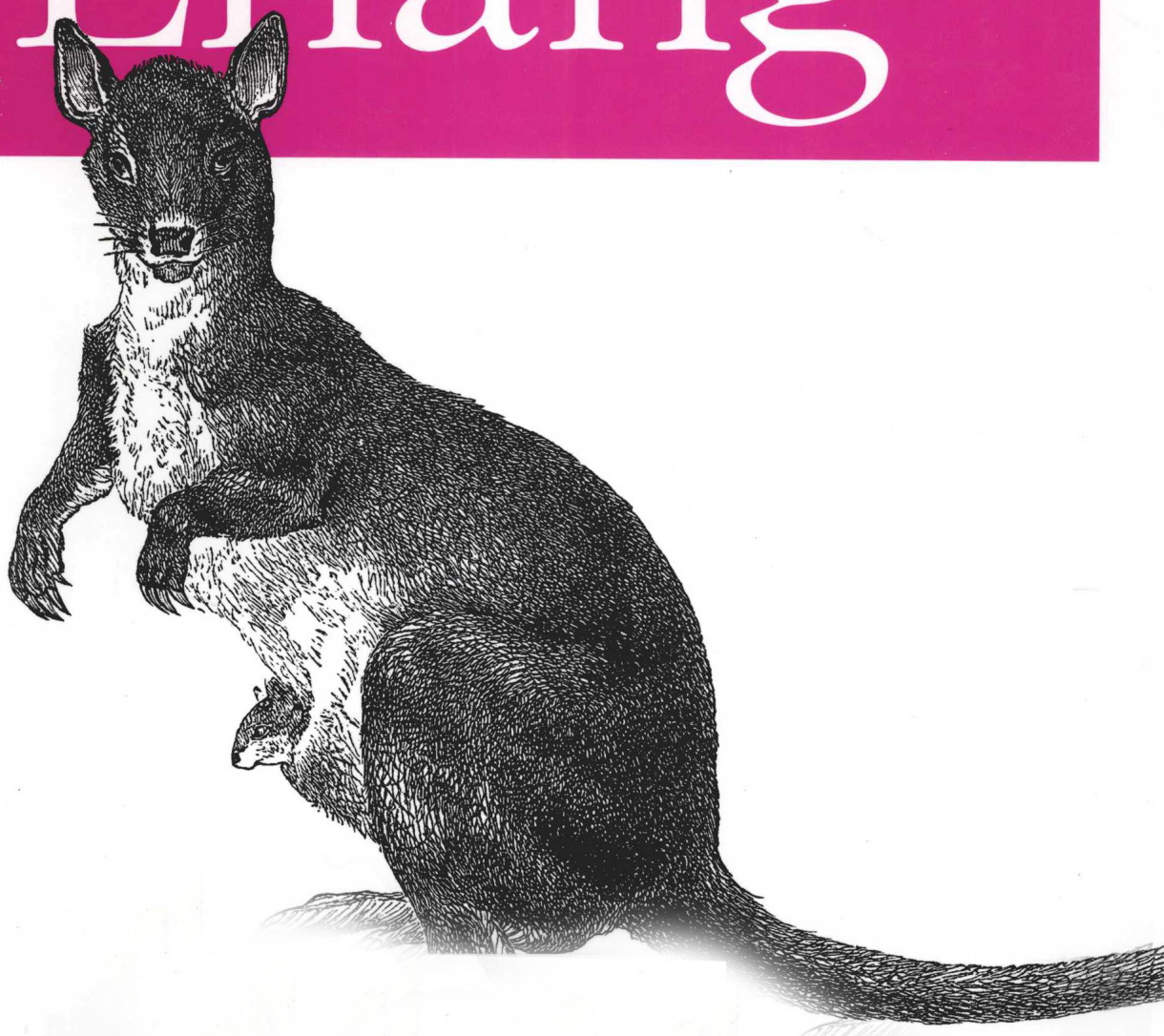
Erlang 编程（影印版）

Programming Erlang



O'REILLY®

東南大學出版社

Francesco Cesarini & Simon Thompson 著

Erlang 编程(影印版)

Erlang Programming

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Taipei • Tokyo

O'Reilly Media, Inc. 授权东南大学出版社出版

东南大学出版社

图书在版编目（CIP）数据

Erlang编程：英文/（瑞典）塞萨芮利（Cesarini, F.），（英）汤普森（Thompson, S.）著. —影印本. —南京：东南大学出版社，2010.6

书名原文：Erlang Programming

ISBN 978-7-5641-2269-0

Ⅰ. ①E… Ⅱ. ①塞… ②汤… Ⅲ. ①程序语言－程序设计－英文 Ⅳ. ①TP312

中国版本图书馆CIP数据核字（2010）第089047号

江苏省版权局著作权合同登记

图字：10-2010-157号

Erlang编程（影印版）

出版发行：东南大学出版社
地　　址：南京四牌楼2号　　邮编：210096
出 版 人：江　汉
网　　址：http://press.seu.edu.cn
电子邮件：press@seu.edu.cn
印　　刷：扬中市印刷有限公司
开　　本：787毫米×980毫米　16开本
印　　张：31印张
字　　数：607千字
版　　次：2010年6月第1版
印　　次：2010年6月第1次印刷
书　　号：ISBN 978-7-5641-2269-0
印　　数：1~1800册
定　　价：64.00元（册）

本社图书若有印装质量问题，请直接与读者服务部联系。电话（传真）：025-83792328

Foreword

Erlang is our solution to three problems regarding the development of highly concurrent, distributed "soft real-time systems":

- To be able to develop the software quickly and efficiently
- To have systems that are tolerant of software errors and hardware failures
- To be able to update the software on the fly, that is, without stopping execution

When we "invented" Erlang, we focused on telecommunication systems, but today these requirements are applicable to a large number of applications, and Erlang is used in applications as divergent as distributed databases, financial systems, and chat servers, among others. Recent interest in Erlang has been fueled by its suitability for use on multicore processors. While the world is struggling to find methods to facilitate porting applications to multicore processors, Erlang applications can be ported with virtually no changes.

Initially, Erlang was slow to spread; maybe it was too daring to introduce functional programming, lightweight concurrency, asynchronous message passing, and a unique method to handle failures, all in one go. It is easy to see why a language such as Java, which is only a small step away from C++, was easier for people to swallow. However, to achieve the goals I've just mentioned, we feel our approach has weathered the test of time. The use of Erlang is expanding rapidly.

This book is an excellent and practical introduction of Erlang, and is combined with a number of anecdotes explaining the ideas and background behind the development of Erlang.

Happy and, I trust, profitable reading.

—Mike Williams
Director of Traffic and Feature Software
Product Development Unit WCDMA, Ericsson AB
one of the inventors of Erlang

Preface

What made us start writing this book in the first place is the enthusiasm we share for Erlang. We wanted to help get the word out, giving back a little of what the community has given to us. Although we both got into Erlang for very different reasons, the end result was the same: lots of fun hours doing lots of fun stuff at a fraction of the effort it would have taken with other languages. And best of all, it is not a tool we use for hobby projects, but one we use on a daily basis in our real jobs!

Francesco: Why Erlang?

The year was 1994. While studying computer science at Uppsala University, one of the courses I took was on parallel programming. The lecturer held up the first edition of *Concurrent Programming in Erlang* (Prentice Hall) and said, "Read it." He then held up a handout and added, "These are the exercises, do them," after which Erlang barely got a mention; it was quickly overshadowed with the theory of threads, shared memory, semaphores, and deadlocks.

As the main exercise for this course, we had to implement a simulated world inhabited by carrots, rabbits, and wolves. Rabbits would roam this world eating carrots that grew in random patches. When they had eaten enough carrots, the rabbits would get fat and split in two. Wolves ran around eating up the rabbits; if they managed to catch and eat enough rabbits, they would also get fat and split. Rabbits and wolves within a certain distance of each other would broadcast information on food and predators. If a rabbit found a carrot patch, other rabbits would quickly join him. If a wolf found a rabbit, the pack would start chasing it.

The final result was amusingly fun to watch. The odd rabbit would run straight into a group of wolves, while others would run in other directions, sometimes stopping to grab a carrot en route. Every carrot patch, rabbit, and wolf was represented as an Erlang process communicating through message passing.

The exercise took me about 40 hours to solve. Although I enjoyed using Erlang and was positively surprised at the simplicity of its concurrency model and lack of OS threads for every process, I did not think that much of it right there and then. After all, it was one of the dozen or so languages I had to learn for my degree. Having used ML

in my functional programming courses and ADA in my real-time programming courses, for me Erlang was just another language in the crowd. That changed a few months later when I started studying object-oriented programming.

In the object-oriented (OO) programming course, we were given the same simulated world lab but had to solve it with Eiffel, an OO language our new lecturer insisted was ideal for simulations. Although I had already solved the same problem and was able to reuse a good part of the algorithms, it took me and a fellow student 120 man-hours to solve.

This was the eye-opener that led me to believe the declarative and concurrent features in Erlang had to be the direction in which software development was heading. At the time, I was not sure whether the language that would lead the way in this paradigm shift was going to be Erlang, but I was certain that whatever language it was, it would be heavily influenced by Erlang and its ancestors. I picked up the phone and called Joe Armstrong, one of the inventors of Erlang. A week later, I visited the Ericsson Computer Science Lab for an interview, and I have never looked back.

Simon: Why Erlang?

I have worked in functional programming since the early 1980s, and have known about Erlang ever since it was first defined about 20 years ago. What I find most attractive about Erlang is that it's a language that was designed from the start to solve real and difficult problems, and to do it in an elegant and powerful way. That's why we've seen Erlang used in more and more systems in recent years.

It's also a small language, which makes writing tools for it much more practical than for a language such as Java, C++, or even Haskell. This, and the quality of the libraries we've been able to build on in our work, has helped the functional programming group at Kent to be very productive in implementing the Wrangler refactoring tool for Erlang.

Who Should Read This Book?

We have written this book to introduce you to programming in Erlang. We don't expect that you have programmed in Erlang before, nor do we assume that you are familiar with functional programming in other languages.

We do expect you to have programmed in Java, C, Ruby, or another mainstream language, and we've made sure that we point out to you where Erlang differs from what you're used to.

How to Read This Book

We wrote this book in two parts, the first to be read sequentially and the second can be read concurrently (or sequentially in whatever order you like), as the chapters are independent of each other.

The first 11 chapters of the book cover the core parts of Erlang:

- Chapter 1 gives a high-level introduction to the language, covering its key features for building high-availability, robust concurrent systems. In doing this, we also describe how Erlang came to be the way it is, and point out some of its high-profile success stories, which explain why you may want to adopt Erlang in one of your projects.
- The basics of sequential programming in Erlang are the subject of Chapters 2 and 3. In these chapters, we cover the central role of *recursion* in writing Erlang programs, as well as how *single assignment* in Erlang is quite different from the way variables are handled in other languages, such as C and Java.
- While covering sequential programming, we also introduce the *basic data types* of Erlang—numbers, atoms, strings, lists, and tuples—comparing them with similar types in other languages. Other types are covered later: records in Chapter 7, and function types and binaries in Chapter 9. Large-scale storage in ETS tables is the topic of Chapter 10.
- Erlang's distinctiveness comes to the fore in Chapters 4–6, which together cover the concurrent aspects of Erlang, embodied in *message passing* communication between concurrently executing *processes* running in separate memory spaces.
- It is possible to "hot-swap" code in a system, supporting *software upgrades* in running systems: this is the topic of Chapter 8.
- To conclude this part of the book, we cover *distributed programming* in Chapter 11. This allows different Erlang runtime systems (or *nodes*), which might be running on the same or different machines, to work together and interact as a distributed system.

In the remaining chapters, we cover a variety of different topics largely independent of each other. These include the following:

- The *Open Telecom Platform* (OTP) gives a set of libraries and design principles supporting the construction of robust, scalable systems in Erlang; this is the subject of Chapter 12.
- The Erlang distribution contains some standard computing applications: we cover the Mnesia *database* in Chapter 13 and the wxErlang *GUI programming* library in Chapter 14.

- Erlang distribution gives one mechanism for linking Erlang systems to each other. Chapter 15 shows how Erlang supports programming across the Internet using *sockets*, and Chapter 16 covers the various ways in which Erlang can *interwork* with systems written in C, Java, and Ruby, as well as many other languages.
- The standard Erlang distribution comes with a number of very useful tools, and we cover some of these next. Chapter 17 explains in depth how all aspects of Erlang systems can be traced without degrading their performance, and Chapter 18 covers tools for checking the correctness of programs, and for constructing documentation for Erlang systems. Unit testing, and how it is supported by EUnit, is the subject of Chapter 19.
- The last chapter, Chapter 20, looks at how to write programs that are elegant, readable, and efficient, and pulls together into one place much of the accumulated experience of the Erlang community.

The Appendix covers how to get started with Erlang, how to use the Erlang shell, popular tools for Erlang, and how to find out more about Erlang.

Each chapter is accompanied by a set of exercises, and you can download all the code in this book from its website:

http://www.erlangprogramming.org

The website also has references to further reading as well as links to the major sites supporting the Erlang community.

We wrote this book to be compatible with Erlang Release 13 (R13-B). Most of the features we describe will work with earlier releases; known incompatibilities with more recent earlier releases are detailed on our website.

Conventions Used in This Book

The following typographical conventions are used in this book:

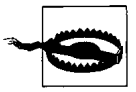
Italic
: Indicates new terms, URLs, filenames, file extensions, and occasionally, emphasis and keyword phrases.

`Constant width`
: Indicates computer coding in a broad sense. This includes commands, options, variables, attributes, keys, requests, functions, methods, types, classes, modules, properties, parameters, values, objects, events, event handlers, XML and XHTML tags, macros, and keywords.

`Constant width bold`
: Indicates commands or other text that the user should type literally.

Constant width italics
: Indicates text that should be replaced with user-supplied values or values determined by context.

This icon signifies a tip, suggestion, or general note.



This icon indicates a warning or caution.

Using Code Examples

This book is intended to help you write programs and systems in Erlang. In general, you may use the code in this book in your programs and documentation.

You do not need to contact the publisher for permission unless you are reproducing a significant portion of the code. For example, if you are writing a program that uses several chunks of code from this book you are not required to secure our permission. Answering a question by citing this book and quoting example code does not require permission.

Incorporating a significant amount of example code from this book into your product's documentation *does* require permission. Selling or distributing a CD-ROM of examples from O'Reilly books *does* require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Erlang Programming*, by Francesco Cesarini and Simon Thompson. Copyright © 2009 Francesco Cesarini and Simon Thompson, 978-0-596-51818-9."

If you feel your proposed use of code examples falls outside fair use or the permission given here, feel free to contact us as *permissions@oreilly.com*.

Safari® Books Online

When you see a Safari® Books Online icon on the cover of your favorite technology book, that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at *http://my.safaribooksonline.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

On the web page for this book we list errata, examples, and any additional information. You can access this page at:

http://www.oreilly.com/catalog/9780596518189

or at:

http://www.erlangprogramming.org

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our website at:

http://www.oreilly.com/

Acknowledgments

In writing this book, we need to acknowledge everyone who made it possible. We start with Jan "Call Me Henry" Nyström, who helped jumpstart this project.

The team at O'Reilly Media provided us with endless support. In particular, our editor, Mike Loukides, patiently guided us through the process and provided encouragement, ensuring that the chapters kept on coming. Special thanks also go out to Audrey Doyle for the copyediting, and to Rachel Monaghan, Marlowe Shaeffer, Lucie Haskins, Sumita Mukherji, and everyone else on the production team.

We continue with the OTP team, and in particular, Bjorn Gustavsson, Sverker Eriksson, Dan Gudmundsson, Kenneth Lundin, Håkan Mattsson, Raimo Niskanen, and Patrik Nyblom, who helped us not only with the undocumented and unreleased features, ensuring that what is in print is in line with the latest release, but also with accuracy and correctness.

Other reviewers who deserve a special mention include Thomas Arts, Zvi Avraham, Franc Bozic, Richard Carlsson, Dale Harvey, Oscar Hellström, Steve Kirsch, Charles McKnight, Paul Oliver, Pierre Omidyar, Octavio Orozio, Rex Page, Michal Ptaszek, Corrado Santoro, Steve Vinoski, David Welton, Ulf Wiger, and Mike Williams.

Although we will not go into detail regarding what each of you did, it is important that you all know that your individual contributions had an influence in making this a better book. Thank you all!

Francesco needs to thank Alison for all her patience and support. I did not know what I was getting into when I agreed to write this book, and neither did you. Until the time to start working on the next book comes, I promise you laptop- and cell phone-free vacations. A thank you also goes to everyone at Erlang Training and Consulting for all the encouragement and to Simon for being such a great coauthor. We should all do it again sometime, as the result was worth it. But now, rest!

Simon wants to say a huge thank you to Jane, Alice, and Rory for their patience and support over the past few very busy months: without your encouragement, it just wouldn't have happened. Thanks, too, to Francesco for inviting me to join the project: it's been really enjoyable working together. I hope we get the chance to do it again, just not too soon....

Table of Contents

Foreword xiii

Preface xv

1. Introduction 1
Why Should I Use Erlang? 1
The History of Erlang 3
Erlang's Characteristics 4
High-Level Constructs 4
Concurrent Processes and Message Passing 5
Scalable, Safe, and Efficient Concurrency 6
Soft Real-Time Properties 6
Robustness 6
Distributed Computation 7
Integration and Openness 8
Erlang and Multicore 9
Case Studies 10
The AXD301 ATM Switch 10
CouchDB 11
Comparing Erlang to C++ 12
How Should I Use Erlang? 14

2. Basic Erlang 15
Integers 15
The Erlang Shell 16
Floats 17
Mathematical Operators 17
Atoms 19
Booleans 20
Tuples 21
Lists 22
Characters and Strings 22

Atoms and Strings 23
Building and Processing Lists 24
List Functions and Operations 25
Term Comparison 28
Variables 30
Complex Data Structures 32
Pattern Matching 33
Functions 38
Modules 40
Compilation and the Erlang Virtual Machine 40
Module Directives 41
Exercises 43

3. Sequential Erlang 45
Conditional Evaluations 46
The case Construct 46
Variable Scope 48
The if Construct 49
Guards 50
Built-in Functions 53
Object Access and Examination 53
Type Conversion 54
Process Dictionary 55
Meta Programming 55
Process, Port, Distribution, and System Information 56
Input and Output 57
Recursion 59
Tail-Recursive Functions 63
Tail-Call Recursion Optimization 66
Iterations Versus Recursive Functions 67
Runtime Errors 68
Handling Errors 70
Using try ... catch 70
Using catch 74
Library Modules 77
Documentation 77
Useful Modules 79
The Debugger 80
Exercises 82

4. Concurrent Programming 89
Creating Processes 90
Message Passing 92

Receiving Messages 94
Selective and Nonselective Receives 97
An Echo Example 100
Registered Processes 102
Timeouts 104
Benchmarking 106
Process Skeletons 107
Tail Recursion and Memory Leaks 108
A Case Study on Concurrency-Oriented Programming 110
Race Conditions, Deadlocks, and Process Starvation 112
The Process Manager 114
Exercises 115

5. Process Design Patterns 117
Client/Server Models 118
A Client/Server Example 119
A Process Pattern Example 125
Finite State Machines 126
An FSM Example 127
A Mutex Semaphore 129
Event Managers and Handlers 131
A Generic Event Manager Example 132
Event Handlers 135
Exercises 137

6. Process Error Handling 139
Process Links and Exit Signals 139
Trapping Exits 142
The monitor BIFs 144
The exit BIFs 145
BIFs and Terminology 146
Propagation Semantics 148
Robust Systems 148
Monitoring Clients 150
A Supervisor Example 152
Exercises 154

7. Records and Macros 157
Records 158
Introducing Records 158
Working with Records 159
Functions and Pattern Matching over Records 160
Records in the Shell 161

Record Implementation 162
Record BIFs 164
Macros 165
Simple Macros 165
Parameterized Macros 166
Debugging and Macros 166
Include Files 168
Exercises 168

8. Software Upgrade 173
Upgrading Modules 173
Behind the Scenes 176
Loading Code 179
The Code Server 180
Purging Modules 182
Upgrading Processes 182
The .erlang File 186
Exercise 186

9. More Data Types and High-Level Constructs 189
Functional Programming for Real 189
Funs and Higher-Order Functions 190
Functions As Arguments 190
Writing Down Functions: fun Expressions 192
Functions As Results 193
Using Already Defined Functions 194
Functions and Variables 195
Predefined, Higher-Order Functions 195
Lazy Evaluation and Lists 197
List Comprehensions 198
A First Example 198
General List Comprehensions 198
Multiple Generators 200
Standard Functions 200
Binaries and Serialization 201
Binaries 202
The Bit Syntax 203
Pattern-Matching Bits 205
Bitstring Comprehensions 206
Bit Syntax Example: Decoding TCP Segments 206
Bitwise Operators 208
Serialization 208
References 210

Exercises 211

10. ETS and Dets Tables 213
ETS Tables 213
Implementations and Trade-offs 214
Creating Tables 216
Handling Table Elements 217
Example: Building an Index, Act I 218
Traversing Tables 220
Example: Building an Index, Act II 222
Extracting Table Information: match 223
Extracting Table Information: select 225
Other Operations on Tables 226
Records and ETS Tables 226
Visualizing Tables 228
Dets Tables 229
A Mobile Subscriber Database Example 231
The Database Backend Operations 232
The Database Server 237
Exercises 242

11. Distributed Programming in Erlang 245
Distributed Systems in Erlang 245
Distributed Computing in Erlang: The Basics 247
Node Names and Visibility 249
Communication and Security 250
Communication and Messages 252
Node Connections 253
Remote Procedure Calls 256
The rpc Module 258
Essential Distributed Programming Modules 258
The epmd Process 260
Distributed Erlang Behind Firewalls 261
Exercises 261

12. OTP Behaviors 263
Introduction to OTP Behaviors 263
Generic Servers 266
Starting Your Server 266
Passing Messages 268
Stopping the Server 270
The Example in Full 271
Running gen_server 273

Supervisors 276
Supervisor Specifications 277
Child Specifications 278
Supervisor Example 279
Dynamic Children 280
Applications 281
Directory Structure 282
The Application Resource File 283
Starting and Stopping Applications 284
The Application Monitor 287
Release Handling 287
Other Behaviors and Further Reading 290
Exercises 291

13. Introducing Mnesia 293
When to Use Mnesia 293
Configuring Mnesia 295
Setting Up the Schema 295
Starting Mnesia 296
Mnesia Tables 296
Transactions 299
Writing 299
Reading and Deleting 300
Indexing 301
Dirty Operations 302
Partitioned Networks 304
Further Reading 305
Exercises 306

14. GUI Programming with wxErlang 309
wxWidgets 309
wxErlang: An Erlang Binding for wxWidgets 310
Objects and Types 311
Event Handling, Object Identifiers, and Event Types 312
Putting It All Together 313
A First Example: MicroBlog 314
The MiniBlog Example 317
Obtaining and Running wxErlang 321
Exercises 321

15. Socket Programming 323
User Datagram Protocol 323
Transmission Control Protocol 327