Hanbit RealTime 117

Early Release
UNFINISHED

임백준의 아카 시작하기

Akka 개념 잡기

임백준 지음

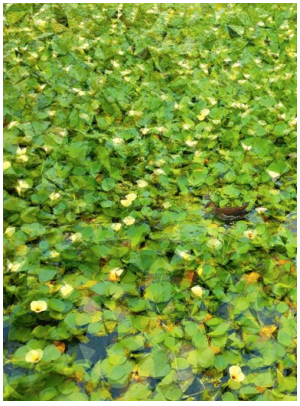
한빛미디어 Hanbit Media, Inc.

임백준의 아카 시작하기

Akka 개념 잡기

임백준 지음

한빛미디어 Hanbit Media, Inc.



표지 사진 **신영호**

이 책의 표지는 신영호님이 보내 주신 풍경사진을 담았습니다.
리얼타임은 독자의 시선을 담은 풍경사진을 책 표지로 보여주고자 합니다.

사진 보내기 **ebookwriter@hanbit.co.kr**

임백준의 아카 시작하기 Akka 개념 잡기

초판발행 2015년 10월 29일

지은이 임백준 / **펴낸이** 김태헌
펴낸곳 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부
전화 02-325-5544 / **팩스** 02-336-7124
등록 1999년 9월 30일 제10-1779호
ISBN 978-89-6848-789-7 15000 / **정가** 11,000원

총괄 배용석 / **책임편집** 김창수
디자인 표지/내지 여동일, 조판 최송실
마케팅 박상용 / **영업** 김형진, 김진불, 조유미

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주십시오.
한빛미디어 홈페이지 www.hanbit.co.kr / **이메일** ask@hanbit.co.kr

Published by HANBIT Media, Inc. Printed in Korea

지금 하지 않으면 할 수 없는 일이 있습니다.
책으로 펴내고 싶은 아이디어나 원고를 메일(ebookwriter@hanbit.co.kr)로 보내주세요.
한빛미디어(주)는 여러분의 소중한 경험과 지식을 기다리고 있습니다.

저자 소개

한빛미디어에서 『나는 프로그래머다 2015』, 『폴리글랏 프로그래밍』, 『누워서 읽는 퍼즐북』, 『프로그래밍은 상상이다』, 『뉴욕의 프로그래머』, 『임백준의 소프트웨어 산책』, 『나는 프로그래머다』, 『누워서 읽는 알고리즘』, 『행복한 프로그래밍』을 출간했고, 로드북에서 『프로그래머 그 다음 이야기』를 출간했다. 현재 월스트리트에서 자바, C#, 스칼라 언어를 이용해 금융 관련 소프트웨어를 개발하고 있으며, 뉴저지에서 아내, 두 딸과 함께 살고 있다. 개발자들을 위한 유쾌한 팟캐스트 〈나는 프로그래머다〉(http://iamprogrammer.io)의 MC로 활동 중이다.

이 책은 아카 입문서다. 아카를 한 번도 사용해본 적이 없는 사람이 최대한 빠른 속도로 아카를 이용하여 코딩을 시작하게 하는 것이 목적이다. 그런 의미에서 매우 실전적인 책이라고 말할 수 있다. 이 책에는 아카가 제공하는 개념과 API가 모두 포함되어 있지 않다. 이론적인 설명은 최대한 줄이고 코드에 집중했기 때문이다. 책에 포함된 코드를 돌려보면 아카의 핵심적인 개념이 저절로 익숙해지도록 책을 구성했다.

이 책을 쓰는 시점에서 나는 맨해튼에 있는 작은 스타트업 회사에서 일하고 있다. 구글 애드워즈AdWords와 비슷한 서비스를 제공하는 회사인데, 하루 평균 10억 개 정도의 HTTP 요청을 받는 서버 클러스터를 담당하고 있다. 서버 클러스터는 원래 톰캣Tomca 기반이었는데, 그것을 아카 기반으로 바꾸는 프로젝트를 진행했다. 프로젝트를 처음 시작하던 무렵에 기존 팀원들을 대상으로 몇 차례 아카를 강의했고, 각자 노트북을 들고 오라고 해서 코딩실습 시간도 가졌다.

그때 강의를 들은 팀원들이 간단한 코드를 통해서 연습하니 아카에 대한 문서만 읽는 것에 비해서 훨씬 빠르고 정확하게 개념을 이해할 수 있었다는 피드백을 주었다. 강의를 진행한 입장에서 반갑고 고무적인 이야기였다. 이 책에서 사용한 소스코드는 대부분 그때 사용한 것이다. 한국에서도 최근 아카에 대한 관심이 높아지고 있다는 이야기를 듣고 이 경험을 책으로 만들어서 공유하고 싶다는 생각을 하게 되었다.

아카는 현재 타입세이프Typesafe의 공동창업자이자 CTO인 요나스 보네어Jonas Boner가 스칼라 언어를 이용해 개발했다. 스칼라가 초기에는 액터 모델을 구현한 별도의 라이브러리를 가지고 있었는데, 보네어가 개발한 아카의 성능과 안정성이 더 뛰어난 것을 확인하고 아카를 스칼라 언어의 공식 라이브러리로 채택했다. 또한, JVM 환경에서는 자바를 사용하는 개발자가 다수이기 때문에 스칼라로 작성된 아카 위에 얇은 자바 API를 입혀놓기도 했다. 나 역시 이 책을 쓰면서 자바 개발자를 염두에 두었기 때문에 책에 포함된 소스코드는 모두 자바를 사용했다.

아카가 구현한 액터 모델actor model은 오래전인 1973년에 칼 휴이트Carl Hewitt가 제안한 수학적 모델을 기초로 삼고 있다. 멀티스레딩 환경에서 동시성 프로그램을 작성하는 일이 점점 어려워지고, 한 대의 컴퓨터가 사용하는 CPU 코어의 수가 빠른 속도로 늘어나는 요즘에, 아카는 동시성 코드를 작성하기 위한 직관적이고 편리한 프로그래밍 모델을 제공한다.

액터 모델이 설명하는 내용은 그 자체로는 간단하기 때문에 누구나 쉽게 이해할 수 있다. 하지만 원리를 실제 코드에 적용시키는 것은 차원이 다른 문제다. 아카를 사용하는 프로젝트를 진행하면서 머리로 이해한 개념을 실전에 적용하지 못해서 애를 먹는 사람을 많이 보았다. 그럴 수밖에 없는 것이, 아카는 몇몇 개념을 이해하면 익힐 수 있는 게 아니라 근본적인 패러다임의 전환을 받아들일 때 비로소 자기 것으로 만들 수 있기 때문이다. 그런 면에서 아카 프로그래밍은 언뜻 생각하기보다 쉽지 않다.

이러한 패러다임 전환의 내용은 다음과 같다.

- 객체의 메서드를 직접 호출할 수 없고, 오직 메시지를 전달할 수 있을 뿐이다.
- 모든 것이 비동기적asynchronous이다.
- 블로킹blocking이 일어나면 안 된다.
- 모든 것이 동시적concurrent이다.

객체지향 개념의 진정한 출발점이라고 일컬어지는 스몰토크Smalltalk를 개발할 때 앨런 케이는 오늘날 우리가 아카를 통해서 보는 방식을 꿈꾸었다. 모든 것이 독립적인 객체고, 객체들이 메시지를 주고받는 방식으로 상호작용하는 세계가 그것이다. 하지만 우리에게 친숙하게 다가온 객체지향 언어는 C++였고, 이후에 자바나 C# 같은 언어가 뒤를 이었다. C++, 자바, C# 등의 언어에서 객체가 상호작용을 하는 방식은 서로의 메서드를 직접 호출하는 것이다. 메서드를 호출하면 이것이 리턴할 때까지 기다려야 하므로 이는 동기적synchronous이고 블로킹을 야기한다.

아카를 사용해 프로그래밍하는 것은 이러한 동기적 세계를 떠나서 비동기적 세계로 들어가는 것을 의미한다. 아카에서는 모든 것이 철저하게 비동기적이다. 동기적인 메서드 호출에 익숙한 사람들에게 이것은 심오한 패러다임 전환이다. 순차적으로 질서정연하게 이루어지던 작업이 모두 해체되어 각각의 유닛이 동시다발적으로 동작하는 세계는 기존의 방식으로 사고reasoning하기 어렵다. 이러한 패러다임 전환에 도전하는 사람은 코딩을 할 때 자기가 알고 있던 접근방식이 뿌리부터 흔들리는 전율을 맛보게 된다. 지적욕구가 충만한 프로그래머에게는 물론 즐겁고 유쾌한 전율이다.

아카는 강력하다. 아카가 동작하는 원리를 알지 못하는 사람에게 그것은 때로 마법처럼 보인다. 아카는 이미 수많은 회사에서 사용하고 있는 중이며, 스파크Apache Spark를 비롯한 수많은 오픈소스 라이브러리에서 활용되고 있다. 아카는 모든 문제를 해결해주지 않지만, 분산 컴퓨팅, 확장성, 동시성, 반응형reactive과 같은 요소를 고려할 때 훌륭한 선택이 되고 있다. 타입세이프에서 주장하는 반응형 플랫폼reactive platform에서 아카는 스파크, 플레이Play와 함께 3대 요소의 자리를 차지하고 있으며, 액터 모델은 STMSoftware Transaction Memroy과 함께 차세대 동시성 프로그래밍 구조물로 손꼽히는 방법이기도 하다.

한국에서도 아카에 대한 관심이 많이 일어나고 있다. 아직은 아카를 실전 코드에 활용하기보다는 실험적으로 접근하는 곳이 많지만, 가까운 장래에 아카에 대한 관심이 폭발적으로 일어날 거라고 확신한다. 아직은 초기라서인지 국내에서는 아카에 대한 좋은 입문서를 찾아보기 어렵다. akka.io에 있는 문서가 자세한 내용을 담고 있기는 하지만 영문으로 되어 있고, 개념을 중심으로 설명을 하기 때문에 아카를 처음 접하는 사람들이 활용하기에는 불편하다.

그런 의미에서 이 책은 아카를 처음 공부하려는 한국의 개발자에게 점프스타트 입문서의 역할을 수행할 수 있을 것이다. 아카를 사용하는 사람이 아직 충분히 많지 않은

지금, 이 책을 읽은 독자들이 열심히 아카를 공부해서 새로운 패러다임의 선구자가 되기를 희망한다.

아카를 소개하는 책을 출간할 수 있도록 선뜻 동의해준 한빛미디어 김태헌 사장님과 김창수 팀장께 고마움을 전한다. 그리고 아카의 세계에 발을 들여놓은 여러분을 진심으로 환영한다. 원래 아카는 라포니아Laponia라는 북부 스웨덴 지역에 있는 아름다운 산의 이름이다. 이제 아카에 발을 들여놓은 그대의 눈앞에 지금까지 경험해보지 못한 환상적인 절경이 펼쳐질 거라는 점을 의심치 않는다.

소프트웨어

이 책에서 사용하는 예제를 실행하려면 다음 소프트웨어가 필요하다.

- 자바 JDK 1.8 버전 이상
- 메이븐 최근 버전
- 아카 2.3.9
- IDE (이클립스나 인텔리제이)

이 책을 쓰는 동안 다음 소프트웨어와 버전을 사용했다. 이러한 소프트웨어를 설치하는 방법에 대해서는 별도로 설명하지 않겠다.

- 윈도우 7
- 자바 JDK 1.8.0_25
- 메이븐 3.1.0
- 아카 2.3.9
- 이클립스 루나 4.4.1

소스코드

이 책에서 설명하는 예제 코드는 모두 깃헙에 공개되어 있다. 다음 사이트를 방문하면 실행 가능한 소스코드를 다운로드할 수 있다. 코드의 자세한 내용은 책의 본문에서 설명한다.

- https://github.com/baekjunlim/AkkaStarting

아카 문서

아카에 대한 책은 다수가 출간되어 있지만 http://akka.io/에 있는 문서를 읽는 것이 가장 좋다. 아카를 사용하는 개발자라면 반드시 akka.io에 있는 문서를 읽고 내용을 숙지해야 한다. 자바 API와 스칼라 API를 모두 포함하고 있으므로 본인이 사용하는 언어에 맞는 내용을 선택해서 읽으면 된다.

아카는 스칼라 언어를 이용해 작성되었다. 하지만 자바 언어를 사용하는 개발자의 수가 훨씬 많아서 자바 개발자들도 아카를 사용할 수 있게 하기 위해 스칼라 코드 위에 자바 API를 입혀놓았다. 이 책에서 사용한 언어는 자바고, 따라서 아카가 제공하는 자바 API를 사용했다.

이 책의 내용이 아카 문서가 설명하고 있는 기능을 모두 포함하지 않는다는 사실을 기억할 필요가 있다. 이 책의 목적은 아카에 포함되어 있는 모든 기능과 개념을 설명하는 것이 아니다. 이 책은 여러분의 손이 지금 당장 키보드를 두드리도록 만들고, 그렇게 하는 동안 자연스럽게 추상적인 개념을 익히도록 의도되었다.

한빛 리얼타임

한빛 리얼타임은 IT 개발자를 위한 전자책입니다.

요즘 IT 업계에는 하루가 멀다 하고 수많은 기술이 나타나고 사라져 갑니다. 인터넷을 아무리 뒤져도 조금이나마 정리된 정보를 찾기도 쉽지 않습니다. 또한, 잘 정리되어 책으로 나오기까지는 오랜 시간이 걸립니다. 어떻게 하면 조금이라도 더 유용한 정보를 빠르게 얻을 수 있을까요? 어떻게 하면 남보다 조금 더 빨리 경험하고 습득한 지식을 공유하고 발전시켜 나갈 수 있을까요? 세상에는 수많은 종이책이 있습니다. 그리고 그 종이책을 그대로 옮긴 전자책도 많습니다. 전자책에는 전자책에 적합한 콘텐츠와 전자책의 특성을 살린 형식이 있다고 생각합니다.

한빛이 지금 생각하고 추구하는, 개발자를 위한 리얼타임 전자책은 이렇습니다.

1 eBook First – 빠르게 변화하는 IT 기술에 대해 핵심적인 정보를 신속하게 제공합니다

500페이지 가까운 분량의 잘 정리된 도서(종이책)가 아니라, 핵심적인 내용을 빠르게 전달하기 위해 조금은 거칠지만 100페이지 내외의 전자책 전용으로 개발한 서비스입니다. 독자에게는 새로운 정보를 빨리 얻을 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내는 선배, 전문가, 고수 분에게는 좀 더 쉽게 집필할 수 있는 기회가 될 수 있으리라 생각합니다. 또한, 새로운 정보와 지식을 빠르게 전달하기 위해 O'Reilly의 전자책 번역 서비스도 하고 있습니다.

2 무료로 업데이트되는 전자책 전용 서비스입니다

종이책으로는 기술의 변화 속도를 따라잡기가 쉽지 않습니다. 책이 일정 분량 이상으로 집필되고 정리되어 나오는 동안 기술은 이미 변해 있습니다. 전자책으로 출간된 이후에도 버전 업을 통해 중요한 기술적 변화가 있거나 저자(역자)와 독자가 소통하면서 보완하여 발전된 노하우가 정리되면 구매하신 분께 무료로 업데이트해 드립니다.

3 독자의 편의를 위해 DRM-Free로 제공합니다

구매한 전자책을 다양한 IT 기기에서 자유롭게 활용할 수 있도록 DRM-Free PDF 포맷으로 제공합니다. 이는 독자 여러분과 한빛이 생각하고 추구하는 전자책을 만들어 나가기 위해 독자 여러분이 언제 어디서 어떤 기기를 사용하더라도 편리하게 전자책을 볼 수 있도록 하기 위함입니다.

4 전자책 환경을 고려한 최적의 형태와 디자인에 담고자 노력했습니다

종이책을 그대로 옮겨 놓아 가독성이 떨어지고 읽기 어려운 전자책이 아니라, 전자책의 환경에 가능한 한 최적화하여 쾌적한 경험을 드리고자 합니다. 링크 등의 기능을 적극적으로 이용할 수 있음은 물론이고 글자 크기나 행간, 여백 등을 전자책에 가장 최적화된 형태로 새롭게 디자인하였습니다.

앞으로도 독자 여러분의 충고에 귀 기울이며 지속해서 발전시켜 나가겠습니다.

차례

chapter 1

아카에 대하여

내가 아카를 처음 사용한 것은 2013년 봄이었다. 모건스탠리에서 헤지펀드를 비롯한 클라이언트들에게 투자기술을 제공하는 부서에서 근무하던 때였다. 수백 개의 헤지펀드 중에서 어느 펀드가 갑자기 거래를 중단하는 일이 발생하면 그들이 맡겨놓은 돈을 돌려주기 위한 자금을 어떻게 충당할 것인가를 미리 예측하는 프로그램이 있었다. 2008년 금융 위기 이후에 미국 정부는 모든 은행에게 이러한 계산을 수행한 결과를 매일 제출하도록 요구했다.

정부가 요구하는 리포트를 만들기 위해서 프로그램은 다양한 방식으로 펀드를 선택했다. 일정한 규칙에 따라서 1개, 10개, 100개 등 여러 펀드를 선택해 그들이 당장 거래를 중단했을 때 (혹은 파산했을 때) 그것이 은행의 자금사정에 어떤 영향을 끼치는지를 정밀하게 분석해서 계산했다. 한 번 계산하고 마는 것이 아니라 선택된 모형마다 시나리오를 1,000,000번씩 실행해 정규분포를 만들어내는 몬테카를로 프로그램이었다.

누군가 자바를 이용해 만들어놓은 이 프로그램은 자바 스레드와 Executor Service를 이용하는 방식으로 작성되어 있었다. 한 번 동작을 시작하면 결과를 내놓기까지 대략 6시간이 걸렸다. 시간을 단축할 필요가 있었다. 비즈니스 요구사항에 따라서 다양한 시나리오를 적용해보고 싶은데 시간이 장애물이었기 때문이다. 이런 경우에 가장 손쉽게 떠올릴 수 있는 방법은 계산해야 하는 시나리오를 분할해 여러 대의 컴퓨터에 분산되어 있는 프로그램에 할당하는 것이다.

하지만 프로그램이 스케일 업scale-up은 가능해도 스케일 아웃scale-out은 가능하지 않은 방식으로 작성된 게 문제였다. 스케일 업은 컴퓨터에 CPU와 메모리를 추가해서 성능을 높이는 방식을 의미하고, 스케일 아웃은 별도의 컴퓨터를 추가해 병렬처리를 수행하는 것을 의미한다. 아무튼, 프로그램의 실행속도를 줄이기 위해 자원을 추가하는 것이 상식적인 접근방법인데, 그렇게 투입된 자원을 소프트웨어 자체가 제대로 활용하지 않으면 가망이 없다.

당시는 금융 위기 이후 회복세를 타기 시작한 월스트리트에 헤지펀드의 수가 빠르게 늘어나는 시점이었기 때문에 더 늦기 전에 프로그램을 대폭으로 리팩토링하기로 결정했다. 이때 염두에 둔 요구사항은 다음과 같다.

- 컴퓨터 1대 위에서 돌아가는 속도가 6시간보다 빠를 것
- 컴퓨터를 추가해 병렬처리가 가능하게 할 것
- 동작하는 방식이 이해하기 편하고 새로운 시나리오 추가가 쉽도록 코드를 만들 것

개념을 중심으로 정리하면 첫째는 일반적인 처리율throughput에 대한 것이고, 둘째는 스케일 아웃에 대한 것이며, 셋째는 코드의 모듈화modularity에 대한 것이다. 이러한 세 가지 요구사항을 충족시키기 위해서 나는 아카를 선택했다.

처리율(throughput)

아카를 이용한 리팩토링을 끝마쳤을 때, 똑같은 컴퓨터 위에서 전과 동일한 몬테카를로 시나리오를 수행하는데 걸리는 시간이 6시간에서 2시간으로 단축되었다. 66%의 시간이 절약된 것이다. 결과를 확인한 사람들은 깜짝 놀랐다. 단순히 자바 스레드에서 아카로 라이브러리를 바꾸었을 뿐인데 그렇게 엄청난 차이가 있을 수 있냐며 고개를 갸웃거렸다.

물론 이런 차이를 일반화할 수는 없다. 이런 결과 하나를 가지고 아카가 자바 스레드보다 3배 빠르다고 말하는 어리석은 사람은 없을 것이다. 아카도 내부적으로 자

바 스레드를 사용하기 때문에 그런 비교 자체가 성립하지 않는다. 하지만 일반적인 차원에서 짚고 넘어갈만한 부분도 있다. 이렇게 커다란 차이가 어디에서 비롯되었는지 이해하려면 우선 암달의 법칙Amdahl's law을 생각해볼 필요가 있다. 암달의 법칙은 이렇다.

"멀티코어를 사용하는 프로그램의 속도는 프로그램 내부에 존재하는 순차적sequential 부분이 사용하는 시간에 의해서 제한된다."

Thread나 Task를 만들어서 ExecutorService에게 제출하는 식으로 동시성 코드를 작성하면 여러 개의 스레드가 동시에 작업을 수행한다. 하지만 프로그램 안에는 Thread나 Task가 포함하지 않는 코드가 존재한다. 여러 개의 스레드가 동시에 작업을 수행하더라도 synchronized 블록이나 데이터베이스, 네트워크 API 호출 등을 만날 때 다른 스레드와 나란히 줄을 서서 순차적으로 작업을 수행해야 하는 경우도 있다. 암달의 법칙은 프로그램이 낼 수 있는 속도의 상한이 이런 순차적 코드가 사용하는 시간에 의해서 제한된다고 말하는 것이다.

이러한 순차적 코드의 또 다른 이름은 블로킹blocking 콜이다. 문제는 스레드 자체가 아니라 스레드를 사용하면서 자기도 모르게 만들어내는 블로킹 콜이다. 조금 과장해서 말하자면 자바 개발자가 스레드를 이용해서 만들어내는 '동시성' 코드는 일종의 신기루다. 사실은 코드 곳곳에 존재하는 블로킹 콜, 순차적 코드 때문에 전체적인 프로그램의 처리율은 이미 상한이 정해져 있지만 여러 개의 스레드가 '동시에' 동작한다는 사실로부터 위안을 받을 뿐이다.

아카 내부에 숨겨진 마법 같은 것은 없다. 아카는 스칼라 언어로 작성되었지만 더 아래로 내려가면 자바의 동시성 패키지를 사용하기 때문에 아카를 사용하는 것은 궁극적으로 자바의 Thread나 Task를 사용하는 것과 마찬가지다. 하지만 아카를 사용하면 프로그램 곳곳에 존재하는 순차적 부분, 블로킹 콜을 전부 없애거나 적어도 최소한으로 만드는 것이 가능해진다. 6시간이 2시간으로 줄어드는 '마법'은

여기에서 비롯된 것이다.

아카는 물리적으로 가벼운 라이브러리지만, 어떤 의미에서는 하나의 패러다임이다. 블로킹blocking 혹은 동기적synchronous 방식의 프로그래밍에 익숙한 우리의 사고방식을 난블로킹non-blocking 혹은 비동기적asynchronous 방식으로 탈바꿈시킨다는 점에서 패러다임 전환을 요구한다. 아카를 이용해서 프로그램을 설계한다는 것은 블로킹 호출이 일어나는 지점을 난블로킹 호출로 전환하는 작업을 수행하는 것을 의미한다. 그렇기 때문에 암달의 법칙에서 이야기하는 순차적 부분이 차지하는 면적이 최소한으로 줄어들게 되고, 프로그램의 전체적인 처리율은 그와 반비례해서 급등하게 된다.

이런 이야기가 아직 구체적으로 감이 잡히지 않아도 걱정할 필요는 없다. 이 책에 담긴 예제코드를 공부하고 아카를 이용한 실전코드를 작성해보면 금방 이해가 될 것이다. 다만 블로킹/동기적 호출은 낡은 방식이고, 난블로킹/비동기적 호출은 현대적 방식이라는 점은 기억할 필요가 있다. 그게 핵심이다. 그리하여 낡은 방식을 고집할 것인가 아니면 현대적 방식을 받아들일 것인가는 결국 자신의 선택임을 잘 생각해보기 바란다.

스케일 아웃(scale out)

아카가 가진 가장 탁월한 장점은 스케일 아웃을 자동적으로 보장해준다는 점이다. 스케일 아웃을 위해서 해야 하는 일이 실제로 거의 없다. 구성파일의 내용을 약간 수정하는 게 전부다. 간단한 예를 생각해보자.

수많은 사람이 방문하는 웹사이트가 있다고 가정하자. 웹서버로 1대의 컴퓨터 위에서 돌아가는 1개의 톰캣 JVM을 사용한다. 그런데 방문자의 수가 늘어서 1대의 서버로는 더 이상 트래픽을 감당할 수 없게 되었다. 이런 경우에 우리는 전통적으로 로드 밸런서load-balancer 뒤에 여러 대의 서버를 설치하는 방식을 사용한다. 로드

밸런싱 클러스터 구축, 혹은 로드 밸런서를 이용한 스케일 아웃이다.

이렇게 하면 어느 정도 수준의 스케일은 충분히 감당할 수 있다. 하지만 최선은 아니다. 무엇보다도 여러 대의 컴퓨터 위에서 돌아가는 톰캣 서버가 서로의 존재를 알지 못한다. 서로 완전히 독립되어 있다. 그러한 독립성이 장점일 수도 있지만 단점일 수도 있다. 예를 들어서 여러 대의 톰캣 서버에게 각각 특정한 작업을 수행하는 역할을 부여할 필요가 있다고 하자. 그렇게 하는 것이 가능하긴 하지만 복잡한 알고리즘을 구현해야 한다. 불편하다.

더 큰 문제는 자원이용의 효율성이다. 예를 들어서 톰캣 위에서 돌아가는 프로그램 내부에 A부터 D까지 개의 컴포넌트가 존재한다고 하자. 이들은 모두 2만큼의 자원을 사용하는데 유독 C만 4만큼의 자원을 사용한다. 여기에서 자원은 CPU가 될 수도 있고 메모리가 될 수도 있다. 그래서 프로그램 전체가 사용하는 자원은 2+2+4+2=10이다. 우리가 사용하는 컴퓨터는 자원을 15까지 지원할 수 있다고 하자. 우리는 컴포넌트 C를 스케일하고 싶다. C의 용량만 두 배로 키우면 시스템의 처리율을 두 배로 올릴 수 있음을 안다(고 가정하자).

하지만 톰캣에서 동작하는 프로그램은 대부분 하나의 큰 덩어리, 영어로 'monolithic'하다고 표현하는 방식으로 작성되어서 하나의 컴포넌트만 떼어서 독립적으로 실행하기가 어렵다. 설령 그렇게 할 수 있다고 해도 그 컴포넌트가 다른 컴포넌트와 커뮤니케이션 하는 것이 문제다. 그래서 보통은 프로그램 전체를 복제한다. 즉, 자원소모가 2+2+4+2=10에 달하는 프로그램 전체를 하나 더 생성해서 실행시키는 것이다.

그런데 우리는 앞에서 컴퓨터가 처리할 수 있는 용량이 15라고 말했다. 따라서 추가적으로 생성된 프로그램은 별도의 컴퓨터 위에서 동작해야 한다. 10만큼의 자원을 사용하는 프로그램을 동시에 2개 실행시킬 수 없기 때문이다. 이 시점에서 생각해보면, 자원 사용량이 4에 불과한 C라는 컴포넌트를 스케일하기 위해 우리

가 자원 사용량이 10에 달하는 프로그램 전체를 복제했음을 알 수 있다. 심지어 새로운 컴퓨터까지 필요하다. 단순히 4를 복제해 8로 만들려고 10을 20으로 복제하는 것은 받아들이기 힘든 비효율성이다.

아카의 경우라면 이야기가 다르다. 아카 세계에서 컴포넌트는 액터다. 어느 액터를 하나만 콕 집어내어 기존의 JVM에서 독립시켜 독자적인 JVM으로 실행하는 것은 거의 아무런 노력이 들지 않는다. 굳이 별도의 JVM으로 독립시키지 않고 동일한 JVM 내부에서 인스턴스의 수만 늘어나게 하는 방법도 가능하다. 본문에서 설명하겠지만 장소 투명성location transparancy라는 기능이 제공되기 때문에 이러한 스케일을 위해서 코드를 수정할 필요가 전혀 없다.

이 예에서 우리가 스케일 하려는 대상은 C라는 컴포넌트다. 아카를 사용하면 다른 부분은 건드리지 않고 C만 두 배로 만들어주는 것이 가능하다. C가 두 배가 되면 자원 사용량이 2+2+8+2=14인데, 컴퓨터가 15까지는 지원할 수 있으므로 별도의 컴퓨터가 필요하지 않다. 이런 유연한 스케일은 시스템 전체의 처리율을 적정한 수준으로 유지하는 작업을 수월하게 만들어준다. 반응형 선언reactive manifesto에서는 이렇게 스케일 업/다운, 아웃/인을 자유자재로 지원하는 기능을 일컬어서 탄력성elasticity이라고 말한다. 이런 특성을 가장 직관적으로, 풍부하게 지원해주는 라이브러리가 바로 아카다.

아카에서 제공하는 클러스터clustering 기능은 매우 강력하고 편리하다. 내가 재작성한 몬테카를로 프로그램은 컴퓨터를 1대만 사용해도 이미 우리가 원했던 것보다 빠른 시간을 보여주었기 때문에 클러스터를 사용할 필요가 없었다. 하지만 내가 지금 다니고 있는 회사에서는 아카 클러스터 기능을 이용해서 여러 가지 흥미로운 패턴을 만들어내고 있는 중이다. 아카 라이브러리가 아니었으면 쉽게 생각하지 못했을 일들을 구현하는 작업이 즐겁다.

모듈화(modularity)

일반적으로 아카 라이브러리에 관심을 갖는 사람들은 아카를 주로 자바의 Thread나 Task를 대신하여 사용할 수 있는 동시성 라이브러리라는 측면에서 접근한다. 이렇게 접근하면 한 가지 중요한 사실을 간과하는 경우가 많다. 아카의 액터 모델이 단순히 동시성 코드를 지원하기 위한 모델이 아니라 오히려 진정한 의미에서의 객체지향을 구현하기 위한 패러다임이라는 사실을 잊는 것이다.

액터가 제공하는 가장 강력한 기능의 하나는 사실 모듈화다. 아카를 이용해서 시스템을 설계하면 '클래스'나 '객체'를 중심으로 생각하던 사고방식이 '액터'를 중심으로 생각하는 방식으로 변하게 된다. 우리는 자바나 C#같은 언어에서 사용하는 객체가 객체지향의 원리를 충실히 구현하고 있다고 생각하지만, 사실 그렇지 않다. 객체들은 서로의 메서드를 동기적인 방식으로 호출하면서 관련을 맺기 때문에 다른 객체의 내부에서 발생하는 사건(예를 들어, 예외exception가 발생하는 것)으로부터 직접적인 영향을 받는다. 서로의 살이 직접적으로 맞닿아 있기 때문이다. 다른 말로 하면, 우리가 사용하는 객체는 서로 밀접하게 결합tightly coupled되어 있어 나쁘다.

그에 비해 액터는 서로 완벽하게 독립적이며, 오로지 메시지를 주고받는 방식으로만 커뮤니케이션하므로 코드의 응집성coherence, 느슨한 결합loosely coupled, 캡슐화encapsulation와 같은 프로그래밍 원리를 완벽하게 구현한다. 이런 원리가 성립하는 방식으로 코드를 설계하다 보면 코드 조각들이 특정한 기능을 중심으로 자연스럽게 모여든다. 그 조각이 하나의 독립적인 액터, 혹은 컴포넌트를 형성하는 것이다.

이런 컴포넌트를 뚝 떼어내서 독립적인 JVM 위에서 실행하려면 클러스터를 구축할 수도 있고, 아예 독자적인 서비스로 만들려면 마이크로서비스 아키텍처를 구현할 수도 있다. 이렇게 아카를 이용해 코드를 작성하다 보면 진정한 의미에서의 객체지향적인 사고방식을 체화하게 되기 때문에 아카를 사용하는지와 상관없이 프로그래밍 실력 일반에 좋은 영향을 받는다. 엄청난 보너스다.

차세대 동시성 모델

소프트웨어 세계는 빠르게 변한다. 하지만 시류와 상관없이 지속적으로 관철되는 법칙도 있다. 그 중에서 하나는 '추상수준 상승의 법칙'이다. 멀리 갈 것 없이 프로그래밍 언어만 생각해도 충분하다. 0과 1로 이루어진 기계어는 사람이 쉽게 이해할 수 있는 기호를 사용하는 어셈블리어로 추상수준이 높아졌고, 그것은 훗날 C와 같은 범용 언어로 추상수준이 한 단계 더 높아졌다. 바이트코드와 가상기계를 사용하는 자바에 이르러서 추상수준이 또 한 계단 상승한 것은 물론이다. 그 자바는 이제 함수 패러다임이 요구하는 추상수준으로 더 올라가야 한다는 압력을 심하게 받고 있는 중이다.

소프트웨어 세계의 모든 문제는 코드와 코드 사이에 하나의 계층을 도입하는 방법, 영어로 'one more indirection'이라고 부르는 방법으로 풀 수 있다는 이야기도 있다. 구체적인 예를 들자면 디자인 패턴에서 사용하는 어댑터adapter 패턴이 대표적이다. 이것도 넓은 의미에서는 추상수준 상승의 법칙과 일맥상통한다. 계속 추상수준이 상승하고 코드와 코드 사이에 또 하나의 계층이 도입된다는 법칙은 모든 분야에 적용되기 때문에 스레드와 관련된 부분도 예외가 아니다.

폴 부처Paul Butcher가 쓴 『7주 동안에 익히는 7개의 동시성 모델 Seven Concurrency Models in Sevel Weeks』(2014, Pragmatic)은 동시성 코딩과 관련해서 자바에서 사용하는 스레드와 잠금장치lock 정도만 알고 있는 개발자에게 현대 프로그래밍이 어떤 새로운 기법들을 추구하고 있는지 잘 설명해준다. 액터 모델은 물론 7개의 모델 중에서 하나다. 모든 개발자가 반드시 알고 넘어가야 하는 차세대 동시성 모델의 하나라는 이야기다.

액터 모델은 동시성 코드를 작성하기 위해서 지금까지 사용해온 구조물과 완전히 다른 차원의 추상수준을 제공한다. 여기에서는 잠금장치lock와 같은 구조물이 없다. 심지어 스레드라는 개념조차 초월해야 한다. (물론 실전에서는 액터와 스레드의 관

계를 제대로 이해할 필요가 있다.) 액터 모델에서는 오직 액터만 존재한다. 그들은 서로 전적으로 독립적인 구조물이며 서로 메서드를 호출할 수도 없고 new를 통해서 새로운 인스턴스를 만들 수도 없다.

폴 부처의 경우에는 동시성 코드를 돕기 위한 도구를 구체적인 코딩의 수준에서 설명하고 있는데 비해서, 요즘 자주 회자되는 반응형 선언은 상당히 추상적인 수준에서 현대 소프트웨어 시스템이 갖춰야 하는 덕목을 4개로 요약하고 있다. 반응성responsiveness, 탄력성elasticity, 유연성resilience, 메시지 중심event-driven이 그들이다.

우선 반응성은 최종사용자의 입장에서 보았을 때 가장 중요한 요소다. 사용자의 필요성에 대해서 즉각적으로 반응하지 않는 소프트웨어는 존재할 이유가 없다. 아무리 다른 요소가 훌륭해도 상관없다. 모든 소프트웨어는 사용자가 필요로 할 때 빠르게 응답하는 것을 목적으로 삼아야 한다. 그것은 곧 소프트웨어 내부에 어떤 문제가 발생했을 때, 그것을 감지하고 수정하는 작업이 전체적인 반응성에 영향을 주지 않아야 함을 뜻한다. 당연한 이야기처럼 들리지만 소프트웨어가 사용량이 폭증했을 때, 데이터베이스나 네트워크에 문제가 발생했을 때, 코드에 버그가 있었을 때, 이런 모든 상황에 구애받지 않고 일관성 있는 반응성을 유지하게 만드는 것은 쉬운 일이 아니다.

탄력성은 앞에서 이미 보았다. 원래 처음에는 확장성scalability이라는 표현이 많이 사용되었는데, 확장성은 스케일 업만 포함하고 스케일 다운을 포함하지 않는 개념이라고 해서 요즘에는 탄력성elasticity이라는 표현을 더 많이 사용한다. 소프트웨어 시스템은 필요에 따라서 더 많은 (CPU나 메모리 같은) 자원을 이용해서 처리율을 높이는 스케일 업이 가능해야 하고, 때에 따라서는 더 적은 자원을 이용하는 스케일 다운도 용이해야 함을 뜻한다. 자유롭게 늘어났다 줄어들었다 할 수 있어야 한다는 의미에서 '탄력'이라는 표현이 사용된다.

유연성은 장애허용fault tolerance과 밀접한 관련이 있는 개념이다. 앞에서 반응성을

이야기할 때 현대 소프트웨어 시스템은 어떤 장애fault나 에러가 발생했을 때 그런 사실에 구애받지 않고 일관성 있는 반응성을 유지해야 한다고 이야기했다. 유연성은 그런 일을 가능하게 만들어주는 구체적인 방법을 포괄한다. 예를 들어서 이러한 유연성은 중복replication, 격리isolation, 대리delegation와 같은 기법을 통해서 확보될 수 있다. 여기에서 중복은 낯선 개념이 아니다. 데이터베이스 시스템이 클러스터로 구성되어 있을 때 똑같은 데이터 조각을 여러 노드에 중복해서 저장하는 것은 하나의 노드가 동작을 멈추었을 때 데이터 손실이 발생하지 않도록 만들기 위한 기법이다. 격리는 소프트웨어를 구성하는 여러 컴포넌트 중에서 어느 한 컴포넌트에서 문제가 발생했을 때, 그것이 나머지 컴포넌트에 영향을 미치지 않도록 일정한 경계를 설정해주는 기법을 의미한다.

메시지 중심은 원래 사건 중심event-driven이라는 개념으로 표현되었는데, 사건event보다 메시지message가 더 적확한 표현이라는 이야기가 나오면서 최근에는 메시지 중심이라는 말을 더 자주 쓴다. 사건과 메시지는 비슷하지만 서로 다른 개념이다. 차이를 하나만 이야기하자면 사건은 목적지destination가 없는데 비해서 메시지는 반드시 목적지를 갖는다. 메시지 중심이라는 개념은 비동기적asynchronous 메시지 전달과 장소 투명성location transparency를 포괄한다.

아카를 학습하다 보면 반응성, 탄력성, 유연성, 메시지 중심이라는 4개의 덕목이 어떻게 아카에 구현되어 있는지 목격하게 된다. 즉 액터 모델은 현대 소프트웨어 시스템이 갖춰야 하는 덕목을 종합적으로 지원해주는 거의 유일한 모델이다. 액터 모델을 구현한 라이브러리가 아카만 있는 것은 아니다. 얼랭Erlang도 액터 모델을 구현해서 사용해왔고, 최근에는 얼랭을 계승한 엘릭서Elixir도 액터 모델을 사용하고 있다. 하지만 JVM 언어를 사용하는 개발자에게는 실질적으로 아카가 유일한 라이브러리다.

이야기를 많이 했으니 이제 코드를 보도록 하자.