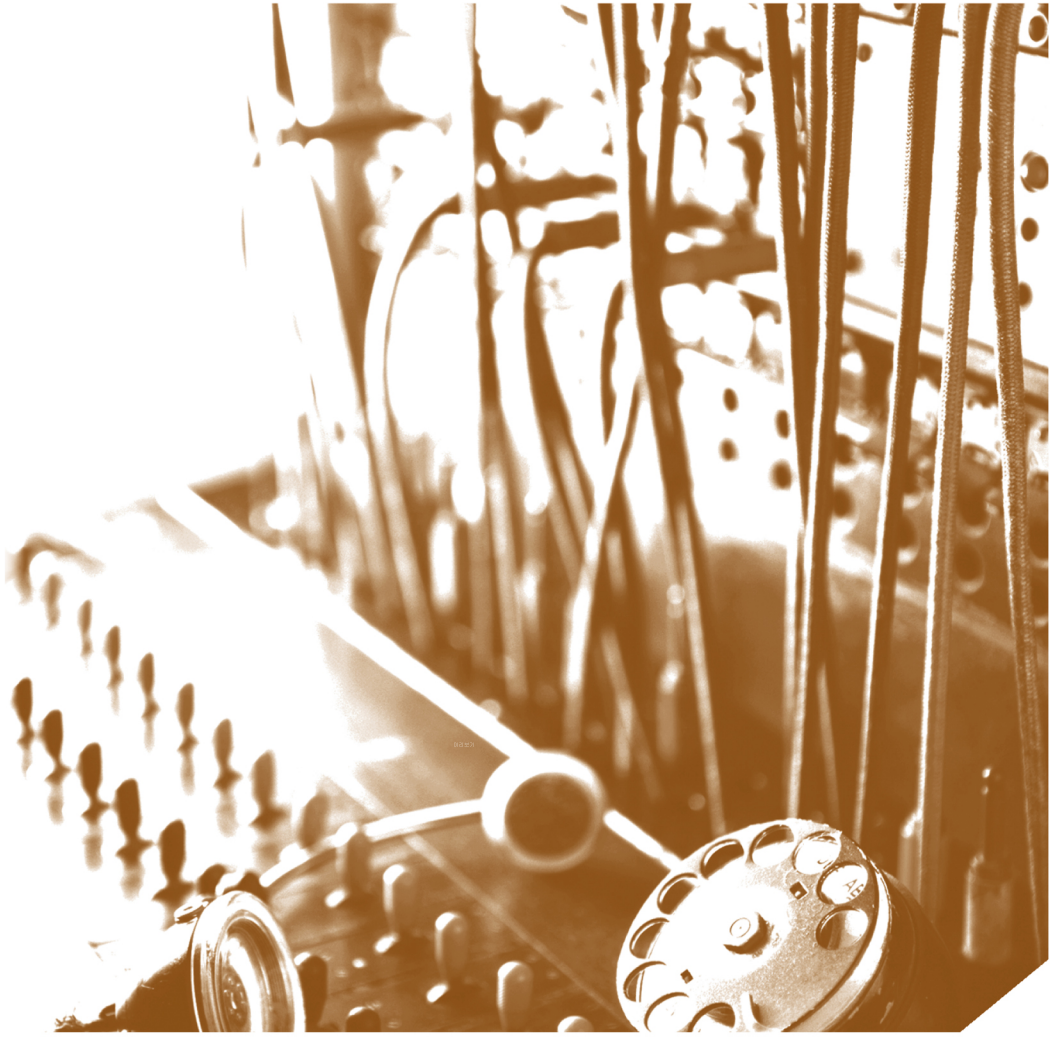
Programming Insight

조 암스트롱 지음 | 김석준 옮김



Programming Erlang

Software for a Concurrent World

프로그래밍 얼랭

인사이트 insight

프로그래밍 **얼랭**

Programming Erlang

PROGRAMMING ERLANG : Software for a Concurrent World
By Joe Armstrong

프로그래밍 얼랭 PROGRAMMING ERLANG

초판 PDF 1.0 2015년 3월 6일 **지은이** 조 암스트롱 **옮긴이** 김석준 **펴낸이** 한기성 **펴낸곳** 인사이트 **편집** 김승호 **등록번호** 제10-2313호 **등록일자** 2002년 2월 19일 **주소** 서울시 마포구 잔다리로 119 석우빌딩 3층 **전화** 02-322-5143 **팩스** 02-3143-5579 **블로그** http://blog.insightbook.co.kr **이메일** insight@insightbook.co.kr **ISBN** 978-89-6626-133-8 이 책의 정오표는 http://www.insightbook.co.kr/17594에서 확인하실 수 있습니다.

ProgrammingInsight

차례

옮긴이의 글

갑자기 누군가 여러분에게 자기소개를 해보라고하면 여러분은 어느 나라 말로 하시겠습니까? (잠깐 멈춤) 그렇다면 이번엔 누군가 불쑥 여러분에게 프로그램을 하나 짜보라 하면요? (다시 잠깐 멈춤) 그렇습니다. 우리 모두에게는 모국어가 있고 습관처럼 익숙한 프로그래밍 언어가 있을 것입니다. 저는 개인적으로 루비언어를 가장 좋아하고 즐겨 다루는 편입니다. 왠만한 숙제는 루비로 처리하곤 하죠. 물론 때에 따라서는 자바도 쓰고 자바스크립트도 다루고 C나 파이썬을 사용해야 할 때도 있습니다. 아, 한 가지 빼먹은 게 있군요. 바로 이 책의 주인공인 얼랭입니다.

얼랭은 애초 에릭슨에서 통신용 교환기에 사용할 목적으로 개발한 언어입니다. 통신쪽 프로그래밍을 해본 분이라면 아마도 그 어마어마한 덩치의 교환기가 쉬지도 않고 24시간 365일 묵묵히 돌아가는 것을 보면서 나도 언젠가 저런 프로그램 한번 만들어봤으면 하고 생각하신 분들도 계셨을 겁니다. 저는 그저 바라보기만 할 뿐 감히 엄두조차 내지 못했지만요.

그런 얼랭이 다시 돌아온 것은 묘하게도 소위 '웹2.0'이라는 트랜드가 IT와 인터넷 세상을 뜨겁게 달구는 시점이었습니다. 구글과 같은 인터넷 기업들이 그 많은 트래픽을 어떻게 감당하는지, '플랫폼'이라는 기치를 내걸고 속속 등장한 웹서비스들은 어떻게 중단없이 지속적으로 사용자들에게 서비스를 제공할 건지 하는 이슈들이 부각되는 속에서, 멀티코어 환경에서 안정적이면서도 쉽게 병행 네트워크 프로그래밍을 할 수 있게 해주는 얼랭이라는 언어가 빛을 발하게 된 것입니다.

얘기는 다르지만, 많이들 읽으셨을 『실용주의 프로그래머』라는 책에 보면 개발자들이 '지식 포트폴리오'를 갖추는 한 방편으로 매년 적어도 한 가지씩 새로운 언어를 배우라는 제안이 나옵니다. 새로운 언어는 동일한 문제를 바라보는 다른 시각을 열어주고 편협한 사고에 갇히는 것을 예방해 주기 때문이라죠. 모르긴 해도 얼랭처럼 기존의 잘 알려진 언어들과는 판이하게 다른 언어를 배우는 경우라면 아마도 더 그럴 것입니다.

얼랭이 왜 다르고 왜 특별한지는 이 책 속에 잘 나와 있으니 여기서 따로 중언부

언하는 것은 괜한 낭비가 될 것 같고. 대신 짬을 내어 '여행'을 권합니다. 문화가 아주 다른 곳을 여행하고 나면 그전보다 훨씬 커진 자기 자신을 느끼게 되듯, 이 책이 안내하는 낯선 프로그래밍의 세계로 한번 여행해 보길 말이죠. 비자는 필요 없습니다. 단, 가벼운 사전 하나 정도는 챙기고 떠나는 게 좋겠죠.

물론 이 책이 얼랭의 전부를 다루는 책은 아닙니다. 온라인에는 이 책보다 더 쉽고 더 심도깊은 자료들도 많이 있습니다. 그렇지만 이 책은 얼랭을 공부하는데 있어 좋은 시작점이자 언제든 곁에 두고 기본을 돌아볼 수 있는 그런 책이라 생각합니다.

이 책을 번역하면서 많은 분들의 도움을 받았습니다. 우선 이 책의 초역에 대해 리뷰를 해 주신 서광열, 양봉열, 이병준, 장회수, 전희원 님께 무엇보다도 큰 감사를 드립니다. 그분들의 하나같이 예리하고 사려깊은 지적과 통찰은 저에게 번역이라는 작업의 의미를 다시 한 번 돌아보게 만드는 계기가 되었습니다. 제게 얼랭을 처음 소개시켜 주신 애자일 컨설팅의 김창준님께도 감사드립니다. 여러모로 부족한 저에게 선뜻 귀한 책의 번역을 맡겨주신 인사이트 한기성 사장님께는 늘 감사하는 마음뿐이고, 제법 딱딱할 수도 있는 문장들을 부드럽게 바로잡아주신 박선희 편집자님께도 고맙다는 말 전하고 싶습니다. 꼭 책에 이름을 달아 달라던 듬직한 후배 고세욱, 그리고 제가 이 무거운 책을 번역해 낼 수 있게끔 힘이 되어 준 한살배기 아들 진형이와 그 어머니 되시는 분께도 감사와 사랑의 마음을 전합니다.

2008년 5월 1일
김석준

1장

Programming **Erlang**

출발

그만! 또 프로그래밍 언어라니! 또 다른 무언가를 배워야 하나? 이만하면 충분하지 않나?

여러분의 반응을 이해한다. 세상은 프로그래밍 언어로 넘쳐나는데, 왜 또 무언가를 배워야 하나?

그러나 여기 여러분이 얼랭(Erlang)을 배워야 하는 다섯 가지 이유가 있다.

- 멀티코어 컴퓨터에서 실행할 때 훨씬 빠르게 실행되는 프로그램을 작성하고 싶다.
- 서비스 중단 없이도 변경할 수 있는 무정지(fault-tolerant) 애플리케이션을 만들고 싶다.
- '함수형 언어'에 관해 들어 보았을 것이다. 그게 진짜로 작동하는지 궁금하다.
- 실제 대규모 산업용 제품에서 실전 테스트된, 방대한 라이브러리와 활발한 사용자 커뮤니티가 있는 언어를 사용하고 싶다.
- 많은 양의 코드 타이핑으로 손가락이 닳아 떨어지게 하고 싶지 않다.

이 모든 걸 할 수 있을까? 20.3절의 'SMP 얼랭 실행하기'에서 우리는 서른 두 개짜리 코어 컴퓨터에서 실행할 때 선형으로 속도가 오르는 몇몇 프로그램을 살펴볼 것이다. 18장 「OTP로 시스템 구축하기」에서는 여러 해 동안 24시간 내내 작동 중인 신뢰성이 아주 높은 시스템을 만드는 법을 살펴볼 것이다. 322쪽의 16.1절,

'제네릭 서버로 가는 길'에서는 서비스를 중단시키지 않고도 소프트웨어를 업그레이드할 수 있는 서버의 작성 기법에 대해 얘기할 것이다.

많은 곳에서 우리는 함수형 언어가 지닌 미덕에 대한 칭찬을 아끼지 않을 것이다. 함수형 언어는 부수 효과(side effect)가 있는 코드를 허락하지 않는다. 부수 효과와 병행성(concurrency)은 함께할 수 없다. 부수 효과가 있는 순차 코드는 가능하다. 또는 부수 효과로부터 자유로운 병행성 코드도 가능하다. 여러분은 선택을 해야 한다. 중간은 없다.

얼랭은 운영체제가 아닌 프로그래밍 언어에 병행성이 들어 있는 언어다. 얼랭은, 세상을 오로지 메시지를 주고받는 것만으로 상호 작용할 수 있는 병렬 프로세스들의 집합으로 모델링함으로써, 병렬 프로그래밍을 쉽게 해준다. 얼랭의 세상은 잠금(lock)도 동기화 메서드도 또한 공유 메모리를 위반할 가능성도 없는 병렬 프로세스들의 세상이다. 공유 메모리(shared memory)란 게 없기 때문이다.

얼랭 프로그램은 수천에서 수백만에 이르는 극단적으로 경량(lightweight)의 프로세스들로 만들 수 있는데, 이 프로세스들은 단일 프로세서(processor)에서 실행될 수도, 멀티코어 프로세서에서 실행될 수도, 또는 프로세서들의 네트워크에서 실행될 수도 있다.

1.1 로드맵

- 9쪽의 2장 「시작」은 간단한 '몸 풀기' 장이다.
- 37쪽의 3장 「순차 프로그래밍」은 순차 프로그래밍을 다룬 두 장 가운데 첫 장이다. 여기서는 패턴 매칭과 비파괴적 할당의 개념을 소개한다.
- 75쪽의 4장 「예외」에서는 예외 처리에 관해 다룬다. 어떤 프로그램도 오류로부터 자유로울 수는 없다. 이 장에서는 순차적 얼랭 프로그램에서 오류를 감지하고 처리하는 것을 살펴볼 것이다.
- 87쪽의 5장 「고급 순차 프로그래밍」은 순차 얼랭 프로그래밍을 다룬 두 번째 장이다. 여기서는 몇 가지 고급 주제와 함께 순차 프로그래밍의 나머지 내용을 다룬다.
- 125쪽의 6장 「프로그램 컴파일하고 실행하기」에서는 프로그램을 컴파일하고

실행하는 여러 가지 방법에 대해 논한다.

- 147쪽의 7장 「병행성」에서 우리는 주제를 바꾼다. 이 장은 비기술적인 내용을 다룬 장이다. 우리 프로그래밍 방식의 기저에 깃든 사상은 무엇인가? 우리는 세상을 어떻게 보는가?
- 151쪽의 8장 「병행 프로그래밍」은 병행성에 관한 장이다. 얼랭에서 병렬 프로세스는 어떻게 생성하는가? 프로세스들은 어떻게 통신하는가? 우리가 만든 병렬 프로세스들은 얼마나 빠른가?
- 173쪽의 9장 「병행 프로그램과 오류」에서는 병렬 프로그램에서의 오류에 관해 이야기한다. 프로세스가 실패하면 어떻게 되는지? 프로세스 실패는 어떻게 감지하며, 그럴 땐 무엇을 할 수 있는지?
- 191쪽의 10장 「분산 프로그래밍」에서는 분산 프로그래밍을 다룬다. 여기서는 여러 개의 작은 분산 프로그램들을 작성하고, 그것들이 얼랭의 노드 클러스터(cluster) 기반 분산 또는 소켓-기반 분산 형태를 사용하는 독립된 호스트에서 어떻게 실행되는지 볼 것이다.
- 209쪽의 11장 「IRC Lite」는 순수하게 애플리케이션에 관한 장이다. 여기서는 병행성과 소켓 기반 분산이라는 두 주제를 엮어서 우리의 그럴듯한 첫 애플리케이션이 될 미니(mini) IRC 클라이언트 서버 프로그램으로 만들어 볼 것이다.
- 233쪽의 12장 「인터페이스 기법」에서는 얼랭과 다른 언어 코드 간의 인터페이스를 망라한다.
- 249쪽의 13장 「파일 프로그래밍」에는 파일 프로그래밍에 관한 많은 예제가 들어 있다.
- 271쪽의 14장 「소켓 프로그래밍」에서는 소켓으로 프로그래밍하는 법을 설명한다. 우리는 얼랭으로 순차 그리고 병렬 서버를 만드는 법을 살펴볼 것이다. 이 장은 제법 규모 있는 우리의 두 번째 애플리케이션인 SHOUTcast 서버로 마무리 지을 것이다. 이 서버는 스트리밍 미디어 서버로, SHOUTcast 프로토콜을 사용하여 MP3 데이터를 스트리밍하는 데 사용할 수 있다.
- 301쪽의 15장 「ETS와 DETS- 대량 데이터 저장소 메커니즘」에서는 저수준(low-level)의 모델인 ets와 dets를 설명한다. ets는 매우 빠르고 파괴적인 메모리 기반의 해시 테이블 작업용 모듈이며, dets는 저수준 디스크 저장소 용도로 설계되었다.

- 327쪽의 16장 「OTP 개론」은 OTP에 대한 개론의 장이다. OTP는 얼랭으로 산업 규모의 애플리케이션을 구축하는 데 사용되는 일련의 얼랭 라이브러리와 연산 프로시저들의 집합이다. 이 장에서는 OTP의 중심 개념인 비헤이비어(behavior)의 개념을 소개한다. 비헤이비어를 사용하면, 문제의 비기능적인 측면은 비헤이비어 프레임워크가 해결하도록 하고, 우리는 구성요소의 기능적인 동작에 집중할 수 있다. 예를 들어, 애플리케이션을 무정지로 만들거나 또는 확장성 있게 만드는 부분은 프레임워크가 담당하고, 비헤이비어 콜백(callback)은 문제에서 특수한 부분에 집중하는 식이다. 이 장은 여러분만의 고유한 비헤이비어를 어떻게 만드는지 하는 일반적인 논의에서 출발하여, 얼랭 표준 라이브러리의 일부인 gen_server 비헤이비어에 대한 설명으로 이어질 것이다.
- 345쪽의 17장 「Mnesia - 얼랭 데이터베이스」에서는 얼랭 데이터베이스 관리 시스템(DBMS)인 Mnesia에 대해 얘기한다. Mnesia는 극도로 빠르고, 유연하며, 실시간의 응답 시간을 가지는 통합된 DBMS이다. Mnesia는 무정지 작업을 위해 데이터를 물리적으로 분리된 여러 노드에 복제하도록 구성할 수 있다.
- 369쪽의 18장 「OTP로 시스템 구축하기」는 OTP에 대한 두 번째 장이다. 여기서는 여러 가지를 한데 엮어 하나의 OTP 애플리케이션으로 구성하는 실용적인 측면을 다룬다. 실제 애플리케이션에는 작고 너저분한 많은 것들이 있으며, 이들은 일관된 방법으로 시작하고 종료해야 한다. 만약 그게 멎거나 또는 하위 구성요소가 멎으면, 그것들은 다시 시작되어야 한다. 그렇게 멎을 경우에 오류 로그가 필요하다. 그래야 그 사건 다음에 뭐가 일어났는지 알 수 있다. 이 장에는 완전하게 갖춰진 OTP 애플리케이션을 만드는 데 필요한 세부 사항들이 모두 나와 있다.
- 403쪽의 19장 「멀티코어 서곡」에서는 얼랭이 왜 멀티코어 컴퓨터 프로그램에 적합한지를 짤막하게 소개한다. 우리는 일반적인 용어로 공유 메모리 병행성과 메시지 전달 병행성에 대해 이야기한다. 또한 우리가 어째서 가변 상태(mutable state)가 없고 병행성을 가진 언어가 멀티코어 컴퓨터 프로그래밍에 이상적으로 적합하다고 강력히 확신하는지 설명한다.
- 405쪽의 20장 「멀티코어 CPU 프로그래밍」은 멀티코어 컴퓨터 프로그래밍에 관한 장이다. 여기서는 멀티코어 컴퓨터에서 얼랭 프로그램이 효율적으로 실

행되는 것을 보장해 주는 기법들에 대해 얘기하고 멀티코어 컴퓨터에서 순차 프로그램의 속도를 높이는 여러 가지 추상화도 소개한다. 마지막으로 우리는 몇 가지 측정을 수행하고 우리의 세 번째 주요 프로그램이 될 전문(full-text) 검색 엔진을 작성할 것이다. 이를 위해 우선 mapreduce라는 함수를 구현한다. 이 함수는 일련의 프로세싱 요소들 위에서 병렬 계산을 수행하도록 해주는 고차원 함수이다.

- 431쪽의 부록 A에서는 얼랭 함수들의 문서를 작성하는 데 사용하는 형(type) 체계를 설명한다.
- 437쪽의 부록 B에서는 윈도 운영체제에서 얼랭을 설정하는 법을 설명한다. (또한 모든 운영체제에서 이맥스(emacs)를 어떻게 구성하는지도 다룬다).
- 441쪽의 부록 C에는 얼랭 리소스들의 카탈로그가 나와 있다.
- 447쪽의 부록 D에서는 lib_chan을 설명하는데, 이것은 소켓-기반 분산 프로그래밍에 사용하는 라이브러리다.
- 465쪽의 부록 E에서는 여러분의 코드를 분석하고, 프로파일링하고, 디버깅하고, 추적하는 기법들을 살펴본다.
- 489쪽의 부록 F에는 얼랭 표준 라이브러리에서 자주 사용되는 모듈들을 한 줄로 요약해 넣었다.

1.2 다시 출발

옛날에 어떤 프로그래머가 재미있는 프로그래밍 언어를 설명해 놓은 책을 발견하였다. 낯선 문법에, 등호는 같음을 의미하지 않았고, 변수는 변함이 허용되지 않았다. 게다가 객체지향도 아니었다. 그 프로그램은, 그러니까, 달랐다……

프로그램만 달랐던 게 아니라 프로그래밍에 접근하는 방식도 대체로 달랐다. 저자는 계속해서 병행성과 분산 그리고 무정지를 말했고, 그게 무슨 의미가 되었건, 병행성-지향 프로그래밍이라는 프로그래밍 방법에 대해 주절거렸다.

그래도 몇몇 예제는 재미있어 보였다. 그날 저녁 그 프로그래머는 채팅 예제 프로그램을 살펴보았다. 비록 문법은 다소 이상했을지언정, 그것은 아주 작았고 이해하기 쉬웠다. 그렇게 쉬울 수가 없었다.

기본 프로그램은 간단했다. 거기에 코드를 몇 줄 더하자 파일 공유라든가 암호화된 대화도 가능해 졌다. 프로그래머는 타이핑하기 시작했다……

이게 도대체 뭐야?

그것은 병행성에 관한 것이다. 또 분산에 관한 것이다. 그리고 무정지에 관한 것이다. 그것은 함수형 언어에 관한 것이다. 그것은 잠금과 뮤텍스 없이 오직 순수하게 메시지 전달만을 사용하여 분산된 병행 시스템을 프로그래밍하는 법에 관한 것이다. 그것은 멀티코어 CPU에서 여러분 프로그램의 속도를 증가시키는 데 관한 것이다. 그것은 사람들이 서로 상호 작용할 수 있는 분산된 애플리케이션의 작성에 관한 것이다. 그것은 병행성을 모델링하고 그 모델을 컴퓨터 프로그램과 맵핑하는 부분에 관한 것으로, 바로 내가 병행성-지향 프로그래밍(concurrency-oriented programming)이라 부르는 절차에 관한 것이다.

나는 이 책을 쓰는 내내 즐거웠다. 여러분도 이 책을 읽으면서 즐겁길 바란다.

이제 책을 읽고 코드를 작성하고 즐기자.

1.3 감사의 말

여러 사람이 이 책을 작성하는 데 도움을 주었다. 이 자리를 빌려 그들 모두에게 감사를 전한다.

우선 내 편집자인 데이브 토머스(Dave Thomas)에게 감사의 마음을 전한다. 데이브는 내게 글 쓰는 것을 가르쳐 주었고 끝없는 질문의 포화를 던져 주었다. 왜 이런가? 왜 저런가? 내가 책을 시작했을 때, 데이브는 내 글쓰기 스타일이 '암상에서 설교하는 것' 같다고 말했다. 그는 이렇게 충고했다. "전 당신이 사람들과 얘기를 했으면 좋겠어요. 설교가 아니구요." 그 덕에 책이 훨씬 나아졌다. 데이브에게 감사한다.

그리고 내 뒤에는 작은 언어 전문가 위원회가 있었다. 그들은 내가 무얼 빼야 하는지 정하고, 설명하기 어려운 몇 가지를 명료하게 하는 데 도움을 주었다. 이 자리를 빌려 Björn Gustavsson, Robert Virding, Kostis Sagonas, Kenneth Lundin, Richard Carlsson 그리고 Ulf Wiger에게 감사를 전한다.

Mnesia에 대해 소중한 조언을 해준 Claes Vikström와 SMP 얼랭 부분에 큰 도움을 준 Rickard Green, 그리고 텍스트 색인 프로그램에서 사용한 스테밍(stemming) 알고리즘에 도움을 준 Hans Nilsson에게도 감사를 전한다.

Sean Hinde와 Ulf Wiger 덕분에 나는 OTP의 다양한 내부를 어떻게 사용하는지 이해할 수 있었고, Serge Aleynikov은 내가 능동형(active) 소켓을 이해할 수 있게 설명해 주었다.

Helen Taylor(내 아내)는 여러 장을 교정해 주었다. 차 생각이 간절할 때마다 그녀가 타준 차만도 수백 잔은 될 것이다. 거기에 더해, 지난 일곱 달 동안 다소 강박적인 내 행동을 묵묵히 참아 주었다. 또한 Thomas와 Claire에게도 감사한다. 그리고 Bach와 Handel, Zorro와 Daisy 그리고 Doris에게도 감사한다. 이들이 있었기에, 나는 평온하게 작업할 수 있었다. 지칠 때마다 어루만져 주었으며, 나를 제자리로 이끌어준 그들에게 감사의 마음을 전한다.

마지막으로, 정오표를 채워준 베타판의 모든 독자에게 감사의 마음을 전한다. 나는 여러분이 원망스럽기도 했고 또 고맙기도 했다. 첫 베타판이 나왔을 때, 이틀만에 책을 다 읽고 모든 페이지를 코멘트로 채워 줄 줄은 몰랐었다. 그렇지만 그 과정이 있었기에 내 예상보다 훨씬 더 나은 책이 나올 수 있었다. (여러 차례에 걸쳐) 수십 명의 사람들이 "이 내용은 이해할 수 없어요."라고 지적했기에, 나는 다시 생각해 보게 되었고 관련된 자료를 새로 작성할 수 있었다. 여러분 모두의 도움에 감사드린다.

조 암스트롱(Joe Armstrong)

2007년 5월

2장

Programming **Erlang**

시작

2.1 개관

모든 배움의 과정이 그렇듯, 얼랭 역시 마스터하기까지 여러 단계를 거치게 된다. 이제부터 이 책에서 다룰 단계들과 그 속에서 우리가 겪게 될 경험을 살펴보자.

1단계. 잘 모르겠는데요

초보자인 여러분은, 시스템은 어떻게 시작하며, 셸에서 명령은 어떻게 실행하고, 간단한 프로그램은 어떻게 컴파일하는지 배울 것이다. 그러면서 얼랭과도 점점 친해질 것이다(얼랭은 작은 언어이니 오래 걸리지는 않을 것이다).

이러한 과정을 조금 더 세세하게 나눠 다음과 같이 정리할 수 있다.

- 컴퓨터에 작동하는 얼랭 시스템이 있는지 확인하기
- 얼랭 셸을 시작하고 종료하는 법 배우기
- 셸에 식을 입력하고, 식을 계산하고, 결과를 해석하는 법 익히기
- 선호하는 텍스트 편집기를 사용하여 프로그램을 만들고 수정하는 방법 익히기
- 셸에서 프로그램을 컴파일하고 실행하는 연습하기

2단계. 얼랭과 친해졌어요

지금쯤이면 여러분은 언어에 대한 실용적인 지식이 생겼을 것이다. 언어를 사용하면서 생기는 문제들은 5장 「고급 순차 프로그래밍」(87쪽)을 보면 된다.

이 단계에서 여러분은 얼랭과 친숙해질 것이고, 우리는 조금 더 흥미로운 주제로 옮겨갈 것이다.

- 좀 더 고급의 셸 사용법을 익힐 것이다. 이를 통해 여러분이 처음 셸을 배웠을 때보다 더 많은 것을 할 수 있다(예를 들면, 이전의 식을 불러와 편집할 수도 있다. 이 부분은 6.5절 '얼랭 셸에서 명령 편집하기'(138쪽)에서 다룬다).
- 라이브러리(얼랭에서 '모듈(module)'이라 부르는)에 대한 학습을 시작한다. 내가 작성한 프로그램은 대부분 lists, io, file, dict, gen_tcp, 이렇게 다섯 모듈로 작성할 수 있다. 따라서 우리는 이 책 전반에 걸쳐 이 모듈들을 많이 사용할 것이다.
- 프로그램이 점점 커지면 컴파일과 실행을 자동화하는 법을 알 필요가 생긴다. 이를 위해 선택한 도구가 make다. 우리는 makefile을 작성하여 프로세스를 통제하는 법을 볼 것이다. 이 부분은 6.4절 'Makefiles로 컴파일 자동화하기'(135쪽)에서 다룬다.
- 얼랭 프로그래밍의 조금 더 큰 세상에서는 OTP라는 광범한 라이브러리 모음을 사용한다.[1] 얼랭 프로그래밍의 경험이 쌓일수록 OTP를 이용하면 시간을 많이 절약할 수 있음을 알게 될 것이다. 누군가 우리가 필요로 하는 기능을 이미 만들어 두었다면, 바퀴를 다시 발명할 필요가 어디 있을까? 우리는 주요 OTP 비헤비어(behavior)들, 특히 gen_server에 대해 학습할 것이다. 이것은 16.2절 'gen_server 시작하기'(332쪽)에서 다룬다.
- 얼랭을 주로 사용하는 경우 중 하나는 분산 프로그램을 작성할 때이며, 이제 우리는 그것을 실습해 볼 것이다. 10장 「분산 프로그래밍」(191쪽)에 나온 예제로 시작하고, 그걸 여러분이 원하는 방식으로 확장할 수 있다.

단계 2.5 부가적인 몇 가지를 배워야 할까 봐요

처음부터 이 책의 모든 장을 읽을 필요는 없다.

여러분이 이전에 봤던 대부분의 언어와는 다르게, 얼랭은 병행 프로그래밍 언어이며, 특히 분산 프로그램을 작성하거나 현대적인 멀티코어나 SMP[2] 컴퓨터 프로

1 오픈 텔레콤 플랫폼(Open Telecom Platform).

그램에 적합하다. 얼랭 프로그램은 대부분 멀티코어 또는 SMP 머신 위에서 실행하는 것만으로도 훨씬 빨라질 것이다.

얼랭으로 프로그래밍하는 것은 내가 '병행성 지향 프로그래밍(concurrency-oriented programming, COP)'이라 이름 붙인 프로그래밍 패러다임을 사용하여 프로그래밍하는 것이다.

COP를 사용할 경우, 여러분은 문제를 쪼개고 그 해법에 담긴 자연적인 병행성을 식별하게 된다. 이것은 어떤 병행 프로그램을 작성하든 거쳐야 할 첫 번째 단계다.

단계 3. 나는야, 얼랭 마스터

이제 여러분은 언어를 마스터했으며 몇몇 유용한 분산 프로그램을 작성할 수 있다. 그러나 진정한 마스터가 되려면 아직 더 배워야 할 게 있다.

- Mnesia. 얼랭 배포판에는 Mnesia라는 고성능의 데이터베이스가 내장되어 있다. 이 데이터베이스는 다중화(replication)를 지원한다. 이것은 원래 성능과 무정지성(fault tolerance)이 필수인 통신 애플리케이션용으로 설계되었다. 오늘날에는 통신용 애플리케이션이 아니더라도 광범하게 사용된다.
- 다른 프로그래밍 언어로 작성된 코드와 인터페이스하는 방법과 링크인 드라이버(linked-in driver)를 사용하는 방법은 12.4절 '링크인 드라이버'(243쪽)에서 다룬다.
- 슈퍼비전 트리를 구축하고, 스크립트를 시작하는 등 OTP 비헤이비어에 대한 완전한 사용법은 18장 「OTP로 시스템 구축하기」(369쪽)에서 다룬다.
- 멀티코어 컴퓨터에서 프로그램을 실행하고 최적화하는 방법은 20장 「멀티코어 CPU 프로그래밍」(405쪽)에서 다룬다.

가장 중요한 교훈

이 책 전반에 걸쳐 여러분이 기억해야 할 규칙이 하나 있다. 바로 프로그래밍은 재미있어야 한다는 것이다. 개인적으로 나는 판에 박힌 순차 애플리케이션을 프로그래밍하는 쪽보다는 채팅 프로그램이나 인스턴트 메시징 같은 분산 애플리케이

2 대칭적 다중처리(symmetric multiprocessing).

션을 프로그래밍하는 쪽이 더 재미있다고 생각한다. 컴퓨터 한 대로 할 수 있는 일에는 한계가 있지만, 네트워크로 연결된 컴퓨터들로 할 수 있는 일은 무한하다. 얼랭은 네트워크 기반 애플리케이션을 실험하고 제품 수준의 시스템을 구축하는 데 이상적인 환경을 제공한다.

출발을 돕고자, 나는 기술적인 장들 중간 중간에 실세계 애플리케이션을 몇몇 섞어 두었다. 여러분은 이 애플리케이션들을 실습의 시작점으로 삼을 수 있을 것이다. 가져다 고치고 내가 상상조차 못한 걸로 만들어 배포해 달라. 그러면 나는 매우 행복할 것이다.

2.2 얼랭 설치하기

무얼 하든 그 전에 여러분은, 여러분 시스템에 작동하는 버전의 얼랭이 있는지 확인해야 한다. 명령 프롬프트로 가서 erl을 쳐보자.

```
$ erl
Erlang (BEAM) emulator version 5.5.2 [source] ... [kernel-poll:false]

Eshell V5.5.2 (abort with ^G)
1>
```

윈도 시스템이라면, 얼랭을 설치한 다음 PATH 환경 변수가 프로그램을 가리키도록 변경해야만 erl 명령이 먹힌다. 표준적인 방법으로 프로그램을 설치했다면, 시작 〉 프로그램 〉 Erlang OTP 메뉴를 통해 얼랭을 호출할 수 있을 것이다. 부록 B(391쪽)에는 내가 MinGW 및 MSYS와 함께 얼랭을 실행했던 방법이 나와 있다.

노트- 배너, 즉 "Erlang (BEAM) ..... (abort with ^G)"라고 적힌 문구는 가끔씩만 보일 것이다. 그냥 보여주는 것이니 이걸 보고 걱정하거나 무엇인지 의아할 필요는 없다. 특별히 관련 있는 경우가 아니라면 대부분 예제에서 생략하겠다.

셸 배너가 보인다면 얼랭이 여러분 시스템에 설치된 것이다. 거기서 빠져나오자(Ctrl+G를 누르고 이어서 문자 Q를 입력한 뒤 엔터 또는 리턴을 누르라).[3] 이제

3 또는 셸에서 q() 명령을 내리자.

2.3절 '이 책의 코드'(15쪽)로 건너뛰자.

그러지 않고, erl은 모르는 명령이라는 오류가 났다면, 여러분의 박스에 얼랭을 설치해야 한다. 그 말은 곧 결정의 순간이 되었다는 뜻이다. 미리 빌드된 바이너리 배포판을 사용할 것인가, 패키지 배포판을 사용할 건가(OS X에서), 또는 소스로부터 얼랭을 빌드할 건가, 아니면 CEAN(Comprehensive Erlang Archive Network)을 사용할 것인가?

바이너리 배포판

얼랭 바이너리 배포판은 윈도와 리눅스 기반 운영체제에서 사용할 수 있다. 바이너리 시스템의 설치 방법은 시스템에 상당히 의존적이기 때문에, 여기서는 시스템별로 살펴보기로 한다.

윈도

http://www.erlang.org/download.html에서 릴리스 목록을 찾자. 가장 최근 버전을 선택하고, 윈도 바이너리 링크를 클릭하자. 이 링크는 윈도 실행 파일과 연결된다. 링크를 클릭하고 지시를 따르라. 이건 표준적인 윈도 설치 방법이라서 별 문제 없을 것이다.

리눅스

데비안(Debian) 기반 시스템을 위한 바이너리 패키지가 존재한다. 데비안 기반 시스템이라면, 다음 명령을 주자.

```
> apt-get install erlang
```

맥 OS X에 설치하기

혹 맥 사용자라면 맥포트(MacPorts) 시스템을 사용하여 미리 빌드된 버전의 얼랭을 설치할 수 있다. 또는 소스로부터 얼랭을 설치할 수도 있다. 맥포트는 사용하기 쉬운데다가 지속적으로 업데이트도 처리해줄 것이다. 그러나 맥포트를 사용해 설치한 얼랭은 최신 릴리즈가 아닐 수도 있다. 일례로, 이 책을 처음 작성할 무렵 맥포트 버전의 얼랭은 현재 버전보다 두 릴리스 이전의 것이었다. 이런 까닭에, 나는 다음 절에 나와 있는 대로 과감하게 소스로부터 얼랭을 설치하길 권한다. 이를 위

해서는 개발자 도구가 설치되어 있는지 확인해야 한다(여러분 머신과 함께 온 소프트웨어 DVD에 들어 있다).

소스로부터 얼랭 빌드하기

바이너리 설치 말고 다른 방법은 소스로부터 얼랭을 빌드하는 것이다. 윈도 시스템이라면 새 릴리스가 될 때마다 윈도 바이너리와 모든 소스를 완전히 제공하기 때문에 이렇게 할 이유가 없다. 그러나 맥과 리눅스 플랫폼에서는, 새로운 얼랭 배포판이 릴리스되는 시점과 바이너리 설치 패키지를 사용할 시점 사이에 약간 간격이 있을 수 있다. 유닉스 기반이라면 어떠한 OS에서도 설치 방법은 동일하다.

1. 최신 얼랭 소스를 가져온다.[4] 소스는 otp_src_R11B-4.tar.gz 같은 식으로 이름이 붙은 파일로 존재할 것이다(지금 이 파일에는 얼랭 버전 11의 네 번째 유지관리 릴리스가 들어 있다).
2. 다음과 같이 풀고, 설정하고, make하고, 설치하자.

```
$ tar -xzf otp_src_R11B-4.tar.gz
$ cd otp_src_R11B-4
$ ./configure
$ make
$ sudo make install
```

노트 - 시스템을 빌드하기 전에 가용한 구성설정 옵션들을 검토하려면 ./configure --help 명령을 사용한다.

CEAN을 이용하기

포괄적 얼랭 아카이브 네트워크(Comprehensive Erlang Archive Network, CEAN)는 공통된 설치기를 기반으로 모든 주요 얼랭 애플리케이션을 한곳에 모으려는 시도다. CEAN을 사용하면 이를 통해 기본적인 얼랭 시스템뿐 아니라 얼랭으로 작성된 많은 수의 패키지들도 관리할 수 있다는 장점이 있다. 즉, 기본적인 얼랭 설치뿐만 아니라 여러분의 패키지들도 최신으로 유지할 수 있다는 뜻이다.

CEAN에는 다양한 운영체제와 프로세서 아키텍처에 맞춰 미리 컴파일된 바이

4 http://www.erlang.org/download.html