Содержание

Введение

Платформа Erlang/OTP в наши дни выросла в замечательную и полезную систему. Отчасти причиной этого стало то, что её открытый исходный код продолжает привлекать внимание программистов со всего мира. Тот факт, что программисты из разных культур, с разной историей и опытом, получили возможность вносить в неё свои идеи, исправления и свой код, помогает увеличить широту возможностей и гибкость системы. В свою очередь это помогает всему сообществу, собравшемуся вокруг Erlang/OTP. Если вы ещё не знакомы с этим сообществом, то мы уверены, что вы сочтёте его одним из самых приятных, полезных и начитанных сообществ среди всех языков программирования.

Мы рады и гордимся тем, что наша книга теперь стала доступна русскоязычным энтузиастам Erlang. Мы надеемся, что детали и рекомендации, представленные в этой книге, помогут вам на вашем пути к становлению в роли эксперта по Erlang/OTP. Возможно вы станете активным и известным участником сообщества, а ваши идеи и код помогут новичкам разобраться с языком и только улучшат качество и полезность платформы Erlang/OTP.

Об авторах этой книги

Франческо Чезарини является основателем и техническим директором компании Erlang Solutions[1]. Он использовал Erlang ежедневно с 1995 года, с момента начала его карьеры практикантом в компьютерной научной лаборатории Ericsson (CSLab), в которой и появился Erlang. Затем он перешёл в отдел повышения квалификации и консультирования Ericsson, где работал над первой версией R1 платформы OTP, используя её для создания приложений и ключевых продуктов компании. В 1999 году, вскоре после того, как исходный код Erlang был открыт, он основал компанию, которая в наши дни известна под именем Erlang Solutions. С отделениями в семи странах и на трёх континентах другие компании предпочитают её в качестве партнёра по разработке масштабируемых высокодоступных решений.

В роли технического директора Франческо руководит командами разработчиков и консультантов в компании Erlang Solutions и отвечает за стратегии создания продуктов и исследований в компании. Также он является соавтором книги «Программирование на Erlang», опубликованной O'Reilly, перевод которой был выпущен издательством «ДМК Пресс». Более десяти лет Франческо преподавал в IT-университете Гётеборга, а с 2010 года вёл курс параллельного программирования в университете Оксфорда. Его можно найти в Твиттере под ником *@FrancescoC*.

Стив Виноски работает разработчиком в компании Arista Networks. Большую часть своей тридцатилетней карьеры он провёл, работая над межплатформенным ПО (*middleware*) и распределёнными вычислительными системами. В 2006 году, после 20 лет разработки таких систем на Java и C++, он открыл для себя Erlang и с тех пор использует его в качестве основного языка разработки. Стив участвовал во множестве известных проектов, таких как база данных Riak и веб-сервер Yaws. Также он внёс в исходный код Erlang/OTP десятки новых возможностей и исправлений ошибок.

Стив также, как сам, так и совместно с другими авторами, выпустил более сотни статей и публикаций на темы межплатформенного ПО, распределённых систем и веб-разработки, а также пару книг. Он вёл колонку «The Functional Web» («Функциональная интернет-паутина») в журнале «*IEEE Internet Computing*» с 2008 по 2012 год, а до того с 2002 года вёл в том же журнале колонку «Toward Integration» («Путь к интеграции»). Стив является членом редакторской команды журнала. Между 1995 и 2005 годом он был соавтором колонки «Object Interconnections» («Взаимосвязи объектов») для «*C++ Report*» и позднее для «*C/C++ Users Journal*». В течение прошедших лет Стив также принял участие в сотнях конференций, отдельных презентаций и уроков по межплатформенному ПО, распределённым системам, веб-разработке и языкам программирования, а также был участником программного комитета в десятках конференций и семинаров.

---

[1] http://www.erlang-solutions.com/.

Колофон

Животное на обложке «Проектирования масштабируемых систем с помощью Erlang/OTP» – это европейская морская камбала (*pleuronectes platessa*), известный вид камбалы. Морская камбала живёт вдоль береговой линии Европы от Средиземного моря до Баренцева моря. Обычно их можно найти на глубине от 10 до 50 метров, где они закапываются в песчаное или илистое дно.

Европейская морская камбала имеет тёмно-зелёную или коричневую чешую и покрыта оранжевыми пятнами. Взрослые особи могут достигать одного метра в длину, но большинство дорастает лишь до половины метра. Питается рыба морскими червями, двустворчатыми моллюсками и ракообразными.

Камбала является одним из основных продуктов северогерманской и датской кухни и обычно добывается рыбаками по всей Европе. Особенно она популярна в Дании в жареном виде с картофелем фри под соусом ремулад. Европейская морская камбала была под угрозой истребления в 1970–1980-х годах, но благодаря усилиям сбережения популяции её численность растёт и в 2012 году оценивалась на наивысшем уровне с 1957 года.

Множество животных на обложках книг издательства O'Reilly находится под угрозой, все они важны для нашего мира. Чтобы узнать подробнее о том, как вы можете им помочь, прочтите информацию на сайте animals.oreilly.com.

Предисловие

Эта книга является тем, что вы получите, если посадите вместе Erlang-энтузиаста, который работал над OTP версии R1 в 1996 году, и специалиста по распределенным системам, который спустя десять лет обнаружил то, как Erlang/OTP позволяет вам сосредоточиться на реальных сложностях развития систем, избегая случайных трудностей.

Описывая, как строятся поведения OTP и зачем они нужны, мы покажем вам, как использовать их для проектирования архитектуры автономных узлов. В нашем первоначальном предложении издательству O'Reilly здесь мы и остановились. Однако при написании книги мы решили поднять планку выше, задокументировав наши практики, проектные решения и распространённые ошибки при проектировании распределенных систем. Эти шаблоны посредством ряда проектных решений и компромиссов дают нам те масштабируемость, надёжность и доступность, которыми славится Erlang/OTP. Вопреки распространенному мнению, это не происходит волшебным образом или не появляется из коробки, но, конечно, это гораздо проще, чем с любым другим языком программирования, который не имитирует семантику Erlang и не выполняется на виртуальной машине BEAM.

Франческо: Почему появилась эта книга?

Кто-то однажды сказал мне, что написать книгу – это как завести детей. После того как вы написали одну книгу и уже держите в руках бумажную копию, волнение захлёстывает вас, и вы быстро забываете напряжённую работу и принесённые жертвы и хотите начать писать следующую. Я был намерен написать продолжение «*Программирования Erlang*» (издательство O'Reilly), начиная с того момента, как я получил в руки её первую бумажную копию в июне 2009 года. Я не имел тогда собственных детей, когда начал этот проект, но в конце концов проект оказался настолько длинным, что в момент выхода этой книги на подходе уже мой второй ребёнок. Кто сказал, что хорошие вещи не стоят того, чтобы их ждать?

Как и в случае с первой книгой, для «*Проектирования масштабируемых систем с помощью Erlang/OTP*» мы взяли за основу примеры из учебных материалов по OTP, разработанных компанией Erlang Solutions. Я использовал примеры и начал объяснять их, преобразовывая попутно мои лекции и подход к обучению в слова книги. Когда я закончил одну главу, я вернулся и перепроверил те части, с которыми у студентов нередко возникали трудности. Вопросы, которые часто задавались лучшими студентами, в конечном итоге оказались в боковых сносках, а длинные главы были разделены на более мелкие. Все шло хорошо, пока мы не добрались до глав 11 и 12, потому что не существовало стандартного лучшего способа работы с релизами и обновлениями ПО. Скорее, имелись инструменты, мно-

жество инструментов. Некоторые были интегрированы в цикл сборки и выпуска кода нашего клиента, другие же работали просто из коробки. Некоторые были непригодными для использования. Мы надеемся, что эти главы станут наилучшим руководством для тех, кто хочет понять, как релизы обновления ПО работают за кулисами. Они также объясняют, что вам нужно знать для поиска и устранения проблем в существующих инструментах или для написания своих собственных.

Но реальные проблемы начались с главы 13. Не имея примеров или учебных материалов, мне пришлось формализовать те знания, которые были в наших головах, и документировать подходы, которые мы предпринимаем при проектировании систем на Erlang/OTP, пытаясь привести их в соответствие с теорией распределенных вычислений. Глава 13 превратилась в четыре главы, на которые ушло столько же времени, сколько и на первые десять глав. Для тех из вас, кто купил версию в раннем доступе, я надеюсь, что ожидание того стоило. Те, кто мудро ждал окончания работы над книгой, прежде чем купить себе копию, – наслаждайтесь!

Стив: Почему появилась эта книга?

Я впервые открыл для себя Erlang/OTP в 2006 году, исследуя способы, как сделать разработку программного обеспечения интеграции предприятия быстрее, дешевле и лучше. Независимо от того, с какой стороны я смотрел на него, Erlang/OTP явно превосходил языки C++ и Java, которые я и мои коллеги давно использовали на то время. В 2007 году я присоединился к новой компании и начал использовать Erlang/OTP для создания коммерческого продукта, и оказалось, что Erlang исполнил всё, что о нём говорили мои предыдущие расследования. Я обучил языку некоторых моих коллег, и спустя некоторое время горстка разработчиков смогла продолжить разработку ПО, оказавшегося в итоге более функциональным, более надёжным, легче развивающимся и готовым к релизу гораздо быстрее, чем аналогичный код, сделанный значительно большей командой программистов на C++. И по сей день я по-прежнему полностью убежден во впечатляюще практической эффективности Erlang/OTP.

С годами я опубликовал немало технических материалов, и предполагаемой целевой аудиторией всегда были другие практикующие разработчики вроде меня. И эта книга не исключение. В первых 12 главах мы даём глубокую детализацию, которая нужна практикующим разработчикам, для того чтобы полностью понять основные принципы OTP. К этим деталям мы подмешиваем крупицы практических знаний – модули, функции и подходы, которые сэкономят вам значительное время и усилия в вашей повседневной работе по проектированию, разработке и отладке. В заключительных четырёх главах мы переводим дух и фокусируемся на общей картине тех компромиссов, которые встречаются в разработке, развертывании и эксплуатации устойчивых масштабируемых распределенных приложений. Из-за ошеломляющего количества знаний, подходов и компромиссов, на которые приходится идти в распределённых системах, отказоустойчивости и DevOps, написание этих глав в лаконичном стиле оказалось трудным, но я считаю, что нам

удалось нащупать верный баланс, предоставив много полезных советов и не потерявшись в несущественных деталях.

Я надеюсь, что эта книга поможет вам повысить качество и полезность программного обеспечения и систем, которые вы разрабатываете.

Кому следует прочесть эту книгу

Целевая аудитория этой книги включает разработчиков и архитекторов на языках Erlang и Elixir, которые прочитали, по крайней мере, одну вступительную книгу по языку и готовы поднять свои знания на следующий уровень. Это не та книга, с которой можно начать изучение, но скорее такая книга, которая подхватит вас с того места, где предыдущая книга закончилась. В главах с 3 по 12 материал основан на материале предыдущей главы, и эти главы следует читать последовательно, так же и главы 13–16. Если вам не нужны основы Erlang, то не стесняйтесь пропустить главу 2.

Как читать эту книгу

Мы писали эту книгу для Erlang версии 18.2. Большинство функций, которые мы описываем, будет работать и с более ранними выпусками, а те особенности, которые не совместимы, отмечены отдельно. В настоящее время неизвестно о несовместимостях с будущими релизами, если такие появятся, мы опишем их подробно на странице найденных ошибок на сайте книги и поправим код в github-репозитории книги. Вам предлагается скачать примеры к книге из нашего репозитория на github[1] и запускать их самостоятельно для лучшего понимания.

Благодарности

Написание этой книги оказалось долгим путём. Пока мы по нему шли, мы получили большую поддержку от множества замечательных людей. Наш редактор Энди Орам был бесконечным источником идей и предложений, терпеливо ведя нас, давая советы и оказывая постоянную поддержку. Спасибо, Энди, мы не смогли бы сделать это без тебя! Саймон Томпсон, соавтор *«Программирования Erlang»*, помог с идеей книги и заложил основу для второй главы. Большое спасибо Роберту Вирдингу за то, что подарил книге некоторые из примеров. Множество читателей, корректоров и добровольцев давали нам советы, по мере того как мы по каплям кормили их содержимым очередных глав.

Рискуя пропустить кого-то, вот эти люди: Ричард Бен Алея, Роберто Алой, Йеспер Луи Андерсен, Боб Баланс, Ева Бихари, Мартин Бодокки, Наталья Чечина, Жан-Франсуа Клутье, Ричард Кроучер, Виктория Фёрдош, Хайнц Гис, Йоаким Хален, Фред Хеберт, Чаба Хош, Торбен Хофманн, Боб Ипполито, Аман Коули, Ян Виллем Луитен, Джей Нельсон, Робби Рашке, Анджей Слива, Дэвид Смит,

[1] https://github.com/francescoc/scalabilitywitherlangotp.

Сэм Таваколи, Премананд Тангамани, Ян Улиг, Джон Уорвик, Дэвид Уэлтон, Ульф Вигер и Александр Йонг.

Если мы вас пропустили, то примите наши искренние извинения! Напишите нам, и вы будете оперативно добавлены. Отдельный привет передаём персоналу Erlang Solutions за чтение глав по мере их написания и всем остальным, кто представил замеченные ошибки в раннем выпуске. Отдельное спасибо всем тем, кто ободрял нас в социальных медиа, особенно другие авторы. Вы знаете, о ком мы! И наконец, но не в последнюю очередь, благодарности командам по производству, маркетингу и конференциям в издательстве O'Reilly, которые продолжали напоминать нам, что путь не закончен, пока вы не держите в руках бумажную копию. Мы действительно ценим вашу поддержку!

Соглашения, используемые по тексту этой книги

Следующие типографические соглашения используются в этой книге:

Курсив
: Указывает новые термины, приложения, URL-ссылки, адреса электронной почты, имена файлов, каталогов и расширения файлов.

`Моноширинный шрифт`
: Используется для листингов программ, а также внутри текста для обозначения элементов программ, таких как имена функций, базы данных, типы данных, переменные окружения, операторы и ключевые слова. Также используется для поведений, команд и параметров командной строки.

`Моноширинный жирный`
: Выделяет команды или прочий текст, который должен быть введён пользователем вручную.

`Моноширинный курсив`
: Выделяет текст, который следует заменить данными пользователя, требующимися по ситуации.


Этот значок указывает на подсказку или совет.


Этот значок означает некоторое замечание.


Этот значок ставится рядом с предупреждением или в опасном месте.

Пользовательский код

Дополнительные материалы (примеры кода, упражнения и т. д.) доступны для скачивания по адресу https://github.com/francescoc/scalabilitywitherlangotp.

Эта книга сделана для того, чтобы помочь вам выполнить вашу работу. В целом вы можете свободно пользоваться кодом из этой книги в ваших программах или документации. Не нужно спрашивать у нас разрешения, кроме тех случаев, когда вы копируете значительную часть кода. Например, для написания программы с использованием нескольких фрагментов кода из этой книги не требуется спрашивать разрешения. Продажа или распространение дисков с примерами из книг O'Reilly требует наличия разрешения. Ответ на чей-то вопрос с помощью цитаты или фрагмента кода из книги не требует разрешения. Копирование значительного количества примеров из этой книги в документацию вашей программы потребует разрешения.

Мы будем благодарны, хотя и не требуем указания нашего авторства. Указание авторства обычно включает название книги, автора, издателя и международный код ISBN. Например: *«Проектирование для масштабируемости с Erlang/OTP»* Франческо Чезарини и Стив Виноски (изд-во O'Reilly). © 2015 Франческо Чезарини и Стивен Виноски.

Если вы считаете, что использование вами примеров из книги выходит за рамки добросовестного использования, обратитесь к представителям издательства по адресу permissions@oreilly.com.

Safari® Books Online

Safari Books Online – это цифровая библиотека, которая позволяет легко выполнять поиск среди 7500 справочников и видеороликов по технологиям и творчеству. Здесь вы быстро сможете найти нужные ответы.

С подпиской вам станет доступна любая книга или видео из нашей онлайн-библиотеки. Читайте книги на вашем мобильном телефоне или планшете. Получайте ранний доступ к материалам до того, как они попадут в печать, а также эксклюзивный доступ к рукописям в процессе их написания с возможностью обратной связи с их авторами. Копируйте и вставляйте примеры кода, организуйте коллекцию избранного, скачивайте главы, ставьте закладки в ключевых секциях, создавайте заметки, печатайте страницы и наслаждайтесь другими возможностями, которые экономят ваше время.

Издательство O'Reilly Media разместило эту книгу в онлайн-сервисе Safari Books Online. Для полного цифрового доступа к этой книге и другим книгам на подобные темы от O'Reilly и других издателей создайте бесплатную учётную запись на сайте http://my.safaribooksonline.com.

Как с нами связаться

Пожалуйста, направляйте комментарии и вопросы по этой книге в адрес издателя:

O'Reilly Media, Inc.
1005 Gravenstein Highway
North Sebastopol, CA 95472
800-998-9938 (звонок из США или Канады)
либо 707-829-0515, 707-829-0104 (факс)

У нас есть веб-страница, посвящённая этой книге, где мы собираем найденные ошибки и любую дополнительную информацию. Вы можете посетить её по этому адресу: http://bit.ly/designing-for-scalability-with-erlangotp.

Чтобы оставить комментарий или задать технические вопросы по этой книге, отправьте письмо на bookquestions@oreilly.com.

Дополнительную информацию о наших книгах, учебных курсах, конференциях, новостях смотрите на сайте издательства: http://www.oreilly.com.

Наша страничка на Facebook: http://facebook.com/oreilly.

Подпишитесь на наш Twitter: http://twitter.com/oreillymedia.

Наш канал на YouTube: http://www.youtube.com/oreillymedia.

Глава 1

Знакомство

Итак, вам требуется создать устойчивую к сбоям, масштабируемую систему мягкого реального времени с требованиями к её высокой доступности. Она должна работать на событиях и реагировать на внешние раздражители, нагрузку и сбои. Она должна всегда продолжать отвечать на запросы. Вы где-то слышали, конечно же, множество историй успеха, которые утверждают, что Erlang является самым подходящим инструментом для такой работы. В самом деле так и есть – но хотя Erlang и является мощным языком программирования, его одного недостаточно, чтобы собрать все эти возможности вместе и строить сложные реактивные системы. Чтобы корректно, быстро и эффективно выполнить данную задачу, вам ещё понадобятся вспомогательное программное обеспечение (middleware), повторно используемые библиотеки, инструментарий, принципы проектирования и программная модель, которая расскажет вам, как построить вашу систему и сделать её распределённой.

В этой книге нашей целью было изучить многогранность доступности и масштабирования систем, а также затронуть близкие темы параллельного исполнения, распределения и устойчивости к сбоям с точки зрения языка программирования Erlang и его библиотеки OTP. Erlang/OTP был создан в лаборатории компьютерных наук компании Ericsson (CS Lab), и целью проекта по его созданию было исследование эффективной разработки нового на то время поколения телекоммуникационных систем в индустрии, где время до выхода продукта на рынок становилось всё критичнее. Это было до прихода Всемирной паутины, до планшетов и смартфонов, MMO-игр, систем обмена сообщениями и эпохи Интернета вещей (IoT).

В то время единственными системами, которые требовали таких уровней масштабируемости и устойчивости к сбоям, которые сегодня мы считаем чем-то само собой разумеющимся, были скучные телефонные коммутаторы. Им приходилось обслуживать массивные всплески объёма звонков в канун Нового года и Рождества, выполнять требования регулятора по обеспечению звонков экстренным службам и избегать болезненных штрафов, которые накладывались на поставщиков, чьё оборудование привело к отказу в обслуживании. С точки зрения простого горожанина, если вы подняли трубку и не услышали сигнала готовности телефонной станции, вы можете быть точно уверены в двух вещах: управляющие

телефонной компании получат серьёзные неприятности, а сбой в обслуживании попадёт на первые страницы утренних газет. Вне зависимости от того, что произошло, этим коммутаторам было запрещено ломаться. Даже когда их компоненты и инфраструктура вокруг них приходили в негодность, приходящие по другим линиям запросы обязаны были быть обслужены. Сегодня регуляторы и штрафы заменились нетерпеливыми пользователями без чувства лояльности, которые, не раздумывая, сменят провайдера, а заголовки на первых страницах газет заменились на массовую истерию в социальных медиа. Но основные проблемы доступности и масштабируемости остаются.

В результате этого коммутаторы связи и подобные им современные системы обязаны реагировать на сбои аналогично тому, как им приходится реагировать на нагрузку или внутренние события. И в то время, пока лаборатория компьютерных наук Ericsson не была готова изобрести новый язык программирования, найденное ими решение этой проблемы оказалось таким новым языком. Это прекрасный пример изобретения языка и модели программирования, которые предназначены для решения конкретного чётко определённого класса задач.

Определение проблемы

Как мы показываем по ходу этой книги, Erlang/OTP уникален среди языков программирования и библиотек и отличается шириной охвата, глубиной и согласованностью возможностей, которые он обеспечивает для масштабируемых систем, устойчивых к сбоям и имеющих требования высокой доступности. Проектирование, реализация, эксплуатация и обслуживание таких системам требуют отдачи всех сил. Команды, достигшие успеха в их построении и запуске, пришли к этому путём перебора этих четырёх фаз, используя подсказки от систем мониторинга и производственных метрик, чтобы найти области в коде, в разработке и эксплуатации, требующие улучшения. Успешные команды также учатся улучшать масштабируемость другими способами, такими как тестирование, экспериментирование и сравнение производительности, а ещё они нажимают на исследования и разработку нужных им характеристик системы. Нетехнические моменты, такие как корпоративные ценности и культура, также могут играть значительную роль в том, соответствуют ли команды требованиям их собственных проектов или даже превосходят эти требования.

Мы использовали термины *распределённый*, *устойчивый к сбоям*, *масштабируемый*, *мягкое реальное время* и *высокодоступный*, чтобы описать системы, которые мы планируем построить с помощью OTP. Но что эти слова обозначают на самом деле?

Масштабируемый ссылается на то, как хорошо система способна адаптироваться к изменениям нагрузки или количества доступных ресурсов. Масштабируемые сайты, к примеру, способны смягчить и обслужить всплески посещений, не теряя запросов клиентов, даже при сбоях в оборудовании. Масштабируемая система чата может быть способна принять тысячи новых пользователей в день, при этом не ухудшив качества обслуживания текущих клиентов.

Термин *распределённый* ссылается на то, как системы группируются вместе и взаимодействуют друг с другом. Кластеры могут проектироваться для горизонтального роста, с добавлением оборудования, доступного в массовой продаже (другими словами, обычного), или когда запускаются дополнительные копии системы на той же или других машинах для лучшего использования доступных ядер процессора. Одиночные машины также могут быть виртуализованы, так что копии операционной системы будут работать поверх другой операционной системы или делить друг с другом доступ к ресурсам оборудования. Добавление вычислительной мощности в кластер баз данных могло бы позволить масштабировать его по количеству данных, которое он сможет хранить, или по количеству запросов в секунду, которые он сможет обработать. Масштабирование в сторону уменьшения часто тоже важно; например, веб-приложение, построенное на облачных технологиях, может задействовать дополнительные мощности в пиковые часы и освободить неиспользуемые серверы, как только волна посетителей спадёт.

Системы, являющиеся *устойчивыми к сбоям*, продолжают вести себя предсказуемо в то время, когда компоненты в их окружении отказывают. Устойчивость к сбоям должна быть заложена в систему с самого начала; даже не думайте добавлять её в конце разработки. А что, если в ваш код вкралась ошибка или состояние программы оказалось повреждено? Или что будет, если ваша сеть отключится или произойдёт аппаратный сбой? Если пользователь посылает сообщение, которое приводит к аварийной остановке процесса, он может быть уведомлён о том, было доставлено сообщение или нет, и может быть уверен, что это уведомление соответствует действительности.

Под *мягким реальным временем* следует понимать предсказуемость ответов и задержек, обслуживание с постоянной пропускной способностью и гарантирование ответа в течение приемлемого времени. Пропускная способность должна оставаться стабильной, несмотря на всплески трафика или количества пришедших одновременно запросов. Не важно, сколько запросов проходит через систему в единицу времени, её пропускная способность не должна падать ниже проектной при тяжёлой нагрузке. Её время ответа, также известное как задержка, должно быть пропорционально количеству параллельно входящих запросов, избегая при этом разброса результатов, возникающего по причине сборки мусора «с остановкой мира» или других узких мест последовательного исполнения. Если пропускная способность вашей системы, например, равняется миллиону сообщений в секунду и внезапно придёт миллион сообщений одновременно, то время обработки должно оказаться равным одной секунде, после чего все получат результат. Но если в момент всплеска придут два миллиона запросов, то не допускается снижение пропускной способности; не некоторые, но все запросы без исключения должны быть обработаны в течение следующих двух секунд.

Высокая доступность минимизирует или совершенно исключает остановку сервиса в результате программных ошибок, обновлений кода или другой операционной деятельности. А что, если процесс упадёт? Что, если система питания вашего центра обработки данных откажет? Есть ли у вас запасной блок питания

или система батарей, дающая вам достаточное время, чтобы перенести ваш кластер или штатно завершить работу серверов, оказавшихся в этой ситуации? Или резервная сеть и резервное оборудование?

Измерили ли вы свою систему и убедились ли, что даже после потери части кластера оставшееся оборудование имеет достаточно ёмкости CPU, чтобы выдержать пиковую нагрузку? Не важно, потеряете ли вы часть вашей инфраструктуры, либо ваш облачный провайдер переживёт стеснительный момент отказа в сервисе, или вы будете заняты работами по обслуживанию, но пользователь, пославший сообщение в чат, желает быть уверенным, что сообщение дойдёт до правильного получателя. Пользователи системы ожидают, что она просто будет работать. Это отличается от *устойчивости к сбоям*, когда пользователю сообщают о том, что отправка не сработала, но вся система при этом не прекращает работу, и ошибка на неё не влияет. Способность Erlang выполнять обновления кода во время выполнения этому помогает. Но если вы задумаетесь о том, что требуется для изменения схемы базы данных или обновления протокола на обратно несовместимый в потенциально распределённом окружении, в то время как на систему продолжают приходить запросы, то простота испаряется. Когда вы работаете со своим онлайн-банком ночью или на выходных, вы хотите уверенности, что не появится это ненавистное сообщение – «закрыто на техническое обслуживание».

Erlang в самом деле способствует решению многих из перечисленных проблем. Но в сухом остатке это всего лишь язык программирования. Для сложных систем, которые вы собираетесь реализовать, вам понадобятся готовые приложения и библиотеки, которые можно использовать сразу из коробки. Вам также понадобятся принципы и шаблоны проектирования, которые наделяют архитектуру вашей системы новой целью – создавать распределённые надёжные кластеры. Вам будут нужны рекомендации по проектированию вашей системы, а также инструменты для реализации, установки, мониторинга, эксплуатации и обслуживания. В этой книге мы постараемся покрыть библиотеки и инструменты, которые позволяют изолировать сбой на уровне узла системы, создать и распределить множество узлов для масштабируемости и высокой доступности.

Вам следует серьёзно задуматься о ваших требованиях и свойствах и постараться выбрать подходящие библиотеки и шаблоны проектирования, которые гарантируют то, что готовая система будет себя вести так, как вам нужно, и будет делать то, что было задумано. В ваших поисках вам придется принять компромиссные решения, которые являются взаимозависимыми, – компромиссы по времени, ресурсам и поддержке функций, компромиссы по доступности, масштабируемости и надежности. Никакая готовая библиотека не сможет вам помочь, если вы сами не знаете, что должна делать ваша система. В этой книге мы проводим вас по шагам к пониманию этих требований и к принятию проектных решений и компромиссов, которые потребуются для их достижения.

OTP

OTP – это набор библиотек, принципов и шаблонов общего назначения, которые направляют и поддерживают структуру, проектирование, реализацию и установку Erlang-систем. Использование OTP в ваших проектах поможет вам избежать случайного увеличения сложности: ненужных усложнений по причине выбора неподходящих инструментов. Но другие проблемы остаются сложными, вне зависимости от выбора инструментов программирования и вспомогательного ПО.

Компания Ericsson очень рано осознала этот факт. В 1993 году одновременно с разработкой первого продукта с использованием Erlang компания Ericsson запустила второй проект для работы над инструментами, вспомогательным ПО и принципами проектирования. Разработчики хотели избежать повторения случайных осложнений, которые были решены ранее, и вместо этого сосредоточить свои усилия на более сложных проблемах. Результатом явилась BOS – Базовая операционная система. В 1995 году BOS объединилась с проектом по разработке Erlang, два проекта оказались под одной крышей, получили имя Erlang/OTP и носят его по сей день. Вы, возможно, слышали о команде мечты, которая занимается поддержкой Erlang, ещё её называют «команда OTP». Эта группа появилась в результате слияния проектов, когда Erlang вышел из исследовательской организации и была сформирована группа разработки продукта с целью его дальнейшего развития и решения возникающих вопросов.

Распространение знаний про OTP может ускорить признание Erlang в корпорациях, которые осторожно предпочитают только «испытанные и верные» решения. Даже простое знание о том, что существует стабильная и зрелая платформа для разработки приложений, помогает технологам «продать» Erlang своему менеджменту. Это – важный шаг к широкому признанию Erlang в индустрии. Стартапы, с другой стороны, просто берут его на вооружение, и Erlang/OTP позволяет им достичь скорости выхода на рынок и снизить затраты на их разработку и эксплуатацию.

Говорят, что OTP состоит из трёх составных частей (рис. 1.1), которые, если их использовать вместе, дают гарантированно надёжный подход к разработке систем в описанной выше сфере деятельности. Это собственно Erlang, инструменты и библиотеки и набор принципов проектирования. Посмотрим на них по порядку.

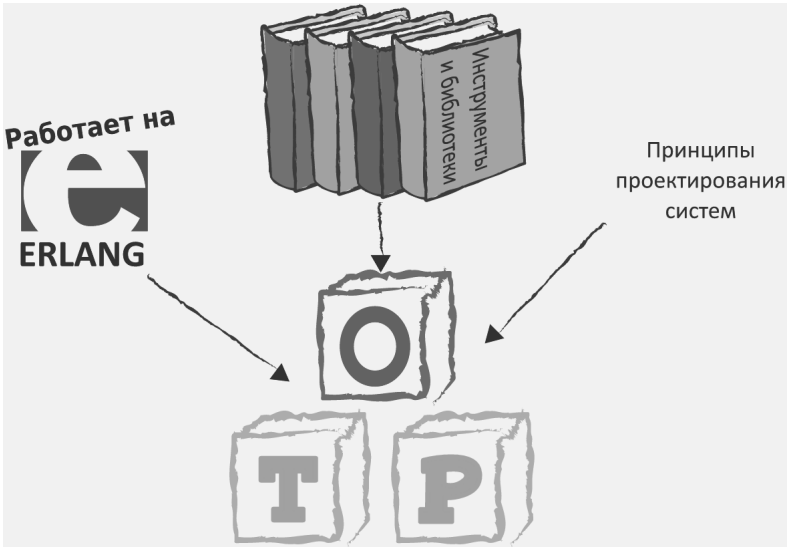

Рис. 1.1 ❖ Компоненты OTP

Что в имени твоём?

Что означает аббревиатура OTP? Лучше бы мы вам этого не рассказывали. Если вы поищете текст«OTP song» (песня OTP), то можете подумать, что сокращение означает «Одна истинная пара» (One True Pair). Или если дать волю воображению и попробовать угадать: «О, это идеально» (Oh This is Perfect), «На телефоне» (On The Phone) или «Открытая платформа транзакций» (Open Transaction Platform). Некоторые могут подумать, что OTP означает «онлайн-обработку транзакций» (Online Transaction Processing), но именно этот термин обычно сокращается как OLTP. Также были сделаны и более политкорректные предложения, когда хипстеры ввязались в попытку сделать Erlang более классным и выпустить продолжение фильма «Erlang the Movie». Увы, ни одна из расшифровок не является правильной. OTP – это сокращение от «Открытая телекоммуникационная платформа» (Open Telecom Platform), было придумано Бьярне Дэккером, бывшим главой лаборатории компьютерных наук (в которой родился язык Erlang) в компании Ericsson.

«Открытый» было модным словом в Ericsson в середине 90-х. Все должно было быть открытым: открытые системы, открытое оборудование, открытые платформы. Маркетологи Ericsson зашли так далеко, что даже печатали плакаты открытых ландшафтов и вешали их в коридорах с подписью «Открытые системы». Никто на самом деле не понимал, что имелось в виду под «открытыми системами» (или любой другой открытостью), но это было модным словечком, и зачем разочаровывать людей и не использовать открывшуюся возможность и (хоть раз) последовать моде? Как результат слово «открытый» в аббревиатуру OTP попало практически само собой.

Сегодня мы утверждаем, что слово «открытый» означает открытость Erlang по отношению к другим языкам, API и протоколам, большая разница с «открытостью» тех дней, когда он впервые был выпущен. Версия R1 библиотеки OTP на самом деле была чем угодно, только не открытой. Сегодня об открытости можно думать в плане существования JInterface, ei и erl_interface, а также протоколов HTTP, TCP/IP, UDP/IP, IDL, ASN.1, CORBA, SNMP и других ориентированных на интеграцию библиотек поддержки, предлагаемых Erlang/OTP.

Слово «телеком» было выбрано в то время, когда Erlang использовался только внутри Ericsson для телекоммуникационных продуктов, задолго до того, как ПО с открытым исходным кодом начало изменять мир. Возможно, это имело смысл в середине девяностых, но с тех пор никто не проводил ребрендинга, так что сегодня можно утверждать, что «телекоммуникационная» в названии ссылается на распределённость, устойчивость к сбоям, масштабируемость и мягкое реальное время с требованием высокой доступности. Эти характеристики имеются в телекоммуникационных системах, но в равной мере применимы и к широкому спектру других областей деятельности. Разработчики OTP решали проблемы телекоммуникационных систем, которые стали актуальными для остальной индустрии разработки ПО только с изобретением и массовым внедрением Интернета, теперь все новые проекты должны проектироваться под нагрузки масштаба Интернета. Erlang был готов к нагрузкам масштаба Интернета ещё до появления самого массового Интернета.

Последнее слово в OTP, «платформа», хоть и является скучным, но по-настоящему верно описывает суть вспомогательной библиотеки OTP (middleware). Оно было выбрано в то время, когда менеджмент компании Ericsson перегибал палку, разрабатывая множество различных платформ. Всё, что касалось программного обеспечения, должно было разрабатываться с использованием какой-нибудь (желательно открытой) платформы.

И в самом деле, Бьярне выбрал аббревиатуру, которая имела смысл и радовала высокопоставленный менеджмент, таким образом укрепляя шансы на продолжение финансирования проекта. Они, возможно, не понимали, над чем работает наша CS Lab, и не представляли, какие проблемы мы вот-вот могли устроить, но, по крайней мере, они были удовлетворены и позволили всему произойти.

С момента открытия исходного Erlang/OTP в 1998 году произошёл ряд дискуссий касательно ребрендинга, но ни одна из них не стала решающей. Поначалу разработчики за пределами индустрии телекоммуникаций по ошибке избегали использования OTP, поскольку, по их словам, «они не разрабатывали приложение для телекома». В наши дни сообщество и сама компания Ericsson утвердились в том, что OTP следует использовать, меньше обращая внимание на слово «телекоммуникационный», но акцентируя его важность. Это выглядит приемлемым компромиссом. Эта вставка в текст книги – единственное место, где слово «телеком» будет упомянуто в качестве части OTP.

Erlang

Первым строительным блоком является сам Erlang, что включает семантику языка и находящуюся в основе его работы виртуальную машину. Ключевые возможности языка, такие как легковесные процессы, отсутствие общей памяти и асинхронный обмен сообщениями, придвигают вас на шаг ближе к вашей цели. Не

менее важны связанные процессы и мониторинг процессов, которые к тому же являются выделенными каналами для распространения сигналов ошибок. Мониторы и сообщения об ошибках позволяют вам с относительной лёгкостью строить сложные иерархии процессов-наблюдателей со встроенным восстановлением после ошибок. Поскольку обмен сообщениями и распространение ошибок являются асинхронными, семантика и логика систем, разработанных для работы на единственном узле Erlang, подходят и для распределённых систем, не требуя никаких изменений кода.

Существенная разница между исполнением программы на одной машине или в распределённом окружении – это задержка, с которой доставляются сообщения и ошибки. Но в системах мягкого реального времени следует учитывать задержки независимо от того, является ли система распределённой и находится ли под нагрузкой. Итак, если вы решили одну из граней этой проблемы, вторая окажется решена автоматически.

Erlang позволяет вам исполнять код поверх виртуальной машины, которая сильно оптимизирована для параллельности, со сборкой мусора отдельно для каждого процесса, что даёт предсказуемое и простое поведение системы. Другие программные окружения не имеют такой роскоши, поскольку им требуется дополнительный слой абстракций для эмуляции модели параллельности Erlang и семантики ошибок. Цитируя Джо Армстронга, одного из изобретателей Erlang: «Вы можете эмулировать логику работы Erlang, но если ваша программа не исполняется на виртуальной машине Erlang, вы не сможете эмулировать семантику». Лишь несколько языков, которые сегодня успешно справляются и с логикой, и с семантикой, работают поверх эмулятора BEAM – виртуальной машины Erlang. Они составили целую экосистему, в которой Elixir и Lisp Flavoured Erlang (Erlang со вкусом Lisp) имеют самую большую динамику развития на момент написания этой книги. Всё, что написано в этой книге об Erlang, также в равной степени подходит и к этим языкам.

Инструменты и библиотеки

Второй строительный блок, который появился до того, как ПО с открытым исходным кодом стало повседневной нормой для программных проектов, включает ряд приложений, которые идут в стандартной поставке Erlang/OTP. Каждое приложение можно рассматривать как способ упаковки ресурсов в OTP, где одни приложения могут иметь зависимости от других приложений. Приложения включают в себя инструменты, библиотеки, интерфейсы к другим языкам и программным окружениям, базы данных и драйверы к базам данных, стандартные компоненты и стеки протоколов. Документация OTP разделяет их на такие подмножества.

Базовые приложения включают в себя следующее:

- система времени выполнения Erlang (*erts*);
- ядро (*kernel*);
- стандартные библиотеки (*stdlib*);
- библиотеки поддержки системной архитектуры (*sasl*).

Они предоставляют инструменты и базовые строительные блоки, которые требуются для проектирования, создания, запуска и обновления кода вашей системы. Мы рассмотрим базовые приложения подробно по ходу чтения этой книги. Вместе с компилятором они формируют минимальное подмножество приложений, необходимых для того, чтобы любая система, написанная на Erlang/OTP, смогла сделать хоть что-то осмысленное.

Приложения поддержки баз данных включают *mnesia*, распределённую базу данных Erlang, и *odbc* – интерфейс, предназначенный для связи с реляционными SQL базами данных. Mnesia является популярным выбором, поскольку она быстрая, выполняется и хранит данные в одной памяти с вашим приложением, ею легко пользоваться, и работать с ней можно прямо из Erlang.

Приложения для операций и обслуживания (OAM) включают в себя *os_mon*, приложение, позволяющее выполнять мониторинг операционной системы, *smnp* – агент и клиент для протокола SNMP и *otp_mibs*, информационные базы, позволяющие управлять Erlang-системами по протоколу SNMP.

Коллекция интерфейсных и коммуникационных приложений предоставляет стеки протоколов и интерфейсы для работы с другими языками программирования, включая компилятор и поддержку времени выполнения для ASN.1 (*asn1*), прямую связь с программами на языках C (*ei* и *erl_interface*) и Java (*jinterface*), а также библиотеку для разбора XML (*xmerl*). Есть и приложения для безопасности SSL/TLS, SSH, криптографии и инфраструктуры открытого ключа. Графические пакеты включают в себя порт библиотеки wxWidgets (*wx*) и лёгкий в использовании интерфейс. Приложение *eldap* предоставляет клиентский интерфейс к протоколу работы с каталогами учётных записей LDAP. И для любителей телекомов в наличии имеется стек поддержки diameter (описан в RFC-6733), который используется для авторизации и контроля политик, а также учёта. Копните немного глубже, и на свет появится стек поддержки Megaco. Megaco, также известный как H.248, – это протокол управления элементами физически разделённого мультимедийного портала (multimedia gateway), отделяющий операцию смены формата медиаматериалов от управления вызовами. Если вы когда-нибудь пользовались смартфоном, то вполне вероятно, что ваши действия разбудили приложения *diameter* или *megaco*, написанные на Erlang, где-нибудь на сервере провайдера.

Коллекция инструментальных приложений, которые поддерживают разработку, установку и эксплуатацию вашей Erlang-системы. В этой книге мы описываем только самые полезные, но назовём их все, чтобы вы были в курсе их существования:

- отладчик (*debugger*) – это графический инструмент, позволяющий пошагово пройти по вашему коду, и при этом можно повлиять на состояние функций;
- наблюдатель (*observer*) – интегрирует монитор приложений и диспетчер процессов вместе с базовыми инструментами мониторинга Erlang-систем как во время их разработки, так и на производственной системе;

- *Dialyzer* – это инструмент статического анализа, который обнаруживает несовпадения типов данных, мёртвый код и другие проблемы;
- трассировщик событий (*et*) – использует порты для сбора событий трассировки в распределённых окружениях, а *percept* позволяет обнаружить узкие места системы с помощью трассировки и визуализации параллельно происходящей активности;
- синтаксические инструменты Erlang (*syntax_tools*) – содержат модули для обработки деревьев синтаксиса языка Erlang способом, совместимым с другими языковыми инструментами. Она также включает в себя инструмент слияния модулей (module merger), позволяющий склеивать Erlang-модули, а также переименователь (renamer), решающий вопрос конфликтов имён в модулях;
- приложение *parse_tools* – содержит генератор исходных кодов синтаксического (*yecc*) и лексического анализаторов для Erlang (*leex*);
- *Reltool* – это инструмент управления релизами, имеющий графический интерфейс и методы вызова его из других систем сборки проектов, может вызываться из более обобщённых систем сборки;
- *Runtime_tools* – это коллекция утилит, в которую входят DTrace, зонды SystemTap и *dbg* – дружественная обёртка вокруг встроенных функций трассировки;
- и наконец, приложение *tools* является коллекцией, в которую входят профайлеры, анализаторы покрытия кода, инструменты проверки межмодульных зависимостей, а также поддержка режима Erlang в редакторе Emacs.

Тестовые приложения предоставляют инструменты модульного тестирования (*eunit*), системного тестирования и метод чёрного ящика. Тестовый сервер (упакован в приложение с названием *test_server*) может быть использован в качестве движка для тестового приложения более высокого уровня. Вполне возможно, он вам никогда не пригодится, поскольку OTP предоставляет другие инструменты для высокоуровневого тестирования в виде приложения *common_test* для тестирования чёрных ящиков. *Common_test* поддерживает автоматическое исполнение Erlang-тестов для большинства целевых систем независимо от их языков программирования.

Следует упомянуть объектные брокеры запросов (ORB) и интерфейсные приложения (IDL) по ностальгическим причинам и напомнить одному из соавторов о его прошлых грехах. В эту группу входят брокер под названием *orber*, компилятор языка IDL под названием *ic* и несколько сервисов, работающих по технологии CORBA, которые в наши дни никому уже не нужны.

Мы опишем и вспомним некоторые из перечисленных приложений и инструментов далее в этой книге. Некоторые из инструментов, которые мы опустим, были описаны в книге «*Программирование в Erlang*», изданной издательством «ДМК Пресс» (оригинал «Erlang Programming» издан O'Reilly), а те, что не попали даже в указанную книгу, описаны на страницах документации и руководства пользователя, идущих в стандартной поставке Erlang/OTP.

Не только эти приложения входят в инструментальную поддержку Erlang, их дополняют тысячи других приложений, разработанных сообществом и доступных с открытым исходным кодом. Мы опишем некоторые из самых популярных приложений во второй половине книги, там, где мы сосредоточимся на распределённых приложениях, доступности, масштабируемости и мониторинге. В этот список попадут Riak Core[1] и библиотека масштабируемого Erlang[2] (SD Erlang), приложения, помогающие с регулировкой нагрузки, такие как *jobs* и *safety-valve*, а также приложения для журналирования и мониторинга *elarm*, *folsom*, *exometer* и *lager*. По прочтении этой книги и до того, как вы начнёте ваш проект, пересмотрите стандартную документацию Erlang/OTP и руководство пользователя, поскольку никогда заранее неизвестно, вдруг что-то из них пригодится.

Принципы проектирования систем

Третий строительный блок OTP состоит из набора абстрактных принципов, правил проектирования и обобщённых поведений. Абстрактные принципы описывают программную архитектуру Erlang-системы, используя процессы в виде обобщённых поведений в качестве основных ингредиентов. Правила проектирования гарантируют, что ваша система будет оставаться совместимой с вашими инструментами. Этот подход обеспечивает стандартный путь решения проблем, делая код более лёгким для понимания и обслуживания, а также обеспечивая общий язык между командами разработчиков.

Обобщённые поведения OTP можно рассматривать как формализацию параллельных шаблонов проектирования. Поведения упакованы в библиотечные модули, содержащие универсальный код, решающий часто встречающуюся проблему. В них встроена поддержка отладки, обновления вашего кода, универсальная обработка ошибок и встроенная функциональность обновления всей системы.

Поведения могут быть *рабочими процессами*, которые выполняют всю трудную работу, или *процессами-наблюдателями*, чьей единственной задачей являются запуск, остановка и мониторинг других рабочих процессов и наблюдателей. Поскольку наблюдатели могут вести мониторинг других наблюдателей, функциональность внутри приложения может быть выстроена в цепочку, и так её станет легче разделить на отдельные модули. Процессы, которые находятся под присмотром наблюдателя, называются его *дочерними процессами* (children).

OTP предоставляет готовые библиотеки для рабочих процессов и наблюдателей, позволяя вам сосредоточиться на бизнес-логике вашей системы. Мы структурируем процессы в иерархические *деревья наблюдения*, получая устойчивые к сбоям структуры, которые изолируют ошибки и помогают восстановлению после них. OTP позволяет упаковать дерево наблюдения в приложение, как показано на рис. 1.2, двойные круги являются наблюдателями, а другие – рабочими процессами.

---

[1] https://github.com/basho/riak_core.

[2] http://www.dcs.gla.ac.uk/research/sd-erlang/.

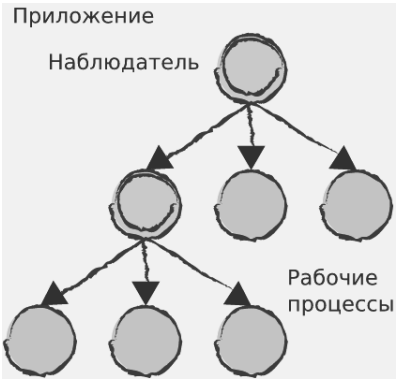

Рис. 1.2 ❖ Приложение OTP

Обобщённые поведения, идущие в составе вспомогательных библиотек OTP, включают в себя:

- обобщённые серверы, реализующие шаблон проектирования «клиент-сервер»;
- обобщённые конечные автоматы, позволяющие реализовать машины состояний;
- обработчики событий и диспетчеры, позволяющие универсально подойти к работе с потоками событий;
- наблюдатели, выполняющие мониторинг других наблюдателей и рабочих процессов;
- приложения, объединяющие под своим началом разные ресурсы, включая деревья наблюдения.

В этой книге мы рассмотрим их подробно, а также объясним, как реализовать собственные. Мы используем поведения для создания деревьев наблюдения, которые будут упакованы в приложение. Затем мы сгруппируем приложения вместе и сформируем *релиз*. Релиз описывает то, что исполняется на Erlang-узле.

Erlang-узлы

Erlang-узел состоит из нескольких слабо связанных между собой приложений, возможно, из некоторых стандартных приложений, описанных ранее в теме «Инструменты и библиотеки» на стр. 27 вместе с приложениями третьих сторон, а также приложениями, написанными вами самими для этого проекта. Эти приложения могут быть независимыми друг от друга или полагаться на службы и API друг друга. Рисунок 1.3 иллюстрирует типичный релиз узла Erlang с виртуальной машиной, выбранной под имеющееся оборудование и операционную систему, Erlang-приложениями, которые выполняются на виртуальной машине и взаимодействуют с другими компонентами, зависящими от ОС и оборудования.

Рис. 1.3 ❖ Узел Erlang

Сгруппируйте кластер из Erlang-узлов, потенциально можете добавить и другие узлы, вероятно, написанные на других языках, и у вас получится распределённая система. Теперь вы можете масштабировать вашу систему до тех пор, пока не упрётесь в некоторые физические ограничения. Они могут зависеть от способа, которым вы организовали доступ к вашим данным, или от ограничений оборудования и вашей сети, или других внешних причин, действующих как узкие места.

Распределение, инфраструктура и многоядерные процессоры

Устойчивость к сбоям – одно из фундаментальных требований к Erlang, пришедшее из его телекоммуникационного прошлого, – и распределение является его основной движущей силой. Без распределения, надёжности и доступности приложение, выполняющееся только на одном сервере, сильно зависит от надёжности самого сервера, его оборудования и ПО. Любые проблемы с процессором сервера, дисками, периферийными устройствами, блоком питания или прочей электроникой могут легко обрушить весь компьютер и вместе с ним ваше приложение. Подобным образом проблемы в операционной системе сервера или в библиотеках поддержки могут обрушить приложение или сделать его недоступным. Достижение устойчивости к сбоям требует наличия нескольких машин с некоторой степенью координации между ними, и средства распределения языка дают возможность это организовать.

В течение десятилетий компьютерная индустрия искала пути реализации распределения в языках программирования. Проектирование языков общего назначения само по себе достаточно сложно, а проектирование их для поддержки распределения существенно эту сложность увеличивает. По этой причине общим подходом к добавлению поддержки распределения к обычным языкам програм-

мирования является добавление библиотек. Этот подход, с одной стороны, хорош, поскольку поддержка распределения может эволюционировать отдельно от самого языка, но часто страдает от расхождения с принципами самого языка, то есть, как говорят разработчики, «неродное», «было прибито гвоздями». Поскольку почти все языки используют вызовы функций в качестве основного способа передачи управления и данных внутри приложения, то и дополнительные библиотеки распределения моделируют обмен данными между удалёнными узлами с помощью вызовов функций. Хотя это удобно, этот подход фундаментально плох, поскольку семантика локальных и удалённых вызовов, а особенно ситуации с ошибками в них, существенно друг от друга отличаются.

В Erlang процессы связываются друг с другом с помощью асинхронного обмена сообщениями. Это работает даже в том случае, если процесс находится на удалённом узле, поскольку виртуальная машина Erlang поддерживает также и обмен сообщениями между узлами. Когда один узел присоединяется к другому, он также узнаёт об остальных узлах, которые были известны другому. Таким образом, все узлы в кластере формируют полный граф, позволяя процессу послать сообщение любому другому процессу на любом узле кластера. Каждый узел в кластере также автоматически отслеживает, живы ли все другие узлы, чтобы как можно раньше узнавать о потере связи с ними. Преимущества асинхронной передачи сообщений в системах, которые выполняются на одном узле, распространяются и на системы, работающие в кластерах, поскольку ответы могут быть приняты вместе с ошибками и тайм-аутами.

Обмен сообщениями в Erlang и примитивные операции кластеризации служат основой для большого количества архитектур распределённых систем. Например, Erlang естественным образом подходит для сервис-ориентированной архитектуры (SOA), особенно её более современному варианту – микросервисам, учитывая легкость разработки и развертывания отдельных серверных процессов. С точки зрения клиентов, эти процессы являются службами, и связаться с ними можно посредством обмена сообщениями. В качестве другого примера учтите, что Erlang-кластеры не требуют наличия ведущих узлов (мастер), и это означает, что неплохо использовать их в качестве децентрализованных систем одинаковых узлов. Клиенты могут отправлять сообщения с запросом обслуживания любому узлу в кластере, и он обработает запрос сам либо передаст его тому узлу, который готов это сделать. Концепция отдельных кластеров, ещё известных как *«группы»*, которые общаются друг с другом посредством отдельных граничных узлов, и эти узлы могут иногда останавливаться, запускаться или терять сеть, реализована в библиотеке под названием SD Erlang. Другая популярная распределенная платформа, вдохновленная работой Amazon «Dynamo paper»[1], опубликованной в 2007 году, – это Riak Core, предлагающая устойчивое хэширование для планирования заданий, восстановления при разделении сети и потере узлов, с помощью отложенной согласованности данных и виртуальных узлов, которые делят состояние и данные на

[1] http://bit.ly/riak-dynamo.

небольшие управляемые сущности, которые могут быть реплицированы и перемещены между узлами.

С распределёнными системами вы также можете достигнуть и масштабирования. На самом деле доступность, согласованность и масштабирование идут рука об руку, влияя друг на друга. Всё начинается с модели параллельного исполнения и концепции передачи сообщения внутри узла, которую мы затем расширяем на всю сеть и начинаем использовать для кластеризации узлов. Виртуальная машина Erlang пользуется преимуществами современных многоядерных систем, позволяя процессам исполняться по-настоящему параллельно, выполняя код одновременно на разных ядрах. Поскольку виртуальная машина Erlang поддерживает симметричную мультипроцессорность (SMP), то и Erlang уже готов к тому, что приложения будут масштабироваться вертикально по мере роста числа ядер процессоров. И поскольку добавление новых узлов к кластеру совсем несложное, всего лишь удостовериться, что новый узел связался с любым существующим и сформировал полный граф, то и горизонтальное масштабирование вполне посильно. Это, в свою очередь, позволяет вам сосредоточиться на настоящих сложностях, когда вы работаете с распределёнными системами, а именно на том, как вы распределите ваши данные и состояние процессов по узлам и по ненадёжной сети.

Подводим итоги

Чтобы облегчить дизайн, реализацию, эксплуатацию и возможность обслуживания и сделать их более устойчивыми, ваш язык программирования и вспомогательные библиотеки должны быть небольшими, их поведение во время исполнения – предсказуемым, а полученный код – лёгким в обслуживании. Мы продолжаем обсуждать устойчивые к сбоям, масштабируемые системы мягкого реального времени с требованиями высокой доступности. Проблемы, которые вам нужно решить, не должны быть сложными, для того чтобы почувствовать выгоду от преимуществ, которые предлагает Erlang/OTP. Эти преимущества будут очевидны, если вы разрабатываете решения, нацеленные на встроенные аппаратные платформы, такие как Parallela Board, Beagleboard или Raspberry Pi. Вы обнаружите, что Erlang/OTP идеально подходит для кода, который занимается «режиссированием» (раздачей команд другим частям системы) во встроенных устройствах, для разработки серверного ПО, куда параллельность подходит естественным образом, и далее до масштабируемых и распределённых многоядерных архитектур и суперкомпьютеров. Он облегчает разработку более сложных проблем ПО, в то время как реализация простых программ становится ещё легче.

Что вы узнаете из этой книги

Книга разделена на две секции. Первая часть включает в себя главы с третьей по десятую и разбирается с проектированием и программированием единственного узла. Эти главы следует читать одну за другой, поскольку их примеры и поясне-

ния построены на предыдущих. Вторая половина книги, от главы 11 и до 16, сосредоточена на инструментах, методиках и архитектурах, которые используются при установке, мониторинге и эксплуатации, одновременно объясняя теоретические подходы к решению проблем с надёжностью, масштабируемостью и высокой доступностью. Вторая половина книги частично строится на примерах из первой половины, но может быть прочитана и отдельно от неё.

Мы начинаем главу 2 с обзора Erlang, который не обучает языку, а скорее служит повторением курса. Если вы ещё не знаете Erlang, мы рекомендуем вам сначала ознакомиться с одной из отличных книг по изучению языка, например *«Вступление в Erlang»* (Саймон Сен Лорен), *«Программирование в Erlang»* (Чезарини и Томпсон) или любой другой из перечисленных в главе 2. Наш обзор касается основных элементов языка, таких как списки, функции, процессы и сообщения, интерактивная консоль Erlang, а также возможности, делающие Erlang уникальным среди языков, такие как связывание и мониторинг процессов, живые обновления и распределение.

Следующая за обзором языка глава 3 погружается в рассмотрение структуры процессов. Процессы Erlang могут решить множество разных задач, но независимо от самих задач или их проблемной области на свет появляются почти одинаковые структуры кода и жизненные циклы процессов, подобно шаблонам проектирования, которые были замечены и документированы для популярных ООП-языков вроде Java или C++. OTP схватывает и формализует эти общие процессно-ориентированные структуры и жизненные циклы в виде *поведений*, которые служат базовыми элементами библиотек многократного использования OTP.

В главе 4 мы изучим в подробностях наш первый рабочий процесс. Это самое популярное и часто используемое стандартное поведение OTP – `gen_server`. Как можно догадаться из его названия, он поддерживает универсальные клиент-серверные структуры, в которых сервер владеет определённым вычислительным ресурсом – возможно, всего лишь простой таблицей ETS или пулом сетевых подключений к удалённому не Erlang-серверу – и обеспечивает своим клиентам доступ к этому ресурсу. Клиенты общаются с обобщёнными серверами синхронно по схеме запрос-ответ, асинхронно с помощью однонаправленных сообщений, называемых *cast*, или обычными средствами обмена сообщениями, которые имеются в Erlang. При рассмотрении этих режимов сообщения нам потребуется подробно присмотреться к мелким деталям: например, что случается, если клиент или сервер аварийно завершит работу посреди обмена сообщениями, как работают таймауты и что может случиться, если сервер получит сообщение, которое не сможет понять. Адресуя эти и другие общие проблемы, `gen_server` обрабатывает множество деталей независимо от проблемной области, позволяя разработчику сосредоточить силы на своём приложении. Поведение `gen_server` настолько полезно, что оно не только появляется в самых разных приложениях на Erlang, но также используется повсюду и в самом OTP.

До того, как мы рассмотрим остальные поведения OTP, добавим к нашему обсуждению `gen_server` обзор некоторых возможностей управления и наблюдения,

предоставляемых поведениями OTP (глава 5). Эти возможности отражают ещё один аспект Erlang/OTP, который выделяет его среди других языков и библиотек: встроенная наблюдаемость. Если вам нужно узнать, чем занят ваш процесс, основанный на `gen_server`, просто включайте отладочную трассировку для этого процесса при компиляции или во время исполнения (через интерактивную консоль). Включенная трассировка заставляет его транслировать отладочную информацию, которая показывает, что за сообщения были получены и какие действия были предприняты для их обработки. Также Erlang/OTP предоставляет функции для заглядывания в работающие процессы, показывающие их текущий стек вызовов, словари процессов, родительские, связанные процессы и прочие подробности. Также есть функции OTP для изучения статуса и внутреннего состояния именно поведений и других системных процессов. По причине наличия этих отладочных возможностей программисты Erlang часто пренебрегают традиционными отладчиками и вместо этого полагаются на трассировку при диагностике ошибочных программ, поскольку часто это быстрее и более информативно.

Затем мы рассмотрим другое поведение OTP, `gen_fsm` (глава 6), которое предлагает универсальный шаблон создания конечного автомата. Как вы, вероятно, уже знаете, конечный автомат – это система, имеющая конечное число состояний, и входящие сообщения могут переводить систему из одного состояния в другое, а побочные эффекты обычно происходят в момент перехода. Например, вы можете рассматривать ваш телевизор в качестве конечного автомата, где текущее состояние – это выбранный канал и факт показа какого-нибудь меню на экране. Нажатие клавиш на вашем пульте ДУ приводит к смене состояния телевизором, вероятно, к выбору другого канала или переводу экранного меню в режим списка каналов или выбора фильма для покупки. Конечные автоматы подходят для множества проблемных областей, поскольку они облегчают разработчикам выбор и реализацию потенциальных состояний и переходов между ними. Знание о том, когда и как использовать `gen_fsm`, может спасти вас от попыток реализовать собственную машину состояний из пластилина и спичек, которая быстро превратится в неуправляемый и нерасширяемый спагетти-код.

Журналирование и мониторинг являются критически важной частью успешных историй о масштабировании, поскольку позволяют вам узнать важную информацию о вашей работающей системе, и это помогает обнаруживать узкие места и проблемные области, требующие дальнейшего расследования. Поведение `gen_event` (глава 7) обеспечивает поддержку подсистем, создающих или управляющих потоками событий, которые отражают изменения в состоянии системы, могущие повлиять на эксплуатационные характеристики, такие как постоянное увеличение нагрузки на процессор, растущие без остановки очереди или отсутствие связи между узлами в распределённом кластере. Такие потоки не обязательно ограничиваются вашими системными событиями. Они могут обрабатывать события вашего приложения, происходящие от действий пользователя, от собранных сетью сенсоров данных или внешних программ. Помимо изучения поведения `gen_event`, мы также взглянем на библиотеки поддержки системной архитектуры OTP (SASL),

дающие нам обработчики событий журналирования, которые обеспечивают гибкость для управления отчетами наблюдателей, отчетами о сбоях и докладами о ходе работы.

Обработчики событий и ошибок являются скрепами во множестве языков программирования, и они невероятно полезны в Erlang/OTP, но не позволяйте их присутствию здесь обмануть вас: обработка ошибок в Erlang/OTP совершенно отличается от подходов, привычных большинству программистов.

После `gen_event` следующим поведением, которое мы изучим, будет наблюдатель (`supervisor`, глава 8), который управляет рабочими процессами. В Erlang/OTP наблюдатели запускают рабочие процессы и затем следят за ними, пока те выполняют задачи вашего приложения. Если вдруг один или несколько рабочих процессов внезапно прекратят работу, наблюдатель может справиться с проблемой одним из нескольких способов, которые мы объясним позже в этой книге. Эта форма обработки ошибок, известная как «позволь ему упасть» («let it crash»), существенно отличается от тактики защитного программирования, которую использует большинство программистов. Наблюдатели и этот принцип «позволь ему упасть» вместе являются критически важным краеугольным камнем Erlang/OTP и на практике очень эффективны.

Затем мы поглядим в последнее из основных поведений OTP, приложение (`application`, глава 9), которое служит первичной точкой интеграции между системой времени выполнения Erlang/OTP и вашим кодом. Приложения в OTP имеют свои файлы конфигурации, где задаются имя приложения, версии, модули, другие приложения, от которых они зависят, и прочие параметры. После запуска системой времени выполнения Erlang/OTP ваше приложение (`application`), в свою очередь, запускает наблюдателя высшего уровня, и он поднимает остальные части приложения. Структурирование модулей кода в приложения также позволяет проводить обновления кода на живых системах. Релиз проекта в Erlang/OTP обычно состоит из ряда приложений, некоторые из которых принадлежат дистрибутиву Erlang/OTP с открытым исходным кодом, а остальные задаёте вы сами.

Изучив стандартные поведения далее, мы обратим наше внимание на написание собственных поведений и специальных процессов (глава 10). Специальные процессы следуют определённым принципам проектирования, что позволяет им быть добавленными в деревья наблюдения OTP. Знание этих правил проектирования не только поможет вам понять подробности реализации стандартных поведений, но и даст знать о возможных компромиссах и позволит вам принять решение, когда использовать одно из стандартных, а когда лучше написать собственное поведение.

Глава 11 описывает, как приложения OTP на одном узле связываются друг с другом и запускаются все вместе. Вам придётся создать собственные файлы релизов, которые в мире Erlang называются *rel*-файлами. В rel-файле перечислены версии приложений и системы времени исполнения, которые используются потом модулем `systools`, чтобы упаковать ПО в отдельный каталог релиза, куда также копируется и виртуальная машина. Каталог релиза после конфигурирова-

ния и упаковки готов к доставке и запуску на целевых машинах. Мы рассмотрим инструменты, разработанные сообществом, *rebar3* и *relx*, в качестве наилучшего способа сборки кода и релизов.

Виртуальная машина имеет настраиваемые системные ограничения и параметры, про которые нужно знать при доставке кода для ваших систем. Есть множество их, начиная от ограничений, регулирующих максимальное число ets-таблиц или процессов, и до путей поиска кода и режимов загрузки модулей. Модули в Erlang загружаются на старте системы либо при первом их вызове. В системах, где используется строгий контроль версий, вам придётся исполнять их во *встроенном* (embedded) режиме, загружая модули при запуске системы и падая аварийно, если что-то не было найдено, либо в *интерактивном* (interactive) режиме, в котором, если модуль недоступен, система попытается найти его, прежде чем завершит процесс. Внешний процесс мониторинга (heart) следит за виртуальной машиной Erlang, посылая ей сигналы «ударов сердца» и вызывая специальный скрипт, который позволяет вам реагировать на ситуацию, когда «удары сердца» перестали подтверждаться. Скрипт вы реализуете самостоятельно, что позволяет вам решать, достаточно ли просто перезапустить узел или же – на основе истории предыдущих перезагрузок – вы пожелаете выполнить эскалацию ситуации и остановить виртуальную машину или перезагрузить сервер целиком.

Хотя динамическая типизация Erlang позволяет вам обновлять модуль во время исполнения, не теряя состояния процесса, она не координирует зависимости между модулями, изменения в состоянии или протоколы, не совместимые с предыдущими версиями. OTP имеет инструментарий для поддержки обновлений системы на системном уровне, включая не только приложения, но и саму систему времени выполнения. Эти принципы и библиотеки поддержки представлены в главе 12, от определения скриптов обновления своего приложения до написания скриптов поддержки обновлений релизов. Подходы и стратегии обращения с изменениями вашей схемы базы данных представлены в виде рекомендаций к обновлению в распределённых окружениях и с несовместимыми протоколами. Большие обновления в распределённых системах, исправления ошибок и протоколов, смены схем базы данных, обновления во время работы системы – занятие не для слабонервных. Они, однако, невероятно мощны и позволяют автоматические обновления и безостановочную эксплуатацию. Моменты, когда вы обнаруживаете, что ваш онлайн-банк недоступен по причине обслуживания, должны давно уже были уйти в прошлое. Если это не так, то отправьте копию этой книги в отдел разработки вашего банка.

Эксплуатация и обслуживание любой системы требуют ясности о том, что же происходит. Масштабирование кластеров требует стратегий того, как организовать общий доступ к данным и состоянию. И устойчивость к сбоям требует подхода к тому, как дублировать и сохранять состояние. В этом деле вам придётся разобраться, как бороться с ненадёжной сетью, сбоями и какие бывают стратегии восстановления. В то время как каждая из этих тем достойна отдельной книги, последние главы данной книги дадут вам теоретические основы, которые могут

понадобиться при распределении ваших систем, масштабировании и повышении их надёжности. Мы предоставляем эту теорию, описывая шаги, требуемые для того, чтобы спроектировать масштабируемую и высокодоступную архитектуру в Erlang/OTP.

Глава 13 даст обзор подходов, важных при проектировании распределённой архитектуры, разделяя вашу функциональность между отдельными узлами сети. Каждый отдельный *тип узла* получит предназначение, например выполнять роль клиентского шлюза, который управляет пулами подключений TCP/IP или обеспечивает сервис авторизации платежей. Для каждого из типов мы определим подход к определению интерфейсов, состояния и данных, которые нужны каждому узлу. Мы завершим эту главу описанием наиболее распространенных распределенных архитектурных шаблонов и различных сетевых протоколов, которые могут быть использованы для подключения узлов друг к другу.

Как только ваша распределённая архитектура будет выбрана, вам понадобится принять ряд проектных решений, которые повлияют на устойчивость к сбоям, стойкость, надёжность и доступность вашего проекта. Уже знаете, какие данные и состояние вам понадобятся в каждом из ваших узлов, но как вы решите вопрос их распределения и сохранения их согласованности? Выберете ли вы подход, при котором «все данные общие» (share-everything), «некоторые данные общие» (share-something) или «никаких общих данных» (share-nothing)? Каковы будут компромиссы, на которые вам придётся пойти при выборе сильной (strong), небрежной (causal) или отложенной (eventual) согласованности данных? В главе 14 мы опишем различные подходы, которые вы можете предпринять, покажем стратегию повторных попыток, о которой следует знать в случае, если ожидание ответа превышает заданное время ожидания в результате или сбоя процесса, узла или сети, или хотя бы в том случае, если ваша сеть или серверы перегружены.

Легко сказать, что вы добавите железа и расширите свою систему горизонтально, но, увы, выбранный вариант дизайна системы (из представленных в главе 14) прямо влияет на возможности масштабирования вашей системы. В главе 15 мы опишем влияние стратегии доступа к общим данным, модели согласованности данных и стратегии повтора попыток. Мы опишем планирование ёмкости, включая нагрузочные, пиковые и стресс-тесты, которым придётся подвергнуть вашу систему, чтобы гарантировать её предсказуемое поведение под сильной нагрузкой, даже когда ПО, оборудование и инфраструктура вокруг неё начинают сыпаться.

Как только вы спроектировали ваши стратегии масштабирования и доступности, надо разобраться с мониторингом. Если вашей целью является достижение пяти девяток доступности системы, вам нужно не только знать, что происходит сейчас, но и определить, что за неприятность только что произошла и какова была причина. Мы завершаем книгу главой 16, рассматривая использование мониторинга для упреждающей поддержки и посмертной отладки систем.

Мониторинг сосредоточивается на сборе метрик, тревожных сигналов и журналов событий. Эта глава обсуждает важность метрик системы и бизнеса. Пример системных метрик: количество использованной памяти на вашем узле, длина оче-

реди сообщений процесса и расход памяти на диске. Комбинируя эти бизнес-метрики, например количество неудачных и успешных попыток авторизации, пропускную способность сообщений в секунду, длительность сессии, можно получить полную прозрачность того, как ваша бизнес-логика влияет на ресурсы вашей системы.

Дополняют метрики *тревожные сигналы*, которые обнаруживают и сообщают об аномалиях, позволяя системе предпринять действия по их решению или предупредить оператора, если вдруг требуется вмешательство человека. Тревожные сигналы могут также сообщать о завершении свободного места на диске (что вызовет автоматически скрипты по очистке диска и сжатию журналов) или о большом количестве неудачно обработанных сообщений (что потребует вмешательства человека по решению проблем с сетью). Упреждающая поддержка в идеале обнаруживает и разрешает проблемы до того, как они обострятся, она очень важна при работе с высокой доступностью. Если у вас нет обзора текущей ситуации в реальном времени, разрешение проблем до их обострения становится очень трудным и громоздким.

И наконец, журналирование основных событий вашей системы помогает обнаружить неисправности после аварийного сбоя, когда вы теряете состояние системы, но можете восстановить ход вызовов какого-то конкретного запроса среди миллионов других, чтобы обработать клиентский запрос на обслуживание или выдать данные для нужд бухгалтерии.

Имея мониторинг наготове, вы будете готовы строить системы, которые смогут не только масштабироваться, но и будут стойкими и высокодоступными. Приятного чтения! Мы надеемся, вы получите столько же удовольствия, читая эту книгу, сколько мы получили, пока её писали.