

# Содержание

<b>Отзывы о первом издании</b> .....	10
<b>Предисловие</b> .....	11
<b>Благодарности</b> .....	12
<b>О книге</b> .....	13
<b>Об авторе</b> .....	17
<b>Иллюстрация на обложке</b> .....	18
<b>Глава 1. Первые шаги</b> .....	19
1.1. Общие сведения об Erlang .....	19
1.1.1. Высокая доступность .....	20
1.1.2. Конкурентная модель Erlang .....	21
1.1.3. Системы на стороне сервера .....	23
1.1.4. Платформа разработки .....	25
1.2. Общие сведения об Elixir .....	26
1.2.1. Упрощенный код .....	27
1.2.2. Композиция функций .....	30
1.2.3. Выводы .....	31
1.3. Недостатки .....	32
1.3.1. Скорость .....	32
1.3.2. Экосистема .....	32
Выводы .....	33
<b>Глава 2. Основы языка</b> .....	34
2.1. Интерактивная оболочка .....	35
2.2. Работа с переменными .....	36
2.3. Организация кода .....	37
2.3.1. Модули .....	37
2.3.2. Функции .....	39
2.3.3. Арность функций .....	42
2.3.4. Видимость функций .....	43
2.3.5. Импорты и псевдонимы .....	44
2.3.6. Атрибуты модулей .....	45
2.3.7. Комментарии .....	47
2.4. Понятие системы типов .....	48
2.4.1. Числа .....	48
2.4.2. Атомы .....	49
2.4.3. Кортежи .....	51
2.4.4. Списки .....	52
2.4.5. Иммутабельность .....	56
2.4.6. Словари .....	59
2.4.7. Бинарные данные и битовые строки .....	62
2.4.8. Строки .....	63
2.4.9. Функции первого класса .....	65
2.4.10. Прочие встроенные типы .....	67
2.4.11. Типы данных более высокого уровня .....	68
2.4.12. Списки ввода-вывода .....	72
2.5. Операторы .....	73
2.6. Макросы .....	74

2.7. Среда выполнения.....	75
2.7.1. Модули и функции в среде выполнения.....	75
2.7.2. Запуск среды выполнения.....	78
Выводы.....	80
<b>Глава 3. Поток управления.....</b>	<b>81</b>
3.1. Сопоставление с образцом.....	81
3.1.1. Оператор сопоставления.....	82
3.1.2. Сопоставление кортежей.....	82
3.1.3. Сопоставление с константой.....	83
3.1.4. Переменные в качестве образцов.....	84
3.1.5. Сопоставление списков.....	85
3.1.6. Сопоставление словарей.....	86
3.1.7. Сопоставление с битовыми строками и бинарными данными.....	86
3.1.8. Сложные сопоставления.....	88
3.1.9. Обобщенное поведение.....	90
3.2. Сопоставление с образцом в функциях.....	90
3.2.1. Функции с несколькими предложениями.....	91
3.2.2. Ограничители.....	94
3.2.3. Анонимные функции с несколькими предложениями.....	96
3.3. Условные конструкции.....	97
3.3.1. Ветвление с помощью функций с несколькими предложениями.....	97
3.3.2. Классические конструкции ветвления.....	99
3.3.3. Специальная форма with.....	101
3.4. Циклы и итерации.....	104
3.4.1. Итерация на основе рекурсии.....	105
3.4.2. Хвостовые вызовы функций.....	106
3.4.3. Функции высшего порядка.....	109
3.4.4. Генераторы.....	114
3.4.5. Потоки.....	116
Выводы.....	119
<b>Глава 4. Абстракции данных.....</b>	<b>121</b>
4.1. Создание абстракций с помощью модулей.....	122
4.1.1. Создание простой абстракции.....	123
4.1.2. Сложные абстракции.....	125
4.1.3. Структурирование данных с помощью словарей.....	126
4.1.4. Абстракции на основе структур.....	127
4.1.5. Прозрачность данных.....	131
4.2. Работа с иерархическими данными.....	133
4.2.1. Генерация идентификаторов.....	134
4.2.2. Обновление записей.....	136
4.2.3. Обновление неизменяемых иерархических данных.....	138
4.2.4. Итеративное обновление.....	140
4.2.5. Практика: импорт из файла.....	141
4.3. Полиморфизм с помощью протоколов.....	143
4.3.1. Общие сведения о протоколах.....	143
4.3.2. Реализация протокола.....	144
4.3.3. Встроенные протоколы.....	145
Выводы.....	147
<b>Глава 5. Основы конкурентности.....</b>	<b>148</b>
5.1. Конкурентность в BEAM.....	148
5.2. Работа с процессами.....	151

5.2.1. Создание процессов .....	152
5.2.2. Обмен сообщениями .....	154
5.3. Серверные процессы с сохранением состояния .....	159
5.3.1. Серверные процессы .....	159
5.3.2. Сохранение состояния процесса .....	163
5.3.3. Изменяемое состояние .....	165
5.3.4. Сложные состояния .....	168
5.3.5. Регистрация процессов .....	172
5.4. Особенности времени выполнения .....	173
5.4.1. Последовательность выполнений действий в процессах .....	173
5.4.2. Бездонные почтовые ящики процессов .....	175
5.4.3. Конкуренность без разделения ресурсов .....	176
5.4.4. Внутреннее устройство планировщиков .....	177
Выводы .....	178
<b>Глава 6. Обобщенные серверные процессы</b> .....	<b>179</b>
6.1. Создание обобщенного серверного процесса .....	179
6.1.1. Подключение к обобщенному коду с помощью модулей .....	180
6.1.2. Реализация обобщенного кода .....	181
6.1.3. Использование обобщенной абстракции .....	182
6.1.4. Поддержка асинхронных запросов .....	184
6.1.5. Упражнение: реорганизация сервера для списка дел .....	185
6.2. Использование GenServer .....	186
6.2.1. Поведения OTP .....	187
6.2.2. Подключение к GenServer .....	187
6.2.3. Обработка запросов .....	188
6.2.4. Обработка простых сообщений .....	190
6.2.5. Прочие особенности GenServer .....	191
6.2.6. Жизненный цикл процесса .....	194
6.2.7. Совместимые с OTP процессы .....	195
6.2.8. Упражнение: создание сервера для списка дел на основе GenServer .....	196
Выводы .....	196
<b>Глава 7. Создание конкурентной системы</b> .....	<b>198</b>
7.1. Работа с проектом mix .....	198
7.2. Управление несколькими списками дел .....	200
7.2.1. Создание кеш-процесса .....	201
7.2.2. Создание тестов .....	203
7.2.3. Анализ зависимостей процесса .....	206
7.3. Сохранение данных .....	208
7.3.1. Кодирование и сохранение .....	208
7.3.2. Использование базы данных .....	210
7.3.3. Анализ системы .....	213
7.3.4. Устранение узкого места процесса .....	214
7.3.5. Упражнение: пул процессов и синхронизация .....	217
7.4. Логика работы процессов .....	218
Выводы .....	219
<b>Глава 8. Основы отказоустойчивости</b> .....	<b>220</b>
8.1. Ошибки времени выполнения .....	221
8.1.1. Типы ошибок .....	221
8.1.2. Обработка ошибок .....	222
8.2. Ошибки в конкурентных системах .....	226
8.2.1. Установка связей между процессами .....	227
8.2.2. Мониторы .....	229

8.3. Супервизоры .....	230
8.3.1. Подготовка существующего кода .....	232
8.3.2. Запуск процесса-супервизора .....	232
8.3.3. Спецификации потомков.....	235
8.3.4. Обертка супервизора.....	237
8.3.5. Использование модуля обратного вызова.....	237
8.3.6. Связывание всех процессов .....	238
8.3.7. Частота перезапусков .....	241
Выводы .....	242
<b>Глава 9. Изолирование последствий ошибок .....</b>	<b>243</b>
9.1. Деревья супервизоров .....	244
9.1.1. Разделение слабо связанных частей .....	244
9.1.2. Усовершенствованное обнаружение процессов .....	247
9.1.3. Via-кортежи.....	249
9.1.4. Регистрация рабочих процессов базы данных .....	251
9.1.5. Наблюдение за рабочими процессами.....	253
9.1.6. Построение дерева супервизоров .....	256
9.2. Динамический запуск рабочих процессов .....	259
9.2.1. Регистрация серверных процессов.....	260
9.2.2. Динамические супервизоры.....	260
9.2.3. Обнаружение серверных процессов.....	262
9.2.4. Использование временных рабочих процессов .....	263
9.2.5. Тестирование системы .....	264
9.3. Let it crash .....	265
9.3.1. Процессы, отказа которых допускать нельзя .....	266
9.3.2. Обработка ожидаемых ошибок .....	267
9.3.3. Сохранение состояния.....	268
Выводы .....	269
<b>Глава 10. За пределами GenServer .....</b>	<b>270</b>
10.1. Задачи .....	270
10.1.1. Задачи с ожиданием ответа .....	271
10.1.2. Задачи без ожидания ответа .....	273
10.2. Агенты.....	275
10.2.1. Использование агентов .....	275
10.2.2. Агенты и конкурентность .....	276
10.2.3. Сервер списка дел на основе модуля Agent .....	277
10.2.4. Пределы возможностей агентов .....	279
10.3. Таблицы ETS .....	281
10.3.1. Основные операции .....	284
10.3.2. Хранилище ключ/значение на основе таблицы ETS .....	287
10.3.3. Прочие операции ETS.....	290
10.3.4. Упражнение: реестр процессов.....	293
Выводы .....	295
<b>Глава 11. Работа с компонентами.....</b>	<b>296</b>
11.1. OTP-приложения.....	296
11.1.1. Создание приложений с помощью инструмента mix .....	296
11.1.2. Поведение приложения.....	298
11.1.3. Запуск приложения .....	299
11.1.4. Библиотечные приложения .....	300
11.1.5. Создание приложения текущей системы.....	300
11.1.6. Структура каталогов приложения .....	302
11.2. Работа с зависимостями .....	304

11.2.1. Добавление зависимости .....	305
11.2.2. Реорганизация пула процессов.....	305
11.2.3. Визуализация системы.....	308
11.3. Создание веб-сервера .....	309
11.3.1. Выбор зависимостей.....	309
11.3.2. Запуск сервера .....	310
11.3.3. Обработка запросов.....	312
11.3.4. Логика работы системы.....	315
11.4. Настройка приложений .....	319
11.4.1. Окружение приложения .....	319
11.4.2. Изменяемость настроек .....	320
11.4.3. Особенности скриптов конфигурации.....	321
Выводы .....	322
<b>Глава 12. Создание распределенной системы.....</b>	<b>323</b>
12.1. Примитивы распределенных вычислений.....	325
12.1.1. Запуск кластера.....	325
12.1.2. Взаимодействие узлов.....	326
12.1.3. Обнаружение процессов.....	329
12.1.4. Ссылки и мониторы.....	332
12.1.5. Прочие сервисы распределения .....	333
12.2. Создание отказоустойчивого кластера.....	335
12.2.1. Устройство кластера .....	336
12.2.2. Распределенный кеш.....	336
12.2.3. Создание репликационной базы данных .....	341
12.2.4. Тестирование системы .....	344
12.2.5. Обнаружение потери связности сети.....	346
12.2.6. Высокодоступные системы .....	347
12.3. Особенности сетевого соединения .....	348
12.3.1. Имена узлов .....	348
12.3.2. Файлы cookie .....	349
12.3.3. Скрытые узлы .....	350
12.3.4. Фаерволы.....	350
Выводы .....	352
<b>Глава 13. Запуск системы.....</b>	<b>353</b>
13.1. Запуск системы с помощью инструментов Elixir .....	353
13.1.1. Использование команд mix и elixir .....	354
13.1.2. Выполнение скриптов.....	355
13.1.3. Компиляция для промышленной эксплуатации.....	356
13.2. OTP-релизы .....	358
13.2.1. Создание релиза с помощью distillery .....	358
13.2.2. Использование релиза.....	360
13.2.3. Структура релиза .....	361
13.3. Анализ поведения системы.....	365
13.3.1. Отладка.....	365
13.3.2. Журналирование.....	367
13.3.3. Взаимодействие с системой.....	367
13.3.4. Трассировка.....	368
Выводы .....	371

# Отзывы о первом издании

Увлекательно и познавательно... море практических советов.

– *Вед Антани, компания Electronic Arts*

Прекрасно показано на реальных примерах, на что способен Elixir по части распределенных вычислений.

– *Кристофер Бэйли, компания HotelTonight*

Если вы хотите научиться думать и решать проблемы как настоящий эликсирщик, эта книга для вас!

– *Космас Чатзимикалис*

Функциональное программирование стало понятнее.

– *Мохсен Мостафа Джокар, газета Hamshari*

Возможно, лучшее введение в Elixir и функциональное программирование.

– *Покупатель интернет-магазина Amazon*

Отличная книга для опытных разработчиков, желающих познакомиться с Elixir поближе.

– *Покупатель интернет-магазина Amazon*

# Предисловие

В 2010 году передо мной стояла задача реализации системы для передачи систематических обновлений нескольким тысячам пользователей в близком к реальному масштабе времени. В моей компании в основном использовали Ruby on Rails, но мне нужно было что-то, более подходящее для такой задачи с высокой степенью конкурентности. Последовав совету технического директора, я обратился к языку Erlang, изучил литературу о нем, сделал прототип и провел нагрузочное тестирование. Я был впечатлен полученными результатами и решил реализовать уже реальный проект на Erlang. Пару месяцев спустя система была готова и с тех пор прекрасно работает.

Со временем я стал всё больше ощущать ценность языка и то, как он помог мне организовать управление такой сложной системой, а постепенно и вовсе предпочел его используемым ранее технологиям. Я начал знакомить людей с языком сначала внутри компании, а потом на местных мероприятиях. В итоге в 2012 году я стал вести блог «The Erlangist» (<http://theerlangelist.com>), где стараюсь продемонстрировать приверженцам ООП все преимущества Erlang.

Поскольку Erlang – особенный язык, я решил попробовать Elixir в надежде, что он поможет мне показать всю красоту Erlang более понятным для ООП-программистов способом. Несмотря на то что Elixir тогда был еще совсем молод (версия 0.8), я был просто поражен его зрелостью и легкой интеграцией с Erlang. Вскоре я начал разрабатывать на Elixir новые функции для своей системы на основе Erlang.

Спустя еще несколько месяцев на меня вышел Майкл Стивенс (Michael Stephens) из издательства Manning и заинтересовался, не хотел бы я написать книгу об Elixir. На тот момент о нем уже готовились две книги, и я подумал, что мог бы добавить к ним ещё одну, где язык рассматривался бы с другого ракурса: акцентируя внимание на конкурентной модели Elixir и философии OTP. Работать над книгой было непросто, но это того стоило.

По прошествии двух лет с момента публикации первого издания я согласился работать над вторым. По сути, это то же первое издание, приведенное в соответствие с последними обновлениями Elixir и Erlang. Наиболее важные изменения коснулись глав 8, 9 и 10 – значительные их части были переписаны, и теперь они включают новые методы работы с супервизорами и реестрами процессов.

Книга «Elixir в действии» содержит актуальную информацию и поможет вам изучить новейшие приемы разработки программного обеспечения на Elixir. Надеюсь, вам понравится моя книга, вы сможете многому научиться и применить свои знания на практике!

# Благодарности

Прежде всего мне хотелось бы поблагодарить мою жену Ренату за нескончаемое терпение и поддержку в то продолжительное время, когда я днями и ночами работал над книгой.

Благодарю издательство Manning за публикацию книги. В частности, Майкла Стивенса (Michael Stephens) за то, что вышел со мной на связь, Марьян Бэйс (Marjan Bace) за предоставленную возможность написать эту книгу, Берта Бэйтса (Bert Bates) за то, что задал мне верное направление, Карен Миллер (Karen Miller) за то, что помогала не сбиться с пути, Александра Драгосавльевича (Aleksandar Dragosavljevic) за вычитку текста, Кевина Салливана (Kevin Sullivan) и Винсента Нордхауса (Vincent Nordhaus) за подготовку книги к публикации, Тиффани Тэйлор (Tiffany Taylor) и Энди Кэррола (Andy Carroll) за преобразование моего разговорного языка в литературный, а также Кэндис Гиллхули (Candace Gillhoolley), Ану Ромак (Ana Romac) и Кристофера Кауфманна (Christopher Kaufmann) за продвижение книги.

Качество содержимого данной книги удалось значительно повысить благодаря отзывам рецензентов и первых читателей. В первую очередь я хотел бы сказать спасибо Эндрю Джибсону (Andrew Gibson), давшему ценные комментарии и помогшему мне преодолеть последний рубеж. Также благодарю Алексея Шолика (Alexei Sholik) и Питера Минтена (Peter Minten) за своевременную помощь по технической части во время написания книги.

Выражаю огромную благодарность Ризе Фахми (Riza Fahmi) и всем остальным техническим редакторам: Элу Рахими (Al Rahimi), Алану Лентону (Alan Lenton), Алексею Галиуллину (Alexey Galiullin), Эндрю Кортэру (Andrew Courter), Аруну Кумару (Arun Kumar), Асхаду Дину (Ashad Dean), Кристоферу Бэйли (Christopher Bailey), Кристоферу Хаупту (Christopher Haupt), Кливу Харберу (Clive Harber), Даниэлю Куперу (Daniel Couper), Йогану О’Доннелу (Eoghan O’Donnell), Фредерику Шиллеру (Frederick Schiller), Габору Ласло Хашбе (Gábor László Hajba), Джорджу Томасу (George Thomas), Хизер Кэмпбелл (Heather Campbell), Джерону Бенкхушсену (Jeroen Benckhuijsen), Хорхе Дефлону (Jorge Deflon), Хоце Валиму (José Valim), Космасу Чатсимихалису (Kosmas Chatzimichalis), Мафинару Хану (Mafinar Khan), Марку Райалу (Mark Ryall), Матиасу Полигкайту (Mathias Polligkeit), Мохсену Мустафе Джокару (Mohsen Mostafa Jokar), Тому Гейденсу (Tom Geudens), Томеру Эльмалему (Tomer Elmalem), Веду Антани (Ved Antani) и Юрию Бодареву (Yurii Bodarev).

Я также хотел бы поблагодарить всех читателей – участников программы раннего доступа издательства «Маннинг» (Manning Early Access Program, MEAP), предоставивших справедливые замечания. Спасибо, что потратили время на прочтение моей писанины и оставили такие полезные отзывы.

Особого упоминания заслуживают люди, подарившие нам Elixir и Erlang, а именно сами создатели, участники их команды и помощники. Спасибо вам за эти замечательные продукты, благодаря которым мне стало проще и интереснее работать с кодом. И наконец, отдельная благодарность всем членам сообщества Elixir! Это самое прекрасное и дружелюбное сообщество программистов из всех существующих!



Elixir – современный функциональный язык программирования, предназначенный для создания масштабируемых, распределенных и отказоустойчивых систем, работающих на основе виртуальной машины Erlang. Этот язык привлекателен сам по себе, но взаимодействие с платформой разработки Erlang – несомненно, его огромное преимущество.

Платформа Erlang была создана как средство обеспечения высокой доступности. Изначально она была предназначена для разработки телекоммуникационных систем, но сегодня ее используют для создания инструментов совместной работы, систем открытых торгов в режиме реального времени, серверов баз данных, многопользовательских онлайн-игр и еще во многих других областях. Система, обслуживающая огромное количество пользователей со всего мира, должна работать непрерывно без ощутимых задержек, независимо от ошибок и проблем с аппаратными средствами, возникающих во время ее эксплуатации. Ни одному конечному пользователю не понравится испытывать частые и продолжительные перебои. Такая система ненадежна и непригодна для использования, а значит, не выполняет свою главную функцию. Высокая доступность – крайне важное свойство системы, и Erlang помогает его достичь.

Elixir призван улучшить и модернизировать разработку систем на основе Erlang. Он объединяет в себе функциональные особенности таких языков программирования, как Erlang, Clojure и Ruby. В его стандартную поставку входят инструменты, упрощающие процессы управления проектом, тестирования, упаковки и создания документации. Можно сказать, что Elixir снижает порог вхождения в Erlang и увеличивает скорость разработки. Благодаря лежащей в основе среде выполнения Erlang при создании систем на Elixir вам доступны любые библиотеки экосистемы Erlang, включая проверенный временем фреймворк OTP.

## Для кого эта книга

Эта книга содержит обучающий материал, изучив который, вы сможете создавать на Elixir готовые к промышленной эксплуатации системы. Вы не найдете здесь подробной информации о каждом аспекте языка или нюансе работы виртуальной машины Erlang. Точность вычислений с плавающей запятой, специфика Unicode, файловые операции ввода-вывода, модульное тестирование и многие другие темы рассмотрены лишь поверхностно или опущены. Все это очень важно, но не является главным фокусом данной книги. При необходимости вы можете изучить эту информацию самостоятельно, а в рамках данной книги сосредоточиться на решении более интересных и необычных задач, а именно на том, как с помощью конкурентного программирования можно сделать систему масштабируемой, отказоустойчивой, распределенной и высокодоступной.

Представленные в книге подходы и решения также рассмотрены не полностью. Некоторые нюансы были опущены для краткости и смещения акцента в сторону главной проблемы. Моей целью было не охватить как можно больше деталей,

а рассказать о лежащих в основе принципах и их объединении в общую картину. Прочитав эту книгу, вы легко сможете получить недостающие вам знания: для этого на протяжении всей книги вставлены все необходимые ссылки и упоминания.

Поскольку темы, рассматриваемые в книге, далеко не для начинающих, вы должны иметь в виду некоторые требования, предъявляемые к читателю. Во-первых, вы должны быть профессиональным разработчиком, имеющим пару лет опыта. Ваш стек технологий значения не имеет. Вы можете писать на Java, C#, Ruby, C++ или любом другом языке программирования общего назначения. Приветствуется опыт в разработке бэкенда (серверной стороны) систем.

Вам не обязательно знать что-либо об Erlang, Elixir или других платформах конкурентного программирования и не нужно разбираться в функциональном программировании. Если вы программируете на объектно-ориентированном языке, вначале изучение Elixir может вызвать трудности. Но как ООП-разработчик с большим стажем спешу вас заверить, что бояться тут нечего. Лежащие в основе Elixir принципы функционального программирования довольно легки в понимании. Конечно, функциональное программирование сильно отличается от привычной вам картины, к нему нужно просто привыкнуть. Но это гораздо проще, чем кажется, и опытному разработчику не составит труда разобраться во всех представленных в книге основах.

## СТРУКТУРА КНИГИ

Книга состоит из трех частей.

Первая часть – введение в Elixir. В ней описываются основы языка и подробно рассматриваются самые часто используемые идиомы функционального программирования:

- в главе 1 приведен обзор технологий Elixir и Erlang, описаны области их применения и их отличия от других языков и платформ;
- в главе 2 описываются такие структурные единицы Elixir, как модули, функции и система типов;
- в главе 3 в подробностях рассматривается сопоставление с образцом и использование его для управления потоком выполнения;
- в главе 4 показывается создание абстракций более высокого уровня на основе иммутабельных структур данных.

Вторая часть строится на изученных в первой части основах. Основное внимание уделяется конкурентной модели Erlang и ее основным преимуществам – масштабируемости и отказоустойчивости:

- в главе 5 представлена конкурентная модель Erlang, а также основные примитивы конкурентных вычислений;
- в главе 6 рассматриваются обобщенные серверные процессы – главные структурные элементы для создания высококонкурентных систем на Elixir/Erlang;
- в главе 7 показывается процесс создания более сложной конкурентной системы;

- в главе 8 вы изучите подходы к обработке ошибок, в которых особое внимание уделяется конкурентности системы;
- глава 9 содержит подробную информацию об изоляции всех типов ошибок и ограничении их влияния на этапе промышленной эксплуатации;
- в главе 10 представлено несколько альтернатив обобщенным серверным процессам, более подходящих для определенного рода ситуаций.

В третьей части рассказывается о системах на этапе промышленной эксплуатации:

- в главе 11 рассматриваются ОТР-приложения, необходимые для упаковки повторно используемых компонентов;
- в главе 12 описываются распределенные системы, призванные улучшить отказоустойчивость и масштабируемость;
- в главе 13 показаны различные способы подготовки Elixir-систем к промышленной эксплуатации, основное внимание уделено ОТР-релизам.

## О ЛИСТИНГАХ С КОДОМ

Исходный код в данной книге представлен

моноширинным шрифтом,

выделяющим его на фоне остального текста. Код многих листингов сокращен в целях обращения внимания на рассматриваемые приемы. Чтобы код мог уместиться в свободное место на странице, в нем используются переносы строк и абзацные отступы.

Все приведенные в книге примеры можно найти в репозитории GitHub по адресу: <https://github.com/sasa1977/elixir-in-action>. Вы также можете загрузить их в сжатом виде со страницы издательства по адресу: [www.manning.com/books/elixir-in-action-second-edition](http://www.manning.com/books/elixir-in-action-second-edition).

## ФОРУМ КНИГИ

При покупке данной книги вы получаете бесплатный доступ к закрытому веб-форуму от издательства Manning Publications ([www.manning.com/books/elixir-in-action-second-edition](http://www.manning.com/books/elixir-in-action-second-edition)), где можете оставить отзыв о книге, задать технические вопросы и получить помощь от авторов и других пользователей. На странице <https://forums.manning.com/forums/about> вы можете узнать больше о форумах Manning и ознакомиться с правилами поведения на них.

Издательство Manning считает своим обязательством предоставить такое пространство, в котором каждый читатель смог бы вести конструктивный диалог с авторами книг. Данные форумы не преследуют цели продвижения того или иного автора, и любая помощь осуществляется им исключительно на добровольной основе. Задавайте авторам свои каверзные вопросы, чтобы их интерес не угасал!

## ОТЗЫВЫ И ПОЖЕЛАНИЯ

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

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

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

## СКАЧИВАНИЕ ИСХОДНОГО КОДА ПРИМЕРОВ

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте [www.dmkpress.com](http://www.dmkpress.com) или [www.дмк.рф](http://www.дмк.рф) на странице с описанием соответствующей книги.

## СПИСОК ОПЕЧАТОК

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу [dmkpress@gmail.com](mailto:dmkpress@gmail.com), и мы исправим это в следующих тиражах.

## НАРУШЕНИЕ АВТОРСКИХ ПРАВ

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Manning очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу электронной почты [dmkpress@gmail.com](mailto:dmkpress@gmail.com) со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

# Об авторе



**Саша Юрич** – разработчик с большим опытом в создании объемных конкурентных систем на стороне сервера, десктопных приложений для Windows, а также приложений для терминалов. После 20 лет программирования на объектно-ориентированных языках он открыл для себя Erlang и Elixir. С помощью обеих технологий он разработал масштабируемый отказоустойчивый HTTP push-сервер и соответствующую серверную систему. На данный момент является членом команды Aircloak, где использует Erlang для создания автоматически конфигурируемого программного решения с обеспечением конфиденциальности данных. Он также ведет свой блог об Elixir и Erlang (<http://theerlangelist.com>).

# Иллюстрация на обложке

Рисунок на обложке книги называется «Русская девушка». Иллюстрация взята из книги в четырех томах «Коллекция костюмов различных народов, античных и современных» (A Collection of the Dresses of Different Nations, Ancient and Modern) Томаса Джеффериса (Thomas Jefferys), изданной в Лондоне между 1757 и 1772 годом. Как отмечается на титульной странице, это раскрашенные вручную гравюры на меди, обработанные гуммиарабиком. Томас Джефферис (1719–1771) носил звание «Географ короля Георга III». Он был английским картографом и крупнейшим поставщиком карт в своем времени. Он выгравировал и напечатал множество карт для правительства и других официальных органов, а также изготовил целый ряд коммерческих карт и атласов, в особенности Северной Америки. Будучи картографом, Джефферис проявлял интерес к одежде местных жителей, которую безупречно отразил в своей четырехтомной коллекции.

В конце XVIII века путешествия в дальние страны ради удовольствия были относительно новым явлением, и коллекции вроде этой пользовались популярностью, так как знакомили самих путешественников и любителей с культурой других народов. Разнообразие рисунков в книгах Джеффериса позволяет увидеть, насколько уникальными и своеобразными были народности во всем мире 200 лет назад. С тех пор дресс-код поменялся, и нам уже не увидеть столь богатого разнообразия в одежде отдельных регионов и стран. В наше время даже сложно отличить друг от друга жителей разных континентов. С оптимистичной точки зрения, можно сказать, что мы променяли культурное и внешнее разнообразие на более насыщенную личную жизнь или более богатую и интересную интеллектуальную и техническую деятельность.

В наше время, когда крайне непросто отличить одну техническую книгу от другой, издательство Manning подчеркивает изобретательность и инициативу сферы компьютерных технологий вот такими обложками, основанными на богатом разнообразии региональной жизни двухвековой давности, претворяя иллюстрации Джеффериса в жизнь.

# Глава 1

## Первые шаги

В главе рассматривается:

- платформа разработки Erlang;
- преимущества Elixir.

Здесь начинается ваше путешествие в мир Elixir и Erlang – двух эффективных и удобных технологий, призванных значительно упростить разработку больших масштабируемых систем. Скорее всего, вы читаете эту книгу в целях изучения Elixir. Но так как Elixir работает поверх платформы Erlang и в значительной степени зависит от нее, стоит для начала ознакомиться с Erlang и ее сильными сторонами. Предлагаю кратко пробежаться по основам Erlang.

### 1.1. ОБЩИЕ СВЕДЕНИЯ ОБ ERLANG

*Erlang* – это платформа для разработки надежных масштабируемых систем с небольшим или нулевым временем простоя. Громкие слова, но Erlang существует именно для этих целей. Идея создания Erlang зародилась в середине 1980-х годов в шведском телекоммуникационном гиганте Ericsson и была обусловлена потребностью в удовлетворении нужд телекоммуникационных систем компании, в которых ключевую роль играли такие свойства, как надежность, быстрое реагирование, масштабируемость и постоянная доступность. Телекоммуникационные системы должны функционировать бесперебойно, независимо от количества одновременных вызовов, непредвиденных ошибок и производимых обновлений ПО и аппаратуры.

Исходя из истории, платформа Erlang должна была стать специализированной технологией для телекоммуникационных систем, но этого не произошло. Она не имеет в своем арсенале определенных средств для программирования телефонов, коммутаторов и прочих телекоммуникационных устройств. Напротив, Erlang – это платформа разработки общего назначения, предоставляющая необходимые инструменты для создания систем, обладающих такими нефункциональными свойствами, как конкурентность, масштабируемость, отказоустойчивость и высокая доступность.

В конце 80-х – начале 90-х годов, когда большинство приложений было десктопными, в высокой доступности нуждались лишь специализированные системы, в частности телекоммуникационные. На сегодняшний день ситуация изменилась: интернет и веб вышли на первый план, и приложения в большинстве своем ра-

ботаю на основе серверных систем, которые обрабатывают запросы, выполняют операции с данными и передают соответствующую информацию большому количеству подключенных клиентов. Популярные сегодня системы – социальные сети, системы управления контентом, мультимедиа по запросу (Media on Demand, MoD) и многопользовательские компьютерные игры – больше рассчитаны на общение и взаимодействие пользователей.

Все вышеперечисленные системы имеют общие нефункциональные требования. Во-первых, система должна сохранять способность к реагированию независимо от количества подключенных клиентов. Во-вторых, непредвиденные ошибки не должны оказывать влияния на целую систему. Не так страшно, если из-за ошибки не выполнится единичный запрос, но если свернется вся система, это уже большая проблема. В идеале система никогда не должна отказывать, даже во время обновления ПО. Она должна всегда предоставлять услуги своим пользователям, работая бесперебойно.

Может показаться, что достичь таких целей довольно сложно, но при разработке систем, которыми люди пользуются каждый день, это просто необходимо. Если система не удовлетворяет требованиям по надежности и стабильности работы, значит, она не выполняет свою функцию. Поэтому системы, работающие на стороне сервера, обязательно должны находиться в постоянном доступе.

Именно для этого и предназначена Erlang. Высокая доступность обеспечивается благодаря таким техническим принципам, как масштабируемость, отказоустойчивость и распределение вычислений. В отличие от других современных платформ разработки, Erlang создавалась с основным фокусом на эти принципы. Команда разработчиков Ericsson под руководством Джо Армстронга (Joe Armstrong) несколько лет потратила на проектирование, прототипирование и тестирование своей платформы. В начале 90-х она имела ограниченную область применения, а сейчас она может быть полезна практически каждой системе.

В последнее время к Erlang проявляется большой интерес. Уже более 20 лет она обеспечивает работу различных объемных систем, а именно: приложения для обмена сообщениями WhatsApp, распределенной базы данных Riak, облачной платформы Heroku, системы автоматизированного развертывания Chef, очереди сообщений RabbitMQ, финансовых систем и бэкендов сетевых игр. Erlang – истинно проверенная временем и масштабом технология. В чем же секрет Erlang? Давайте же разберемся, как создавать надежные системы с высокой доступностью при помощи Erlang.

### 1.1.1. Высокая доступность

Erlang изначально была создана для разработки высокодоступных систем – онлайн-систем, способных предоставлять услуги своим пользователям даже при возникновении непредвиденных обстоятельств. На первый взгляд задача может показаться незамысловатой, но, как вы, возможно, догадываетесь, на этапе эксплуатации многое может пойти не по плану. Чтобы обеспечить бесперебойную работу системы, потребуется удовлетворить ряд технических требований:

- *отказоустойчивость*. Система не должна прерывать работу в случае возникновения непредусмотренных ошибок: отказа отдельных компонентов, разрыва сетевого соединения, выключения устройства, на котором она



запущена. В любом случае необходимо локализовать негативные последствия ошибки, насколько это возможно, устранить их и вернуть систему к жизни;

- *масштабируемость*. Система должна быть готова к нагрузкам любой сложности. Разумеется, не нужно судорожно скупать всю существующую технику, если все население планеты внезапно захочет воспользоваться вашей системой. Но стоит позаботиться о случаях повышенной нагрузки путем выделения дополнительных аппаратных ресурсов, не меняя ничего в коде. В идеале это должно осуществляться без перезагрузки системы.
- *распределенные вычисления*. Для создания непрерывно работающей системы необходимо использовать несколько компьютеров. Так вы сможете увеличить общую надежность системы: при отказе одного компьютера другой придет ему на замену. Кроме того, становится возможным горизонтальное масштабирование – добавив больше машин, вы сможете справиться с проблемой повышенной нагрузки;
- *способность к реагированию*. Понятно, что системе всегда необходимо быть достаточно быстрой и отзывчивой. Время обработки запросов не должно существенно увеличиваться даже при большой нагрузке на систему или возникновении непредвиденных ошибок. В частности, периодически возникающие длительные операции не должны оказывать существенного влияния на производительность системы или выводить ее из строя;
- *автоматическое обновление*. В некоторых случаях бывает необходимо запустить новую версию программы, не перезагружая при этом серверы. К примеру, в системе телефонной связи недопустимо во время обновления программы прервать текущие вызовы.

Если вышеперечисленные требования будут удовлетворены, ваша система будет высокодоступной, предоставляя услуги пользователям, несмотря ни на что.

Erlang вооружена всеми необходимыми для этого инструментами, ведь она была разработана для этих целей. При помощи конкурентной модели Erlang система неизбежно станет высокодоступной, обладая всеми присущими такой системе свойствами.

Давайте же разберемся, как реализуется конкурентность в Erlang.

## 1.1.2. Конкурентная модель Erlang

*Конкурентность* – основное оружие разработанных на Erlang систем. Практически каждая нетривиальная Erlang-система является системой с высокой степенью многопоточности, а сам язык даже иногда называют конкурентно-ориентированным. Вместо использования тяжеловесных потоков и процессов операционной системы (ОС) Erlang реализует конкурентность по своим собственным правилам, как показано на рис. 1.1.

Простейший элемент этой схемы называется *процессом Erlang* (не путать с процессом ОС или потоком). В типовых Erlang-системах выполняются тысячи или даже миллионы таких процессов. Виртуальная машина Erlang *BEAM* (*Bogdan/Björn's Erlang Abstract Machine*) имеет свой планировщик, который распределяет выполняемые процессы по доступным ядрам центрального процессора (ЦП). То, как реализованы процессы, дает множество преимуществ.

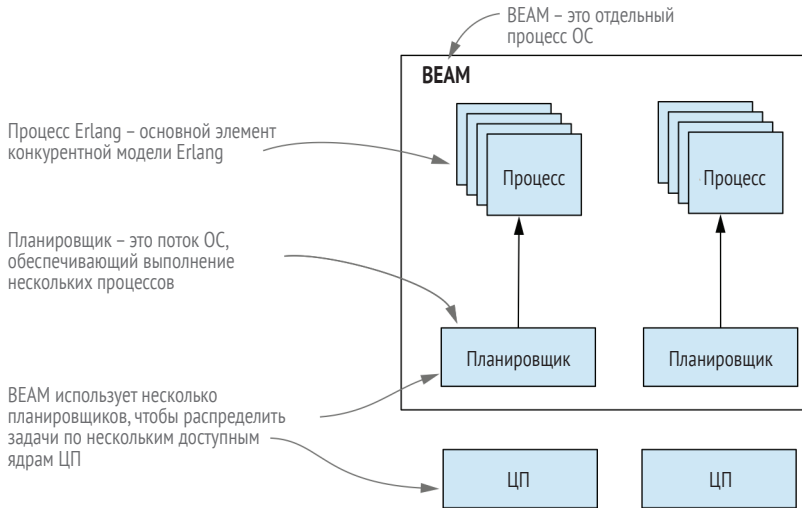


Рис. 1.1 ❖ Конкурентность в виртуальной машине Erlang

### Отказоустойчивость

Процессы Erlang полностью изолированы друг от друга. У них нет общей памяти, а аварийное завершение одного процесса никак не сказывается на других. Именно это помогает ограничить последствия непредвиденной ошибки. Более того, Erlang предоставляет средства для обнаружения аварийного завершения процесса и последующего принятия решения. Обычно вместо старого процесса запускается новый.

### Масштабируемость

Процессы, не имеющие общей памяти, взаимодействуют между собой посредством обмена асинхронными сообщениями. Это означает, что они обходятся без таких сложных механизмов синхронизации, как блокировки, мьютексы и семафоры. Соответственно, взаимодействие конкурентных объектов гораздо проще понять и реализовать.

Типовые Erlang-системы строятся на большом количестве конкурентных процессов, которые выполняют основную функцию системы (предоставляют услуги пользователям) путем взаимодействия между собой. Виртуальная машина по максимуму распараллеливает выполнение процессов, используя все доступные ядра процессора, что предоставляет Erlang-системам возможность масштабирования.

### Распределенное выполнение

Взаимодействие между процессами происходит одинаково, независимо от того, где они находятся – в одном экземпляре BEAM или в двух разных экземплярах на двух разных удаленных компьютерах. Таким образом, написанная на Erlang система заведомо подготовлена к распределению вычислений между несколькими компьютерами, а это, в свою очередь, дает возможность горизонтального мас-

штабирования – использования группы компьютеров (кластера), разделяющих общую нагрузку системы. Помимо того, запуск программ на нескольких машинах делает систему поистине отказоустойчивой, ведь если одна машина выйдет из строя, другая сможет ее заменить.

### **Отзывчивость**

Среда выполнения настроена таким образом, чтобы повысить общую отзывчивость системы. Как уже отмечалось ранее, в Erlang выполнение многочисленных процессов организовано специальными планировщиками. Реализуется вытесняющая многозадачность: планировщик предоставляет каждому процессу определенный временной промежуток, а по его истечении ставит его на паузу и запускает другой процесс. Поскольку промежуток совсем небольшой, процесс с долгим временем выполнения не останавливает всю систему. К тому же операции ввода-вывода выполняются в разных потоках, или используется служба опроса ядра ОС, если доступна. Это означает, что любой процесс, ожидающий выполнения операции ввода-вывода, не будет блокировать выполнение других процессов.

Даже процесс сбора мусора в Erlang направлен на улучшение отзывчивости системы. Напоминаю, процессы полностью изолированы и не имеют общей памяти. Сбор мусора осуществляется отдельно для каждого процесса. Это происходит намного быстрее, система не блокируется на долгое время и продолжает работать. В действительности в системах с многоядерными процессорами можно сделать так, чтобы одно ядро ЦП осуществляло быстрый сбор мусора, в то время как остальные ядра были бы заняты выполнением стандартных задач.

Как видите, конкурентность – ключевое понятие Erlang, и это не простой параллелизм. Благодаря своей реализации конкурентность обеспечивает отказоустойчивость, распределенное вычисление и отзывчивость системы. Типовые Erlang-системы выполняют множество конкурентных задач, используя тысячи или даже миллионы процессов. Особенно это может быть актуально при разработке систем на стороне сервера, которые в большинстве случаев можно полностью реализовать на Erlang.

## **1.1.3. Системы на стороне сервера**

На Erlang разрабатывают различные системы и приложения. Существуют примеры десктопных приложений, написанных на Erlang, но в основном они используются во встраиваемых системах. На мой взгляд, больше всего платформа подходит для разработки систем на стороне сервера – систем, запускаемых на одном или более серверах и обслуживающих множество клиентов одновременно. Термин *система на стороне сервера* дает понять, что это нечто большее, чем простой обрабатывающий запросы сервер. Это целая система, которая вдобавок к обработке запросов должна выполнять различные фоновые задачи и управлять данными на сервере (server-wide), хранящимися в памяти, как показано на рис. 1.2.

Система на стороне сервера обычно функционирует на нескольких компьютерах, работающих совместно для решения единой бизнес-задачи. Чтобы обеспечить баланс нагрузки и добавить сценарии обработки отказа, можно разместить различные компоненты на разных машинах, а также развернуть их на разных серверах.

Конец ознакомительного фрагмента.

Приобрести книгу можно

в интернет-магазине

«Электронный универс»

[e-Univers.ru](http://e-Univers.ru)