Hanbit RealTime 113

You Don't Know JS

this와 객체 프로토타입

카일 심슨 지음 / 이일웅 옮김

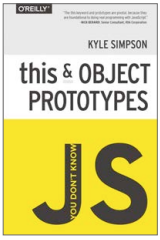
O'REILLY® 한빛미디어 Hanbit Media, Inc.

Hanbit RealTime 113

You Don't Know JS

this와 객체 프로토타입

카일 심슨 지음 / 이일웅 옮김



이 도서는
this & OBJECT PROTOTYPES(O'REILLY)의
번역서입니다

O'REILLY 한빛미디어 Hanbit Media, Inc.

You Don't Know JS **this와 객체 프로토타입**

초판발행 2015년 8월 27일

지은이 카일 심슨 / **옮김이** 이일웅 / **펴낸이** 김태헌
펴낸곳 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부
전화 02-325-5544 / **팩스** 02-336-7124
등록 1999년 6월 24일 제10-1779호
ISBN 978-89-6848-774-3 15000 / **정가** 14,000원

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

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

추천의 글

추천사를 제의받고 이 책을 읽어보니 처음 내가 자바스크립트를 배웠던 시절부터 지금껏 개발자로 지내온 15년 동안 자바스크립트가 얼마나 많이 변했는지 격세지감이다.

당시 자바스크립트에 입문했을 때 CSS/JS 같은 HTML 이외의 기술을 웹 페이지에 적용하는 것을 DTHML 또는 동적Dynamic HTML이라 불렀었다. 그 시절 자바스크립트는 페이지에 눈발이 날리게 하거나 상태 표시줄에 시계를 보여주는 등 정말 다양한 용도로 쓰였다. 인터넷을 잘 찾아보면 남들이 짜놓은 따끈따끈한 코드가 널려있어서 솔직히 난 자바스크립트 자체에는 별 관심이 없었다.

2005년에 이르러서야 자바스크립트가 좀 더 집중해서 공부해야 할 진정한 프로그래밍 언어라는 사실을 깨닫게 되었다. 구글 맵 1차 베타 릴리스 제품을 분석하면서 자바스크립트의 가능성에 흠뻑 매료된 것이다. 당시 구글 맵은 순수 자바스크립트만으로 마우스를 지도 위에 대고 마음대로 움직여 확대/축소하면서 페이지 새로 고침 없이 서버 요청이 가능한 전대미문의 획기적인 애플리케이션이었다. 내 눈엔 그저 마술처럼 신기하기만 했다!

뭔가가 희한한 마술처럼 보인다는 것은 새로운 기술과 테크닉에 눈을 뜨기 시작했다는 좋은 신호다. 아, 나 역시 예외는 아니어서 그때 이후 지금껏 죽 달려오면서, 자바스크립트를 클라이언트/서버 사이드 프로그래밍의 주 언어로 애용해왔다. 자바스크립트가 아니면 어찌 구현했을까 싶기도 하다.

한 가지 크게 아쉬운 점은 자바스크립트의 가능성을 2005년 이전에는 진지하게 알아보지 못했다는 것이다. 자바스크립트가 C++, C#, 자바 등 다른 언어처럼 유용한, 진짜 프로그래밍 언어로 발전하리라는 선견지명을 가지지 못한 탓이다.

초심자 시절로 다시 돌아가 'You Don't Know JS' 시리즈를 일독했더라면 내 커리어는 지금과 아주 많이 달라졌을 것이다. 재미있고 유익한 문체로 자바스크립트를 하

나둘 이해시키는 카일 심슨의 책들은 정말 사랑하지 않을 수 없다.

『this와 객체 프로토타입』은 더없이 훌륭한 'You Don't Know JS' 시리즈의 후속작으로 이전 작 『You Don't Know JS: 스코프와 클로저』(한빛미디어, 2014년)를 토대로 자바스크립트의 핵심 요소인 this 키워드와 프로토타입의 무대로 여러분의 지식을 자연스럽게 확장하게 한다. this와 객체 프로토타입은 자바스크립트 프로그래밍의 살아있는 근간이므로 차후 여러분이 어떤 책을 읽게 되더라도 대단히 중요하다. 특히 몸집이 크고 복잡한 애플리케이션을 제작하려면 자바스크립트로 객체를 생성하고 객체 간 관계를 맺고 확장하여 표현하는 방법을 개념부터 잘 이해해야 한다. 어설프게 알고 있다간 자바스크립트로(구글 맵 같은) 복잡한 애플리케이션을 개발할 생각은 아예 접는 게 좋다.

아마 대부분 웹 개발자들은 자바스크립트를 버튼과 AJAX 요청 따위를 붙이는 '이벤트 접착제' 정도로 사용해왔을 테고 자바스크립트 객체를 직접 건드려본 일이 한 번도 없는 사람들도 많을 것이다. 솔직히 나 역시 한때 그런 부류의 개발자였지만, 자바스크립트 프로토타입과 객체 생성에 대해 숙달하니 새로운 세상이 눈앞에 펼쳐지는 걸 느낄 수 있었다. 여러분도 과거의 나와 같다면 이 책을 붙들고 정독하기 바란다. 뭔가 새로운 지식을 찾아 헤매는 사람에게 안성맞춤 소스가 될 것이다. 어쨌든 여러분을 실망시키진 않을 테니 그냥 내 말을 믿고 책장을 넘기자!

– 닉 베라르디Nick Berardi

http://nickberardi.com/, @nberardi

저자 소개

지은이_ **카일 심슨**Kyle Simson

텍사스 오스틴 출신의 카일 심슨은 오픈 웹 전도사로, 자바스크립트, HTML5, 실시간 P2P 통신과 웹 성능에 누구 못지않은 열정을 갖고 있다. 안 그랬으면 이미 오래전에 질려버렸을 것이다. 저술가, 워크숍 강사, 기술 연사이며 오픈 소스 커뮤니티에서도 활약 중이다.

역자 소개

옮긴이_ **이일웅**

10년 넘게 국내, 미국 등지에서 대기업/공공기관 프로젝트를 수행한 웹 개발자이자, 두 딸의 사랑을 한몸에 받고 사는 행복한 딸 바보다. 자바 기반의 서버 플랫폼 구축, 데이터 연계, 그리고 다양한 자바스크립트 프레임워크를 응용한 프런트엔드 화면 개발을 주로 담당해 왔다. 시간이 날 때는 피아노를 연주한다.

- 개인 홈페이지: http://www.bullion.pe.kr

항상 그렇지만 단순해 보이는 것일수록 더 어렵습니다.

자바스크립트 언어는 '모든 것이 객체'일 뿐 여타 언어에서 제공하는 부가적인 장치는 거의 없지만, 역설적으로 이처럼 지극히 단순하여서 그때그때 상황에 맞게 역동적으로 변화하여 지금까지 진화해온 것 같습니다. 물론 너무 자유자재로 유연하게 변신하다 보니 미리부터 기가 질려 자바스크립트를 제대로 공부하려고 마음먹기보다는 주변 고수들의 소스 코드를 베껴 쓰고 싶은 충동을 느껴지는 건 어쩌면 당연할지도 모르겠습니다. 그러나 남의 코드만 가져다 쓰는 타성을 버리지 못하고 스스로 문제 해결을 할 수 있는 능력을 갖추지 않으면 언제까지나 자바스크립트는 그저 나랑은 상관없는, 대가들이 현란한 마술을 부리는 신비의 도구 정도로 남게 될 것입니다.

'You Don't Know JS' 시리즈의 세 번째 도서인 『this와 객체 프로토타입』은 자바스크립트 언어에서 가장 헷갈리고 난해한 동시에 그 무엇보다도 중요한 기본 개념인 this 바인딩과 프로토타입 체인, 이 두 주제를 집중적으로 파고든 책입니다. 실제로 제이쿼리jQuery 등 유명한 자바스크립트 프레임워크 소스를 읽어보면 `this.`나 `.prototype` 같은 코드가 유독 눈에 많이 띄고 이들을 매우 정교하게 잘 활용했다는 걸 알 수 있습니다. 이 책의 저자, 카일 심슨의 가르침을 차근차근 잘 따라가다 보면 예전에는 암호처럼 어렵기만 했던 고급 자바스크립트 코드가 비로소 하나둘 눈에 익기 시작할 것입니다. 그리고 일단 원리를 깨치고 나면 여러분 자신도 제이쿼리 같은 프레임워크를 자체 개발하거나 기능을 확장하는 작업을 훨씬 수월하게 진행할 수 있을 것입니다.

그리고 저자도 줄곧 강조한 것처럼 클래스 지향class-oriented 방식의 코드 체계가 이 세상에서 유일무이한, 절대적인 디자인 패턴이 아니라는 점과 자바스크립트는 위임 연결delegate link이라는 전혀 다른 방식의 객체 간 유기적인 연결 체계가 오히려 기존의 클래스 지향 방식보다 더 많은 장점과 가능성을 가지고 있다는 사실을 이해하면 한층 더 깊이 있고 확장된 지식을 가진 개발자로 거듭날 것입니다.

다만, 이 책은 커튼을 열어젖히고 자바스크립트의 내부적인 구조를 들여다보는 것이므로 『this와 객체 프로토타입』을 온전히 이해하려면 적지 않은 두통이 따르는 건 피할 길이 없습니다. 코드와 설명만으로 참조 흐름이 한눈에 그려지지 않는 분들은 제가 삽입한 그림처럼 직접 손으로 그려보면서 이해하려고 노력하면 효과가 있을 것입니다.

모쪼록 부족한 번역이나마 많은 자바스크립트 개발자가 곁에 두고 자주 찾고 싶은 도서가 되길 바라며, 이 시리즈 번역을 맡겨주신 한빛미디어 김창수 팀장님과 스마트미디어팀 여러분, 그리고 주말과 휴일 내내 많은 시간 함께 해주지 못한, 사랑하는 제 아내와 두 딸, 제이와 솔이에게 이 역서를 바칩니다. 언제나 아들에게 변함없는 믿음과 사랑을 보내주신 부모님께 늘 감사드립니다.

2015년 여름 문턱에서

이일웅

1. 한글 맞춤법 표준어 규정과 외래어 표기법을 준수한다.
2. 기술 용어, 제품명 등 고유 명사 형태의 원어는 최대한 한글로 음차하여 옮긴다(예: JavaScript → 자바스크립트). 약자 형태로 축약된 원어나 그밖에 한글 음차로 옮기는 것보다 원어 그대로 표기하는 편이 독자의 이해에 도움이 된다면 원어로 표기한다(예: PK).
3. 2번에서 한글 음차 시 최초 1회 영문을 위 첨자 형태로 병기하고, 이후 반복하여 등장할 경우 이를 생략한다. 그러나 이렇게 병기한 용어가 다시 나올 경우에도 독자의 이해를 위해 필요한 경우 다시 영문을 병기한다.
4. 'You Don't Know JS' 시리즈 도서는 대체로 일상생활에서 대화를 나누는 듯한 구어적인 표현이 많은데, 이러한 특성을 한글 번역본에도 최대한 반영하려고 노력하였다.
5. 이미 업계나 기술자들 사이에서 많이 사용되어 외래어처럼 굳어진 용어는 어차피 이 도서의 대상 독자가 일반인이 아니므로 굳이 우리말로 번역하지 않고 원어를 그대로 음차한다(예: type → 형 타입, copy and paste → ~~복사 후 붙여넣기~~ 카피 앤 페이스트).
6. 저자가 기술한 원문이 직역 시 이해가 어렵다고 판단하면 문장 구조를 재배열하거나 관련 문구를 추가하는 식으로 의역을 병행한다. 필요하다면 번안 수준의 번역을 일부 적용한다.
7. 예제 코드의 주석 및 기술적인 내용과는 무관한 상수 문자열은 한글 번역을 하되, 프로그램 로직을 파악하는 데 오히려 방해되거나 코드 가독성을 떨어뜨릴 수 있는 경우 원래의 코드를 유지한다.

이미 눈치챘겠지만, 시리즈 제목 일부인 'JS'는 자바스크립트를 깎아내릴 의도로 쓴 약어가 아니다. 물론 자바스크립트 언어에 숨겨진 기벽quirk이 만인이 비난하는 대상임은 부인할 수 없겠지만!

웹 초창기 시절부터 자바스크립트는 사람들이 대화하듯 웹 콘텐츠를 소비할 수 있게 해준 기반 기술이었다. 마우스 트레일을 깜빡이거나 팝업 알림창을 띄워야 할 수요에서 비롯되어 20년 가까이 흐른 지금, 자바스크립트는 엄청난 규모로 기술적 역량이 성장하였고, 세계에서 가장 널리 사용되는 소프트웨어 플랫폼이라 불리는, 웹의 심장부를 형성하는 핵심 기술이 되었다.

그러나 프로그래밍 언어로서의 자바스크립트는 끊임없는 비난과 논란의 대상이기도 했는데, 부분적으론 과거로부터 전해 내려온 폐해 탓이기도 하지만 그보다 설계 철학 자체가 문제시되기도 했다. 브렌단 아이크Brendan Eich[01]의 표현을 빌자면, '자바스크립트'란 이름 자체가 좀 더 성숙하고 나이 많은 형인 '자바' 아래의 '바보 같은 꼬마 동생dumb kid brother'같은 느낌을 준다. 하지만 이름은 정치와 마케팅 사정상 우연히 그렇게 붙여진 것일 뿐, 두 언어는 여러 중요한 부분에서 이질적이다. '자바스크립트'와 '자바'는 '카메라'와 '카car'만큼이나 무관하다.

C 스타일의 절차 언어에서 미묘하며 불확실한 스킴SchemeLisp 스타일의 함수형 언어에 이르기까지 자바스크립트는 서너 개 언어로부터 근본 개념과 구문 체계를 빌려왔기 때문에 꽤 폭넓은 개발자층을 확보하는데 대단히 유리했고, 심지어 프로그래밍 경력이 별로 없는 사람들도 쉽게 배울 수 있었다. 'Hello World'를 자바스크립트로 출력하는 코드는 너무 단순해서 출시 당시엔 나름의 매력이 있었고 금방 익숙해졌다.

01 역자주_자바스크립트의 창시자. 1995년 넷스케이프 근무 당시 열흘 만에 자바스크립트 언어를 고안했습니다.

자바스크립트는 처음 시작하고 실행하기가 가장 쉬운 언어지만, 독특한 기벽 탓에 다른 언어들보다 언어 자체를 완전히 익히고 섭렵한 달인은 주변에서 찾아보기 어렵다 C/C++ 등으로 전체 규모full-scale의 프로그램을 작성하려면 언어 자체를 깊이 있게 알고 있어야 가능하지만, 자바스크립트는 언어 전체의 능력 중 일부를 수박 겉핥기 정도만 알고 사용해도 웬만큼 운영 서비스가 가능하다.

언어 깊숙이 뿌리를 내려 자리 잡은, 정교하고 복잡한 개념이 외려 (콜백 함수를 다른 함수에 인자로 넘기는 것처럼) 겉보기에 단순한 방식으로 사용해도 괜찮게끔 유도하고, 그러다 보니 자바스크립트 개발자는 내부에서 무슨 일들이 벌어지든 있는 그대로의 언어 자체를 사용하여 개발할 수 있다.

그러나 간단하고 쓰기 쉬운 언어일수록 여러 가지 의미와 복잡하고 세밀한, 다양한 기법들이 결집해 있어서 꼼꼼하게 학습하지 않으면 제아무리 노련한 개발자 할지라도 올바르게 이해하지 못한다.

이것이 바로 자바스크립트의 역설이자 아킬레스건이며, 이 책을 읽고 여러분이 넘어야 할 산이다. 다 알지 못해도 사용하는 데 문제가 없다 보니 끝내 자바스크립트를 제대로 이해하지 못하고 넘어가는 경우가 비일비재하다.

목표

자바스크립트의 놀랍거나 불만스런 점들을 마주할 때마다 자신의 블랙리스트에 추가하여 금기시한다면(이런 일에 익숙한 사람들이 더러 있다) 자바스크립트란 풍성함의 빈 껍데기에 머무르게 될 것이다.

누군가 '좋은 부분The Good Parts'이란 유명한 별칭을 달아놓았는데, 부디 독자 여러분! '좋은 부분'이라기 보다는 차라리 '쉬운 부분', '안전한 부분', 또는 '불완전한 부분'이라고 하는 편이 더 정확할 것이다.

'You Don't Know JS' 시리즈는 정반대 방향으로 접근한다. 자바스크립트의 모든 것, 그 중 특히 '어려운 부분The Tough Part'을 심층적으로 이해하고 학습할 것이다!

나는 자바스크립트 개발자들이 정확히 언어가 어떻게, 그리고 왜 그렇게 작동하는지 알려 하지 않고, "그냥 이 정도면 됐지" 식으로 이해하고 대충 때우려는 자세를 직접 거론할 것이다. 험한 길을 마주한 상황에서 쉬운 길로 돌아가라는 식의 조언은 절대 하지 않으련다.

코드가 일단 잘 돌아가니 이유는 모른 채 그냥 지나치는 건 내 성질에 용납할 수 없다. 여러분도 그래야 한다. 여러분이 나와 함께 험난한 '가시밭길'을 탐험하면서 자바스크립트가 무엇인지, 자바스크립트로 뭘 할 수 있는지 포괄적으로 배우기 바란다. 이런 지식을 확실히 보유하고 있으면, 테크닉, 프레임워크, 금주의 인기 있는 머리글자 따위는 여러분 손바닥 위에서 벗어나지 않을 것이다.

이 시리즈는 자바스크립트에 대해 가장 흔히 오해하고 있거나 잘못 이해하고 있는, 특정한 핵심 언어 요소를 선정하여 아주 깊고 철저하게 파헤친다. 여러분은 이론적으로만 알고 넘어갈 것이 아니라 실전적으로 '내가 알고 있어야 할' 내용을 분명히 다 알고 간다는 확신을 가지고 책장을 넘기기 바란다.

아마도 지금 여러분이 알고 있는 자바스크립트는 다른 사람들이 불완전한 이해로 구워낸 단편적인 지식을 물려받은 정도일 것이다. 이런 자바스크립트는 진정한 자바스크립트의 그림자에 불과하다. 여러분은 지금 자바스크립트를 제대로 모르지만 이 시리즈를 열독하면 완벽히 알게 될 것이다. 동료, 선후배 여러분, 포기하지 말고 계속 읽기 바란다. 자바스크립트가 여러분의 두뇌를 기다리고 있다.

정리하기

자바스크립트는 굉장한 언어다. 적당히 아는 건 쉬워도 완전히(충분히) 다 알기는 어렵다. 헷갈리는 부분이 나오면 개발자들은 대부분 자신의 무지를 탓하기 전에 언어 자체를 비난하곤 한다. 이 시리즈는 이런 나쁜 습관을 바로잡고 이제라도 여러분이 자바스크립트를 제대로, 깊이 있게 이해할 수 있도록 도와주는 것을 목표로 한다.

이 책의 예제 코드를 실행하려면 현대적인 자바스크립트 엔진(예: ES6)이 필요하다. 구 엔진(ES6 이전)에서는 코드가 작동하지 않을 수 있다.

예제 코드

예제 코드와 연습 문제는 http://bit.ly/ydkjs-this-code에서 내려받을 수 있습니다.

한빛 리얼타임

한빛 리얼타임은 IT 개발자를 위한 eBook입니다.

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

한빛이 지금 생각하고 추구하는, 개발자를 위한 리얼타임 전자책은 이렇습니다.

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

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

2 무료로 업데이트되는 전자책 전용 서비스입니다

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

3 독자의 편의를 위해 DRM-Free로 제공합니다

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

4 전자책 환경을 고려한 최적의 형태와 디자인에 담고자 노력했습니다

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

앞으로도 독자 여러분의 충고에 귀 기울이며 지속해서 발전시켜 나가도록 하겠습니다.

차례

chapter 1

this라나 뭐라나

자바스크립트에서 가장 헷갈리는 체계mechanism 중 하나가 바로 this 키워드다. this는 모든 함수 스코프 내에 자동으로 설정되는 특수한 식별자로 경험 많은 자바스크립트 개발자도 정확히 무엇을 가리키는지 짚어내기가 만만치 않다.

> **모든 기술이 고도로 발전하면 마술과 구별하기 어려워진다.**
>
> – 아서 클라크Arthur C. Clarke

자바스크립트 this가 이 정도로 고도화된 체계는 아니지만, 개발자들은 '고도화'란 표현을 '복잡하다', '혼란스럽다' 같은 단어로 바꿔 표현하곤 한다. 어쨌거나 한 가지 분명한 사실은 명확하게 이해하지 못하는 한 this는 그저 신비로운 마술이나 다름없다는 점이다.

1.1 this를 왜?

숙련된 개발자조차 헷갈릴 정도라면 대체 this는 뭐에 쓰는 물건인지 궁금할 것이다. 공들여 학습할 가치는 있을까? '어떻게' 이전에 '왜'부터 따져보자.

먼저 this의 유용함과 사용 동기를 알아보자. 다음 코드가 '어떻게' 작동하는지 혼란스럽다면 일단 걱정 붙들어 매시라! 뒤에서 다 설명한다. '어떻게'는 잠시 접어두고 '왜'라는 문제에만 집중하자.

identify( ), speak( ) 두 함수는 객체별로 따로따로 함수를 작성할 필요 없이 다중 콘텍스트context 객체인 me와 you에서 모두 재사용할 수 있다.

```
function identify() {
    return this.name.toUpperCase();
}

function speak() {
    var greeting = "Hello, I'm " + identify.call( this );
    console.log( greeting );
}

var me = {
    name: "Kyle"
};

var you = {
    name: "Reader"
};

identify.call( me );  // KYLE
identify.call( you ); // READER

speak.call( me );     // Hello, I'm KYLE
speak.call( you );    // Hello, I'm READER
```

this를 안 쓰고 identify( ), speak( ) 함수에 콘텍스트 객체를 명시할 수도 있다.

```
function identify(context) {
    return context.name.toUpperCase();
}

function speak(context) {
    var greeting = "Hello, I'm " + identify( context );
    console.log( greeting );
}

identify( you );  // READER
speak( me );      // Hello, I'm KYLE
```

하지만 암시적인 객체 레퍼런스를 '함께 넘기는passing along' this 체계가 좀 더 API 설계가 깔끔하고 명확하며 재사용하기 쉽다.

사용 패턴이 복잡해질수록 보통 명시적인 파라미터로 콘텍스트를 넘기는 방법이 this 콘텍스트를 사용하는 것보다 코드가 더 지저분해진다. 뒷부분에서 객체와 프로토타입을 배우고 나면 여러 함수가 적절한 콘텍스트 객체를 자동 참조하는 구조가 얼마나 편리한지 실감하게 될 것이다.

1.2 헷갈리는 것들

앞에서 '어떻게'를 곧 설명한다고 했는데, 그 전에 코드가 실행되지 않을 거란 의구심은 떨쳐버리자. 'this'란 이름 자체가 글자 그대로만 생각하게 해 헷갈리게 하는 구석이 있다. 사람들은 보통 두 가지 의미로 해석하는데, 결론부터 미리 말하면 둘 다 틀렸다.

1.2.1 자기 자신

우선 this가 함수 그 자체를 가리킨다는 오해다. 그럴싸한 문법적 추론이다. 함수가 내부에서 자기 자신을 가리킬 일이 있을까? 재귀(함수 내부에서 자기 자신을 다시 호출) 로직이 들어가는 경우도 있고, 최초 호출 시 이벤트에 바인딩된 함수 자신을 언바인딩unbinding할 때도 자기 참조가 필요할 것이다.

자바스크립트 체계에 생소한 개발자들은 보통 함수를 객체(자바스크립트의 모든 함수는 객체다!)로 참조함으로써 함수 호출 간 '상태state'(프로퍼티 값)를 저장할 수 있을 거라 생각한다. 분명 그렇게 할 수 있고 제한적이나마 가능하지만, 함수 객체 말고도 더 좋은 장소에 상태를 저장하는 패턴에 대해서는 이 책의 나머지 부분에서 설명한다.

우선 여기서는 사람들의 생각과는 달리 함수가 this로 자기 참조를 할 수 없다는 걸 증명하기 위해 한 가지 패턴을 살펴보겠다. 다음은 함수(foo) 호출 횟수를 추적하는 예제다.

```
function foo(num) {
    console.log( "foo: " + num );

    // 'foo'가 몇 번 호출되었는지 추적한다.
    this.count++;
}

foo.count = 0;

var i;

for (i=0; i<10; i++) {
    if (i > 5) {
        foo( i );
    }
}

// foo: 6
// foo: 7
// foo: 8
// foo: 9

// 'foo'는 몇 번 호출되었을까?
console.log( foo.count ); // 0 — 엥?
```

분명 console.log에 네 차례의 foo(..) 함수 호출 횟수가 표시되었는데, foo.count 값은 0이다. this.count++에서의 this를 너무 글자 그대로 해석하면 헷갈릴 수 있다.

foo.count = 0을 하면 foo라는 함수 객체에 count 프로퍼티가 추가된다. 하지만 this.count에서 this는 함수 객체를 바라보는 것이 아니며, 프로퍼티 명이 똑같아 헷갈리지만 근거지를 둔 객체 자체가 다르다.

진지한 개발자라면 이쯤 해서 손들고 질문할 것이다. "애초 의도한 대로 count 프로퍼티가 증가한 게 아니라면 그럼 대체 어떤 count가 증가한 겁니까?" 조금 더 파고들면 우연히도 그 장본인은 전역 변수 count로(이 과정은 2장에서 다시 설명한다) 현재 값은 NaN임을 알 수 있다. 물론 그래도 또 다른 의문이 꼬리를 물 것이다. "좋아요, 그게 전역 변수라 치면 지금은 왜 숫자 값이 아닌 NaN이 들어있는 거죠?"(2장 참고)

많은 개발자가 이 부분에서 잠시 생각을 멈추고 "왜 this 참조가 이상하게 이루어졌을까?"라는 중요한 질문에 스스로 답을 찾지 않고, 그저 이슈 자체를 피해가거나 count 프로퍼티를 다른 객체로 옮기는 등의 우회책을 떠올린다.

```
function foo(num) {
    console.log( "foo: " + num );

    // 'foo'가 몇 번 호출되었는지 추적한다.
    data.count++;
}

var data = {
    count: 0
};

var i;
for (i=0; i<10; i++) {
    if (i > 5) {
        foo( i );
    }
}

// foo: 6
// foo: 7
// foo: 8
// foo: 9

// 'foo'는 몇 번 호출되었을까?
console.log( data.count ); // 4
```

이렇게 해서 해결할 수도 있지만 아쉽게도 this가 뭔지, 작동 원리는 무엇인지 모르는 채 문제의 본질을 벗어나 어휘 스코프lexical scope라는 편리한 장치에 몸을 내맡긴 꼴이다.

어휘 스코프를 사용하지 말라는 게 아니다(『You Don't Know JS 스코프와 클로저』[02] 참고). 어휘 스코프 그 자체는 훌륭하고 유용한 시스템이다. 그러나 this에 대해 어렴풋이 짐작만 하다가 뭐가 좀 안 된다 싶으면 그냥 어휘 스코프를 다시 찾는 식으로는 발전이 없다.

01 http://www.hanbit.co.kr/ebook/look.html?isbn=9788968486333

함수가 내부에서 자신을 참조할 때 일반적으로 this만으로는 부족하며 어휘 식별자lexical identifier(변수)를 거쳐 함수 객체를 참조한다.

다음 두 함수를 보자.

```
function foo() {
    foo.count = 4;    // 'foo'는 자기 자신을 가리킨다.
}

setTimeout( function(){
    // 익명 함수(이름이 없는 함수)는 자기 자신을 가리킬 방법이 없다.
}, 10 );
```

'이름 붙은 함수named function'라 불리는 foo 함수는 foo라는 함수명 자체가 내부에서 자신을 가리키는 레퍼런스로 쓰인다. 하지만 setTimeout(. .)에 콜백으로 전달한 함수는 이름 식별자name identifier가 없으므로(그래서 익명 함수anonymous function라고 한다) 함수 자신을 참조할 방법이 마땅치 않다.

arguments.callee도 실행 중인 함수 객체를 가리키지만, 꽤 오래전 등장하여 지금은 권장하지 않는deprecated 레퍼런스다. 과거에는 익명 함수 객체를 내부에서 접근할 때 쓰는 유일한 방법이었지만, 자기 참조가 필요하다면 익명 함수를 쓰기보단 이름 붙은 함수(표현식)를 사용하는 게 최선이다. arguments.callee는 이제 옛 유물이니 더는 쓰지 말자.

앞 예제에서 this 없이 함수 객체 레퍼런스로 foo 식별자를 대신 사용해도 문제없이 작동한다.

```
function foo(num) {
    console.log( "foo: " + num );

    // 'foo'가 몇 번 호출되었는지 추적한다.
    foo.count++;
}

foo.count = 0;
```

```
var i;

for (i=0; i<10; i++) {
    if (i > 5) {
        foo( i );
    }
}

// foo: 6
// foo: 7
// foo: 8
// foo: 9

// 'foo'는 몇 번 호출되었을까?
console.log( foo.count ); // 4
```

그러나 이 역시 this를 제대로 이해하지 않은 채 문제를 회피하여 foo 어휘 스코프에 의존하기는 마찬가지다.

foo 함수 객체를 직접 가리키도록 강제하는 것도 방법이다.

```
function foo(num) {
    console.log( "foo: " + num );

    // 'foo'가 몇 번 호출되었는지 추적한다.
    // 참고: 'this'는 'foo'를 어떻게 호출하느냐에 따라 진짜 'foo'가 된다.
    this.count++;
}

foo.count = 0;
var i;

for (i=0; i<10; i++) {
    if (i > 5) {
        // 'call(..)' 함수로 호출하므로
        // 'this'는 이제 확실히 함수 객체 'foo' 자신을 가리킨다.
        foo.call( foo, i );
    }
}

// foo: 6
// foo: 7
```

```
// foo: 8
// foo: 9

// 'foo'는 몇 번 호출되었을까?
console.log( foo.count ); // 4
```

this를 피하지 않고 그대로 적용했다. 지금까지 열거한 기법들에 대해서는 곧 자세히 설명할 테니 지금은 조금 골치가 아프더라도 너무 걱정하지 마시길!

1.2.2 자신의 스코프

this가 바로 함수의 스코프를 가리킨다는 말도 아주 흔한 오해다. 어떤 면에선 맞지만 잘못 이해한 것인데…… 사실 까다로운 문제다.

분명한 건 this는 어떤 식으로도 함수의 어휘 스코프를 참조하지 않는다는 사실! 내부적으로 스코프는 별개의 식별자가 달린 프로퍼티로 구성된 객체의 일종이나, 스코프 '객체'는 자바스크립트 구현체인 '엔진'의 내부 부품이기 때문에 일반 자바스크립트 코드로는 접근하지 못한다.

넘지 말아야 할 선을 넘어 this가 암시적으로 함수의 어휘 스코프를 가리키도록 해보자(물론 실패한다!).

```
function foo() {
    var a = 2;
    this.bar();
}

function bar() {
    console.log( this.a );
}

foo(); //참조 에러: a는 정의되지 않았습니다(ReferenceError: a is not defined).
```

이 코드는 여러 곳에서 실수를 했다. 억지로 만든 예제 코드처럼 보이지만 실제로

많은 커뮤니티 사이트에서 공유되는 코드다. 슬프게도 this가 얼마나 빈번한 오해의 대상인지 알 수 있는 단면이다.

bar( ) 함수를 this.bar( )로 참조하려고 한 것부터가 문제다. 곧 설명하지만 여기서 에러가 나지 않더라도 우연일 뿐이다. bar( ) 앞의 this를 빼고 식별자를 어휘적으로 참조하는 것이 가장 자연스러운 호출 방법이다.

물론 이 코드의 작성자는 foo( )와 bar( )의 어휘 스코프 사이에 어떤 연결 통로를 만들어 bar( )가 foo( )의 내부 스코프에 있는 변수 a에 접근하게 하고 싶었을 것이다. 그러나 그런 연결 통로는 없다. 어휘 스코프 안에 있는 뭔가를 this 레퍼런스로 참조하기란 애당초 가능하지 않다.

this와 어휘 스코프 참조가 계속 헷갈리는 독자들은 조용히 이 말을 되뇌기 바란다. '연결 통로 따윈 없어!'

1.3 this는 무엇인가?

부정확한 추측을 이제 정리하고 this 체계가 어떻게 작동하는지 알아보자.

this는 작성 시점이 아닌 런타임 시점에 바인딩되며, 함수 호출 당시 상황에 따라 콘텍스트가 결정된다고 말했다. 함수 선언 위치와 상관없이 this 바인딩은 오로지 어떻게 함수를 호출했느냐에 따라 정해진다.

어떤 함수를 호출하면 활성화 레코드activation record, 즉 실행 콘텍스트execution context가 만들어진다. 여기엔 함수가 호출된 근원(콜스택call-stack)과 호출 방법, 전달된 파라미터 등의 정보가 담겨있다. this 레퍼런스는 그중 하나로 함수가 실행되는 동안 이용할 수 있다.

다음 장에서는 this 바인딩을 결정짓는 함수 호출부call-site에 대해 설명한다.

1.4 정리하기

this 바인딩은 그 작동 원리를 철저하게 학습하지 않은 자바스크립트 개발자들에게 예나 지금이나 애매한 주제다. 대충 감으로 때려잡고 시행착오를 반복하다 결국 Stack Overflow[02] 같은 사이트에서 아무렇게나 남의 코드를 카피 앤 페이스트copy-and-paste하는 식으로는 this 체계의 핵심을 반의반도 파악하기 어렵다.

this를 제대로 배우고 싶다면 먼저 가지고 있는 별의별 오해와 추측을 내버리고 this가 함수 자신이나 함수의 어휘 스코프를 가리키는 레퍼런스가 아니라는 점을 분명히 인지해야 한다.

this는 실제로 함수 호출 시점에 바인딩되며 전적으로 함수를 호출한 코드에 의해 무엇을 가리킬지 결정된다.

02 http://stackoverflow.com/