RESTful Web API

웹 API를 위한 모범 전략 가이드

RESTful Web APIs

By Leonard Richardson, Mike Amundsen, Sam Ruby

Authorized Korean translation of the English edition of RESTful Web APIs. ISBN 9781449358068

RESTful Web API

전자책 1쇄 발행 2022년 12월 8일 (종이책 초판 3쇄 반영) **지은이** 레오나르드 리처드슨, 마이크 애먼슨, 샘 루비 **옮긴이** 박세현, 박진형
펴낸이 한기성 **펴낸곳** (주)도서출판인사이트 **편집** 송우일 **등록번호** 제2002-000049호 **등록일자** 2002년 2월 19일 **주소** 서울시 마포구 연남로5길 19-5 **전화** 02-322-5143 **팩스** 02-3143-5579 **블로그** https://blog.insightbook.co.kr **이메일** insight@insightbook.co.kr
ISBN 978-89-6626-382-0

프로그래밍 인사이트

RESTful Web API

웹 API를 위한 모범 전략 가이드

레오나르드 리처드슨·마이크 애먼슨·샘 루비 지음
박세현·박진형 옮김

인사이트

차례

옮긴이의 글

이제는 모바일 플랫폼을 기본으로 지원해야 하는 시대가 되었습니다. 이에 따라 API 서버를 작성하는 일도 많아지고 REST라는 개념 역시 훨씬 더 자연스럽게 다가옵니다. 그래서 REST를 공부해야 하나 하는 궁금증이 생길지도 모르겠습니다. 『RESTful Web API』는 이제 막 API 프로그래밍을 접해 보는 분들뿐 아니라 이미 REST에 대해 어느 정도 알고 있고 현업에서도 많이 사용하는 분들에게도 도움이 되는 책이 되리라 생각합니다.

한국어로 번역하기 전 책을 읽어보며 리뷰하는 과정에서 개발자들에게 도움이 많이 될까 하는 의문이 있었지만 번역을 마친 시점에서는 도움이 되리라는 확신이 생겼습니다. REST라는 개념이 나타난 배경이나 이와 관련된 여러 가지 다양한 표준에 대해 더 정확하게 알 수 있고, API 설계 시에 어떤 점을 고려해야 하는지, 개발자들이 많이 힘들어 하는 네이밍과 관련하여 API를 작성할 때 어떻게 해결할 수 있는지 등 현업에서도 바로 적용해 볼 수 있는 내용이 많습니다. 특히 기존에 잘 정의된 개념을 재활용하여 API 작성을 장려하는 부분들은 많은 교훈을 줍니다.

기존에 관련 서적이 번역된 사례가 없어서 번역을 하는 과정에서 용어 선정에 상당히 많은 고민을 하였습니다. 여러 책을 번역해 보았지만 이번 책만큼 용어 선정이 어려웠던 책은 없었던 것 같습니다. 최대한 번역하여 한국어로 소개할 수 있도록 하려고 했지만 원어를 사용하지 않으면 의미 전달이 어려워지는 경우는 어쩔 수 없이 원어를 사용하기도 하였습니다. 책의 마지막 부록 「용어 해설」 부분에서 최대한 잘 설명하려고 노력하였습니다. 이 부분이 책을 이해하는 데 많은 도움이 되었으면 좋겠습니다.

꾹 참으며 번역 원고를 기다려 주신 한기성 사장님에게 먼저 감사의 말을 전합니

다. 또한 같이 번역하며 고생한 박세현 님에게 감사의 말을 전합니다. 두꺼운 이 책을 빠르게 리뷰해 준 김영후 님에게도 감사의 말을 전합니다. 책 번역하는 동안 아빠에게 놀아달라고 보채지도 않고 잘 참고 기다려 준 첫째 딸 리나, 그리고 리나와 로이를 보느라 고생한 아내 효정이에게도 감사의 말을 전합니다.

마지막으로 이 책을 구입해 주신 독자 여러분들에게 정말 감사하단 말을 전하고 싶습니다. 정말 조금이라도 도움이 될 수 있었으면 좋겠습니다. 감사합니다!

-박진형

무한한 인내로 계속 늘어지는 일정을 기다려 주시고 좋은 책을 만들어 주시는 한기성 사장님께 감사를 표한다. 책 번역과 여러 가지 고생을 공유하는 진형 군에게도 고마운 마음과 축하인사를 건네고 싶다. 늘 시간에 쫓기는 날 이해해 주고 참아준 사랑하는 아내 소영, 매일 웃음을 주는 아들 도준에게 사랑과 감사를 전한다.

-박세현

추천사

사용자 인터페이스 디자인에서 단계적 공개(Progressive Disclosure)란 바로 그 시점에 필요한 정보만 사용자에게 표시해야 한다는 개념이다. 여러모로 이 책이 그 원칙의 예에 속한다. 사실, 7년 전만 하더라도 이 책은 전혀 팔리지 않았을 것이다.

이 책보다 앞서 『RESTful Web Services』(『RESTful 웹 서비스』, 강정민 옮김, 정보문화사 펴냄)가 쓰인 이후, 지금의 프로그래밍 세계는 그때와 굉장히 많이 달라졌다. 그 시절엔 'REST'라는 용어는 거의 사용되지 않았다. 사용되더라도 잘못 쓰인 경우가 많았고, 사람들도 그 의미를 오해하곤 했다.

그런데 이미 1990년대 후반부에는 HTTP와 HTML 같이 REST의 기반을 이루는 표준이 대략 현재의 형태로 개발되었고 IETF와 W3C 표준이 되었다. 또, 이 책이 기반을 두고 있는 로이 필딩(Roy Fielding)의 2000년도 논문에서 REST란 용어가 소개되었다.

레오나르드 리처드슨(Leonard Richardson)과 나는 이런 부당한 현실을 고치기로 결심했다. 이를 위해 먼저 HTTP의 기초 개념에 집중하고, 애플리케이션에 이런 개념을 어떻게 적용할 수 있는지 실용적인 가이드를 제공했다.

나는 우리가 그때 눈사태처럼 커져버린 REST 지원의 첫 삽을 떴다고 생각하고 싶다. REST는 곧 자리를 잡았고 전문적인 유행어가 되었다. 사실 이제 너무 흔해져서 웹 인터페이스를 표시한다 하면 거의 기본적으로 REST라 부르게 될 정도다. 몇 년 새에 정말 많이 바뀌었다.

REST 용어가 과하게 쓰이기도 하고, 잘못 사용되는 경우가 많다는 것을 인정한다. 그렇지만 모든 걸 고려해도, 리소스와 URI의 개념이 애플리케이션 인터페이스 디자인에 잘 스며들어 기쁘다. 결국 웹은 탄력적인 곳인지라 비록 불완전하더라도

새로운 인터페이스가 등장하면 기존 기술을 대체해 훨씬 더 나은 환경을 제공한다.

물론 지금보다 더 잘 해낼 수 있다.

이제 소재가 갖춰졌으니 한 발짝 뒤로 물러나 지형을 관찰하고, 이들 개념 위에 쌓아 올려보자. 자연히 다음 단계는 일반적인 미디어 종류, 특히 하이퍼미디어 유형을 알아보는 것이다. 첫 책이 제대로 된 HTTP 활용을 집중적으로 다룬 반면, 이번에는 HTML과 같은 하이퍼미디어 뒤의 개념을 깊이 파고 들어 간다. 즉, 단일 애플리케이션이나 벤더에 긴밀히 묶이지 않는 미디어 유형을 다룬다.

HTML은 여전히 이런 하이퍼미디어 유형의 주요 예이며, 웹 아키텍처에서 특수한 위치를 차지하고 있다. 사실 내 개인적인 여정을 하나 밝히자면, 난 이제 HTML5가 된 WC3의 HTML 표준 개발에 깊이 관여하고 있다. HTML이 이 새 책에서 독보적인 위치를 가지고 있지만 하이퍼미디어도 아주 많은 내용을 포함하는 주제다. 그래서 레오나르드는 나 대신 이 내용을 써줄 능력 있는 공동 저자 마이크 애먼슨(Mike Amundsen)을 찾아냈다.

이 책이 쓰이는 과정을 보는 것은 매우 기쁜 일이었고, 책을 읽으며 다른 데서는 보지 못한 미디어 유형도 배웠다. 더 중요한 것은, 이 책이 이런 유형들의 공통점, 차이점, 각각의 특징을 잘 알려준다는 것이다.

이 책이 뜬 첫 삽이 전작과 같은 효과를 내길 기대한다. 이 책이 7년 뒤에 REST(Representational State Transfer)에서 여전히 저평가된 어떤 부분을 돋보이게 해줄지 누가 알겠는가!

- 샘 루비(Sam Ruby)

머리말

"대다수 소프트웨어 시스템은 전체 시스템이 한 개체의 컨트롤 아래 있거나 적어도 시스템에 참여하는 모든 개체가 동일한 목표를 가지고 있고 상반되는 의도를 가지고 있지 않다고 은연중에 가정한다. 만일 시스템이 인터넷에서 공개되어 동작한다면 이런 가정이 문제없을 거라고 안심할 수 없다."
- 로이 필딩(Roy Fielding)
「아키텍처 스타일과 네트워크 기반 소프트웨어 아키텍처 디자인(Architectural Styles and the Design of Network-based Software Architectures)」

"디스코르디안은 항상 공식적인 디스코르디안 문서 넘버링 시스템을 사용해야 한다."
- 그레고리 힐(필명: Malaclypse the Younger)·케리 웬델 쏜리(필명: Lord Omar Khayyam Ravenhurst)
디스코르디아 원리(Principia Discordia: 불화의 원리, http://principiadiscordia.com)

나는 이 책에서 역사상 가장 성공적인 분산 처리 시스템인 월드 와이드 웹에 내재하는 아이디어를 사용해 분산 컴퓨팅의 개선된 방법을 보여줄 것이다. 부디 여러분이 이 책을 읽고 있는 이유가 회사에서 웹 API를 만들 필요가 있다고 본인이나 매니저가 결정했기 때문이기를 바란다. 공개 API인지, 내부 API인지, 신뢰하는 파트너에게만 공개된 API인지는 중요하지 않다. 모두 REST의 사고방식에서 혜택을 얻을 수 있다.

이 책은 API 클라이언트를 어떻게 작성하는지 배우기에는 적절하지 않다. 현존하는 대부분의 API 디자인은 수년 전에 세워진 가설에 기반을 두고 있으며, 나는 개인적으로 이 가설들을 제거해 버리고 싶다.

오늘날 대다수 API는 큰 문제를 가지고 있다. 한 번 배포되면 바꿀 수가 없다는 것이다. 이름난 API는 바꾸는 게 너무 어렵기 때문에, 주변 산업 자체가 바뀌더라도 배포된 그 상태로 수년간 유지된다.

하지만 RESTful 아키텍처는 변화를 다룰 수 있도록 설계되었다. 월드 와이드 웹 세계에서는 수백만 개의 사이트가 수천 개의 다른 서버 구현 위에서 돌아가고 있으며 정기적으로 디자인이 갱신된다. 또 수십억 명의 사용자가 수십 개의 하드웨어 플랫폼 위에 올라간 수백 개의 다른 클라이언트 구현 위에서 웹 사이트에 접속하고 있다. 여러분이 배포한 웹 사이트가 이런 난장판은 아니겠지만, 웹의 스케일이 더 커질수록 이 모습을 더 명확히 보게 될 것이다.

항상 간단한 시스템이 변경이 쉽다. 작은 스케일에서, 버튼 하나 누르면 끝나는 솔루션보단 RESTful 시스템이 초기 디자인 비용이 더 많이 든다. 그렇지만 API가 더 커지고 바뀌어감에 따라 REST 같은 방식으로 변화에 적응해야 할 필요성을 느낄 것이다.

- 상업적으로 성공한 API는 수년간 유효할 것이다. 어떤 API는 수백 또는 수천 명의 사용자가 있다. 문제의 도메인이 가끔 바뀌는 수준이라 할지라도 고객들에게 미치는 영향의 총합은 엄청나게 크다.
- 어떤 API는 새 데이터 요소와 비즈니스 규칙이 계속 추가됨에 따라 항상 변화한다.
- 어떤 API는 각 고객이 필요에 따라 작업 흐름을 변경할 수 있다. API 자체는 결코 변화되지 않지만 각 고객마다 API를 사용하는 경험은 달라질 수 있다.
- 보통 API 클라이언트를 작성하는 사람들은 서버를 작성하는 사람과 다른 팀에서 일한다. 모든 오픈된 API는 이 카테고리에 속한다. 어떤 종류의 클라이언트가 있는지 잘 모른다면, 변화를 가하기 전에 매우 주의를 기해야만 한다. 아니면 모든 클라이언트를 망가뜨리지 않고 변화할 수 있는 디자인이 필요하다.

API에 기존 디자인을 베끼면 아마 과거의 실수를 다시 반복하게 될 것이다. 불행

히도, 대부분의 개선책은 표면 아래의 실험과 매우 느린 속도의 표준화 절차를 통해 이루어진다. 이 책에서 수십 개의 특정 기술들을 다루는데, 그중 일부는 여전히 개발 중인 기술이다. 그렇지만 이 책의 주목적은 독자에게 REST의 밑바탕을 이루는 원칙을 가르치는 것이다. 이걸 배우고 나면, 어떤 실험 단계의 기술이든 승인된 표준이든 활용할 수 있을 것이다.

이 책에서 해결하고자 하는 두 가지 문제점이 있다. 비슷한 노력을 되풀이하는 것과 하이퍼미디어를 사용하지 않는 문제다. 이것들을 한번 살펴보자.

노력의 반복

오늘날 발표되는 API들은 보통 회사의 이름을 딴다. '트위터 API', '페이스북 API', '구글+ API'를 종종 언급하곤 한다. 이 세 API는 비슷한 일을 수행한다. 모두 사용자 계정 정보를 제공하고 (다른 많은 것들도 하지만) 계정에 텍스트를 올리도록 하는 기능을 제공한다. 하지만 각 API는 완전히 다른 디자인을 가지고 있다. 한 API를 배운다 해서 다른 API를 익히는 데 별다른 도움이 되지 않는다.

물론 트위터, 페이스북, 구글은 서로 경쟁하는 대기업이다. 당연히 경쟁자의 API를 쉽게 배우기를 바라지 않는다. 하지만 작은 회사와 비영리 기업들도 마찬가지다. 다들 다른 누군가가 결코 비슷한 아이디어를 가진 적이 없었던 것처럼 API를 디자인한다. 이게 사실 이 API들을 사용할 사람을 얻고자 하는 목표에 방해가 된다.

예를 하나 들어보겠다. ProgrammableWeb(http://www.programmableweb.com/)이라는 웹 사이트는 8000개 이상의 API 디렉터리를 가지고 있다. 이 책을 쓰는 현재 57개의 마이크로 블로깅 API가 있는데, 이 API들의 주목적은 사용자 계정에 텍스트를 포스팅하는 것이다.[1] 이쪽 분야에 57개 회사가 API를 공개하고 있다는 게 훌륭한 일이긴 하지만, 정말 57개의 다른 디자인이 필요한 것일까? 뭐 보험 정책이나 법률 준수 사항 같이 복잡한 무언가를 얘기하고 있는 것이 아니다. 그저 사용자 계정에 텍스트 좀 포스팅하는 것에 대한 이야기다. 58번째 마이크로블로깅 API를 만드는 누군가가 되고 싶은가?

분명한 해결책은 마이크로블로깅 API 표준을 만드는 것이다. 하지만 이미

1 ProgrammableWeb API 중 microblogging 태그(http://www.programmableweb.com/apitag/microblogging)가 붙은 API 전체 목록에서 이 API들의 정보를 찾을 수 있다

APP(Atom Publishing Protocol)라는 잘 동작하는 표준도 있다. 2005년에 공개되었는데, 거의 아무도 사용하지 않고 있다. 사업 측면에서 볼 땐 말이 안 되지만 API를 만드는 모두가 처음부터 자신만의 설계를 하길 원하는 이유가 있다.

나 혼자 힘으로 이런 노력의 낭비를 멈출 수 있다고 생각하지 않는다. 하지만 문제를 이해할 수 있는 단위로 쪼개고, 새 API에서 이미 만들어진 작업을 재사용할 수 있는 방법을 제공하고자 한다.

하이퍼미디어는 어렵다

지난 2007년 레오나르드 리처드슨과 샘 루비는 이 책의 전신인 『RESTful 웹 서비스』를 썼다. 그 책에서도 두 가지 큰 문제를 해결하고자 했다. 문제 하나는 해결되었지만, 다른 하나는 전혀 해결되지 않았다.[2]

첫 번째 문제를 보자. 2007년 REST 가르침을 따른 API 디자인은 SOAP 기반의 경쟁 가르침과의 교착 상태를 버텨야만 했고, REST 학계 자체의 정당성을 계속 의심받아야만 했다. 『RESTful 웹 서비스』는 이 상황의 지원 폭격이 되어 SOAP 학계의 공격을 RESTful 디자인 원칙으로 방어할 수 있게 해주었다.

이제 과거의 이 교착 상태는 끝나고 REST가 이겼다. SOAP API가 여전히 사용되긴 하지만, 처음부터 SOAP 계열을 지지하던 큰 회사 내에서나 그러하다. 이제 거의 모든 공개된 API는 다 RESTful 원칙을 칭송하고 있다.[3]

이제 두 번째 문제를 살펴보자. REST는 기술적인 용어로만 쓰이는 게 아니다. 마케팅 버즈워드로도 많이 쓰인다. 아주 오랫동안 REST는 SOAP 계열에 대한 반대 슬로건으로 쓰여 왔다. SOAP를 쓰지 않는 API라면 REST의 원칙에 맞지 않거나 반할지라도 REST로 홍보되었다. 이는 부정확하고 헷갈릴 뿐 아니라 기술 용어로서의 REST에 악영향을 미쳤다.

이 상황은 2007년 이후 많이 개선되었다. 이제는 이 책의 앞 장에서 설명하는 개념을 이해하는 개발자들이 만든 새 API를 보곤 한다. REST의 깃발을 흔드는 대다수의 개발자는 이제 리소스와 표현, URL로 리소스 부르기, HTTP 메서드 제대로 사용

2 『RESTful 웹 서비스』는 오라일리의 오픈 북 프로젝트의 일부로 무료 제공된다. 책 페이지에서 PDF 본을 다운로드할 수 있다.

3 혹시 궁금했다면 이것이 바로 책 제목을 바꾼 이유다. '웹 서비스'라는 용어는 SOAP와 너무 긴밀히 연결되어 있어서 SOAP 시대가 저물면서 이 용어도 같이 자취를 감췄다. 요즘은 다들 API에 대한 얘기를 주로 한다.

하기 등을 잘 이해하고 있다. 이 책의 세 장은 새 개발자들이 빠르게 적응할 수 있도록 도와준다.

하지만 대다수의 개발자가 여전히 잘 이해하지 못하는 REST의 측면이 있다. 바로 하이퍼미디어다. 우리는 하이퍼미디어를 웹이라는 콘텍스트에서 이해한다. 사실 하이퍼미디어 웹은 링크를 멋지게 표현한 것에 지나지 않는다. 웹 페이지는 서로 링크되어 있고, 그 결과 하이퍼미디어로 돌아가는 월드 와이드 웹이 탄생한다. 그러나 웹 API에서 하이퍼미디어를 다룰 때는 뭔가 정신적인 벽이 존재하는 것 같이 느껴진다. 하이퍼미디어야말로 웹 API가 변화를 잘 수용할 수 있도록 해주는 기능이기에 매우 심각한 문제라 할 수 있다.

4장부터는 『RESTful 웹 서비스』의 목표를 이어받아 하이퍼미디어가 어떻게 동작하는지 가르칠 것이다. 이 용어를 처음 접한다면, 다른 중요한 REST 개념들과 함께 배우게 될 것이다. 하이퍼미디어를 들은 적이 있지만 그 개념이 위협적으로 느껴진다면, 용기를 북돋아줄 것이다. 하이퍼미디어를 온전히 이해하지 못했다면, 독자가 이해할 때까지, 생각할 수 있는 모든 방면에서 알려줄 것이다.

『RESTful 웹 서비스』에서도 하이퍼미디어를 다뤘지만 책의 핵심 내용은 아니었다. 하이퍼미디어 파트를 건너뛰고도 동작하는 API를 디자인할 수 있었기 때문이다. 반면 『RESTful Web API』는 하이퍼미디어에 대한 책이라 할 수 있다.

이렇게 한 이유는 하이퍼미디어야말로 REST에서 가장 중요한 요소이면서도 가장 잘 이해되지 않았기 때문이다. 우리 모두가 하이퍼미디어를 잘 이해하기 전까지는 REST는 분산 컴퓨팅의 복잡성을 다루기 위한 진지한 시도가 아니라 마케팅 버즈워드로 인식될 것이다.

이 책에 포함된 것들

첫 네 장은 웹 API에 적용되는 REST 뒤의 개념을 다룬다.

1장. 웹 서핑하기

이미 친숙한 RESTful 시스템인 웹 사이트를 통해 기본 용어를 설명한다.

2장. 간단한 API

1장에서 다룬 웹 사이트와 같은 기능을 수행하는 프로그래밍 가능한 API를 다룬다.

3장. 리소스와 표현

리소스는 HTTP 하부의 핵심 개념이고 표현은 REST 하부의 핵심 개념이다. 이 둘이 어떻게 연결되는지 설명한다.

4장. 하이퍼미디어

하이퍼미디어는 표현을 한데 모아 긴밀한 API로 만들어주는 필수 요소다. 하이퍼미디어가 어떤 일을 할 수 있는지를 친숙한 데이터 형식인 HTML을 위주로 설명한다.

다음 네 장은 하이퍼미디어 API를 디자인하는 다른 전략들을 설명한다.

5장. 도메인 특화 설계

가장 명백한 전략은 처해 있는 그 문제를 해결하기 위해 완전히 새로운 표준을 디자인하는 것이다. Maze+XML 표준을 예로 들었다.

6장. 컬렉션 패턴

여러 패턴 중 컬렉션 패턴은 API 설계에서 반복해서 나온다. 이 패턴을 설명하기 위해 Collection+JSON과 AtomPub이라는 두 개의 다른 표준을 설명한다.

7장. 순수 하이퍼미디어 설계

컬렉션 패턴이 요구 사항에 맞지 않는다면 원하는 표현을 다목적의 하이퍼미디어 포맷을 사용해 전송할 수 있다. 세 개의 다목적 하이퍼미디어 포맷(HTML, HAL, 사이렌)을 통해 어떻게 동작하는지 알아본다. 또, HTML 마이크로포맷과 마이크로데이터를 소개한다.

8장. 프로파일

프로파일은 (다양한 API가 사용할 수 있는) 데이터 포맷과 특정 API 구현 사이를 메워준다. 추천하는 프로파일 포맷은 ALPS이지만 XMDP와 JSON-LD도 설명한다.

이 장에서 내 조언은 이 책이 쓰이는 시점의 최신 기술을 앞지른다. 충분한 포맷이 없어 이 책을 위해 ALPS 포맷을 작성했다. 하이퍼미디어 기반 디자인이 친숙하다면 8장까지는 건너뛰어도 좋지만 8장은 넘기지 말았으면 좋겠다.

9장부터 13장은 올바른 하이퍼미디어 포맷 선택하기, HTTP 프로토콜 최대한 활용하기 같은 실용적인 주제를 다룬다.

9장. 설계 절차

이 책에서 앞 장까지 다룬 모든 내용을 모아 RESTful API를 설계하는 단계적 가이드를 제시한다.

10장. 하이퍼미디어 동물원

하이퍼미디어가 어떤 것들을 할 수 있는지 보여주기 위해 20개의 표준화된 하이퍼미디어 데이터 포맷을 설명한다. 이들 대다수는 이 책의 다른 부분에서는 전혀 다루지 않는다.

11장. API를 위한 HTTP

API를 구현할 때 HTTP를 모범적으로 사용하는 방법을 알려준다. 또 HTTP 2.0 프로토콜을 포함해 HTTP 확장에 대해서도 설명한다.

12장. 리소스 설명과 연결된 데이터

연결된 데이터(Linked Data)는 REST에 대한 시맨틱 웹 커뮤니티의 접근법이다. JSON-LD는 연결된 데이터 표준에서 가장 중요한 내용이다. 8장에서 간단히 다루었는데, 여기서 다시 설명한다. RDF 데이터 모델과 10장에서 다루지 못한 RDF 기반의 하이퍼미디어도 설명한다.

13장. CoAP: 임베디드 시스템을 위한 REST

HTTP를 전혀 사용하지 않는 RESTful 프로토콜인 CoAP을 다루며 책의 주요 부분을 마감한다.

부록 A. 상태 목록

11장의 확장 측면으로, 이 부록은 HTTP 규약에 정의된 41개의 표준 상태 코드와 확장으로 정의된 유용한 코드들도 살펴본다.

부록 B. 헤더 목록

부록 A와 비슷하게 이 부록도 11장의 확장이다. HTTP 표준에 정의된 46개의 요청/응답 헤더 코드와 몇 개의 확장 코드를 설명한다.

부록 C. API 디자이너를 위한 필딩 논문 가이드

이 부록은 API의 설계에서 REST의 의미와 REST의 기반을 이루는 문서에 대해 설명한다.

용어 해설

이 용어집은 RESTful 웹 API를 만들 때 자주 만날 용어의 정의를 포함한다. 기본 개념에 익숙해지거나 특정 개념의 정의를 빠르게 보고 싶을 때 보면 좋다.

이 책에 없는 것들

『RESTful 웹 서비스』는 REST에 대해 길게 설명한 첫 책으로 굉장히 많은 내용을 다뤄야 했다. 다행히 이제 REST의 다양한 측면을 다루는 책이 꽤 나와서 『RESTful Web API』는 핵심에 집중할 수 있게 되었다.

책의 내용이 집중될 수 있도록 넣을 만한 주제 중 몇 가지는 의도적으로 뺐다. 혹시나 책을 사고 실망하는 경우가 없도록 이 책에 없는 것을 미리 알려주고 싶다.

- 클라이언트 프로그래밍은 다루지 않는다. 하이퍼미디어 기반 API를 사용할 클라이언트를 작성하는 것은 또 다른 차원의 도전 과제를 제공한다. 일단, API 클라이언트 대신 HTTP 요청을 보낼 라이브러리를 사용한다. 2007년에도 그랬지만 여전히 그렇다. 문제는 서버 측에 있다.

  기존 API를 위해 클라이언트를 작성하면, API 설계자의 인정에 매달리게 된다. 현재는 API에 일관성이 없기 때문에 일반적인 조언을 주는 것이 불가능하다. 그래서 이 책을 통해 서버 측을 일관성 있게 만들려는 열정을 좀 살려보고자 하는 것이다. API가 서로 더 비슷해지면 더 복잡한 클라이언트 도구를 작성할 수 있게 될 것이다.

  5장에서 일부 클라이언트 샘플 구현으로 클라이언트의 다른 종류를 구분하긴

하지만, API 클라이언트에 대한 책을 원한다면 이 책이 아니라 다른 책을 알아봐야 한다. 지금 그런 종류의 책은 없는 것으로 알고 있다.

- 가장 널리 사용되는 API 클라이언트는 자바스크립트의 XMLHttpRequest 라이브러리다. 모든 웹 브라우저에 사본이 들어 있으며 대다수의 웹 사이트는 XMLHttpRequest로 소비하도록 설계한 API들 위에 만들어져 있다. 이는 이 책에서 다루기엔 너무 큰 영역이다. 각 자바스크립트 라이브러리에 대해 쓰인 책이 많다.
- HTTP의 동작 구조에 대해 꽤 많은 시간을 할애하는데(11장과 부록 A, B) HTTP 주제를 깊이 있게 다루진 않는다. 특히 캐시나 프락시 같은 HTTP 매개체는 거의 설명하지 않는다.
- 『RESTful Web Services』는 비즈니스 요구 사항을 상호 연결된 리소스 조합으로 쪼개는 과정을 매우 깊이 다룬다. 2007년 이후의 경험을 통해 API 디자인을 리소스 디자인으로 보는 것이, 하이퍼미디어에 대한 생각을 피하는 굉장히 효과적인 방법이라는 확신이 들었다. 이 책은 다른 접근법을 취하는데, 리소스보다는 표현과 상태 전이에 집중한다.

  그렇지만 이 리소스 디자인 접근법은 여전히 유요하다. 이쪽 방향으로 더 알고 싶다면 수부 알라마라주(Subbu Allamaraju)가 쓴 『RESTful Web Services Cookbook』(오라일리)을 추천한다.

주의 사항

이 책의 지은이는 두 명(레오나르드와 마이크)이지만, 책을 쓰는 동안 한 명의 저자 '나'로 합쳐서 표현한다.

이 책의 내용 중 어떤 것도 특정 언어에 매이지 않는다. 모든 코드는 네트워크 프로토콜(주로 HTTP) 위에 보내지는 메시지(주로 JSON이나 XML 문서) 형태를 띤다. 나는 독자가 안티패턴, 너비 우선 탐색 같은 일반 프로그래밍 개념에 친숙하며 월드 와이드 웹이 어떻게 돌아가는지 기본적인 이해를 하고 있다고 가정한다.

책에 제시하지는 않지만 1장, 2장, 5장에서 설명하는 서버와 클라이언트의 실제 코드가 있다. 『RESTful Web API』 깃허브 리포지터리(https://github.com/RESTful-Web-APIs)나 공식 웹 사이트(http://www.restfulwebapis.org)에서 소스 코드를 받

아 직접 돌려볼 수 있다. 이 클라이언트와 서버는 노드(Node.js) 라이브러리를 사용하는 자바스크립트로 작성했다.

노드를 사용한 이유는 클라이언트와 서버에 동일한 프로그래밍 언어를 사용할 수 있기 때문이다. 클라이언트-서버 트랜잭션을 이해하기 위해 프로그래밍 언어를 바꿔야 하는 수고를 들이지 않아도 된다. 노드는 오픈 소스이며 윈도, 맥, 리눅스 시스템에서 사용할 수 있다. 운영 체제에 설치하기 쉬워 예제 구동에 별 어려움이 없다.

최신 구현 버전으로 업데이트하기 쉽도록 코드를 깃허브에 올렸다. 독자들이 다른 프로그래밍 언어로 클라이언트-서버 예제를 포팅할 수도 있을 것이다.

표준 이해하기

월드 와이드 웹은 과학적으로 접근하고 연구할 수 있는 대상이 아니다. 어떤 일을 어떤 방식으로 하자고 한 의견 일치의 모음, 즉 사회적 구조라 할 수 있다. 다행히도 (에티켓 같은) 여타 사회적 구조와 달리, 웹을 구성하는 동의 사항들은 거의 다 의견이 일치한다. 웹을 이루는 핵심 동의 사항은 RFC 2616(HTTP 표준), HTML 4에 해당하는 W3C 규약, ECMA-262(ECMAScript라고도 알려진 자바스크립트 표준)가 있다. 각 표준은 서로 다른 일을 수행하며 이 책의 전반에 걸쳐 설명한다. API에서 사용하도록 특별히 설계된 다른 표준들도 설명할 것이다.

이런 표준이 있어서 가장 좋은 점은 든든한 기준을 제공한다는 것이다. 이것들을 사용해 다른 사람들이 사용해 보지 않은 새로운 웹 사이트나 API를 만들 수도 있다. 전체 시스템을 모든 사용자에게 설명하지 않고 새로운 부분만 알려주면 된다.

안 좋은 점도 있는데, 이런 동의 사항들이 읽기 어려운 경우가 많다는 것이다. 길고 긴 아스키(ASCII) 텍스트가 두통을 유발하는 정확한 영어로 작성됐다. 예를 들어, 일상용어인 "should(~해야 한다)"가 기술적인 의미를 가진 대문자 "SHOULD"[4]로 사용된다. 사람들이 많은 수의 기술 서적을 구매하는 이유는 바로 이런 표준 문서를 읽길 원치 않기 때문이다.

나도 뭔가 보장할 수는 없다. 만일 이 표준 중 하나가 작업에 필요하다면, 직접 뛰어들어서 규약의 내용을 파악하고 정말 이해할 각오를 하거나 더 상세히 다루는 책을 사야 할 것이다. 사이렌(Siren), CoAP, 히드라(Hydra) 같은 표준은 간단히 소

4 SHOULD의 뜻은 RFC 2119에 정의되었다.

개할 공간밖에 없다. 너무 상세히 다루면 이런 표준을 전혀 필요로 하지 않는 독자들이 지루해 할 것이 분명하다.

표준의 숲을 탐험할 때는, 모든 표준이 동일한 힘을 지니지 않는다는 사실을 염두에 두는 것이 유용하다. 일부는 굉장히 잘 자리 잡아서 모두가 사용하며, 표준을 어길 경우 굉장히 많은 문제에 봉착하게 된다. 어떤 표준은 그저 한 사람의 의견일 뿐이고 내 의견과 수준 차이가 안 나는 경우도 있다.

표준을 살펴볼 때 명목 표준, 개인 표준, 기업 표준, 개방형 표준의 네 카테고리로 나누는 것이 유용하다. 이 용어들을 이 책에서 계속 사용하므로 각각을 설명하겠다.

명목 표준

명목 표준은 사실 표준이 아니고 행동 양식이라 할 수 있다. 어느 누구도 동의하지 않는 표준으로 누군가가 무언가를 하는 방식을 설명한 것에 지나지 않는다. 이런 행동 양식을 문서화할 수 있지만, '다른 사람들도 동일한 방식으로 해야 한다'는 표준의 핵심 가정이 빠져 있다.

오늘날 거의 모든 API가 명목 표준으로 특정 회사에 결부된 일회용 설계다. 그래서 '트위터 API', '페이스북 API', '구글+ API'에 대한 얘기를 하는 것이다. 이 설계에 맞춘 클라이언트를 작성하거나 작업을 수행하려면 이것들의 설계를 이해해야만 할 수도 있다. 하지만 해당하는 회사에서 일하는 게 아니라면 이것들의 설계를 내 API에 사용해야 할 필요는 없다. 만일 명목 표준을 재사용하면 API가 표준을 따른다고 하지 않고, 그냥 복제본이라 부른다.

이 책에서 내가 해결하고자 하는 주된 문제는 수백 년에 해당하는 설계에 사용되는 수백 년에 해당하는 인력이 재사용이 불가능한 명목 표준에 묶여 있다는 것이다. 더 이상 계속되어서는 안 된다. 오늘날 새 API를 디자인하는 것은 굉장히 여러 단계를 걸쳐 바퀴를 재발명하는 것을 의미한다. API가 완료되면 클라이언트 개발자들도 마찬가지로 클라이언트의 바퀴를 재발명해야 한다.

최적의 상황이더라도 각각의 비즈니스 요구 사항이 약간씩 다를 것이므로 여러분이 만든 API는 명목 표준이 되고 말 것이다. 이상적으로 말하자면 명목 표준은 몇 개의 다른 표준 위에 약간 추가된 정도일 것이다.

명목 표준을 설명할 때는, 사람이 읽을 수 있는 문서를 링크해 놓았다.

개인 표준

개인 표준은 표준이다. 문서를 읽을 수 있고, 표준을 직접 구현할 수 있다. 하지만 이 표준은 단 한 사람의 의견이다. 5장에서 설명하는 Maze+XML 표준이 좋은 예다. Maze+XML이 미로(maze) 게임 API의 표준 구현법인 것은 아니지만, 만일 처한 상황에 적용할 수 있다면, 사용할 수 있는 표준이다. 누군가가 설계 작업을 대신 해준 것이다.

개인 표준은 다른 종류의 표준보다 좀 덜 공식적인 언어를 사용한다. 많은 수의 개방형 표준도 개인 표준으로 시작해서 많은 실험을 거쳐 공식화된 것이다. 7장에서 설명하는 사이렌이 좋은 예다.

개인 표준을 설명할 때는 표준 명세서를 링크한다.

기업 표준

기업 표준은 공통으로 경험하는 문제를 해결하려는 기업 연합체나, 반복해서 발생하는 고객의 문제를 대신 해결해 주고자 하는 단일 기업에 의해 만들어진다. 기업 표준은 개인 표준보다 더 잘 정의되고 공식적인 언어를 사용하는 경우가 많지만 개인 표준보다 더 큰 강제력을 갖진 않는다. 그저 한 회사 또는 회사 집단의 의견일 뿐이다.

기업 표준은 10장에서 설명하는 액티비티 스트림즈(Activity Streams)와 schema.org의 마이크로데이터 스키마를 포함한다. 많은 산업 표준은 기업 표준으로 시작한다. 역시 10장에서 설명하는 OData는 마이크로소프트의 프로젝트로 시작했지만 2012년 OASIS(Organization for the Advancement of Structured Information Standards)에 보내져 OASIS 표준이 됐다.

기업 표준을 설명할 때는 표준 명세서를 링크한다.

개방형 표준

개방형 표준은 위원회에 의한 설계 과정을 거치거나, 그렇지 못했다면 적어도 많은 사람이 명세서를 읽고 불평하고 개선 방안을 제안할 수 있도록 공개 논평(Open Comment) 기간을 거친 표준을 말한다. 이 과정의 끝부분에 표준 기구로 인정되는 기관에 의해 이 명세서가 승인을 받는다.

이 과정으로 개방형 표준이 어느 정도 도의적인 힘을 얻는다. 만일, 원하는 수준을 충족시키는 개방형 표준이 있다면 명목상 표준을 만들기보다는 개방형 표준을 사용하는 것이 좋다. 직접 명목상 표준을 만들 때는 너무 늦게 발견할 법한 많은 이슈가 이 설계 과정과 논평 기간 동안 제기되고 수정되었을 것이다.

일반적으로 개방형 표준은 표준화 과정에 관여한 회사로부터 특허 공격을 받을 일 없이 구현할 수 있다는 일종의 약속이 암묵적으로 존재한다. 반면, 누군가의 명목상 표준을 구현하다가 특허 침해로 소송을 당할 수도 있다.

이 책에서 다루는 몇 가지 개방형 표준은 ANSI, ECMA, ISO, OASIS, 특히 W3C 같은 대형 표준 기구로부터 나왔다. 이런 표준 기구에서 위원으로 활동한 적이 없어서 어떤 기분일지 잘 모르겠다. 하지만 가장 중요한 표준 기구[5]인 IETF에는 누구나 기여할 수 있다. 이 기구는 모든 중요한 RFC를 관리한다.

RFC(Requests for Comments)와 인터넷 드래프트

대다수의 RFC는 표준 트랙(Standards Track)이라는 절차를 거쳐 탄생한다. 이 책을 통해 각기 다른 표준 트랙에 놓인 문서들을 언급할 것이다. 이 트랙이 어떻게 동작하는지 간단히 설명할 텐데, 내 추천을 어느 정도로 받아들여야 하는지 알려준다.

RFC는 인터넷 드래프트(Internet-Draft)로 시작한다. 이 문서는 표준 문서와 비슷해 보이지만, 이걸 가지고 구현해서는 안 된다. 이 문서는 명세서의 문제를 찾고 피드백을 제공하는 용도다.

인터넷 드래프트는 수명이 6개월이다. 발간되고 6개월이 지난 드래프트는 RFC로 승인되거나 업데이트된 드래프트로 교체되어야 한다. 이 둘 중 어느 것에도 해당하지 않는 드래프트는 만료되어 아무 용도로도 사용할 수 없다. 반면 드래프트가 승인되면, 바로 만료되고 RFC로 교체된다.

인터넷 드래프트의 이런 태생적인 만료 기간과 딱히 어떤 표준도 아니라는 특성 때문에 인터넷 드래프트를 이 책에서 언급하기는 좀 위험하다. 한편 API 설계는 굉장히 빠르게 변하는 분야이고 인터넷 드래프트가 있는 건 아무것도 없는 것보다 낫다. 책에서 언급하는 많은 인터넷 드래프트는 큰 변화 없이 RFC가 될 거라 가정하고 있다. 여태까진 이 가정이 잘 맞아떨어졌다. 여기서 언급한 몇 가지 인터넷 드래프트는 책이 쓰이는 동안 RFC가 되었다. 특정 드래프트가 RFC가 되지 못한다면

5 이 책의 목적에서 말이다. 만일 스크루나 볼트의 표준 크기 같은 게 필요하다면 ANSI나 ISO를 봐야 한다.

미리 사과한다.

RFC와 인터넷 드래프트는 코드명이 부여된다. 이것들을 설명할 때, 명세의 링크를 걸지는 않을 것이다. 그냥 코드명을 얘기하면 직접 찾아보면 된다. 예를 들어, HTTP/1.1 명세는 RFC 2616이라 부를 것이다. 인터넷 드래프트는 이름으로 부를 것이다. 예로 LINK와 UNLINK 메서드를 HTTP에 추가하는 제안서는 "draft-snell-link-method"라 부를 것이다.

이런 코드명을 보고 웹 검색을 통해 RFC나 인터넷 드래프트의 최신판을 찾을 수 있다. 책이 출간된 뒤 인터넷 드래프트가 RFC가 되면, 마지막 버전의 인터넷 드래프트가 RFC로 링크될 것이다.

W3C나 OASIS 표준을 설명할 때는, 별도의 코드명이 부여되지 않으므로 명세서를 링크할 것이다.

책에서 사용하는 규칙

이 아이콘은 팁, 제안 등을 표시한다.

이 아이콘은 경고나 주의를 표시한다.

코드 예제 사용하기

이 책은 독자의 일을 완료하는 데 도움을 줄 목적으로 썼다. 일반적으로, 책에 포함된 코드 예제는 독자의 프로그램이나 문서에 사용해도 된다. 코드의 상당 부분을 복제하는 것이 아니라면 별도의 허가를 구할 필요가 없다. 예를 들어, 이 책에서 나오는 코드의 여러 부분을 사용하는 프로그램을 작성한다면 허가가 필요하지 않다. 이 책 예제 코드의 여러 부분을 여러분 제품 문서에 넣는 데 허가가 필요하지 않다.

감사의 글

Glenn Block에게 감사한다. 그는 아이디어를 듣고 그 아이디어를 테스트할 수 있는 실제 코드를 짜는 데 많은 시간을 썼다. RESTFest의 Benjamin Young과 다른 모든 친구들에게 감사한다. 우리 실험에 참여해 엄청난 피드백과 조언을 해주었다.

Layer 7 Technologies에서 일하는 Dimitri Sirota와 Matt McLarty를 비롯한 마이크의 동료는 마이크가 이 집필 작업을 할 수 있도록 지지하고 격려해 주었다. 이 책의 전신인 『RESTful 웹 서비스』를 쓴 Sam Ruby와 Mike Loukides에게 감사하고, Sumana Harihareswara와 Leonard의 부인에게 감사한다. REST와 API에 관해 협력하고 대화를 나눌 수 있는 공간을 마련해준 멋진 커뮤니티, 특히 야후의 REST-Discuss, 구글 그룹스의 API-Craft, LibreList의 하이퍼미디어 그룹에 감사한다.

마지막으로 이 책의 초고를 읽고 비평과 격려를 보내준 Carsten Bormann, Todd Brackley, Tom Christie, Timothy Haas, Jamie Hodge, Alex James, David Jones, Markus Lanthaler, Even Maler, Mark Nottingham, Cheryl Phair, Sergey Shishkin, Brian Sletten, Mark Stafford, Stefan Tilkov, Denny Vrandečić, Ruben Verborgh, Andrew Wahbe에게 감사한다.

1장

RESTful Web APIs

웹 서핑하기

월드 와이드 웹이 인기를 끈 이유는 일반 사람들이 약간의 훈련으로 굉장히 유용한 일들을 할 수 있기 때문이다. 하지만 이런 측면 외에도 웹은 분산 컴퓨팅을 위한 강력한 플랫폼이기도 하다.

웹을 일반 사용자들이 사용할 수 있게 만든 원칙은 '사용자'가 자동화 소프트웨어 에이전트인 경우에도 동일하게 동작한다. 은행 사이에 돈을 송금하도록 설계된 소프트웨어나 실세계의 다른 일을 처리하는 소프트웨어 역시 사람이 사용하는 동일한 기본적인 기술을 사용해 작업을 수행할 수 있다.

이 책에서는 웹이 URL 명명 규칙, HTTP 프로토콜, HTML 문서 형식의 세 가지 기술로 이루어져 있다고 본다. URL과 HTTP는 간단하지만, 분산 프로그래밍에 사용하려면 일반 웹 개발자들보다 더 깊이 이해해야 한다. 이 책의 처음 몇 장은 이 내용을 이해하는 데 할애했다.

HTML 이야기는 좀 더 복잡하다. 웹 API 세계에는 HTML의 자리를 차지하려는 수십 개의 데이터 형식이 존재한다. 5장부터 이 책의 몇 장에 걸쳐 이 형식들을 살펴볼 것이다. 일단은 URL과 HTTP에 집중하고 HTML만 예로 들겠다.

월드 와이드 웹의 설계 원칙과 성공의 이유를 설명할 간단한 얘기로 시작한다. 이미 웹이 익숙하겠지만, 웹을 동작하게 하는 개념은 들어본 적이 없을 수 있으므로

간단하게 하도록 한다. 간단하지만 견고한 예제로 "애플리케이션 상태의 엔진으로서 하이퍼미디어" 같은 용어로 인해 혼란스러워질 경우를 방지할 수 있다.

자, 시작해 보자.

에피소드 1: 광고판

어느 날 앨리스는 마을을 돌아다니다가 광고판을 하나 보게 된다(그림 1-1).

그림 1–1 광고판

(이 가상의 광고판은 이 책을 위해 만든 실제 웹 사이트를 광고한다. 한번 방문해 보자.)

앨리스는 1990년대 중반을 기억할 만큼 나이를 먹었으므로 광고판에 처음 URL이 등장했을 때 사람들의 반응을 떠올려 본다. 먼저 사람들은 이 이상한 글자들을 놀려댔었다. 'http://'나 'youtypeitwepostit.com'이 무슨 의미인지 명확하지 않았기 때문이다. 그러나 20년이 지난 지금은 누구나 URL을 가지고 어찌 해야 할지 안다. 웹 브라우저의 주소 바에 입력하고 엔터를 치면 된다.

그리고 앨리스도 바로 그렇게 했다. 휴대 전화를 꺼내 브라우저의 주소 바에 http://www.youtypeitwepostit.com/을 입력한다. 우리의 첫 이야기는 요 아슬아슬한 시점에서 끝난다. 이 URL의 끝엔 뭐가 있을까?

리소스와 표현

이야기 중간에 끊어서 미안하지만 몇 가지 기본 용어를 설명해야 할 필요가 있다. 앨리스의 웹 브라우저가 지금 막 URL http://www.youtypeitwepostit.com/의 웹 서버에 HTTP 요청을 보내려 하고 있다. 하나의 웹 서버가 여러 개의 URL을 호스트할 수 있고, 각 URL은 서버의 다른 데이터에 접근하게 해준다.

보통 URL은 어떤 것들의 URL이라고 말한다. 예를 들면 제품, 사용자, 홈페이지의 URL 같은 식이다. 이렇게 URL이 붙은 것들의 기술적 용어가 리소스다.

URL http://www.youtypeitwepostit.com/은 리소스를 식별한다. 아마 광고판에 나온 웹 사이트의 홈페이지일 것이다. 하지만 앨리스의 웹 브라우저가 HTTP 요청을 보내는 이야기를 다 듣기 전엔 분명히 알 수 없다.

웹 브라우저가 리소스에 HTTP 요청을 보내면 서버는 응답으로 문서를 보낸다(주로 HTML 문서지만 때로 바이너리 이미지거나 다른 것일 수도 있다). 서버가 어떤 문서를 보내건, 그 문서를 그 리소스의 표현(representation)이라고 부른다.

각 URL은 하나의 리소스를 식별한다. 클라이언트가 URL에 HTTP 요청을 보내면, 하부 리소스의 표현을 받아오는 것이다. 클라이언트가 그 리소스를 직접 보는 일은 없다.

3장에서 리소스와 표현에 대해 더 많이 다룰 예정이다. 지금은 리소스와 표현이라는 용어를 사용해 주소 지정 가능성(addressability)의 원칙을 설명한다.

주소 지정 가능성

하나의 URL은 단 하나의 리소스만 식별한다. 만일 웹 사이트가 두 개의 개념적인 다른 리소스를 가지고 있다면, 이 사이트는 두 개의 다른 리소스를 각기 다른 URL로 처리할 것이다. 만일 웹 사이트가 이 규칙을 어긴다면 굉장히 좌절할 것이다. 레스토랑 웹 사이트들이 보통 이 문제를 많이 일으킨다. 아주 많은 경우에 전체 웹 사이트가 플래시 인터페이스로 되어 있고, 메뉴나 레스토랑의 위치 지도 등을 가리키는 URL을 찾을 수가 없다. 우리는 이런 것들을 각각의 주소를 가지고 얘기하고 싶은데 말이다.

주소 지정 가능성의 원칙은 모든 리소스는 각각 자신의 URL을 가져야 한다는 말이다. 애플리케이션에 중요한 무언가가 있다면 그것의 고유 이름, URL을 가져야 사

용자가 분명히 언급할 수 있다.

에피소드 2: 홈페이지

우리 이야기로 돌아가자. 앨리스가 광고판의 URL을 브라우저 주소 바에 입력하면 인터넷을 통해 http://www.youtypeitwepostit.com/에 있는 웹 서버로 HTTP 요청을 보낸다.

```
GET / HTTP/1.1
Host: www.youtypeitwepostit.com
```

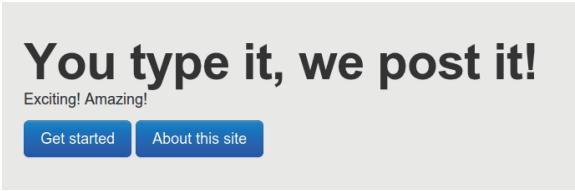
웹 서버는 이 요청을 처리하고(앨리스와 웹 브라우저는 어떻게 그렇게 하는지 알 필요가 없다) 응답을 보낸다.

```
HTTP/1.1 200 OK
Content-type: text/html

<!DOCTYPE html>
<html>
    <head>
        <title>Home</title>
    </head>
    <body>
        <div>
          <h1>You type it, we post it!</h1>
          <p>Exciting! Amazing!</p>

          <p class="links">
            <a href="/messages">Get started</a>
            <a href="/about">About this site</a>
          </p>
        </div>
    </body>
</html>
```

응답 앞부분의 200은 상태 코드(status code)인데, 응답 코드(response code)라고 부르기도 한다. 요청이 어떻게 되었는지를 서버가 클라이언트에 빠르게 알려주는 방법이다. HTTP 상태 코드는 굉장히 많은데 부록 A에서 모두 설명한다. 하지만 가장 많이 보게 될 녀석이 바로 이 녀석이다. 200(OK)은 요청이 아무 문제없이 잘 처리되었음을 나타낸다.

앨리스의 웹 브라우저는 응답을 HTML 문서로 해독해 화면에 표시한다(그림

그림 1–2 You Type It... 홈페이지



1-2).

이제 앨리스는 웹 페이지를 보고 광고판이 무슨 얘기를 하는지 이해할 수 있게 되었다. 트위터와 비슷한 마이크로블로깅 사이트를 광고하고 있었던 것이다. 광고판의 설명처럼 흥미롭진 않지만 예제로는 충분하다.

앨리스의 웹 서버와의 진정한 첫 상호 작용은 웹의 중요한 기능을 몇 가지 더 보여준다.

짧은 세션

이야기의 이 시점에서 앨리스의 웹 브라우저는 사이트의 홈페이지를 보여주고 있다. 그녀의 입장에서는 이 페이지에 '도착'했고, 그곳이 사이버 세계에서 그녀의 현재 '위치'다. 그런데 서버 입장에서 보면, 앨리스는 어느 곳에 있는 것이 아니다. 서버는 이미 그녀의 존재를 잊어버리고 말았다.

HTTP 세션은 하나의 요청 동안 유지된다. 클라이언트가 요청을 보내면 서버는 응답한다. 이 말은 앨리스가 밤에 전화기를 꺼뒀다가 다음 날 페이지를 내부 캐시로부터 불러들인 후, 링크 중 하나를 클릭해도 여전히 동작한다는 뜻이다(컴퓨터가 꺼지면 종료되는 SSH 세션과 비교해 보라).

앨리스가 이 웹 페이지를 6개월간 열어뒀다가 링크를 클릭해도, 웹 서버는 마치 그녀가 몇 초간 기다린 것처럼 동작할 것이다. 웹 서버는 앨리스를 걱정하며 밤새 기다리던 게 아니다. HTTP 요청을 하지 않으면 서버는 그녀가 존재하는지도 모를 것이다.

이 원칙은 때로 무상태(statelessness)라 부르기도 한다. 내 생각에는 좀 혼동을 주는 용어로 보이는데, 이 시스템에서 클라이언트와 서버가 모두 상태를 저장하고 있