세상의 속도를
따라잡고 싶다면

Do it!

웹 서버 | 채팅 서버 | 모바일 서버 | 위치 기반 서비스 서버 | JSON-RPC 서버

실제 서버로 구동 가능한 코드로 배운다!

Node.js 프로그래밍

자바스크립트로 만들고 클라우드 업로드까지! 서버 개발 전과정 총망라!

이 책에서 배운 모든 서버, 간편하게 바꿔 강력하게 써먹는다!

저자 직강
동영상 강의
제공

academy
대표강사 정재곤 지음

인천부평도서관
EM233950

node JS · 최신 Current 버전 사용!

이지스퍼블리싱

진정한 국내 최고의 프로그래밍 전문가,
정재곤 박사의 강의 노트를 공개한다!

지은이 정재곤 박사

4년 연속 IT 베스트셀러인 《Do it! 안드로이드 앱 프로그래밍》의 저자. 현재 서울디지털대학교 초빙 교수이자 벤처 기업의 CTO이기도 한 정재곤 박사는 모바일 전문 컨설팅으로 바쁜 일상을 보내고 있다. T아카데미에서 앱과 웹 분야의 전문 강의를 진행하고 있으며, 국내 대기업의 개발자들을 대상으로 한 강의와 여러 곳의 대학 특강을 통해 실무 전문가, 강사, 대학생 등을 가르쳐 왔다. 최신 IT 트렌드와 각종 프로그래밍 과정의 교육 커리큘럼을 개발하였으며, 미국과 일본 등 해외에서 최신 기술을 적용하는 프로젝트에 관여하고 있다.

약력

- 서울디지털대학교 컴퓨터공학과 초빙 교수
- ㈜유엔에스네트웍스 CTO
- 서울대학교 대학원 박사, GIS 전공
- SK텔레콤 T아카데미 전문 강사
- SK텔레콤 T아카데미 안드로이드 커리큘럼 등 교육 커리큘럼 개발 다수
- 국내/ 해외 모바일 앱 및 웹 서비스 컨설팅/ 설계/ 개발 경력 10년 이상
- 웹 서비스, 서버 플랫폼, 모바일 서버, 모바일 기기용 자바 VM 설계/ 개발 등 다수

집필 도서

《Do it! 안드로이드 앱 프로그래밍》, 2016년 1월, 이지스퍼블리싱
《자바+안드로이드를 다루는 기술》, 2014년 10월, 길벗

mike.jung.global@gmail.com

혼자 이 책을 공부하기가 막막하신가요?

Do it!

변화의 속도를
즐기게 될 것입니다.

실제 서버로 구동 가능한 코드로 배운다!

Do it! Node.js 프로그래밍

초판 1쇄 인쇄 | 2016년 09월 19일
초판 1쇄 발행 | 2016년 09월 26일

지은이 | 정재곤
펴낸이 | 이지연
펴낸곳 | 이지스퍼블리싱(주)

기획 및 책임 편집 | 배호종 / **편집 진행** | 홍연의 / **교정교열** | 오유진 / **디자인 및 전산편집** | 이승현
인쇄 | 보광문화사 / **제본소** | 정성문화사 / **영업 및 강의자료 PPT 문의** | 이주동(nlrose@easyspub.co.kr)

출판사 등록번호 | 제 313-2010-123호
주소 | 서울시 영등포구 당산로 41길 11, SK V1센터 323호
대표 전화 | 02-325-1722 / **팩스** | 02-326-1723

홈페이지 | www.easyspub.co.kr / **이메일** | service@easyspub.co.kr / **트위터** | @easyspub
DO IT! 스터디룸 | cafe.naver.com/doitstudyroom

잘못된 책은 구입한 서점에서 바꿔 드립니다.
이 책에 실린 모든 내용, 디자인, 이미지, 편집 구성에 대한 판면권과 저작권은 이지스퍼블리싱(주)과 지은이에게 있습니다. 허락 없이 복제하거나 옮겨 실을 수 없습니다.

서버 개발 입문자도 단계별로 하나씩, 레고 블록 쌓듯 완성품을 만든다!

사람들은 '직접 프로그램을 만들고 싶어'라고 생각했을 때 보통 웹을 가장 먼저 생각합니다. 그리고 어떤 사람들은 스마트폰에서 앱을 사용하다가 '나도 이런 걸 한번 만들어 보면 어떨까?' 하는 생각에 스마트폰 앱 프로그래밍의 세계로 뛰어듭니다. 예전에는 컴퓨터공학이나 전산학을 전공한 사람들만 관심을 갖던 분야인데 이제는 창업자들이나 비전공자들도 웹과 앱의 세계에 뛰어들고 있습니다.

처음 시작하는 서버 개발, 《Do it! Node.js 프로그래밍》과 함께 쉽고 빠른 Node.js를 만나 보세요!

웹이든 앱이든 개발에 약간 익숙해지면 '서버를 직접 만들어야겠구나'라고 생각합니다. 웹 페이지는 웹 서버에서 서비스하는 것이므로 사용자의 입맛에 맞는 데이터를 보여 주려면 웹 서버에 기능을 추가할 수 있어야 합니다. 앱의 경우에도 스마트폰에서 혼자 동작하는 앱보다는 서버의 데이터를 가져오거나 서버에 사용자 데이터를 저장하는 경우가 많아졌습니다. 하지만 오랜 기간 동안 서버 쪽 프로그램을 만드는 일은 "아주 복잡하고 어려운 일인걸!"이라고 말하는 사람들이 많았습니다. 특히 C나 C++와 같은 프로그래밍 언어를 사용해야 하고 눈에 보이는 결과 화면 없이 기능만을 구성해야 할 때가 많다 보니 더더욱 어렵게 느껴졌습니다.

이 어려움을 해결하는 데 도움을 준 것이 바로 Node.js입니다. Node.js로 서버를 만드는 사람들이 전 세계적으로 크게 늘어난 이유는 무엇보다도 쉬운 언어인 자바스크립트를 사용하며 속도도 아주 빠르기 때문입니다. 또한 개발자의 목적에 맞는 서버를 짧은 코드로 쉽게 만들 수 있다는 장점도 있습니다.

실제 개발 업무에서 사용하는 코드를 책에 그대로 옮겨 놓은 실전 코딩 노하우 전수!

막상 Node.js를 사용해서 서버를 만들려고 하면 '이 코드가 실제 서비스에도 사용될 수 있을까?' 하는 궁금증이 생깁니다. 왜냐하면 그동안 많은 프로그래밍 강의에서 알려주는 단편적인 코드들이 실무에서는 사용할 수 없다는 사실을 알기 때문입니다. 특히 Node.js로 웹 서비스나 앱을 구동하는 서버를 만들 때는 실제로 서비스할 목적으로 만들기 때문에 더더욱 실무 중심의 내용이 필요합니다.

이 책은 실제로 실무에서 사용하는 코드를 중심으로 구성하였습니다. 만약 여러분이 프로젝트에 투입되어 지금 당장 서버를 만들어야 한다면, 이 책을 빠르게 훑어보세요. 책에서 제공하는 서버 프로그램을 약간만 수정하면 바로 실무에 사용할 수 있습니다. Node.js 서버에 대해 좀 더 깊이 있게 이해하고 싶다면 책 내용을 하나하나 따라해 보세요. 이 책의 예제들은 단편적인 코드만 나열해서 해당 코드의 정의만 내리지 않고, 단계별로 하나씩 만들게 하므로 레고 블록을 쌓아 완성품을 만드는 구조로 전개됩니다. 소설을 읽을 때 이야기의 흐름을 따라가듯이 코드를 만드는 과정을 한 호흡으로 따라가면 누구나 실제로 구동하는 서버를 만들 수 있습니다.

여러분이 실제로 사용할 서버를 만들 수 있습니다!

이 책을 통해 Node.js API의 A to Z를 사전식으로 풀어내는 기존의 방식보다는 여러분이 실제로 사용할 서버를 어떻게 만들 수 있는지 알려주려고 합니다. 각자에게 필요한 서버를 만들면서 "서버, 그렇게 어렵지 않네? 서버를 만드는 일이 조금은 쉬워졌어!"라고 말할 수 있다면 좋겠습니다. 해외에서는 PHP나 자바로 만든 많은 서버들을 하나씩 Node.js로 바꾸고 있습니다. 이런 때에 국내 IT 시장도 해외보다 먼저 다양한 IT 기술이 시도되고 적용되기를 바랍니다. Node.js도 그중 하나가 될 수 있습니다. Node.js를 사용해 여러분이 원하는 서버를 직접 만들어서 독창적이고 앞선 기술을 클라우드에 올려 전 세계 사람들에게 서비스할 수 있기를 기대해 봅니다.

가을의 길목에서……. 지은이 정재곤 드림

mike.jung.global@gmail.com

"이 책 덕분에 Node.js의 매력에 푹 빠졌습니다!"

서버를 만들려는 앱 개발자에게 꼭 필요한 필독서!

앱 개발자 - 박영석

지금까지 앱을 개발하면서 서버 쪽도 공부해야겠다고 생각했지만 '어렵지 않을까?'하는 부담이 있었습니다. 하지만 이 책은 쉬운 Node.js를 더욱 쉽게 배우면서 서버를 만들 수 있도록 도와줍니다. 자바스크립트를 잘 몰라도 책 내용을 따라가다 보면 어느새 Node.js를 알게 됩니다. 짧은 코드 몇 줄로도 앱에서 요구하는 다양한 기능에 적합한 서버 기능을 만들 수 있다는 점이 신기하고 매력적입니다. 더구나 아마존 웹 서비스(AWS)에 올려 보니 이제야 전 세계 사람들에게 서비스할 수 있는 앱을 완성했다는 생각이 듭니다. 앱 개발자라면 꼭 보아야 하는 필독서입니다.

다양한 용도의 서버를 만들면서 기본기를 다지기 좋은 입문서!

웹 개발자 - 엄나윤

Node.js를 이해하는 데 어려움을 겪고 있을 때 이 책을 접하게 되었습니다. Node.js 입문자로서 알지 못했던 기본 개념, 그리고 웹 서버가 어떤 과정으로 만들어지고 개발이 진행되는지도 잘 설명되어 있습니다. 그래서 웹 서버 개발에 필요한 기본적인 내용을 정리하는 데 많은 도움을 받았습니다. 또한 입문자가 간과할 수 있는 '오류가 발생했을 때 대안', '효율적인 코드 작성 방법' 등을 짚어주어 필요한 코드를 작성할 때 적용할 수 있는 부분이 많았습니다. 다양한 용도의 서버 기능을 만들기 위해 매번 코드를 작성하기 때문에 기본기를 다지기에 매우 좋은 책입니다.

처음 Node.js를 공부하는 사람들에게 좋은 길잡이가 되어 줄 책

UI 개발자 - 배영

저는 UI 개발자이지만 자바스크립트를 공부하다가 서버에도 관심이 생겨 이 책을 보게 되었습니다. 네트워크와 서버에 대해 꼭 짚고 가야 할 내용은 빠뜨리지 않으면서도, 초심자들에게 너무 부담 주지 않는 내용이 마음에 들었습니다. 화면 UI 구성부터 서버, 데이터베이스까지 다루는 여러 가지 예제를 통해 자바스크립트의 폭넓은 활용성을 느낄 수 있습니다. 중간중간 나오는 '정박사의 한마디' 코너도 빼놓지 않고 읽어 보시면 본문에서 다 설명을 해주지 못한 자투리 지식도 넓힐 수 있습니다.

제가 Node.js 수업을 들을 당시에 이 책을 읽어 볼 수 있었다면 훨씬 도움이 되었을 텐데 정말 아쉽네요. 이 책을 통해 Node.js에서 사용하는 개념을 확실하게 이해할 수 있었고, 예전에는 무턱대고 따라 쓰기 바빴던 코드가 어떤 뜻인지 제대로 알 수 있어서 좋았습니다.

자바스크립트도 전혀 몰랐지만 이해하기 쉽게 잘 구성!

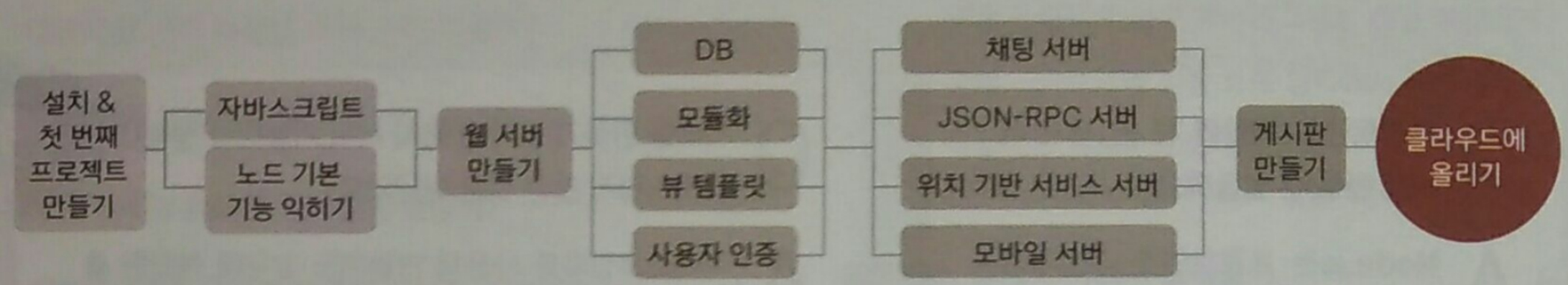
홍익대학교 컴퓨터공학과 3학년 - 강석원

서버라는 건 저에게는 솔직히 조금 먼 이야기였습니다. 서버를 구축해 웹 사이트를 돌리는 건 아주 오래전부터 해보고 싶었지만, 어디서부터 시작해야 할지 무엇부터 해야 할지 참 난감했었거든요. 대부분의 사람들이 저와 비슷할 것이라 생각합니다.

이 책은 서버 프로그래밍에 아주 쉽게 다가갈 수 있는 책입니다. Node.js는커녕 자바스크립트를 처음 다뤄 본 저조차도 이해하기 쉽도록 잘 구성되어 있습니다. 명쾌한 설명 덕분에 지루하지 않고, 재미있는 예제들도 많습니다. 여러분도 예제를 따라가다 보면 새로운 지식을 얻는 데 시간 가는 줄 모르실 겁니다. 특히 단순한 웹 서버 구현부터 DB, 채팅 서버, 위치기반 서버, 모바일 서버까지 다양한 서버들을 직접 만들어 볼 수 있는 점이 이 책의 장점입니다. 서버 프로그래밍에 대해 막연한 고민을 가지고 계신 분이라면, 이 책을 적극 추천합니다!

목적에 맞는 서버 제작 기법을 친절히 알려 주는 Node.js의 A to Z

이 책은 Node.js를 서버 프로그래밍 실무에 적용할 때 필요한 내용을 모두 담았습니다. 자바스크립트를 사용해 웹 서버뿐만 아니라 채팅 서버, 모바일 서버, 위치 기반 서비스 서버까지 만들고 아마존 클라우드에 올리는 방법까지 알려 줍니다.

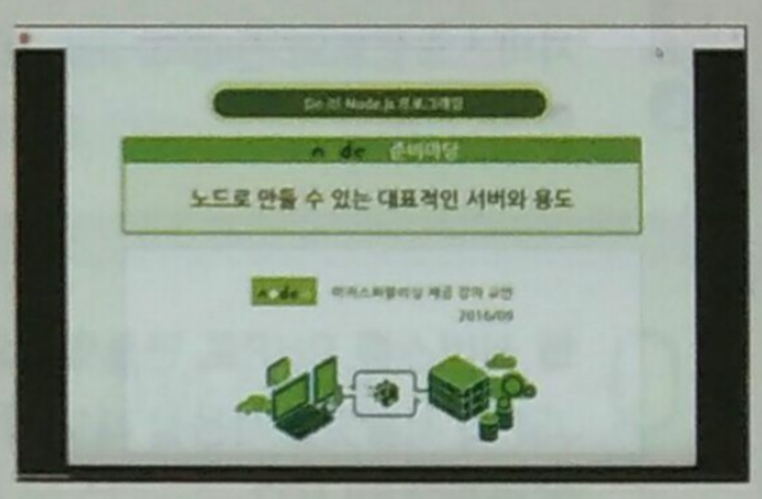


코드 간단하게 수정해서 실무에 바로 쓰는 실전용 서버 프로그래밍!

이 책에서 다루는 서버 프로그램은 간단한 수정만으로 실무에서 바로 사용할 수 있습니다. 한 단계씩 따라 만들다 보면 어느새 사용자에게 필요한 서버를 만들어서 서비스할 수 있을 것입니다.

동영상 강의로 학습 효율을 절정으로 끌어 올리세요!

Node.js를 더욱 쉽게 익힐 수 있는 저자 직강 동영상 강의도 유튜브에 무료로 공개되어 있습니다.



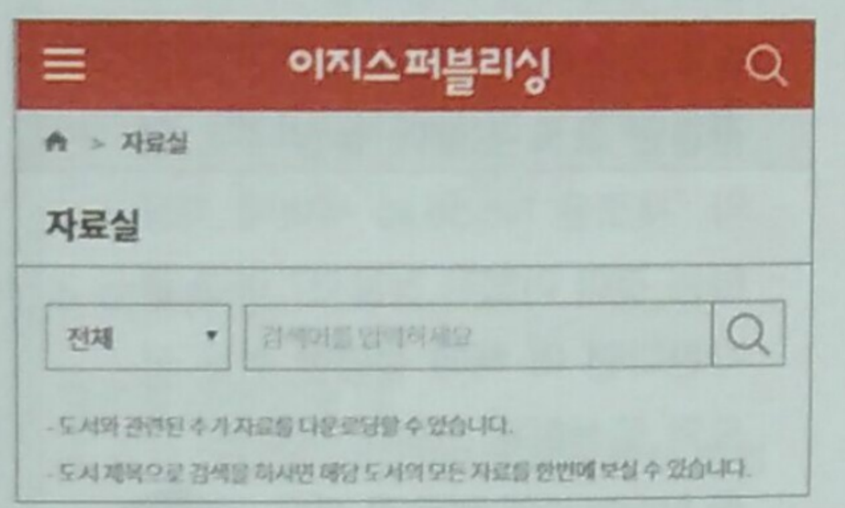
노드를 더욱 쉽게 익힐 수 있는 저자 직강! ▶

같이 공부할 친구를 찾아보세요!

Do it! 공부단(cafe.naver.com/doitstudyroom)에서 이 책을 같이 공부할 친구를 찾아보세요.
Node.js뿐 아니라 다른 IT 도서도 스터디할 수 있습니다.

실습 예제 다운로드 안내

이지스퍼블리싱 홈페이지에 오시면 다양한 학습 지원뿐 아니라 실습 예제 파일도 한꺼번에 내려받을 수 있습니다. 궁금한 점이 있다면 저자가 직접 운영하는 '노드 특강 사이트'를 이용해 보세요.



- 전체 실습 파일 한꺼번에 내려받기
  이지스퍼블리싱 홈페이지
  www.easyspub.co.kr → 자료실
  회원 가입을 해야 합니다(무료)

- 저자 운영 사이트에 질문 올리기
  Node.js 특강(http://tech-town.org)

Node.js가 왜 좋은데요? 제가 이 책을 봐도 될까요?

Node.js는 클라우드에 올려서 서비스하기 쉽고 저렴한 비용 면에서 다른 서버와 확연한 차이를 보입니다. 서버를 만들려는 사람과 서버를 옮기려는 사람들이 자주 묻는 질문을 보면서 이 책에서 다루는 내용을 확인해 보세요!

Q 프런트엔드 개발만 해 본 제가 서버를 만들 수 있을까요?

A Node.js는 프런트엔드 개발자가 다루는 자바스크립트 언어로 서버를 만들기 때문에 웹 개발을 해 보았던 사람이라면 더욱 쉽게 접근할 수 있죠. 코드의 양도 많지 않으니 그리 어렵지 않아요. 만약 자바스크립트 언어가 잘 기억나지 않아서 걱정이라면 이 책의 03장 '노드의 자바스크립트와 친해지기'를 보시면 됩니다!

Q 웹 서비스를 PHP로 만들었어요. JSP와 서블릿을 사용할 때도 있는데 바꿔야 하나요?

A 물론 웹 서비스를 Node.js로 바꿀지 결정하는 것은 여러분의 몫입니다. 하지만 Node.js를 한번 경험해 보면 JSP나 서블릿보다 빠른 성능 때문에 바꾸고 싶다는 생각이 스스로 들 것입니다. 또한 많은 사람들이 범용처럼 사용하는 자바스크립트 언어로 만들면서도 PHP만큼 간결한 코드 진행에 놀랍니다. 만약 '새로운 Node.js 서버에 적응하는 것이 어렵지 않을까?'라고 생각한다면 이 책에 설명한 쉬운 코드와 도식화된 그림이 여러분을 Node.js에 빠르게 적응하도록 도와줄 것입니다.

Q 서버를 만들기 쉽다면 혹시 서버 기능이나 성능이 만족스럽지 않은 것은 아닌가요?

A 자바스크립트를 사용해 만들어도 실무에 적합한 훌륭한 서버가 됩니다. 성능도 뛰어나고 안정성도 검증된 서버이니 실무에서 바로 사용할 수 있어요.

Q 혹시 보안에 취약점이 있진 않나요? 실무에서 Node.js를 쓰는 기업은 어떤 곳이 있나요?

A Node.js는 이미 안정성과 보안이 검증되어 기업이나 단체에서 사용하던 서버를 대체하고 있습니다. Node.js를 사용하는 대표적인 기업은 페이팔, 그루폰, 야후, 이베이 등이 있습니다. 현재는 대기업과 소규모 기업을 가리지 않고 Node.js를 도입하고 있습니다.

Q 앱을 만드는 데 채팅 기능도 들어가고 지도도 들어가요. 어떻게 서버 기능을 만들어서 서비스하죠?

A Node.js에는 채팅 서버를 쉽게 만들 수 있는 기능이 있습니다. 그 방법을 이 책의 10장에서 친절하게 알려 드립니다. 그리고 지도에 커피숍이나 은행처럼 특정 위치를 찾아서 보여주려면 위치 기반 서비스 서버가 필요합니다. 이런 위치 좌표를 다루는 서버는 보통 복잡한 과정을 거쳐야 만들 수 있지만 Node.js와 몽고디비를 사용하면 쉽게 만들 수 있습니다. 그 방법을 이 책의 12장에서 설명하는 기능을 활용해 만들 수 있으며, 이렇게 만든 서버는 클라우드에 올려서 전 세계로 서비스할 수 있습니다.

00 | 노드로 만들 수 있는 대표적인 서버와 용도

서버는 왜 필요할까? ····· 13

대표적인 서버 유형은 어떤 것이 있을까? ····· 16

웹 서버의 기능은 무엇일까? ····· 17

채팅 서버의 중요한 기능은 무엇일까? ····· 19

JSON-RPC 서버의 중요한 기능은 무엇일까? ····· 20

위치 기반 서비스 서버의 중요한 기능은 무엇일까? ····· 22

모바일 서버의 중요한 기능은 무엇일까? ····· 24

이 책에서 만들어 볼 화면 ····· 26

- 설치할 프로그램 목록 미리 확인하기
- 채팅 서버에 사용자 목록과 일대일 채팅 화면 만들기
- 위치 기반 서비스 서버에 커피숍 검색 화면 만들기

01 | 노드에 대해 알아보고 개발 도구 설치하기

노드란 무엇일까? ····· 30

- 노드의 비동기 입출력 방식
- 노드에서 구현하는 이벤트 기반 입출력 방식
- 노드를 더 쉽게 사용할 수 있게 하는 모듈

개발 도구 설치하기 ····· 38

- 설치할 프로그램 목록 미리 확인하기
- 자바 설치하기
- 이클립스 설치하기
- 노드 설치하기
- 이니드 스튜디오 플러그인 설치하기

02 | 노드 간단하게 살펴보기

첫 번째 노드 프로젝트 만들기 ····· 49

콘솔에 로그 뿌리기 ····· 55

- 명령 프롬프트 열고 자바스크립트 파일 실행하기
- 명령 프롬프트에 직접 코드 입력해서 실행하기
- 이클립스 프로젝트 안에 자바스크립트 파일 만들고 실행하기

프로세스 객체 간단하게 살펴보기 ····· 64

노드에서 모듈 사용하기 ····· 68

- 더하기 함수를 모듈로 간단히 분리하기
- module.exports로 메인 파일에 더하기 함수 호출하기

간단한 내장 모듈 사용하기 ····· 77

- 시스템 정보를 알려 주는 os 모듈
- 파일 패스를 다루는 path 모듈

03 | 노드의 자바스크립트와 친해지기

자바스크립트의 객체와 함수 이해하기 ····· 82

- 변수로 자료형 알아보기
- 자바스크립트의 함수 만들어 보기

배열 이해하기 ····· 90

- 배열의 모든 요소 하나씩 확인하기
- 배열에 값 추가 및 삭제하기
- splice( ) 메소드로 배열 요소 여러 개를 한꺼번에 추가하거나 삭제하기

콜백 함수 이해하기 ····· 100

- 함수를 호출했을 때 또 다른 함수를 파라미터로 전달하는 방법
- 함수 안에서 값을 반환할 때 새로운 함수를 만들어 반환하는 방법

프로토타입 객체 만들기 ····· 104

04 | 노드의 기본 기능 알아보기

주소 문자열과 요청 파라미터 다루기 ····· 108

주소 문자열을 URL 객체로 변환하기

요청 파라미터 확인하기

이벤트 이해하기 ····· 112

이벤트 보내고 받기

계산기 객체를 모듈로 만들어 보기

파일 다루기 ····· 117

파일을 읽어 들이거나 파일에 쓰기

파일을 직접 열고 닫으면서 읽거나 쓰기

버퍼 객체 사용하는 방법 알아보기

스트림 단위로 파일 읽고 쓰기

http 모듈로 요청받은 파일 내용을 읽고 응답하기

fs 모듈로 새 디렉터리 만들고 삭제하기

로그 파일 남기기 ····· 129

Do it! 도전 문제 **Mission01 / Mission02** ····· 132

05 | 웹 서버 만들기

간단한 웹 서버 만들기 ····· 135

클라이언트가 웹 서버에 요청할 때 발생하는 이벤트 처리하기

클라이언트에서 요청이 있을 때 파일 읽어 응답하기

파일을 스트림으로 읽어 응답 보내기

파일을 버퍼에 담아 두고 일부분만 읽어 응답 보내기

서버에서 다른 웹 사이트의 데이터를 가져와 응답하기

익스프레스로 웹 서버 만들기 ····· 150

새로운 익스프레스 프로젝트 만들기

use( ) 메소드를 사용한 미들웨어 설정 알아보기

미들웨어 직접 만들어 보기

여러 개의 미들웨어를 등록하여 사용하는 방법 알아보기

익스프레스의 요청 객체와 응답 객체 알아보기

redirect( ) 메소드로 무조건 다른 페이지로 이동하기

익스프레스에서 요청 객체에 추가한 메소드 알아보기

미들웨어 사용하기 ····· 166

static 미들웨어

body-parser 미들웨어

요청 라우팅하기 ····· 170

라우터 미들웨어 사용하기

URL 파라미터 사용하기

오류 페이지 보여 주기

express-error-handler 미들웨어로 오류 페이지 보내기

토큰과 함께 요청한 정보 처리하기

쿠키와 세션 관리하기 ····· 180

쿠키 처리하기

세션 처리하기

파일 업로드 기능 만들기 ····· 190

multer 미들웨어 설치해서 파일 업로드하기

클라이언트의 요청 처리 함수 추가하기

Do it! 도전 문제 **Mission03 / Mission04** ····· 198

06 | 데이터베이스 사용하기

몽고디비 시작하기 ····· 201

몽고디비란?

몽고디비 사용을 위한 프로그램 설치하기

몽고디비에 데이터를 추가하거나 조회하기

익스프레스에서 몽고디비 사용하기 ····· 208

새로운 프로젝트 만들기

mongodb 모듈을 사용하여 로그인 기능 만들기

사용자가 보내온 아이디와 비밀번호 비교하기

로그인 처리를 요청하는 패스에
라우팅 함수 추가하기

사용자 추가 기능 만들기

데이터베이스 관리 도구 사용하기

몽구스로 데이터베이스 다루기 ····· 226

몽구스 모듈 사용하기

몽구스로 사용자 인증하기

인덱스와 메소드 사용하기 ····· 235

사용자 리스트 조회 기능 추가하기

비밀번호 암호화하여 저장하기 ····· 243

virtual 함수 사용하기

스키마 객체의 virtual( ) 함수 사용법 알아보기

비밀번호 암호화하여 저장하는 코드 적용하기

MySQL 데이터베이스 사용하기 ····· 258

관계형 데이터베이스 간단하게 이해하기

MySQL 설치하기

화면이 있는 관리 도구 설치하기

MySQL을 사용하는 사용자 추가 기능 만들기

사용자 추가 요청을 처리하는 함수 만들기

MySQL에 들어 있는 사용자 정보로 로그인하기

Do it! 도전 문제 Mission05 / Mission06 ····· 280

07 | 익스프레스 프로젝트를 모듈화하기

모듈화 방법 자세히 살펴보기 ····· 283

다양한 방법으로 모듈 만들기

exports에 객체 지정하기

module.exports를 사용해서
객체를 그대로 할당하기

module.exports에 함수만 할당하기

exports와 module.exports를 함께 사용하기

require( ) 메소드의 동작 방식 이해하기

모듈을 분리할 때 사용하는 전형적인 코드 패턴

함수를 할당하는 코드 패턴

인스턴스 객체를 할당하는 코드 패턴

프로토타입 객체를 할당하는 코드 패턴

사용자 정보 관련 기능을 모듈화하기 ····· 300

스키마 파일을 별도의 모듈 파일로 분리하기

사용자 처리 함수를 별도의 모듈 파일로
분리해 보기

설정 파일 만들기 ····· 307

설정 파일 분리하기

설정 파일에 데이터베이스 스키마 정보 넣기

설정 파일에 라우팅 정보 넣기

UI 라이브러리로 웹 문서 예쁘게 꾸미기 ····· 316

Semantic UI 라이브러리로 웹 문서 꾸미기

Card 컴포넌트 추가하기

<table> 태그로 로그인 입력 상자와 버튼 추가하기

<style> 태그로 전체 화면 모양 만들기

로그인 카드를 모바일 화면에 맞도록
CSS 조정하기

반응형 웹으로 웹 문서를 구별해서 보여 주기

사용자 리스트 웹 문서 수정하기

사용자 조회에 응답하는 웹 문서 꾸미기

사용자 추가용 웹 문서 꾸미기

Do it! 도전 문제 Mission07 / Mission08 ····· 334

08 | 뷰 템플릿 적용하기

ejs 뷰 템플릿 사용하기 ····· 337

뷰 템플릿으로 로그인 웹 문서 만들기

뷰 템플릿으로 사용자 리스트 웹 문서 만들기

뷰 템플릿으로 사용자 추가 웹 문서 만들기

jade 뷰 템플릿 사용하기 ····· 350

jade로 HTML 문서 만들기

jade 템플릿으로 로그인 웹 문서 만들기

jade 템플릿으로 사용자 리스트 웹 문서 만들기

jade 템플릿으로 사용자 추가 웹 문서 만들기

Do it! 도전 문제 **Mission09 / Mission10** ····· 364

09 | 패스포트로 사용자 인증하기

패스포트로 로그인하기 ····· 367

로컬 인증을 위해 데이터베이스 스키마와 패스포트 설정하기

로그인과 회원가입 화면을 만들기 위한 라우팅 함수 등록하기

로그인과 회원가입 화면을 만들기 위한 뷰 템플릿 만들기

로그인과 회원가입 기능 실행하여 확인하기

패스포트 관련 코드를 모듈화하기 ····· 390

페이스북으로 로그인하기 ····· 396

Do it! 도전 문제 **Mission11 / Mission12** ····· 410

10 | 채팅 서버 만들기

socket.io 사용하기 ····· 413

socket.io를 사용하기 위해 모듈 설치하기

app.js 메인 파일에 기본 코드 하나씩 추가하기

사용자가 웹 브라우저에서 볼 웹 문서 만들기

서버에 보낸 메시지를 그대로 받기

일대일 채팅하기 ····· 431

그룹 채팅하기 ····· 438

방 만들기

그룹 채팅에서 메시지 보내기

채팅 웹 문서 예쁘게 꾸미기 ····· 449

Do it! 도전 문제 **Mission13 / Mission14** ····· 464

11 | JSON-PRC 서버 만들기

JSON-PRC를 웹 서버에 적용하기 ····· 467

JSON-RPC 모듈 설치하여 사용하기

echo 함수 만들어 실행하기

echo 함수의 오류 테스트하기

계산기 모듈 추가하여 실행하기 ····· 480

1단계: 핸들러 모듈 파일 만들기

2단계: 핸들러 모듈 파일 등록하기

3단계: 클라이언트 웹 문서에서 호출하기

데이터베이스에서 사용자 리스트 조회하기 ····· 485

데이터 부분을 암호화하기 ····· 491

Do it! 도전 문제 **Mission15 / Mission16** ····· 496

12 | 위치 기반 서비스 서버 만들기

커피숍 위치 저장하기 ····· 499

커피숍 위치 정보 다루기

커피숍 스키마 만들기

커피숍 정보를 추가하고 커피숍 리스트 조회하기

가장 가까운 커피숍 찾기 ····· 513

영역 안의 커피숍 찾기 ····· 519

반경 안의 커피숍 찾기 ····· 524

지도에 커피숍의 위치 표시하기 ····· 528

지도에 내 위치 보여 주기

가장 가까운 커피숍을 찾아 지도 위에 보여 주기

일정 범위 안의 커피숍을 찾아 지도 위에 보여 주기

Do it! 도전 문제 Mission17 / Mission18 ····· 546

13 | 모바일 서버 만들기

모바일 단말에서 웹 서버로 요청하기 ····· 549

안드로이드 앱 개발 도구 설치하기

실제 모바일 단말에 연결하기

모바일용 서버 프로젝트 만들기

안드로이드 앱 프로젝트 만들기

모바일 단말 관리 기능 만들기 ····· 565

데이터베이스 스키마 추가하기

라우팅 함수 추가하기

단말 정보 추가를 요청하는 앱 만들기

웹 브라우저에서 단말 리스트 조회하기

모바일 단말로 푸시 메시지 전송하기 ····· 582

단말의 등록 ID를 확인하여
모바일 서버로 전송하기

웹 브라우저에서 푸시 메시지 전송하기

Do it! 도전 문제 Mission19 / Mission20 ····· 594

14 | 게시판 만들기

스키마를 추가하고
페이지 단위 조회 방식 이해하기 ····· 594

페이지 단위로 조회하기

글쓰기와 글 조회 기능 만들기 ····· 604

페이지 단위로 글 목록 조회하기 ····· 612

1단계 - 라우팅 함수 만들기

2단계 - config.js 파일에 라우팅 함수 등록하기

3단계 - 응답 웹 문서를 구성할 뷰 템플릿 만들기

4단계 - 글 목록 조회를 요청하는 웹 페이지 만들기

Do it! 도전 문제 Mission21 / Mission22 ····· 620

15 | 클라우드에 올리기

Git 클라이언트 설치하기 ····· 623

헤로쿠 클라우드에 올리기 ····· 623

헤로쿠 사이트에 회원가입하고 앱 만들기

Toolbelt 설치하기

배포에 필요한 설치 및 준비하기

소스 업로드하기

오픈시프트 클라우드에 올리기 ····· 638

아마존 클라우드에 올리기 ····· 648

배포에 필요한 기능 알아보기 ····· 660

비정상으로 종료되었을 때
자동으로 다시 시작하기

클러스터링 사용하기

레디스 사용하기

레디스로 subscribe, publish 하기

채팅 서버에서 레디스를 사용하는 샘플

00 노드로 만들 수 있는 대표적인 서버와 용도

이 책에서는 노드제이에스(Node.js) 또는 노드(Node)라고 부르는 것을 이용해서 서버를 만드는 방법을 설명합니다. 그런데 서버라고 하면 왠지 복잡하고 어려울 것 같다고 생각하는 분들이 많습니다. 그러나 요즘에는 노드를 이용하면 오히려 여러 종류의 서버를 예전보다 더 쉽고 간단하게 만들 수 있습니다.

지금부터 다양한 목적의 서버를 만드는 방법에 대해 자세하게 설명할 것입니다. 그러나 모든 종류의 서버를 다룰 수는 없기 때문에 현업에서 많이 쓰는 서버를 중심으로 알아보겠습니다.

도해로 정리하기 ····· 개발자, 분야, 프로젝트에 따른 서버 유형 길잡이

웹 서버 ➡
- 누가? 웹 디자이너, 웹 퍼블리셔, 웹 개발자
- 어디에? 홈페이지, 쇼핑몰 사이트, 기업 업무 페이지
- 언제? 웹 서비스 프로젝트, 기업 업무 개발 프로젝트

채팅 서버 ➡
- 누가? 웹 개발자, 앱 개발자, 윈도우 프로그램 개발자
- 어디에? 웹 채팅, 스마트폰 채팅, 실시간 의견 공유
- 언제? 채팅 앱 개발 프로젝트, 공유 솔루션 개발 프로젝트

JSON-RPC 서버 ➡
- 누가? 앱 개발자, 윈도우 프로그램 개발자
- 어디에? 기능 중심의 사이트, 기업 업무 페이지, 스마트폰 앱
- 언제? 기업 업무 개발 프로젝트, 앱 개발 프로젝트

위치 기반 서비스 서버 ➡
- 누가? 웹 개발자, 앱 개발자
- 어디에? 모바일 웹, 스마트폰 앱
- 언제? 매장 찾기 프로젝트, 친구 찾기 프로젝트

모바일 서버 ➡
- 누가? 앱 개발자
- 어디에? 스마트폰 앱
- 언제? 앱 개발 프로젝트, 단말 관리 프로젝트

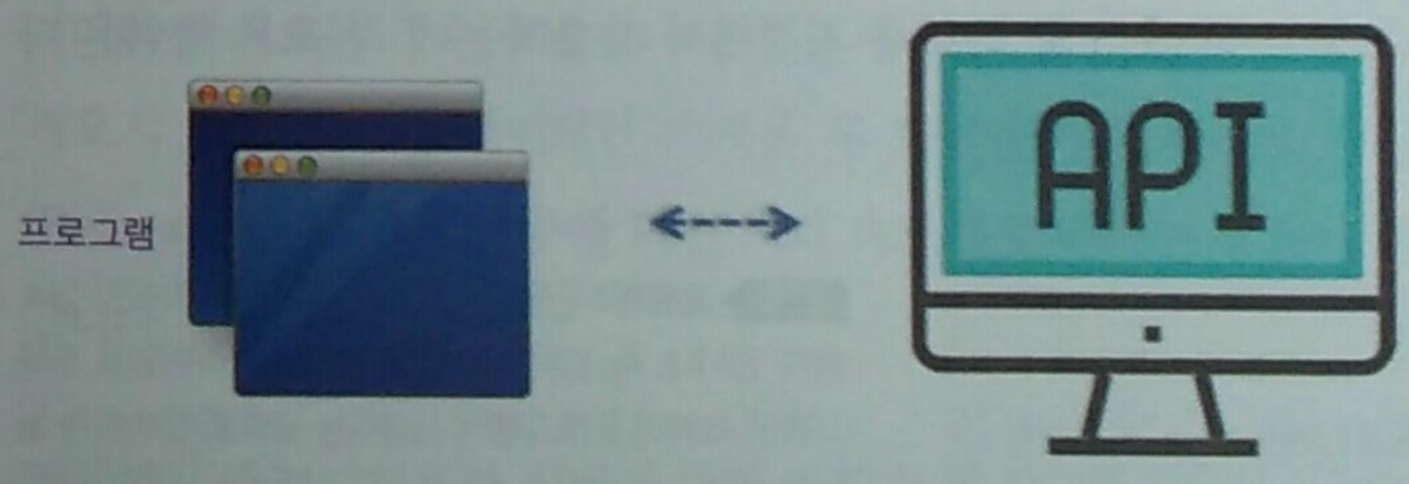
00-1 | 서버는 왜 필요할까?

최근에 안드로이드(Android)나 아이오에스(iOS) 기반의 모바일 앱을 만드는 개발자는 서버에 대한 지식이 전혀 없는 상태로 웹 서버나 채팅 서버를 만들기도 합니다. 이런 경우라도 책 내용대로 서버를 만들다 보면 필요했던 서버 기능을 대부분 문제없이 만들 수 있을 것입니다. 그렇다면 노드가 무엇인지 배우기 전에 서버가 무엇이며, 어떤 종류의 서버가 있는지 알고 있어야겠죠? 먼저 서버에 대해 간단히 알아본 후 서버의 기능을 하나씩 설명하겠습니다.

우리는 PC나 스마트폰에서 여러 가지 프로그램(Program)을 사용합니다. 이 프로그램을 **애플리케이션(Application)**이라고 부르며, 스마트폰에서는 **앱(App)**이라고도 부릅니다. PC에서 동작하는 프로그램 중 대표적인 것이 파워포인트, 워드, 엑셀과 같은 문서 프로그램인데 이런 프로그램은 사용법만 익혀도 충분합니다. 하지만 PC나 스마트폰에서 동작하는 프로그램을 직접 만들어보고 싶다면 API(Application Programming Interface)를 알아야 합니다.

API란 프로그램을 쉽게 제작할 수 있게 미리 만들어 놓은 것들의 모음이라고 할 수 있습니다. 예를 들면, 윈도우(Windows)라는 PC용 운영체제(OS)에서 동작하는 프로그램을 쉽게 만들 수 있도록 윈도우용 API가 제공됩니다. 윈도우용 API에는 PC 화면에 버튼처럼 보이는 것들을 미리 만들어 넣어두었기 때문에 간단한 코드 몇 줄로도 버튼을 만들 수 있습니다. 스마트폰의 경우에도 안드로이드 OS나 아이폰용 OS를 위한 API가 있습니다. 이렇게 우리는 운영체제의 API를 이용해서 하나의 단말 기기(Device)에서 동작하는 프로그램을 만들게 됩니다.

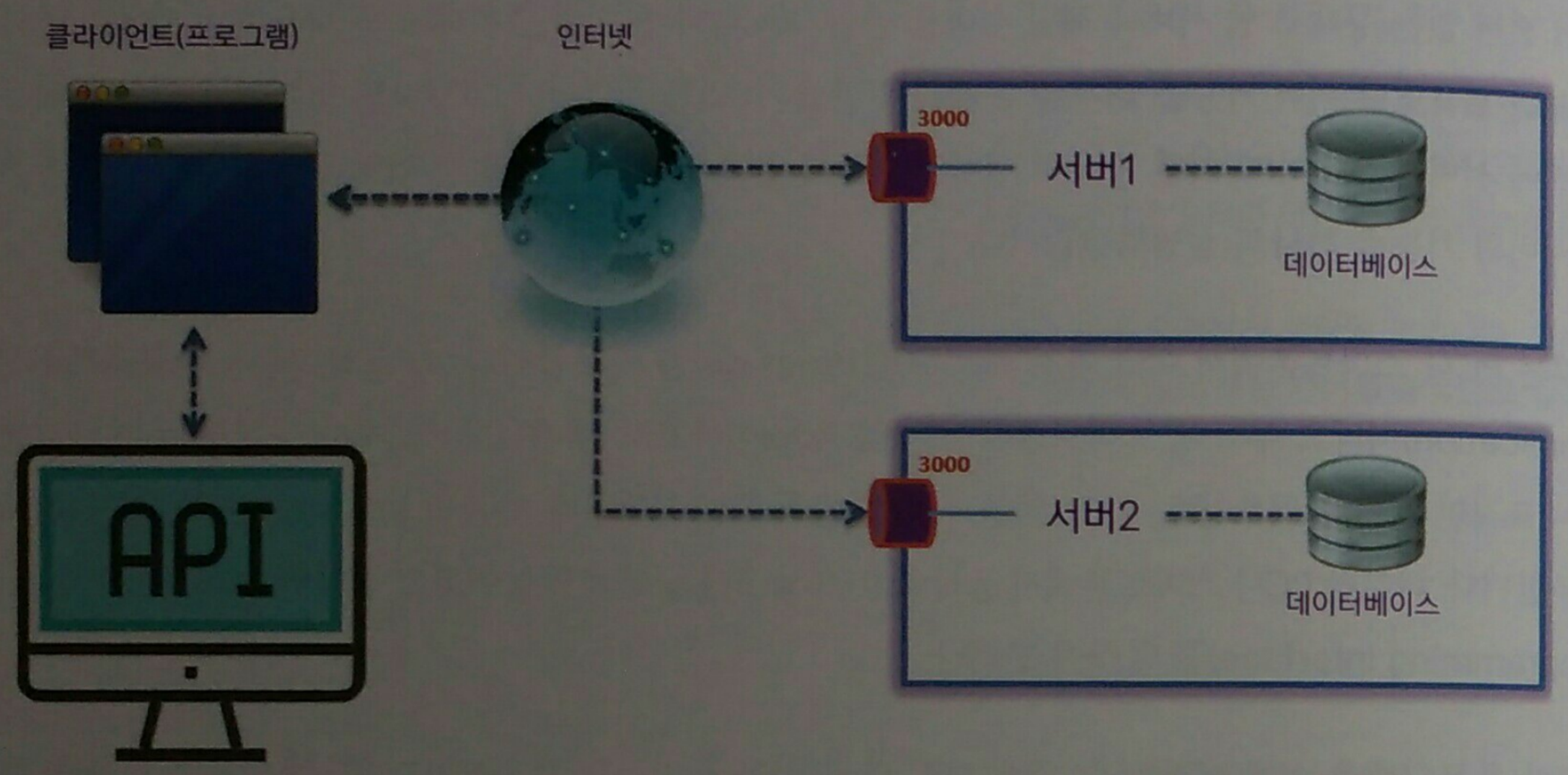
용어! **단말 기기란?** PC나 스마트폰 같은 독립된 기기(Device)를 말합니다. 이 책에서는 줄여서 '단말'이라고 부르겠습니다.



▲ API를 사용해 만든, 하나의 단말에서 독립적으로 실행되는 프로그램

요즘에는 인터넷을 이용해서 다른 장소에 있는 데이터를 가져와 보여주는 경우가 많습니다. 인터넷 익스플로러나 크롬 웹 브라우저를 열고 구글이나 네이버 같은 포털 사이트에 접속하는 것도 웹 브라우저라는 프로그램이 인터넷을 사용해서 웹 서버에 웹 문서를 달라고 요청할 수 있기 때문에 가능한 것입니다.

여러분이 프로그램을 만들 때도 인터넷을 이용해서 다른 곳의 데이터를 가져와 보여주거나 다른 곳에서 보내온 데이터를 여러분의 컴퓨터에 저장하는 경우가 많이 생깁니다. 이렇게 다른 곳에 있는 단말에 데이터를 달라고 요청하는 프로그램을 **클라이언트(Client)**, 다른 곳에서 요청받은 명령을 처리해 주는 프로그램을 **서버(Server)**라고 합니다.



▲ 클라이언트는 서버에 접속하여 데이터를 가져옴

여러분이 만든 프로그램이 인터넷에 연결하기 위해서는 단말에 **네트워크 카드(Network Card 또는 Ethernet Card)**가 들어있어야 합니다. 요즘에는 유선 인터넷뿐만 아니라 무선 인터넷을 사용하는 경우가 많아서 단말에는 와이파이(WiFi)나 3G/LTE로 인터넷에 접속할 수 있는 네트워크 카드가 들어 있습니다. 두 개의 서로 다른 단말에 모두 네트워크 카드가 들어 있다면 한쪽은 클라이언트, 다른 한쪽은 서버 역할을 할 수 있습니다. 이때 서버는 포트(Port)를 지정하여 그 포트로 요청을 받을 수 있습니다. 하나의 단말에서 사용할 수 있는 포트는 보통 0~65535 중 하나를 지정해서 사용하는데 각 포트 영역은 다음과 같이 구분됩니다.

포트 번호	설명
0번 ~ 1023번	잘 알려진 포트(Well-known port)
1024번 ~ 49151번	등록된 포트(Registered port)
49152번 ~ 65535번	동적 포트(Dynamic port)

용어! **포트란?** 단말에 접속하기 위한 '논리적인 접속 위치'입니다. 즉, 인터넷을 사용할 때는 클라이언트 프로그램은 서버의 프로그램이 지정하는 포트로 접속하게 됩니다.

윈도우나 안드로이드 같은 OS가 시작될 때는 이미 잘 알려진 포트(0~1023번)에서 대부분을 사용합니다. 따라서 여러분이 직접 만드는 서버 프로그램은 보통 1024번 이상의 포트 번호를 사용합니다.

서버를 만들어 실행하면 지정된 포트에서 클라이언트로부터 요청을 받아 처리하게 됩니다. 또한 많은 경우에 서버는 데이터베이스에 연결할 수 있도록 구성되기 때문에 클라이언트에서 보내온 데이터를 저장하거나 저장된 데이터를 조회한 후 클라이언트에 보내 줍니다. 서버 중에서 여러분이 많이 사용하는 웹 브라우저에서 접속하는 서버를 **웹 서버**라고 하며, HTTP 프로토콜을 사용합니다.

여기에서 **프로토콜**(Protocol)이란 데이터를 서로 어떤 형태로 주고받을 것인지를 정한 것으로, 간단하게 데이터의 형태라고 생각하면 이해하기 쉽습니다. 따라서 HTTP란 인터넷상에서 데이터를 어떻게 주고받으면 되는지를 정의해 둔 것이고 모든 웹 브라우저가 이 방식을 따른다고 할 수 있습니다. 웹 브라우저에서 HTTP로 요청한 데이터를 받은 웹 서버는 필요한 작업을 수행한 후 HTTP로 응답을 보냅니다. 웹 브라우저에서 볼 수 있는 웹 문서는 보통 HTML과 CSS, 그리고 자바스크립트를 이용해서 만들어 둔 것입니다. 웹 브라우저가 웹 서버로부터 데이터를 전달받으면 웹 브라우저 화면에 웹 문서를 띄우게 되는데, 이 웹 문서가 클라이언트 프로그램의 역할을 하게 됩니다. 따라서 한번 화면에 띄운 웹 문서는 필요할 때마다 웹 서버로 또 다른 웹 문서를 요청하여 받아옵니다.

▲ 웹 브라우저에서 웹 서버에 요청하여 웹 문서를 가져오는 경우

스마트폰과 같은 모바일 단말에서는 웹 문서를 웹 서버가 아닌 단말 내부에 저장해 두었다가 화면에 띄운 후 웹 서버로 필요한 데이터만 요청하는 방식을 사용하기도 합니다. 이때 **에이잭스**(Ajax) 방식으로 데이터만 받아오는 경우가 많습니다. 이렇게 하면 매번 웹 문서를 가져오지 않고 필요한 데이터만 가져온 후 화면을 부분적으로 갱신(Refresh)할 수 있기 때문에 화면 처리 속도가 빨라지는 장점이 생깁니다.

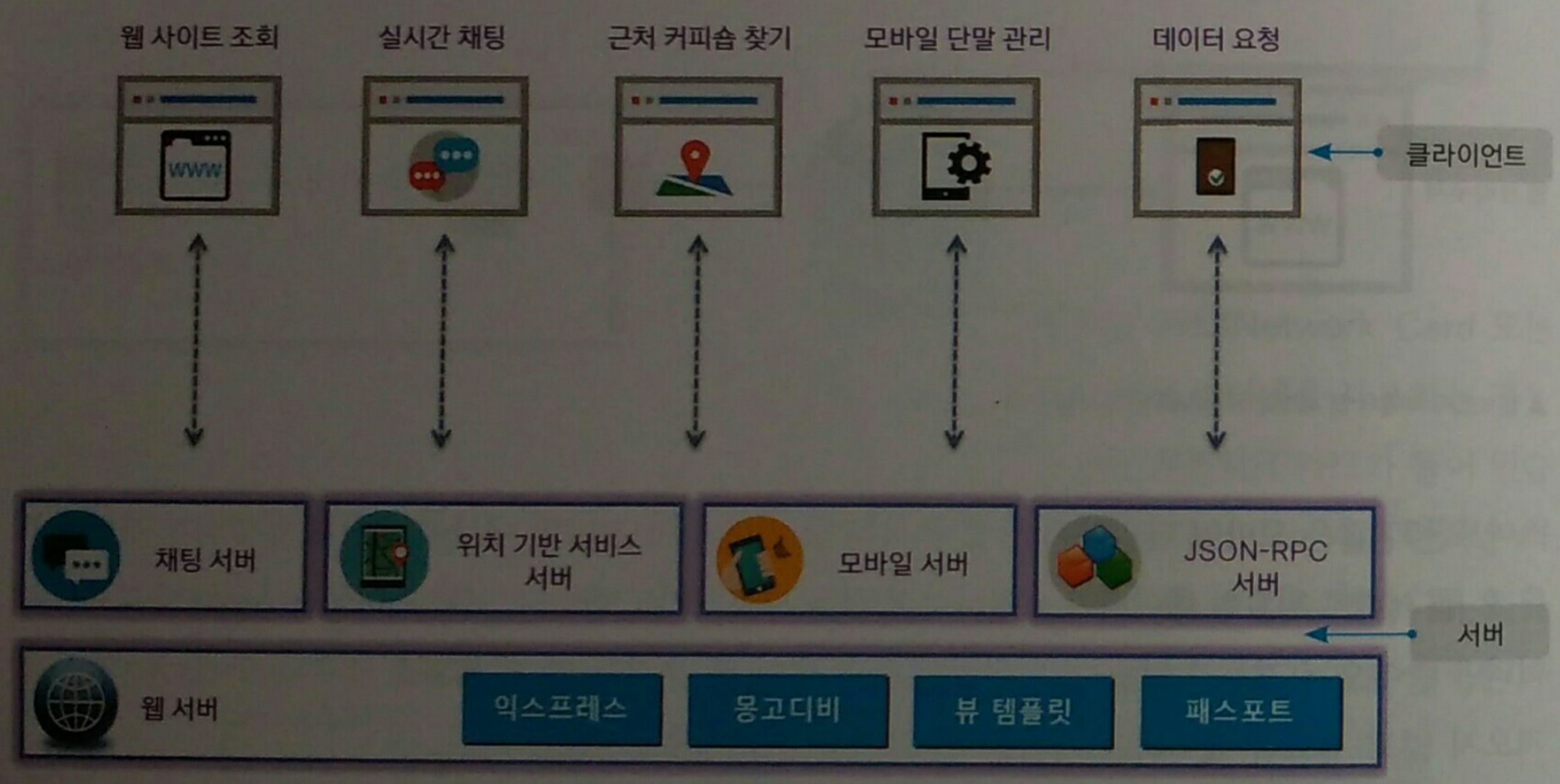
용어! **에이잭스**(Ajax)는 비동기식 자바스크립트 XML(Asynchronous Javascript And XML)의 약자로 웹 서버에서 웹 문서를 받아오는 것이 아니라 데이터만 받아오기 위한 방법과 기술을 말합니다.

지금까지 클라이언트와 서버에 대해 간략하게 설명했습니다. 서버에 대해 조금이나마 이해를 했다면 인터넷 환경에서 가장 많이 사용하는 서버가 웹 브라우저에서 접속하는 웹 서버라는 것도 이해했을 것입니다. 이 책의 독자들은 이미 프로그램을 만들어 본 경험이 있는 분이 많을 것이라 생각되니 서버에 대한 이야기는 이 정도로 정리하겠습니다. 다음 장에서는 이 책에서 다루는 대표적인 서버의 유형에 대해 살펴보겠습니다.

00-2 | 대표적인 서버 유형은 어떤 것이 있을까?

서버의 유형은 굉장히 다양합니다. 따라서 만들려는 애플리케이션이 무엇인지에 따라 필요한 서버 유형이 달라집니다. 예를 들어, 카카오톡이나 라인처럼 실시간 채팅 앱을 만들고 싶다면 실시간 채팅 기능을 지원하는 채팅 서버가 있어야 합니다. 그리고 다양한 센서로부터 값을 전달받아 동작하는 **사물인터넷**(IOT, Internet Of Things) 기능을 만들고 싶다면 여러 센서들과 데이터를 주고받을 수 있는 서버가 있어야 합니다. 요즘에는 모바일 앱을 많이 만들기 때문에 스마트폰 같은 모바일 단말에 필요한 기능을 제공하는 모바일 서버가 필요합니다.

이 책에서는 다양한 서버 중에서 프로그래머들이 많은 관심을 가지는 서버 유형 위주로 살펴봅니다. 다음 그림은 이 책에서 다룰 대표적인 서버 유형을 정리한 것입니다.

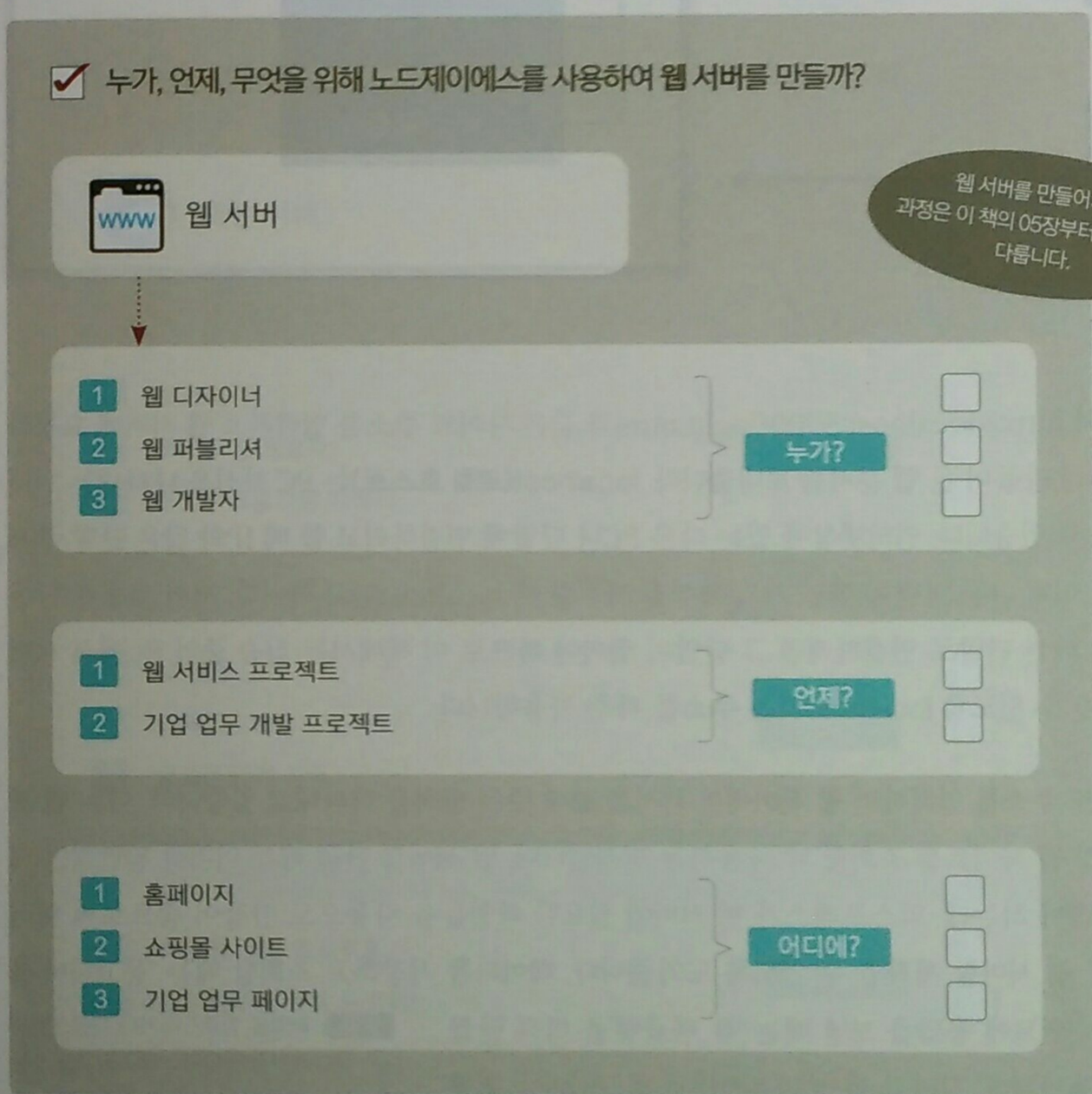


▲ 이 책에서 다룰 대표적인 서버 유형

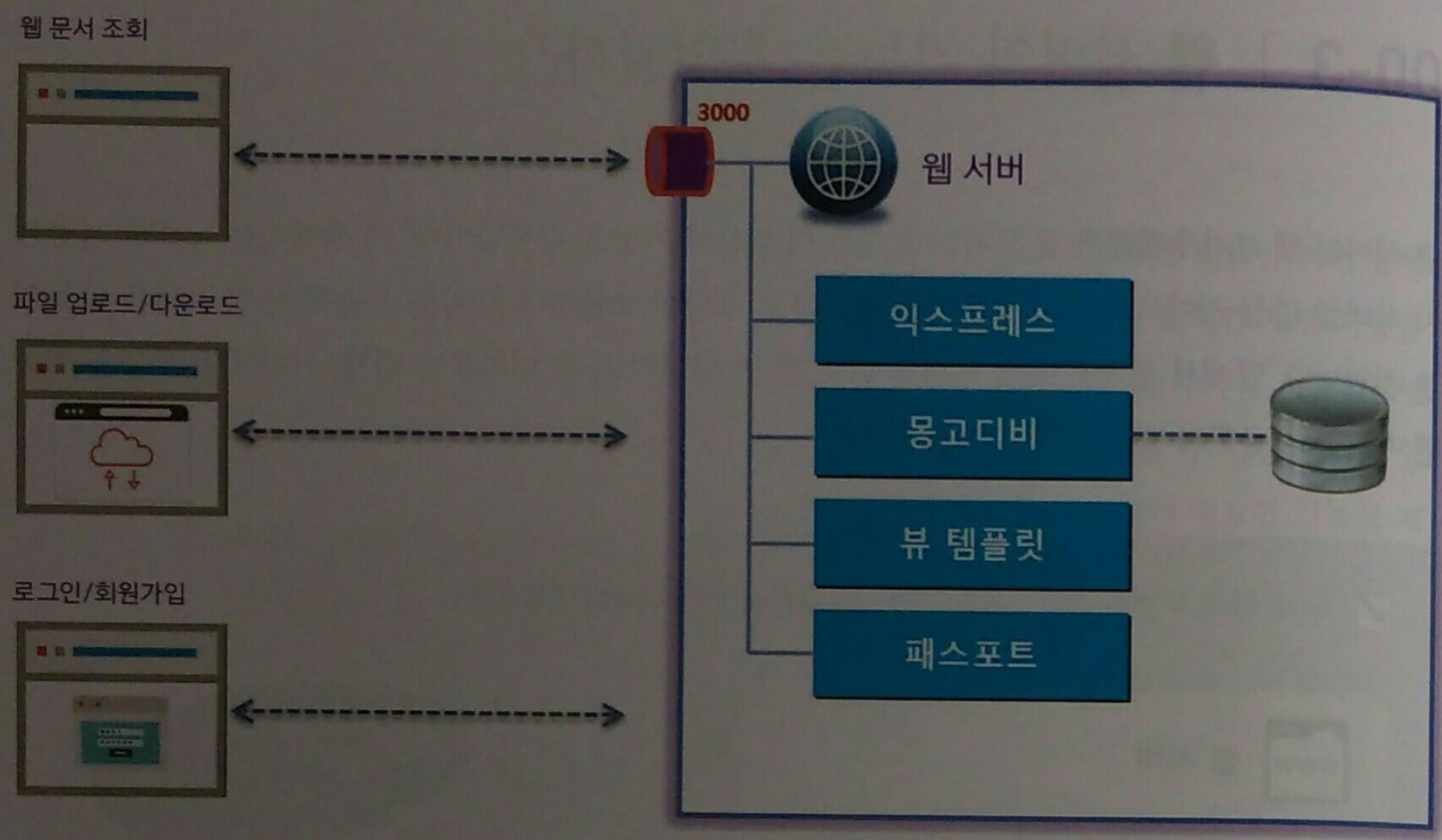
그림의 윗부분은 클라이언트이고 아랫부분은 서버를 나타낸 것입니다. 서버의 유형은 크게 다섯 가지로 나뉘어 있습니다. 웹 서버는 모든 서버의 기본입니다. HTTP 프로토콜로 데이터를 주고받을 수 있는 웹 서버를 만들고 나면 그 위에 채팅 서버나 위치 기반 서비스 서버 등을 위한 기능을 추가해서 새로운 서버를 만들 수 있습니다. 그러면 웹 서버에 대한 설명부터 시작하여 그 위에 기능을 추가해서 만들 수 있는 서버에 대해 하나씩 살펴보겠습니다.

00-3 | 웹 서버의 기능은 무엇일까?

웹 서버는 웹 브라우저에서 웹 문서를 요청할 때 필요한 기능을 수행합니다. 이 책에서는 노드를 사용해서 서버를 만들 것이므로 먼저 노드에서 제공하는 다양한 **모듈**(Module)을 사용하여 웹 서버를 만들어 볼 것입니다. 앞에서 살펴본 서버 유형을 다시 한 번 확인한 다음 여러분이 언제, 어디에서, 무엇을 위해 웹 서버를 사용할지 스스로 체크해 보세요.



여기에서는 서버가 어떻게 동작하는지 간단히 알아보고 넘어가겠습니다. 여러분이 만들 웹 서버는 웹 프레임워크인 **익스프레스**(Express)로 기본 구조를 만들고 **몽고디비**(MongoDB)를 이용해서 데이터를 저장하거나 조회합니다. **뷰 템플릿**(View Template)은 클라이언트에 응답을 보낼 때 사용하려고 미리 **웹 문서의 원형**(Template)을 만들어 놓은 것이며, **패스포트**(Passport)는 사용자 로그인이나 회원가입을 위해 사용합니다. 그렇다면 이런 것들을 사용해 만드는 웹 서버의 주요 기능은 무엇일까요? 다음 그림을 보면서 먼저 개략적인 구조만이라도 이해해 보겠습니다.



▲ 웹 서버의 주요 기능

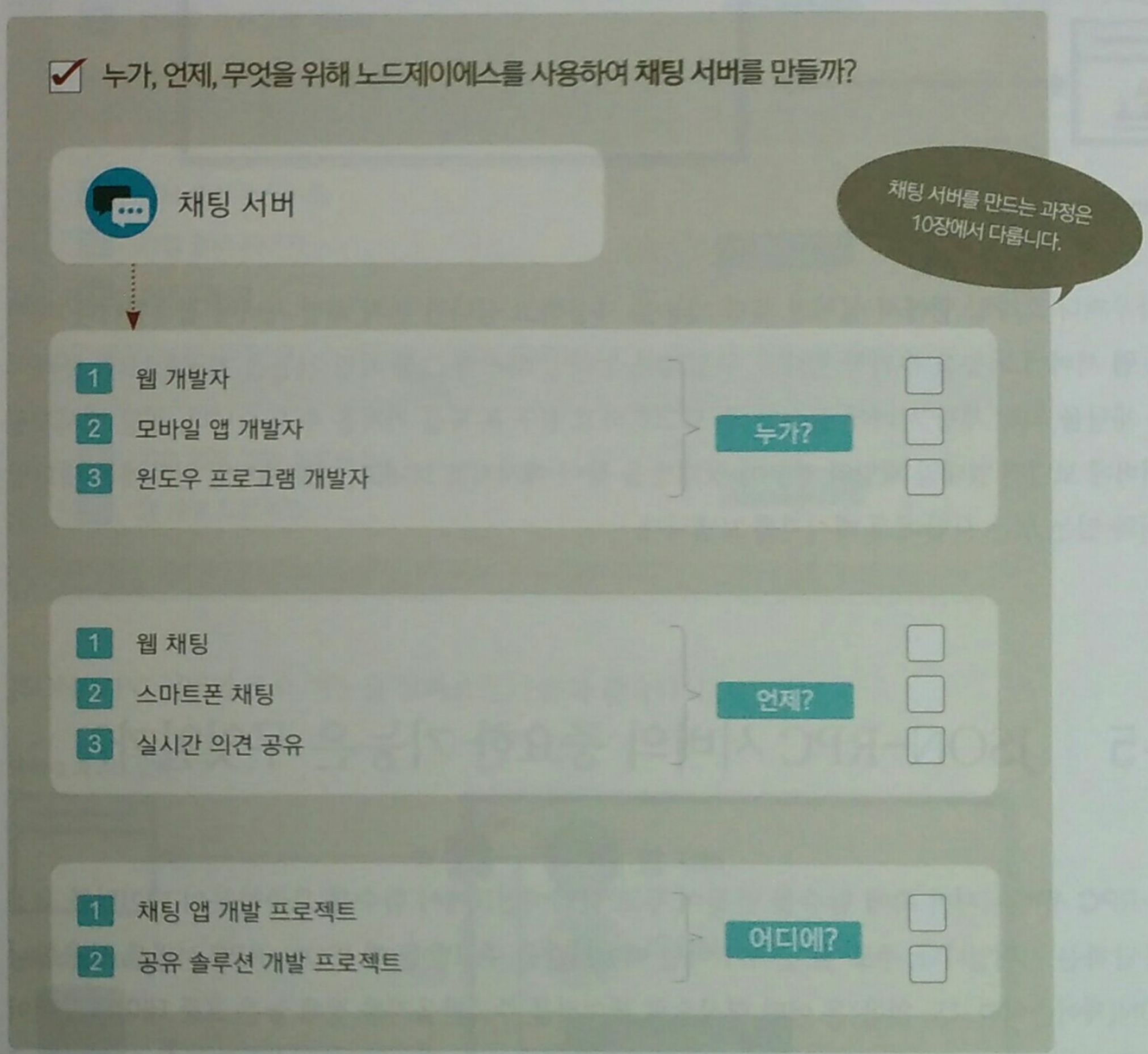
웹 브라우저에서 http://localhost:3000/login.html과 같은 사이트 주소를 입력하고 웹 서버에 요청하면 웹 서버는 미리 만들어 둔 웹 문서를 보내줍니다. localhost(로컬 호스트)는 PC 자신을 나타내는 인터넷 주소라고 할 수 있습니다. 인터넷상에 있는 다른 PC나 단말에 연결하려고 할 때 IP와 같은 단말 주소를 사용하는데, 이때 다른 단말이 아닌 자기 자신을 가리킬 때 localhost라고 합니다. 서버 기능을 만든 후 테스트할 때 다른 단말로 연결하려면 그 단말이 있어야 하므로 이 책에서는 실습 중인 PC에서 서버 기능을 테스트할 수 있도록 localhost라는 주소를 자주 사용합니다.

위 그림은 사이트 주소를 입력하면 웹 서버에서 문서를 보내 주는 방식을 나타내고 있습니다. 이는 웹 브라우저에서 특정 웹 사이트를 조회할 때 사용하는 방식입니다. 웹 서버를 만들 때는 이처럼 일반적인 기능을 먼저 구현해야 하는데, 익스프레스가 웹 서버에 필요한 파일들을 자동으로 만들어 줌으로써 여러분이 좀 더 쉽게 웹 서버를 제작할 수 있도록 도와줍니다. 데이터를 저장하고 조회할 때는 '몽고디비'를 사용하고, 클라이언트에 응답을 보낼 때는 '뷰 템플릿'을 미리 만들어 두었다가 사용합니다. 그리고 웹 브라우저에서 로그인이나 회원가입 기능을 쉽게 만들 수 있도록 도와주는 것이 패스포트입니다.

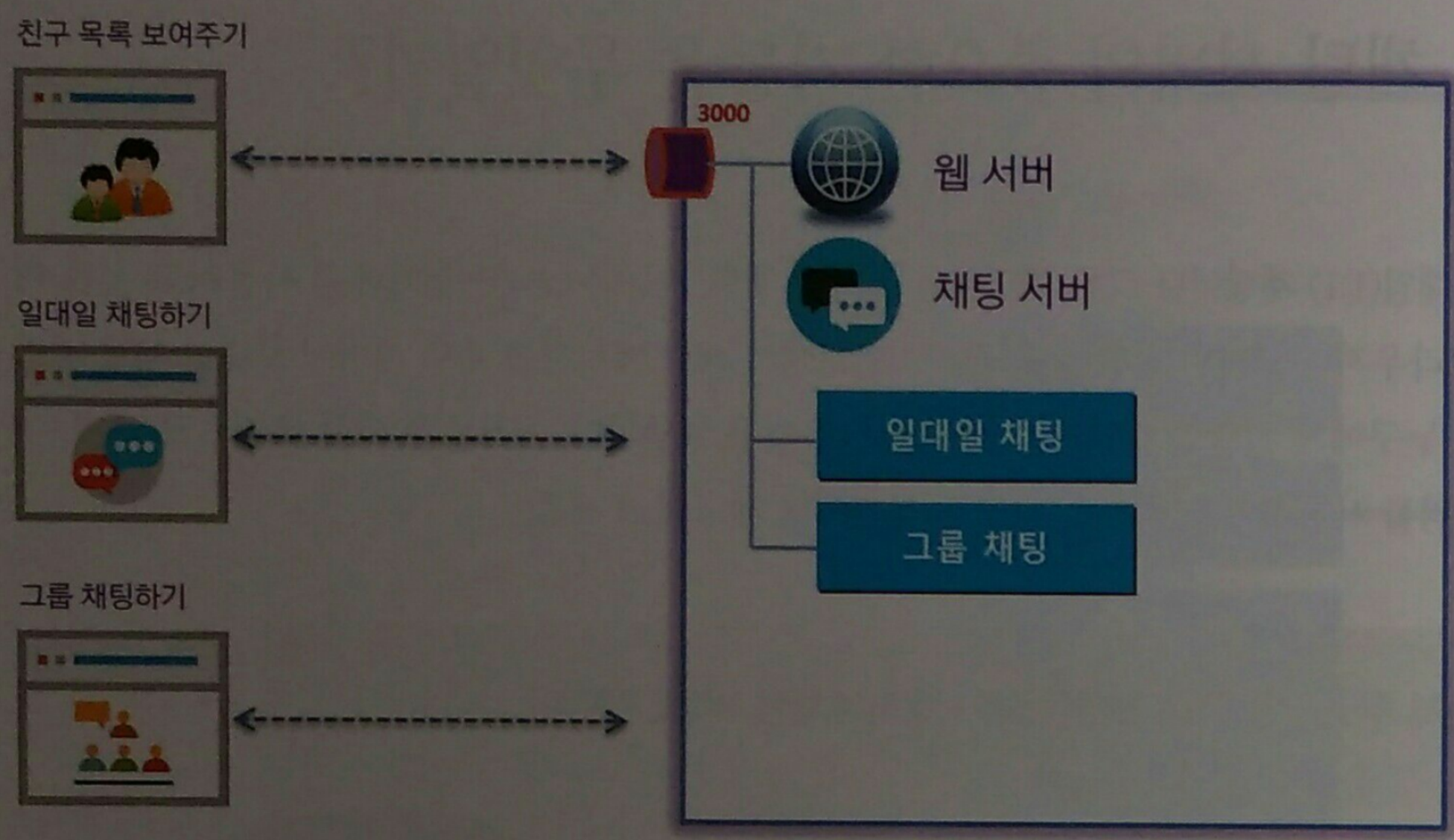
주의! 파일을 업로드하거나 다운로드하는 기능, 세션을 만들거나 쿠키를 저장하는 기능들도 웹 서버를 만들 때 필요하다는 것을 기억해 두세요.

00-4 | 채팅 서버의 중요한 기능은 무엇일까?

채팅 서버는 일대일(1:1) 채팅이나 그룹 채팅처럼 실시간 채팅 기능을 클라이언트에서 사용하고 싶을 때 만듭니다. 웹 브라우저나 모바일 단말 같은 클라이언트에서 채팅 메시지를 채팅 서버에 보내면 채팅 서버는 이 메시지를 누구에게 전달할 것인지를 결정한 후 응답을 보냅니다. 여러분은 어떤 목적으로 언제, 어디에서 채팅 서버를 사용하려고 하는지 다음 질문지에서 체크해 보세요.



채팅 서버의 주요 기능을 살펴보면 다음과 같습니다.



▲ 채팅 서버의 주요 기능

웹 브라우저나 모바일 앱에서 실시간 채팅 기능을 사용하고 싶다면 먼저 채팅 서버에 접속합니다. 채팅 서버는 웹 서버에 기능을 추가한 형태로 구성되며, 일대일 채팅과 그룹 채팅 기능을 제공합니다. 클라이언트는 채팅을 위해 채팅 서버에 접속한 후 로그인하고 친구 목록을 가져올 수 있습니다. 채팅 메시지를 채팅 서버에 보내면 일대일 채팅의 경우에 상대방을 찾아 메시지를 보내고, 그룹 채팅일 경우에는 대화방에 들어와 있는 모든 사람에게 메시지를 보냅니다.

00-5 | JSON-RPC 서버의 중요한 기능은 무엇일까?

JSON-RPC 서버는 서버 쪽에 함수를 만들어 두고 클라이언트에서 함수를 호출하듯이 데이터를 요청하면 응답하는 서버입니다. 주로 웹 문서가 아닌 데이터만을 주고받을 때 JSON-RPC 서버를 사용합니다. JSON(제이슨이라고도 읽음)은 어떤 형식으로 데이터를 주고받을지를 정해 놓은 **표준 데이터 포맷**이며, RPC(Remote Procedure Call) 방식으로 데이터를 주고받습니다. RPC는 서버 쪽에 함수를 만들어 두고 클라이언트에서 함수를 호출하듯이 데이터를 요청할 수 있도록 만들어 주는 표준입니다. JSON과 RPC 표준을 함께 사용하면 서버 쪽에 구축해야 하는 기능을 함수별로 명확하게 구분하여 만들 수 있습니다. 이 때문에 서버의 유지 관리가 쉬워집니다. 그리고 클라이언트에서 웹 문서가 아닌 데이터만 요청하는 경우에도 JSON-RPC 서버를 종종 사용합니다. 여러분은 언제, 어디에서 JSON-RPC 서버를 사용할 것인지 다음 질문지에서 확인해 보세요.

✔ 누가, 언제, 무엇을 위해 노드제이에스를 사용하여 JSON-RPC 서버를 만들까?

JSON-RPC 서버

JSON-RPC 서버를 만드는 과정은 11장에서 다룹니다.

누가?

1. 앱 개발자 ☐
2. 윈도우 프로그램 개발자 ☐

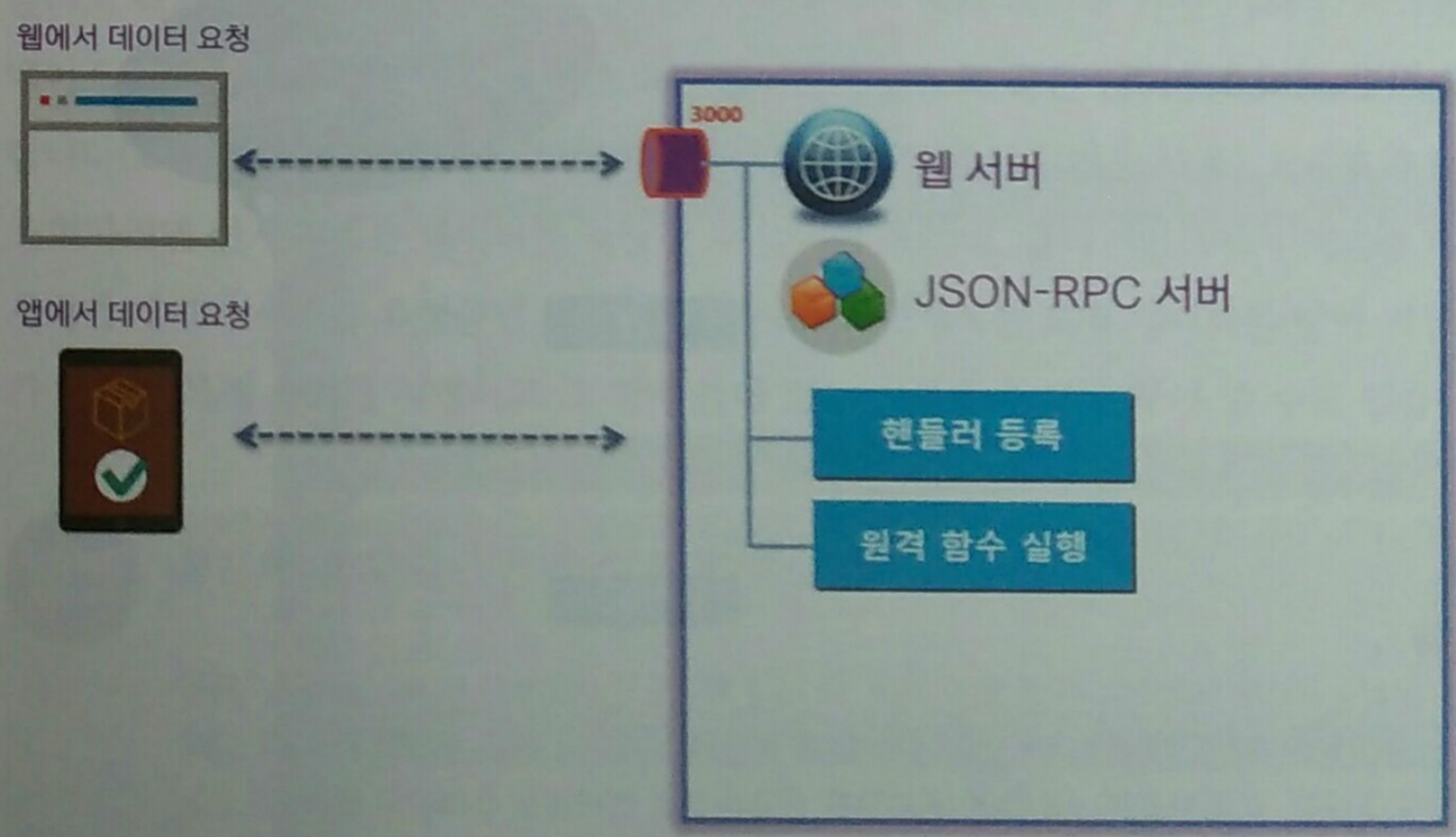
언제?

1. 기능 중심의 사이트 ☐
2. 기업 업무 페이지 ☐
3. 스마트폰 앱 ☐

어디에?

1. 기업 업무 개발 프로젝트 ☐
2. 앱 개발 프로젝트 ☐

JSON-RPC 서버의 주요 기능을 살펴보면 다음과 같습니다.



▲ JSON-RPC 서버의 주요 기능

웹 브라우저나 앱에서는 일반 함수를 호출하듯이 서버에 등록된 함수를 호출할 수 있습니다. 서버에 등록된 함수를 보통 핸들러라고 부르는데, 더하기나 빼기 같은 계산 기능을 하나의 함수로 만들어 등록하면 클라이언트에서 더하기나 빼기 기능을 호출하여 계산 결과를 받아 볼 수 있습니다. 클라이언트가 웹 서버에 웹 문서를 요청하는 것이 아니라 데이터만 요청하고 싶다면 이 JSON-RPC 서버를 사용하는 것이 훨씬 쉽고 간단합니다. 따라서 데이터를 요청하고 응답을 받아 처리하는 데 에이잭스 방식이 아닌 JSON-RPC를 사용하기도 합니다.

00-6 | 위치 기반 서비스 서버의 중요한 기능은 무엇일까?

위치 기반 서비스 서버는 위치 정보를 저장하고 조회할 때 사용합니다. 내가 있는 위치에서 가장 가까운 커피숍을 찾고 싶다면 나의 위치 정보와 커피숍의 위치 정보를 조회하고 비교하는 과정이 필요한데, 위치 기반 서비스 서버가 이런 역할을 수행합니다. 그런데 일반 웹 서버에서 커피숍의 위치 정보를 요청받아 위치 데이터를 저장하거나 조회하면 안 되는 걸까요? 왜 굳이 위치 기반 서비스 서버라는 것을 사용해야 할까요? 먼저 여러분이 언제, 어디에서 위치 기반 서비스 서버를 사용하게 되는지 다음 질문지에서 확인해 보세요.

☑ 누가, 언제, 무엇을 위해 노드제이에스를 사용하여 **위치 기반 서비스 서버**를 만들까?

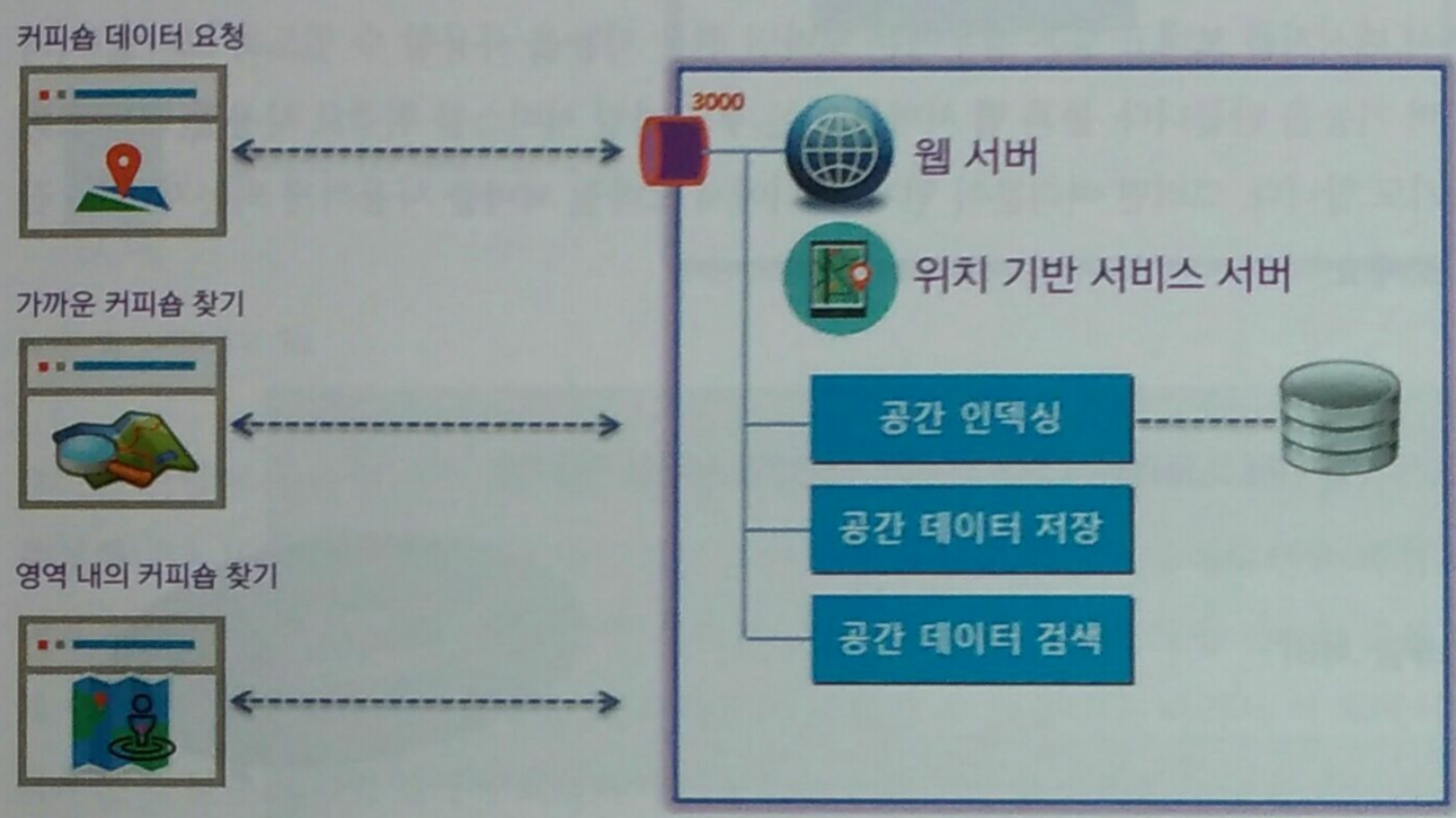
위치 기반 서비스 서버

위치 기반 서비스 서버를 만드는 과정은 12장에서 다룹니다.

1 웹 개발자	누가?	☐
2 앱 개발자		☐
1 모바일 웹	언제?	☐
2 스마트폰 앱		☐
1 매장 찾기 프로젝트	어디에?	☐
2 친구 찾기 프로젝트		☐

위치 정보는 경도와 위도 좌표를 사용하는데, 위도와 경도 두 가지 숫자 값을 데이터베이스에 그대로 저장하면 데이터 양이 늘어나게 됩니다. 또한 두 개의 값을 한꺼번에 비교하여 조회한다면 검색 속도는 현저하게 떨어집니다. 따라서 위치 정보를 효율적으로 저장하고 조회하는 별도의 방법이 필요합니다. 이러한 방법을 제공하는 것이 위치 기반 서비스 서버입니다. 이 책에서는 노드에서 작동하는 몽고디비 데이터베이스를 사용해서 위치 정보를 저장하고 조회하는 방법을 살펴봅니다.

위치 기반 서비스 서버의 주요 기능을 살펴보면 다음과 같습니다.



▲ 위치 기반 서비스 서버의 주요 기능

위치 정보는 **공간 데이터(Spatial Data)**라고도 합니다. 만약 현재 위치에서 가장 가까운 커피숍을 찾아달라고 위치 기반 서비스 서버에 요청하면 서버에서는 현재 위치와 가장 가까운 커피숍의 위치를 데이터베이스에서 검색합니다. 이때 검색 속도가 떨어지지 않도록 **공간 인덱싱(Spatial Indexing)** 방법을 사용합니다(보통 일반 데이터는 데이터베이스에서 **B-Tree 인덱싱** 방법을 사용). 커피숍과 관련 있는 데이터를 조회하려면 먼저 커피숍 데이터를 저장할 수 있어야 하는데, 클라이언트에서 커피숍 데이터를 확인한 후 서버에 데이터 추가를 요청하면 위치 기반 서비스 서버에서는 요청 데이터를 받아 저장합니다. 현재 위치가 아니라 특정 영역을 지정하고 그 영역 안에 있는 커피숍을 모두 찾아 줄 수도 있습니다.

정박사의 한마디

공간 데이터와 공간 인덱싱이 무엇인가요?

공간 데이터(Spatial Data)란 지구상에 있는 집, 도로 등을 점이나 선 등의 형태로 나타낸 위치 데이터를 말하며 위도, 경도로 된 좌표 값으로 표현할 수 있습니다. 그런데 1차원 데이터를 위해 만든 B-Tree 인덱싱 방법으로 2차원 데이터를 저장하면 조회 속도가 현저히 떨어집니다. 따라서 위도, 경도와 같은 공간 데이터, 즉 2차원 데이터의 좌표 값을 데이터베이스에 저장할 때는 공간 인덱싱 방법을 사용합니다. 공간 인덱싱은 데이터베이스에서 2차원 공간 데이터를 빠르게 조회하기 위해 만든 2차원 인덱싱 방법입니다. 2차원 인덱싱 방법에는 R-Tree, R*Tree 등이 있습니다.

00-7 | 모바일 서버의 중요한 기능은 무엇일까?

모바일 서버는 모바일 단말을 사용할 때 필요한 기능을 제공하는 서버입니다. 보통 안드로이드폰이나 아이폰에서 동작하는 앱에서 서버 기능이 필요할 때는 웹 서버를 만들어 데이터를 주고받는 경우가 많습니다. 이는 웹 서버의 기능만으로도 앱에서 서버에 데이터를 보내 저장하거나 서버의 데이터를 조회하는 기능을 충분히 만들 수 있기 때문입니다. 그러나 사용자가 어떤 모바일 단말을 사용하는지 알고 싶거나 모바일 단말로 푸시 메시지를 보내고 싶은 경우에는 모바일 전용 기능을 사용할 수 있도록 모바일 서버라는 이름으로 서버 기능을 만듭니다. 물론 웹 서버를 만든 후 모바일 서비스를 위주로 사용할 때도 **모바일 서버**라고 부르기도 합니다. 그러면 여러분이 언제, 어디에서 모바일 서버를 사용하게 되는지 다음 질문지에서 확인해 보세요.

☑ 누가, 언제, 무엇을 위해 노드제이에스를 사용하여 **모바일 서버**를 만들까?

모바일 서버

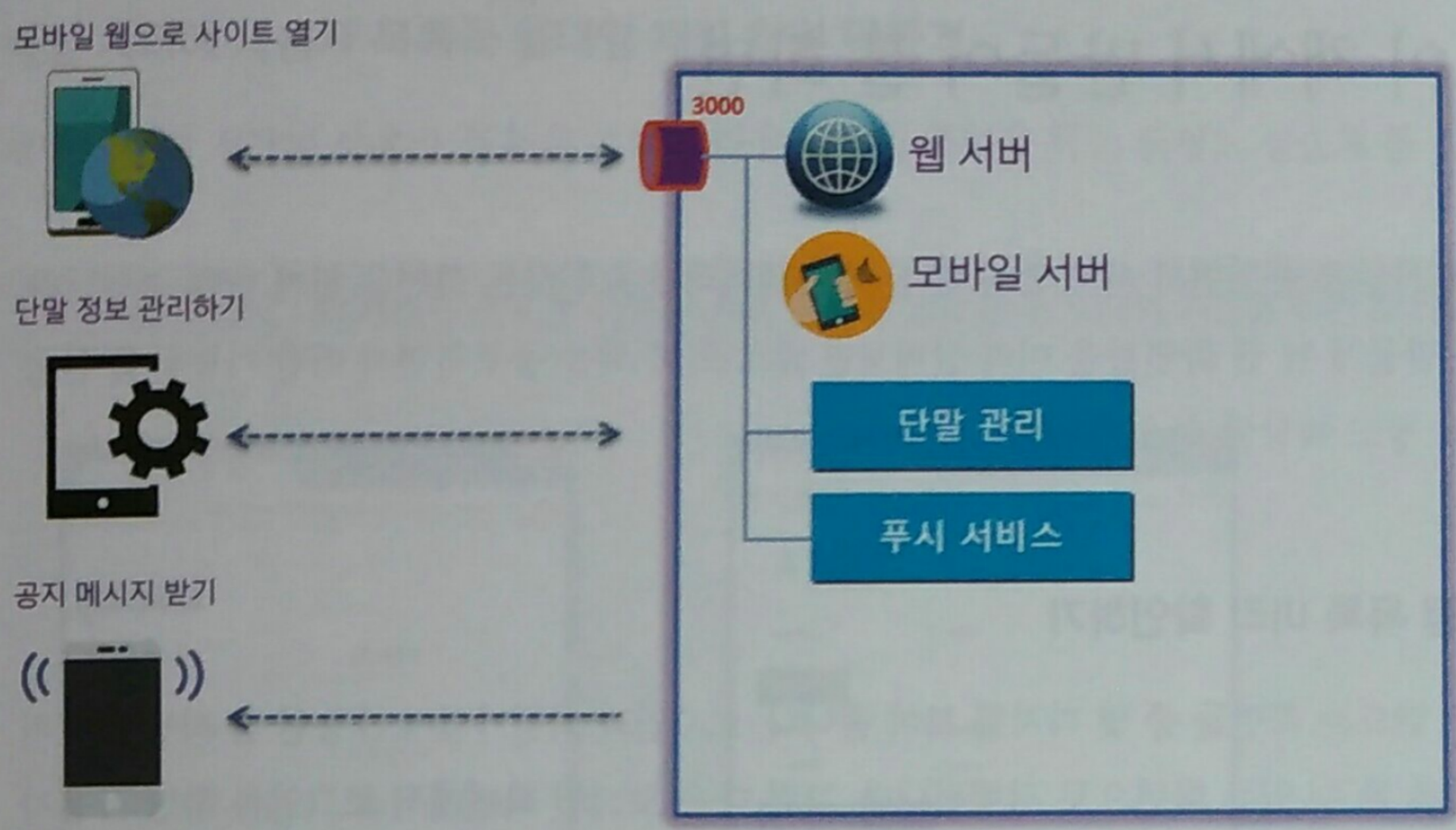
모바일 서버를 만드는 과정은 13장에서 다룹니다.

1 앱 개발자 } 누가? ☐

1 스마트폰 앱 } 언제? ☐

1 앱 개발 프로젝트
2 단말 관리 프로젝트 } 어디에? ☐ ☐

모바일 서버의 주요 기능을 살펴보면 다음과 같습니다.



▲ 모바일 서버의 주요 기능

모바일 단말에서 웹 브라우저를 열고 웹 문서를 요청할 경우에는 앞에서 설명한 웹 서버의 기능만으로 충분합니다. 또한 모바일 앱에서 채팅 기능을 만들어 사용할 때에도 앞에서 만든 채팅 서버의 기능을 그대로 사용할 수 있습니다. 다시 말하면, 모바일 단말에서는 웹 서버나 채팅 서버와 같은 서버 기능을 웹으로 만든 클라이언트에서 사용하는 것과 똑같이 사용할 수 있습니다. 따라서 이 책에서 모바일 서버라고 다루는 부분은 모바일 앱에서 요청하여 사용하는 서버 기능을 의미하는 것은 아니며, 주로 모바일 단말 전용으로 사용하는 부분만을 말합니다. 모바일 서버를 만들 때 이 점에 주의하기 바랍니다.

이 책에서 다루는 모바일 서버의 내용은, 사용자가 사용하는 단말을 서버에 등록한 후 관리자가 단말 정보를 알 수 있는 방법, 그리고 모바일 단말 쪽으로 푸시 메시지를 보내 공지사항 등을 알려주는 방법입니다.

앱이 아닌 모바일 웹을 만들어 사용하는 경우에는 PC용 웹 브라우저에서 작동하는 웹 서버를 그대로 사용할 수 있습니다. 따라서 모바일 웹이나 하이브리드 앱에서 작동하는 서버 기능을 만들고 싶다면 이 책의 모바일 서버를 설명하는 부분 외에 웹 서버, 채팅 서버 등을 만드는 과정도 차근차근 따라가며 공부해야 합니다.

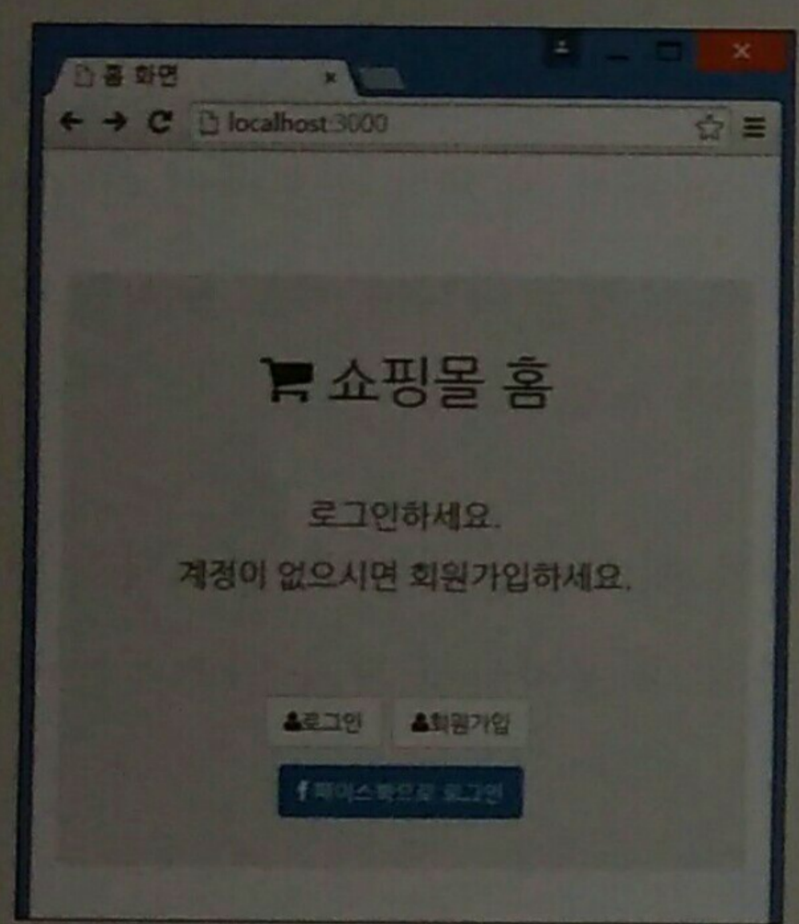
00-8 | 이 책에서 만들어 볼 화면

이 책에서는 서버 기능을 만들면서 서버 쪽에 웹 문서나 데이터를 요청하는 화면을 함께 만들 것입니다. 따라서 여러분이 만들게 될 웹 화면들을 미리 살펴보면 앞으로 이 책을 공부하면서 어떤 기능들을 만들게 될 것인지 어느 정도 예상할 수 있습니다.

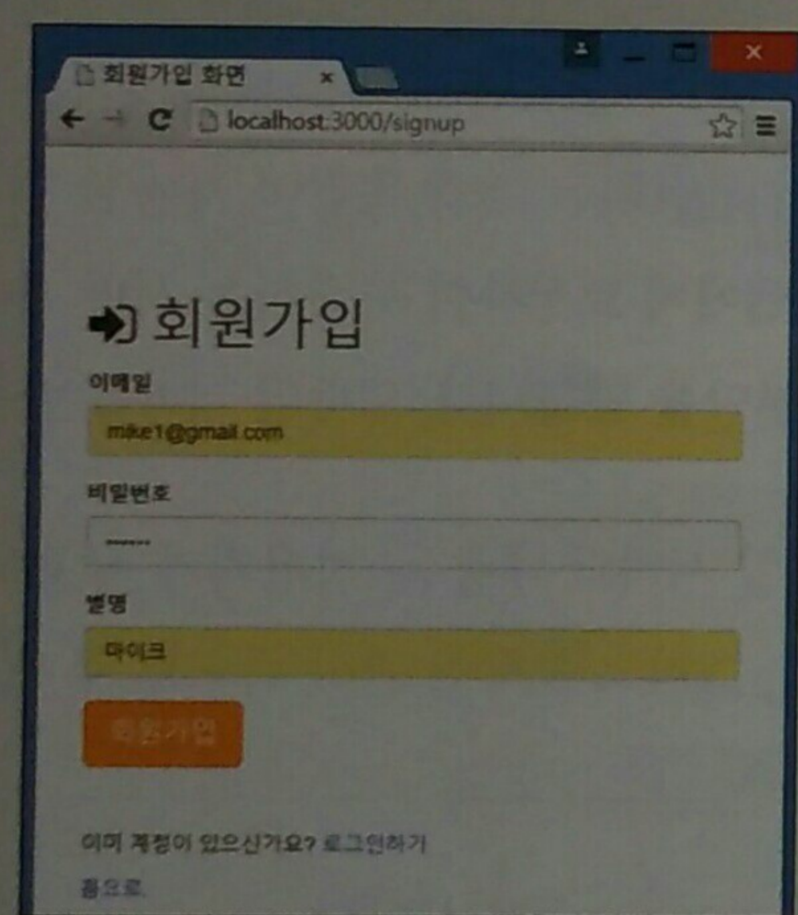
설치할 프로그램 목록 미리 확인하기

다음은 이 책에서 만드는 화면들 중 몇 가지를 보여 줍니다. 로그인과 회원가입이 가능한 홈 화면을 먼저 만들고, 회원가입을 할 수 있는 화면으로 진행됩니다. 그런 다음 로그인 화면에서 로그인을 하면 사용자의 프로필 화면으로 이동하는 것까지 이어집니다.

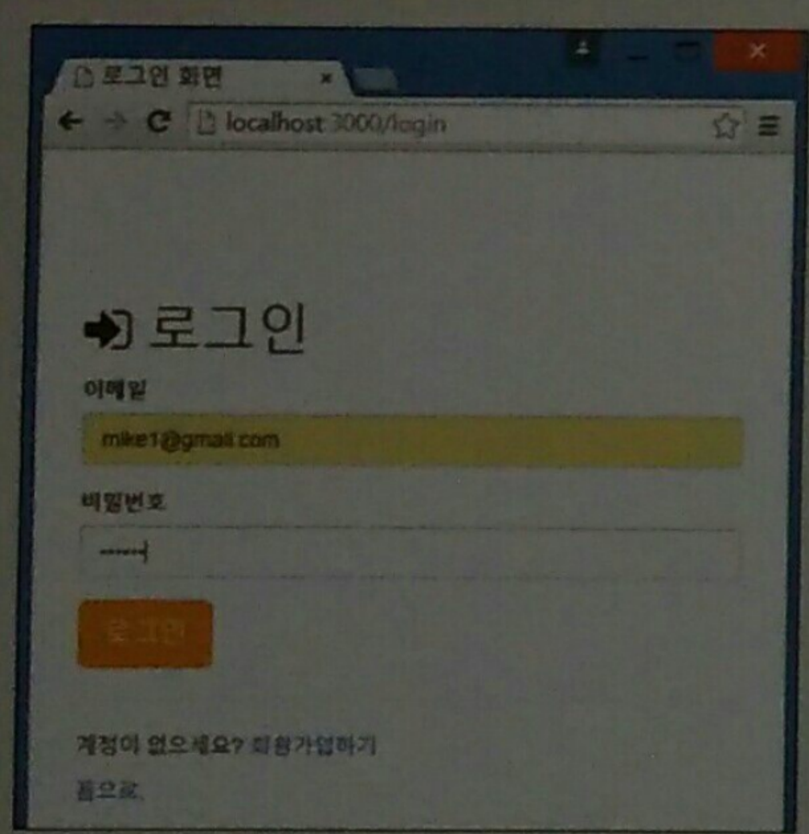
로그인과 회원가입이 가능한 홈 화면 → 회원가입 화면 → 로그인 화면 → 사용자 프로필 화면



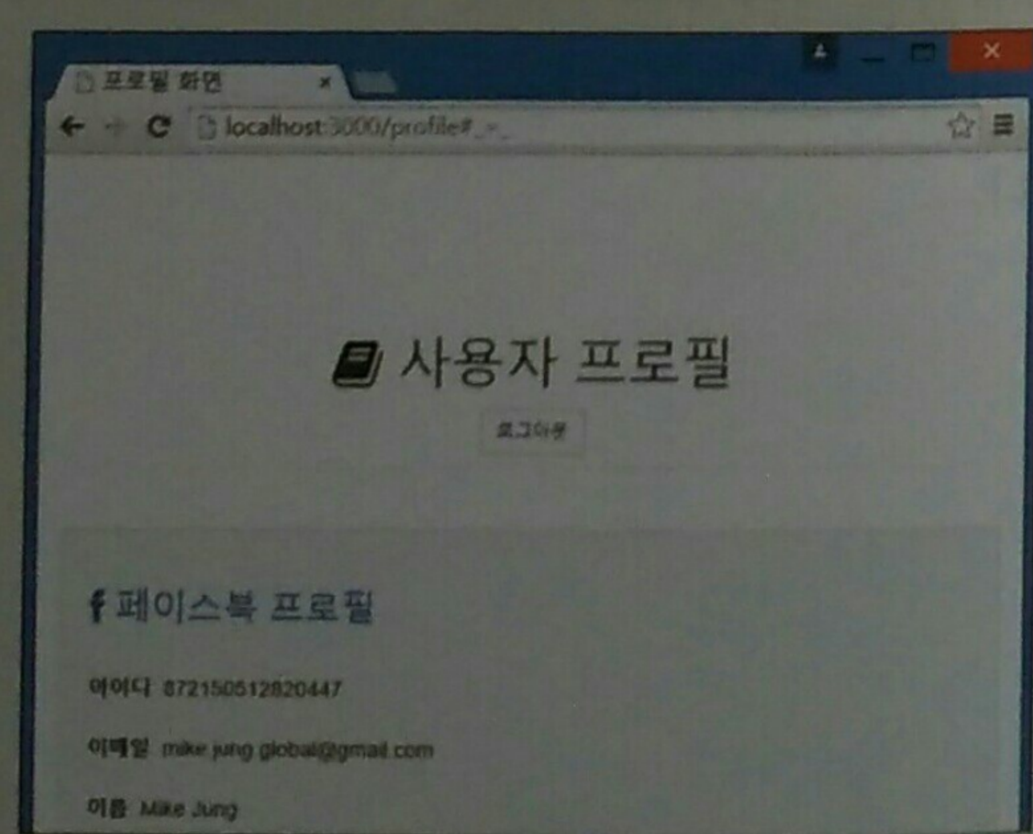
▲ 로그인과 회원가입이 가능한 홈 화면



▲ 회원가입 화면



▲ 로그인 화면

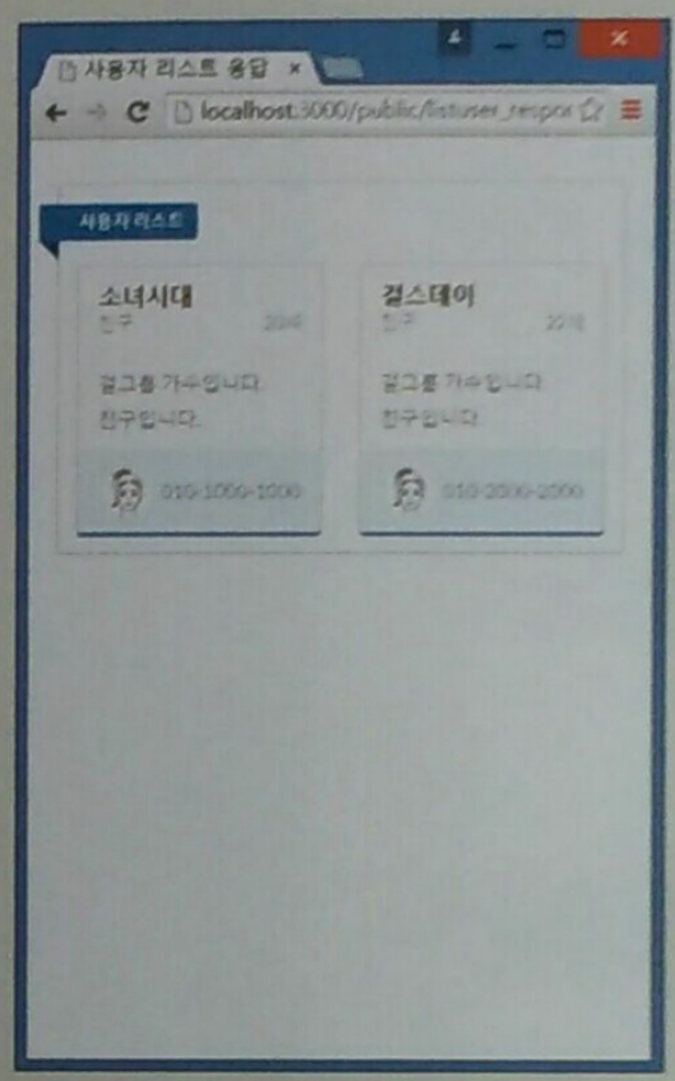


▲ '페이스북으로 로그인' 기능 실행 후 표시된 사용자 프로필 화면

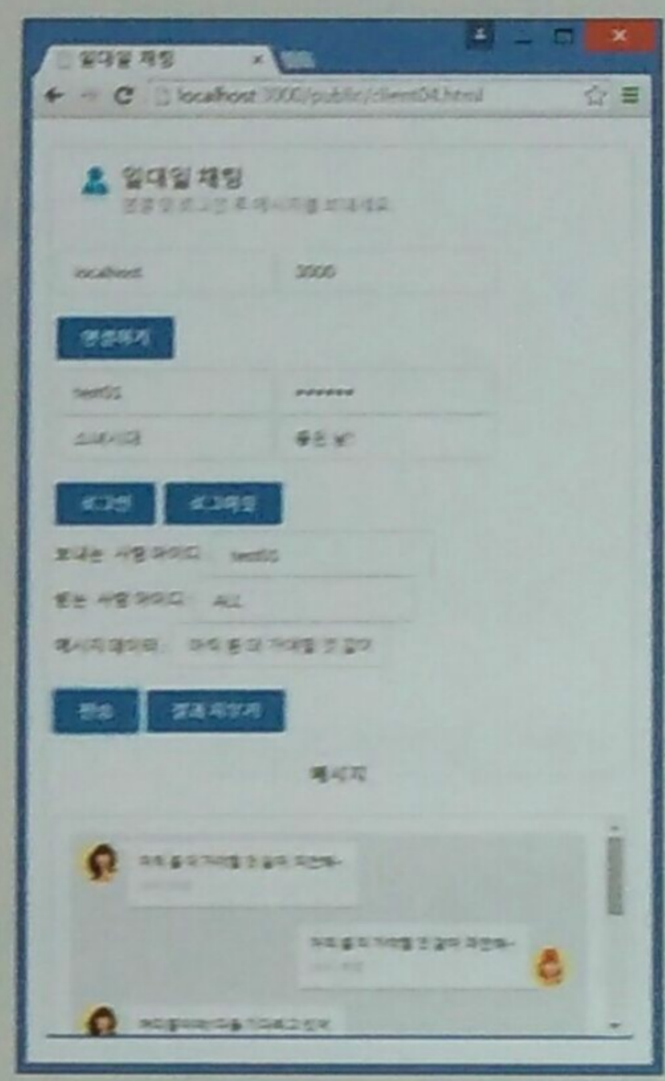
채팅 서버에 사용자 목록과 일대일 채팅 화면 만들기

채팅 서버에 저장된 사용자 목록을 조회한 다음 일대일 채팅을 하는 단계도 실습해 볼 것입니다.

서버에 저장된 사용자들을 조회한 화면 → 채팅 서버에 접속한 후 일대일 채팅을 하는 화면



▲ 채팅 서버에 저장된 사용자 조회 화면

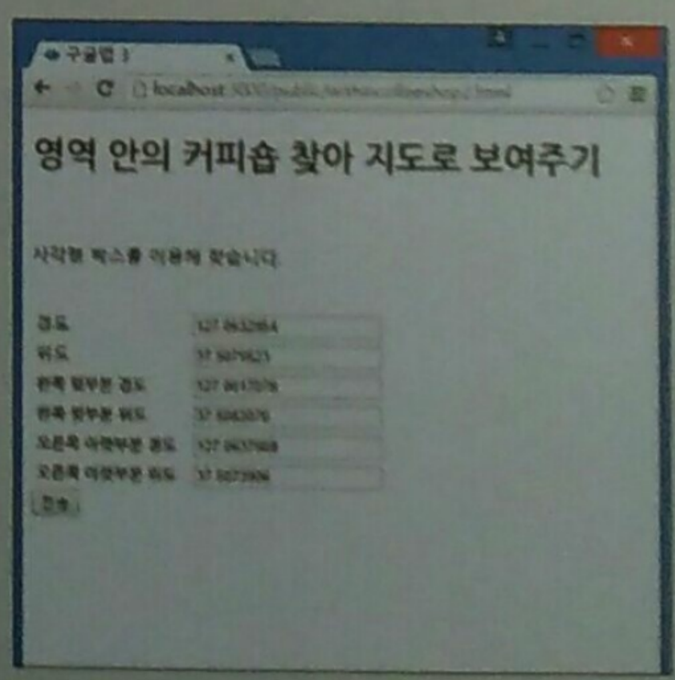


▲ 일대일 채팅 화면

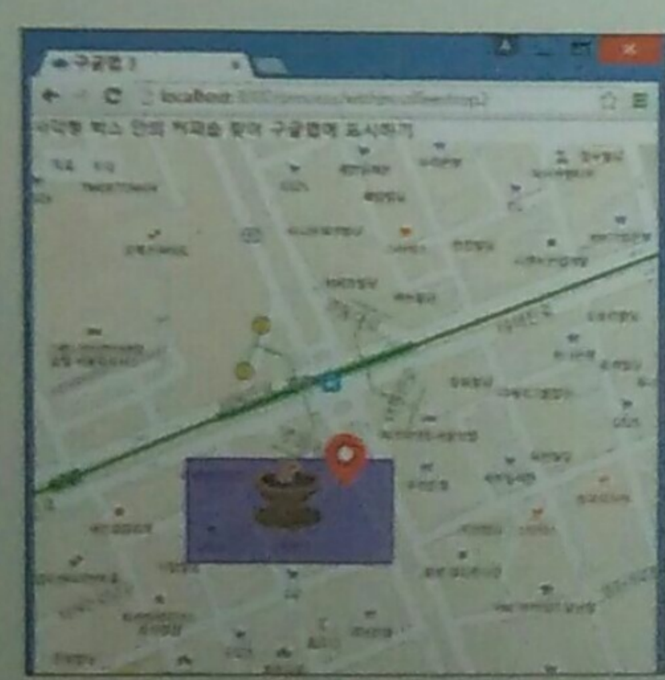
위치 기반 서비스 서버에 커피숍 검색 화면 만들기

위치 기반 서비스 서버를 만들어서 활용하는 방법도 알아봅니다. 자신이 위치한 주변에 사각형 영역을 지정하고 그 영역 안에 있는 커피숍을 찾아서 지도 위에 보여 주는 작업입니다.

영역 안의 커피숍 찾기를 요청하는 화면 → 찾은 커피숍의 데이터를 보여 주는 화면



▲ 커피숍 찾기 요청 화면



▲ 찾은 커피숍의 데이터를 보여 준 화면

이 외에도 모바일 서버와 JSON-RPC 서버를 구축하면서 여러 가지 기능을 확인할 수 있는 화면을 만들어 봅니다. 이 화면들이 앞으로 만들 화면의 전부는 아니지만 어떤 화면들을 만들지 대충은 예상할 수 있을 것입니다. 여러분이 원하는 화면과 원하는 기능이 있을 것 같나요? 이 책의 각 장을 따라 하면서 여러분에게 필요한 서버를 좀 더 재미있게 만들 수 있기를 기대합니다.

자, 이제 첫 장을 넘겨 보세요.

01 노드에 대해 알아보고 개발 도구 설치하기

노드(Node.js)가 무엇이기에 웹 서비스를 만들던 개발자들이 하나둘 노드에 관심을 가지게 된 것일까요? 노드는 어떤 방식으로 또 무엇을 위해 만들어졌을까요? 01장에서는 노드가 무엇인지 먼저 살펴봅니다. 그다음에는 노드를 사용해서 서버를 만들 수 있도록 개발 도구를 설치하겠습니다. 개발 도구를 설치한 다음에는 노드에 대해 전혀 모르더라도 누구나 쉽게 따라할 수 있는 가장 간단한 기능을 만들어 볼 것입니다. 이 과정을 통해 노드가 무엇인지 대략적이나마 알 수 있게 될 것입니다.

도해로 정리하기

노드는 왜 만들었을까요?

노드의 대표적 특징을 알아볼까요?

비동기 입출력

이벤트 기반 입출력

노드의 모듈

01-1 | 노드란 무엇일까?

노드(Node.js)는 자바스크립트를 이용해서 서버를 만들 수 있는 개발 도구입니다.

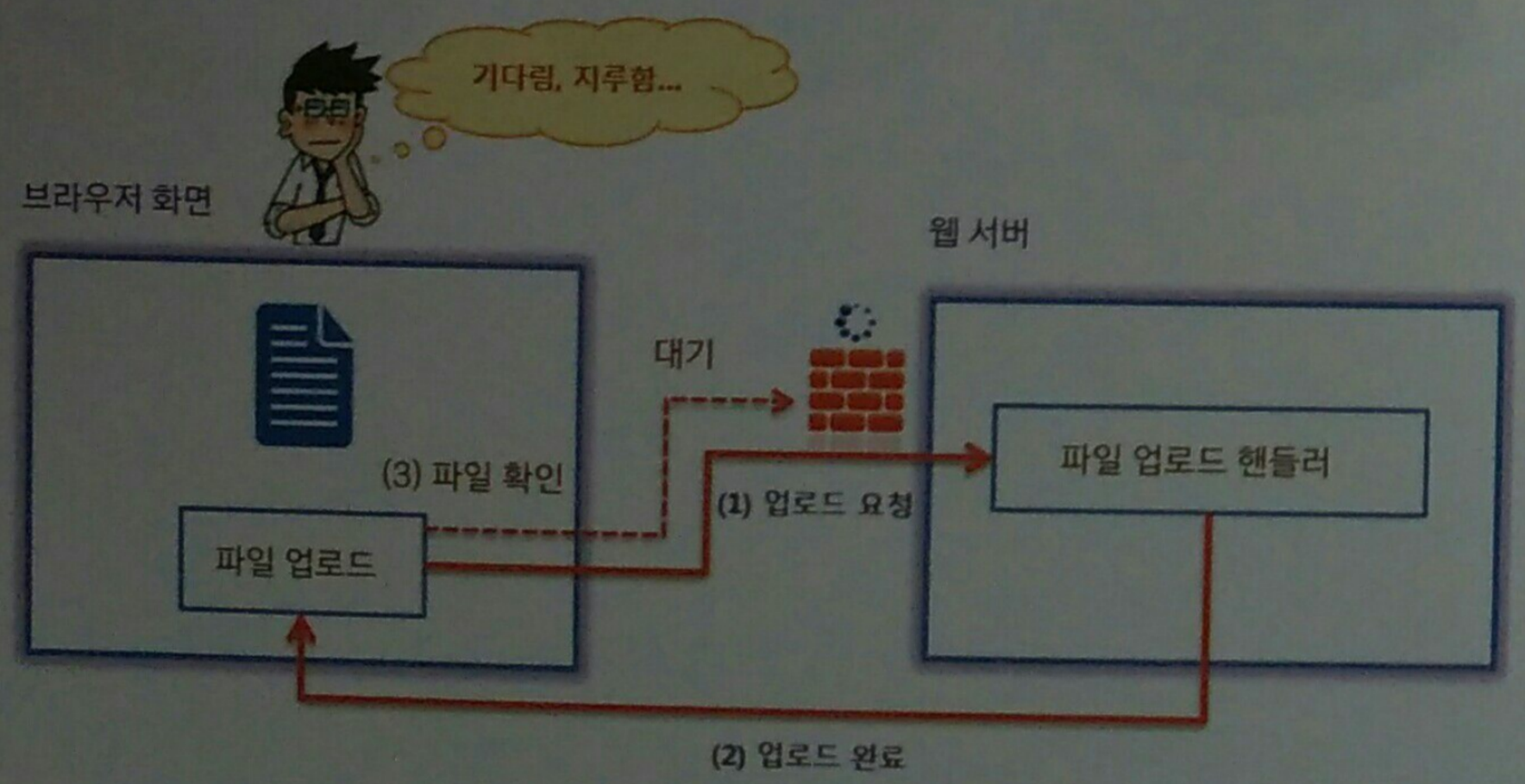
정박사의 한마디 노드제이에스라고 불러도 좋고 노드라고 불러도 좋습니다

노드제이에스(Node.js)는 노드(Node)라는 원래의 이름 뒤에 자바스크립트를 사용한다는 의미에서 제이에스(.js)를 붙인 형태입니다. 따라서 노드제이에스를 간단하게 노드라고도 부릅니다.

그런데 이미 서버를 만드는 개발 도구는 수없이 많습니다. 이 때문에 여러분은 왜 하필 노드를 사용하는지 궁금할 것입니다.

> "웹 서비스와 같은 서버 기능을 만들 수 있는 언어나 도구들은 이미 많은데, 노드(Node.js)라는 새로운 개발 도구는 왜 또 만든 것일까?"

노드라는 개발 도구를 새로 만들게 된 이유는 의외로 아주 단순합니다. 노드는 2009년에 라이언 달(Ryan Dahl)이 만들었는데 그 당시에는 웹 서버에 파일을 업로드할 때, 업로드가 완료되기 전까지 웹 서버에서 데이터를 조회한다거나 하는 등의 다른 작업을 할 수 없었습니다. 이 문제를 해결하기 위해 새로운 방식의 서버 개발 도구를 만들기 시작했는데 그것이 노드입니다.



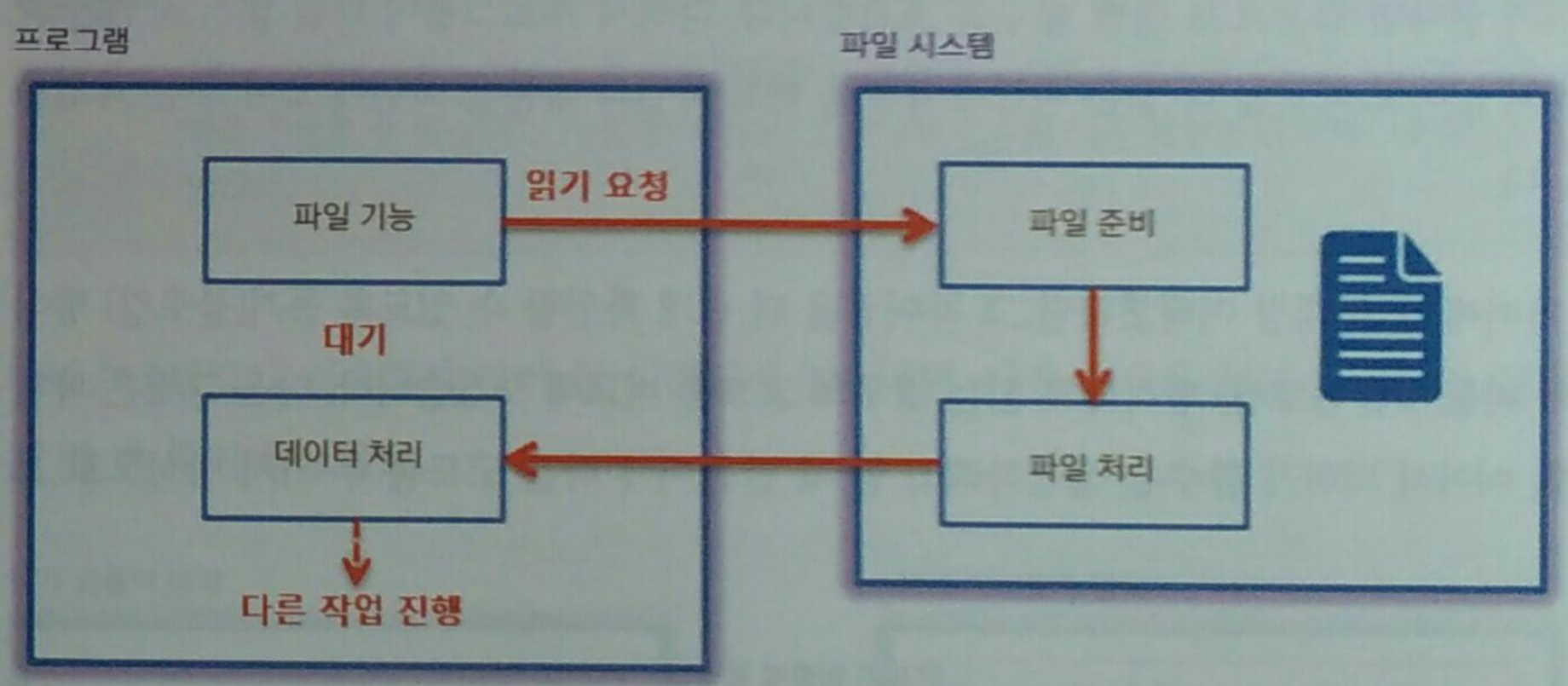
▲ 웹 서버에 파일을 업로드하면서 다른 작업을 동시에 처리하기 힘든 문제가 있습니다.

웹 브라우저를 통해 내 PC에 있는 문서 파일 하나를 업로드하고 싶다면 먼저 웹 서버에 업로드를 요청해야 합니다. 이때 웹 서버에는 파일 업로드 기능을 담당하는 **핸들러**(Handler)를 하나 만들어 둡니다. 그런데 파일의 크기가 크다면 파일을 업로드하는 데 1분 또는 그 이상의 시간이 걸릴 수 있습니다. 그런데 파일 업로드를 완료하기 전에는 서버에 있는 다른 파일의 정보를 확인하거나 파일 업로드가 어떻게 진행되고 있는지 요청하는 것이 불가능해서 업로드가 완료될 때까지 대기해야 합니다. 지금은 이런 문제를 해결할 수 있는 여러 가지 방법이 나와 있어 큰 문제가 되지 않습니다. 하지만 그 당시에는 웹 서버의 파일 업로드 핸들러가 하나의 요청이 끝날 때까지 다른 요청을 대기시켜 응답 시간이 길어지거나, 서버에서 처리해야 하는 요청의 수가 증가하면 CPU나 메모리 사용량도 크게 증가하는 문제가 많이 발생했습니다.

노드의 비동기 입출력 방식

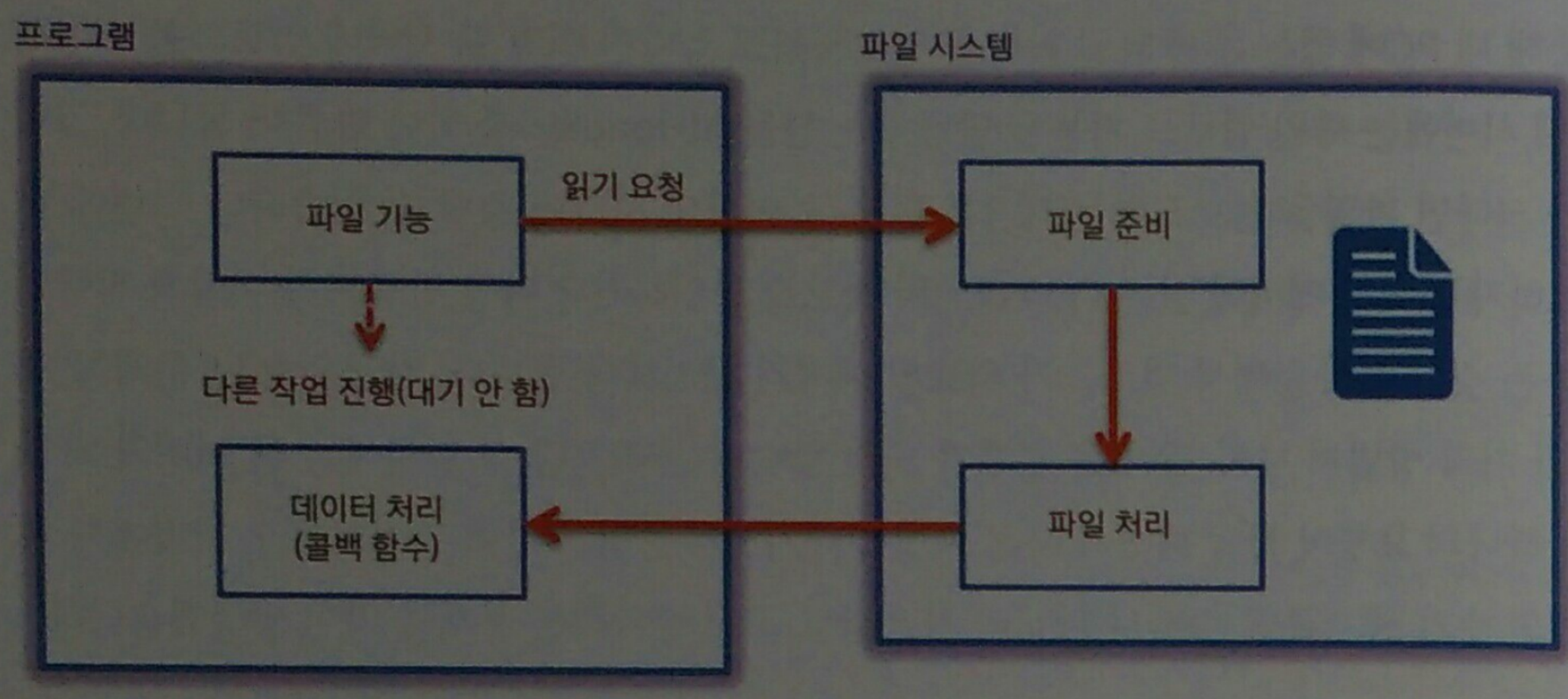
이런 문제를 해결하기 위해 만든 게 노드입니다. 즉, 하나의 요청 처리가 끝날 때까지 기다리지 않고 다른 요청을 동시에 처리할 수 있는 **비동기 입출력(논블로킹 입출력, Non-Blocking IO)** 방식을 적용했습니다.

비동기 입출력 방식이 노드의 대표적인 특징입니다. 비동기 입출력 방식을 이해하려면 반대되는 의미의 **동기 입출력(블로킹, Blocking IO)** 방식으로 파일을 읽는 과정을 먼저 알아야 합니다.



▲ '동기 입출력' 방식으로 파일을 읽는 경우

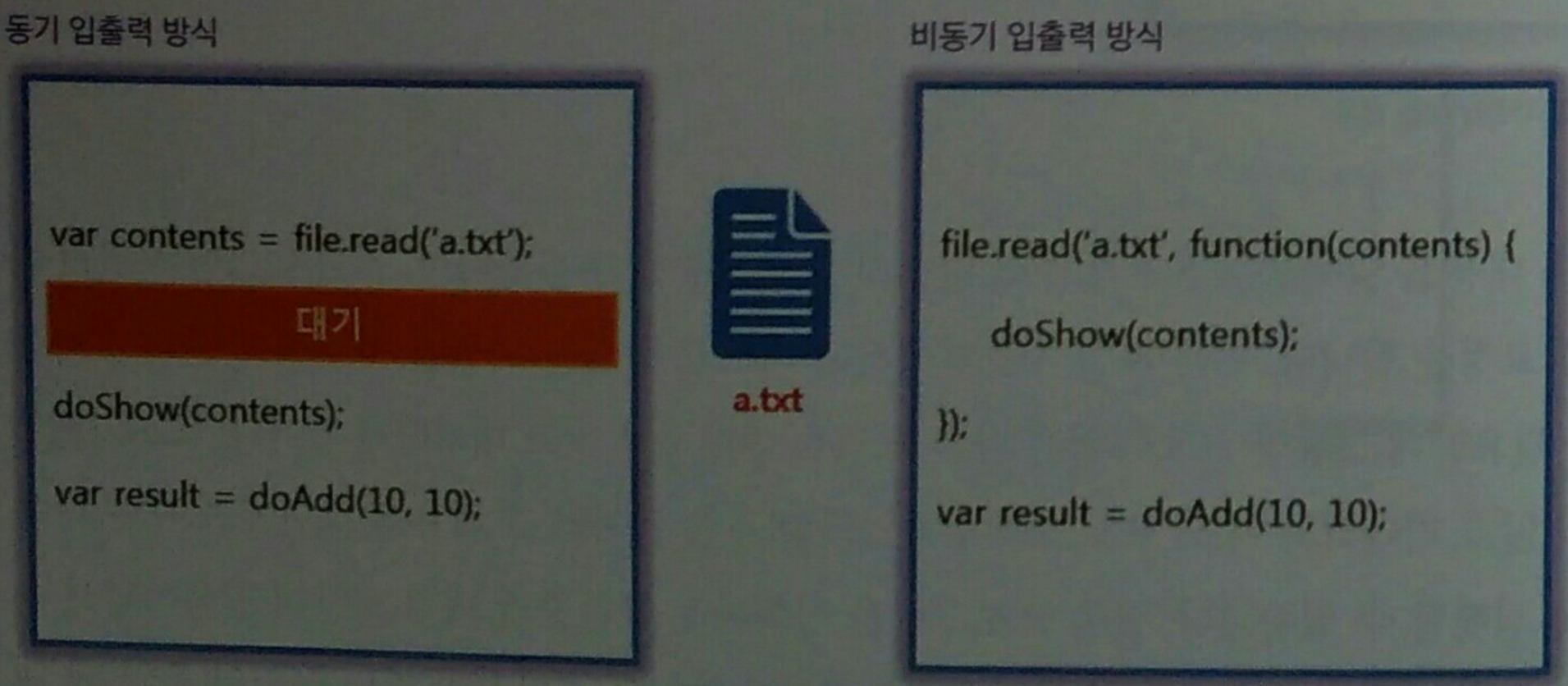
PC에 있는 파일의 내용을 읽어오는 동기 입출력 방식의 프로그램을 만들었다면 그 프로그램은 먼저 파일 시스템에 읽기 요청을 합니다. 이때 파일 시스템에서는 디스크에 있는 파일을 확인하고 준비한 다음 처리하는 시간이 필요한데 그동안 프로그램은 다른 작업을 진행하지 않고 대기하게 됩니다. 프로그램이 아무 동작을 하지 않고 기다리는 거죠. 프로그램이 파일의 내용을 읽어와 그 내용을 화면에 보여 준 다음에야 다른 작업을 진행할 수 있습니다. 결국 프로그램을 실행하는 중간중간 대기 시간이 발생하면서 속도가 느려지는 문제가 생기는 것입니다. 그런데 이것을 비동기 방식으로 바꾸면 파일을 읽어 들일 때 대기하는 시간이 없어집니다.



▲ '비동기 입출력' 방식으로 파일을 읽는 경우

파일을 '비동기 방식으로 읽는다'는 것은 파일 시스템에 읽기 요청을 한 후에 프로그램이 대기하지 않고 다른 작업을 진행한다는 것을 의미합니다. 그러면 어떻게 프로그램에서 다른 작업을 진행하는 중간에 파일 시스템에서 파일 처리를 할 수 있는 걸까요? 프로그램에서는 파일을 위한 작업을 언제 다시 시작할 수 있을까요? 프로그램에서 해당 파일의 내용을 처리할 수 있는 시점이 되면 **콜백 함수**(Callback Function)가 호출됩니다. 프로그램에서는 파일 읽기 요청을 하기 전에 콜백 함수를 등록하는데, 파일 시스템은 파일 처리가 끝나면 자동으로 콜백 함수를 호출합니다. 따라서 프로그램이 파일 읽기 작업이 끝날 때까지 대기하지 않아도 파일을 다 읽은 시점에 통보를 받고 파일의 내용을 화면에 보여 주는 작업을 진행할 수 있습니다.

두 가지 방식의 차이에 대해 대강 이해했다면 그 차이점을 더 쉽게 확인할 수 있도록 동기(블로킹) 방식으로 읽는 경우와 비동기(논블로킹) 방식으로 읽는 경우의 코드를 비교해 보겠습니다. 다음 그림은 파일 읽기를 요청한 후, 이어서 더하기 함수를 실행하려고 할 때 만들어야 하는 코드를 두 가지 방식으로 보여 줍니다.



▲ 동기(Blocking IO) 방식과 비동기(Non-Blocking IO) 방식의 코드 패턴 비교

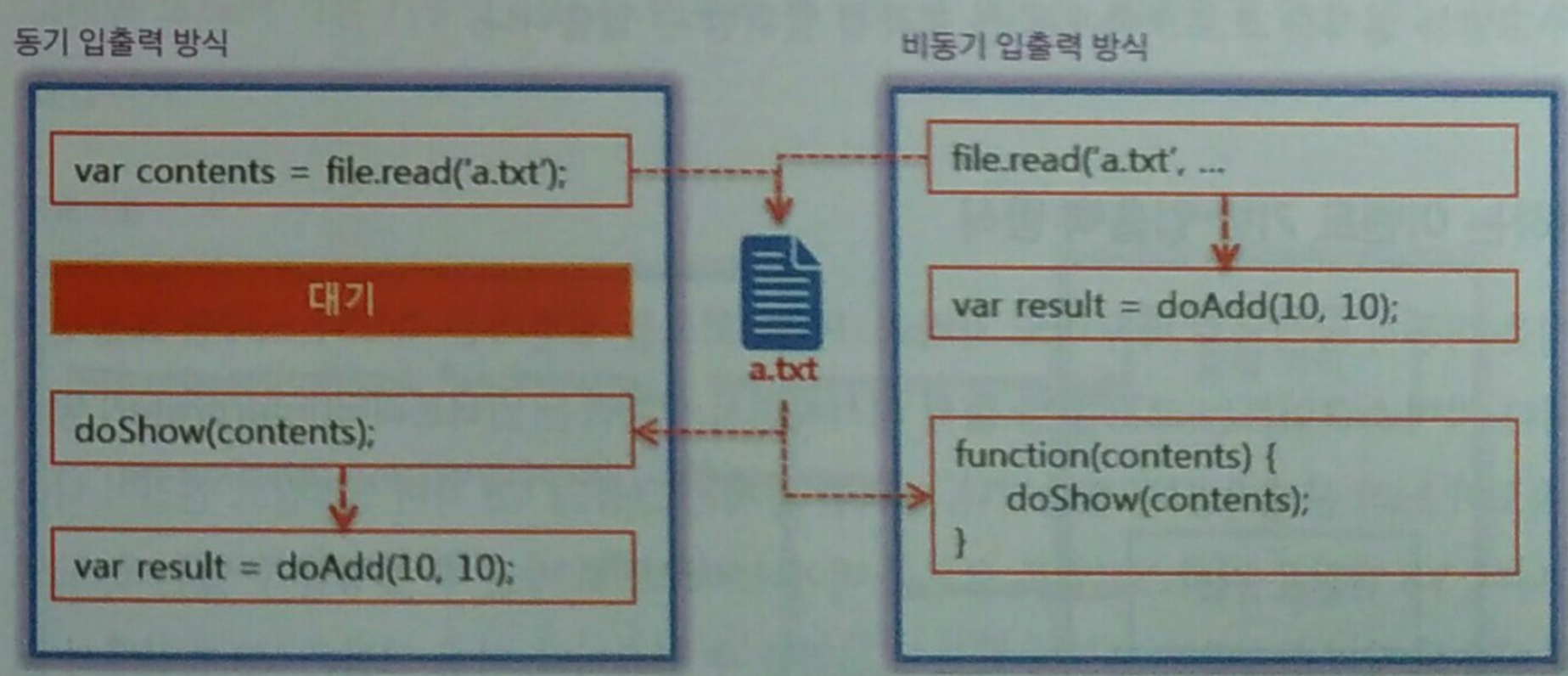
자바스크립트 코드가 익숙하지 않겠지만, 앞에서 확인한 코드가 어떤 차이를 갖는지는 대충 이해할 수 있을 것입니다. 먼저 왼쪽의 **동기(블로킹) 방식**으로 표현한 코드를 살펴보겠습니다. file 객체가 파일을 나타내고 그 객체에 정의된 함수 중에서 파일을 읽어 들이는 함수를 read( )라고 하면, file.read( ) 코드를 실행해서 파일의 내용을 읽어 올 수 있습니다. 이때 파일의 내용을 다 읽어 올 때까지 프로그램은 대기하고, 읽는 작업이 끝나야 doShow( ) 함수를 실행하여 파일에서 읽어 온 결과를 화면에 보여 줍니다. 파일의 내용을 화면에 보여 준 후에 doAdd( ) 함수를 실행하여 다른 작업을 진행합니다. 이 doAdd( ) 함수는 두 개의 숫자를 더하는 더하기 함수를 미리 만들어 둔 것이며, 이 프로그램의 가장 마지막에 실행됩니다.

이에 비해 오른쪽의 비동기(논블로킹) 방식으로 표현한 코드는 형태가 약간 다릅니다. 똑같은 기능을 실행하지만 file.read( ) 함수를 실행할 때 파일 이름뿐만 아니라 파일을 다 읽었을 때 실행할 함수도 파라미터로 전달합니다. 파일 읽기를 요청한 다음에는 파일을 다 읽어 화면에 보여 주기 전이라도 doAdd( ) 함수를 호출합니다. 이 때문에 파일을 읽어 들이는 작업 시간이 오래 걸리더라도 대기하지 않고 doAdd( ) 함수를 먼저 실행할 수 있습니다. 파일을 읽는 작업이 완료되었다면 file.read( ) 함수를 실행할 때 전달했던 함수가 자동으로 호출됩니다. 이 함수가 바로 콜백 함수(Callback Function)입니다.

정박사의 한마디 콜백 함수의 정의를 꼭 알아두세요

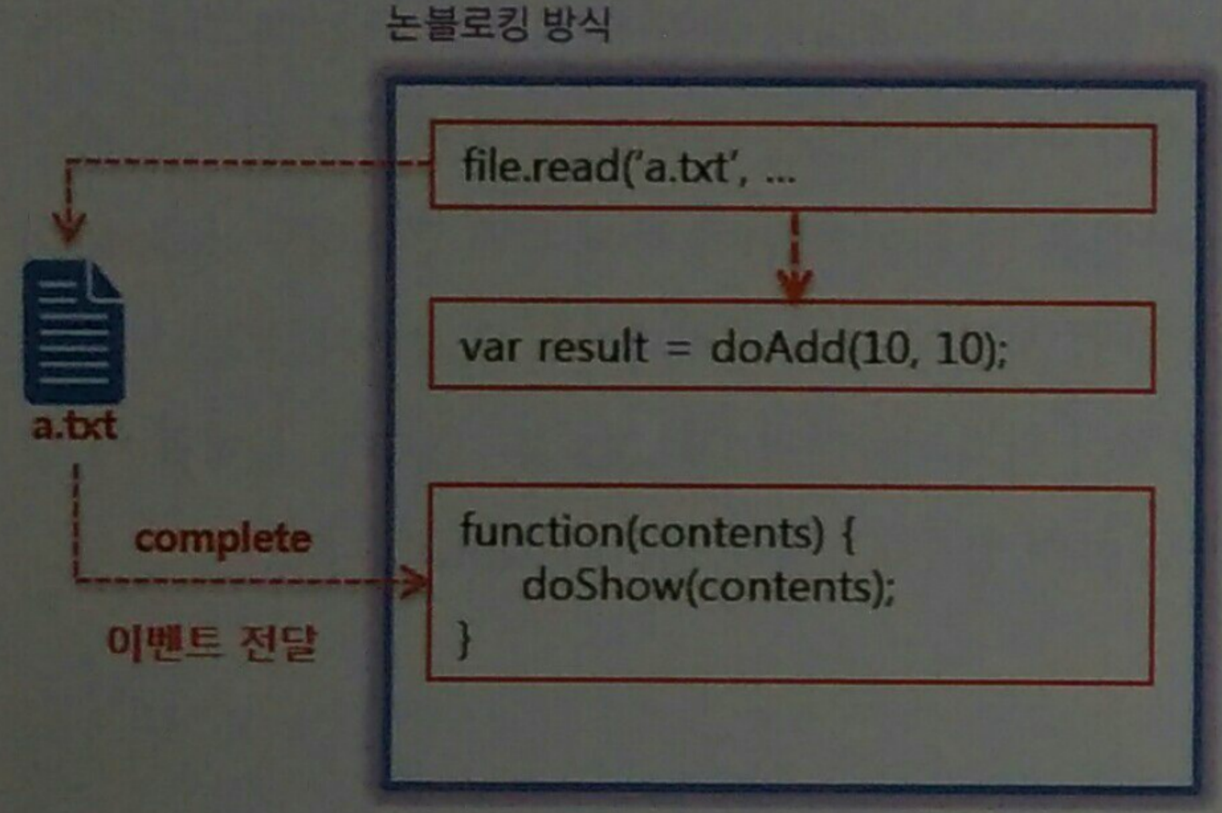
자바스크립트에서는 변수에 함수를 할당할 수 있습니다. 따라서 변수에 할당된 함수를 다른 함수의 파라미터로 전달할 수 있습니다. 이렇게 파라미터로 전달된 함수를 다른 함수의 내부에서 호출하는 것이 콜백 함수입니다.

자바스크립트가 익숙하지 않아 잘 이해되지 않는다면, 다음 그림을 보면서 코드가 실행되는 순서를 다시 한 번 확인해 보세요.



▲ 동기(블로킹) 입출력 방식과 비동기(논블로킹) 입출력 방식의 코드 실행 순서 비교

각각의 코드를 빨간색 사각형으로 분리하여 표시해 보았습니다. 왼쪽의 경우에는 첫 번째 코드 줄이 실행된 후 대기하는 시간이 있고, 파일 시스템에서 파일 읽기가 완료되었다고 알려 주면 대기 시간이 끝납니다. 대기 시간이 길어지더라도 doAdd( ) 함수는 실행되지 않고 계속해서 기다립니다. 오른쪽의 경우에는 파일 읽기를 요청한 후 바로 이어서 doAdd( ) 함수를 호출합니다. 파일 읽기가 끝나는 시점에 doShow( ) 함수가 자동으로 호출됩니다. 파일 읽기가 완료되었을 때 파일 시스템에서 콜백 함수를 호출하는데, 파일 시스템이 이벤트와 함께 호출하는 방식이면 **이벤트 기반 입출력**(Event driven IO) 모델이라고 부릅니다.



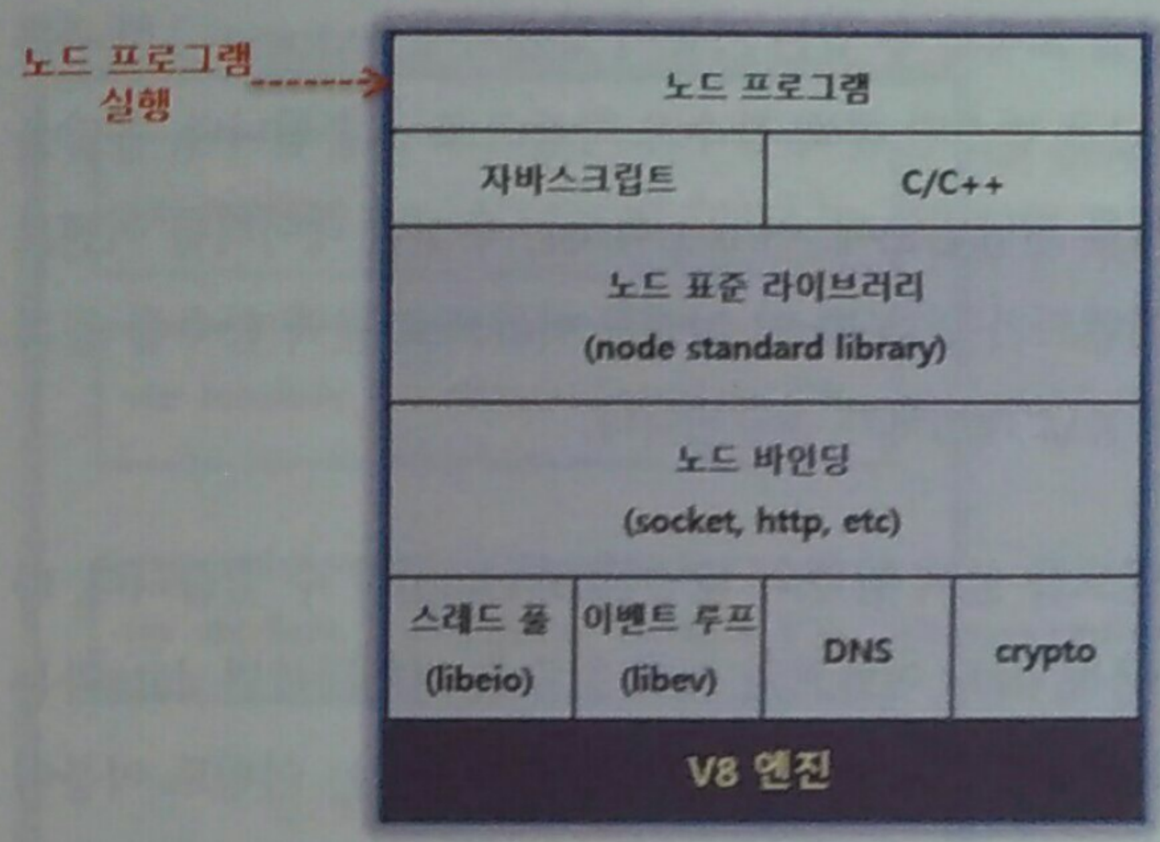
▲ 비동기(논블로킹) 입출력 방식에서의 이벤트 전달

파일 시스템에서 파일 읽기가 완료되었다는 이벤트만 전달하면 프로그램에서는 그 이벤트를 받아 콜백 함수를 실행할 수 있으며, 콜백 함수는 이벤트가 발생하기 전에 미리 등록합니다.

지금까지 살펴본 내용을 종합해 보면, 노드는 비동기 방식을 사용하는 이벤트 기반 입출력 모델(Event driven, Non-blocking I/O model)입니다. 노드에서 사용하는 이 모델은 상당히 가볍고 효율적이며 많은 데이터를 주고받는 실시간 프로그램에서 큰 효과를 발휘할 수 있습니다.

노드에서 구현하는 이벤트 기반 입출력 방식

데이터 처리 방식을 비동기 방식으로 바꾸어도 자바스크립트 코드를 실행하는 속도가 느리면 효율성이 떨어질 수 있습니다. 자바스크립트는 코드를 한 줄씩 해석하면서 실행하는 **인터프리터**(Interpreter) 방식을 사용하므로 속도가 느려 문제가 되곤 했습니다. 이러한 문제는 크롬의 V8 자바스크립트 엔진이 나오면서 해결되었습니다. V8 엔진은 자바스크립트 코드를 네이티브 코드로 바꾼 후 실행할 수 있는데, 노드는 이 V8 엔진을 이용해 자바스크립트 코드를 빠르게 실행할 수 있습니다. 다음 그림은 V8 엔진에서 노드가 동작할 수 있도록 만든 **시스템 구조**(아키텍처, Architecture)를 보여 줍니다.

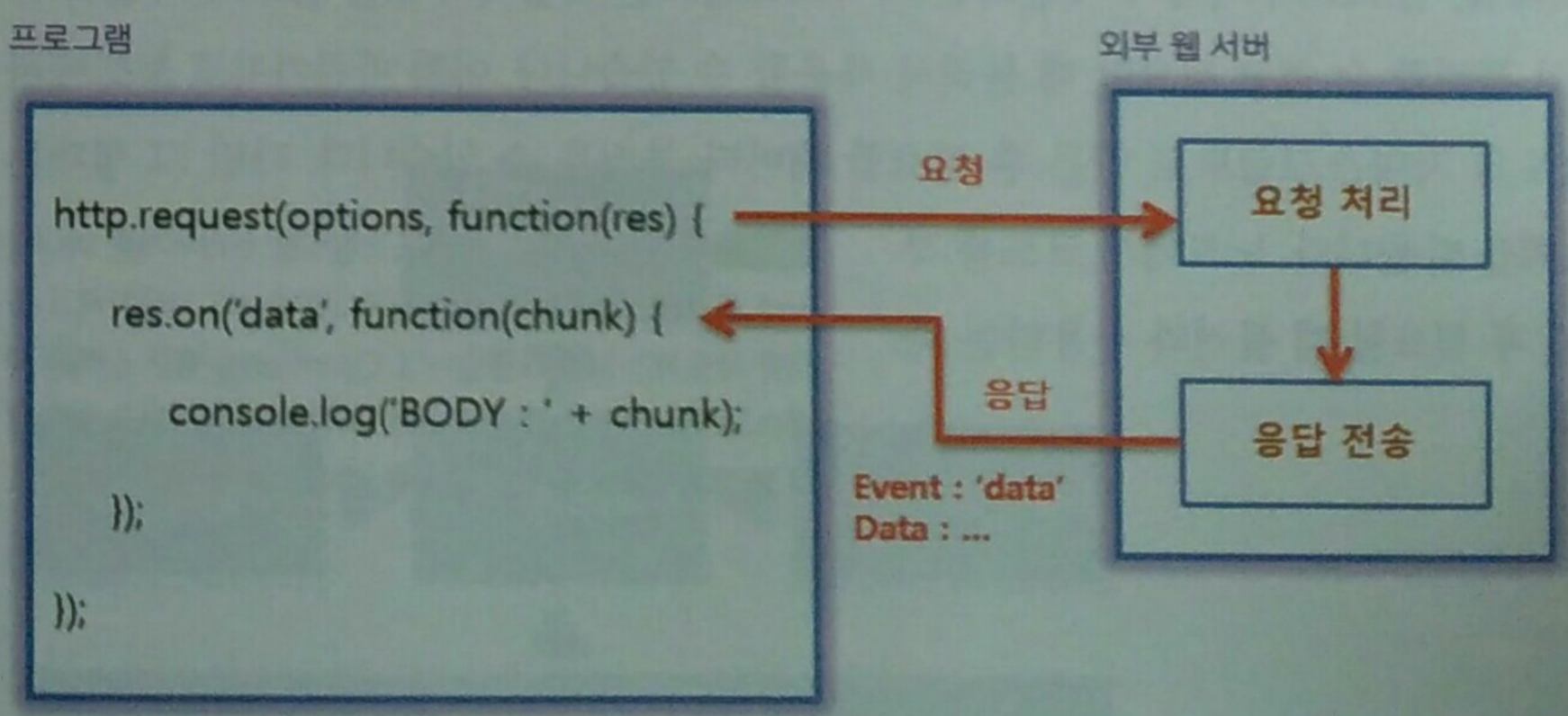


▲ V8 엔진에서 동작하는 노드의 아키텍처

노드를 설치한 다음 노드로 프로그램을 만들어 실행하면 크롬 V8 엔진 위에서 실행됩니다. V8 엔진에는 필요한 기능을 병렬로 실행하는 '스레드 풀'과 이벤트를 받아 처리하는 '이벤트 루프' 등의 기본 기능이 있으며, 그 위에 네트워킹 기능을 담당하는 **소켓(Socket), http 라이브러리**들이 있습니다. 그리고 다시 그 위에 표준 라이브러리가 구현되어 있습니다. 이런 라이브러리를 사용하고 싶다면 자바스크립트로 프로그램을 만들면 됩니다. 개발자들이 각자의 목적에 맞게 만든 노드 프로그램은 그 라이브러리 위에서 동작하게 됩니다.

노드 프로그램이 동작할 수 있도록 만든 이 아키텍처에서 가장 중요한 부분 중 하나가 이벤트를 받아 처리하는 기능입니다. 특히 노드는 서버 쪽에서 동작하는 프로그램을 만드는 것이 주 임무이기 때문에 기본 기능인 소켓이나 HTTP 프로토콜을 사용해 데이터를 송수신하는 기능을 포함하고 있습니다. 따라서 데이터 송수신 부분에도 이벤트 처리 방식을 그대로 사용할 수 있습니다.

그러면 노드의 기본 기능인 http를 이용해 웹 서버로 요청을 보내고 응답을 받는 코드의 일부를 살펴보겠습니다.



▲ http를 이용해 웹 서버로 요청을 보내고 응답을 받는 방식

http 객체는 HTTP 프로토콜로 웹 서버에 데이터를 요청할 수 있는 기능이 있습니다. request( ) 함수를 호출하여 웹 서버에 데이터를 요청할 수 있고 응답을 받으면 콜백 함수가 자동으로 호출됩니다. 응답을 처리할 수 있는 콜백 함수는 res라는 이름의 객체를 전달받는데, 이 res 객체는 수신한 데이터를 이벤트 이름으로 구별할 수 있습니다. 예를 들어, data 이벤트가 있을 때 이 이벤트 이름으로 콜백 함수를 등록하면, 등록한 콜백 함수에 data 이벤트로 전달된 응답 데이터가 전달됩니다.

자바스크립트에서는 on( ) 메소드를 사용해 이벤트를 콜백 함수와 **바인딩(Binding)**할 수 있습니다. 따라서 응답 객체인 res 객체의 on( ) 메소드를 사용해 data 이벤트와 콜백 함수를 바인딩하면 data라는 이름의 이벤트를 받았을 때 등록한 콜백 함수가 실행됩니다. 앞에서 살펴본 코드에서는 이벤트 이름이 data이며 응답으로 받은 데이터는 **chunk 변수**에 들어 있습니다.

정박사의 한마디 이벤트를 바인딩한다는 것이 어떤 의미일까요?

바인딩(Binding)이란 서로 묶어서 연결해 준다는 의미입니다. 예를 들어, 버튼이 하나 있는데 이 버튼을 클릭했을 때 click 이벤트가 발생한다고 가정합니다. 이때 click 이벤트를 함수 객체와 바인딩하면 이 click 이벤트가 발생했을 때 등록한 함수 객체가 실행됩니다. 즉, 사용자가 버튼을 클릭하면 이 함수가 실행됩니다. 그 형식은 다음과 같습니다.

```
[객체].on([이벤트 이름], [함수 객체]);
```

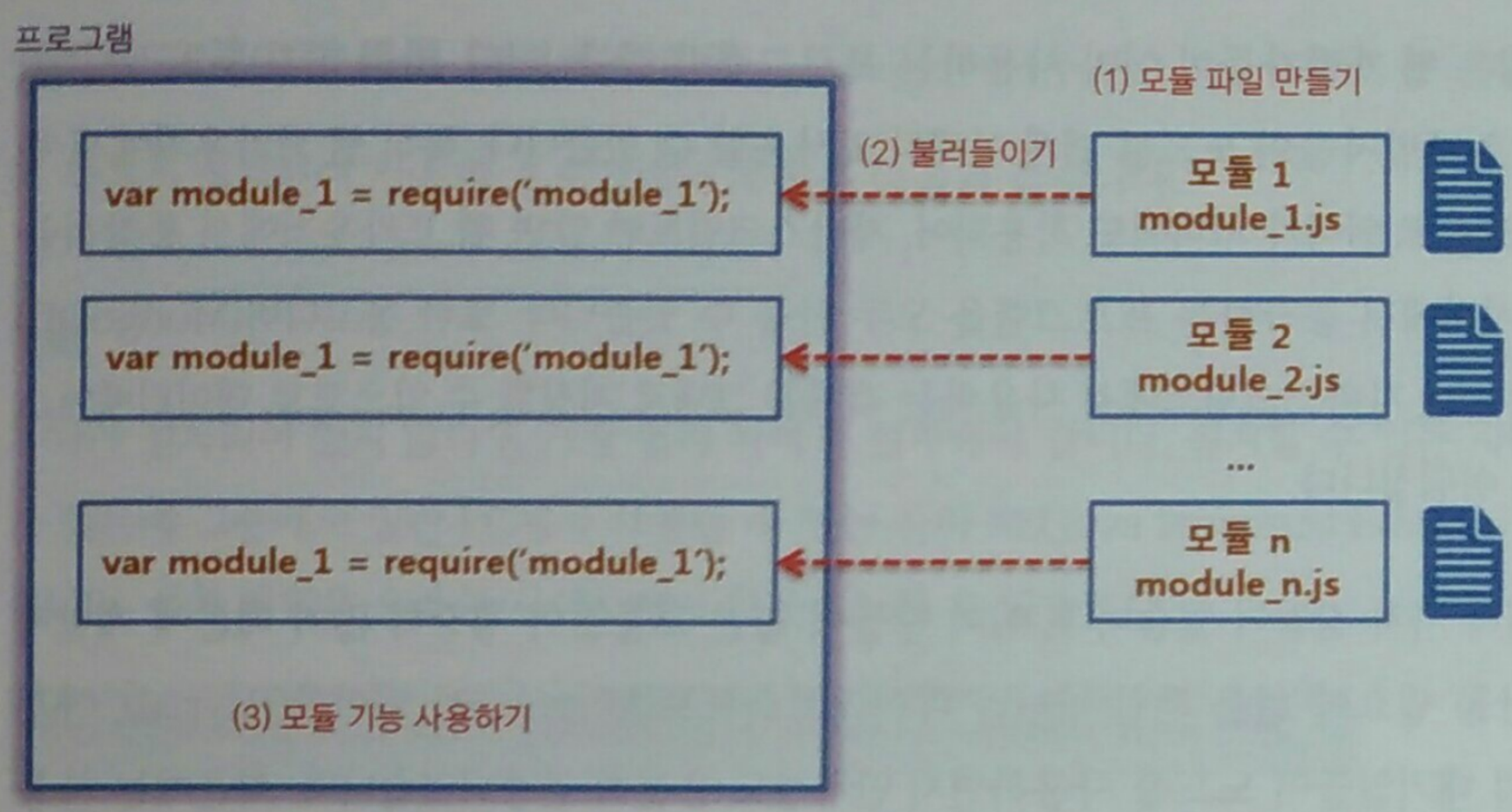
실제 코드 패턴을 예로 들면 다음과 같습니다.

```
res.on('data', function( ) { ... });
```

노드를 더 쉽게 사용할 수 있게 하는 모듈

소스 파일 하나에 실행하려는 기능이 모두 들어 있다면 코드의 양이 많을 뿐만 아니라 복잡해집니다. 이미 잘 알고 있는 것처럼, 웹 브라우저에서 사용하는 자바스크립트는 확장자가 js인 별도의 파일로 만들면 코드를 분리해서 관리할 수 있고 필요할 때 불러서 사용할 수 있습니다. 이와 마찬가지로 노드에서도 필요한 기능을 별도의 자바스크립트로 만든 후 필요할 때마다 불러올 수 있습니다. 다만 그 형태는 **CommonJs 표준 스펙**을 따릅니다. 노드에서 코드를 각각의 파일로 분리시킨 후 필요할 때 불러와 사용하는 과정을 살펴보겠습니다.

용어! CommonJs는 자바스크립트를 브라우저뿐 아니라 서버 쪽 프로그램이나 PC용 프로그램에서도 사용하려고 조직한 자발적인 워킹 그룹입니다. CommonJs 표준 스펙은 자바스크립트 코드를 별도의 모듈 파일로 분리시키고 필요할 때 불러와 사용할 수 있는 방식을 정의해 둔 표준입니다.



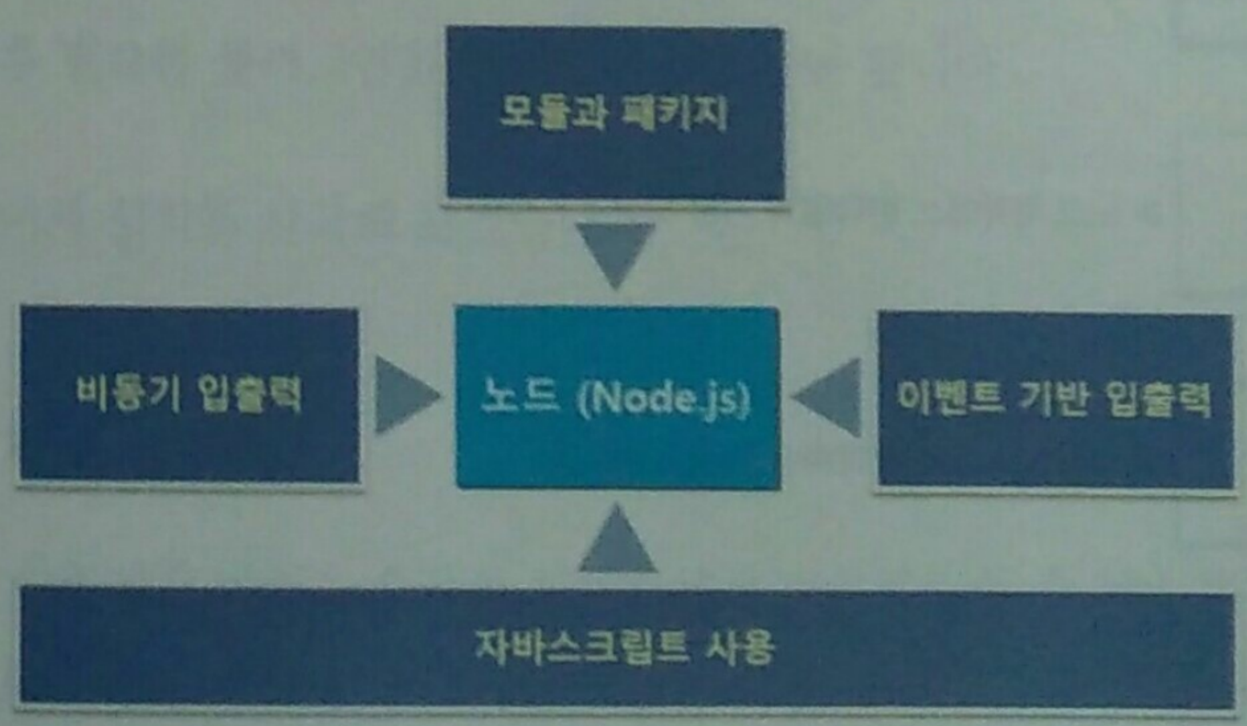
▲노드의 모듈 사용 방식

먼저 메인이 되는 자바스크립트 파일의 일부 코드를 떼어 별도의 파일로 만들 수 있는데 이것을 **모듈(Module)**이라고 부릅니다. 예를 들어, 코드의 일부를 떼어 module_1.js라는 이름의 파일로 저장한다면 이 파일이 모듈입니다. 그리고 이 파일에 들어 있는 코드를 사용하고 싶다면 require( ) 함수로 모듈을 호출하면 됩니다. 이렇게 불러들인 파일은 자바스크립트 객체로 인식되며, 그 객체를 참조하여 파일에 넣어 둔 기능을 사용할 수 있습니다.

이렇게 여러 프로그램에서 공통으로 사용하는 기능은 모듈로 분리하여 구성하는 것이 일반적입니다. 또 여러 개의 모듈을 합쳐서 하나의 **패키지(Package)**로 만들어 두면, 다른 프로그래머들도 npm 프로그램으로 손쉽게 패키지를 설치하여 사용할 수 있습니다. 노드에서 모듈을 어떻게 만들고 어떻게 사용하는지 그리고 npm을 어떻게 사용하는지는 나중에 코드를 만들면서 자세하게 알아보겠습니다.

용어! npm은 Node Package Manager의 줄임말로 다른 프로그래머가 미리 개발하여 올려 둔 패키지를 찾아 설치하는 방법을 제공합니다.

지금까지 노드가 어떻게 만들어졌고 노드의 주요 특징은 무엇인지 알아보았습니다. 다시 한 번 정리하면, 노드는 기본적으로 자바스크립트를 프로그래밍 언어로 사용하며 다음과 같은 세 가지 주요 특징이 있습니다.



◀노드의 주요 특징

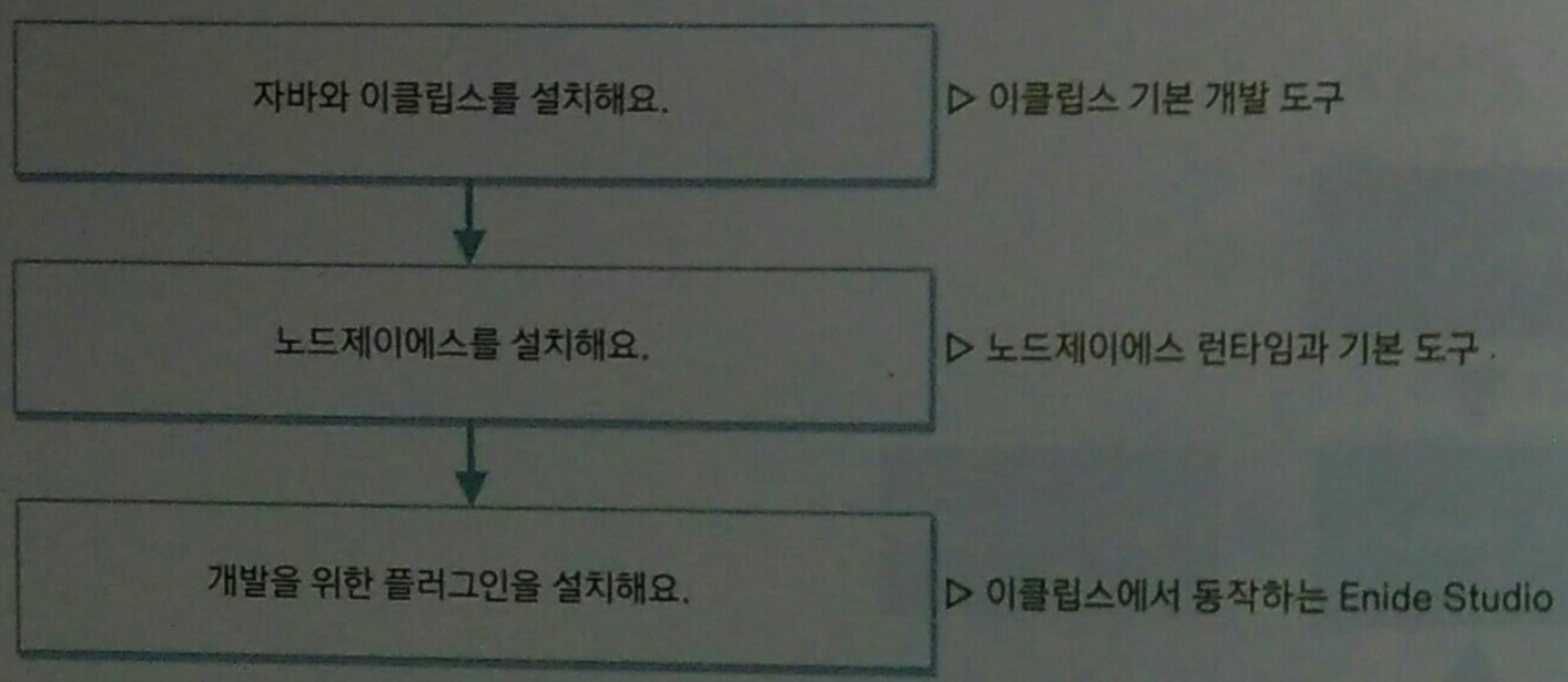
자바스크립트는 수많은 웹 개발자들이 이미 사용하는 프로그래밍 언어입니다. 따라서 자바스크립트에 익숙한 전 세계의 많은 개발자들이 노드를 쉽게 이해하고 사용할 수 있습니다. 특히 웹 브라우저에서 주로 사용하던 자바스크립트 언어가 서버에도 적용되어, 자바스크립트만 알면 웹 브라우저에서 동작하는 클라이언트 기능과 서버에서 동작하는 프로그램을 모두 만들 수 있습니다. 또한 몽고디비(MongoDB) 데이터베이스를 사용하면 자바스크립트에서 사용하는 객체를 그대로 저장할 수 있으므로 데이터베이스를 다루는 것도 훨씬 쉬워집니다.

이 외에도 노드는 여러 가지 장점이 있습니다. 이미 만들어 놓은 모듈들이 굉장히 많기 때문에 개발자가 새로운 기능을 만들 필요가 없을 뿐만 아니라 마이크로소프트(Microsoft), 페이팔(Paypal), 야후(Yahoo) 같은 글로벌 대기업들이 노드를 사용하면서 안정성도 충분히 검증되었습니다. 최근에는 사용할 수 있는 모듈의 수가 폭발적으로 증가하고 있으며, 클라우드의 성장과 더불어 빠르게 발전해 이제 노드는 서버를 구축할 때 필수로 배워야 할 개발 도구이자 플랫폼이 되었습니다.

노드에 대해 대략적으로 알고 나니 노드를 빨리 사용해보고 싶은 욕구가 생기지 않나요? 그러면 이제 개발 도구를 설치하고 가장 간단한 프로그램을 만들어 보겠습니다.

01-2 | 개발 도구 설치하기

노드를 개발하는 데 필요한 개발 도구를 설치해 보겠습니다. 이 책에서 제공하는 샘플 코드를 실행하기 위해서는 이 장의 설명에 따라 개발 도구를 먼저 설치해야 합니다. 노드 개발 도구에는 여러 가지가 있지만, 이 책에서는 이클립스(Eclipse) 기반에서 동작하는 **이니드 스튜디오(Enide Studio)**를 설치하겠습니다. 순서대로 하나씩 과정을 따라가면 어렵지 않게 설치할 수 있습니다.



▲ 노드 개발 도구 설치 과정

설치할 프로그램 목록 미리 확인하기

이 책에서 설치하여 사용할 프로그램의 목록은 다음과 같습니다.

자바

자바가 설치되어 있지 않다면 가장 먼저 자바를 설치해야 합니다. 설치할 수 있는 자바 버전은 여러 가지가 있는데 그중에서 일반 PC에서 사용할 수 있는 자바 SE(Java Standard Edition) 버전을 설치해 보겠습니다. 설치 파일은 오라클 사이트에서 다운로드할 수 있습니다.

http://www.oracle.com/technetwork/java/javase/downloads/index.html

이클립스

이클립스는 여러 가지 언어로 개발할 수 있도록 지원하는 개발 도구이며 자바스크립트와 웹 개발에서도 사용할 수 있습니다. 이클립스 중에서도 Java EE 버전을 설치해야 웹 개발이 가능합니다. 설치 파일은 이클립스 사이트에서 다운로드할 수 있습니다.

http://eclipse.org/

노드

노드는 여러분이 개발한 코드를 실행하기 위해 설치해야 하는 프로그램입니다. 설치 파일은 노드 사이트에서 다운로드할 수 있습니다.

http://nodejs.org/

이니드 스튜디오 플러그인

이클립스에서 노드를 위한 프로젝트를 만들려면 이니드 스튜디오(Enide Studio) 플러그인을 추가로 설치해야 합니다. 이클립스의 플러그인은 기능을 추가하고 싶을 때 설치할 수 있습니다. 이클립스를 실행한 후 필요한 플러그인을 검색하여 설치하면 됩니다.

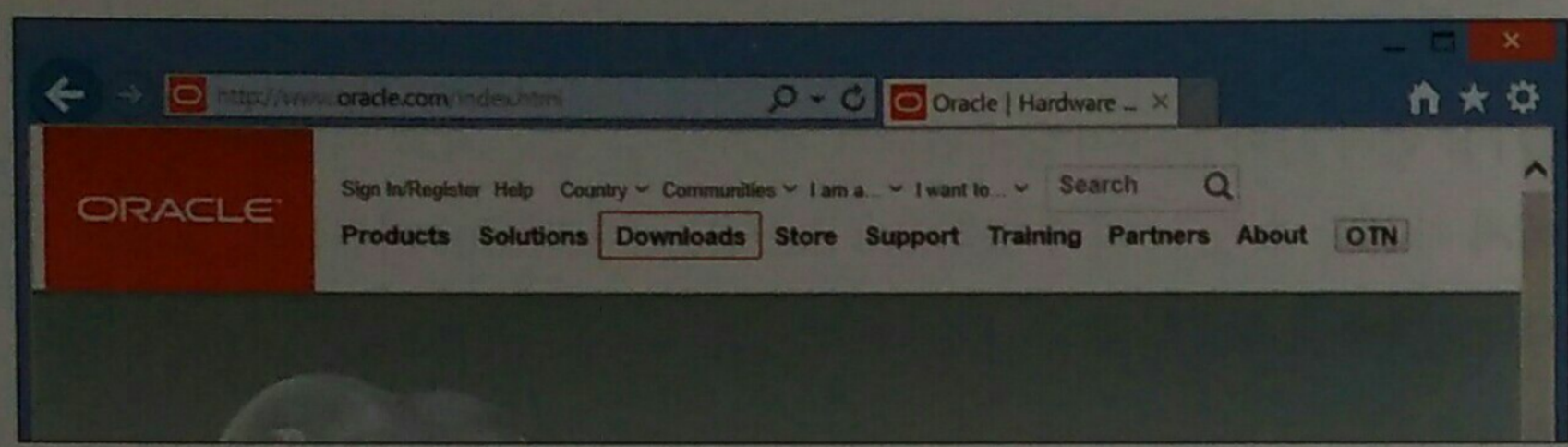
주의! 설치한 프로그램들은 자주 업데이트되므로 언제 설치했는지에 따라 버전이 약간씩 다를 수 있습니다.

이제 설치를 시작해 보겠습니다.

자바 설치하기

자바 설치 파일을 오라클 사이트에서 다운로드한 후 자바를 사용할 노트북이나 PC에 설치합니다. 다운로드하는 시점에 따라 최신 버전은 다를 수 있으므로 가장 최신 버전을 다운로드하십시오.

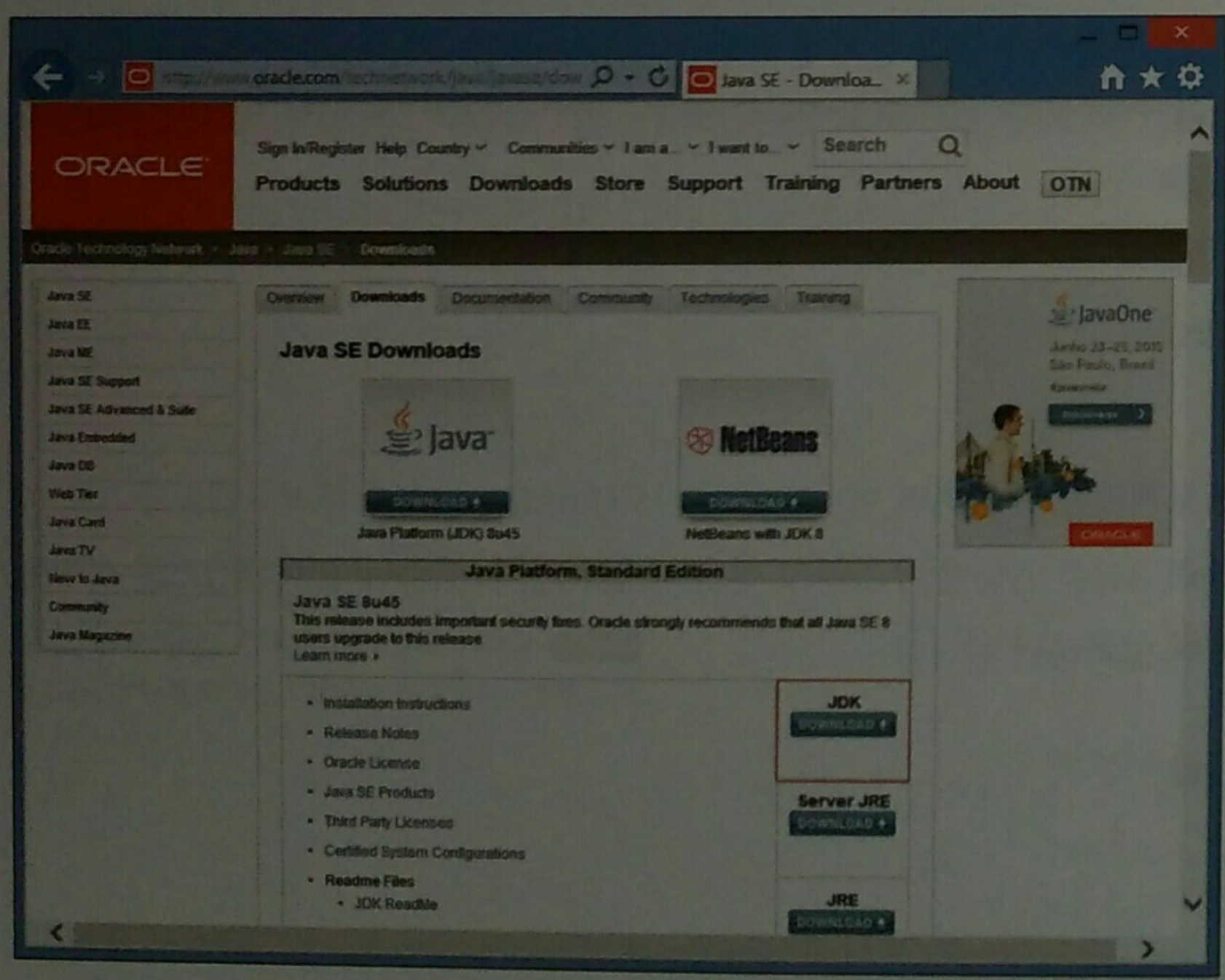
❶ 웹 브라우저를 열고 주소창에 www.oracle.com을 입력한 후 Enter를 누릅니다. 새로 열린 페이지에서 [Downloads] 탭 위로 마우스 커서를 올리면 다운로드 목록이 나타납니다. 목록의 왼쪽 위에 있는 [Popular Downloads] 항목 중에서 [Java for Developers]를 클릭합니다.



정박사의 한마디 내 컴퓨터가 32비트인지 64비트인지 확인하는 방법

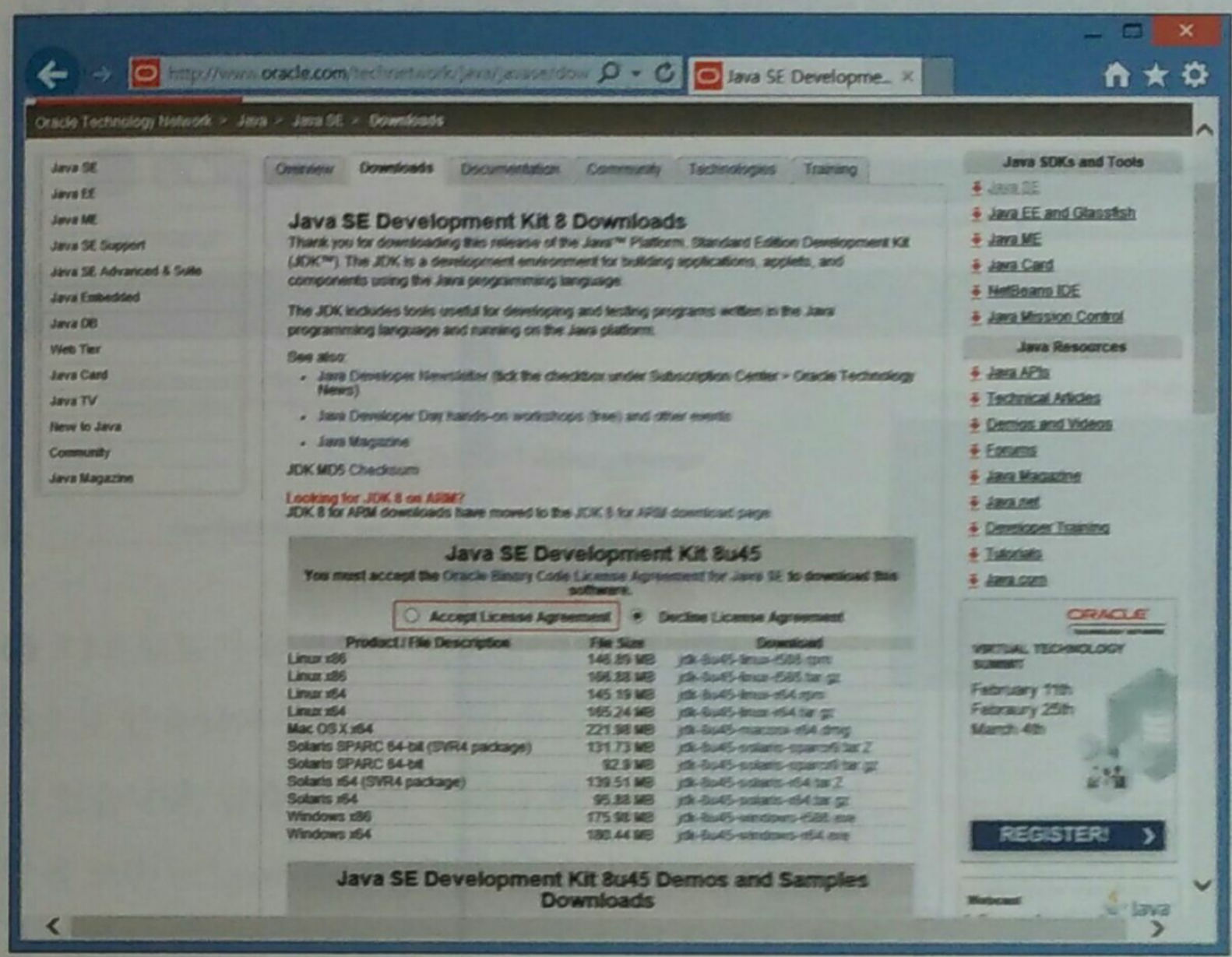
내 컴퓨터 아이콘 위에서 마우스 오른쪽 버튼을 누르면 [속성] 메뉴를 선택할 수 있습니다. 속성 화면에 시스템에 대한 정보가 있으며 여기에서 32비트용인지 64비트용인지 알 수 있습니다.
시작 메뉴에서 찾아보려면 [시작 → 제어판 → 시스템 및 보안 → 시스템] 메뉴를 눌러 나타나는 화면에서 확인할 수 있습니다.

❷ 스크롤바를 화면의 아래쪽으로 내리면 JDK라는 글자 아래에 [DOWNLOAD] 버튼이 보입니다. 이 [DOWNLOAD] 버튼을 클릭합니다.



❸ 파일 링크 표시를 눌러 다운로드하기 전에 Accept License Agreement를 체크합니다. 그런 다음 자신의 PC 사양에 맞는 파일 링크를 클릭하여 다운로드합니다. 사용자의 윈도우가 32비트라면 x86(jdk-

택하면 됩니다. 파일 다운로드가 끝나면 설치를 진행합니다. 설치는 [Next] 버튼만 클릭하면 되므로 쉽게 진행할 수 있습니다.

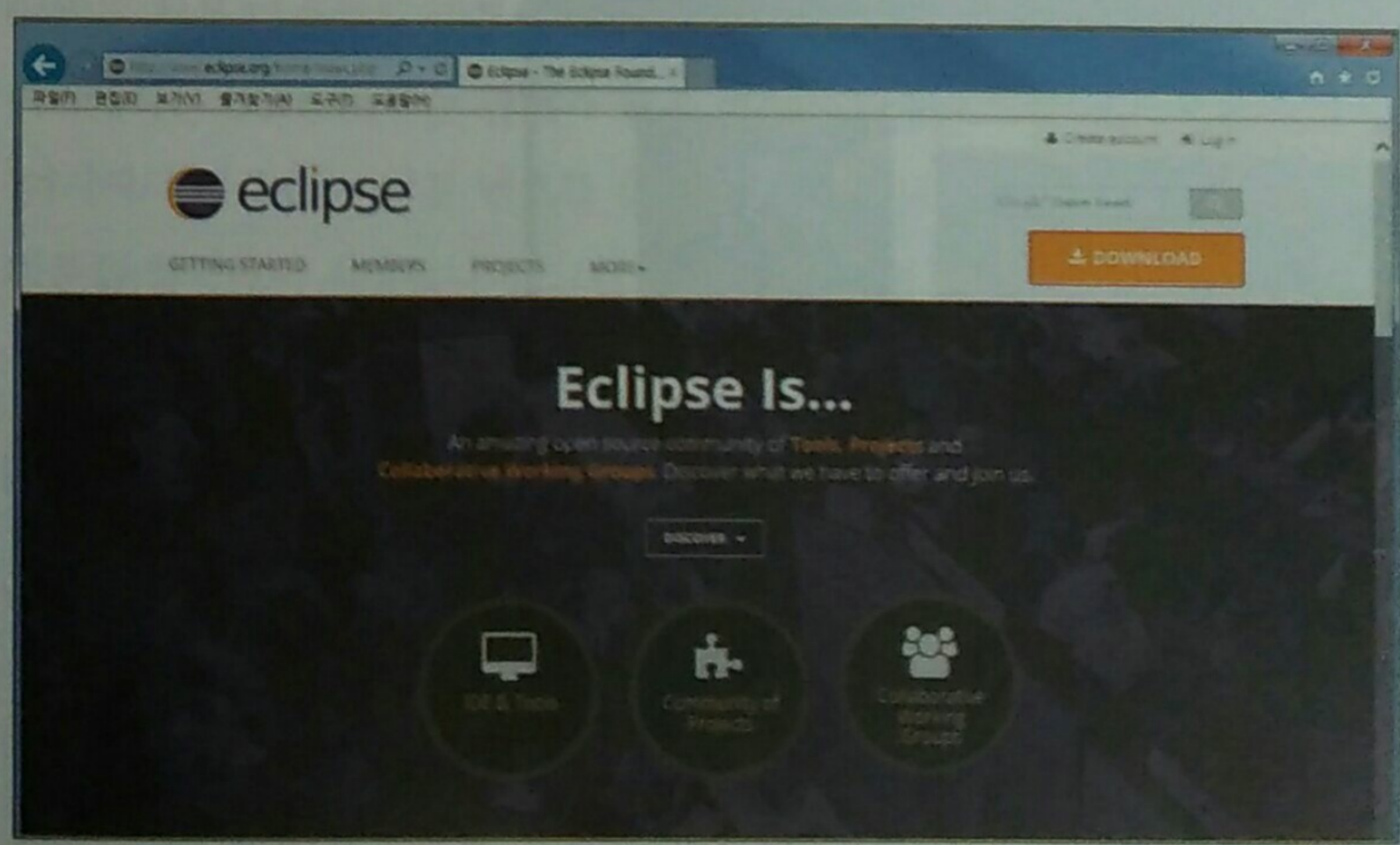


자바 설치가 끝났습니다. 이제 이클립스를 설치해 보겠습니다.

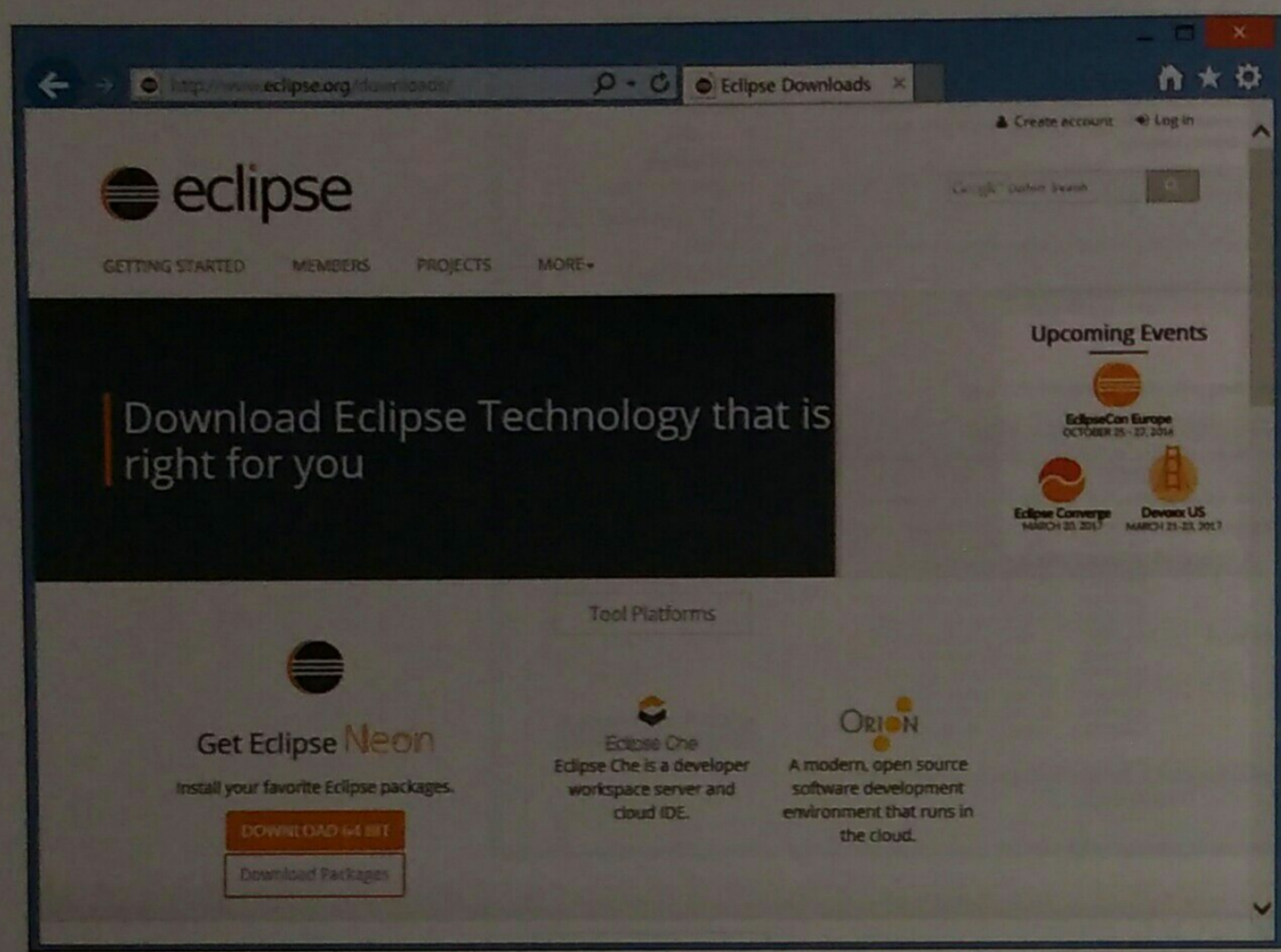
이클립스 설치하기

이클립스는 자바로 만든 프로그램이므로 자바를 먼저 설치해야 사용할 수 있습니다. 이클립스는 별도의 설치 과정이 필요하지 않기 때문에 사이트에서 다운로드한 후 압축을 풀면 바로 사용할 수 있습니다.

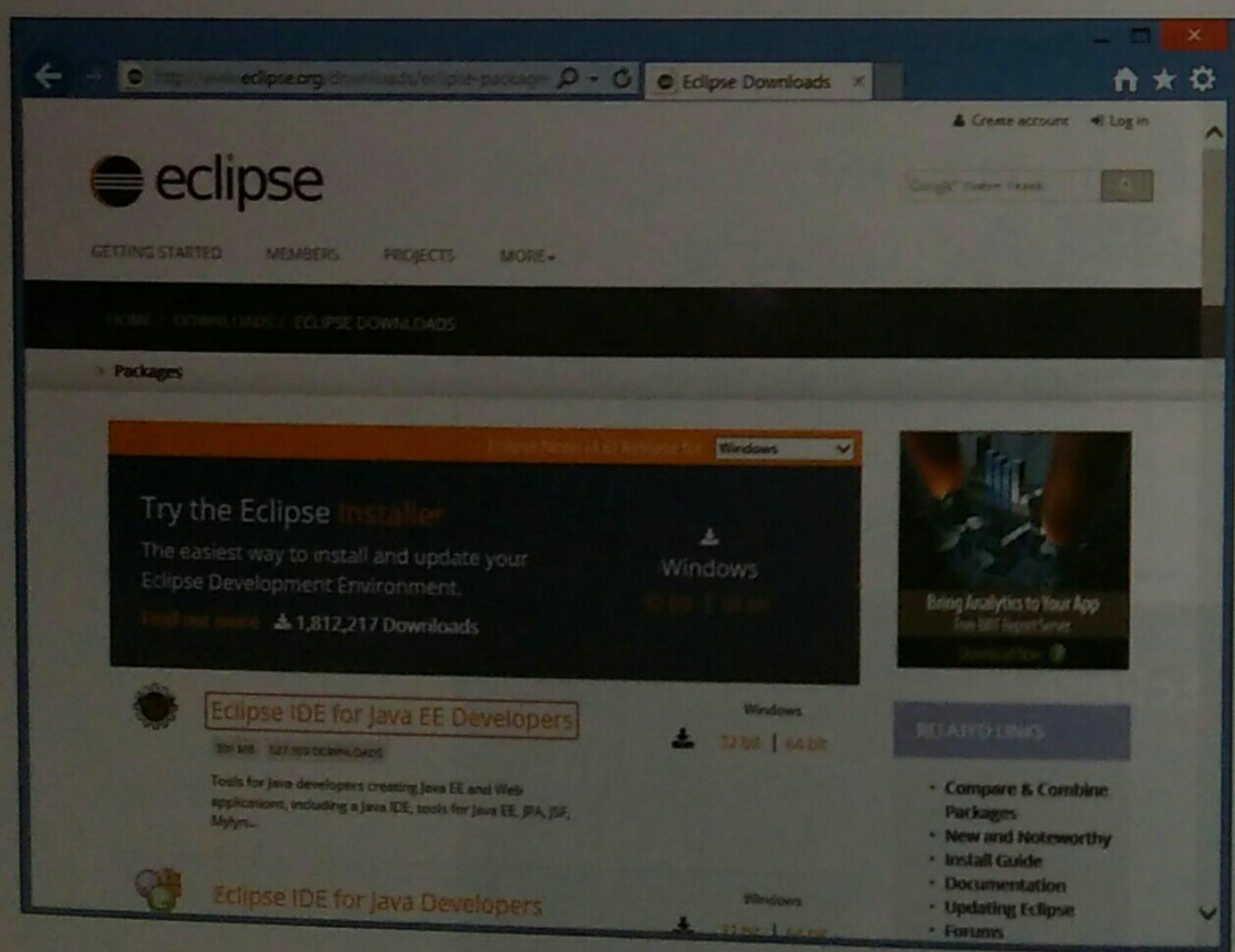
❶ 이클립스 사이트(http://eclipse.org)에 접속하면 다음과 같은 화면을 볼 수 있습니다. 화면 오른쪽에 있는 [Download] 버튼을 클릭하면 다운로드 화면으로 넘어갑니다.



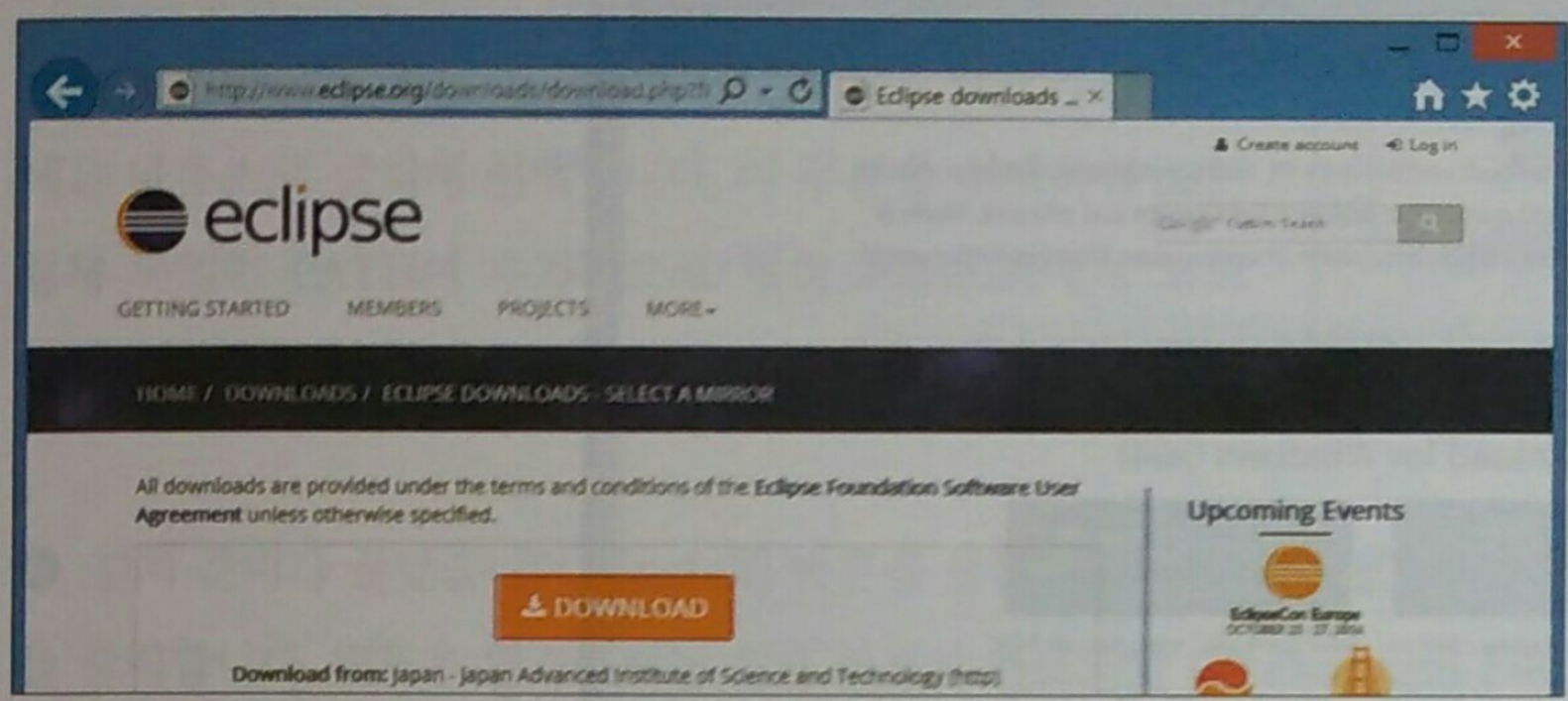
❷ 다운로드 화면의 왼쪽 아래에 [DOWNLOAD 64 BIT] 버튼이 보입니다. 이 버튼은 기본적인 이클립스 설치 파일을 다운로드하는 것입니다. 여러분은 웹 개발을 할 수 있는 이클립스 버전을 설치하기 위해 버튼 아래에 있는 Download Packages 링크를 클릭합니다. 그러면 다운로드할 수 있는 여러 가지 파일들이 나타납니다.



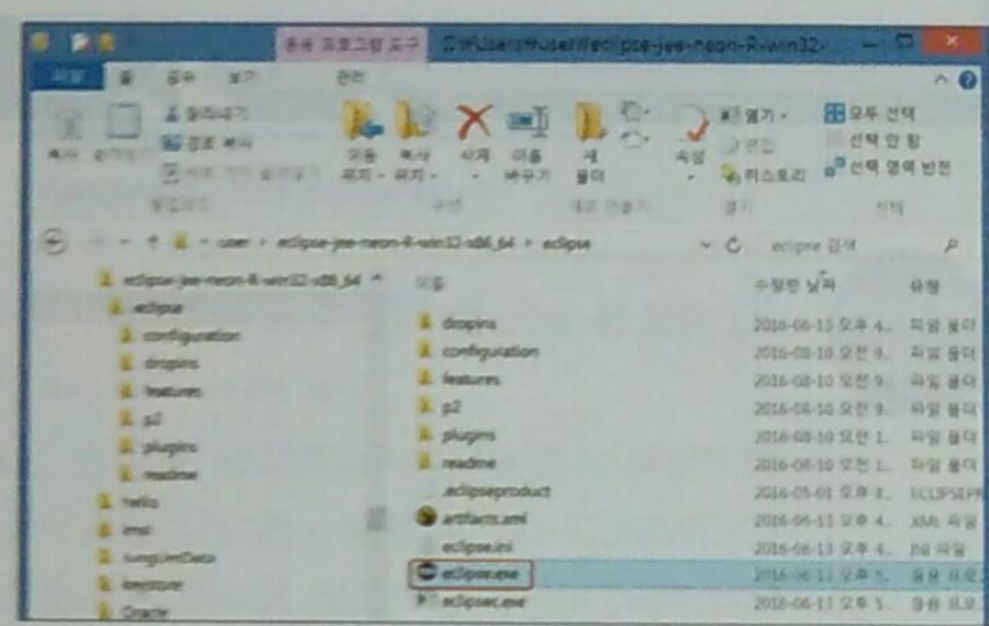
❸ 여러 파일들 중에서 Eclipse IDE for Java EE Developers를 눌러서 다운로드합니다. 자바를 설치할 때 확인했던 것처럼 설치하려는 노트북이나 PC가 32비트용이면 32 Bit, 64비트용이면 64 Bit 링크를 누릅니다.



❹ 다운로드할 수 있는 서버는 여러 곳이 있는데 그중 하나를 누르면 다운로드가 시작됩니다. 여기에서는 가운데 있는 [DOWNLOAD] 버튼을 클릭해서 다운로드합니다.



❺ 다운로드가 완료되면 압축을 풀어 줍니다. zip 포맷으로 압축되어 있으므로 일반 압축 프로그램으로 풀 수 있습니다. 압축을 풀면 폴더가 만들어지는데, 폴더 안을 보면 eclipse.exe 실행 파일이 있습니다. 이 파일을 더블클릭하면 이클립스를 실행할 수 있습니다. 필요하면 바탕 화면에 바로 가기 아이콘을 만들어 사용하면 됩니다.



이클립스 설치가 끝났습니다. 이제 노드를 설치해 보겠습니다.

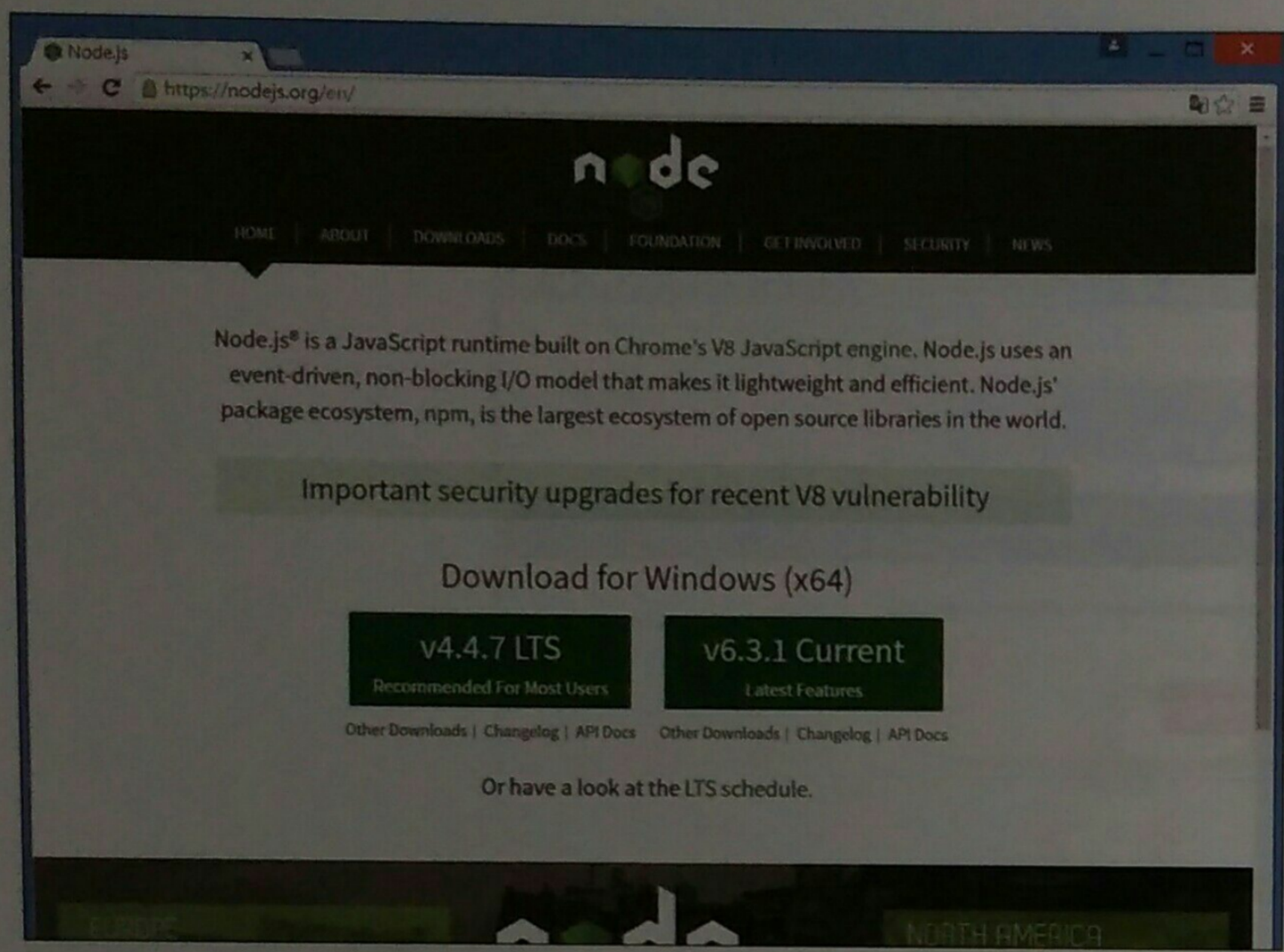
노드 설치하기

노드 설치 프로그램은 노드 사이트에서 다운로드할 수 있습니다.

❶ 노드 사이트(https://nodejs.org/en)에 접속하면 첫 화면에 Download for Windows (x64)라고 64비트 전용 다운로드 버튼이 두 개 나타납니다. 두 개의 버튼 중 어떤 버튼을 클릭하든 상관없습니다. 여기에서는 Current라는 단어가 붙은 버전을 다운로드합니다. 오른쪽에 있는 [v0.0.0 Current] 버튼(여기에서 0.0.0는 버전 번호를 의미)을 클릭합니다. 실행 및 저장을 위한 팝업 창이 나타나면 [저장] 버튼을 클릭해서 저장합니다. 그런 다음 저장이 완료되면 [실행] 버튼을 클릭해서 설치를 시작합니다.

정박사의 한마디 LTS와 Current는 차이가 약간 있어요

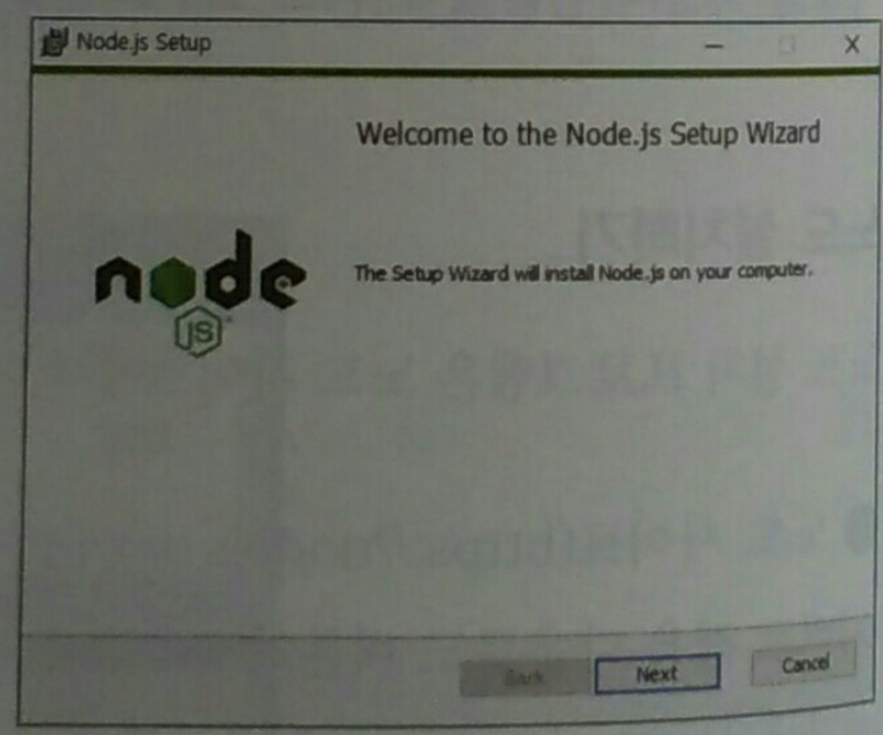
LTS(Long Term Supported) 버전은 서버 환경에서 안정적으로 동작하도록 지원하는 버전입니다. 노드를 사용하는 목적이 서버 개발인데 실제로 서버를 운영할 때는 안정성과 보안이 중요하므로 LTS 버전을 사용하는 것이 좋습니다. LTS 버전에 비해 Current 버전은 최신 기능을 포함합니다. 새로운 기능을 추가하거나 기존 버그들을 우선적으로 수정하여 적용하므로 최신 기능을 사용할 때는 Current 버전을 설치하는 것이 좋습니다.



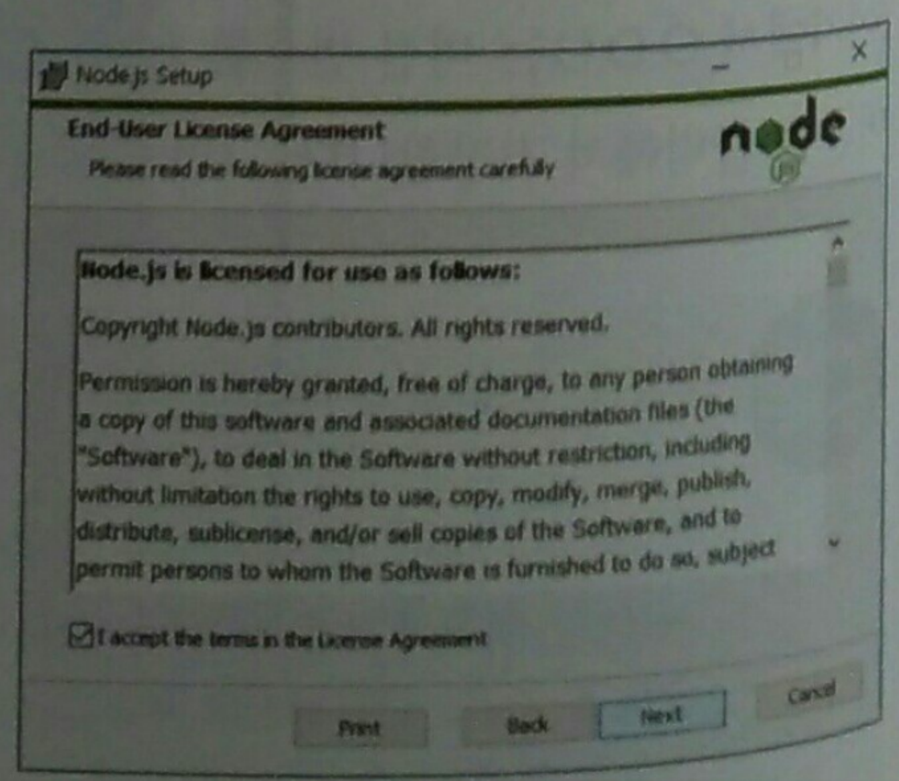
정박사의 한마디 32비트용 노드를 다운로드하려면?

설치하려는 컴퓨터가 32비트일 때에는 노드 사이트(https://nodejs.org/en) 위쪽에 있는 탭 중에서 [DOWNLOADS] 버튼을 누릅니다. 윈도우용과 매킨토시용 다운로드 링크를 확인할 수 있습니다. [Windows Installer] 버튼을 클릭하면 32비트 윈도우용 설치 파일을 다운로드할 수 있습니다.

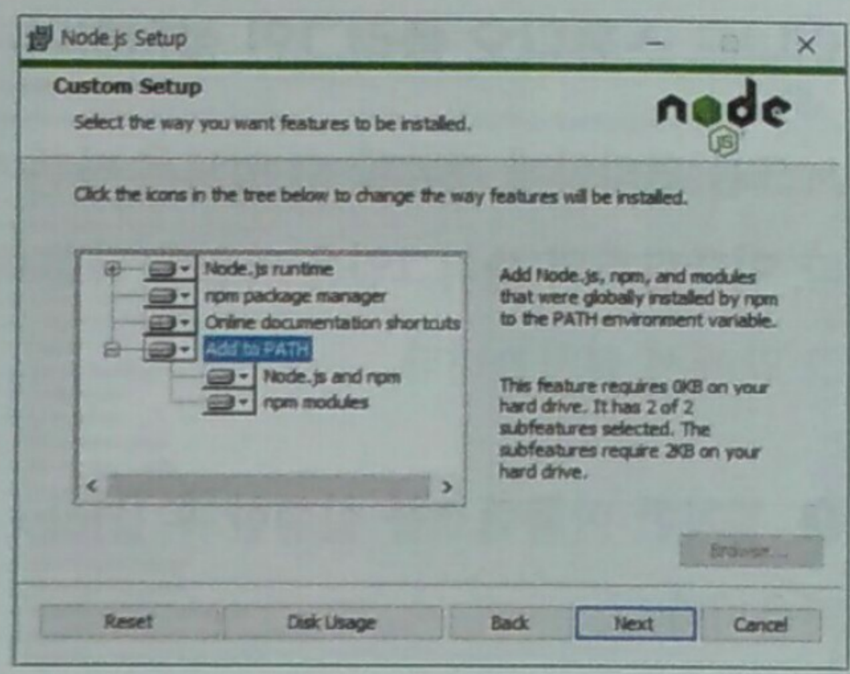
❷ 설치를 진행하면 다음과 같은 설치 화면이 나타납니다. 특별히 설정하거나 선택하는 내용이 없으므로 [Next] 버튼을 누르면서 몇 단계의 화면만 따라가면 설치가 완료됩니다.



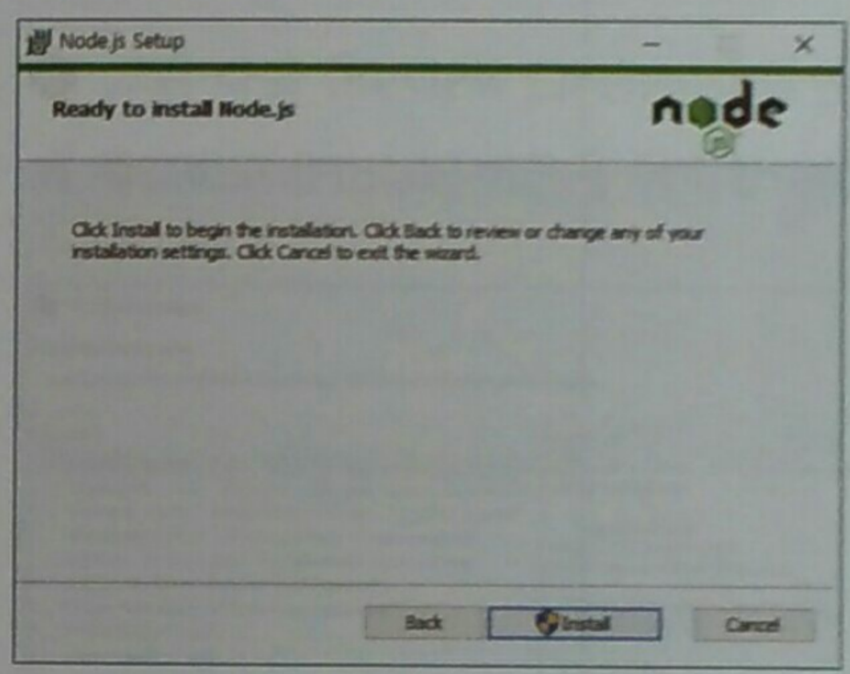
❸ 라이선스에 동의를 요구하는 화면이 나타나면 아래쪽의 I accept the terms in the License Agreement에 체크를 한 다음 [Next] 버튼을 눌러 설치를 계속 진행합니다.

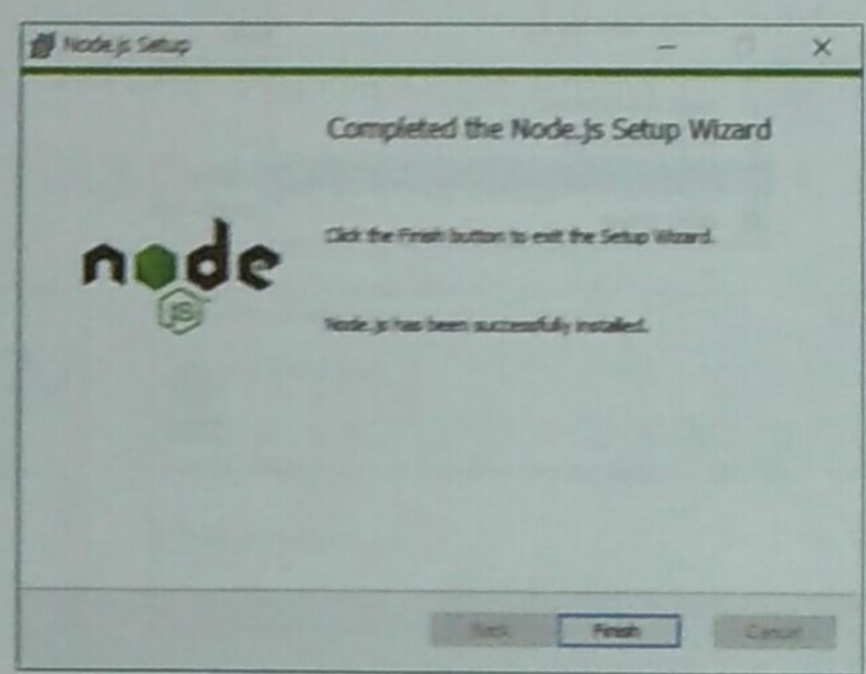


❹ 설치 화면 중에서 Custom Setup 화면이 보이면 Add to PATH 항목이 선택되었는지 확인합니다. 설치하려는 PC에 따라 이 항목이 선택되지 않은 경우가 있으므로 선택 유무를 먼저 확인합니다. Add to PATH가 선택되어 있지 않을 경우에는 마우스로 클릭해 선택합니다. 이 부분을 선택하면 노드 실행 파일이 PATH에 추가되므로 명령 프롬프트에서 노드 명령어를 사용할 수 있습니다.

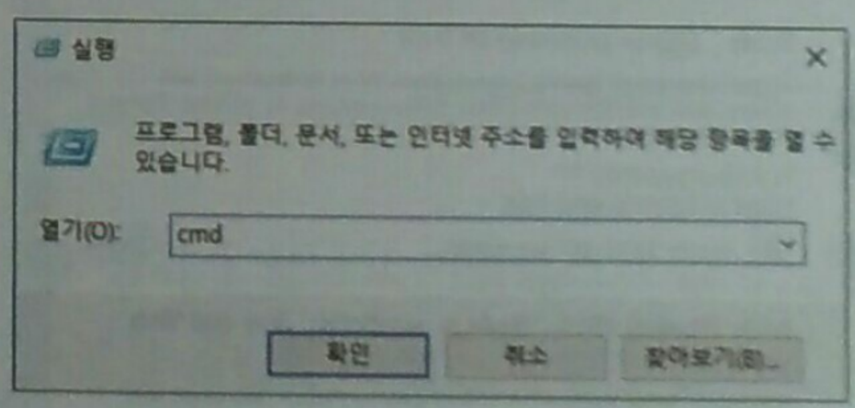


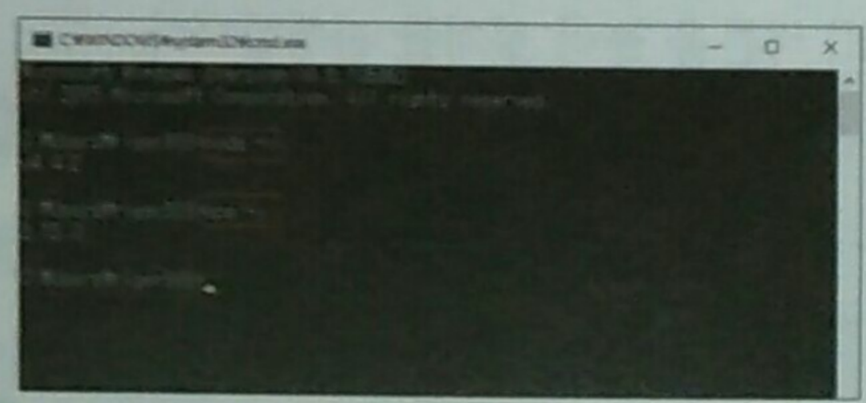
❺ 설치 준비가 끝났습니다. [Install] 버튼을 클릭해서 설치를 진행한 후 설치가 완료되면 [Finish] 버튼을 클릭합니다. 윈도우 환경에서는 설치 과정이 완료되면 C:\Program Files\nodejs 폴더 경로에 노드가 설치됩니다.





❻ 설치가 완료되면 [시작 → 실행...]을 누른 후 cmd를 입력하고 [확인] 버튼을 클릭해서 명령 프롬프트 창을 띄웁니다. 여기에서 node -v 명령을 입력하면 노드 버전이 표시됩니다. npm은 패키지 설치를 도와주는 도구이며 npm -v 명령을 입력하면 버전이 표시됩니다. 이 두 가지가 문제없이 표시되면 잘 설치된 것입니다.

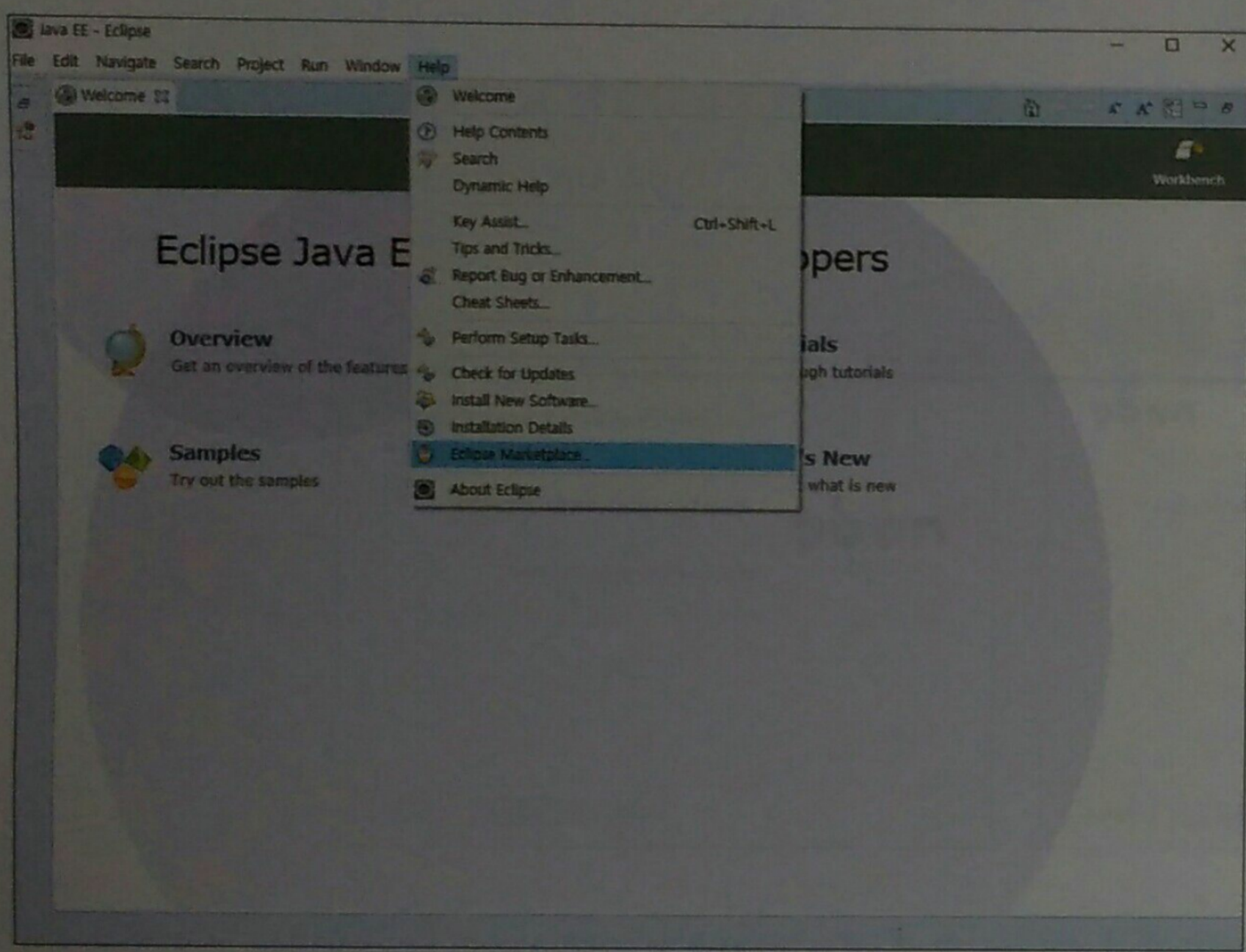




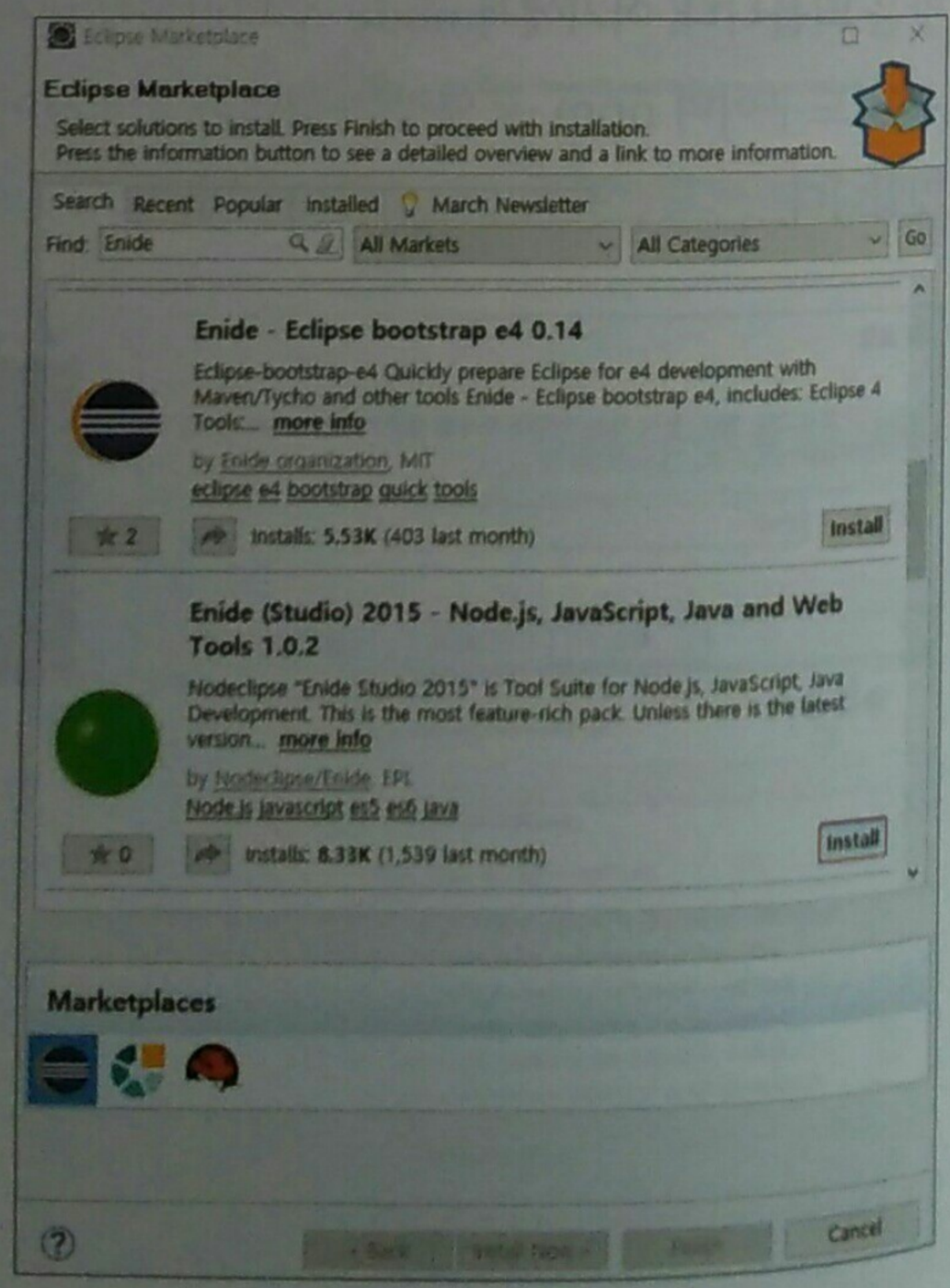
이니드 스튜디오 플러그인 설치하기

노드를 편리하게 개발할 수 있도록 만든 이니드 스튜디오는 이클립스의 플러그인으로 설치할 수 있습니다. 이클립스의 플러그인은 파일을 다운로드하여 설치하는 것이 아니라 이클립스 개발 도구를 실행한 후 그 안에서 설치합니다.

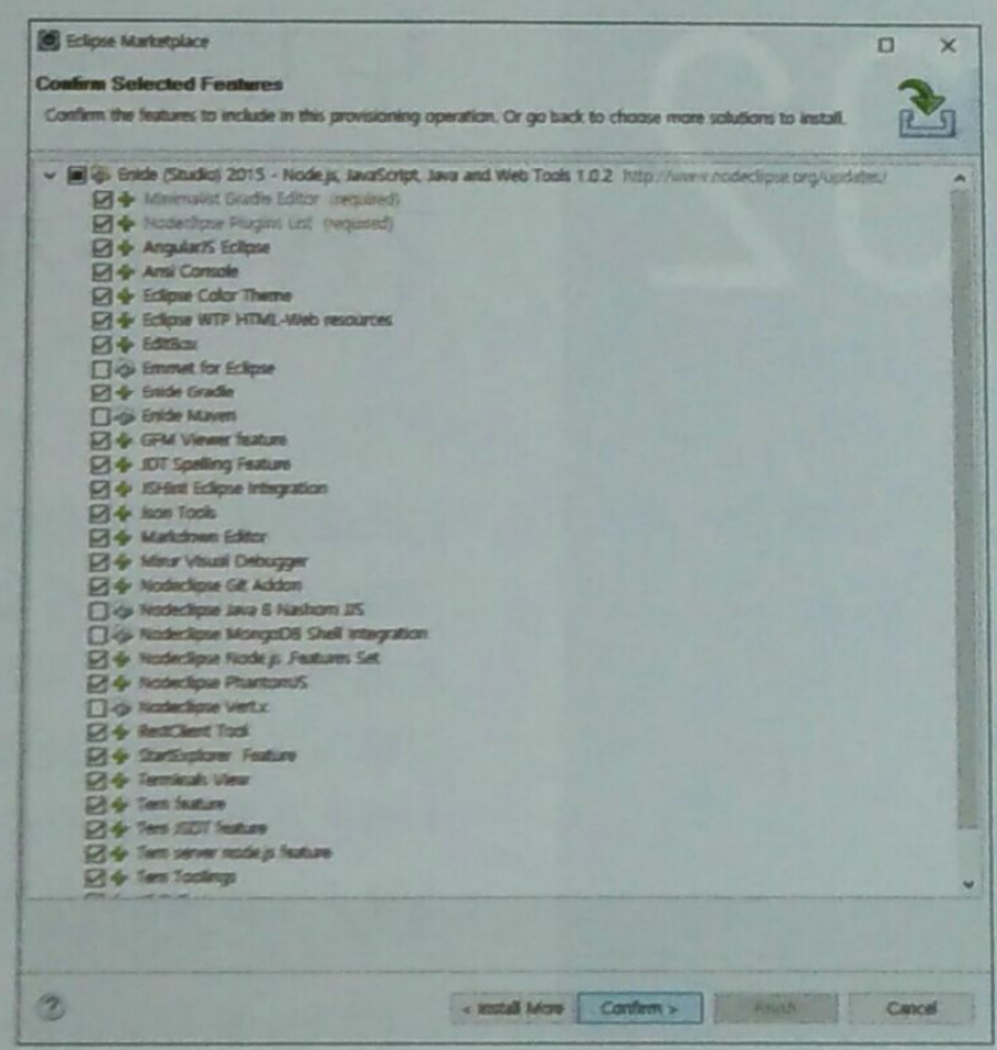
❶ 설치한 이클립스를 실행한 후 [Help → Eclipse Marketplace] 메뉴를 눌러 이클립스 장터 화면을 띄웁니다.



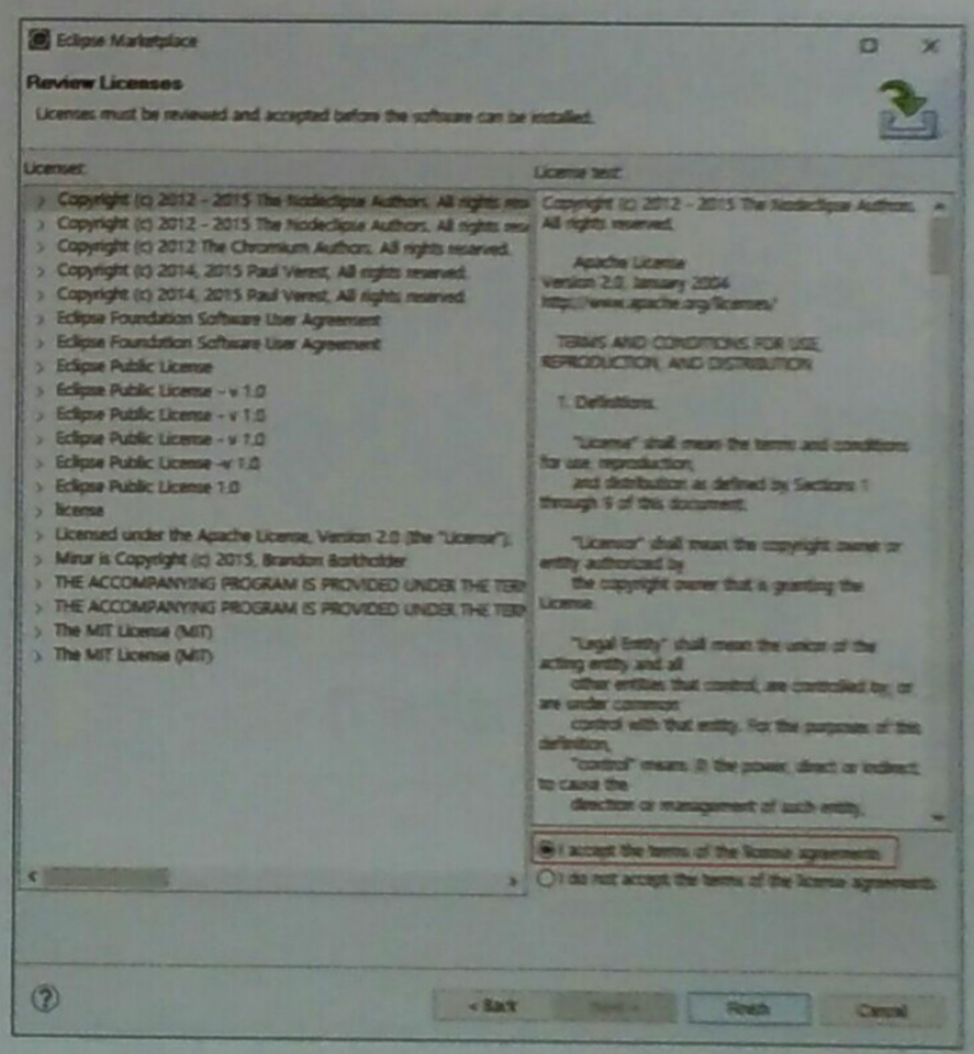
❷ 이클립스 장터 대화상자가 보이면 위쪽의 Find라는 검색란에 Enide라고 입력한 후 Enter를 눌러 검색합니다. 스크롤바를 내려서 아래쪽에 Enide Studio 항목이 보이면 [Install] 버튼을 눌러 설치를 진행합니다. 여기에서는 최신 버전인 Enide (Studio) 2015를 선택했습니다.

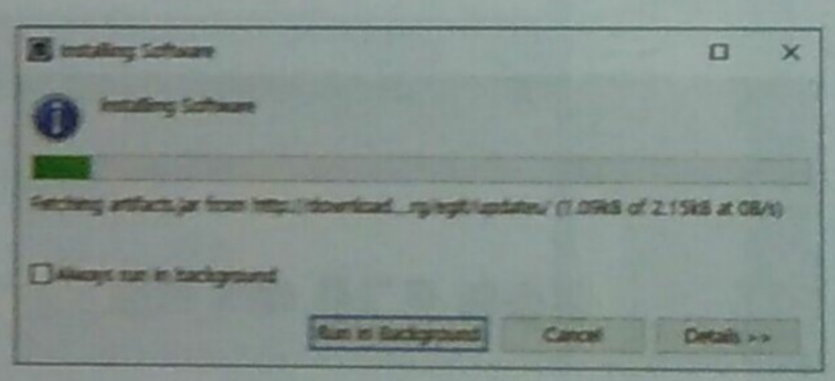


❸ 플러그인들의 설치 목록이 나타나면 선택 목록을 확인하는 [Confirm Selected Features] 대화상자가 나타납니다. [Confirm] 버튼을 클릭하고 넘어갑니다.

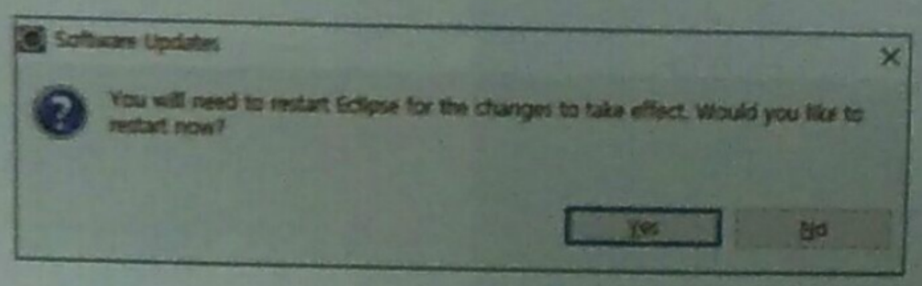


❹ 그런 다음 [Review Licenses] 화면이 나타나면 I accept the terms of the license agreements에 체크하고 [Finish] 버튼을 클릭합니다. 소프트웨어 설치가 진행된 후 모든 설치 과정이 끝납니다.





❺ 설치가 끝나고 이클립스가 다시 실행되면 이제 드디어 노드를 개발할 수 있는 상태가 됩니다. 개발 도구를 모두 설치했으니 다음 장에서는 첫 번째 노드 프로그램을 만들어 실행해 볼 것입니다.



02 노드 간단하게 살펴보기

노드로 프로그램을 만든다는 것은 자바스크립트 파일을 만든다는 것과 같습니다. 자바스크립트로 만든 파일은 앞 장에서 설치한 노드 실행 파일을 사용해서 실행할 수 있습니다. 이 장에서는 누구나 직접 만들 수 있는 첫 번째 노드 프로젝트를 단계별로 살펴봅니다. 그 다음 노드의 기본 사용법에 대해서도 알아봅니다. 이 장의 내용을 하나씩 따라하다 보면 새로운 프로젝트를 만들 수 있으며, 프로젝트 안에 자바스크립트 코드를 만들어 넣어 노드 실행 파일로 실행하기까지 모든 과정을 차근차근 살펴볼 수 있습니다.

도해로 정리하기

첫 번째 노드 프로젝트를 만들어 봐요. ➡ 02-1. 첫 번째 노드 프로젝트 만들기

콘솔에 로그를 뿌려 봐요. ➡ 02-2. 콘솔에 로그 뿌리기

한두 가지 기본 객체에 대해 살펴봐요. ➡ 02-3. 프로세스 객체 간단히 살펴보기

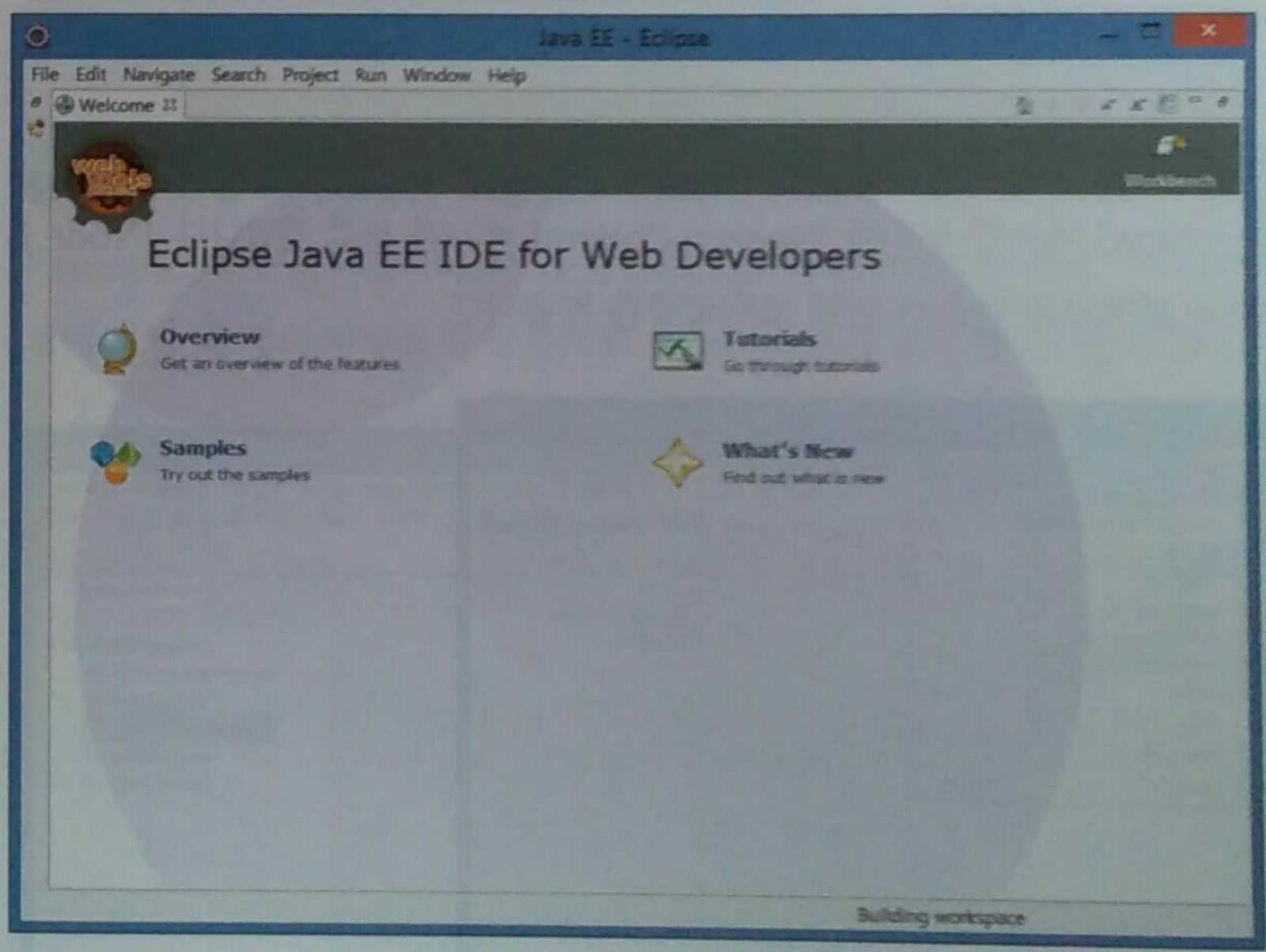
모듈을 만들어 사용하는 방법을 알아봐요. ➡ 02-4. 노드에서 모듈 사용하기

노드에 설치된 기본 내장 모듈 사용법을 알아봐요. ➡ 02-5. 간단한 내장 모듈 사용하기

02-1 | 첫 번째 노드 프로젝트 만들기

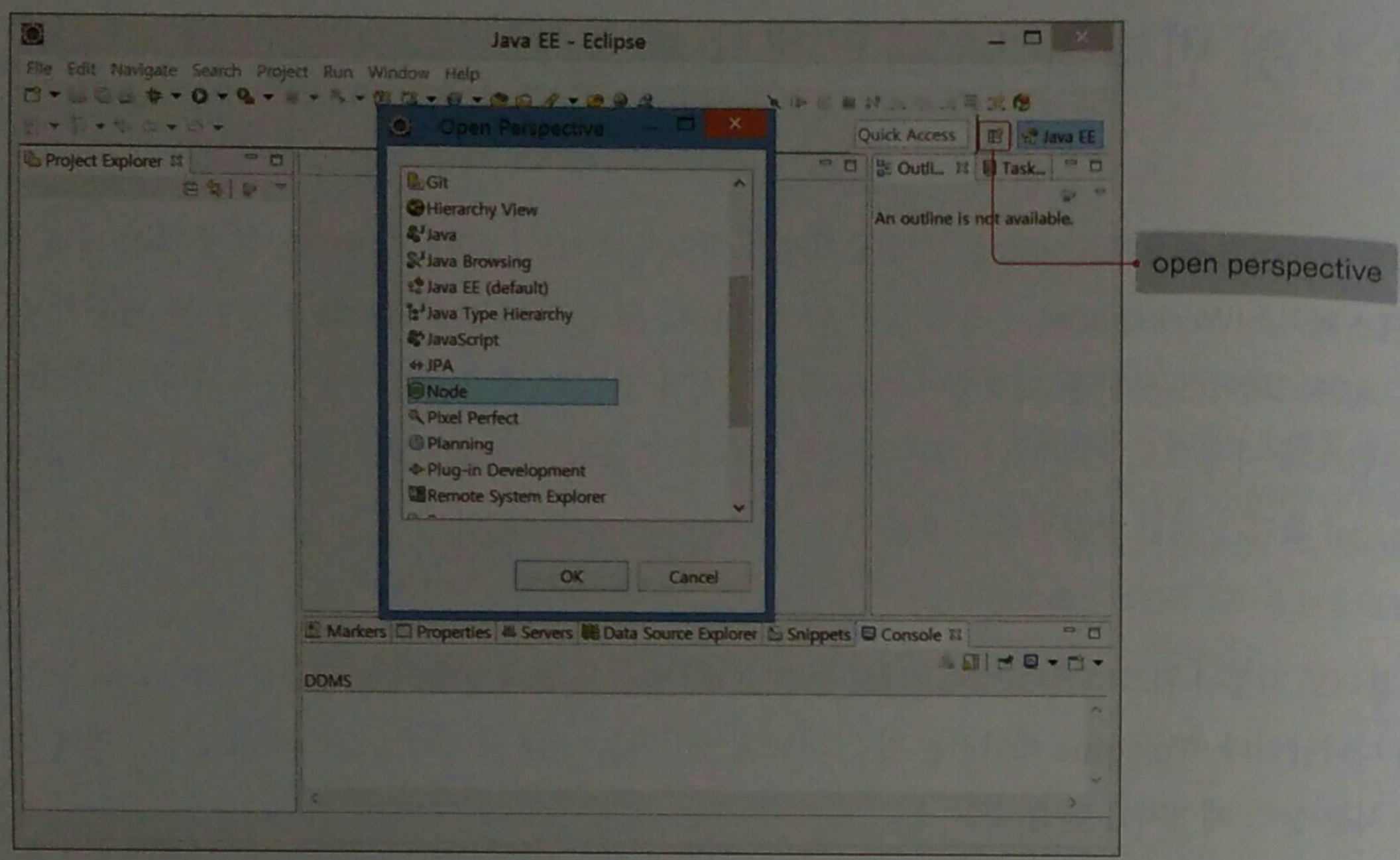
이클립스를 실행하면 프로젝트 파일을 어디에 저장할 것인지 물어보는 워크스페이스 설정 창이 나타납니다. **워크스페이스(Workspace)**는 작업 공간을 의미하며, 새로 만든 프로젝트와 파일들을 저장할 폴더입니다. 워크스페이스는 한번 지정하면 계속해서 지정한 폴더를 작업 공간으로 사용합니다. 여기에서는 이클립스가 자동으로 지정하는 기본 폴더를 그대로 사용합니다. 일반적으로 [사용자] 폴더 밑의 [workspace] 폴더가 작업 공간으로 지정됩니다.

❶ 기본 폴더로 지정된 워크스페이스를 그대로 둔 채 [OK] 버튼을 눌러 실행하면 다음과 같이 Welcome 페이지가 나타납니다. Welcome 페이지는 말 그대로 환영한다는 안내 페이지이므로 [Welcome] 탭에 있는 [X] 표시를 눌러 페이지를 닫습니다.



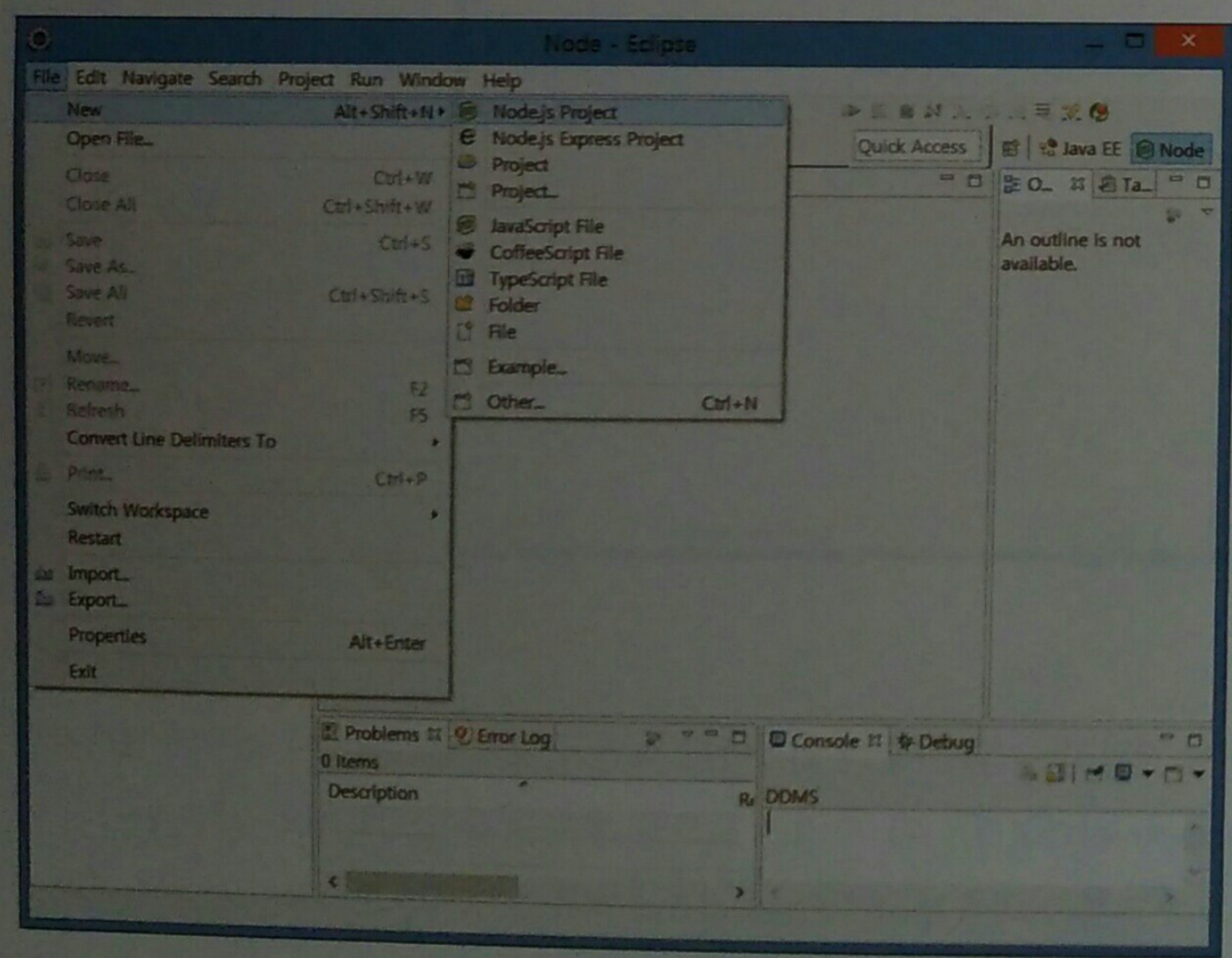
▲ 처음 나타나는 Welcome 페이지

❷ 이클립스는 다양한 언어로 개발할 수 있는데 각 언어마다 화면 구성이 달라집니다. 이렇게 달라지는 각 화면을 **퍼스펙티브(Perspective)**라고 합니다. 우리는 노드로 개발할 것이므로 Node라는 이름의 퍼스펙티브로 전환해야 합니다. 오른쪽 위의 [Open Perspective] 버튼을 누르면 퍼스펙티브 선택 대화상자가 나타납니다. 그 안에 들어 있는 항목들 중에서 Node 퍼스펙티브를 선택합니다.



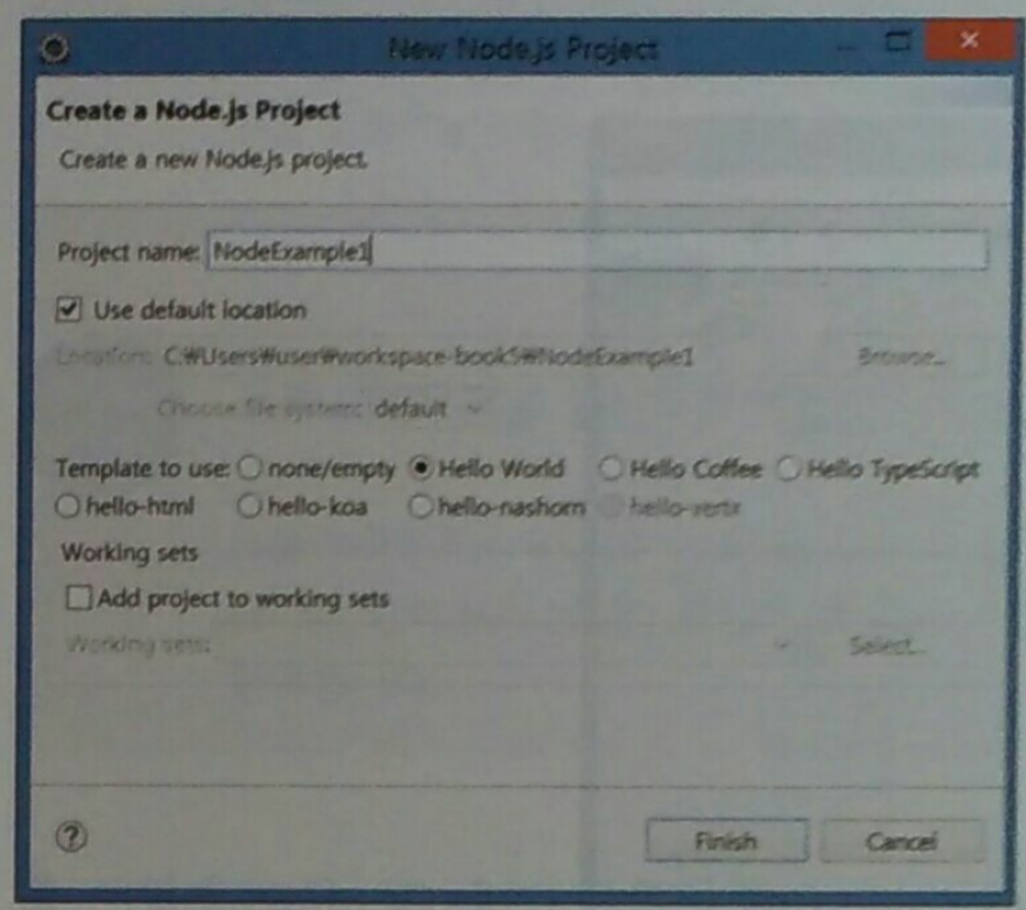
▲ Node 퍼스펙티브를 선택하기 위해 띄운 퍼스펙티브 선택 대화상자

❸ 이전 화면과 크게 다르지 않지만 지금 보고 있는 화면은 노드 개발을 위한 퍼스펙티브 화면입니다. 이제 이클립스 위쪽의 [File → New] 메뉴를 누르면 [Node.js Project] 메뉴가 보일 것입니다. [Node.js Project] 메뉴를 눌러 새로운 프로젝트를 만들기 위한 대화상자를 띄웁니다.



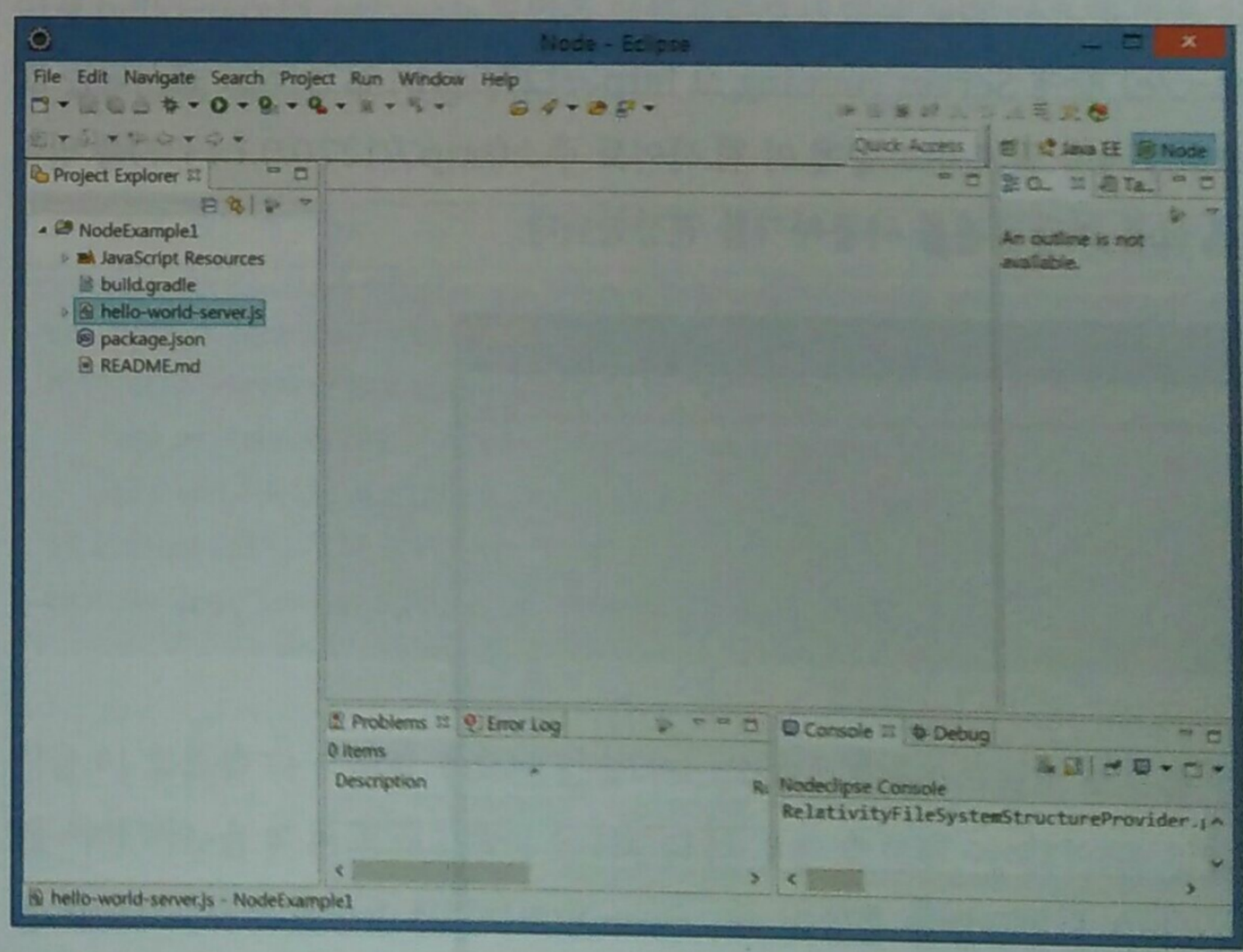
▲ [Node.js Project] 메뉴를 눌러 새로운 노드 프로젝트 만들기

❹ 새로운 프로젝트를 만드는 화면이 나타나면 Project name 입력 상자에 NodeExample1이라고 입력합니다. 그 아래쪽에는 여러 가지를 선택할 수 있는 항목이 보이는데, 그중에서 Template to use 항목을 Hello World로 선택한 후 [Finish] 버튼을 누릅니다.



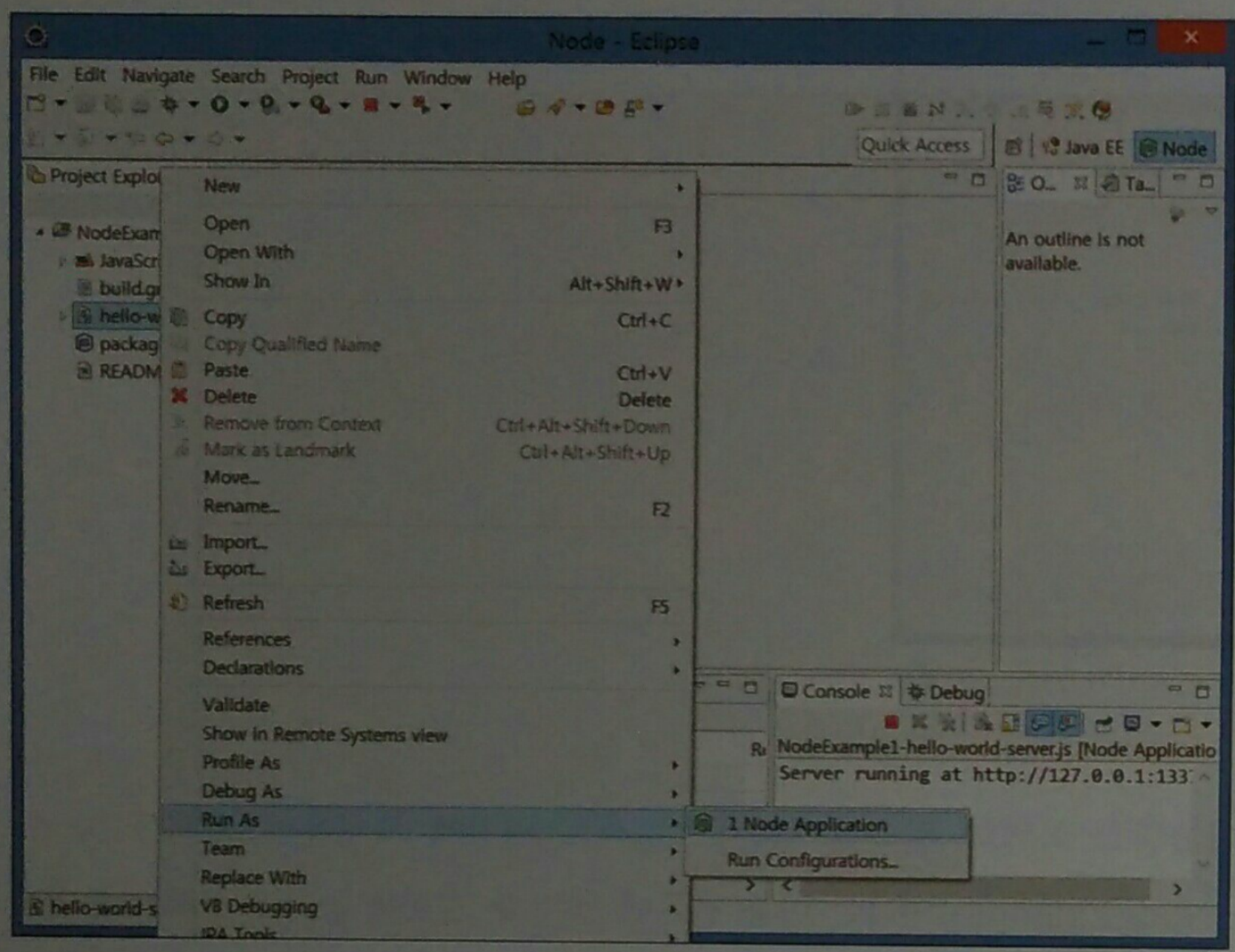
▲ 새로운 프로젝트 만들기 대화상자

❺ 대화상자가 사라지면 이클립스 화면의 왼쪽에 있는 프로젝트 영역에 새로운 프로젝트 NodeExample1이 만들어집니다. NodeExample1이라는 이름 왼쪽에 있는 ▷ 기호를 눌러 프로젝트 폴더를 열어 보면 몇 개의 파일들이 자동으로 만들어진 것을 확인할 수 있습니다.



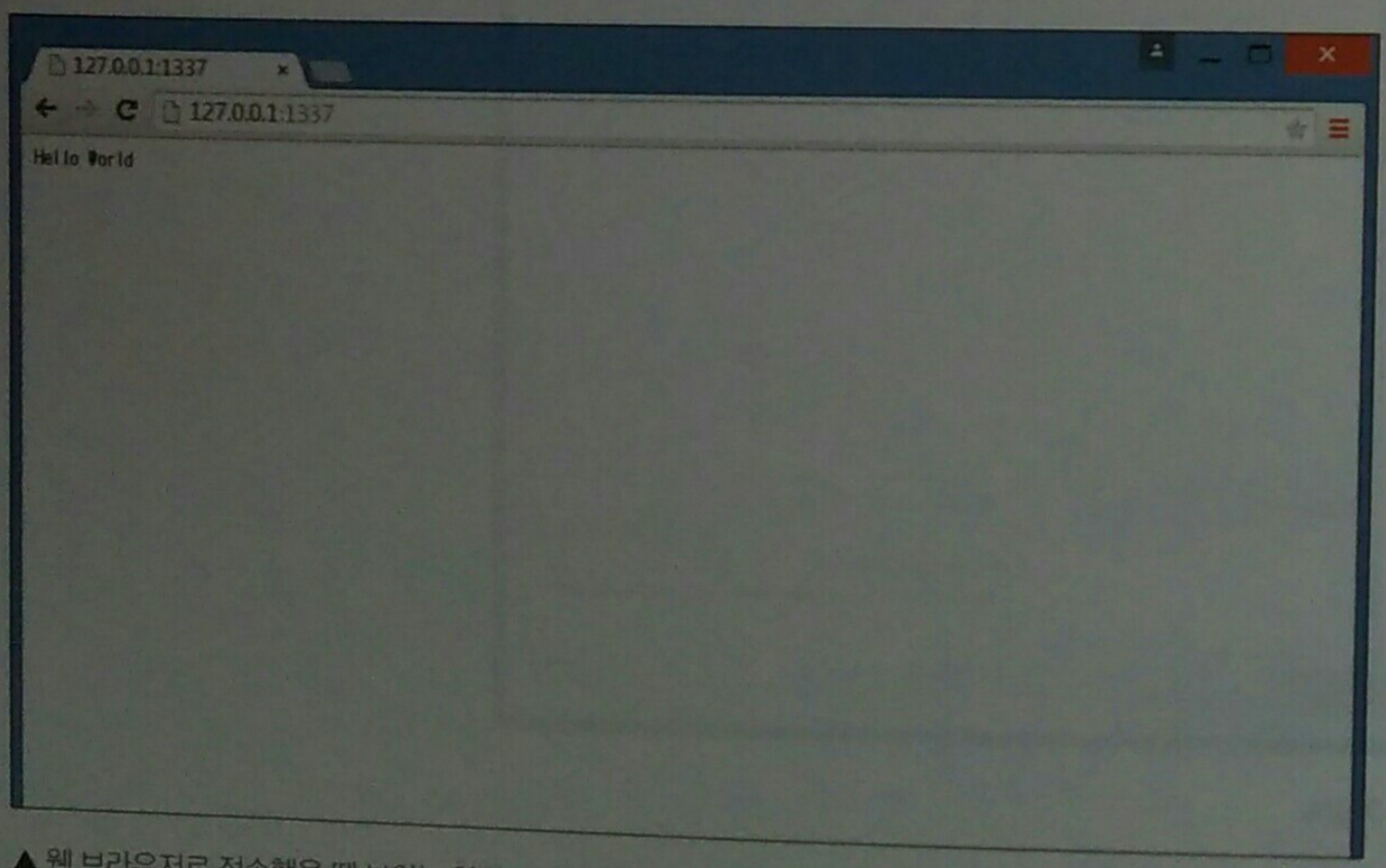
▲ 새로 만들어진 프로젝트에 들어 있는 파일들

❻ 프로젝트 안에 있는 파일 중에서 hello-world-server.js 파일의 확장자가 js인 것을 보면 자바스크립트 코드가 들어 있는 파일이라는 것을 짐작할 수 있습니다. 이 파일을 선택한 후 마우스 오른쪽 버튼을 클릭하면 [Run As] 메뉴가 표시됩니다. [Run As → 1 Node Application] 메뉴를 누르면 파일이 실행되면서 아래의 콘솔 창에 로그가 출력됩니다.



▲ 자동으로 만들어진 자바스크립트 파일 실행하기

❼ 아래쪽에 있는 콘솔(Console) 창에 Server running at http://127.0.0.1:1337이라고 표시되는 것을 볼 수 있습니다. 이제 웹 브라우저를 열고 주소창에 이 웹 사이트 주소(http://127.0.0.1:1337)를 입력합니다. 웹 브라우저는 되도록 크롬 브라우저를 사용하기를 권장합니다.



▲ 웹 브라우저로 접속했을 때 보이는 화면

정박사의 한마디 왜 크롬 브라우저를 권장할까요?

크롬 브라우저는 최신의 웹 표준을 가장 잘 준수하는 브라우저입니다. 그리고 빠르고 안정적으로 웹 사이트를 보여주기 때문에 많은 사람들이 사용하고 있습니다. 반면에 윈도우의 인터넷 익스플로러는 더 이상 사용되지 않을뿐더러, 윈도우의 에지(Edge) 브라우저는 아직까지 사용자가 그렇게 많지 않아 크롬 브라우저 사용을 권장하는 것입니다.

정박사의 한마디 크롬 브라우저를 설치하는 방법은 쉬워요

크롬 브라우저는 구글 사이트(http://www.google.co.kr)에 접속하여 설치 버튼을 눌러 바로 설치할 수 있습니다. 만약 설치 버튼이 보이지 않는다면 구글 검색창에서 Chrome을 검색하여 설치 사이트를 찾아 설치하면 됩니다.

❽ 어떻게 Hello World가 출력된 화면이 보이는 걸까요? 여러분은 코드를 한 번도 입력한 적이 없지만 새로 만든 프로젝트 안에 자동으로 만들어진 자바스크립트 파일이 실행되면서 웹 서버가 동작한 것입니다. 이 웹 서버가 웹 브라우저로부터 요청을 받아 처리한 결과를 응답으로 보내 준 후 웹 브라우저가 화면에 표시한 것이죠. 한번 실행된 서버는 중지시킬 때까지 계속 실행됩니다. 다음번에 다시 실행할 수도 있기 때문에 일단 콘솔 창의 위쪽에 있는 빨간색 네모 버튼을 눌러 서버를 중지합니다.

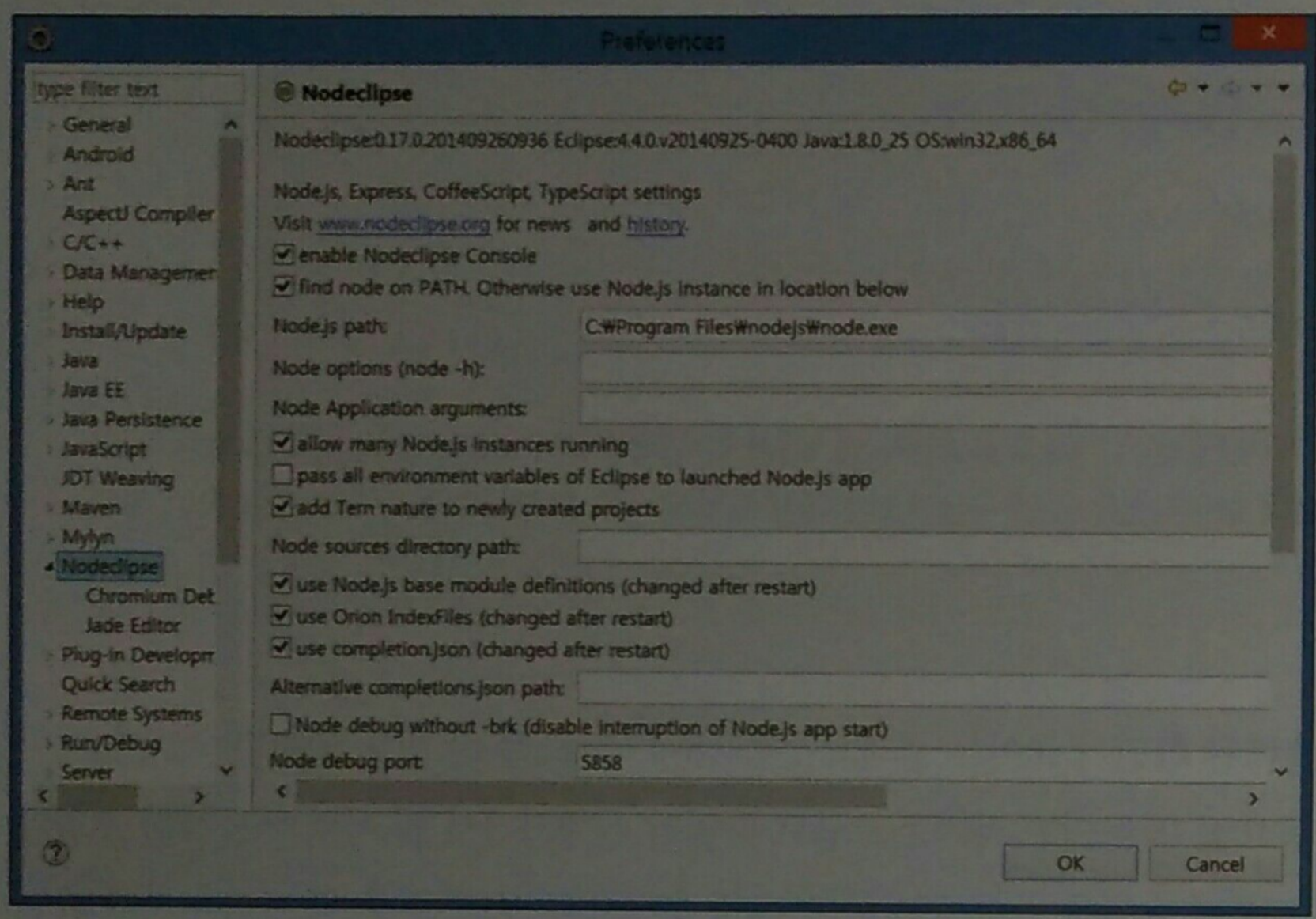
이제 hello-world-server.js 파일을 더블클릭해서 열어 보면 다음과 같은 자바스크립트 코드가 만들어져 있는 것을 확인할 수 있습니다.

참조 파일 | NodeExample1>hello-world-server.js

```
var http = require('http');
http.createServer(function handler(req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello World\n');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

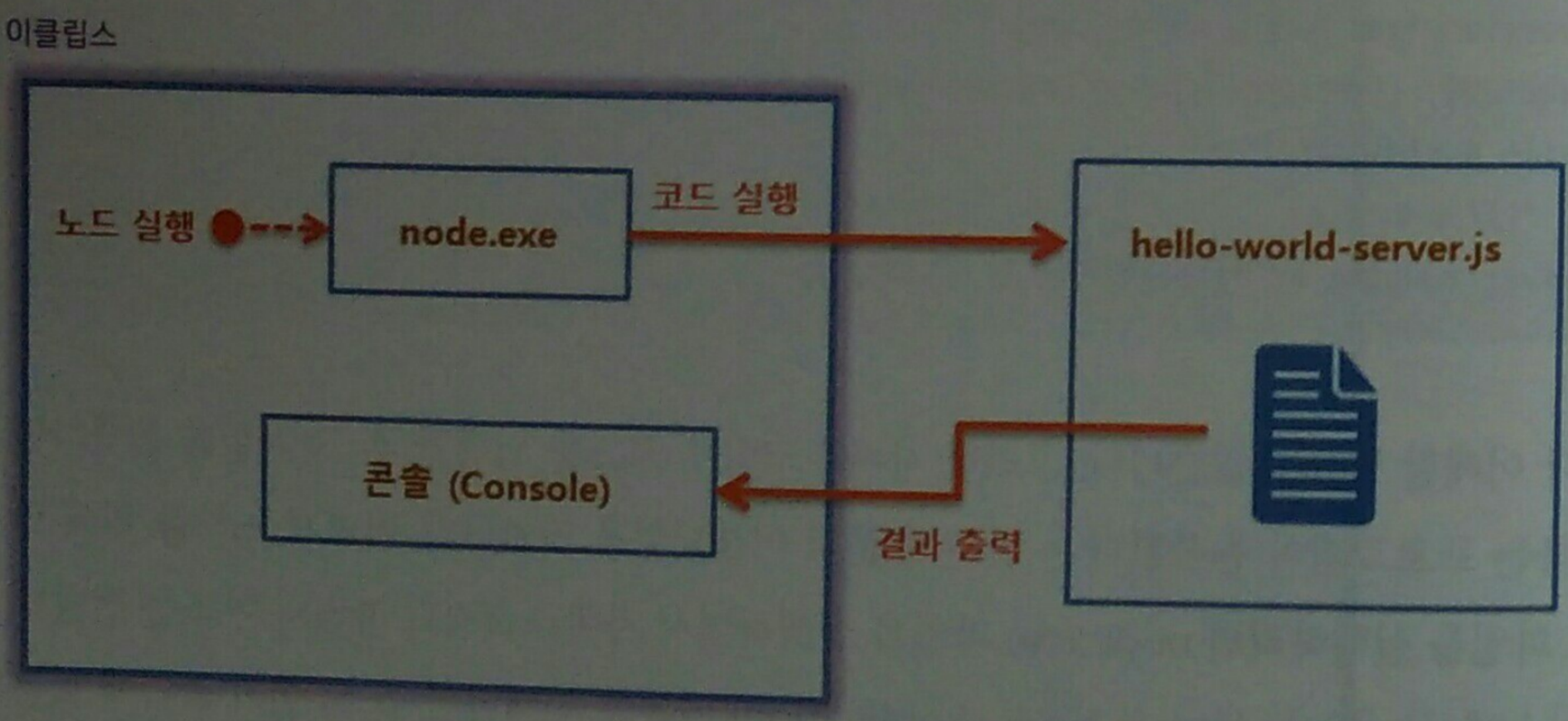
지금 이 코드를 다 이해할 필요는 없습니다. 그러나 자바스크립트 코드를 입력한 후 그 파일을 실행하면 웹 서버 기능을 하는 프로그램이 동작한다는 것은 충분히 이해되었을 것입니다. 이렇게 노드를 위해 만든 자바스크립트 파일을 실행하려면 node.exe 파일을 실행하면서 자바스크립트 파일의 이름을 전달해야 합니다. 이클립스에서는 [Run As → 1 Node Application] 메뉴를 선택하면 노드 설치 폴더에 있는 이 파일을 쉽게 실행할 수 있도록 도와줍니다.

❾ 그러면 이클립스에 설정된 노드 정보를 한번 확인해 보겠습니다. 이클립스 위쪽의 [Window → Preferences] 메뉴를 누른 후 [Preferences] 대화상자의 왼쪽 메뉴에서 [Nodeclipse]를 선택하면 다음과 같은 화면이 나타납니다.



▲ 이클립스의 노드클립스 설정 화면

앞 장에서 설치한 이클립스의 이니드 스튜디오(Enide Studio) 플러그인은 그 내부에서 **노드클립스**(Nodeclipse)라는 이름으로 표시됩니다. 이 플러그인은 노드를 이용해 프로그램을 좀 더 쉽게 만들 수 있도록 노드 프로젝트를 자동으로 만들어 주며, node.exe 파일을 사용해서 자바스크립트 파일을 실행할 수 있습니다. 화면에서 Node.js path 항목을 보면 노드 설치 폴더가 보이고 그 안에 있는 node.exe 파일이 지정되어 있습니다. node.exe 파일을 실행하면 노드 실행 환경이 되고, 자바스크립트 파일의 코드가 한 줄씩 해석되어 실행된 후 그 결과가 콘솔 창에 출력되는 것입니다.



▲ 노드 프로그램의 실행 과정

이렇게 노드 프로그램의 실행 과정을 살펴보면 자바스크립트 파일 안에 들어 있는 코드가 어떤 과정을 거쳐 실행되는지 좀 더 쉽게 이해할 수 있습니다. 다시 정리하면 이클립스 내부에서는 node.exe 파일을 실행할 수 있으며, 이 파일이 실행되면서 hello-world-server.js 파일에 있는 코드를 읽어와 해석하여 실행합니다. 실행된 프로그램의 결과는 콘솔에 출력되어 사용자가 볼 수 있습니다.

지금까지 첫 번째 노드 프로젝트를 만들어 보았습니다. 프로젝트를 만들면 바로 웹 서버가 만들어지고 웹 브라우저에서 그 내용을 요청하여 볼 수 있습니다. 제대로 된 웹 사이트를 만들기 위해서는 많은 작업 과정이 필요하지만, 몇 번의 클릭만으로도 웹 서버로 동작하는 프로그램을 만들어 실행할 수 있습니다. 그리고 그 코드가 많은 사람들이 쉬운 프로그래밍 언어라고 생각하는 자바스크립트입니다. 이제 여러분도 이 책의 내용을 차근차근 따라하다 보면 웹 서버를 직접 만들 수 있을 것입니다.

02-2 | 콘솔에 로그 뿌리기

첫 번째 프로젝트를 만든 후 그 안에 들어 있는 자바스크립트 파일을 실행하는 일련의 과정은 이클립스 없이 명령 프롬프트에서도 진행할 수 있습니다. 명령을 사용해 자바스크립트 파일이 실행되는 과정을 살펴보면 이클립스 내부에서 동작하는 방식을 좀 더 쉽게 이해할 수 있습니다. 그러면 명령 프롬프트를 열어 앞에서 알아본 자바스크립트 파일을 실행해 보겠습니다.

명령 프롬프트 열고 자바스크립트 파일 실행하기

윈도우에서 [실행]을 눌러 cmd를 입력해서 윈도우에서 명령 프롬프트를 실행하면 명령을 실행할 수 있는 창이 뜨며 사용자 폴더가 기본 위치로 지정됩니다. 사용자 폴더 아래에는 이클립스를 시작할 때 만들어진 [workspace] 폴더가 있고 그 안에는 [NodeExample1] 프로젝트 폴더가 있습니다. cd 명령을 사용하여 [NodeExample1] 폴더로 이동합니다.

명령 프롬프트에 입력

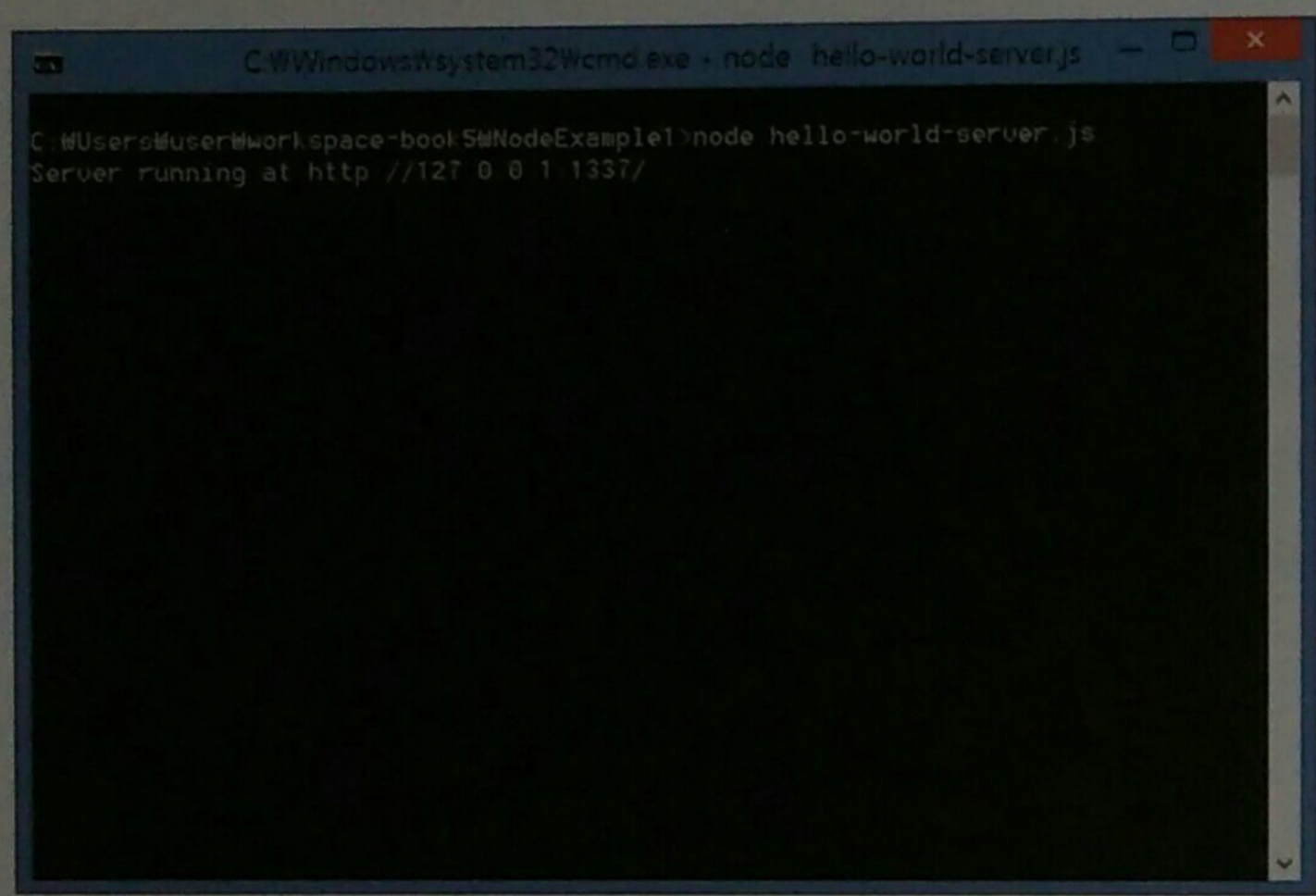
```
% cd workspace₩NodeExample1
```

주의! 명령 프롬프트 창에 입력하는 코드 중에서 가장 앞에 있는 % 기호는 입력하지 않아야 합니다. %는 명령 프롬프트의 입력창을 나타내는 기호로 표시한 것이기 때문입니다. 다시 말해서 % 기호는 명령 프롬프트에서 명령을 입력하기 위해 대기하는 기호를 단순하게 표시한 것입니다.

다음 명령을 입력하고 Enter 키를 누르면 이클립스에서 실행했던 것과 똑같은 메시지인 웹 사이트 주소(http://127.0.0.1:1337)가 표시되면서 프로그램이 실행됩니다.

명령 프롬프트에 입력

```
% node hello-world-server.js
```



▲ 명령 프롬프트에서 자바스크립트 파일을 노드 실행 파일로 실행하기

이렇게 명령 프롬프트에서 **노드 실행 파일**(node.exe)을 실행해 보면 자바스크립트 파일을 실행할 때 항상 node 프로그램이 사용된다는 것을 알 수 있습니다. node 뒤에는 공백을 하나 넣은 후 대상이 되는 자바스크립트 파일 이름을 지정해야 한다는 것을 다시 한 번 기억하기 바랍니다.

명령 프롬프트에 직접 코드 입력해서 실행하기

이번에는 코드를 직접 한 부분씩 입력하면서 실행해 보겠습니다. 먼저 키보드에서 Ctrl 키와 함께 C 키를 누르면 실행했던 프로그램이 종료됩니다. 그다음에는 명령 프롬프트 창에 node만 입력하고 Enter 키를 눌러 실행하면 코드를 바로바로 입력하여 실행할 수 있습니다.

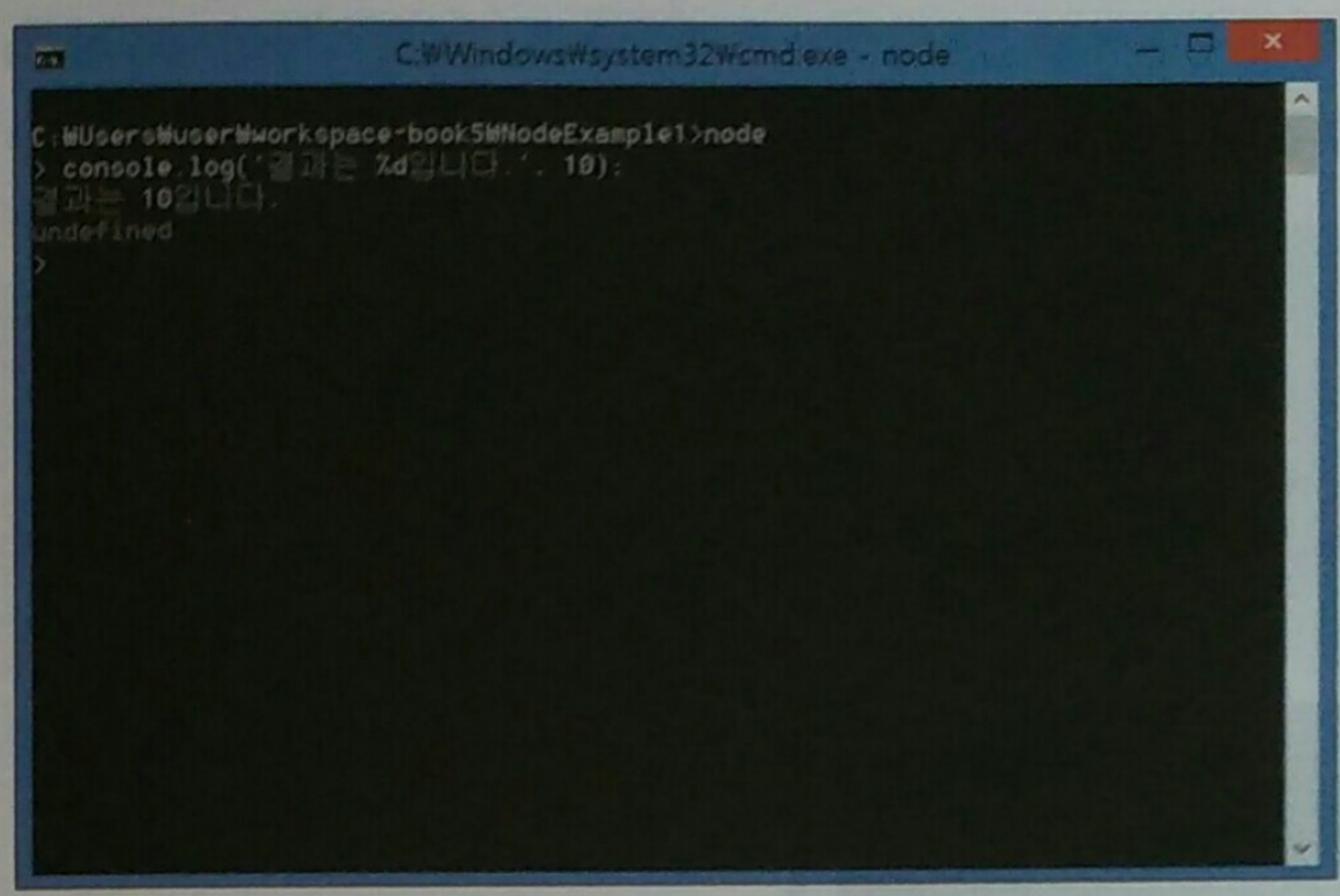
명령 프롬프트에 입력

```
% node
```

다음과 같은 코드를 입력하고 실행하면 화면에 문자열이 표시됩니다.

명령 프롬프트에 입력

```
% console.log('결과는 %d입니다.', 10);
```



▲ 명령 프롬프트에서 자바스크립트 코드를 바로 실행하기

여기에서 console 객체는 **전역 객체(Global Object)**라고 부르며 필요할 때 코드의 어느 부분에서나 사용할 수 있습니다.

정박사의 한마디 전역 객체는 언제 어디에서나 사용할 수 있어요

자바스크립트에서는 다른 언어들처럼 함수를 만들 수 있고 함수 안에서 변수를 선언하여 사용할 수 있습니다. 그런데 함수 안에서 선언한 변수는 그 함수 안에서만 유효합니다. 즉, 그 함수 안에서만 사용할 수 있습니다. 이와 달리 전역 객체는 함수 안과 밖에서 모두 사용할 수 있도록 범위를 제한하지 않는 객체입니다. 따라서 코드의 어느 부분에서나 사용할 수 있습니다.

대표적인 전역 객체들은 여러분이 프로그램을 만들면서 필요할 때마다 사용할 수 있어야 하므로 간단하게 그 기능을 알아보겠습니다. 노드에서 사용할 수 있는 대표적인 전역 객체들은 다음과 같습니다.

전역 객체 이름	설명
console	콘솔 창에 결과를 보여주는 객체
process	프로세스의 실행에 대한 정보를 다루는 객체
exports	모듈을 다루는 객체

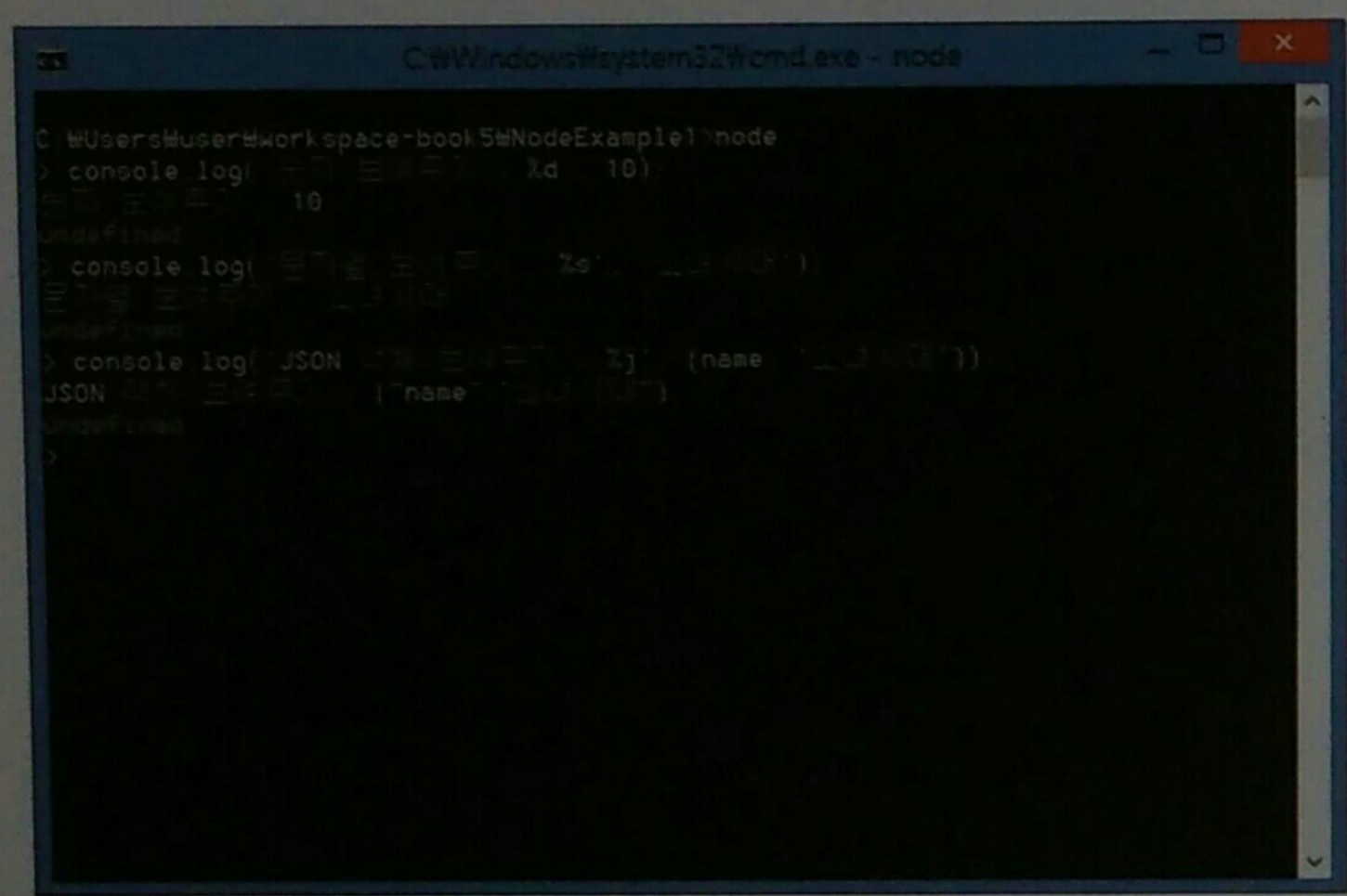
[표] 자바스크립트의 전역 객체 자료형

전역 객체들 중에서 console 객체는 가장 많이 쓰이는 것 중 하나입니다. console 객체는 콘솔 화면에 결과를 보여주기 위해 어디에서나 사용할 수 있으며 console 객체에 정의된 log( ) 메소드를 호출하면서 문자열을 파라미터(매개변수)로 전달하면 그대로 출력됩니다. 이때 %d, %s, %j와 같은 특수문자를 같이

전달하면 그 뒤에 오는 파라미터를 각각 숫자, 문자열, JSON 객체로 인식한 후 문자열로 변환시켜 화면에 보여 줍니다. 다음과 같은 코드를 하나씩 입력하여 간단하게 화면에 출력해 봅니다.

명령 프롬프트에 입력

```
% console.log('숫자 보여주기 : %d', 10);
% console.log('문자열 보여주기 : %s', '소녀시대');
% console.log('JSON 객체 보여주기 : %j', {name: '소녀시대'});
```



▲ console 객체의 log( ) 메소드로 화면에 출력하기

각각의 코드를 실행했을 때 함께 출력되는 undefined는 호출한 log( ) 함수에서 반환하는 값이 없을 때 나타납니다. 자바스크립트의 함수는 항상 값을 반환하도록 되어 있는데, 함수를 만들 때 값을 반환하는 코드를 넣지 않으면 undefined가 반환됩니다. undefined는 정의되어 있지 않은 값이라는 말인데, 비어 있는 값이라는 의미의 null 값도 있습니다. 그러면 undefined와 null은 어떤 차이가 있을까요? undefined는 단순히 값이 존재하지 않는다고 생각하면 되고, null은 의도적으로 값을 비웠다고 생각하면 이해하기 쉽습니다.

정박사의 한마디 JSON 포맷에 대해 잘 알고 있나요?

JSON 포맷은 자바스크립트의 객체 포맷으로 단말끼리 데이터를 주고받을 때 많이 사용합니다. 따라서 어떤 형식으로 만드는지를 잘 알아 두는 것이 좋습니다.

JSON 포맷은 자바스크립트에서 객체를 만들 때 사용하는 형식과 같습니다. 즉 중괄호 { }를 이용해 객체를 만들 수 있으며, 그 안에 키(Key)와 값(Value)으로 구성된 속성들은 콤마(,)로 구분합니다. 예를 들어, 사람 이름과 전화번호를 속성으로 가지고 있는 객체를 만든다면 다음처럼 JSON 포맷으로 표현할 수 있습니다.

```
{"name":"소녀시대","age":20}
```

console 객체에는 자바스크립트 객체의 속성들을 한꺼번에 출력해 주는 dir( ) 메소드, 그리고 코드 실행 시간을 측정할 수 있는 time( )과 timeEnd( ) 메소드 등이 들어 있습니다.

메소드 이름	설명
dir(object)	자바스크립트 객체의 속성들을 출력합니다.
time(id)	실행 시간을 측정하기 위한 시작 시간을 기록합니다.
timeEnd(id)	실행 시간을 측정하기 위한 끝 시간을 기록합니다.

[표] console 객체의 주요 메소드

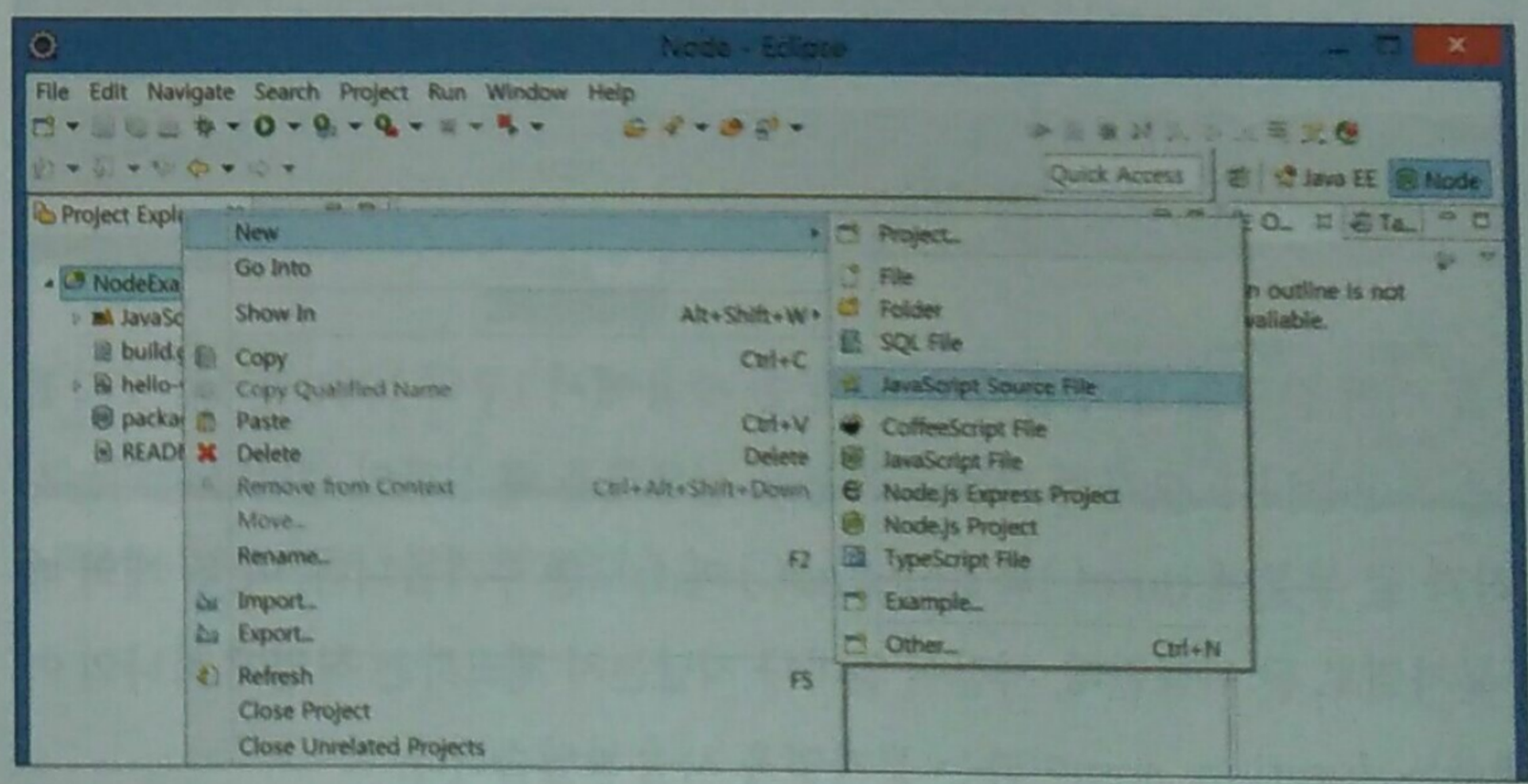
정박사의 한마디 자바스크립트에서 사용하는 '함수'라는 말과 '메소드'라는 말은 같은 건가요?

함수(Function)라는 말은 프로그래밍 언어에서 공통으로 사용합니다. 예를 들어, 더하기 기능을 가진 함수 상자를 만들고 그 상자에 두 개의 공을 넣으면 아래쪽으로 하나의 공이 나온다고 정의할 수 있습니다. 이런 기능을 가진 하나의 단위를 함수라고 합니다. 그런데 객체 지향 언어에서는 이 함수를 메소드(Method)라고도 부릅니다. 자바스크립트는 객체 지향 언어의 특징인 객체(Object)를 만들 수 있는데, 객체를 만들었을 때 그 안에 정의한 함수는 특별히 메소드라고 부릅니다. 그러나 이름을 어떻게 부르든 의미 전달에는 큰 차이가 없기 때문에 이 책에서는 함수와 메소드를 정확히 구별하지 않고 사용합니다.

이클립스 프로젝트 안에 자바스크립트 파일 만들고 실행하기

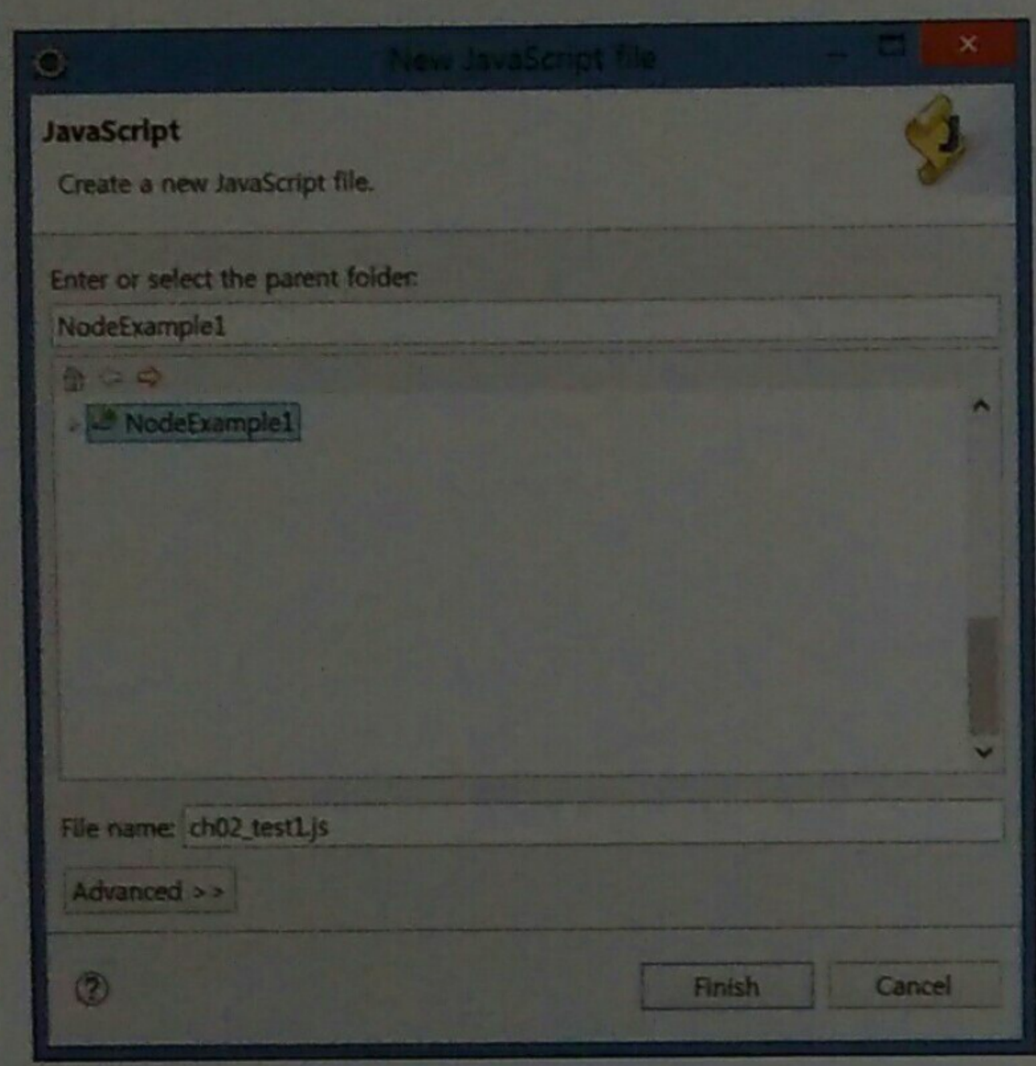
이번에는 이클립스의 프로젝트 안에 새로운 자바스크립트 파일을 만들고 간단한 코드 몇 줄을 입력한 후 실행해 보겠습니다.

❶ 왼쪽 프로젝트 영역 창에서 [NodeExample1] 프로젝트를 선택한 후 마우스 오른쪽 버튼을 누릅니다. 메뉴가 나타나면 [New → JavaScript Source File] 메뉴를 선택합니다.



▲ 프로젝트에 새로운 자바스크립트 파일을 추가하는 메뉴

❷ 새로운 자바스크립트 파일을 만드는 대화상자가 나타나면 File name: 입력 상자에 ch02_test1.js라고 입력한 후 [Finish] 버튼을 누릅니다.



▲ 새로운 자바스크립트 파일을 만들기 위한 대화상자

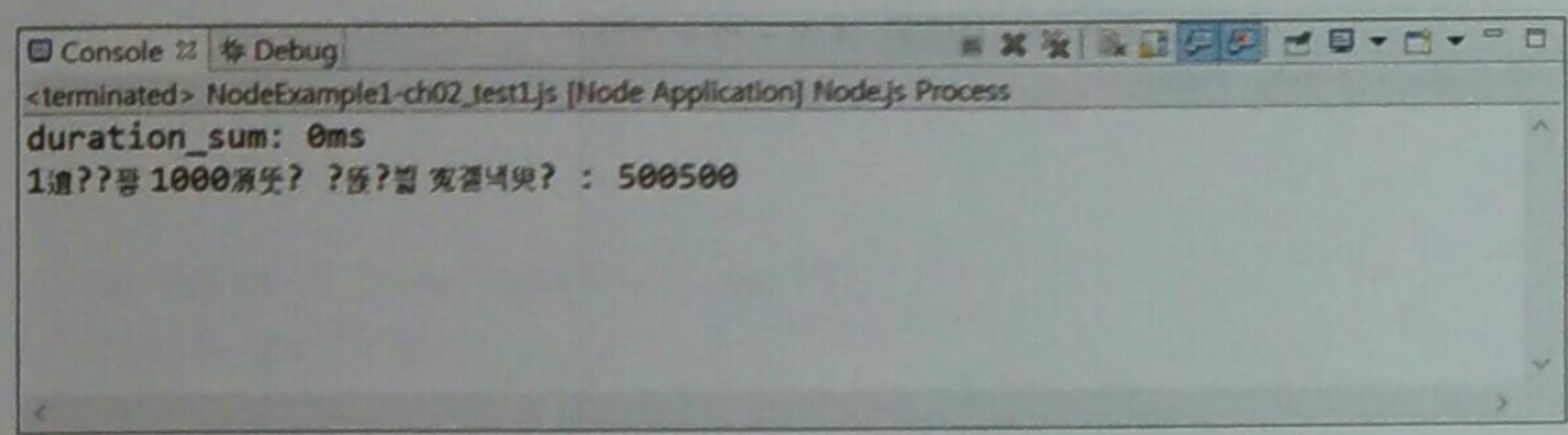
❸ 새로운 ch02_test1.js 파일이 만들어지면 이클립스 화면 가운데 있는 작업 영역에 새로운 탭이 나타납니다. 그 탭에는 ch02_test1.js라는 이름이 표시됩니다. 탭 안의 코드 입력 창에 다음 코드를 입력합니다.

참조 파일 | NodeExample1>ch02_test1.js

```
var result = 0;

console.time('duration_sum');

for (var i = 1; i <= 1000; i++) {
        result += i;
}

console.timeEnd('duration_sum');
console.log('1 부터 1000 까지 더한 결과물 : %d', result);
```

코드는 1부터 1000까지 숫자를 더한 결과를 알려 줍니다. for문을 사용해서 1부터 1000까지의 숫자를 더하면 result 변수에 결과 값을 저장합니다. 숫자를 더하는 기능을 실행했을 때 시간이 얼마나 걸리는지 알고 싶다면, 해당 코드의 시작과 끝 부분에 time( )과 timeEnd( ) 메소드를 추가합니다. 이 두 개의 메소드로 전달되는 파라미터는 문자열로 된 이름인데, 시간이 얼마나 지났는지 체크하는 작업에 하나의 이름을 붙여 구별합니다. 여기에서는 duration_sum이라는 문자열을 사용하였습니다.

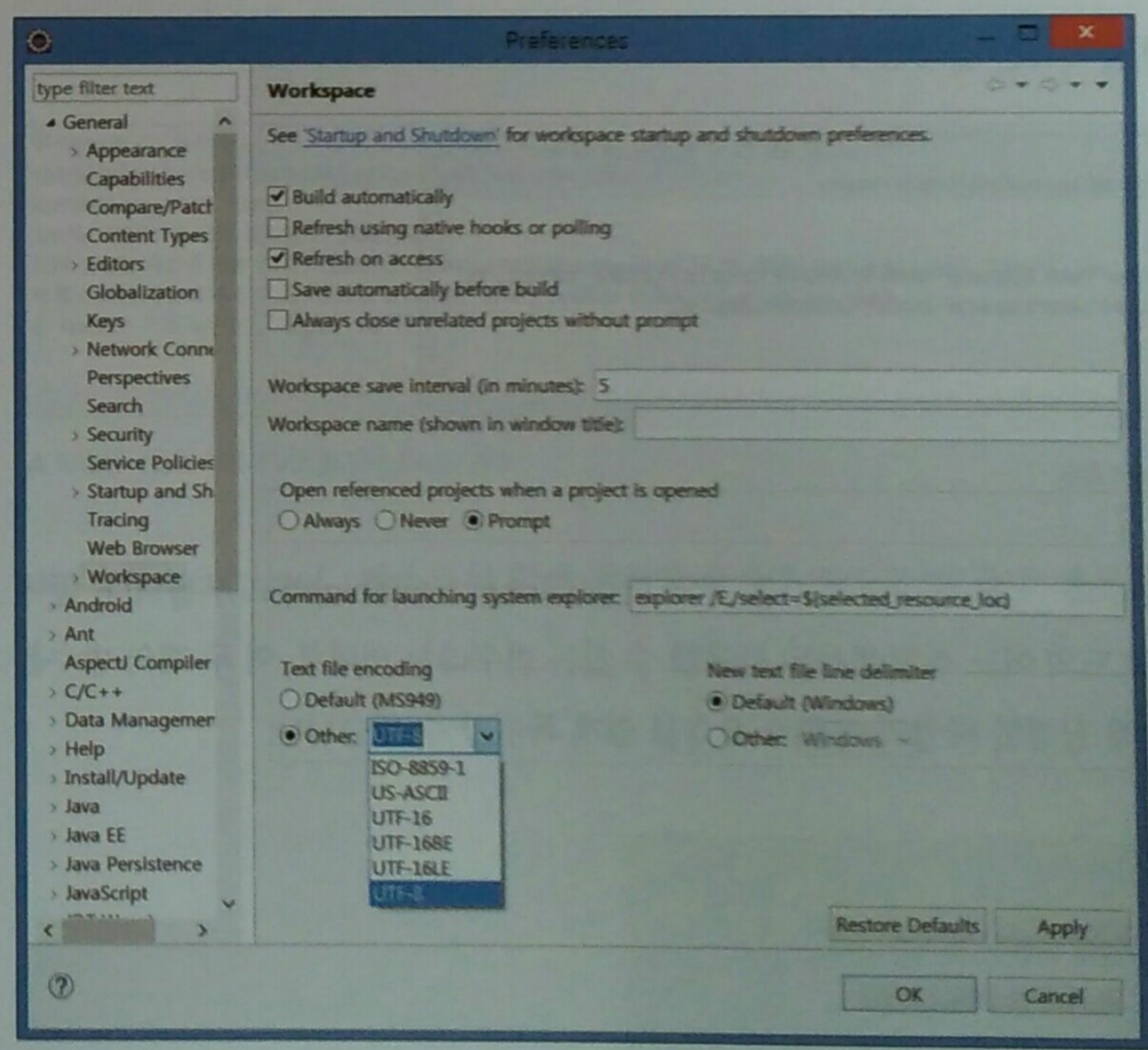
❹ 코드를 모두 입력하고 나면 Ctrl + S를 눌러서 저장합니다. 새롭게 추가한 ch02_test1.js 파일이 잘 만들어졌다면 이 파일을 선택한 후 마우스 오른쪽 버튼을 누릅니다. 나타난 메뉴에서 [Run As → 1 Node Application] 메뉴를 선택하여 실행하면 다음과 같은 결과가 콘솔 창에 표시됩니다.



▲ 콘솔에 표시된 프로그램 실행 결과

주의! ch02_test1.js에 코드를 추가한 후 저장해야 합니다. 만약 파일을 다시 저장하지 않고 실행하면 저장 전의 상태로 인식되어 컴파일되므로 원하는 결과가 나오지 않을 수 있습니다.

❺ 이때 콘솔 창에 표시된 한글이 깨져 보이는 경우가 있습니다. 이것은 소스에 입력한 글자의 한글 인코딩과 노드에서 자바스크립트 코드를 실행하여 보여 줄 때의 한글 인코딩이 달라서 발생하는 문제입니다. 다음과 같이 이클립스의 [Window → Preferences] 메뉴를 누른 다음 설정 화면이 보이면 왼쪽의 [General → Workspace] 메뉴를 누릅니다. 그런 다음 Text file encoding 항목에서 Other를 체크하고 콤보 박스에서 'UTF-8'을 선택하여 인코딩을 바꿔 줍니다.



▲ 이클립스의 설정 화면에서 한글 인코딩 바꾸기

❻ 콤보박스에서 UTF-8을 선택했으면 [OK] 버튼을 누릅니다. 이제 다시 ch02_test1.js 파일을 선택한 후 [Run As → 1 Node Application] 메뉴를 실행하면 다음과 같이 한글이 정상적으로 표시되는 것을 볼 수 있습니다.

```
Console  Debug
<terminated> NodeExample1-ch02_test1.js [Node Application] Node.js Process
duration_sum: 0ms
1부터 1000까지 더한 결과물 : 500500
```

▲ 콘솔에 한글이 정상적으로 표시되는 경우

참조 파일 | NodeExample1>ch02_test1.js

```
……

console.log('현재 실행한 파일의 이름 : %s', __filename);
console.log('현재 실행한 파일의 패스 : %s', __dirname);
```

다음은 파일을 실행했을 때 출력되는 결과입니다.

```
Console  Debug
<terminated> NodeExample1-ch02_test1.js [Node Application] Node.js Process
duration_sum: 0ms
1부터 1000까지 더한 결과물 : 500500
현재 실행한 파일의 이름 : C:\Users\user\workspace-book5\NodeExample1\ch02_test1.js
현재 실행한 파일의 패스 : C:\Users\user\workspace-book5\NodeExample1
```

▲ 실행한 파일의 이름과 패스를 같이 표시한 경우

여기에서 사용한 __filename과 __dirname 변수를 **전역 변수**(Global Variable)라고 부릅니다. 앞에서 설명했지만 전역 변수는 코드의 어느 부분에서든 사용할 수 있는 변수라는 의미로, 이 두 개의 변수를 사용하면 node.exe를 사용해 실행한 파일의 이름과 패스를 쉽게 확인할 수 있습니다.

변수 이름	설명
__filename	실행한 파일의 이름을 출력합니다. 파일의 전체 패스가 출력됩니다.
__dirname	실행한 파일이 들어 있는 폴더를 출력합니다. 폴더의 전체 패스가 출력됩니다.

[표] 실행한 파일 정보를 알려 주는 전역 변수

❼ 마지막으로 console 객체의 dir( ) 메소드를 이용해 객체 안에 들어 있는 속성들을 확인해 보겠습니다. ch02_test1.js 파일의 마지막 부분에 다음 코드를 추가로 입력한 후 저장합니다.

참조 파일 | NodeExample1>ch02_test1.js

```
……

var Person = {name:"소녀시대", age:20};
console.dir(Person);
```

자바스크립트의 객체는 중괄호 { }를 사용해서 만들 수 있습니다. 중괄호 안에는 콤마(,)로 구분된 속성들이 들어가며, 이 속성은 속성 이름과 속성 값 사이에 콜론(:)을 넣어서 구분합니다. 즉, 다음과 같은 코드 형식을 사용합니다.

```
{ 속성 이름1 : 속성 값1 , 속성 이름2 : 속성 값2, ……. }
```

이렇게 만들어진 객체는 Person 변수에 들어가는데, dir( ) 메소드를 호출하면 그 객체 안에 들어 있는 모든 속성이 콘솔에 출력됩니다. 파일을 실행하면 다음과 같이 출력됩니다.

```
Console  Debug
<terminated> NodeExample1-ch02_test1.js [Node Application] Node.js Process
duration_sum: 0ms
1부터 1000까지 더한 결과물 : 500500
현재 실행한 파일의 이름 : C:\Users\user\workspace-book5\NodeExample1\ch02_test1.js
현재 실행한 파일의 패스 : C:\Users\user\workspace-book5\NodeExample1
{ name: '소녀시대', age: 20 }
```

▲ 실행한 파일의 이름과 패스를 같이 표시한 경우

dir( ) 메소드는 log( ) 메소드와 함께 객체의 속성을 확인할 때 자주 사용합니다.

02-3 | 프로세스 객체 간단하게 살펴보기

process 객체는 프로그램을 실행했을 때 만들어지는 프로세스 정보를 다루는 객체입니다. process 객체의 주요 속성과 메소드는 다음과 같습니다.

속성/메소드 이름	설명
argv	프로세스를 실행할 때 전달되는 파라미터(매개변수) 정보
env	환경 변수 정보
exit( )	프로세스를 끝내는 메소드

[표] process 객체의 주요 속성과 메소드

argv 속성은 프로세스를 실행할 때 전달되는 파라미터 정보를 가지고 있습니다. 이 정보를 확인하기 위해 새로운 자바스크립트 파일을 추가해 보겠습니다. 자바스크립트 파일을 추가하는 방법은 잊지 않으셨죠? 왼쪽 목록 창에서 [NodeExample1] 프로젝트를 선택한 후 마우스 오른쪽 버튼을 눌러 [New → JavaScript Source File] 메뉴를 선택합니다. ch02_test2.js라는 새로운 자바스크립트 파일을 만드는 대화상자가 나타나면 File name: 입력 상자에 ch02_test2.js라고 입력한 후 [Finish] 버튼을 누릅니다.

새로운 창이 생기면 다음 코드를 입력합니다.

참조 파일 | NodeExample1>ch02_test2.js

```
console.log('argv 속성의 파라미터 수 : ' + process.argv.length);
console.dir(process.argv);
```

이 파일을 실행하면 다음과 같이 파라미터의 개수와 그 값들을 볼 수 있습니다.

```
Console  Debug
<terminated> NodeExample1-ch02_test2.js [Node Application] Node.js Process
argv 속성의 파라미터 수 : 2
[ 'node',
  'C:\\Users\\user\\workspace-book5\\NodeExample1\\ch02_test2.js' ]
```

▲ process.argv 속성의 값 확인하기

자바스크립트 파일을 실행하기 위해 사용한 node.exe 파일의 이름이 첫 번째 파라미터가 되고 자바스크립트 파일의 패스가 두 번째 파라미터가 됩니다. 만약 파일을 실행할 때 파라미터를 더 넣어 주면 이 argv 속성에 그 파라미터가 추가됩니다. 자바스크립트에서는 여러 개의 값이 들어갈 수 있는 배열 객체를 [ ] 기호를 사용해 만들 수 있으며, 로그로 출력했을 때도 [ ] 기호 안에 배열 객체의 값들을 콤마로 구분하여 표시합니다.

정박사의 한마디 첫 번째 파라미터의 값이 다르게 표시되나요?

첫 번째 파라미터의 값이 'node'가 아니라 C:\\Program Files\\nodejs\\node.exe처럼 표시될 수도 있습니다. 이것은 node.exe 실행 파일의 위치까지 함께 표시된 것입니다.

console 객체의 dir( ) 메소드를 사용하면 객체가 가지고 있는 속성을 그대로 출력할 수 있습니다. 콘솔에 표시된 결과에서 알 수 있듯이, process 객체에 들어 있는 argv 속성은 배열 객체이며 파일을 실행하기만 해도 node 명령과 파일 패스가 파라미터 값으로 들어간다는 것을 알 수 있습니다.

참조 파일 | NodeExample1>ch02_test2.js

```
.....

if (process.argv.length > 2) {
  console.log('세 번째 파라미터의 값 : %s', process.argv[2]);
}

process.argv.forEach(function(item, index) {
  console.log(index + ' : ', item);
});
```

여기에는 세 번째 파라미터의 값을 확인하는 코드가 들어 있습니다. 따라서 이 파일을 실행할 때 파라미터를 추가로 전달하면 그 값을 확인할 수 있습니다. argv 속성은 배열 객체이므로 인덱스 2를 사용하면 세 번째 파라미터의 값을 확인할 수 있습니다. 만약 argv 속성에 들어 있는 모든 값을 하나씩 출력하고 싶다면 forEach( ) 메소드를 사용합니다. forEach( ) 메소드는 배열 안에 들어 있는 각 아이템 값과 인덱스를 함께 전달하므로 배열 객체에 들어 있는 값들을 하나씩 확인하기에 좋습니다.

윈도우에서 [실행]을 눌러 cmd를 입력해서 명령 프롬프트 창을 열고 작업 공간에 만들어 둔 [NodeExample1] 폴더로 먼저 이동합니다. 명령 프롬프트에서 다음 명령을 실행하면 다음과 같은 결과가 출력됩니다.

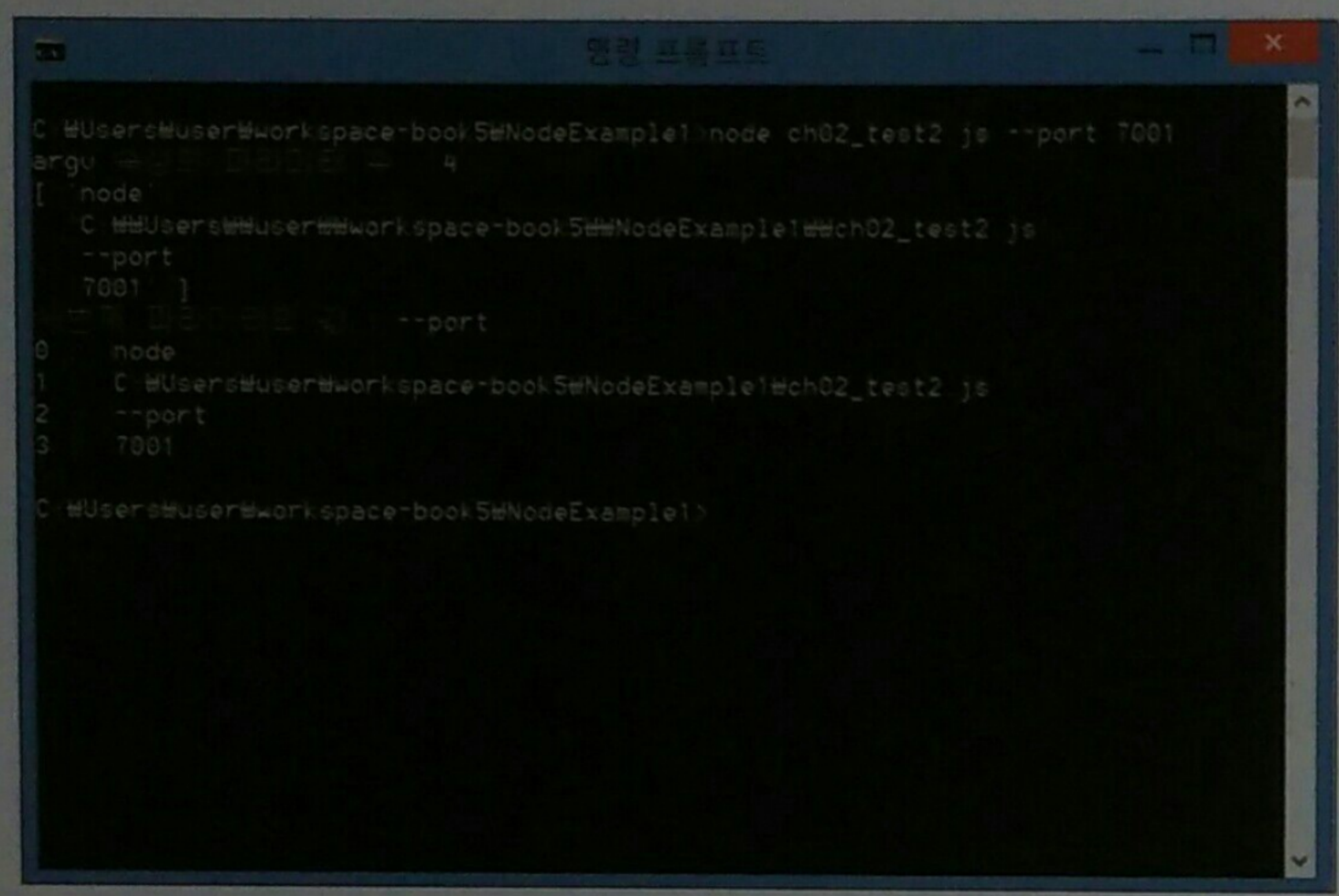
명령 프롬프트에 입력

```
% cd workspace₩NodeExample1
```

▶

명령 프롬프트에 입력

```
% node ch02_test2.js __port 7001
```



▲ process.argv 속성을 명령 프롬프트에서 확인하기

이번에는 process 객체의 env 속성을 이용해서 환경 변수의 값을 확인해 보겠습니다. 처음 노드를 설치할 때 자바를 함께 설치했으므로 JAVA_HOME **환경 변수**가 시스템에 설정되어 있을 것입니다. 이 값을 확인하기 위해 새로운 자바스크립트 소스 파일 ch02_test3.js를 만든 후 다음 코드를 입력합니다.

참조 파일 | NodeExample1>ch02_test3.js

```
console.dir(process.env);

console.log('JAVA_HOME 환경 변수의 값 : ' + process.env[JAVA_HOME]);
```

정박사의 한마디 자바를 설치한 후에 JAVA_HOME 환경 변수를 설정했나요?

JAVA_HOME 환경 변수는 설치하는 PC에 따라 설정이 필요한 경우가 있습니다. 이 때문에 되도록 자바 설치 후 JAVA_HOME 환경 변수를 설정하기를 권장합니다. OS가 윈도우8 이상인 경우에는 다음과 같이 환경 변수를 설정합니다.

(1) 컴퓨터에 설치된 자바 아이콘을 선택한 후 마우스 오른쪽 버튼으로 눌러 메뉴가 나타나면 [속성] 메뉴를 선택합니다.

(2) 컴퓨터 속성 대화상자에서 [고급 시스템 설정] 메뉴를 누릅니다. 이때 나타나는 대화상자의 [고급] 탭에서 아래쪽 [환경 변수] 버튼을 클릭합니다.

(3) 시스템 변수 부분의 [새로 만들기...] 버튼을 눌러 환경 변수의 이름에 JAVA_HOME을 입력하고 값으로는 자바가 설치된 폴더의 패스를 입력합니다.

(예: C:₩Program Files₩Java₩jdk1.8.0_25로 입력하되 폴더 이름에 있는 숫자는 최신 버전에 따라 다름)

(4) 마지막으로 [확인] 버튼을 눌러 저장하면 설정이 반영됩니다.

그런데 이 파일을 저장한 후 실행해 보면 다음과 같이 오류가 발생하는 것을 볼 수 있습니다.

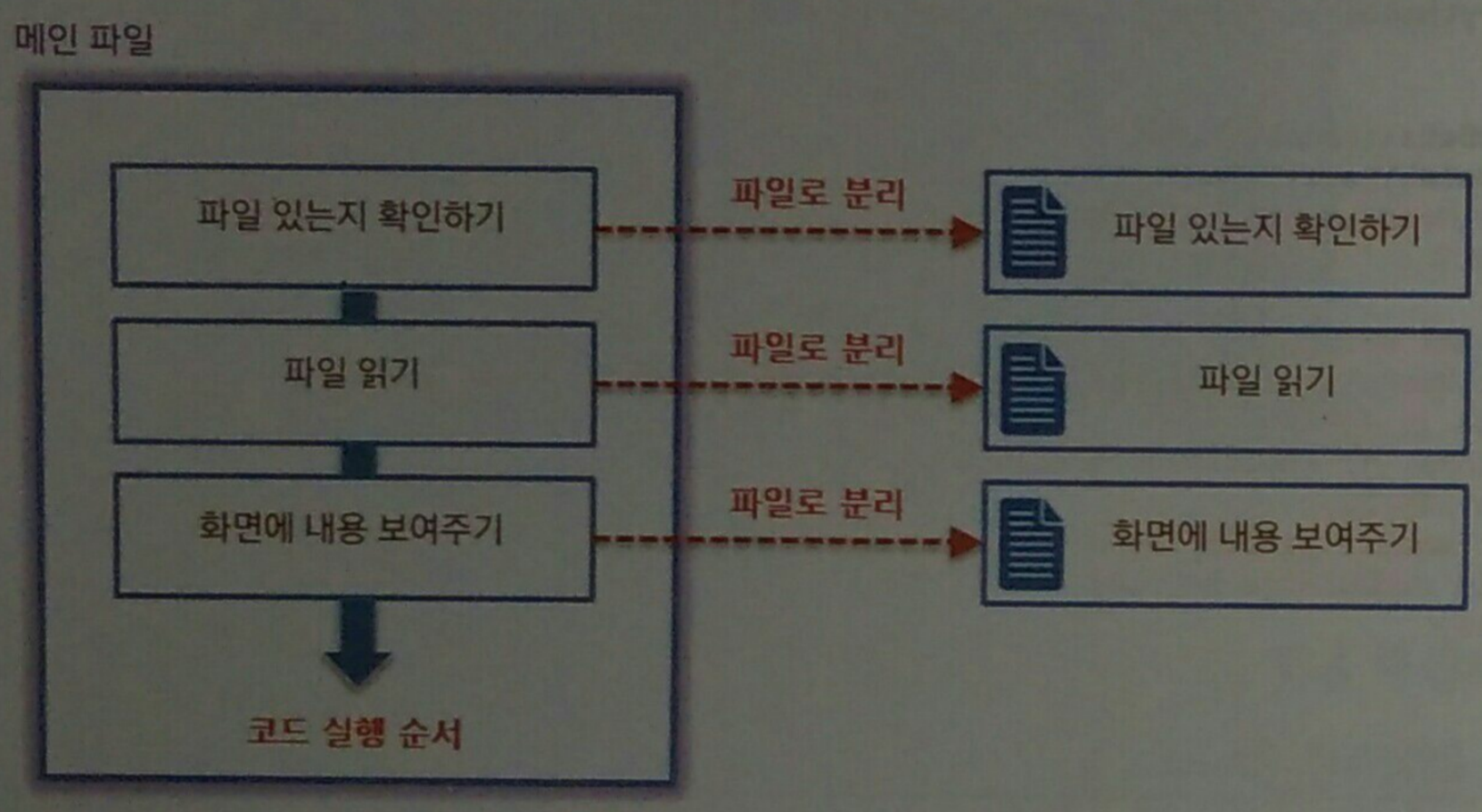
```
Console  Debug
<terminated> NodeExample1-ch02_test3.js [Node Application] Nodejs Process
{ PATH: 'C:\\Python34\\;C:\\Python34\\Scripts;C:\\app\\user\\product\\11.2
  SystemDrive: 'C:',
  SystemRoot: 'C:\\Windows',
  TEMP: 'C:\\Users\\user\\AppData\\Local\\Temp',
  TMP: 'C:\\Users\\user\\AppData\\Local\\Temp' }
C:\Users\user\workspace-book5\NodeExample1\ch02_test3.js:9
console.log('JAVA_HOME 환경변수의 값 : ' + process.env[JAVA_HOME]);
                                                        ^

ReferenceError: JAVA_HOME is not defined
    at Object.<anonymous> (C:\Users\user\workspace-book5\NodeExample1\ch02_
    at Module._compile (module.js:460:26)
    at Object.Module._extensions..js (module.js:478:10)
    at Module.load (module.js:355:32)
    at Function.Module._load (module.js:310:12)
    at Function.Module.runMain (module.js:501:10)
    at startup (node.js:129:16)
    at node.js:814:3
```

▲ process.env 속성으로 JAVA_HOME 환경 변수를 확인할 때의 오류 메시지

JAVA_HOME 변수를 시스템 환경 변수로 넣어 두었는데도 오류가 발생하는 것은 process.env 속성에는 **사용자 정의 환경 변수(user variables)**만 들어 있기 때문입니다. process.env 객체에 들어 있는 속성만으로는 JAVA_HOME과 같은 **시스템 환경 변수(system variables)**에 접근할 수 없습니다. 여기에서 사용자 정의 환경 변수란 현재 사용자 계정에만 적용되는 값을 말합니다. 그리고 시스템 환경 변수는 모든 사용자 계정에 적용되는 값입니다. 이 오류를 해결하고 시스템 변수에 접근하는 방법에 대해서는 다음 장에서 설명하는 모듈을 이해한 후 다시 알아보겠습니다.

02-4 | 노드에서 모듈 사용하기

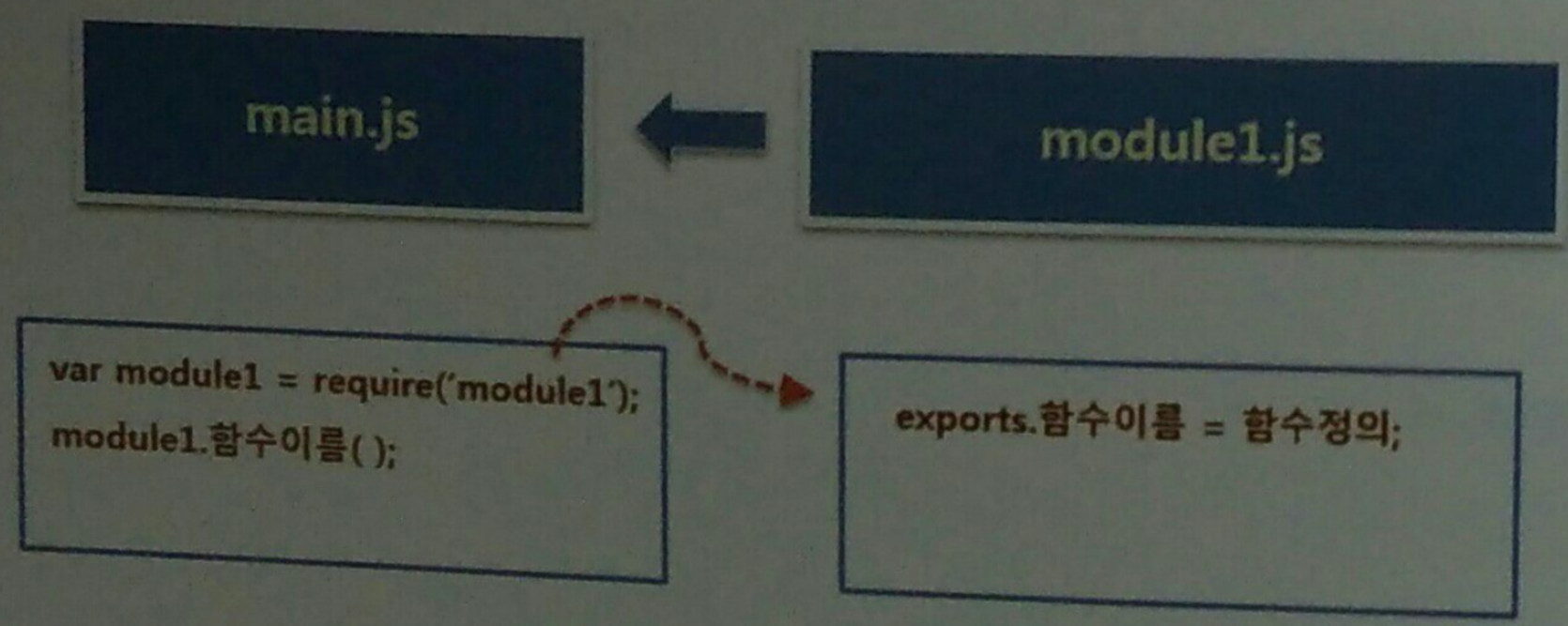
하나의 함수 안에 모든 기능을 넣는 것보다 기능별로 여러 개의 함수로 나눈 후 필요한 함수만 가져다 사용하면 다른 곳에서도 이 함수를 재사용할 수 있기 때문에 훨씬 효율적으로 프로그램을 만들 수 있습니다. 또한 각각의 기능을 나누어 프로그램 관리가 더 쉬워집니다. 코드를 하나의 파일이 아니라 여러 개의 파일로 나누어 만들 때에도 이런 장점을 그대로 살릴 수 있습니다. 다음은 파일의 내용을 확인하기 위해 만든 프로그램에서 각각의 기능을 별도의 파일로 분리한 모양을 그림으로 표현한 것입니다.



▲ 파일 내용 확인 프로그램의 코드를 각각의 파일에 나누어 넣은 그림

메인 파일의 코드 중에서 독립적인 기능은 별도 파일로 분리할 수 있으며, 메인 파일에서는 전체적인 실행 순서나 흐름만을 제어합니다. 이렇게 분리된 파일을 노드에서는 모듈이라고 부릅니다. 모듈이란 별도의 파일로 분리된 독립 기능의 모음이라서 모듈을 만들어 놓으면 다른 파일에서 모듈을 불러와 사용할 수 있습니다.

각각의 기능을 분리시킬 때는 단순히 별도의 파일에 코드를 나누어 놓는다고 끝나는 것은 아닙니다. 분리되어 있는 모듈 파일을 불러와서 사용할 수 있는 방법도 함께 만들어주어야 합니다. 노드는 CommonJs의 표준 스펙을 따라 모듈을 사용할 수 있게 합니다. 이 과정에서 exports 전역 객체를 사용합니다.



◀ 노드제이에스에서 모듈을 만들어 사용하는 대표적인 방식

모듈을 만들 때는 module1.js처럼 별도의 자바스크립트 파일을 만든 후 그 코드에서 exports 객체를 사용합니다. exports 객체의 속성으로 변수나 함수를 지정하면 그 속성을 main.js와 같은 메인 자바스크립트 파일에서 불러와 사용할 수 있습니다.

모듈을 불러올 때는 require( ) 메소드를 사용하며, 모듈로 만들어 둔 파일의 이름을 이 메소드의 파라미터로 전달합니다. require( ) 메소드를 호출하면 모듈 객체가 반환되는데, 모듈에서 exports 객체에 설정한 속성들은 이 모듈 객체를 통해 접근할 수 있습니다.

그런데 여기에서 주의할 점은 exports 외에 module.exports를 사용할 수도 있다는 것입니다. exports에는 속성을 추가할 수 있어 여러 개의 변수나 함수를 각각의 속성으로 추가할 수 있습니다. 이에 반해 module.exports에는 하나의 변수나 함수 또는 객체를 직접 할당합니다. 일반적으로는 객체를 그대로 할당하며, 이렇게 할당한 객체 안에 넣어 둔 변수나 함수를 메인 파일에서도 사용할 수 있습니다.

module1.js VS. **module2.js**

```
exports.add = function(a, b) {
    return a + b;
};

exports.multiply = function(a, b) {
    return a * b;
};
```

```
var calc = { };

calc.add = function(a, b) {
    return a + b;
};

calc.multiply = function(a, b) {
    return a * b;
};

module.exports = calc;
```

▲ 모듈 파일을 만들 때 exports와 module.exports를 사용하는 경우

정박사의 한마디 exports를 사용할지 module.exports를 사용할지 혼동되나요?

아직 자바스크립트에 익숙하지 않은데, 잘 모르는 exports 객체가 나오니 혼동되나요? 걱정할 것 없습니다. 다음 장에서 자바스크립트 기초 중에서 노드를 사용할 때 알아야 할 중요한 내용을 정리하여 설명합니다. 그때 지금까지 알고 있던 내용을 정리해 보거나 몰랐던 내용을 알아가면 됩니다. 여기에서는 'exports와 module.exports라는 것을 사용할 수 있다'는 내용, 그리고 '어떤 것을 사용하는가에 따라 메인 파일에서 모듈을 불러와 사용하는 방식이 약간 다를 수 있다'는 정도만 알아도 됩니다.

더하기 함수를 모듈로 간단히 분리하기

그러면 간단하게 더하기 기능 함수를 모듈로 분리하는 방법에 대해 알아보겠습니다. ch02_test4.js 파일을 만든 후 다음 코드를 입력합니다.

참조 파일 | NodeExample1>ch02_test4.js

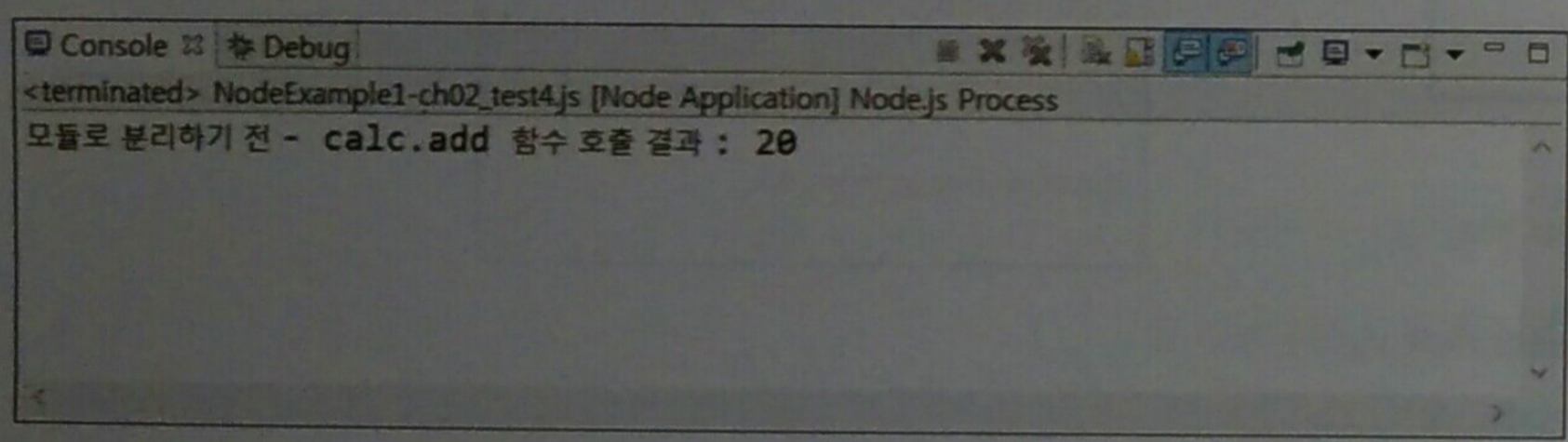
```
var calc = {};
calc.add = function(a, b) {
  return a + b;
}

console.log('모듈로 분리하기 전 - calc.add 함수 호출 결과 : %d', calc.add(10, 10));
```

calc 객체를 하나 만든 후 그 객체에 add 속성을 추가하고 더하기 함수를 할당했습니다. 이렇게 만든 함수는 calc.add( )와 같은 형태로 호출할 수 있습니다.

정박사의 한마디 자바스크립트의 함수를 호출하는 방법은 다들 알고 있겠죠?

자바스크립트에서는 함수를 정의할 때 그 앞에 function 키워드를 붙여 줍니다. 이렇게 정의한 함수는 함수 이름 뒤에 소괄호를 붙여 호출합니다. 만약 객체 안에 함수가 정의되어 있다면 객체 이름 뒤에 점(.)을 붙이고 함수 이름과 소괄호를 붙여 함수를 호출할 수 있습니다.

이 파일을 실행하면 다음과 같이 20이라는 결과가 표시됩니다.



▲ 하나의 파일 안에서 더하기 함수를 정의한 후 호출한 경우

이 코드 중에서 더하기 함수를 속성으로 추가했던 calc 객체를 별도의 파일로 분리하면 모듈 파일이 만들어집니다. 먼저 더하기 함수에 해당하는 코드를 새로 만든 파일에 넣은 후 약간 수정해야 합니다. 프로젝트에 새로운 자바스크립트 파일 calc.js를 추가한 후 다음 코드를 입력합니다.

참조 파일 | NodeExample1>calc.js

```
exports.add = function(a, b) {
  return a + b;
}
```

이렇게 하면 더하기 함수는 exports 객체에 add 속성이 추가됩니다. 메인 파일에서 이 파일을 모듈로 불러들이면 add 함수를 호출하여 사용할 수 있습니다. 이제 이 모듈 파일을 불러들여 사용하는 파일을 만들어 볼까요? ch02_test5.js 파일을 새로 만들고 다음 코드를 입력합니다.

참조 파일 | NodeExample1>ch02_test5.js

```
var calc = require('./calc');
console.log('모듈로 분리한 후 - calc.add 함수 호출 결과 : %d', calc.add(10, 10));
```

이렇게 만든 ch02_test5.js 파일은 메인 파일(Main File)이 되며, calc.js 모듈 파일을 불러와서 사용합니다. calc.js 파일을 모듈로 불러오기 위해 먼저 require( ) 함수를 호출합니다. 이때 require( ) 함수의 파라미터로 모듈 파일의 이름을 전달하는데, 파일 이름 전체를 다 사용하지 않고 확장자를 뺀 calc라는 이름만 사용합니다. 파일 이름 앞에 ./를 붙인 것을 보면 파일 이름은 상대 패스로 지정하였다는 것을 알 수 있습니다. 이렇게 불러들인 모듈은 calc 변수에 할당하였습니다.

정박사의 한마디 모듈의 이름만 지정하면 해당 파일을 찾아 불러들이지만 만약 파일이 없다면 폴더 이름을 찾아 그 안에 있는 파일을 불러들이기도 해요

require( ) 함수에 전달하는 모듈의 이름을 require('./calc')처럼 지정하면 calc.js 파일을 찾아보고 파일이 없는 경우에는 [calc] 폴더가 있는지 확인합니다. 만약 [calc] 폴더가 있다면 그 안에 있는 index.js 파일을 불러들입니다.
즉, [calc] 폴더를 만들고 그 안에 index.js 파일을 넣은 후 index.js 파일 안에 모듈 기능을 위한 자바스크립트 코드를 입력한 경우에도 require('./calc') 코드로 모듈 파일을 불러올 수 있습니다.

모듈 파일을 만들 때 사용한 exports 객체는 메인 파일에서 모듈 파일을 불러들일 때 반환되므로 이 객체를 변수에 할당하여 사용할 수 있다고 생각하면 좀 더 쉽게 이해할 수 있습니다. 즉, 여기에서는 메인 파일에 정의한 calc 변수가 모듈 파일에서 사용한 exports 객체와 같습니다.

이렇게 하면 모듈 파일에서 exports.add로 추가한 add 속성을 메인 파일의 calc 객체에서도 접근할 수 있으므로 calc.add( ) 코드로 함수를 호출할 수 있습니다.

파일을 실행하면 다음과 같은 결과를 볼 수 있습니다.

```
Console  Debug
<terminated> NodeExample1-ch02_test5.js [Node Application] Node.js Process
모듈로 분리한 후 - calc.add 함수 호출 결과 : 20
```

◀ 더하기 함수를 모듈로 분리한 후 모듈을 불러들여 호출한 경우

함수를 호출하여 사용하는 코드는 모듈로 분리하기 전과 후가 다르지 않습니다. 즉, 모듈로 분리하기 전에도 calc.add( ) 코드를 사용했고 모듈로 분리한 후에도 calc.add( ) 코드를 사용했습니다. 하지만 모듈로 분리한 후에는 별도의 자바스크립트 파일이 만들어지므로 하나의 기능을 모듈로 정의해 두면 필요에 따라 원하는 모듈만 불러들여 사용할 수 있습니다.

module.exports로 메인 파일에 더하기 함수 호출하기

이번에는 모듈 파일에서 exports가 아닌 module.exports를 사용했을 때 메인 파일에서 어떻게 더하기 함수를 호출할 수 있는지 알아보겠습니다. 프로젝트에 새로운 자바스크립트 파일 calc2.js를 추가한 후 다음 코드를 입력합니다.

참조 파일 | NodeExample1>calc2.js

```
var calc = {};

calc.add = function(a, b) {
  return a + b;
}

module.exports = calc;
```

이 파일에서는 calc.js 파일처럼 더하기 함수를 만들었지만 exports의 속성으로 만들지는 않았습니다. calc 객체를 만들고 그 객체의 속성으로 더하기 함수를 할당한 후 마지막에는 module.exports에 calc 객체를 할당했습니다. 이렇게 코드를 만들면 calc 객체는 모듈을 불러들이는 쪽에서 그대로 사용할 수 있습니다. 이제 ch02_test5.js 파일에 다음 코드를 추가합니다.

참조 파일 | NodeExample1>ch02_test5.js

```
……

var calc2 = require('./calc2');
console.log('모듈로 분리한 후 - calc2.add 함수 호출 결과 : %d', calc2.add(10, 10));
```

require( ) 함수를 호출하여 calc2.js 모듈 파일을 불러오면 이때 반환되는 객체는 모듈 파일 안에서 module.exports에 할당했던 객체가 됩니다. 따라서 calc2 변수에 그 객체가 그대로 할당되므로 calc2.add( ) 코드를 사용해 함수를 호출할 수 있습니다.

파일을 실행하면 다음과 같은 결과를 볼 수 있습니다.

```
Console  Debug
<terminated> NodeExample1-ch02_test5.js [Node Application] Node.js Process
모듈로 분리한 후 - calc.add 함수 호출 결과 : 20
모듈로 분리한 후 - calc2.add 함수 호출 결과 : 20
```

▲ module.exports를 사용한 모듈 파일을 불러와 더하기 함수를 호출한 경우

지금까지 모듈을 만들어 사용하는 방식에 대해 알아보았습니다. 이제는 앞에서 해결하지 못했던 JAVA_HOME 환경 변수 접근 문제를 모듈로 해결해 보겠습니다. 다행히 이미 다른 사람이 시스템 환경 변수를 접근할 수 있는 모듈을 만들어 두었으므로 여러분은 그 모듈을 불러와 실행하면 됩니다. 이렇게 다른 사람이 만들어 둔 모듈을 **외장 모듈**이라고 합니다.

시스템 환경 변수에 접근할 수 있는 모듈의 이름은 nconf이며, 이 모듈은 설정과 관련된 유용한 기능뿐 아니라 시스템 환경 변수를 접근하는 기능도 포함하고 있습니다. ch02_test6.js 파일을 만든 후 다음 코드를 입력합니다.

참조 파일 | NodeExample1>ch02_test6.js

```
var nconf = require('nconf');
nconf.env();

console.log('JAVA_HOME 환경 변수의 값 : %s', nconf.get('JAVA_HOME'));
```

nconf가 모듈로 만들어져 있으므로 require( ) 함수를 호출하여 불러옵니다. 그런데 여러분이 직접 만든 모듈이 아니라면 상대 패스가 아닌 모듈의 이름만 지정해서 불러옵니다. nconf 모듈은 그 모듈 안에 정의한 env( ) 함수를 호출하면 환경 변수에 대한 정보를 가져와서 속성으로 보관합니다.

그런데 한 가지 궁금한 점이 있습니다. 다른 사람이 만든 모듈은 도대체 어디에 있을까요? 다른 사람이 만들어 둔 모듈이 있다면 어딘가에 모듈 파일을 올려놓았을 것입니다. npm **패키지**를 사용하면 다른 사람이 만들어 올려놓은 패키지를 다운로드하여 설치할 수 있습니다.

정박사의 한마디

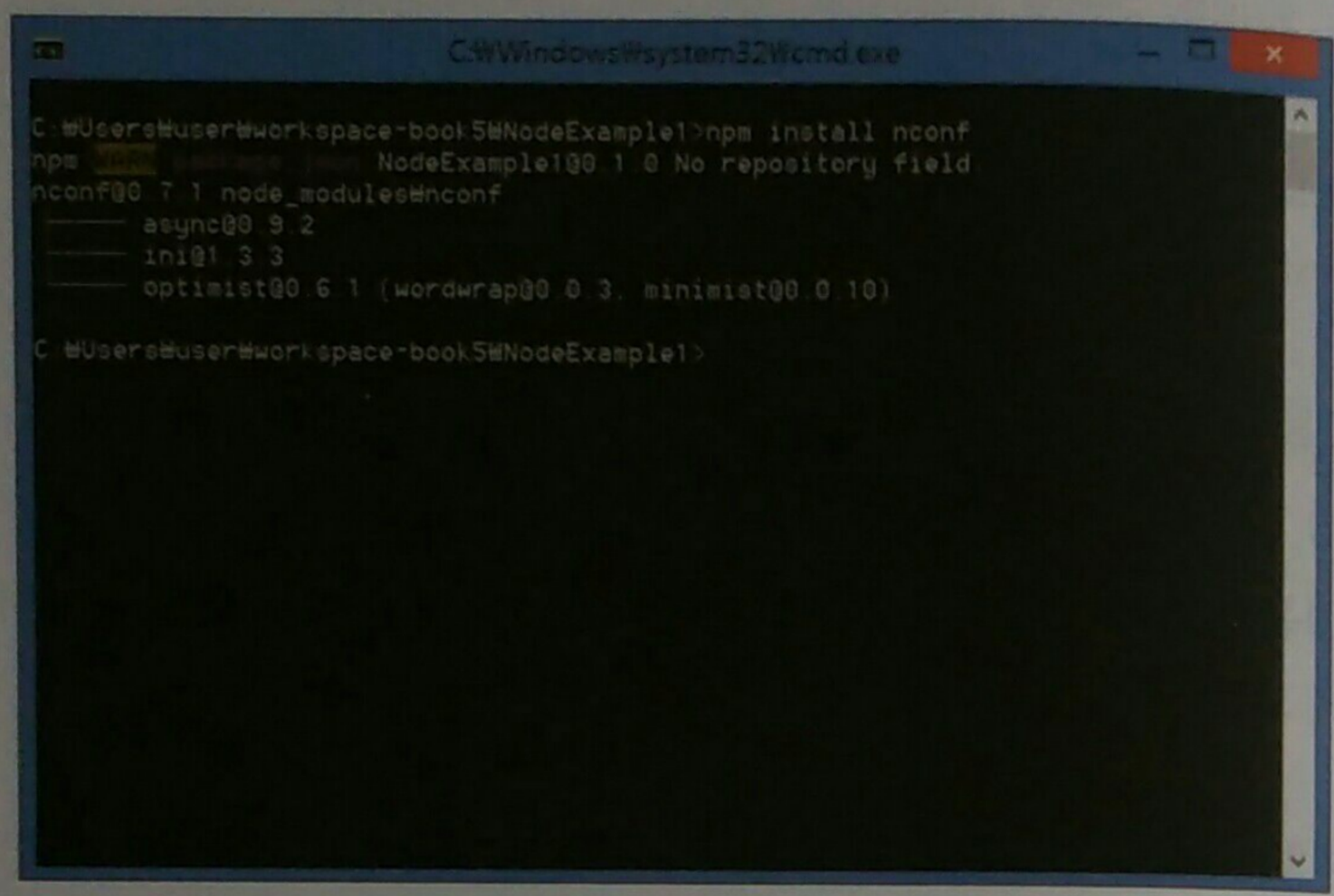
npm의 역할은 무엇일까요?

npm은 Node Package Manager의 약자로 노드의 패키지를 사용할 수 있도록 설치 및 삭제 등을 지원하는 프로그램입니다. 패키지는 여러분이 만들어 본 모듈에 몇 가지 정보를 더 넣은 것이라고 할 수 있으며, 보통 패키지를 만든 후 다른 사람이 사용할 수 있도록 인터넷에 올려 공유합니다. npm 프로그램은 이렇게 인터넷에 올려 공유하는 노드 패키지를 다운로드하여 설치할 수 있도록 도와줍니다.

명령 프롬프트를 열고 작업 공간(workspace) 폴더 안에 있는 [NodeExample1] 폴더로 이동한 후 다음 명령을 실행하여 nconf 패키지를 설치합니다.

명령 프롬프트에 입력

```
% npm install nconf
```

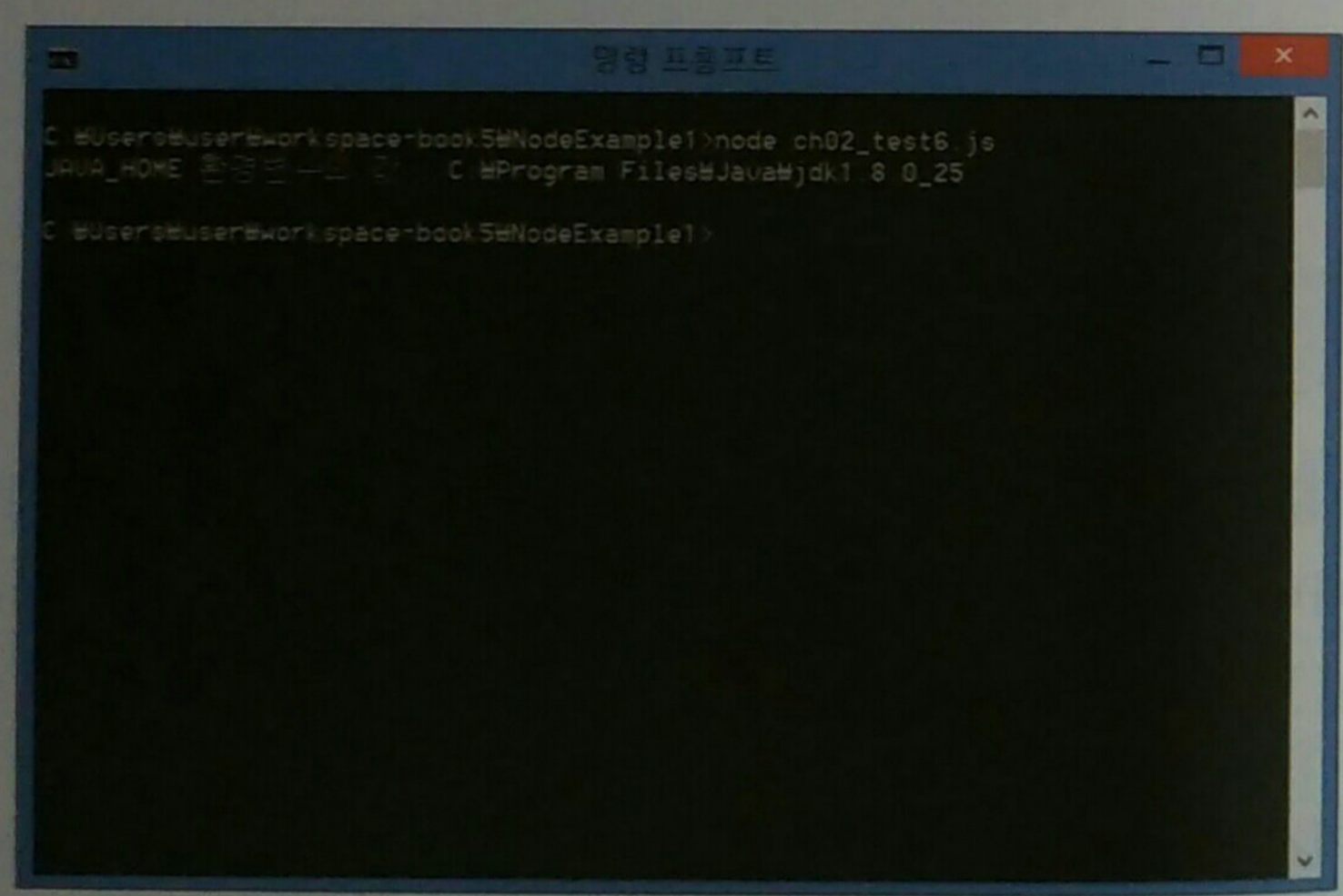


▲ nconf 패키지 설치하기

이제 명령 프롬프트 창에서 ch02_test6.js 파일을 실행하면 JAVA_HOME 환경 변수의 값을 확인할 수 있습니다.

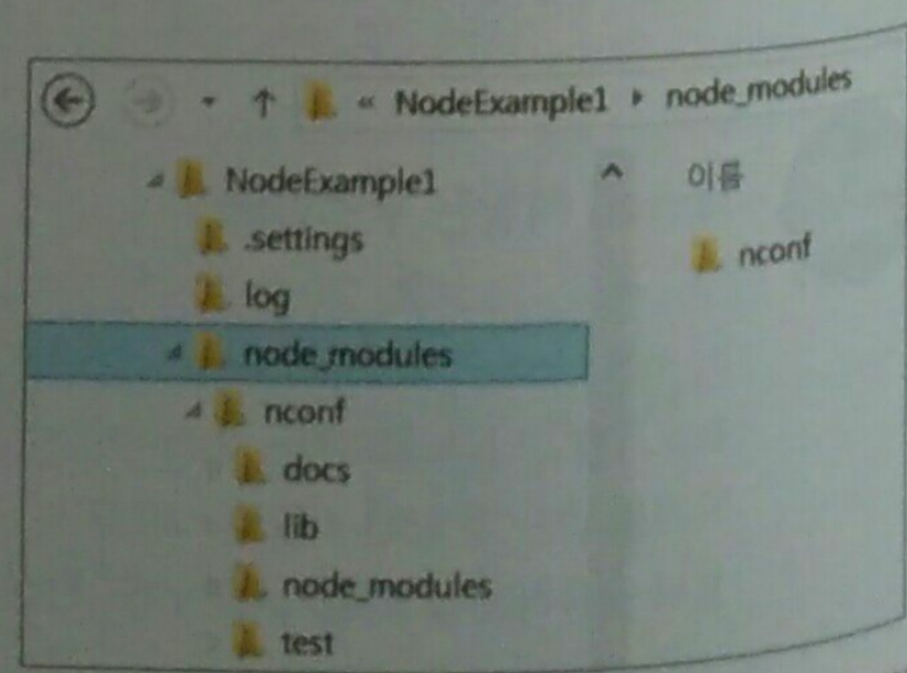
명령 프롬프트에 입력

```
% node ch02_test6.js
```

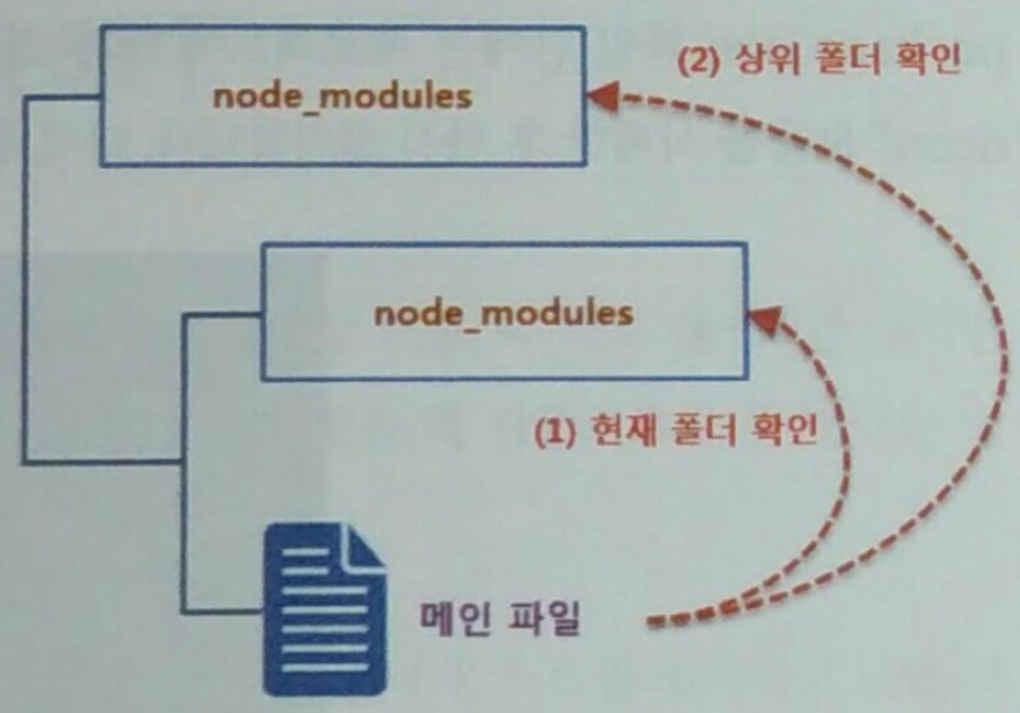


▲ 환경 변수를 확인하여 표시한 경우

그런데 npm으로 설치한 패키지는 어디에 설치되었을까요? npm으로 설치한 외부 패키지는 그 폴더 안에 만들어진 [node_modules] 폴더 안에 설치됩니다. 파일 탐색기를 열고 프로젝트 폴더를 찾아보면 [node_modules] 폴더를 찾을 수 있습니다. 그리고 그 안에 [nconf] 폴더가 있는 것을 확인할 수 있습니다.



그렇다면 프로젝트별로 패키지 설치가 가능하다고 이해할 수 있습니다. 그런데 새로운 프로젝트를 만들었다면 매번 npm으로 외부 패키지를 설치해야 할까요? 그렇지는 않습니다. 만약 모든 프로젝트에 적용하고 싶다면 [node_modules] 폴더를 프로젝트들의 상위 폴더인 [workspace] 폴더로 옮기면 간단히 해결됩니다. 또는 패키지를 설치할 때부터 상위 폴더에 설치하면 됩니다. 메인 파일이 실행될 때는 먼저 현재 폴더에 [node_modules] 폴더가 있는지 확인하고 만약 없다면 상위 폴더를 검색합니다. 그래도 찾지 못하면 상위 폴더를 순차적으로 검색합니다.



▲ node_modules 폴더를 찾는 순서

만약 다른 프로젝트나 PC에서 여러분이 만든 프로그램을 실행하려 할 때 설치한 외부 패키지가 많다면 일일이 다시 설치해야 합니다. 노드에서는 외부 패키지의 수만큼 npm 명령을 입력해야 하는 번거로움을 없앨 수 있도록 package.json 파일 안에 설치한 패키지들의 정보를 넣어 둘 수 있습니다. 다음은 프로젝트 파일 안에 만든 package.json 파일의 내용입니다.

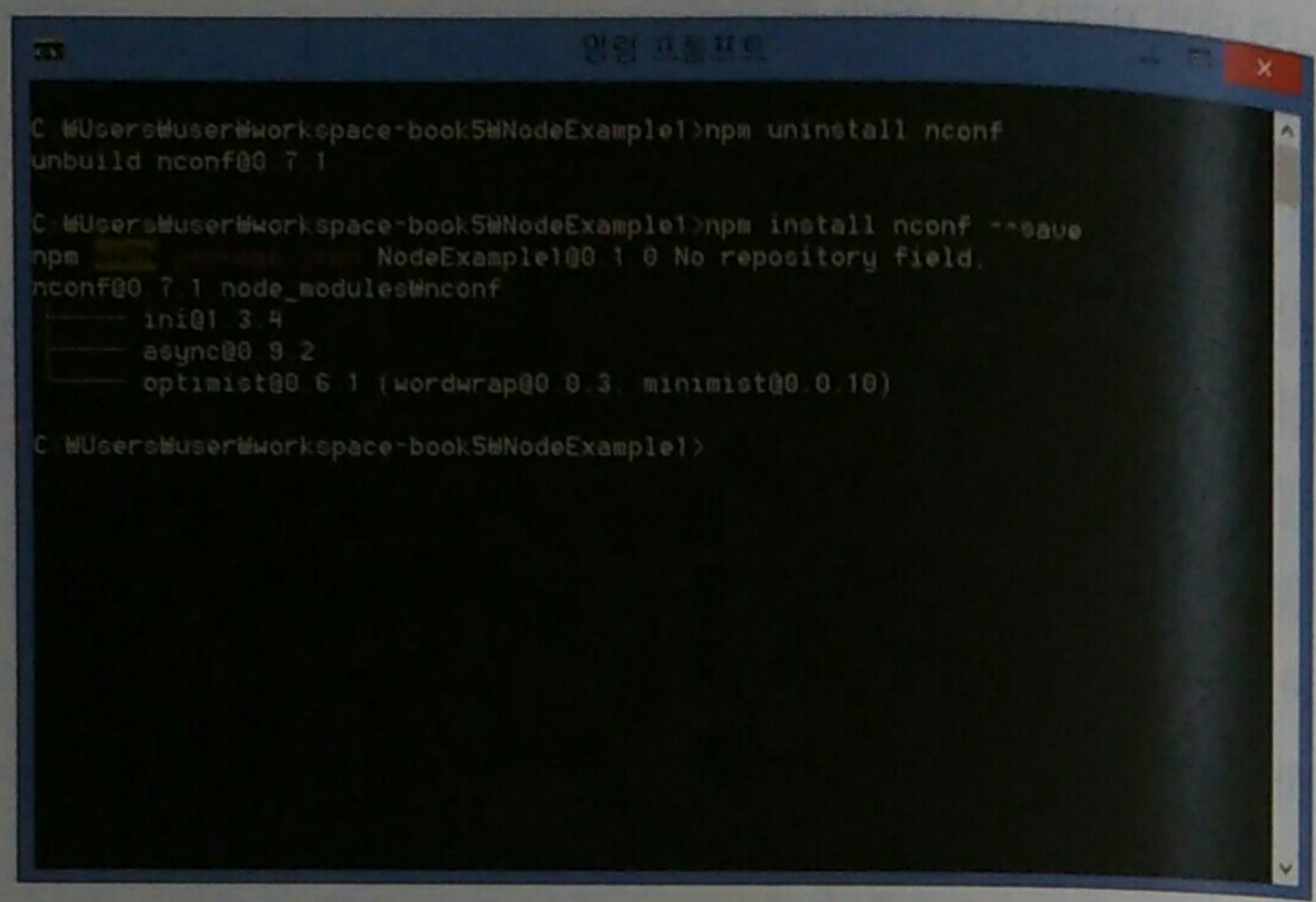
참조 파일 | NodeExample1>package.json

```
{
  "name": "NodeExample1",
  "version": "0.1.0",
  "description": "NodeExample1",
  "main": "hello-world-server.js",
  "scripts": {
    "test": "echo \"Error: no test specified! Configure in package.json\" && exit 1"
  },
  "repository": "",
  "keywords": [
    "node.js",
    "eclipse",
    "nodeclipse"
  ],
  "author": "",
  "license": "MIT",
  "readmeFilename": "README.md"
}
```

package.json 파일 안에는 프로젝트에 대한 기본 정보가 들어 있습니다. 이제 명령 프롬프트를 열고 nconf 모듈을 삭제한 후 다시 설치합니다. 다시 설치할 때는 npm 명령어에 --save 옵션을 추가합니다.

명령 프롬프트에 입력

```
% npm uninstall nconf
% npm install nconf --save
```



▲ nconf 모듈을 삭제했다가 --save 옵션으로 다시 설치한 경우

이제 package.json 파일을 다시 열어 보면 마지막 부분에 "dependencies" 속성이 추가된 것을 볼 수 있습니다.

참조 파일 | NodeExample1>package.json

```
{
  ......
  "dependencies": {
    "nconf": "^0.7.1"
  }
}
```

이렇게 설치한 패키지의 정보가 모두 들어간 package.json 파일이 만들어졌습니다. 이 프로젝트에서 사용한 모듈을 다른 PC에서 그대로 사용하고 싶다면 package.json 파일만 다른 PC로 옮긴 후 다음 명령을 입력하면 그 안에 들어 있는 모든 패키지가 한꺼번에 설치됩니다.

명령 프롬프트에 입력

```
% npm install
```

이 명령은 package.json 파일을 찾은 후 그 안에 있는 "dependencies" 속성의 값을 참조하여 패키지를 설치합니다.

02-5 | 간단한 내장 모듈 사용하기

모듈을 직접 만들어 사용하는 방법과 다른 사람들이 만들어 놓은 외장 모듈, 즉 외부 패키지를 설치한 후 불러와서 사용하는 방법을 알아보았습니다. 이번에는 노드를 설치했을 때 기본적으로 들어 있는 몇 가지 내장 모듈을 간단하게 살펴보겠습니다.

내장 모듈은 자주 사용하는 기본 기능을 노드에 포함시켜 제공하는 것입니다. 노드를 설치하고 나면 그 안에 제공되는 다양한 모듈을 사용할 수 있습니다. 이렇게 미리 포함되어 있는 모듈을 **내장 모듈**이라고 하며, 개발자가 직접 만들어 올린 모듈을 외장 모듈이라고 합니다. 외장 모듈은 npm으로 설치해야 하지만 내장 모듈은 바로 사용할 수 있습니다. 여기에서는 내장 모듈 중에서 os **모듈**과 path **모듈**에 대해 알아봅니다.

정박사의 한마디

내장 모듈보다 외장 모듈이 더 편리한 경우도 있어요

내장 모듈의 기능이 불편하거나 충분하지 못하다면 외장 모듈을 사용하는 게 더 편리할 수 있습니다. 여러분이 노드에서 동작하는 프로그램을 좀 더 쉽게 만들 수 있도록 이미 수많은 외장 모듈이 만들어져 있습니다. 따라서 이 책에서도 실무에서 사용할 수 있는 외장 모듈이 있다면 내장 모듈보다 먼저 설명합니다.

내장 모듈에 대한 정보는 다음 사이트에서 찾아볼 수 있습니다.

▶ http://nodejs.org/api

시스템 정보를 알려주는 os 모듈

실무에서 서버 기능을 만들다 보면 시스템의 CPU나 메모리 또는 디스크 용량이 얼마나 남아 있는지 확인하는 모니터링이 필요할 때가 있습니다. 이때 다음 표로 작성한 것처럼 os 모듈에 있는 메소드들을 사용할 수 있습니다. os 모듈이 가지고 있는 주요 메소드는 다음과 같습니다.

메소드 이름	설명
hostname( )	운영체제의 호스트 이름을 알려 줍니다.
totalmem( )	시스템의 전체 메모리 용량을 알려 줍니다.
freemem( )	시스템에서 사용 가능한 메모리 용량을 알려 줍니다.
cpus( )	CPU 정보를 알려 줍니다.
networkInterfaces( )	네트워크 인터페이스 정보를 담은 배열 객체를 반환합니다.

[표] os 모듈의 주요 메소드

새 파일을 ch02_test7.js로 만들고 다음과 같이 os 모듈을 사용하는 코드를 입력합니다.

참조 파일 | NodeExample1>ch02_test7.js

```
var os = require('os');

console.log('시스템의 hostname : %s', os.hostname());
console.log('시스템의 메모리 : %d / %d', os.freemem(), os.totalmem());
console.log('시스템의 CPU 정보\n');
console.dir(os.cpus());
console.log('시스템의 네트워크 인터페이스 정보\n');
console.dir(os.networkInterfaces());
```

파일을 실행하면 메모리, CPU 그리고 IP 주소를 포함한 네트워크 인터페이스 정보를 확인할 수 있습니다.

```
Console  Debug
<terminated> NodeExample1-ch02_test7.js [Node Application] Node.js Process
시스템의 hostname : bear
시스템의 메모리 : 2481160192 / 8518651904
시스템의 CPU 정보
[ { model: 'Intel(R) Core(TM) i7-4700MQ CPU @ 2.40GHz',
    speed: 2394,
    times:
     { user: 10557093,
       nice: 0,
       sys: 5473578,
       idle: 229783250,
       irq: 1027406 } },
  { model: 'Intel(R) Core(TM) i7-4700MQ CPU @ 2.40GHz',
    speed: 2394,
    times:
     { user: 7470000,
       nice: 0,
       -------
```

▲os 모듈을 이용해 메모리와 CPU 정보 등 확인하기

파일 패스를 다루는 path 모듈

파일을 다룰 때는 파일 패스에서 파일 이름을 폴더 이름과 구별하거나 파일 이름에서 확장자 또는 확장자를 제외한 파일 이름을 알아내는 작업이 많이 필요합니다. path 모듈은 파일 패스를 다룰 때 필요한 메소드를 제공합니다. path 모듈의 주요 메소드는 다음과 같습니다.

메소드 이름	설명
join( )	여러 개의 이름들을 모두 합쳐 하나의 파일 패스로 만들어 줍니다. 파일 패스를 완성할 때 구분자 등을 알아서 조정합니다.

dirname( )	파일 패스에서 디렉터리 이름을 반환합니다.
basename( )	파일 패스에서 파일의 확장자를 제외한 이름을 반환합니다.
extname( )	파일 패스에서 파일의 확장자를 반환합니다.

[표] path 모듈의 주요 메소드

ch02_test8.js 파일을 새로 만들고 다음과 같이 path 모듈을 사용하는 코드를 입력합니다.

참조 파일 | NodeExample1>ch02_test8.js

```
var path = require('path');

// 디렉터리 이름 합치기
var directories = ["users", "mike", "docs"];
var docsDirectory = directories.join(path.sep);
console.log('문서 디렉터리 : %s', docsDirectory);

// 디렉터리 이름과 파일 이름 합치기
var curPath = path.join('/Users/mike', 'notepad.exe');
console.log('파일 패스 : %s', curPath);
```

여러 개의 이름을 담고 있는 배열 객체를 만들었을 때 배열 객체의 join( ) 메소드를 사용하면 하나의 디렉터리 이름이나 파일 이름으로 만들 수 있습니다. 또한 path 객체의 join( ) 메소드를 사용해 디렉터리 이름과 파일 이름을 합칠 수도 있습니다.

거꾸로 파일 패스에서 디렉터리 이름이나 파일 이름을 구별하고 싶을 때는 dirname( ), basename( ), extname( ) 메소드를 사용하면 됩니다. ch02_test8.js 파일에 다음과 같이 추가로 입력합니다.

참조 파일 | NodeExample1>ch02_test8.js

```
.....

// 패스에서 디렉터리, 파일 이름, 확장자 구별하기
var filename = "C:\\Users\\mike\\notepad.exe";
var dirname = path.dirname(filename);
var basename = path.basename(filename);
var extname = path.extname(filename);

console.log('디렉터리 : %s, 파일 이름 : %s, 확장자 : %s', dirname, basename, extname);
```

파일을 실행하면 여러 개의 이름을 합쳐서 만든 디렉터리 이름을 확인할 수 있습니다. 그리고 하나의 파일 패스에서 구분한 파일의 이름도 확인할 수 있습니다.

```
Console  Debug
<terminated> NodeExample1-ch02_test8.js [Node Application] Node.js Process
문서 디렉토리 : users\mike\docs
파일 패스 : \Users\mike\notepad.exe
디렉토리 : C:\Users\mike, 파일 이름 : notepad.exe, 확장자 : .exe
```

▲ path 모듈을 이용해 파일 패스 다루기

지금까지 첫 번째 노드 프로젝트를 만들어 보았습니다. 모듈이 무엇인지 알아보았으며 노드에서 제공하는 몇 가지 기능도 간단하게 살펴보았습니다. 모듈에 대한 내용은 나중에 실제 코드를 만들면서 자세하게 살펴볼 것이므로 이 장에서 배운 내용이 충분히 이해되지 않아도 걱정할 필요는 없습니다.

그런데 혹시 지금까지 자바스크립트를 조금씩 사용하면서 자바스크립트가 어렵거나 익숙하지 않다고 느껴지나요? 노드에서는 자바스크립트를 기본 언어로 사용하므로 자바스크립트에 익숙해야 합니다. 다음 장에서는 자바스크립트의 주요 내용을 간단하게 정리할 것입니다. 자바스크립트에 익숙하지 않은 독자들을 위한 내용이지만, 이미 자바스크립트에 익숙한 개발자라도 한 번 더 정리해보기 바랍니다. 노드의 기능을 하나씩 확인하는 데 큰 도움이 될 테니까요.