка получателей	219
8.2.4. Реализация компонента объединения с использованием агрегатора	221
8.2.5. Объединение компонентов в реализацию шаблона параллельной обработки дроблением	227
8.3. Маршрутизация.....	229
8.4. В заключение	234

Глава 9. Маршрутизация сообщений	236
9.1. Шаблон «Маршрутизатор».....	237
9.2. Балансировка нагрузки с помощью маршрутизаторов Akka.....	238
9.2.1. Маршрутизатор с пулом	242
9.2.2. Маршрутизатор с группой	250
9.2.3. Маршрутизатор ConsistentHashing	257
9.3. Реализация шаблона маршрутизатора с применением акторов.....	262
9.3.1. Маршрутизация по содержимому.....	262
9.3.2. Маршрутизация на основе состояния.....	263
9.3.3. Реализации маршрутизаторов	265
9.4. В заключение	266
Глава 10. Каналы обмена сообщениями	268
10.1. Типы каналов.....	269
10.1.1. Точка-точка.....	269
10.1.2. Издатель/подписчик	270
10.2. Специальные каналы	280
10.2.1. DeadLetter.....	281
10.2.2. Гарантированная доставка	283
10.3. В заключение	289
Глава 11. Конечные автоматы и агенты	291
11.1. Использование конечного автомата.....	292
11.1.1. Краткое введение в конечные автоматы	292
11.1.2. Создание модели конечного автомата	294
11.2. Реализация модели конечного автомата.....	295
11.2.1. Реализация переходов.....	296
11.2.2. Реализация действий при входе в состояния	301
11.2.3. Таймеры в конечном автомате.....	305
11.2.4. Завершение конечного автомата	308
11.3. Реализация общего состояния с помощью агентов.....	309
11.3.1. Простой доступ к общим данным с помощью агентов	310
11.3.2. Ожидание изменения состояния.....	312
11.4. В заключение	313
Глава 12. Интеграция с другими системами	315
12.1. Конечные точки сообщений	315
12.1.1. Нормализатор	317
12.1.2. Модель канонических данных.....	319
12.2. Реализация конечных точек с использованием Apache Camel	322
12.2.1. Реализация конечной точки-потребителя для приема сообщений из внешней системы	323
12.2.2. Реализация конечной точки-производителя для отправки сообщений во внешнюю систему.....	330

12.3. Реализация HTTP-интерфейса	335
12.3.1. Пример HTTP-интерфейса.....	336
12.3.2. Реализация конечной точки REST на основе akka-http	338
12.4. В заключение	344
Глава 13. Потоквые приложения.....	346
13.1. Основы потоковой обработки	347
13.1.1. Копирование файлов.....	351
13.1.2. Материализация запускаемых графов	355
13.1.3. Обработка событий в потоке	360
13.1.4. Обработка ошибок в потоках.....	364
13.1.5. Создание протокола с BidiFlow.....	366
13.2. Потоквая передача данных через HTTP	369
13.2.1. Прием потока данных по HTTP	370
13.2.2. Возврат потока данных по HTTP	372
13.2.3. Согласование контента	373
13.3. Ветвление и слияние со специализированным языком описания графов	378
13.3.1. Ветвление потоков	378
13.3.2. Слияние потоков	381
13.4. Посредничество между производителями и потребителями	384
13.4.1. Использование буферов.....	385
13.5. Обособление частей графа, действующих с разной скоростью	389
13.5.1. Медленный потребитель, накопление событий в блоках	389
13.5.2. Быстрый потребитель, дополнительные показатели	390
13.6. В заключение	391
Глава 14. Кластеры.....	393
14.1. Зачем нужны кластеры?	393
14.2. Членство в кластере	395
14.2.1. Присоединение к кластеру	396
14.2.2. Выход из кластера	404
14.3. Обработка заданий в кластере	410
14.3.1. Запуск кластера	412
14.3.2. Распределение заданий с использованием маршрутизаторов.....	414
14.3.3. Надежная обработка заданий.....	417
14.3.4. Тестирование кластера.....	424
14.4. В заключение	428
Глава 15. Хранимые акторы.....	430
15.1. Восстановление состояния с технологией Event Sourcing.....	432
15.1.1. Обновление записей на месте	432
15.1.2. Сохранение состояния без изменения.....	433
15.1.3. Event Sourcing для акторов	435

15.2. Хранимые акторы	436
15.2.1. Хранимый актор	437
15.2.2. Тестирование	441
15.2.3. Моментальные снимки	443
15.2.4. Запрос хранимых событий	449
15.2.5. Сериализация	451
15.3. Кластер на основе хранимых акторов	457
15.3.1. Расширение cluster singleton	461
15.3.2. Расширение cluster sharding	465
15.4. В заключение	470
Глава 16. Советы по повышению производительности	471
16.1. Анализ производительности	472
16.1.1. Производительность системы	472
16.1.2. Показатели производительности	474
16.2. Оценка производительности акторов	477
16.2.1. Сбор данных в почтовом ящике	478
16.2.2. Сбор и обработка данных	485
16.3. Улучшение производительности устранением узких мест	487
16.4. Настройка диспетчера	489
16.4.1. Выявление проблем с пулами потоков	489
16.4.2. Использование нескольких экземпляров диспетчеров	491
16.4.3. Изменение размера пула потоков статически	493
16.4.4. Изменение размера пула потоков динамически	496
16.5. Изменение поведения механизма освобождения потоков	498
16.5.1. Ограничения механизма освобождения потоков	500
16.6. В заключение	502
Глава 17. Заглядывая вперед	504
17.1. Модуль akka-typed	505
17.2. Akka Distributed Data	509
17.3. В заключение	509
Предметный указатель	511

Предисловие

Разработка хороших, конкурентных и распределенных приложений – сложная задача. Завершив очередной проект на Java, в котором потребовалось использовать массу кода для управления низкоуровневыми потоками выполнения, я задался целью найти более простой инструмент для реализации следующего проекта, который обещал быть еще более сложным.

В марте 2010 г. я увидел твит Дина Вамплера (Dean Wampler), который заставил меня обратить внимание на Akka:

```
W00t! RT @jboner: #akka 0.7 is released: http://bit.ly/9yRGSB
```

После недолгого изучения исходного кода и создания прототипа мы решили использовать Akka. Мы сразу заметили, что новая модель программирования действительно упрощает решение задач, с которыми мы испытали немало сложностей в предыдущем проекте.

Я убедил Роба Баккера (Rob Bakker) совершить со мной увлекательное путешествие в мир новой ультрасовременной технологии, и мы вместе отважно принялись за реализацию первого проекта с применением Scala и Akka. Мы сразу же обратились к Джонасу Бонеру (Jonas Bonér, создатель Akka) за помощью и, как выяснилось потом, оказались первыми широко известными пользователями Akka. Мы завершили этот проект, за которым последовало множество других, и всякий раз убеждались в преимуществах использования Akka.

В ту пору в Интернете было мало информации об Akka, поэтому я решил начать вести блог об этом фреймворке и таким способом внести свой вклад в его развитие.

Я был очень удивлен, когда мне предложили написать эту книгу. Я спросил у Роба, хочет ли он присоединиться ко мне. Позже мы поняли, что нам не обойтись без посторонней помощи, и пригласили Роба Уильямса (Rob Williams). К тому моменту он уже имел опыт создания проектов на Java и Akka.

Мы рады, что наконец смогли закончить эту книгу, описывающую версию Akka (2.4.9), которая включает исчерпывающий набор инструментов для создания распределенных и конкурентных приложений. Мы благодарны читателям, участвующим в программе MEAP (Manning Early Access Program) издательства Manning, за их отзывы. Также для нас, начинающих авторов, бесценной оказалась поддержка издательства Manning Publications.

Всех нас объединило понимание, полученное с опытом работы до использования Akka, что для разработки распределенных и конкурентных приложений на JVM необходим простой и надежный инструмент. Надеемся, нам удастся убедить вас, что Akka является именно таким инструментом.

Раймонд Рестенбург

Благодарности

Нам потребовалось много времени, чтобы написать эту книгу. На всем его протяжении нам помогало множество людей, и мы благодарны им за эту помощь. Я благодарю всех читателей, участвовавших в программе MEAP и купивших ранний выпуск этой книги, за их отзывы, которые помогли значительно улучшить книгу, и за их терпение в течение нескольких лет. Мы надеемся, что вам понравится конечный результат и вы многому научились, участвуя в программе MEAP.

Отдельное спасибо членам проекта Akka, в особенности Джонасу Бонеру (Jonas Bonér), Виктору Клангу (Viktor Klang), Роланду Куну (Roland Kuhn), Патрику Нордвалу (Patrik Nordwall), Бьерну Антонссону (Björn Antonsson), Эндре Варга (Endre Varga) и Конраду Малавски (Konrad Malawski) – все они вдохновляли нас и внесли свой бесценный вклад в книгу.

Мы также хотим поблагодарить Эдвина Рестенбурга (Edwin Roostenburg) и компанию CSC Traffic Management из Нидерландов, доверивших нам начать использовать Akka в важнейших проектах и давших невероятную возможность получить первый опыт с Akka. Мы также хотим сказать спасибо компании Xebia, позволившей Рею в рабочие часы писать книгу и предоставившей потрясающее рабочее место для экспериментов с Akka.

Мы благодарим издательство Manning Publications за оказанное нам доверие. Это наша первая книга, поэтому мы знаем, что это было рискованное предприятие для них. Мы хотим поблагодарить следующих сотрудников Manning за их превосходный труд: Майка Стефенса (Mike Stephens), Джеффа Блейла (Jeff Bleiel), Бена Берга (Ben Berg), Энди Кэрролл (Andy Carroll), Кевина Салливана (Kevin Sullivan), Кэти Теннант (Katie Tennant) и Дотти Марсико (Dottie Marsico).

Мы благодарим Дуга Уоррена (Doug Warren), выполнившего техническую корректуру всех глав. А также многих рецензентов, давших нам ценные советы во время работы над книгой: Энди Хикса (Andy Hicks), Дэвида Гриффита (David Griffith), Дюшана Кайсела (Dušan Kysel), Иэна Старкса (Iain Starks), Джереми Пьерре (Jeremy Pierre), Кевина Эслера (Kevin Esler), Марка Янссена (Mark Janssen), Майкла Шлейхардта (Michael Schleichardt), Ричарда Джеппса (Richard Jepps), Робина Перси (Robin Percy), Рона Ди Франго (Ron Di Frango) и Уильяма Е. Вилера (William E. Wheeler).

Напоследок, но не в последнюю очередь, мы хотим сказать спасибо всем, кто значит для нас больше всего на свете и поддерживал нас во время работы над книгой. Рей благодарит свою жену Шанель (Chanelle), а Роб Уильямс – свою маму, Гейл (Gail) и Лауру (Laurie).

О книге

В этой книге рассказывается о фреймворке Akka и описываются его наиболее важные модули. Мы сосредоточимся на модели программирования с акторами и на модулях поддержки акторов, часто используемых при создании конкурентных и распределенных приложений. На протяжении книги мы постоянно будем показывать, как тестировать код, что является важным аспектом повседневного труда разработчика программного обеспечения. Во всех наших примерах мы будем использовать язык программирования Scala.

После знакомства с основами программирования и тестирования акторов мы рассмотрим все важные аспекты, с которыми вам придется столкнуться при использовании фреймворка Akka в приложениях.

Кому адресована эта книга

Эта книга адресована всем, кто желает узнать, как создавать приложения с Akka. Примеры написаны на Scala, поэтому мы предполагаем, что вы уже имеете некоторое знакомство с этим языком программирования или желаете освоить его в процессе чтения. Также предполагается, что вы знакомы с языком Java, особенно если учесть, что Scala действует поверх JVM.

Содержание книги

Книга состоит из семнадцати глав.

Глава 1 знакомит с акторами Akka. Здесь вы узнаете, как модель программирования с акторами решает некоторые ключевые проблемы, которые традиционно осложняют масштабирование приложений.

Глава 2 сразу же погружается в пример HTTP-службы, реализованной с применением Akka, чтобы показать, как быстро можно создать действующую службу и запустить ее в облаке. Она позволит вам понять, о чем рассказывается в последующих главах.

Глава 3 рассказывает о модульном тестировании акторов с использованием ScalaTest и модуля akka-testkit.

Глава 4 объясняет, как мониторинг акторов позволяет создавать надежные, отказоустойчивые системы.

Глава 5 знакомит с объектами Future, чрезвычайно удобным и простым инструментом объединения результатов функций, выполняющихся асинхронно. Здесь вы также узнаете, как объединить акторы и объекты Future.

Глава 6 рассказывает о модуле akka-remote, позволяющем взаимодействовать с распределенными актерами по сети. Здесь вы также узнаете, как осуществлять модульное тестирование распределенной системы акторов.

Глава 7 объясняет, как для настройки Akka использовать библиотеку Typesafe Config Library. В ней также рассказывается, как можно использовать эту библиотеку для настройки компонентов вашего приложения.

Глава 8 описывает шаблоны структуризации приложений на основе акторов. Здесь вы узнаете, как реализовать пару классических шаблонов интеграции корпоративных приложений.

Глава 9 объясняет, как применять маршрутизаторы. Маршрутизаторы можно использовать для переключения, широковещательной рассылки и балансировки сообщений между актерами.

Глава 10 знакомит с каналами сообщений, которые можно использовать для передачи сообщений от одного актора другому. Здесь вы познакомитесь с каналами вида точка-точка и публикация/подписка, а также с каналами для недоставленных сообщений и с каналами гарантированной доставки.

Глава 11 обсуждает вопросы конструирования конечных автоматов акторов с использованием модуля FSM и знакомит с агентами, которые можно использовать для асинхронной передачи состояния.

Глава 12 объясняет приемы интеграции с другими системами. В этой главе вы узнаете, как организовать поддержку различных протоколов с помощью Apache Camel и как сконструировать http-службу с применением модуля akka-http.

Глава 13 знакомит с модулем akka-stream. Здесь вы узнаете, как с использованием Akka конструировать потоковые приложения. В этой главе подробно описывается процесс создания потоковой HTTP-службы для обработки событий в журнале.

Глава 14 объясняет, как пользоваться модулем akka-cluster. Здесь вы узнаете, как динамически масштабировать акторы в сетевом кластере.

Глава 15 знакомит с модулем akka-persistence. В этой главе вы узнаете, как записывать и восстанавливать состояние с использованием хранимых акторов, как использовать кластерные расширения для создания приложения покупательской корзины в кластере.

Глава 16 обсуждает ключевые аспекты производительности систем акторов и дает советы по анализу проблем, связанных с производительностью.

Глава 17 заглядывает вперед и знакомит с двумя грядущими нововведениями, которые, как нам кажется, приобретут особую важность: модулем

akka-typed, который делает возможным проверку сообщений акторов на этапе компиляции, и модулем akka-distributed-data, который обеспечивает распределение состояния в памяти кластера.

Соглашения об оформлении программного кода

Весь исходный код в листингах или в тексте оформлен моноширинным шрифтом, чтобы выделить его на фоне обычного текста. Многие листинги сопровождаются примечаниями и комментариями, подчеркивающими важные понятия. Код примеров из этой книги доступен для загрузки на веб-сайте издательства Manning www.manning.com/books/akka-in-action и в репозитории GitHub <https://github.com/RayRoestenburg/akka-in-action>.

Требования к программному обеспечению

Все примеры написаны на языке Scala. Они были протестированы с версией Scala 2.11.8. Найти дистрибутив Scala можно здесь: <http://www.scala-lang.org/download/>.

Обязательно установите последнюю версию sbt (0.13.12 на момент написания этих строк); если у вас установлена более старая версия sbt, вы рискуете столкнуться с проблемами. Загрузить дистрибутив sbt можно отсюда: <http://www.scala-sbt.org/download.html>.

Фреймворк Akka версии 2.4.9 требует наличия Java 8, поэтому вам также придется установить эту версию Java. Ее можно найти по адресу: <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>.

Автор в сети

Одновременно с покупкой «Akka в действии» вы получаете бесплатный доступ к частному веб-форуму, организованному издательством Manning Publications, где можно оставлять комментарии о книге, задавать технические вопросы, а также получать помощь от автора и других пользователей. Чтобы получить доступ к форуму и зарегистрироваться на нем, откройте в веб-браузере страницу <https://www.manning.com/books/akka-inaction>. Здесь описывается, как попасть на форум после регистрации, какие виды помощи доступны и правила поведения на форуме.

Издательство Manning обязуется предоставить своим читателям место встречи, где может состояться содержательный диалог между отдельными читателями и между читателями и автором. Но со стороны автора отсутствуют какие-либо обязательства уделять форуму какое-то определенное внимание – его присутствие на форуме остается добровольным (и неоплачиваемым). Мы предлагаем задавать автору стимулирующие вопросы, чтобы его интерес не угасал!

Форум и архивы предыдущих обсуждений будут доступны на сайте издательства, пока книга находится в печати.

Об авторах

Раймонд Рестенбург (Raymond Roestenburg) – опытный разработчик, программист-полиглот и архитектор ПО. Активный член сообщества Scala, внештатный разработчик Akka, принимавший участие в разработке модуля Akka-Camel.

Роб Баккер (Rob Bakker) – опытный разработчик ПО, занимающийся созданием серверных систем и их интеграцией. Использует Scala и Akka начиная с версии 0.7.

Роб Уильямс (Rob Williams) – основатель онтометрики (ontometric), практики, ориентированной на Java-решения, включая машинное обучение. Впервые использовал акторы десять лет тому назад и на их основе реализовал несколько проектов.

Об иллюстрации на обложке

Иллюстрация с изображением китайского императора на обложке «Akka в действии» взята из книги «Collection of the Dresses of Different Nations, Ancient and Modern» (Коллекция костюмов разных народов, античных и современных) Томаса Джеффериса (Thomas Jefferys), опубликованной в Лондоне между 1757 и 1772 годом. На титульной странице указано, что это выполненная вручную каллиграфическая цветная гравюра, обработанная гуммиарабиком.

Томас Джефферис (1719–1771) носил звание «Географ короля Георга III». Английский картограф, был ведущим поставщиком карт того времени. Он выгравировал и напечатал множество карт для нужд правительства, других официальных органов и широкий спектр коммерческих карт и атласов, в частности Северной Америки. Будучи картографом, интересовался местной одеждой народов, населяющих разные земли, и собрал блестящую коллекцию различных платьев в четырех томах.

Очарование далеких земель и дальних путешествий для удовольствия было относительно новым явлением в конце XVIII века, и коллекции, такие как эта, были весьма популярны, знакомя с внешним видом жителей других стран. Разнообразие рисунков, собранных Джефферисом, свидетельствует о проявлении народами мира около 200 лет яркой индивидуальности и уникальности. С тех пор стиль одежды сильно изменился, и исчезло разнообразие, характеризующее различные области и страны. Теперь трудно отличить по одежде даже жителей разных континентов. Если взглянуть на это с оптимистической точки зрения, мы пожертвовали культурным и внешним разнообразием в угоду разнообразию личной

жизни, или в угоду более разнообразной и интересной интеллектуальной и технической деятельности.

В наше время, когда трудно отличить одну техническую книгу от другой, издательство Manning проявило инициативу и деловую сметку, украшая обложки книг изображениями, основанными на богатом разнообразии жизненного уклада народов двухвековой давности, придав новую жизнь рисункам Джеффериса.

Глава 1

Введение в Akka

В этой главе:

- почему масштабирование вызывает сложности;
- написано однажды, масштабируется как угодно;
- введение в модель программирования с акторами;
- акторы Akka;
- что такое Akka?

Вплоть до середины 90-х, как раз незадолго до интернет-революции, было общепринято создавать приложения, выполняющиеся на единственном компьютере с единственным процессором. Если приложение оказывалось недостаточно быстродействующим, обычным решением было подождать, пока появится компьютер с более мощным процессором. Проблема решалась сама собой, без изменения исходного кода. Программисты по всему миру таскали бесплатный сыр, и жизнь была прекрасна.

В 2005 г. Херб Саттер (Herb Sutter) написал в журнале *Dr. Dobbs Journal* о необходимости фундаментальных изменений (<http://www.gotw.ca/publications/concurrency-ddj.htm>). Суть статьи в том, что достигнут физический предел увеличения тактовой частоты процессора и бесплатный обед закончился.

Если приложениям нужно обрабатывать больше данных или обеспечить поддержку большего числа пользователей, они должны стать *конкурентными*. (Строгое определение этого термина мы дадим позже, а пока просто считайте его аналогом термина *многопоточные*. На самом деле это не совсем верно, но на данный момент вполне приемлемая аналогия.)

Масштабируемость – это мера способности системы адаптироваться к изменению спроса на ресурсы без отрицательного влияния на производительность. *Конкуренция* – это средство достижения масштабируемости: предполагается, что при необходимости в серверы могут быть добавлены дополнительные процессоры, и приложение автоматически начнет их

использовать. Это следующее отличное место, откуда можно таскать бесплатный сыр.

Примерно в 2005 г., когда Херб Саттер написал свою замечательную статью, можно было найти компании, запускавшие приложения на кластерах из многопроцессорных серверов (часто их число не превышало двух-трех, на случай, если один из них выйдет из строя). Поддержка конкурентного выполнения в языках программирования имела, но была очень ограниченной и многими смертными программистами считалась черной магией. Херб Саттер предсказал в своей статье, что «языки программирования... будут вынуждены реализовать все лучшие инструменты поддержки конкуренции».

А теперь посмотрим, какие изменения произошли за десятилетие! Вернувшись в сегодняшний день, мы сразу замечаем приложения, выполняющиеся на большом количестве серверов в облаке, интегрирующем множество систем в разных вычислительных центрах. Все возрастающие потребности конечных пользователей подталкивают требования к производительности и стабильности создаваемых вами систем.

И в чем заключаются эти новые средства поддержки конкуренции? Поддержка конкурентного выполнения в большинстве языков программирования, и особенно в JVM, почти не изменилась. Детали прикладного интерфейса (API) механизма конкуренции значительно усовершенствовались, но программисту все еще приходится работать с низкоуровневыми конструкциями, такими как потоки выполнения и блокировки, которые, как известно, довольно сложны в обращении.

Помимо вертикального масштабирования (увеличения объема вычислительных ресурсов; например количества процессоров на имеющихся серверах), есть также возможность осуществлять *горизонтальное масштабирование* – добавлять в кластер новые серверы. Поскольку с 90-х мало что изменилось в поддержке сетевых взаимодействий языками программирования, многие технологии по-прежнему используют механизм RPC (Remote Procedure Calls – вызовы удаленных процедур) для обмена данными через сеть.

Между тем успехи в организации облачных услуг и создании процессов с многоядерной архитектурой сделали вычислительные ресурсы еще более доступными.

Предложения PaaS (Platform as a Service – платформа как служба) упростили создание и развертывание очень больших распределенных приложений, которые когда-то могли себе позволить только крупнейшие игроки в индустрии информационных технологий (ИТ). Облачные услуги, такие как AWS EC2 (Amazon Web Services Elastic Compute Cloud) и Google Compute Engine, дают возможность за минуту развернуть буквально тысячи серверов, а поддержка таких инструментов, как Docker, Puppet, Ansible и многих других, упростит управление приложениями на виртуальных серверах.

Количество ядер в процессорах также постоянно растет: даже мобильные телефоны и планшетные компьютеры стали оснащаться многоядерными процессорами.

Но это не означает, что вы можете использовать сколько угодно ресурсов для решения своих задач. В конце концов, все сводится к стоимости и эффективности. То есть приложения должны обеспечивать максимальную эффективность масштабирования, чтобы вложенные деньги окупили себя. Вы никогда не будете использовать алгоритм сортировки с экспоненциальной временной сложностью, и точно так же вы должны задумываться о стоимости масштабирования.

С масштабированием приложений мы обычно связываем два ожидания:

- поскольку добиться соответствия возрастающим требованиям с конечным объемом ресурсов практически нереально, в идеале мы рассчитываем на более медленный рост потребностей в ресурсах, чем требований к приложению. На рис. 1.1 изображена зависимость между требованиями и необходимыми ресурсами;

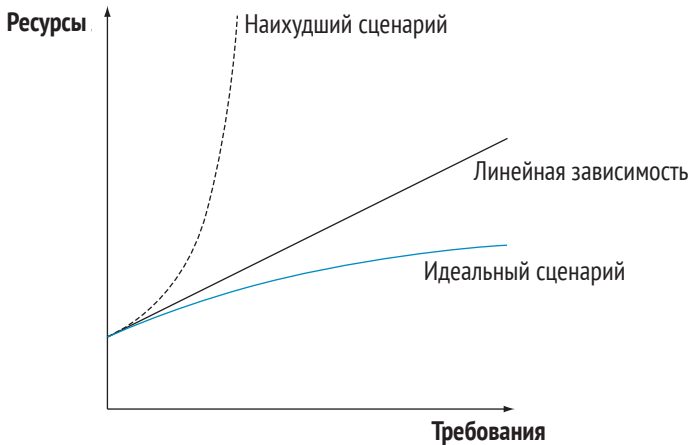


Рис. 1.1. Зависимость требований от ресурсов

- в идеале, если возникает необходимость в наращивании ресурсов, хотелось бы, чтобы сложность приложения оставалась на прежнем уровне или хотя бы росла как можно медленнее. (Вспомните бесплатный сыр в прошлом, когда для ускорения приложения не требовалось увеличивать его сложность!) На рис. 1.2 изображена зависимость между объемом ресурсов и сложностью приложения.

Объем требуемых ресурсов и сложность приложения составляют основную стоимость масштабирования.

Мы исключили множество факторов из этих упрощенных расчетов, но легко видеть, что обе эти составляющие вносят наибольший вклад в общую стоимость масштабирования.

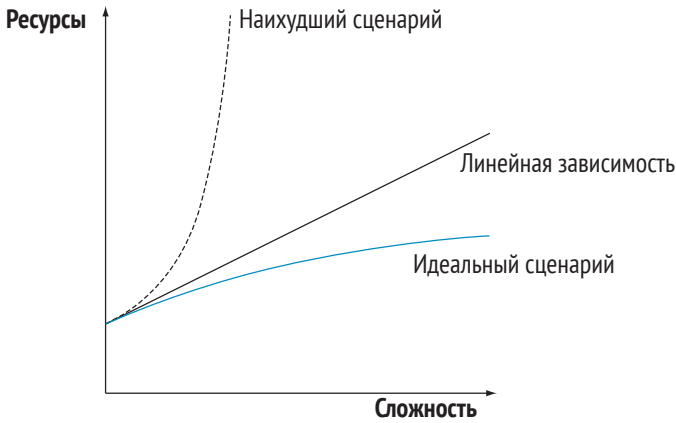


Рис. 1.2. Зависимость сложности от объема требуемых ресурсов

Один из наихудших сценариев – когда приходится платить все больше и больше за все более недоиспользуемые ресурсы. Другой жуткий сценарий – когда сложность приложения переливается через край с добавлением новых ресурсов.

В связи с этим возникает две цели при масштабировании приложения: сложность должна оставаться на как можно более низком уровне и ресурсы должны использоваться максимально эффективно.

Можно ли для достижения этих целей использовать привычные инструменты (потoki и RPC)? Масштабирование по горизонтали с применением RPC и по вертикали с применением низкоуровневых потоков выполнения – не лучший выход. Механизм RPC делает сетевые вызовы не отличимыми от вызовов локальных методов. Каждому вызову RPC приходится блокировать текущий поток выполнения и ждать ответа из сети для работы абстракции вызова локального метода, что может обходиться слишком дорого. Это препятствует эффективному использованию ресурсов.

Другая проблема заключается в необходимости точно знать, как будет выполняться масштабирование – по горизонтали или по вертикали. Многопоточное программирование и сетевое программирование на основе RPC – это как яблоки и груши: они применяются в разных контекстах, используют разную семантику и работают на разных уровнях абстракции. Вам приходится жестко определять, какие части приложения используют потоки для вертикального масштабирования, а какие – RPC для масштабирования по горизонтали.

Наличие жестко заданных методов, действующих на разных уровнях абстракции, значительно увеличивает сложность. А теперь подумайте, что проще – программирование с применением двух запутанных программных конструкций (RPC и потоков) или только с одной конструкцией? Многопрофильный подход к масштабированию намного сложнее, чем необходимо для гибкой адаптации к изменяющимся требованиям.

Запустить тысячу серверов сегодня проще простого, но, как вы увидите в этой первой главе, этого нельзя сказать об их программировании.

1.1. Что такое Akka?

В этой книге мы покажем, как инструменты Akka, открытого проекта, разрабатываемого компанией Lightbend, обеспечивают более простую модель программирования конкурентных и распределенных приложений – *модель акторов*. Акторы не являются чем-то принципиально новым. Они представляют способ масштабирования приложений для JVM в обоих направлениях – по вертикали и по горизонтали. Как вы увидите далее, Akka эффективно использует ресурсы и позволяет сохранить сложность масштабируемых приложений на относительно низком уровне.

Главной целью Akka является простота создания приложений, разворачиваемых в облаке или запускаемых на многоядерных процессорах, которые эффективно используют все имеющиеся вычислительные ресурсы. Этот фреймворк предоставляет модель акторов, среду времени выполнения и инструменты, необходимые для создания масштабируемых приложений.

1.2. Акторы: краткий обзор

Основу Akka составляют акторы. Большинство компонентов в Akka тем или иным способом поддерживают работу с акторами, будь в том числе средства настройки акторов, соединения акторов с сетью, управления работой акторов и создания кластеров из акторов. Фреймворк Akka выгодно отличается простотой поддержки и наличием инструментов для создания приложений на основе акторов, благодаря чему вы легко сможете сосредоточиться на мышлении и программировании в терминах акторов.

Акторы можно сравнить с очередями сообщений, не имеющими накладных расходов на установку и настройку брокера сообщений. Они подобны программируемым очередям сообщений, сжатым до микроскопических размеров, – вы легко сможете создать тысячи и даже миллионы акторов. Они «ничего не делают», если не посылают сообщения.

Сообщения – это простые структуры данных, которые нельзя изменить после создания, то есть они *неизменяемы*.

Акторы могут получать сообщения по очереди и выполнять в ответ некоторые действия. В отличие от очередей, они могут также посылать сообщения (другим акторам).

Все свои действия акторы выполняют асинхронно. Проще говоря, вы можете отправить сообщение актору и продолжить работу, не дожидаясь ответа. Акторы не похожи на потоки выполнения, но сообщения, посылаемые им, в какой-то момент пересекают некоторый другой поток. Позднее

вы увидите, как настраиваются связи акторов с потоками, а пока просто знайте, что это не жесткие отношения.

Манифест реактивного программирования

Манифест реактивного программирования «The Reactive Manifesto» (<http://www.reactivemanifesto.org/>) – это инициатива, цель которой – помочь в создании систем, более надежных, более устойчивых, более гибких и лучше соответствующих требованиям современности. Коллектив разработчиков Акка с самого начала был вовлечен в создание манифеста и фреймворк Акка – это результат воплощения идей, выраженных в этом манифесте.

Драйвером большей части манифеста является эффективное использование ресурсов и возможность автоматического масштабирования приложений (также называется *эластичностью*):

- блокирующий ввод/вывод ограничивает возможности параллельного выполнения, поэтому предпочтительнее использовать неблокирующий ввод/вывод;
- синхронные взаимодействия ограничивают возможности параллельного выполнения, поэтому предпочтительнее использовать асинхронные взаимодействия;
- последовательный опрос требует больше ресурсов, поэтому предпочтительнее использовать подходы, основанные на событиях;
- если выход из строя одного узла может нарушить нормальную работу всех остальных узлов, такая организация является напрасным расходом ресурсов, поэтому нужно изолировать ошибки (увеличить устойчивость), чтобы исключить вероятность потери всей вашей работы;
- системы должны быть эластичными: с уменьшением объема работы система должна потреблять меньше ресурсов; с увеличением объема работы система должна потреблять больше ресурсов, но не больше необходимого минимума.

Сложность составляет основную часть стоимости, поэтому если вы не можете что-то протестировать, изменить или запрограммировать – у вас большие проблемы.

(Перевод «The Reactive Manifesto» на русский язык можно найти по адресу: <http://mrdekk.ru/2014/10/17/reactive-manifesto-rus/>. – Прим. перев.)

Мы детально исследуем акторы в этой книге, но сейчас самое важное, что вы должны понять: принцип действия приложений на основе акторов основан на отправке и приеме сообщений. Сообщения могут обрабаты-

ваться локально, в параллельном потоке выполнения, или удаленно, на другом сервере. Точное место обработки сообщения и где должен находиться актер – все это можно решить позднее. Этим модель акторов выгодно отличается от непосредственного использования потоков выполнения и сетевых взаимодействий в стиле RPC. Акторы упрощают сборку приложений из мелких компонентов, напоминающих сетевые службы, сжатые до микроскопических размеров в смысле занимаемого места и административных издержек.

1.3. Два подхода к масштабированию: подготовка примера

В оставшейся части главы мы исследуем пример коммерческого приложения чата и посмотрим, с какими сложностями приходится сталкиваться при попытке масштабировать его за счет увеличения количества серверов (и при необходимости одновременно обрабатывать миллионы событий). Мы рассмотрим традиционное решение, которое, вероятно, знакомо вам из опыта создания подобных приложений (с применением потоков выполнения и блокировок, RPC и т. п.), и сравним его с подходом на основе использования Akka.

Демонстрацию традиционного подхода мы начнем с простого приложения, размещающегося в памяти, которое затем превращается в приложение, целиком и полностью опирающееся на базу данных как для поддержки конкуренции, так для хранения изменяемого состояния. Чтобы повысить интерактивность такого приложения, нам не остается ничего иного, как организовать опрос базы данных. Мы покажем, что сочетание базы данных и RPC-взаимодействий влечет плохо контролируемое увеличение сложности приложения. Мы также покажем, что изолировать ошибки в этом приложении становится все труднее и труднее с каждым новым шагом вперед. Мы уверены, что многие из этих проблем давно знакомы вам.

Затем мы посмотрим, как модель акторов упрощает приложение и как фреймворк Akka помогает написать приложение однажды и масштабировать его при любой возможности (автоматически решая проблемы конкурентного выполнения). В табл. 1.1 перечислены различия между двумя подходами. Некоторые пункты пока будут непонятны, но мы проясним их в следующих разделах.

Вообразите, что мы решили покорить мир с помощью современного приложения чата, революционизирующего область онлайн-сотрудничества. Его цель – помогать членам одной команды быстро находить друг друга и работать вместе. У нас очень много идей, которые мы могли бы реализовать в этом приложении, в том числе интеграция с инструментами управления проектами и с существующими службами связи.

Конец ознакомительного фрагмента.

Приобрести книгу можно

в интернет-магазине

«Электронный универс»

e-Univers.ru