PART 1
INTRODUCTION

i
Browsers and the Web

I—this is Chris speaking—have known the web[1] for all of my adult life. The web for me is something of a technological companion, and I've never been far from it in my studies or my work. Perhaps it's been the same for you. And using the web means using a browser. I hope, as you read this book, that you fall in love with web browsers, just like I did.

i.1 The Browser and Me

Since I first encountered the web and its predecessors,[2] in the early 1990s, I've been fascinated by browsers and the concept of networked user interfaces. When I surfed [5] the web, even in its earliest form, I felt I was seeing the future of computing. In some ways, the web and I grew together—for example, 1994, the year the web went commercial, was the same year I started college; while there I spent a fair amount of time surfing the web, and by the time I graduated in 1999, the browser had fueled the famous dot-com speculation gold rush. Not only that, but the company for which I now work, Google, is a child of the web and was founded during that time.

In my freshman year at college, I attended a presentation by a RedHat salesman. The presentation was of course aimed at selling RedHat Linux, probably calling it the "operating system of the future" and speculating about the "year of the Linux desktop". But when asked about challenges RedHat faced, the salesman mentioned not Linux but *the web*: he said that someone "needs to make a good browser for Linux".[3] Even back then, in the first years of the web, the browser was already a necessary component of every computer. He even threw out a challenge: "How hard could it be to build a better browser?" Indeed, how hard could it be? What makes it so hard? That question stuck with me for a long time.[4]

How hard indeed! After eleven years in the trenches working on Chrome, I now know the answer to his question: building a browser is both easy and incredibly hard, both intentional and accidental. And everywhere you look, you see the evolution and history of the web wrapped up in one codebase. It's fun and endlessly interesting.

So that's how I fell in love with web browsers. Now let me tell you why you will, too.

[1] Broadly defined, the web is the interlinked network ("web") of web pages [1] on the internet. If you've never made a web page, I recommend MDN's Learn Web Development [2] series, especially the Getting Started [3] guide. This book will be easier to read if you're familiar with the core technologies.

[2] For me, bulletin board systems (BBSs [4]) over a dial-up modem connection. A BBS, like a browser, is a window into dynamic content somewhere else on the internet.

[3] Netscape Navigator was available for Linux at that time, but it wasn't viewed as especially fast or featureful compared to its implementation on other operating systems.

[4] Meanwhile, the "better Linux browser than Netscape" took a long time to appear...

i.2 The Web in History

The web is a grand, crazy experiment. It's natural, nowadays, to watch videos, read news, and connect with friends on the web. That can make the web seem simple and obvious, finished, already built. But the web is neither simple nor obvious (and is certainly not finished). It is the result of experiments and research, reaching back to nearly the beginning of computing,[5] about how to help people connect and learn from each other.

In the early days, the internet was a world-wide network of computers, largely at universities, labs, and major corporations, linked by physical cables and communicating over application-specific protocols. The (very) early web mostly built on this foundation. Web pages were files in a specific format stored on specific computers. The addresses for web pages named the computer and the file, and early servers did little besides read files from a disk. The logical structure of the web mirrored its physical structure.

A lot has changed. The HyperText Markup Language (HTML) for web pages is now usually dynamically assembled on the fly[6] and sent on demand to your browser. The pieces being assembled are themselves filled with dynamic content—news, inbox contents, and advertisements adjusted to your particular tastes. Even the addresses no longer identify a specific computer—content distribution networks route requests to any of thousands of computers all around the world. At a higher level, most web pages are served not from someone's home computer[7] but from a major corporation's social media platform or cloud computing service.

With all that's changed, some things have stayed the same, the core building blocks that are the essence of the web:

- The user uses a *user agent*, called a *browser*, to navigate the web.
- The web is a *network of information* linked by *hyperlinks*.
- Information is requested with the *HyperText Transfer Protocol (HTTP)* and structured with the *HTML document format*.
- Documents are identified by Uniform Resource Locators (URLs), *not* by their content, and may be dynamically generated.
- Web pages can link to auxiliary assets in different formats, including images, videos, Cascading Style Sheets (CSS), and JavaScript.
- All these building blocks are open, standardized, and free to use or reuse.

As a philosophical matter, perhaps one or another of these principles is secondary. One could try to distinguish between the networking and rendering aspects of the

[5] And the web *also* needed rich computer displays, powerful user-interface-building libraries, fast networks, and sufficient computing power and information storage capacity. As so often happens with technology, the web had many similar predecessors, but only took its modern form once all the pieces came together.

[6] "Server-side rendering" is the process of assembling HTML on the server when loading a web page. Server-side rendering can use web technologies like JavaScript and even headless browsers [6]. Yet one more place browsers are taking over!

[7] People actually did this! And when their website became popular, it often ran out of bandwidth or computing power and became inaccessible.

web. One could abstract linking and networking from the particular choice of protocol and data format. One could ask whether the browser is necessary in theory, or argue that HTTP, URLs, and hyperlinking are the only truly essential parts of the web.

Perhaps.[8] The web is, after all, an experiment; the core technologies evolve and grow. But the web is not an accident; its original design reflects truths not just about computing, but about how human beings can connect and interact. The web not only survived but thrived during the virtualization of hosting and content, specifically due to the elegance and effectiveness of this original design.

The key thing to understand is that this grand experiment is not over. The essence of the web will stay, but by building web browsers you have the chance to shape its future.

i.3 Real Browser Codebases

So let me tell you what it's like to contribute to a browser. Some time during my first few months of working on Chrome, I came across the code implementing the `<br>` [7] tag—look at that, the good old `<br>` tag, which I've used many times to insert newlines into web pages! And the implementation turns out to be barely any code at all, both in Chrome and in this book's simple browser.

But Chrome as a whole—its features, speed, security, reliability—*wow*. *Thousands* of person-years went into it. There is constant pressure to do more—to add more features, to improve performance, to keep up with the "web ecosystem"—for the thousands of businesses, millions of developers,[9] and billions of users on the web.

Working on such a codebase can feel daunting. I often find lines of code last touched 15 years ago by someone I've never met; or even now discover files and classes that I never knew existed; or see lines of code that don't look necessary, yet turn out to be important. What does that 15-year-old code do? What is the purpose of these new-to-me files? Is that code there for a reason?

Every browser has thousands of unfixed bugs, from the smallest of mistakes to myriad mix ups and mismatches. Every browser must be endlessly tuned and optimized to squeeze out that last bit of performance. Every browser requires painstaking work to continuously refactor the code to reduce its complexity, often through the careful[10] introduction of modularization and abstraction.

[8] It is indeed true that one or more of the implementation choices could be replaced, and perhaps that will happen over time. For example, JavaScript might eventually be replaced by another language or technology, HTTP by some other protocol, or HTML by a successor. Yet the web will stay the web, because any successor format is sure to support a *superset* of functionality, and have the same fundamental structure.

[9] I usually prefer "engineer"—hence the title of this book—but "developer" or "web developer" is much more common on the web. One important reason is that anyone can build a web page—not just trained software engineers and computer scientists. "Web developer" also is more inclusive of additional, critical roles like designers, authors, editors, and photographers. A web developer is anyone who makes web pages, regardless of how.

[10] Browsers are so performance-sensitive that, in many places, merely the introduction of an abstraction—a function call or branching overhead—can have an unacceptable performance cost!

What makes a browser different from most massive code bases is their *urgency*. Browsers are nearly as old as any "legacy" codebase, but are *not* legacy, not abandoned or half-deprecated, not slated for replacement. On the contrary, they are vital to the world's economy. Browser engineers must therefore fix and improve rather than abandon and replace. And since the character of the web itself is highly decentralized, the use cases met by browsers are to a significant extent *not determined* by the companies "owning" or "controlling" a particular browser. Other people—including you—can and do contribute ideas, proposals, and implementations.

What's amazing is that, despite the scale and the pace and the complexity, there is still plenty of room to contribute. Every browser today is open source, which opens up its implementation to the whole community of web developers. Browsers evolve like giant research projects, where new ideas are constantly being proposed and tested out. As you would expect, some features fail and some succeed. The ones that succeed end up in specifications and are implemented by other browsers. Every web browser is open to contributions—whether fixing bugs or proposing new features or implementing promising optimizations.

And it's worth contributing, because working on web browsers is a lot of fun.

i.4 Browser Code Concepts

HTML and CSS are meant to be black boxes—declarative application programming interfaces (APIs)—where one specifies *what* outcome to achieve, and the *browser itself* is responsible for figuring out *how* to achieve it. Web developers don't, and mostly can't, draw their web pages' pixels on their own.

That can make the browser magical or frustrating—depending on whether it is doing the right thing! But that also makes a browser a pretty unusual piece of software, with unique challenges, interesting algorithms, and clever optimizations. Browsers are worth studying for the pure pleasure of it.

What makes that all work is the web browser's implementations of inversion of control [8], constraint programming [9], and declarative programming [10]. The web *inverts control*, with an intermediary—the browser—handling most of the rendering, and the web developer specifying rendering parameters and content to this intermediary.[11] Further, these parameters usually take the form of *constraints* between the relative sizes and positions of on-screen elements instead of specifying their values directly;[12] the browser solves the constraints to find those values. The same idea applies for actions: web pages mostly require *that* actions take place without specifying *when* they do. This *declarative* style means that from the point of view of a developer, changes "apply immediately", but under the hood, the browser can be lazy

[11] For example, in HTML there are many built-in form control elements [11] that take care of the various ways the user of a web page can provide input. The developer need only specify parameters such as button names, sizing, and look-and-feel, or JavaScript extension points to handle form submission to the server. The rest of the implementation is taken care of by the browser.

[12] Constraint programming is clearest during web page layout, where font and window sizes, desired positions and sizes, and the relative arrangement of widgets is rarely specified directly.

[12] and delay applying the changes until they become externally visible, either due to subsequent API calls or because the page has to be displayed to the user.[13]

There are practical reasons for the unusual design of a browser. Yes, developers lose some control and agency—when pixels are wrong, developers cannot fix them directly.[14] But they gain the ability to deploy content on the web without worrying about the details, to make that content instantly available on almost every computing device in existence, and to keep it accessible in the future, mostly avoiding software's inevitable obsolescence.

To me, browsers are where algorithms *come to life*. A browser contains a rendering engine more complex and powerful than any computer game; a full networking stack; clever data structures and parallel programming techniques; a virtual machine, an interpreted language, and a just-in-time compiler; a world-class security sandbox; and a uniquely dynamic system for storing data.

And the truth is—you use a browser all the time, maybe for reading this book! That makes the algorithms more approachable in a browser than almost anywhere else, because the web is already familiar.

i.5 The Role of the Browser

The web is at the center of modern computing. Every year the web expands its reach to more and more of what we do with computers. It now goes far beyond its original use for document-based information sharing: many people now spend their entire day in a browser, not using a single other application! Moreover, desktop applications are now often built and delivered as *web apps*: web pages loaded by a browser but used like installed applications.[15] Even on mobile devices, apps often embed a browser to render parts of the application user interface (UI).[16] Perhaps in the future both desktop and mobile devices will largely be containers for web apps. Already, browsers are a critical and indispensable part of computing.

So given this centrality, it's worth knowing how the web works. And in particular, it's worth focusing on the browser, which is the user agent[17] and the mediator of the web's interactions, which ultimately is what makes the web's principles real. The browser is also the *implementer* of the web: its sandbox keeps web browsing safe; its

[13] For example, when exactly does the browser compute HTML element styles? Any change to the styles is visible to all subsequent API calls, so in that sense it applies "immediately". But it is better for the browser to delay style recalculation, avoiding redundant work if styles change twice in quick succession. Maximally exploiting the opportunities afforded by declarative programming makes real-world browsers very complex.

[14] Loss of control is not necessarily specific to the web—much of computing these days relies on mountains of other people's code.

[15] Related to the notion of a web app is a Progressive Web App, which is a web app that becomes indistinguishable from a native app through progressive enhancement [13].

[16] The fraction of such "hybrid" apps that are shown via a "web view" is likely increasing over time. In some markets like China, "super-apps" act like a mobile web browser for web-view-based games and widgets.

[17] The user agent concept views a computer, or software within the computer, as a trusted assistant and advocate of the human user.

algorithms implement the declarative document model; its UI navigates links. Web pages load fast and react smoothly only when the browser is hyper-efficient.

i.6 Browsers and You

This book explains how to build a simple browser, one that can—despite its simplicity—display interesting-looking web pages and support many interesting behaviors. As you'll see, it's surprisingly easy, and it demonstrates all the core concepts you need to understand a real-world browser. The browser stops being a mystery when it becomes code.

The intention is for you to build your own browser as you work through the early chapters. Once it is up and running, there are endless opportunities to improve performance or add features, some of which are suggested as exercises. Many of these exercises are features implemented in real browsers, and I encourage you to try them—adding features is one of the best parts of browser development!

The book then moves on to details and advanced features that flesh out the architecture of a real browser's rendering engine, based on my experiences with Chrome. After finishing the book, you should be able to dig into the source code of Chromium, Gecko, or WebKit and understand it without too much trouble.

I hope the book lets you appreciate a browser's depth, complexity, and power. I hope the book passes along a browser's beauty—its clever algorithms and data structures, its co-evolution with the culture and history of computing, its centrality in our world. But most of all, I hope the book lets you see in yourself someone building the browser of the future.

Links

[1] https://en.wikipedia.org/wiki/Web_page
[2] https://developer.mozilla.org/en-US/docs/Learn
[3] https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web
[4] https://en.wikipedia.org/wiki/Bulletin_board_system
[5] https://www.pcmag.com/encyclopedia/term/web-surfing
[6] https://en.wikipedia.org/wiki/Headless_browser
[7] https://developer.mozilla.org/en-US/docs/Web/HTML/Element/br
[8] https://en.wikipedia.org/wiki/Inversion_of_control
[9] https://en.wikipedia.org/wiki/Constraint_programming
[10] https://en.wikipedia.org/wiki/Declarative_programming
[11] https://developer.mozilla.org/en-US/docs/Learn/Forms/Basic_native_form_controls
[12] https://en.wikipedia.org/wiki/Lazy_evaluation
[13] https://en.wikipedia.org/wiki/Progressive_enhancement