History of Web Programming

Chapter · June 2014

DOI: 10.1007/978-1-4302-6482-8_20

1 author:

David Kopec
Champlain College
15 PUBLICATIONS **7** CITATIONS

SEE PROFILE

Dart for Absolute Beginners

David Kopec

Apress®

Dart for Absolute Beginners

ISBN-13 (pbk): 978-1-4302-6481-1

ISBN-13 (electronic): 978-1-4302-6482-8

President and Publisher: Paul Manning
Lead Editor: Ben Renow-Clarke
Development Editor: Chris Nelson and Douglas Pundick
Technical Reviewers: Matthew Butler and David Titarenco
Editorial Board: Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Louise Corrigan, Jim DeWolf, Jonathan Gennick, Jonathan Hassell, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Gwenan Spearing, Matt Wade, Steve Weiss
Coordinating Editor: Christine Ricketts
Copy Editor: Michael G. Laraque
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `www.springeronline.com`. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science+Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a Delaware corporation.

For information on translations, please e-mail `rights@apress.com`, or visit `www.apress.com`.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at `www.apress.com/bulk-sales`.

Any source code or other supplementary material referenced by the author in this text is available to readers at `www.apress.com`. For detailed information about how to locate your book's source code, go to `www.apress.com/source-code`.

APPENDIX B

History of Web Programming

To fully appreciate Dart's place in the Web development ecosystem and its purpose, it's necessary to understand a little bit about where that ecosystem has come from. This appendix is not about Dart; it's about what led up to Dart. It's about the languages that have been used for programming the Web throughout its history.

Note This appendix is not meant to provide reference-quality material. It contains speculation, suppositions, and opinion. The technologies/programming languages mentioned were built by incredibly smart and dedicated people who did not have the foresight of twenty-five years of the Web that we have today. Any slights pointed toward them must be read with that context in mind.

Client Side

In the beginning, the Web was static. A page was downloaded, and it didn't change. When Tim Berners-Lee released the first web browser in 1990, web pages consisted of text with a little styling and links to other web pages. It was a web of information—not of interactivity. One has to remember the technology of the time. A cutting-edge microprocessor in 1990 operated at 50 megahertz, and the average personal computer had a few megabytes of RAM. The speed of computers has improved exponentially since then.

Over time, the technologies available in web browsers for content presentation became more impressive. Images were added. The ability to play sounds was added. Browsers included the ability to extend themselves with plug-ins. CSS was introduced to improve styling. New HTML tags were introduced to improve layout.

Even as web pages became prettier, they still lacked the capabilities of native applications. Web browsers were in essence just HTML document viewers. They lacked the ability to respond to users in any kind of dynamic way. In order for the Web to become a platform that could compete as a venue for interactive applications, client-side browser programming languages were developed.

Java Applets

Java 1.0 was released in 1995 by Sun Microsystems,[1] and it made a big splash. It promised to be a new, memory-safe, secure, object-oriented, platform-independent programming language/software runtime with familiar syntax. A cool aspect of the new language/platform were so-called applets. Java applets are Java programs that run with a graphical user interface in a predefined rectangle on a web page.

[1]Oracle, "The History of Java Technology," `www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html`.

The promise of Java applets was that programs could be written once and run on any platform that had a Java runtime environment and a web browser. This write once, run anywhere philosophy aimed to make the distinction between running a program on a Mac versus on Windows (or on a Unix workstation) irrelevant. Netscape Navigator was the first web browser to support Java applets.

The reality in the late 1990s was that Java applets were slow to load, had inconsistent user interfaces, and suffered from incompatibilities after Microsoft bundled its own Java runtime environment with Internet Explorer (a lawsuit was brought over this), which had subtle differences from the runtime deployed by Sun. Java applets can still be run today on any web browser with a Java plug-in, but they're becoming increasingly rare. The lack of support for Java applets on mobile platforms like iOS has sped along their demise.

JavaScript

The introduction of JavaScript happened practically in parallel with that of Java applets. JavaScript also launched in 1995, and Netscape Navigator was the first web browser to ship with it. Indeed, JavaScript was the creation of Brendan Eich, a Netscape employee, and the story goes that he designed the whole language in ten days. JavaScript was not originally meant to be used for the same kind of applications as Java applets. It was supposed to be a lightweight scripting language. There had been previous attempts at providing scripting languages in a web browser, but none of them had the backing of a major player like Netscape or Microsoft until JavaScript. The name *JavaScript* was a marketing ploy to take advantage of the hype around Java. JavaScript is not based on Java in any meaningful way.

JavaScript was standardized as ECMAScript by Ecma International. JavaScript was popular enough that Microsoft incorporated a slightly modified version of it in Internet Explorer the following year. The incompatibilities between Microsoft's version of JavaScript and Netscape's version, despite the ECMAScript standard, plagues web developers to this day when they need to maintain compatibility with older versions of Internet Explorer (newer versions are much more standards compliant).

While JavaScript quickly gained popularity, for the first decade of its life it was utilized almost exclusively for what we would consider today to be quite small use cases. A major reason for this was its speed (or lack thereof). The original JavaScript engines were interpreted. It wasn't until the 2000s that JavaScript engines commonly included a just-in-time (JIT) compiler.

Over the past decade, as the speed of JavaScript engines increased, its features expanded, and its compatibility across browsers improved; it began to be used for increasingly significant applications. Browser apps increased in complexity to the point where they rivaled native software. JavaScript also increasingly became popular on the server (see the "Server-Side" section later in this appendix).

Today, JavaScript is the lingua franca of the Web. There are very few, if any, of the most popular web sites in the world that are not powered largely by JavaScript on the client side. By any measure, JavaScript is one of the most popular programming languages in existence. For a little more on JavaScript from a technical standpoint, check out Chapter 17's section on it.

VBScript

Microsoft once had a tendency to leverage its near monopoly status in the industry to push proprietary technologies that would encourage platform lock-in. VBScript was arguably one such proprietary technology. VBScript was added to Internet Explorer in 1996 as a kind of Microsoft ecosystem alternative to JavaScript. VBScript leveraged the Visual Basic name in a similar fashion to JavaScript leveraging Java's brand (in name only).

VBScript attempted to fill a niche similar to that JavaScript had—a lightweight scripting language for making web pages more interactive. Due to its proprietary nature, however, web sites with VBScript could not function properly in non-Microsoft web browsers. This was okay for a while, when Internet Explorer had about 90% market share in the early 2000s; however, proprietary technologies are considered antithetical to the ideals of the open Web.

VBScript lost out to JavaScript but found its way into some other Microsoft products. As of Internet Explorer 11, it has been deprecated as a scripting language for the web browser.[2] It hasn't been a real player on the Web for more than a decade.

Flash

Flash is a platform for interactive content that is most commonly used for games and streaming video. It actually extends beyond the Web, but we'll keep our discussion limited to the Flash web browser plug-in. Flash was originally developed in the 1990s by several different companies before coming under the guise of Adobe, after its 2005 acquisition of Macromedia. Flash is programmed with the ActionScript language. Interestingly, ActionScript is derived from ECMAScript, the same standard as JavaScript.

In a world where Java applets were too slow-loading, and JavaScript was too unsophisticated, Flash came to dominate the niche of web-delivered richly interactive experiences. The Flash browser plug-in was at one point installed on more than 90% of web browsers and commonly came bundled with new computers. Flash was everywhere.

Flash's big challenger came in the form of improved JavaScript engines and APIs. The smartphone revolution rendered another blow to Flash, as mobile web browsers didn't include plug-in support, most famously the original iPhone. Steve Jobs wrote an open letter in 2010 criticizing Flash as a proprietary, legacy technology whose time had passed.[3] The letter was controversial, but it ultimately signaled a shift in the industry away from Flash and toward HTML5 and JavaScript technologies.

However, Flash is still the main language used for web games and streaming video on laptop and desktop web browsers. There are JavaScript and HTML5 technologies that are making inroads, but the Flash plug-in will likely still be with us for years to come. With that said, it's probably not a good idea to develop new web applications in Flash.

Silverlight

Silverlight is another Microsoft technology that was seemingly developed as an answer to an industry de-facto standard. As VBScript was an answer to JavaScript, Silverlight was an answer to Flash. Microsoft Silverlight is based on the technologies of the .NET Framework, Microsoft's sophisticated software stack that underlies most of its modern platforms. Silverlight is delivered chiefly in the form of a web browser plug-in.

Silverlight applications can be programmed from several different programming languages, but the browser plug-in never saw widespread adoption outside of the Microsoft ecosystem. The notable exception has been Netflix, which plans to move away from Silverlight in a future version.[4] First released in 2007, Silverlight may have simply come too late to market to garner widespread adoption.

The official Silverlight browser plug-in is only available on Windows and OS X. Although a free software version of Silverlight is available that runs on Linux and other operating systems, as with Flash, you will be leaving iOS and Android users in the lurch, if you develop your site with Silverlight. Further, Microsoft's support for Silverlight appears to be waning. Stick with Dart and JavaScript.

Recent Developments

It seems a rule in web technology (if not all technology) that any platform that comes to dominate a space will quickly have newcomers attempting to displace it. When it comes to JavaScript, the strategy seems to be more one of acceptance of it as a standard, while attempting to work around it. *Embrace and extend* if you will.

---

[2]Microsoft, "VBScript is no longer supported in IE11 edge mode for the Internet zone," `http://msdn.microsoft.com/en-us/library/ie/dn384057(v=vs.85).aspx`.

[3]Steve Jobs, "Thoughts on Flash," `www.apple.com/hotnews/thoughts-on-flash`, 2010.

[4]Anthony Park and Mark Watson, "HTML5 Video at Netflix," `http://techblog.netflix.com/2013/04/html5-video-at-netflix.html`, 2013.

The chief JavaScript alternatives all compile to JavaScript. You obviously already know about **Dart**, Google's effort that launched in 2011. Dart is an entire alternative platform. **CoffeeScript** is a less ambitious effort. It's basically a set of syntactic sugar that makes JavaScript more readable and adds a few more advanced language constructs that reduce the size of an equivalent program (in terms of lines of code). Of course, the benefits of a syntax change are up to the whims of an individual. Some people will prefer CoffeeScript's syntax (which borrows from Ruby and Python), while others will prefer traditional JavaScript.

CoffeeScript has gained some traction. It is a popular language on GitHub and was even included in the 3.1 release of the popular web development framework Ruby on Rails.[5] CoffeeScript does not represent a paradigm shift. It represents an influential direction for client-side languages, as far as programmatic syntax is concerned.

Microsoft had an answer to JavaScript and an answer to Flash. Surely, it must have an answer to Dart… Of course it does! **TypeScript**, first appearing in 2012, is Microsoft's player in this space. Unlike Dart, TypeScript does not represent a completely separate platform but instead is a superset of JavaScript. In fact, many of TypeScript's features are expected to appear in the forthcoming JavaScript standard ECMAScript 6.

Like Dart, TypeScript aims to relieve the difficulty of cleanly building modern, significantly sized applications that live in the browser. Also like Dart, it includes support for classes and optional types. That's where the similarities end. TypeScript does not represent a new platform. It represents an improved environment for building JavaScript applications. There is no TypeScript VM. If Dart didn't exist, TypeScript would be a more compelling pure JavaScript alternative. But, as it stands, most of the features inherent in TypeScript will be included in the near future in JavaScript anyway. As a superset of JavaScript, TypeScript also inherits JavaScript's faults.

Although seldom mentioned in the tech media, **Haxe** is a rather remarkable language. It not only compiles to JavaScript but also to Flash, PHP, C++, Java, and more. It's truly a platform-independent language, if there ever was one. At the same time, its syntax is close to ECMAScript, so it looks familiar to JavaScript and Flash developers. Haxe first appeared in 2005. RedHat's experimental language, **Ceylon**, also compiles to JavaScript.

Server Side

The difference between server-side and client-side programming is delineated in Chapter 7. The diversity of solutions for server-side development far outstrips that of the client side. The alleviating factor that enables so many more possibilities for the server side is the lack of necessity to run in the browser. There are web frameworks available for just about every programming language in mainstream use. The job of a server-side web framework basically boils down to receiving a request, processing it (and any user-supplied parameters), and outputting HTML back to the web server, ready to be served to the user. The story of server-side web development has been a story of making this process (and everything that goes with the process step, such as database access) easier.

CGI

CGI stands for *Common Gateway Interface*. It's an interface, first developed in 1993, that allows a request sent to a web server to result in the execution of a program on the server. The content generated by the program can then be sent back to the client via the web server. A CGI program can essentially be written in any language. The interface was developed as a standard that all web servers could implement.

Although uncommon today, it wasn't unusual to see large CGI programs built in C or C++ in the 1990s. They were ubiquitous languages that offered tremendous speed at a time when computer hardware, even on the server side, struggled to keep up with the demands that were placed on it. The problem with C and C++ is that they're memory unsafe and don't have tremendous built-in facilities for text processing, which is the purview of much web work. These points make writing CGI programs in C or C++ arguably more difficult than in other languages. That didn't stop Amazon from building a large part of its original 1995 site in C.[6]

[5]Peter Cooper, "Rails 3.1 Adopts CoffeeScript, jQuery, Sass and… Controversy," `www.rubyinside.com/rails-3-1-adopts-coffeescript-jquery-sass-and-controversy-4669.html`, 2011.
[6]Brad Stone, *The Everything Store: Jeff Bezos and the Age of Amazon* (Boston, MA: Little, Brown, 2013), p. 36.

C and C++ were so widely used (outside the Web) that their syntax became the template for most of the languages used on both the client side and the server side of the Web in the 1990s, including JavaScript, Java, Perl, and PHP. Today, it's rare for a server-side web application to be developed in C or C++. However, a program built in another language may call a C/C++ library for performance reasons. An example would be a Python-based genetics research web app that calls C code for doing numerical analysis. The logic of building the app may be all Python, but the computationally intense bit will be C. For a little more information about C, see Chapter 17.

Many of the languages that became popular on the Web began by being utilized through CGI. Most of them later gained their own web server extensions that bypass CGI for performance reasons. CGI continues to be an option on modern web servers.

Perl

For many years, Perl was the glue that held together the Internet, operating through the Common Gateway Interface to power many early web sites, both large and small. It was the first higher-level language to be widely deployed for server-side web programming. Perl, created by Larry Wall in the late 1980s as a scripting language, actually predates the Web. Perl has extensive capabilities for processing text—a task critical in web development (HTML is just text, after all).

So, why didn't Perl maintain its popularity? One of Perl's slogans is "There's more than one way to do it." That's not always a good thing. Perl gained a reputation for being the language most likely to be used for writing cryptic, unreadable code. If everyone does things their own way, and individual style trumps common conventions, then that's a recipe for a lot of mutually difficult-to-read code.

Perl has been stuck in a versioning nightmare since a new redesigned version, Perl 6, was announced in 2000. Fourteen years later, Perl 6 continues to be in development, while Perl 5 has diminished in popularity. Larry Wall continues to lead Perl as its Benevolent Dictator for Life (BDFL). Several popular web sites continue to be developed in Perl, including Craigslist, which hired Larry Wall in late 2013.[7]

PHP

PHP began as a set of C CGI scripts that Ramsus Lerdorf used on his personal home page in 1994. It evolved into a templating language, and over time, a distinct programming language that somewhat resembles Perl. PHP lives side by side with HTML. It has the ability to be embedded in tags within an HTML document, which makes deployment as simple as uploading a PHP file to a web server. This simplicity of deployment is likely what rocketed PHP to become the most popular server-side web language among small-to-medium-size web sites. It's practically unheard of to sign up for a small business web host and not have the web server come with built-in PHP support.

PHP even powers the most popular content-management system in the world, WordPress. In fact, PHP is so popular, even today, that it gets mocked by developers from other language communities for the poor quality of the code developed with it. As a popular, easy-to-deploy, ubiquitous language, PHP attracts a wide swath of developers, including many just starting out. Naturally, if inexperienced developers make up a large set of your user base, code quality will suffer. It's in a similar situation on the server side as JavaScript on the client side, which is also sometimes mocked by other language communities.

Are these criticisms fair? The answer is largely no. High-performance, beautifully engineered code is written by top-tier talent in PHP every day. Facebook was largely written in PHP.[8] PHP is relatively high-performance compared to some of the other popular modern server-side offerings. PHP may have been the accident of one man's tinkering, and largely unplanned as a language, but it's clearly been successful for a reason: it's easy to get started with and relatively high-performance.

[7]Jim Buckmaster, "Artist. Formerly known," http://blog.craigslist.org/2013/10/15/artist-formerly-known.
[8]Brian Shire, "PHP and Facebook," www.facebook.com/notes/facebook/php-and-facebook/2356432130, 2007.

ASP

ASP was Microsoft's original entry into the server-side web technology war. ASP, which stands for *Active Server Pages*, could, in its original 1996 incarnation, most readily be compared to PHP. It had an emphasis on embedding logic in tags alongside HTML. With an emphasis on Windows Server operating systems, ASP never had a fighting chance of reaching PHP's level of ubiquity, given that the majority of web servers run Linux or other Unix-like operating systems. ASP was replaced by ASP.NET, a version tailored for Microsoft's multi-language .NET platform. Microsoft released a paper in 2003 titled "Migrating from PHP to ASP.NET," comparing PHP and ASP.NET, that may be interesting from a historical perspective.[9]

Java

The big excitement when Java launched surrounded using it for building write-once-run-anywhere desktop applications and Java applets. Over the years, Java did achieve dominance across a wide array of devices from mobile phones and credit cards to DVD players, but it never achieved prominence on the desktop outside of the corporate world. It did, however, find a surprising niche as the power behind many industrial-strength web infrastructures. J2EE (Java 2 Enterprise Edition), released in 1999, was the first major version of Java that explicitly targeted large-scale web infrastructure developers.

Sun Microsystems, the creator of Java, envisioned Java as an open specification that other corporations could implement. IBM, RedHat, and others created J2EE implementations that could run the same server-side programs that ran on Sun's J2EE application server. This created an ecosystem of industrial-strength enterprise-grade Java application servers backed by the world's biggest enterprise tech companies. Java flourished and became the preferred enterprise server-side technology.

JSP, or Java Server Pages, a part of J2EE, allows Java to be embedded with HTML similarly to PHP. Java continues to be widely deployed in the enterprise world for server-side stacks, due to its high performance, maturity, large pool of developers, and advanced libraries. Newer languages that run on the Java Virtual Machine (JVM—similar in principal to the Dart VM), such as Scala, Clojure, and Groovy, have their own popular web frameworks (Play is one, for example) that have the added benefit of being able to interact with legacy Java libraries. Further, versions of Ruby and Python (JRuby and Jython), two popular languages to be mentioned shortly, also run on the JVM.

For all of the reasons expounded in the last paragraph, the Java platform continues to be compelling. It probably has lost web developer mindshare, not so much because of its faults, but, instead, because what's come after it is so compelling. A concentration during the last decade on rapid application development frameworks with batteries included, such as Ruby on Rails and Python with Django, has captured the imagination of the industry, because these frameworks allow developers to build more quickly than with traditional solutions like Java.

Python

Python, like Perl, was originally created by a single developer as a scripting language for Unix-like systems. In this case, the year was 1991 (just four years after Perl), and the creator was Guido van Rossum. While Perl was successful on the server side of the Web almost from the beginning, it took until the 2000s for Python to become a mainstream server-side web language. For more about Python, the language, see Chapter 17.

Python's clean, easy-to-follow syntax and batteries-included philosophy makes it a great language for getting things done quickly. However, that didn't make it a great language for building web apps until web frameworks that matched the elegance of the core language became available. The most prominent Python web framework is by far Django. Django began life in 2003 for internal use at a newspaper and was released to the public initially in 2005. Django follows a classic and powerful software development paradigm known as MVC (Model View Controller), which emphasizes the separation of data from display.

[9]Microsoft, "Migrating from PHP to ASP.NET," `http://msdn.microsoft.com/en-us/library/aa479002.aspx`, 2003.

Django is not the only game in town. Python is somewhat notorious for the proliferation of so-called microframeworks. These are small web frameworks that don't necessarily include all of the features of a full-service framework like Django but make it very easy to get smaller web apps built quickly, due to their logical defaults and concise setup. The most popular microframework is probably Flask. Hello World in Flask (showing a web page that displays "Hello World") is just a few lines of Python.

If Python's syntax is so praised and it has such great frameworks available, then why isn't everyone using it for server-side web app development? Python's popular implementation has relatively poor performance compared to some of the other options available, which makes it unsuitable for some applications. Further, Python deployment is trickier for beginners than something like PHP, where a file with embedded HTML just gets dropped on a server. However, Python is still very popular today and probably doesn't compete as much with PHP developers as it does with Ruby developers. Also, similarly to Perl, Python has suffered from a botched version transition from Python 2 to Python 3 (no letters please!), although Python 3 has been gaining traction as of late, five years after its debut. Like Larry Wall with Perl, Guido van Rossum continues working as Python's BDFL.

Ruby

Ruby is a language first released by Yukihiro Matsumoto in 1995. Like Python, it features clean syntax and a thoroughly object-oriented design. Outside of Japan, Ruby failed to gain significant popularity until the release of the web framework Ruby on Rails (also known just as Rails). Ruby on Rails led to an explosion of interest in Ruby (mostly to use Rails), due to high developer productivity in the framework.

Ruby is coupled with a popular packaging system called RubyGems that is analogous to Dart's pub. The combination of a vibrant gems ecosystem with Ruby on Rails has allowed for the rapid development of many web apps. It is a pretty commonly held belief that an equivalent web application can be built with fewer lines of code with Ruby on Rails than with comparable web frameworks for other languages.

There's a lot of "magic" that goes on behind the scenes of a Ruby on Rails web application that allows just a few lines of code to provide an impressive array of functionality. This magic has been criticized, because it is thought that it leads to a situation in which novice developers build complex web applications without understanding fully what's going on behind the scenes. It is so easy with RubyGems to include an open source project that adds significant functionality to a Rails application that it is trivial. This notoriously leads to many poorly supported gems underlying the architecture of Rails applications.

Also like Python, Ruby's most popular implementation suffers from performance problems (indeed, even worse ones than Python's). Twitter was originally developed largely in Ruby on Rails but, due to performance and scaling considerations, had to move several pieces of its stack to JVM-based languages (Scala and Java) as it grew.[10] Ruby on Rails development offers many similar advantages and pitfalls to Python with Django. Choosing between the two has more to do with personal preference (with regard to language syntax and how the frameworks work) than any inherent advantage of one over the other. When you don't need to worry about large scale, as most beginning developers don't, both are excellent options for modern server-side web app development.

JavaScript

JavaScript has only become a popular server-side option since the advent of high-performance JavaScript VMs during the last decade. Netscape actually shipped a server-side JavaScript framework as early as the mid-'90s, but it was not really until the release of Node.js in 2009 that server-side JavaScript began to flourish. Node.js is a platform for building asynchronous web apps using JavaScript. One of the most popular frameworks for Node.js is Express.js.

[10]Alex Williams, "Once Again, Twitter Drops Ruby for Java," http://readwrite.com/2011/04/11/twitter-drops-ruby-for-java#awesm=~oCSaqvGb5QSBsn, April 11, 2011.

Why is JavaScript on the server side a good idea? The key benefit is the ability to use one language for both the client side and the server side. This familiarity arguably enables an environment in which code reuse is optimized and developer skillsets can more easily span both domains.

Node.js provides not just an application server but also a packaging environment. npm is a package manager, analogous to pub. Plain vanilla JavaScript does not provide a clear path for package management.

JavaScript on the server-side, and specifically Node.js, is seeing adoption in the corporate world. Microsoft helped port Node.js to Windows.[11] Walmart is utilizing Node.js to back its online store.[12] These are recent developments within the last three years. Clearly, server-side JavaScript has momentum.

Trends

The biggest trend in server-side web development of the past ten years has actually come from the client side. The advent of fast client-side JavaScript and the popularity of AJAX has allowed logic that was once exclusively the realm of the server side to be pushed into the browser. This has caused the focus of server-side development to be less on building full MVC applications and more on supplying data to be displayed by complex JavaScript front ends. It is not uncommon for a single server-side application, serving JSON, to deliver data for a web client built in JavaScript, an iOS client built in Objective-C, and an Android client built in Java. In fact, this may be the epitome of what's viewed as a "modern" architecture.

Another clear trend is use of the same programming language on the client and server. This is most obvious with JavaScript's Node.js, but there are other examples. Dart is one. There are several projects that put Python in the browser via compilation of Python to JavaScript (pyjs, Skulpt, PyPy.js). There are similar JavaScript compilers for Haskell, Clojure, and many other languages that are also used on the server side. The best known other-language-to-JavaScript compiler may be another Google-sponsored project, called Google Web Toolkit (GWT). Google Web Toolkit compiles Java to JavaScript.

Finally, one more trend has been language experimentation. The number of languages with viable server-side web frameworks seems to be ever-expanding. There are languages that are beautiful in their functional purity, like Haskell, and languages bent on simplicity, like Go. There are languages that thrive when put to the test of concurrency, like Erlang, and languages that are hyper-modern in their feature set, like F#. All of them are being used to build industrial-strength web applications that would've been built with PHP or Java a decade ago.

The overall focus of all of these trends is on simplifying the toolset and letting developers work in the environment they are most comfortable with. There's an incredible diversity of options available for server-side web programming today. With a lot of choice also comes a lot of fragmentation and little fiefdoms. Do you have a big server-side web app coded in Haskell? It's going to be a very specialized (and probably expensive) set of developers that can service it.

How do you pick a server-side language/framework in such a fast-moving landscape? Take advantage of these trends. If your application can move much of its logic to the client side, do so. Then you can code it largely in Dart. If you can simplify your life by using Dart on both the client and server, go for it! If the specialization of one of the many languages that have robust server-side frameworks available is good for your application, then take advantage of it, as long as its ecosystem looks viable going forward.

[11]Ryan Dahl, "Porting Node to Windows with Microsoft's Help," `http://blog.nodejs.org/2011/06/23/porting-node-to-windows-with-microsoft%25e2%2580%2599s-help`, June 23, 2011.
[12]Eran Hammer (Interview), "Node Black Friday at Walmart with Eran Hammer," `http://thechangelog.com/116`, January 11, 2014.

Where Does Dart Fit In?

Dart emerged at the end of these time lines, both on the client side and the server side. It incorporates a lot of great thinking from its predecessors. It's a balanced language. It's not necessarily overtly focused on a single niche in the same way that a Haskell or PHP might be. It's most comparable to JavaScript in that it runs both in the browser and on the server. However, syntactically it shares a great resemblance with Java.

Dart is meant to be an accessible language suitable for developing both small and large web apps. It contains facilities for rapid prototyping. Yet, it also has features that make building large applications easier than with JavaScript. At the same time, it's compatible with today's client-side infrastructure, because it compiles to JavaScript. Dart tries to take a Goldilocks position.

Dart has many of the modern features present in languages that have emerged over the past few years, yet its syntax is reminiscent of older languages. Dart is not the fastest language available, but it's faster than JavaScript (when run on its own VM) and the most popular implementations of Ruby and Python. Is a lack of specialization a problem for Dart when JavaScript is seen as good enough by many? Only time will tell.

Evolution of the Web Browser

Alongside the ever-changing landscape of client-side and server-side web programming languages has been a parallel history of browser development and enhancements to HTML/CSS. In the beginning (i.e., 1990), there was Tim Berners-Lee's original Web Browser for NeXTSTEP called WorldWideWeb. The HTML spec and that original browser didn't even support images, let alone CSS. Mosaic was the first widely used multiplatform web browser, and it was released in 1993 by a team at the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign.

Microsoft and Netscape Duke It Out[13]

Mosaic led to two spinoff browsers. Netscape Navigator was a commercial program released in 1994 and built by a team of ex-Mosaic engineers at the nascent Netscape Communications Corporation. As a commercial entity, the Netscape team had to purposely avoid using any of the old source code from Mosaic. Netscape quickly succeeded Mosaic as the browser of choice for early web users.

Microsoft, realizing the web platform was a threat to its dominance of desktop personal computing, licensed the technology behind Mosaic from an entity called Spyglass Inc., which in turn had licensed the source code to Mosaic from the NCSA. Microsoft released Internet Explorer 1.0 in 1995. Over several revisions during the next few years, Internet Explorer took significant market share from Netscape.

The period of Netscape and Internet Explorer competition (roughly 1995 to 2000) was one in which several client-side technologies previously mentioned were developed and popularized, including JavaScript, VBScript, Java, and Flash. Unfortunately, at least with regard to the former three, they led to significant incompatibilities between dueling feature sets in Netscape and Internet Explorer.

A JavaScript program written for Netscape would not necessarily run unmodified on Internet Explorer. Likewise, a Java program coded for Microsoft's Java Virtual Machine (JVM) would not necessarily run correctly in Netscape/Sun's JVM. A VBScript program wouldn't run in Netscape at all.

Even the HTML spec itself was up for dueling visions. There were tags that one browser supported and the other didn't. Even more common, a tag/style displayed in one browser could look very different on the other. Even today, it's important to check how a site looks in all of the popular browsers, but back then, it was also a matter of coding around the inconsistencies and building specialized versions of a program/site for each browser.

[13]For more on the early history of the Web, check out The Internet History Podcast at `www.internethistorypodcast.com`.

During this period, there were no major platforms that mattered other than personal computers, and they ran Microsoft Windows or Mac OS. Those were the main platforms that Netscape and Internet Explorer supported. There was no worry about supporting browsers popular on smartphones or tablets or game consoles (although early versions of browsers for all of these existed).

Microsoft's coupling of Internet Explorer with its dominant Windows operating system and proprietary version of the JVM led to lawsuits, but in the scheme of things, they didn't really matter too much for the average web developer. By the year 2000, Internet Explorer had won. Microsoft's browser attained more than 80% market share. Netscape was sold to AOL, and the source code to its browser was given to a not-for-profit foundation known as Mozilla.

Firefox Emerges from the Ashes of Netscape

Mozilla released an open source browser, based on the last version of Netscape, that made little headway in challenging Internet Explorer's hegemony. During the period of Internet Explorer's domination, it was not uncommon for sites to be coded for it to such a degree that they would not work in alternative web browsers. The first significant challenger to Internet Explorer came out of Mozilla in 2002. Firefox was a stripped down, more lightweight version of the open sourced Netscape components that gained a quick following in some international markets.

The interest in Firefox coincided with a reemphasis on open standards in the industry (including specifications like CSS and HTML 4). The goal was to avoid the fragmentation of the 1990s and ease the difficulty of web developers' day jobs. It didn't happen quickly. There was little incentive for Microsoft to move quickly to adopt standards to help erode its dominant position.

In 2003, Apple released its own web browser, Safari, only for Macintosh, and based on open source technology earlier used in the Linux world. The same year, Microsoft announced it would not release any new versions of Internet Explorer for Macintosh. Although Windows had more than 90% market share among personal computers, this did provide a further impetus for web developers to build sites based on open standards and cross-browser compatibility.

Mobile and a Revitalized Browser Ecosystem

There were versions of Safari for Windows that didn't make much of a dent in the market, but it was the version of Safari that came with the original iPhone in 2007 that really changed the game. Mobile web browsers had existed since the 1990s, but they were limited in functionality, due to the weak microprocessors and user-interface technologies that existed on mobile phones up to that point. The iPhone was the first smartphone to offer a full web browser experience that made web sites designed for desktop browsers easy to read on the go. And the iPhone's Safari depended on open standards.

Most important, the iPhone's Safari didn't support plug-ins. This meant no JVM (no Java applets) and no Flash. The iPhone was a sensation, and if web developers wanted to capitalize on its success, they would need to code to the open web standards (namely JavaScript, HTML, and CSS) that it supported. It was also the beginning of concern for making sites that worked well on many highly different screen sizes. While the iPhone could display sites written for big monitors that accompany desktop web browsers, sites looked better when their content was responsive to different screen sizes.

Google's Android followed in 2008 and largely reinforced the changes brought about by the iPhone. And these were yet further reinforced by the iPad and Android tablets. It's unusual today for a site not to be coded to look great on both mobile and desktop.

Google also released its own desktop web browser, Google Chrome, in 2008, based on the same open source project, WebKit, as Apple's Safari. Chrome's superfast JavaScript VM, V8, was a revelation. It enabled applications to be coded for the browser that were previously only done natively. It led to an arms race in JavaScript VMs between all of the major players—Mozilla, Apple, Microsoft, and Google—which ultimately pushed the state of the art on the Web forward.

Whole platforms emerged, based on browser technologies. Palm launched the ill-fated WebOS, which was ultimately scattered in pieces across HP and LG. Google launched an operating system called Chrome OS that has seen some success in low-end laptops. Mozilla launched Firefox OS, an operating system for smartphones targeted at emerging markets and low-end handsets.

By the mid-2010s, Microsoft was no longer dominant in the browser space. Web developers today care as much, if not more, about what Google, Apple, and Mozilla think about web standards as they do about Microsoft. On the other hand, there are still many older versions of Internet Explorer floating around corporate environments that need be coded to for major applications. Microsoft, whether as a result of a change in its corporate culture or a shifting competitive landscape, has embraced open standards in the past few years. The most recent versions of Internet Explorer are on the cutting-edge, standards wise.

HTML5, the current version of the HTML specification, embraces facilities for managing the changes that have occurred over the past decade with regard to the web development ecosystem. For instance, it includes facilities for displaying rich content like video, audio, and graphics that were once the purview of Java applets and Flash. It also has built-in support for displaying content in ways that is responsive to differing screen sizes.

Firefox and Chrome have adopted rapid release cycles that bring innovation to users as quickly as possible. At the same time, users are treated to constant upgrades with little discernable difference between sequential versions. Table B-1 summarizes the browser world today.

Table B-1. *Web Browsers Today*

	Internet Explorer	Firefox	Safari	Chrome
Developer	Microsoft	Mozilla	Apple	Google
First Appeared	1995	2002	2003	2008
Release Cycle	Traditional	Rapid	Traditional	Rapid
Rendering Engine	Trident	Gecko	WebKit	Blink (forked from WebKit in 2013)
JavaScript Engine	Chakra	SpiderMonkey	Nitro	V8
Open Source?	No	Yes	Engine Only	Based on open source Chromium project
Desktop Platforms	Windows	Cross-Platform	OS X	Cross-Platform
Mobile Platforms	Windows	Android, Firefox OS	iOS	Android, iOS
Other Platforms	Xbox			Chrome OS

There are other browsers too, such as Opera, which has mobile versions, desktop versions, and versions for specialty markets like the Nintendo Wii. Most alternatives to the big four use the same rendering engine as one of them and so can be expected to render things in a similar fashion. For example, Opera has adopted the Blink engine used by Google Chrome. Another example is the experimental browser included on some versions of Amazon's e-ink based Kindle e-readers, which runs an engine based on WebKit, the same engine used in Apple's Safari.

There are differences between the mobile and desktop versions of the big four. In other words, Internet Explorer for Windows Phone doesn't necessarily support all of the same technologies as Internet Explorer for desktop computers. More and more, the emphasis for new web sites has been on designing mobile first, as usership naturally moves to mobile devices.

Importance Today

Why does it matter how web apps were coded 20 years ago? It matters because there's been a clear progression, both on the server side and the client side, in web technology. Each technology exists today as a response to what came before it. PHP would likely never have existed if writing web apps in C wasn't cumbersome. Dart is a direct response to the difficulty of building large-scale applications in JavaScript.

The browser wars ultimately resulted in the adoption of open standards. That's arguably good for almost everybody (save the former dominance of Internet Explorer, perhaps). Web developers today largely don't need to worry about coding platform-specific hacks.

By understanding the history, it's possible to tease out trends that will continue to affect the industry. By understanding these trends, a developer can ensure that he's riding the wave of technology, instead of swimming against it. If you could see the direction the industry was going in ten years ago, you certainly would have been careful not to code a large application in VBScript. Today, that might be Flash.

There are right answers when choosing a web technology to build with. It's important to understand what your alternatives are. It's important to code with an eye toward the future and understand the mistakes that others have made in the past. The Web can be complicated, but it's generally getting less painful with each passing year.