Посвящается Паулине Съцежка, самому честному и прямому человеку из всех моих знакомых.

За доверие и помощь, отнюдь не ограничивающуюся написанием книги.

За мою жизнь, которую она изменила в куда большей степени, чем может себе представить.

– Томаш Нуркевич

Оглавление

Предисловие

28 октября 2005 года Рэй Оззи (Ray Ozzie), только что назначенный главным архитектором Майкрософт, разослал своим сотрудникам получившее скандальную известность письмо, озаглавленное «Крах Интернет-служб». В нем он по сути дела описывает тот мир, который мы знаем сегодня, – мир, где такие корпорации, как Microsoft, Google, Facebook, Amazon и Netflix используют веб в качестве основного канала доставки своих услуг.

В письме Оззи есть мысль, которую разработчики не часто слышат от топ-менеджеров крупной корпорации:

> Сложность убивает. Она высасывает соки из разработчиков, затрудняет планирование, сборку и тестирование продуктов, порождает проблемы в части безопасности и служит причиной горьких разочарований пользователей и администраторов.

Прежде всего, следует принять во внимание, что в 2005 году крупные ИТ-компании были всем сердцем влюблены в умопомрачительно сложные технологии типа SOAP, WS-* и XML. В то время еще не было слова «микросервисы» и даже на горизонте не просматривалась простая технология, которая позволила бы разработчикам справиться с проблемами составления асинхронной композиции простых служб для получения более сложных, не упуская из виду такие аспекты, как обработка ошибок, задержки, безопасность и эффективность.

Для моей группы облачных программных продуктов письмо Оззи стало призывом не заниматься ерундой, а сосредоточиться на придумывании простой модели программирования, которая позволит создавать крупномасштабные асинхронные архитектуры Интернет-служб для работы с большими объемами данных. После многих фальстартов на нас наконец снизошло озарение: взяв за основу интерфейсы Iterable/Iterator для синхронных коллекций, мы могли бы получить пару интерфейсов для представления потоков асинхронных событий и применять всем хорошо знакомые операции над последовательностями – map, filter, scan, zip, groupBy и другие – к преобразованию и комбинированию асинхронных потоков данных. Так летом 2007 года родилась идея Rx. В процессе реализации мы поняли, что необходимо как-то управлять конкурентностью и временем и с этой целью обобщили идею исполнителей в Java, дополнив ее виртуальным временем и кооперативной многозадачностью.

После напряженной двухлетней работы, когда были опробованы и отвергнуты разнообразные проектные решения, 18 ноября 2009 года мы наконец выпустили

первую версию Rx.NET. Вскоре после этого мы перенесли Rx на Microsoft.Phone. Reactive для Windows Phone 7 и приступили к реализации Rx на таких языках, как JavaScript и C++, а заодно поэкспериментировали с Ruby и Objective-C.

Внутри Майкрософт первым пользователем Rx стал Джафар Хусейн (Jafar Husain), эту технологию он взял с собой, когда перешел в Netflix в 2011 году. Джафар всячески пропагандировал Rx в компании и в конце концов полностью переделал клиентскую часть пользовательского интерфейса Netflix на основе асинхронной обработки потоков. И, к счастью для всех нас, он заразил своим энтузиазмом Бена Кристенсена, занимавшегося в Netflix разработкой API промежуточного уровня. В 2012 году Бен начал работать над RxJava и в начале 2013 разместил весь код на Github, сделав его открытым. Еще одним из ранних приверженцев Rx в Майкрософт был Пол Беттс (Paul Betts), позже он перешел в Github и убедил коллег, в т. ч. Джастина Спар-Саммерса (Justin Spahr-Summers) реализовать и выпустить ReactiveCocoa для Objective-C, что и произошло весной 2012.

Поскольку Rx завоевывал популярность в отрасли, мы убедили отдел Microsoft Open Tech раскрыть код Rx.NET, это случилось осенью 2012. Вскоре после этого я ушел из Майкрософт, основал компанию Applied Duality и посвятил все свое время тому, чтобы сделать Rx стандартным кросс-языковым и кросс-платформенным API для асинхронной обработки потоков данных в режиме реального времени.

К 2016 году популярность Rx стремительно возросла, как и число пользователей. Весь трафик, проходящий через Netflix API, обрабатывается RxJava. То же можно сказать о библиотеке обеспечения отказоустойчивости Hystrix, лежащей в основе всего внутреннего трафика службы, и сопутствующих реактивных библиотеках RxNetty и Mantis. Сейчас Netflix работает над полностью реактивным сетевым стеком для связывания всех внутренних служб, пересекающих границы процессов и машин. RxJava нашла также весьма полезные применения в экосистеме Android. Компании SoundCloud, Square, NYT, Seatgeek используют RxJava в своих приложениях и вносят вклад в разработку дополнительной библиотеки RxAndroid. Такие поставщики NoSQL-решений, как Couchbase и Splunk, также предлагают основанные на Rx интерфейсы к уровню доступа к данным. Среди других Java-библиотек, воспринявших RxJava, упомянем Camel Rx, Square Retrofit и Vert.x. В сообществе JavaScript широко распространена библиотека RxJS, лежащая в основе популярного каркаса Angular 2. Сообщество поддерживает сайт *http://reactivex.io/*, на котором можно найти информацию о реализациях Rx на многих языках, а также фантастические камешковые диаграммы с пояснениями, созданные Дэвидом Гроссом (@CallHimMoorlock).

С самого начала проект Rx развивался в соответствии с потребностями сообщества разработчиков и при его активном участии. В оригинальной реализации Rx в .NET упор был сделан, прежде всего, на преобразовании асинхронных потоков событий и использовании асинхронных перечислимых объектов в ситуациях, где требуется противодавление. Поскольку в Java нет языковой поддержки асинхронного ожидания, сообщество дополнило типы Observer и Observable концепцией реактивного вытягивания и добавило интерфейс Producer. Благодаря усилиям

многих разработчиков реализация RxJava получилась весьма изощренной и в высшей степени оптимизированной.

Несмотря на то что детали RxJava несколько отличаются от других реализаций Rx, библиотека все равно ориентирована специально на разработчиков, стремящихся выжить в прекрасном новом мире распределенной обработки данных в реальном времени и сконцентрироваться на эссенциальной, а не акцидентальной сложности, высасывающей из нас все соки. Эта книга содержит глубокое и подробное изложение концепций и принципов использования Rx вообще и RxJava в частности, написанное двумя авторами, которые потратили бессчетное количество часов на реализацию RxJava и применение ее к реальным задачам. Если вам нужна «реактивность», то лучшего способа, чем купить книгу, не придумаешь.

– Эрик Мейер, основатель и президент компании Applied Duality, Inc.

Вступление

Для кого написана эта книга

Книга ориентирована на Java-программистов средней и высокой квалификации. Читатель должен свободно владеть языком Java, но предварительное знакомство с реактивным программированием не предполагается. Многие описываемые концепции относятся к функциональному программированию, но знакомство с ним также не обязательно. Особенно полезна книга будет двумя группам программистов.

- Профессионалы, которым нужно повысить производительность сервера или сделать код для мобильных устройств более удобным для сопровождения. Если вы из их числа, то найдете здесь идеи и готовые решения реальных проблем, а также практические советы. А RxJava тогда следует считать просто еще одним инструментом, который книга поможет освоить.
- Любопытные разработчики, которые слыхали о реактивном программировании или конкретно о RxJava и хотели бы понять, что это такое. Если вы относите себя к этой категории и не планируете немедленно использовать преимущества RxJava в производственном коде, то книга заметно обогатит ваш багаж знаний.

Наконец, это книга станет подспорьем для практикующего архитектора программного обеспечения. RxJava оказала сильное влияние на общую архитектуру целых систем, поэтому знать об этой технологии полезно. Но даже если вы только начинаете путешествие в мир программирования, все равно попробуйте прочитать первые главы, в которых объяснены основы. Понятия преобразования и композиции универсальны и не являются спецификой реактивного программирования.

Несколько слов от Бена Кристенсена

В 2012 году я работал над новой архитектурой Netflix API. По ходу дела стало ясно, что для достижения поставленных целей необходимо включить конкурентность и асинхронные сетевые запросы. Исследуя различные подходы, я столкнулся с Джафаром Хусейном (*https://github.com/jhusain*), который попытался заинтересовать меня технологией Rx, с которой познакомился, работая в Майкрософт. В то время я довольно сносно владел техникой конкурентного программирования,

но размышлял о нем в императивных терминах и преимущественно с точки зрения Java, поскольку именно программированием на Java зарабатывал себе на хлеб.

Поэтому мне трудно было воспринять пропагандируемый Джафаром подход из-за его функциональной ориентированности, и я не поддавался на его убеждения. За этим последовали месяцы споров и дискуссий, архитектура системы становилась все более зрелой, а мы с Джафаром снова и снова встречались у доски, пока я, наконец, не врубился в теоретические принципы, а затем и не оценил элегантность и эффективность подходов, предлагаемых Rx.

Мы решили включить модель программирования Rx в Netflix API и в конечном итоге создали реализацию на Java, которую назвали RxJava, следуя заданным Майкрософт образцам: Rx.Net и RxJS.

За примерно три года, когда создавалась библиотека RxJava, по большей части на GitHub, в виде открытого кода, я имел удовольствие работать с растущим сообществом и соавторами, каковых было 120 с лишним, и вместе нам удалось превратить RxJava в зрелый продукт, используемый во многих системах как на стороне сервера, так и на стороне клиента. Он собрал больше 15 000 звезд на GitHub, что позволило войти в первые 200 проектов (*https://github.com/search?p=11&q=stars:%3E1&s=stars&type=Repositories*) и занять третье место среди проектов на Java (*https://github.com/search?l=Java&p=1&q=stars:%3E1&s=stars&type=Repositories*).

Джордж Кэмпбелл (George Campbell), Аарон Талл (Aaron Tull) и Мэтт Джекобс (Matt Jacobs) из Netflix много сделали для превращения RxJava из первых сборок в то, чем она является теперь. В частности, им проект обязан добавлением `lift`, `Subscriber`, противодавления и поддержки других языков на базе JVM. Дэвид Карнок (David Karnok) присоединился к проекту позже, но уже обошел меня по числу фиксаций и написанных строк кода. Ему проект в значительной мере обязан своим успехом, а теперь он возглавил его.

Хочу поблагодарить Эрика Мейера, который создал Rx во время работы в Майкрософт. С тех пор как он уволился оттуда, я урывками общался с ним в Netflix, когда трудился над RxJava, а теперь счастлив работать вместе в Facebook. Я считаю большой честью обсуждать с ним различные вопросы у доски и учиться у него. С таким наставником, как Эрик, поднимаешься на новый уровень мышления.

Попутно мне приходилось много раз выступать на конференциях с докладами о RxJava и реактивном программировании, и я повстречал много людей, которые помогли мне узнать о коде и архитектуре куда больше, чем я мог бы достичь собственными силами.

Netflix оказала мне феноменальное содействие, позволяя тратить время и силы на этот проект и выделив специалистов для написания технической документации, – сам я с этим точно не справился бы. Проект с открытым исходным кодом такого масштаба и качества никогда не стал бы успешным, если бы у меня не было возможности заниматься им в рабочее время и привлекать людей с разными знаниями и умениями.

Первая глава книги представляет собой мою попытку объяснить, почему реактивное программирование вообще полезно и как конкретно в RxJava реализованы общие принципы.

Весь остальной текст написан Томашем, который проделал потрясающую работу. У меня была возможность читать черновики и вносить предложения, но это его книга, и именно он писал обо всех деталях, начиная со второй главы.

Несколько слов от Томаша Нуркевича

Я впервые столкнулся с RxJava в 2013 году, работая в одной финансовой организации. Мы занимались обработкой больших потоков рыночных данных в реальном времени. В тот момент конвейер состоял из Kafka (доставка сообщений), Akka (обработка данных о торговых сделках), Clojure (преобразование данных) и специально разработанный язык для распространения изменений по всей системе. Технология RxJava казалась очень соблазнительным выбором, поскольку предлагала единообразный API, отлично приспособленный для работы с разными источниками данных.

Со временем я пробовал применять реактивное программирование и в других ситуациях, где требовалась высокая масштабируемость и пропускная способность. Для реализации реактивных систем, безусловно, приходится прикладывать больше усилий. Но и выгода велика, в частности, более полное использование возможностей оборудования и, стало быть, экономия энергии. Чтобы по-настоящему оценить преимущества этой модели программирования, разработчик должен располагать относительно простым инструментарием. Мы полагаем, что Reactive Extensions – удачный компромисс между уровнем абстракции, сложностью и производительностью.

В этой книге рассматривается версия RxJava 1.1.6, если явно не оговорено противное. Хотя RxJava может работать с версиями Java, начиная с Java 6, почти во всех примерах применяется синтаксис лямбда-выражений, появившийся в Java 8. В некоторых примерах из главы 8, посвященной Android, продемонстрированы более многословные синтаксические конструкции без лямбда-выражений. Но все же мы не везде используем самый лаконичный синтаксис (например, ссылки на методы), стремясь сделать код понятнее там, где это имеет смысл.

О содержании книги

Книга написана так, что наибольшую пользу даст последовательное чтение от корки до корки. Но если столько времени у вас нет, то можете выбирать самые интересные для себя части. Если какое-то понятие было введено раньше, то в большинстве случаев мы даем на него обратную ссылку. Ниже приведен краткий обзор глав.

- В *главе 1* содержится очень краткое введение в основные идеи и понятия RxJava (*Бен*).
- В *главе 2* объясняется, как в вашем приложении может появиться библиотека RxJava и как с ней взаимодействовать. Здесь все довольно просто, но ясное понимание таких понятий, как горячий и холодный источник, очень важно для дальнейшего (*Томаш*).

- *Глава 3* – краткий экскурс в многочисленные операторы, имеющиеся в RxJava. Мы познакомимся с выразительными и мощными функциями, лежащими в основе этой библиотеки (*Томаш*).
- *Глава 4* носит более практический характер, здесь показано, как включать RxJava в различные места кода. Затрагивается также вопрос о конкурентности (*Томаш*).
- В более продвинутой *главе 5* объясняется, как реализовать реактивное приложение от начала до конца (*Томаш*).
- В *главе 6* рассказано о важной проблеме управления потоком и о том, как она решается в RxJava с помощью механизмов противодавления (*Томаш*).
- В главе 7 описаны методы автономного тестирования, сопровождения и отладки приложений на основе Rx (*Томаш*).
- В *главе 8* приведены избранные примеры приложений RxJava, особенно в распределенных системах (*Томаш*).
- *Глава 9* посвящена планам развития RxJava 2.x (*Бен*).

Ресурсы в Сети

Все камешковые диаграммы, встречающиеся в книге, взяты из официальной документации по RxJava (*https://github.com/ReactiveX/RxJava/wiki*), опубликованной на условиях лицензии Apache License Version 2.0.

Графические выделения

В книге применяются следующие графические выделения:

Курсив

Новые термины, URL-адреса, адреса электронной почты, имена и расширения имен файлов.

`Моноширинный`

Листинги программ, а также элементы кода в основном тексте: имена переменных и функций, базы данных, типы данных, переменные окружения, предложения и ключевые слова языка.

`Моноширинный полужирный`

Команды и иные строки, которые следует вводить буквально.

`Моноширинный курсив`

Текст, вместо которого следует подставить значения, заданные пользователем или определяемые контекстом.

Таким значком обозначается совет или рекомендация общего характера.

Таким значком обозначается замечание общего характера.

Таким значком обозначается предупреждение или предостережение.

Как с нами связаться

Вопросы и замечания по поводу этой книги отправляйте в издательство:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (в США и Канаде)
707-829-0515 (международный или местный)
707-829-0104 (факс)

Для этой книги имеется веб-страница, на которой публикуются списки замеченных ошибок, примеры и прочая дополнительная информация. Адрес страницы: *http://bit.ly/reactive-prog-with-rxjava*.

Замечания и вопросы технического характера следует отправлять по адресу *bookquestions@oreilly.com*.

Дополнительную информацию о наших книгах, конференциях и новостях вы можете найти на нашем сайте по адресу *http://www.oreilly.com*.

Читайте нас на Facebook: *http://facebook.com/oreilly*.

Следите за нашей лентой в Twitter: *http://twitter.com/oreillymedia*.

Смотрите нас на YouTube: *http://www.youtube.com/oreillymedia*.

Благодарности

От Бена

Этой книги не было бы без Томаша, который написал большую часть текста, и Нэн Барбер, нашего редактора, которая с достойным восхищения терпением помогала нам до самого конца работы. Спасибо Томашу, откликнувшемуся на мое объяв-

ление в Твиттере (*https://twitter.com/benjchristensen/status/632287727749230592*) о поиске автора, в результате чего книга стала реальностью!

Я также высоко ценю поддержку центра Netflix Open Source (*https://netflix.github.io/*) и Дэниэля Джекобсона (*https://twitter.com/daniel_jacobson*), оказанную мне лично и проекту в целом. Они были прекрасными спонсорами проекта, только благодаря им я мог уделять столько времени сообществу. Спасибо!

И еще я благодарен Эрику, который создал Rx, столь многому научил меня и согласился написать предисловие к книге.

От Томаша

Прежде всего, я хочу сказать спасибо родителям, которые купили мне мой первый компьютер почти 20 лет назад (это был 486DX2 с 8 МБ памяти, такое не забудешь). Так началось мое путешествие в мир программирования. Несколько людей внесли свой вклад в создание этой книги. И первый среди них – Бен, который согласился написать первую и последнюю главу, а также рецензировать мой текст.

И раз уж речь зашла о рецензентах, то Венкат Субраманиам (Venkat Subramaniam) немало постарался, чтобы придать книге ясную и логичную структуру. Нередко он предлагал поменять порядок предложений, абзацев и глав или даже удалить целые страницы, не имеющие отношения к делу. Еще одним рецензентом – весьма знающим и опытным – был Дэвид Карнок. Будучи руководителем проекта RxJava, он нашел десятки ошибок, состояний гонки, несогласованностей и других проблем. Оба рецензента написали сотни замечаний, которые заметно улучшили качество книги. На ранних этапах работы рукопись читали многие мои коллеги, поделившиеся своим мнением. Выражаю благодарность Дариушу Бачински, Шимону Хома, Петру Петжаку, Якубу Пилимону, Адаму Войцику, Марчину Зайончковски и Мачею Зярко.

Глава 1. Реактивное программирование с применением RxJava

RxJava – это конкретная реализация технологии реактивного программирования для Java и Android, на которую большое влияние оказало функциональное программирование. В RxJava отдается предпочтение композиции функций без глобального состояния и побочных эффектов, а также применению потоков для составления асинхронных событийно-ориентированных программ. За основу взят паттерн Наблюдатель, абстрагирующий обратные вызовы производителя и потребителя, и затем расширен десятками операторов, обеспечивающих композицию, преобразование, диспетчеризацию, дросселирование (throttling), обработку ошибок и управление жизненным циклом.

RxJava – зрелая библиотека с открытым исходным кодом (*https://github.com/ReactiveX/RxJava*), широко используемая как на серверах, так и на мобильных устройствах на платформе Android. Вокруг библиотеки RxJava и реактивного программирования в целом сложилось активное сообщество разработчиков (*http://reactivex.io/tutorials.html*), которые дополняют проект, выступают на конференциях, пишут статьи и помогают друг другу.

Эта глава содержит краткий обзор библиотеки RxJava – что это такое и как работает – а в остальной части книги детально описано, как ее использовать в приложениях. Для чтения книги необязательно иметь опыт реактивного программирования, мы начнем с самого начала и познакомим вас с идеями и практическим применением RxJava, чтобы вам было проще понять, поможет ли она в вашей конкретной ситуации.

Реактивное программирование и RxJava

Термином «реактивное программирование» обозначают технологию программирования, в которой акцент делается на реакции на изменения, например, на изменение значений данных или на события. Это можно сделать – и зачастую делается – императивно. Императивное реактивное программирование основано на обратных вызовах. Отличный пример реактивного программирования – элек-

тронные таблицы: если одна ячейка зависит от других, то она автоматически «реагирует» на изменение значений в них.

Функциональное реактивное программирование?

Хотя идеи функционального программирования оказали большое влияние на «реактивные расширения» (Reactive Extensions – Rx вообще и RxJava в частности), эту технологию нельзя назвать «функциональным реактивным программированием» (FRP). FRP – это весьма специфический частный случай реактивного программирования (*http://stackoverflow.com/questions/1028250/what-is-functional-reactive-programming/1030631#1030631*), в котором рассматривается непрерывное время, тогда как RxJava имеет дело только с дискретными событиями во времени. Я и сам впадал в это заблуждение в начале работы над RxJava, когда описывал библиотеку как «функциональную реактивную», пока не осознал, что это кажущееся таким естественным сочетание слов много лет назад уже было зарезервировано для чего-то совсем другого. Поэтому не существует никакого общепринятого термина, который описывал бы назначение RxJava более конкретно, чем «реактивное программирование». Акроним FRP все еще широко – и неправильно – используется для описания RxJava и других подобных библиотек, а в Интернете продолжают спорить, что правильнее: расширить значение термина (поскольку он неформально употребляется в этом смысле уже несколько лет) или сохранить его только за реализациями с непрерывным временем.

Устранив это недоразумение, мы можем сосредоточиться на том факте, что RxJava действительно создана под влиянием функционального программирования и в ее основу сознательно положена модель, отличная от императивного программирования. В этой главе слово «реактивный» обозначает реактивно-функциональный стиль, характерный для RxJava. Напротив, говоря «императивный», я не имею в виду, что реактивное программирование нельзя реализовать императивно, а лишь подчеркиваю, что императивное программирование противоположно функциональному подходу, принятому в RxJava. Сравнивая императивный и функциональный подходы, я буду употреблять термины «реактивно-функциональный» и «реактивно-императивный».

В современных компьютерах любой подход в какой-то момент оказывается императивным, потому что любая программа в конечном итоге опускается на уровень операционной системы и оборудования. Компьютеру необходимо явно сказать, что и как делать. Люди думают иначе, чем процессоры и основанные на них системы, поэтому мы добавляем уровни абстрагирования. Реактивно-функциональное программирование – это абстракция, как и высокоуровневые идиомы императивного программирования, абстрагирующие машинные команды. О том, что в конечном итоге любая программа императивна, не следует забывать, потому что это помогает выстроить умозрительную модель задачи, решаемой реактивно-функциональным программированием, и понять, как она в итоге выполняется, – никакой магии здесь нет.

Таким образом, реактивно-функциональное программирование – это такой подход к программированию (абстракция поверх императивных систем), кото-

рый позволяет писать асинхронные и событийно-ориентированные программы, не заставляя себя думать, как компьютер, и императивно описывать сложные взаимодействия состояний, особенно пересекающих границы потоков и сетей. Возможность не подражать «мышлению» компьютера – вещь полезная при разработке асинхронных и событийно-ориентированных систем, поскольку тут речь идет о вопросах конкурентности и параллелизма, где корректность и эффективность крайне важны, но трудно достижимы.

В сообществе Java эталонами глубины и широты охваты тематики, связанной со сложностями конкурентного программирования, считаются книги Brian Goetz «Java Concurrency in Practice» и Doug Lea «Concurrent Programming in Java» (Addison-Wesley), а также форумы типа «Mechanical Sympathy» (*https://groups.google.com/forum/m/#!forum/mechanical-sympathy*). Общаясь с экспертами, появляющимися на этих форумах, и вообще с членами сообщества в период начала работы над RxJava, я острее, чем когда-либо прежде, осознал, как трудно написать высокопроизводительную, эффективную, масштабируемую и вместе с тем корректную конкурентную программу. А мы ведь даже не упомянули распределенные системы, где конкурентность и параллелизм поднимаются на совсем другой уровень.

Таким образом, короткий ответ на вопрос, чем занимается реактивно-функциональное программирование, звучит так: конкурентностью и параллелизмом. А если прибегнуть к неформальной терминологии, то оно устраняет «ад обратных вызовов», являющийся неизбежным результатом императивного решения реактивных и асинхронных задач. Реактивное программирование в той форме, какая реализована в RxJava, основано на применении функционального программирования, в нем используется декларативный подход, позволяющий обойти типичные для реактивно-императивного стиля ловушки.

Когда возникает нужда в реактивном программировании

Реактивное программирование полезно в следующих ситуациях.

- Обработка событий, инициированных пользователем, например: перемещение мыши, щелчки мышью, ввод с клавиатуры, изменение сигналов GPS вследствие перемещения пользователя вместе со своим устройством, сигналы от встроенного гироскопа, события касания пальцем и т. д.
- Обработка любых событий ввода-вывода от диска или сети, характеризующихся наличием задержки. В этих случаях ввод-вывод по самой своей природе асинхронный (отправлен запрос, проходит время, затем получен – или не получен – ответ, запускающий дальнейшую обработку).
- Обработка событий или данных, поступающих приложению от производителя, которого оно не может контролировать (системные события сервера, вышеупомянутые пользовательские события, сигналы от оборудования, события аналоговых датчиков и т. д.).

Если программа обрабатывает только один поток событий, то реактивно-императивный стиль на основе обратных вызовов вполне может подойти, а написание реактивно-функциональной программы не принесет особой выгоды. Даже если различных потоков событий много сотен, но все они независимы, то и тогда императивное программирование годится. В таких простых случаях императивный подход оказывается наиболее эффективным, потому что отсутствует уровень абстракции реактивного программирования, т. е. программа оказывается ближе к тому уровню, для которого оптимизированы современные операционные системы, языки и компиляторы.

Но в большинстве программ приходится комбинировать события (или асинхронные ответы на вызовы функций или обращения в сеть), в них присутствует условная логика взаимодействия между событиями и необходимо обрабатывать ошибки, в т. ч. освобождать захваченные ресурсы. И вот тогда реактивно-императивный подход приводит к неприемлемому росту сложности, а достоинства реактивно-функционального программирования становятся очевидны. В результате я сформулировал для себя такую ненаучную точку зрения: начальные затраты на изучение реактивно-функционального программирования гораздо выше, но сложность результирующей программы гораздо ниже, чем в случае реактивно-императивного программирования.

Стало быть, одной фразой технологию Rx вообще и RxJava в частности можно описать так: «библиотека для разработки асинхронных событийно-ориентированных программ». Библиотека RxJava – это конкретная реализация принципов реактивного программирования, созданная под влиянием идей функционального программирования и программирования потоков данных. Есть различные подходы к «реактивности», и RxJava – лишь один из них. Разберемся, как она работает.

Как работает RxJava

В центре RxJava находится тип `Observable`, представляющий поток данных или событий. Он предназначен для проталкивания (push) (реактивность), но может использоваться и для вытягивания (pull) (интерактивность). Тип является ленивым (lazy), а не энергичным (eager). Допускает как синхронное, так и асинхронное использование. Может представлять 0, 1, много и даже бесконечно много значений или событий во времени.

В этом абзаце много технических терминов и деталей, нуждающихся в пояснении. Полное описание приведено в разделе «Анатомия rx.Observable» главы 2.

Проталкивание и вытягивание

Весь смысл реактивности RxJava в том, чтобы поддержать режим проталкивания, поэтому сигнатуры методов типа `Observable` и связанного с ним типа `Observer` поддерживают поступление входящих событий. Это естественно сопровождается поддержкой асинхронности, о чем пойдет речь в следующем разделе. Но тип `Observable` поддерживает также асинхронный канал обратной связи (ино-