The power of the Dart programming language for modern, high-performance web application development

Moć programskog jezika Dart za razvoj modernih veb aplikacija visokih performansi

Tamara Ranisavljević[1], Aleksandar Šijan[2], Luka Ilić[3]

[1] *Fakultet za primenjeni menadžment, ekonomiju i finansije, Univerzitet Privredna akademija u Novom Sadu, Jevrejska 24, Beograd, Srbija, tamara.ranisavljevic@gmail.com*

[2] *Fakultet za primenjeni menadžment, ekonomiju i finansije, Univerzitet Privredna akademija u Novom Sadu, Jevrejska 24, Beograd, Srbija, aleksandar@mef.edu.rs*

[3] *Fakultet za primenjeni menadžment, ekonomiju i finansije, Univerzitet Privredna akademija u Novom Sadu, Jevrejska 24, Beograd, Srbija, luka.ilic@mef.edu.rs*

Abstract: *Dart is an open-source and general-purpose programming language. It was originally developed by Google and later approved as a standard by the ECMA organisation. It is a new optimised programming language designed to develop fast applications on any platform. Dart's primary goals include making the language look familiar and being able to produce complex, high-performance applications. Dart was created as a programming language that combines two well-known worlds, JavaScript and Java or C#, and as such has the dynamics of JavaScript with the power and structure of languages like C# or Java. The goal of this paper is to present different aspects and features of this additional and modern programming language. The goal of this paper is to present the main features that make Dart an extremely powerful and attractive web application development solution for developers, while revealing the further potential of this young and modern programming language.*

Keywords: *Dart, JavaScript, Java, C#, web development*

Apstrakt: *Dart je programski jezik otvorenog koda i opšte namene. Prvobitno ga je razvio Google, a kasnije ga je kao standard odobrila ECMA organizacija. To je novi optimizovani programski jezik dizajniran za razvoj brzih aplikacija na bilo kojoj platformi. Primarni ciljevi Darta uključuju da jezik izgleda poznato i da bude u stanju da proizvodi složene aplikacije visokih performansi. Dart je kreiran kao programski jezik koji kombinuje dva dobro poznata sveta, JavaScript i Java ili C#, i kao takav ima dinamiku JavaScript-a, sa snagom i strukturom jezika poput C# ili Java. Cilj ovog rada je predstavljanje glavnih karakteristika koje Dart čine izuzetno moćnim i programerima atraktivnim rešenjem za razvoj veb aplikacija, istovremeno otkrivajući dalji potencijal ovog mladog i modernog programskog jezika.*

Ključne reči: *Dart, JavaScript, Java, C#, veb razvoj*

Introduction

Dart is an open-source, structured and flexible programming language, primarily oriented towards web development, but not exclusively. Dart is also a general-purpose object-oriented programming language with C-style syntax and was developed by Google in 2011. The purpose of Dart programming language is to create front-end user interfaces for web and mobile applications .

One of the primary goals for Dart was to make the language seem familiar. Therefore, the development of the programming language was inspired by other programming languages such as JavaScript, Java and C#. This new programming language is designed to make life easier for

developers, allowing them to develop more complex web applications with better maintainability and improved performance.

Dart supports the most common programming language concepts such as classes, interfaces, functions and is type safe language. On the other hand, it does not support arrays directly. It supports collections used to replicate data structures such as strings, generics, and optional types.

Behind Dart programming language are Lars Buck and Kasper Lund, two world-renowned Danish computer scientists and the authors of the V8 JavaScript engine for Google Chrome. Although Lars and Kasper started the project and continue to lead it, there are many pivotal players on the Google team. The third author worth mentioning who was also involved in the creation of Dart is Gilad Bracha. He is the author of the Newspeak programming language, the co-author of the Java specification and the second edition of Java Virtual Machine Development. Considering these three leading authors, it is not surprising that language performance and efficiency are two factors that were very important during the development of this language.

Adoption of the Dart programming language

Dart has many modern features found in languages that have emerged in the last few years, but its syntax still resembles older languages. The Dart is the best option for really large web projects with a special emphasis on the client side. This programming language allows better code organisation and can be much easier to maintain than other programming languages, such as JavaScript (Kopec, 2014).

One of the key design decisions was to make Dart familiar to both JavaScript and Java or C# developers. This design helps developers new to Dart to quickly pick up the language. The idea is that if developers are familiar with these other languages, they will be able to read and understand the purpose of the Dart code without much trouble (Hassan, 2020).

Java and C# developers are generally comfortable with type systems, classes, inheritance, and other such concepts. On the other hand, JavaScript developers range from UI designers who copy and paste code to add interactivity to a web page to experienced JavaScript developers who understand closures and prototype inheritance (Hassan, 2020).

To help with this diversity of developers, Dart has an optional typing feature, which allows developers to specify absolutely no types, using the var keyword, as in JavaScript, or to use type annotations, such as String, int, Object. But Dart also allows using any combination of these two approaches (Kopec, 2014).

By using type information in code, a developer provides documentation of its intent, which can be useful to both automated tools and fellow developers. A typical workflow when building a Dart application is to build type information incrementally as the code takes shape. Adding or removing type information does not affect how the code runs, but it allows the virtual machine to validate code more efficiently. This allows Dart's type system to bridge the gap between JavaScript's dynamic type system and Java and C# static type systems (Belchin & Juberias, 2015).

The features of Dart programming language

For the web application client side, Dart code is compiled to JavaScript code, so that applications developed in Dart can run on almost all modern browsers, as well as mobile devices that support JavaScript. Also included is a library based on the HTML 5 Document Object Model draft specification, so it's intended to be well-suited for rich front-end applications. On the server side, Dart applications can be run in a specific Dart virtual machine, enabling the creation of web server-specific applications such as those currently written in JavaScript and Node.js (Kopec, 2014; Dart Documentation).

Therefore, Dart emerged at both the client side and the server side. It incorporates a lot of great thinking from its predecessors. It is a balanced language and is not necessarily overly focused on one niche in the same way that, for example, PHP might be. It is most comparable to JavaScript in that it works both in the browser and on the server. However, syntactically it has a lot of similarities with Java (Hassan, 2020; Mohanty & Dey, 2014).

Dart is intended to be an accessible language suitable for developing small and large web applications. Programs can start small and grow over time with the support for top-level functions, classes and libraries. It contains facilities for rapid prototyping. However, Dart also has features that makes it easier to build larger applications than JavaScript. At the same time, it is compatible with today's client-side infrastructure because it compiles to JavaScript (Mezzeti, et al., 2016; Dart Documentation).

This language has its own ecosystem of tools, which includes multiple runtime environments, language and editor tools, and comprehensive libraries. They are all designed to improve the developer's workflow when building complex web applications. It can be said that the key tool for Dart developers is Dartium, which allows writing or editing Dart code and seeing it running by loading the file and refreshing the browser. Dartium combined with the Dart Editor provides the additional benefit of round-trip debugging (Kopec, 2014; Swathiga, et al., 2021).

Wide array of libraries includes built-in types and fundamental features such as collections, dates and regular expressions. Dart has built-in library support for files, directories, sockets and even web servers. Programmers code import a library, and libraries can be re-exported leading to code sharing (Dart Documentation).

Dart is an asynchronous programming language, which means it supports multithreading. The asynchronous programming is made in Dart thanks to Future API, which allows running tasks and processes in the background without waiting to get the results of these processes. The language also supports safe, simple concurrency execution of code with isolates. The isolates are the independent entities that are related to threads. This concurrency model allows for parallel execution. Communication happens by sending messages over ports (Mohanty & Dey, 2014).

In Dart, the isolate is the unit of work. It has its own memory allocation, which helps with the provision of an isolated security model. Sharing memory between isolates isn't allowed. Each isolate can pass messages to another isolate. When an isolate receives a message, which might be some data to process, an event handler can process that message in a way similar to how it would process an event from a user clicking a button. Within an isolate, which passes a message, the receiving isolate gets a copy of the message sent from the sending isolate. Changes to the received data aren't reflected on the sending side, which means it is necessary to send another message back (Kopec, 2014; Mezzeti, et al., 2016; Dart Documentation).

Dart Virtual Machine

Dart produces very readable code and works in major browsers. Applications created in Dart can be executed either by using a browser that directly supports Dart code or by compiling the Dart code into JavaScript (Mezzeti, et al., 2016).

Dart code can be converted to JavaScript using the dart2js compiler. This means that Dart applications can run in all modern web browsers. Moreover, it can be hosted in a Dart Virtual Machine, allowing both the client and server parts of applications to be coded in the same language. Dart supports all primary operating systems such as Windows, Linux, Macintosh, etc. Its own virtual machine is responsible for running Dart code on every operating system. In addition to running Dart code on the web, Dart code can also be run on the command line. Dart Virtual Machine is a managed runtime platform on the computer. Running code on a Virtual Machine is simple, because the emulated machine is a specific computer system which means the code does not have to be recompiled for each

computer, but rather compiled only once for the emulated machine. In this sense Dart Virtual Machine acts as an abstract layer between the real machine and the emulated machine (Belchin & Juberias, 2015; Mohanty & Dey, 2014; Dart Documentation).

Dart code run by Dart Virtual Machine is twice as fast as JavaScript. Figure 1 demonstrates the Tracer test and proves that Dart code compiled to JavaScript is far faster than native JavaScript code running on a V8 machine (Belchin & Juberias, 2015).

.

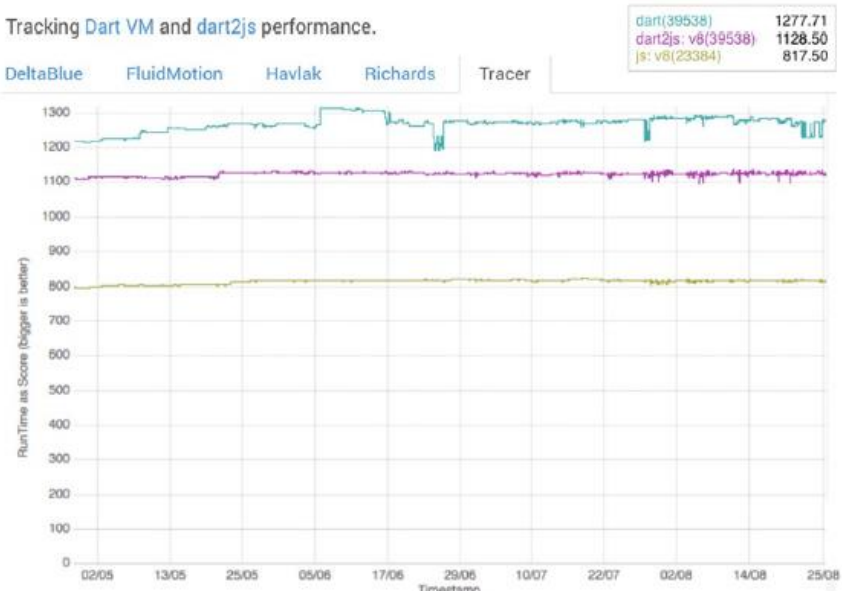

Figure 1. Dart Virtual Machine, dart2js, and JavaScript V8 performance
Source: Belchin & Juberias, 2015

In general, single-page applications use a fast client-side virtual machine to move processing from the server to the client. This allows the server to serve more requests, because the processing involved in building the layout is moved onto the client. By using Dart's HTML libraries to incorporate modern HTML5 browser-storage and caching technologies, applications can also cache data in the browser to improve application performance further or even allow users to work offline (Mohanty & Dey, 2014).

Optional typing feature

Dart uses optional and dynamic types, meaning that the programmer can choose to specify the type of variables and parameters or leave it as dynamic. It can be said that Dart is the type safe language, which means it uses both static type checking and runtime checks to confirm that a variable's value always matches the variable's static type, sometimes known as the sound typing. Although types are required, type annotations are optional because of type interference. This makes the code more readable. Another advantage of a type-safe language is that when there is a change in a part of the code, the system warns about that modification that has taken place (Bucket, 2013; Dart Documentation).

One of the key differences between JavaScript and Dart is that Dart has the concept of types baked into the language. Fortunately, by using Dart's option typing, the developer can get the benefit of strong typing through type annotations. Optional type annotations are used in variable declarations, for function parameter definitions and return types, and in class definitions. The following code snippet shows four ways of declaring the string variable message (Mohanty & Dey, 2014).

The first two have no type annotations, and the second two provide the String type annotation, indicating to developers and tools that you intend a string value to be used in the variable.

```
1  var messageA;
2  var messageB = "Hello Dart world!";
3  String messageC;
4  String messageD = "Hello Dart world!";
```

Adding this type of information doesn't change the running of the Dart application, but it provides useful documentation that tools and the Dart Virtual Machine can use to validate the code and find type errors. Dart can be said to be documentary typed because the code will run the same without the types. Any type information provided can help the tools during static analysis (Bucket, 2013; Dart Documentation).

Class-based structure

Dart uses classes and interfaces in a traditional and unsurprising object-oriented way. It supports single inheritance and multiple interfaces. All Dart classes inherit by default from the Object class. They can have public and private members, and a useful getter and setter syntax allows using fields interchangeably with properties without affecting users of the class (Swathiga, et al., 2021).

Dart is a lexically scoped language, which means that the scope of variables is determined statically, simply by the layout of the code. Everything can be placed in a variable is an object, and every object is an instance of a class. Even numbers, functions, and null are objects. The developer can pass around functions in Dart as objects, in a manner similar to in JavaScript. It is possible to pass a function as a parameter, store a function in a variable, and have anonymous functions to use as callbacks (Ecma International, 2015; Dart Documentation).

Dart has the ability to break source code files into logical structures. It's possible to write an entire Dart application in a single .dart file, but doing so doesn't make for great code navigation or organisation. To address this issue, Dart has libraries baked into the language (Swathiga, et al., 2021).

A library in Dart is a collection of source code files that could have been a single file but have been split up to aid human interaction with the code. A library is defined using the library keyword, imports other libraries using import, and refers to other source files using part, as shown in the following listing (Dart Documentation).

Libraries can pull a group of source files into the same scope. A library can be made up of any number of source files, including zero, and all the source files put together are equivalent to having a single library file containing all the separate files' code. As such, each source file can reference code that's in another source file, as long as both source files are part of the same library (Ecma International, 2015).

Dart consists of many useful built-in libraries including Software Development Kit, core, math, asynchronous, convert, html, IO, etc. It also provides the facility to organise the Dart code into libraries with proper name spacing (Dart Documentation).

String interpolation

Usually, strings are used in many places throughout web applications. The fact is that creating strings with variables makes a programmer's life much easier. So, it's no surprise that the language supports string interpolation. String interpolation is a process of evaluating the final string by injecting a variable or an expression in a string literal (Swathiga, et al., 2021).

Dart provides a number of ways for you to convert expressions into strings, either via the toString() function that's built into the base Object class or by using string interpolation. Dart internally calls

toString() on objects that are being interpolated. String interpolation uses the $ character or ${ } expression within single or double quotes. When a programmer wants to convert an expression to a string, then he can use a variable name with a $ prefix, such as $name. A Dart String object contains an array of UTF-16 code units (Bucket, 2013).

The main() function

Each Dart script has a single entry-point function called main() that is the first function executed by the Dart Virtual Machine. Thus, developers can rely on all code that defines an application when the main function is called. The main() function returns void and has an optional List<String> parameter.

This single feature helps writing Dart applications that fit the single-page application architecture, because every developer can be sure the code will execute as a single, known unit of code. The Dart Virtual Machine uses this feature to improve application start-up time, using heap snapshots to load apps much more quickly than the equivalent JavaScript application (Bucket, 2013).

In a web page, each separate script, containing a main() function, runs in its own isolate. It is possible to have scripts for different parts of Dart application, such as one for a news feed, one for offline syncing, and so on. Dart code can spawn a new isolate from running code in a way similar to how Java or C# code can start a new thread. Where isolates differ from threads is that an isolate has its own memory. There is no way to share a variable between isolates. Instead, the only way to communicate between isolates is via message passing (Dart Documentation).

First-class functions

Dart functions are similar in declaration to Java and C#, in that they have a return type, a name, and a list of parameters. Unlike JavaScript, they don't require the keyword function to declare that they're functions. Unlike Java and C#, the parameter types and return types are all optional, as part of Dart's optional type system (Hassan, 2020; Mohanty & Day, 2014).

A function in Dart or in any programming language has a specific name and has a set of programming statements. The function can be called at any location of the program to run the statements it has and returns a result, or performs some operations. This process saves time and effort in writing all the function statements one time, then at a certain location of the code call this function by its name to perform a specific procedure or return a value. Also, functions help in organising the program to structured parts which help in program maintenance and any future modifications (Bucket, 2013).

A side effect of most types being objects is that functions are also objects in Dart. Functions are first class citizens in Dart because they also support all operations that are available to objects. They can be assigned to a variable, passed as an argument to another function, returned from a function, stored in other Data collections, and created in any scope. Functions can have static type signatures and they can have dynamic type signatures. Functions can also be anonymous or in other terms, closures.

The term first-class function means that a function can be stored within a variable and passed as such to the application. There's no special syntax for first-class functions, and all functions in Dart are first class. To access a function object, simply look at the function name without the parentheses that would normally be used to pass parameters to the function. When this is done, the function object is accessed (Ecma International, 2015).

The largest scoping block in Dart is a library, and any functions that are not wrapped in another block, such as another function or a class definition, exist in the library scope. They are considered to be at the highest level of the library.

Functions can be declared inside another function. They are considered to be in the scope of a function like any other variable declared in a function, such as a string or an int. These function-scoped functions can only be accessed within the block where they are declared, unless such functions are

passed to another function as a parameter or as a return value, just like other variables (Ecma International, 2015).

Conclusion

Dart is a fully featured, modern language. It has its roots in Smalltalk and is influenced by many other languages including Java, C#, and JavaScript. Dart is the best option for really large web projects with special emphasis on the client side. This programming language allows for better code organisation and can be much easier to maintain a project than with other programming languages, such as JavaScript.

Dart is intended to provide a platform that is purpose-built to support future needs, as well as all new software and hardware platforms. As such, it hides the low-level details of the underlying platform while allowing developers to take advantage of the powerful capabilities that new platforms can offer.

Dart is an alternative for web development, especially in the client side. Currently the trend is that programmers try to develop most of the tasks in the client side allowing the server to be smaller and faster. Thus, with a very simple, lighter server it can manage more requests per second.

Despite what it seems Dart was not developed to replace JavaScript, however it was developed to offer an additional, modern option for web development with better performance and above all for massive, complex projects in which the maintenance process is complicated.

References

Belchin, M. & Juberias, P. (2015). Web Programming with Dart. Apress Media, LLC, New York, 2-12.

Bucket, C. (2013). Dart in Action. Manning Publications Co, Shelter Island, New York, 80-93.

Ecma International. (2015). Dart Programming Language Specification. Ecma International Organisation, Geneva, 13-22.

Hassan, A. M. (2020). Java and Dart programming languages: conceptual comparison. Indonesian Journal of Electrical Engineering and Computer Science, 17(2), 845~849.

Kopec, D. (2014). Dart for absolute beginners. Springer Science+Business Media, New York, 275-284.

Mezzeti, G. Moller, A. P. & Strocco, F. (2016). Type unsoundness in practice: an empirical study of Dart. ACM SIGPLAN Notices, 52(2), 13-24.

Mohanty, S. & Dey, S. R. (2014). DART Evolved for Web - A Comparative Study with JavaScript. International Journal of Computer Applications, (0975 – 8887), 73-77.

Swathiga, S. Vinodhini, P. & Sasikala, V. (2021). An interpretation of Dart programming language. UGC Care Group I Journal, 11(10), 144-149.

https://dart.dev/guides/language/language-tour (04.08.2022.)