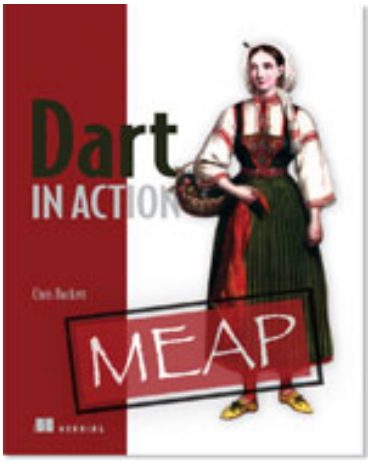
MANNING PUBLICATIONS



Dart in Action
By Chris Buckett

As a language on its own, Dart might be just another language, but when you take into account the whole Dart ecosystem, Dart represents an exciting prospect in the world of web development. In this green paper based on Dart in Action, author Chris Buckett explains how Dart, with its ability to either run natively or be converted to JavaScript and coupled with HTML5 is an ideal solution for building web applications that do not need external plugins to provide all the features.

You may also be interested in…

What is Dart?

The quick answer to the question of what Dart is that it is an open-source structured programming language for creating complex browser based web applications. You can run applications created in Dart by either using a browser that directly supports Dart code, or by converting your Dart code to JavaScript (which happens seamlessly). It is class based, optionally typed, and single threaded (but supports multiple threads through a mechanism called *isolates*) and has a familiar syntax. In addition to running in browsers, you can also run Dart code on the server, hosted in the Dart virtual machine.

The language itself is very similar to Java, C#, and JavaScript. One of the primary goals of the Dart developers is that the language seems familiar. This is a tiny dart script:

```
main() {                                #A
  var d = "Dart";                       #B
  String w = "World";                   #C
  print("Hello ${d} ${w}");             #D
}
```

#A Single entry point function main() executes when the script is fully loaded
#B Optional typing (no type specified)
#C Static typing (String type specified)
#D Outputs "Hello Dart World" to the browser console or stdout

This script can be embedded within `<script type="application/dart">` tags and run in the Dartium experimental browser, converted to JavaScript using the Frog tool and run in all modern browsers, or saved to a `.dart` file and run directly on the server using the dart virtual machine executable.

Dart is more than just the language, though. There is a whole ecosystem of projects, including the runtime environments, multiple runtime environments, editor tools, and comprehensive libraries, all designed to make developers lives better when developing complex web applications.

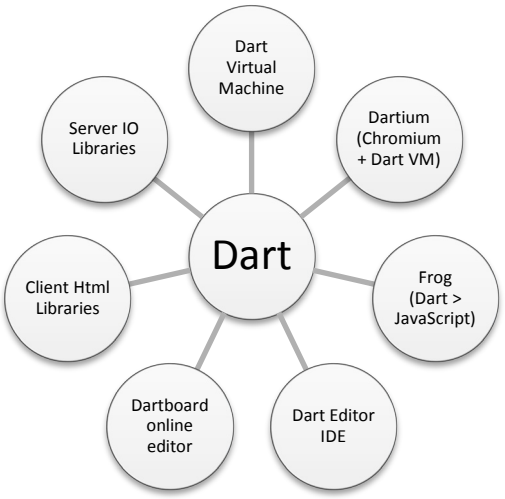


Figure 1 Dart is more than just the language. The Dart project has a whole ecosystem.

One of the stated aims of Dart is to make developers lives better when developing complex web applications. To understand how developers' lives *could* be made better, need to look at the reason for Dart's existence.

Finding a language for developing browser applications

Dart was created by Google, a company well known for developing complex large-scale web applications. One of the biggest hurdles to developing these kinds of applications is maintaining and documenting a large code base over a number of developers who may also be separated geographically. One of the biggest assets when creating complex applications in this way are automated tools, such as IDEs, continuous build systems, and static analysis. These tools help to flag up errors early on, when it is cheap to fix.

Flexibility has a price

Imagine you check out of the source control system a library that your code uses. A different developer has modified a method signature from `sum(a, b)` to `sum(a, b, c)`. In Java or C# (for example), your tools would immediately alert you to this fact. If your code had already been committed, the continuous build system would have alerted the other developer that their change had broken code elsewhere.

This is great for server-side application programming, but, imagine that the `sum(a, b, c)` function was a JavaScript function. It would still be valid to call it with `sum(1,2)` or `sum("Hello", "World")` or some such. Is this still correct? In fact, you could have added the third argument as an optional parameter, which would mean that these calls would be correct. Unfortunately, the only way you would know is to read the documentation on that method and hope that the developer updated it before they committed.

Potential solutions for building structured web applications

A number of projects from a variety of parties addressed this-client side web development problem by bringing to the browser the kinds of tools that server side developers have been used to for years. Whether it was by using C# in a Silverlight plugin, ActionScript in Flash plugins, or a variety of languages that have had JavaScript converters written for them (including the popular Google GWT that converts Java to JavaScript), the problem is well known.

Dart enters the scene

In October 2011, Google launched a technology preview of the Dart language. It consisted of a server-side virtual machine, and a cross-compiler to convert Dart into JavaScript so that it could run in the browser (known as Frog). Although much of the developer community poured scorn on the fact that an un-optimized "Hello World" Dart application converted to 9 MB of JavaScript (the "Hello World" JavaScript application also included the complete Dart runtime converted to JavaScript), the main aim of this technology preview was to gain feedback on the

language itself. The view within Google was that it was better to get the language out into the open well before it was *done* and a number of months were spent gathering input from the developer community, especially in the areas of reflection, concurrency, and library management.

With experience from the GWT project, as well as having access to developers from the high performance Chromium JavaScript engine, and recruiting some Java library design experts, Google has not stopped at the language design itself. Google has developed the rest of the ecosystem tools (such as the Eclipse-based Dart Editor) and runtime environments (such as the experimental Dartium browser) to provide a rich developer experience and allow running Dart applications natively in the browser (without converting to JavaScript). Core libraries for both the client side and the server side provide aids to development, including a rewriting of the browser DOM manipulation APIs to make them natural to work with in the web browser in Dart.

Flexible like JavaScript, yet structured

One of the key design decisions that were made with Dart was that it should appeal to both JavaScript and Java/C# developers.

This poses an interesting problem—Java and C# developers are generally comfortable with type systems, classes, inheritance, and so on. JavaScript developers, on the other hand, can range from UI designers who copy and paste code to add some interactivity to a webpage (and will never have used a type) to seasoned JavaScript programmers who understand closures and prototypical inheritance. To help with this, Dart has Optional Typing, which allows developers to specify absolutely no types (by using the `var` keyword, as in JavaScript), or use types everywhere (such as `String`, `int`, `Object`), or any mixture of the two. By using types, you are providing documentation to both tools and humans, but, whether you use types or not, this does not affect the running of the application.

Dart for complex in-browser web applications

Although Dart code can be used as a direct replacement to JavaScript by embedding it directly into html script tags to add pieces of interactivity to individual web pages, Dart was created by Google for building large-scale, complex web applications. The *single page application* typifies this; the application code for the entire application (or at least all the use cases for a major portion of the application) is loaded into a single web page. Think of Google Plus, Google Instant Search, and Google Analytics, to name but a few. Before we move on to look at Dart further, it is helpful to understand this context.

Introducing the single page application

The single page web application is becoming the de-facto standard in modern applications. It can provide a rich experience that users expect from the web today, on all systems with a web browser including smartphones and tablets. A typical modern application exhibits the following attributes. It:

1. Is served over the web.
2. Hosted in a browser.
3. Executes in a single page.
4. Uses the web as a data source.

There are many examples of single page applications, such as the Google Instant Search homepage, Gmail, and Google Reader. These single page applications require a large amount of client-side JavaScript and a fast browser engine in which to execute the script. Traditional web sites (for example, Wikipedia) are great for viewing documents linked to other documents but less good when you are trying to get a series of tasks done because there is a break in the applications flow as it transitions between pages while each page is sent from a client to the server and back.

Good candidates for becoming single page applications could be Amazon or EBay—rather than navigating from page to page to view each product, you could simply browse, select, and buy within the same single page.

In addition to providing a better experience for users, it also makes sense to utilize the processing power available on the users' computers. Instead of having the server load up the data and then combine that data with a

view template (using some kind of server-side framework (such as JSP, MVC, and so on), why not take advantage of the spare capacity of the users' computer to do the combining of data and view? This will free up resources on the server to allow it to serve more clients. Figure 2 demonstrates the traditional model.

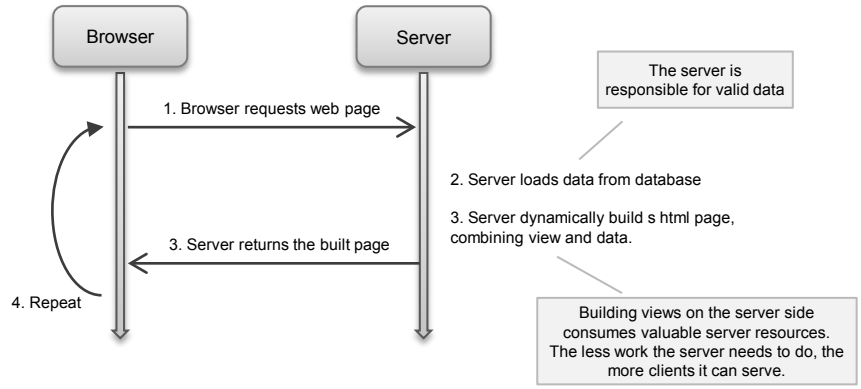


Figure 2 The traditional server model utilizes processing power on the server to combine data and views.

Single page applications allow all the application code to be loaded into the browser first, in a *bootstrap* phase. Once the browser has started running the application code, it then uses the web server as a data source to feed data into the application as required, as shown in figure 3.

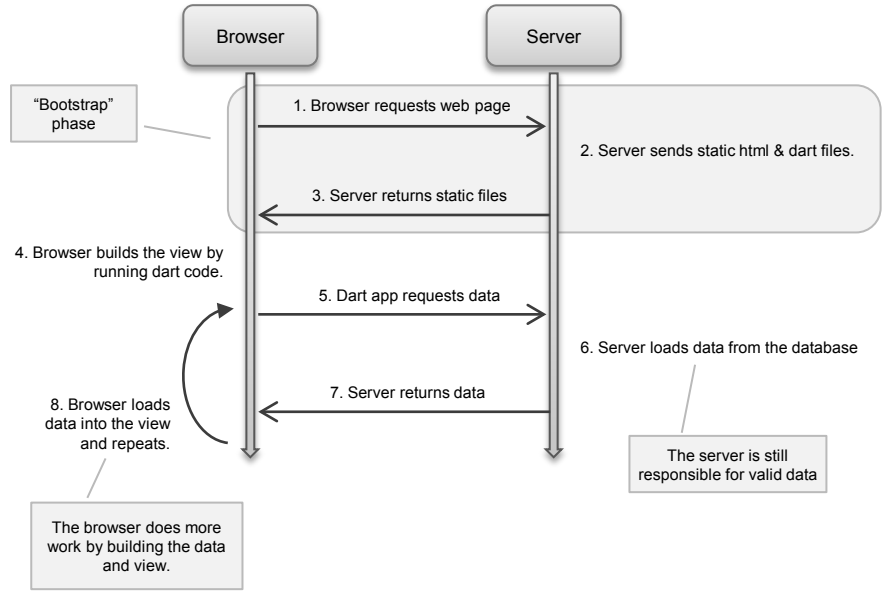


Figure 3 The single page application runs in the browser, only requesting data from the server.

This allows the client to take on more of the work that the server used to perform, which is good for you because it means your server can serve more customers. With HTML5 browser storage technologies, applications can also cache data in the browser to improve application performance further or even allow the user to work offline.

JavaScript is too flexible

Building a single page application requires moving a lot more code onto the client side and this, invariably, means JavaScript. JavaScript is a great language; it is easy to learn, can provide useful scripting in a web page, and, with the advent of frameworks such as jQuery, it can be pushed to limits never considered a few years ago.

Moving a large amount of business code from server side to the client side is not without its problems, though. Traditional server side languages such as Java and C# have allowed developers to work in teams with rich IDEs supporting refactoring and static analysis, unit test frameworks to ensure code quality, and continuous integration servers to verify that, when I commit my code, it does not break my neighbor's code.

Unfortunately, JavaScript is so flexible that it can be too flexible for some of this tooling to be effective. Coming to JavaScript from server-side languages can often seem like a step backwards in terms of tooling, especially when using code from more than one developer (this includes any third party library that you might use, as well as other members in your team).

In a trivial example of JavaScript's flexibility, a developer can write the following JavaScript function:

```
//add two numbers together, returning a number
function sum(a,b) { return a + b; }
```

and inform the team (or third-party library users) that the function is available for use. You can then use this function by writing `sum(1,2);` and expect to get the value 3 returned. There are a number of problems with this, though:

- You can also write `sum("Hello ", "World");`.
- You can also write `sum(1,2,3,4,blah);`.
- The developer can change the function signature to read function `sum(a,b,c)`.
- The developer can change the return type of the function to return something other than a number.

This is not good news—all of the above would still be valid JavaScript; you will only find the error when you run your application (or at least run unit tests). Even worse, the function comments are the only indication that the function even takes two numbers, and we all know how up to date our function comments are!

Types aren't always needed (but they can help)

There have been numerous attempts to introduce server side tooling into JavaScript. Google went around the problem by creating Google Web Toolkit (GWT), which allows development of JavaScript applications in Java, with a final compile step, which converts the Java into JavaScript. This meant that you get the rich tooling and test frameworks that you used on the server side without having to worry about JavaScript—if your Java GWT code integrates into the build, the JavaScript will be the output. GWT has been successful in its aim, with several notable projects being written with GWT, such as Chrome's Angry Birds, Google Groups, and Blogger.

A number of other languages have had JavaScript converters written for them at various points in the past, allowing you to develop in Python, Ruby or another language of your choice, which also allows you to take advantage of some of the tooling offered by those languages.

New languages, such as CoffeeScript or HaXe, were created with conversion to JavaScript as a design feature. These languages provide lighter weight or more cohesive syntax than JavaScript directly, while also making it easier to adhere to best practices (such as not using variables in global scope). Dart also has this design feature—everything that can be written in Dart can be converted to JavaScript.

Dart is solving problems for web developers

Dart is an addition to this list of new languages since one of its targets is direct conversion to JavaScript and you could use Dart in just such fashion, as an alternative JavaScript language. Google, however, has a bigger aim for Dart, and that is embedding the Dart Virtual Machine directly in the browser, much like the Google V8 JavaScript engine, the Dart VM now exists in Chromium (the open-source version of Chrome), meaning that you can run Dart directly in the Chromium browser without converting to JavaScript, for example:

```
<html>
  <body>
    <script type="application/dart">          #A
```

```
        #import("dart:html");                    #B

        main() {                                 #C
           var para = new Element("p");          #C
           para.innerHTML = "Hello World";       #C
           window.body.nodes.add(para)           #C
        }                                        #C
    </script>
  </body>
</html>
```

#A Dart mime type to invoke the Dart virtual machine
#B Importing an external library
#C The Dart main() function is the entry point of the script

The above html page contains embedded Dart, as denoted by the `application/dart` mime type. You can also link to external script files to remove the need to embed your script directly in the page (in the same way as JavaScript).

Starting with a single point of entry

Each Dart script has a single entry point function, called `main()`, which is the first function that is executed by the Dart VM. This means that you can rely upon all code that defines an application when main function is called—you cannot define and execute a function within running code as you can with JavaScript—there is no `eval()`.

One benefit of having Dart directly available within the browser is that it becomes as easy to write Dart code and see it running as it is to write JavaScript. Simply edit your file and refresh the browser. Even if you have to convert your application to JavaScript to support Internet Explorer, you will not lose the benefit you have when writing and debugging your application against a Dart-enabled browser.

A choice to use dart

Google has always been open about the fact that Dart was not designed to replace JavaScript, and Google as an organization will continue to push for JavaScript evolution. Dart, however, is another option available to web application developers (including Google), who want to develop applications for the web and mobile platforms, with the main competition for Dart being the fragmented mobile platforms rather than JavaScript.

Dart performance is a feature

By embedding the Dart VM directly in the browser, it is also possible for Google to make the Dart VM more performant than the equivalent JavaScript V8 engine due to the nature of the differences between the languages. A Dart application is fully known to the VM at application startup, which means that there are number of assumptions and optimizations that the VM can make to improve performance. The Dart VM also has the ability to snapshot the application to improve startup times. JavaScript VM's cannot provide these benefits due to the dynamic nature of JavaScript—it is possible to modify the executing JavaScript code whilst the code itself is running.

Not only a language for the browser

While Dart has been designed to run in the browser, the server side has not been forgotten. The Dart VM can also run on the server and has a number of other APIs available to enable communication with the file system, TCP sockets to communicate with other systems such as databases, and other server side APIs.

This is similar to Node.js, which is a fork of the Google V8 JavaScript engine that allows JavaScript developers to write JavaScript targeted on the server. Dart has been created with this in mind from the start, with all the APIs that you need to write server side code, from simple scripts to database drivers and web servers.

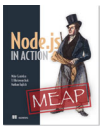
Summary

We have covered a lot, so here is a recap:

- JavaScript is great for web development, but too flexible complex applications
- The "single page application" is a growing web development pattern
- A type system can greatly aid the development process amongst teams of developers

- It can be helpful to use the same language on the server as in the browser.

Here are some other Manning titles you might be interested in:



Node.js in Action
Mike Cantelon, TJ Holowaychuk and Nathan Rajlich

Third-Party JavaScript
Ben Vinegar and Anton Kovalyov

HTML5 in Action
Robert Crowther, Joe Lennon, and Ash Blue

Last updated: May 11, 2012