Simplifying your Java development with Clojure

A white paper

Jesse Laprade
TWR2009
December, 2018

Executive summary

The purpose of this white paper is to highlight the usability and efficiency of the Clojure programming language for software developers and teams who develop in Java, or have existing libraries that they prefer to use from Java, but would prefer to move to a different hosted language on top of the Java Virtual Machine.

With Java still being used after 22 years from its first release in 1996, Java developers world-wide depend on a lot of existing Java code/libraries. It is possible to reduce the amount of code that is needed to write and maintain software projects, and reduce the amount of time it takes to create this code.

Currently, a Java developer must concern themselves with data types, the slow workflow of writing, compiling, then debugging. This is time consuming and inefficient, and the concern with data types also makes maintaining code unideal, as the developer not only must write code, but also concern themselves with data types.

Clojure eliminates the slow process of compiling to debug, by providing the developer with an instance feedback using the REPL. Clojure's core functions reduce the lines of code needed to create functions, and its easy-to-read and easy-to-write syntax unifies developers to easily work on each other's code, while creating maintainability.

Comparing Clojure to Java demonstrates that Clojure is an effective solution to finding the right host language for the Java Virtual Machine.

Table of Contents

Glossary

JVM: An acronym for *Java Virtual Machine*, a virtual machine that allows programs that were compiled into Java bytecode to run on a computer.

REPL: An acronym for *read-eval-print-loop*, an interactive programming environment that takes user input, evaluates it, displays it to the users, and returns a prompt to a user again, for further input.

Interoperability: A feature of a programming language that allows the developer to access functions/libraries from another programming language that it is hosted on.

1 Introduction

The purpose of this paper is to highlight the usability of the Clojure programming language for software developers who develop in Java.

Java has been used since 1996, and is continued to be used today as the number one programming language of 2018, totalling at an age of 22 years. Libraries are built on top of other libraries, and as a result, code complexity and lines of code are rapidly growing. It is possible to reduce the amount of code that is needed to create and maintain software projects, and it is possible to do so in a time-efficient manner. The use and experimentation of new programming languages to solve real-world problems is popular right now. Software developers and architects work closely together to find the right language for the job. As most programming languages are free-to-use, software developers and architects have an abundant catalogue of these languages to choose from, including the modern, Java-hosted LISP [1], Clojure. Clojure's design reduces the lines of code written with its easy-to-read-and-use LISP syntax, provides excellent built-in core functions, provides a time-saving REPL [2], and provides interoperability [3] with Java.

This paper will discuss:

- The need for a more efficient development medium on top of the JVM
- Clojure's usability
- The Clojure REPL

2 Is Java syntax slowing down development?

Xiabei Zhong, a software developer, states that her lines of Java code are reduced by about 1/2 or 1/3, when using Clojure. She states that she would much rather develop in Clojure, rather than Java, because of the REPL, LISP syntax, and abstraction of the type system.

She mentions that Java does not have a REPL to test live code inside of, and finds it time-consuming to debug software written in Java, because she has to recompile the code every time to debug. She also mentions that the LISP syntax is convenient because it's not only easy to write code, but also easy to read another person's code. Clojure's abstraction away from dealing with data types is also convenient as it focuses on creating the functionality of the program, she mentions [4].

3 Market drivers

Two main factors exist that create reasoning behind the need for a new medium for writing Java:

- Java libraries already exist, and time has been invested into creating these
- Java is renown for being a troublesome to debug in

3.1 Existing Java libraries

Java continues to be the top programming language in 2018, based on popular search engine hits from Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube, and Baidu, but ratings (hits based on the mentioned search engines) seem to be decreasing every year [5]. In 2016, Java had a rating of 21.456% on TIOBE, but in 2018, its rating dropped to 15.932%, resulting in a drop of 5.524% in two years (See Figure 1 below).

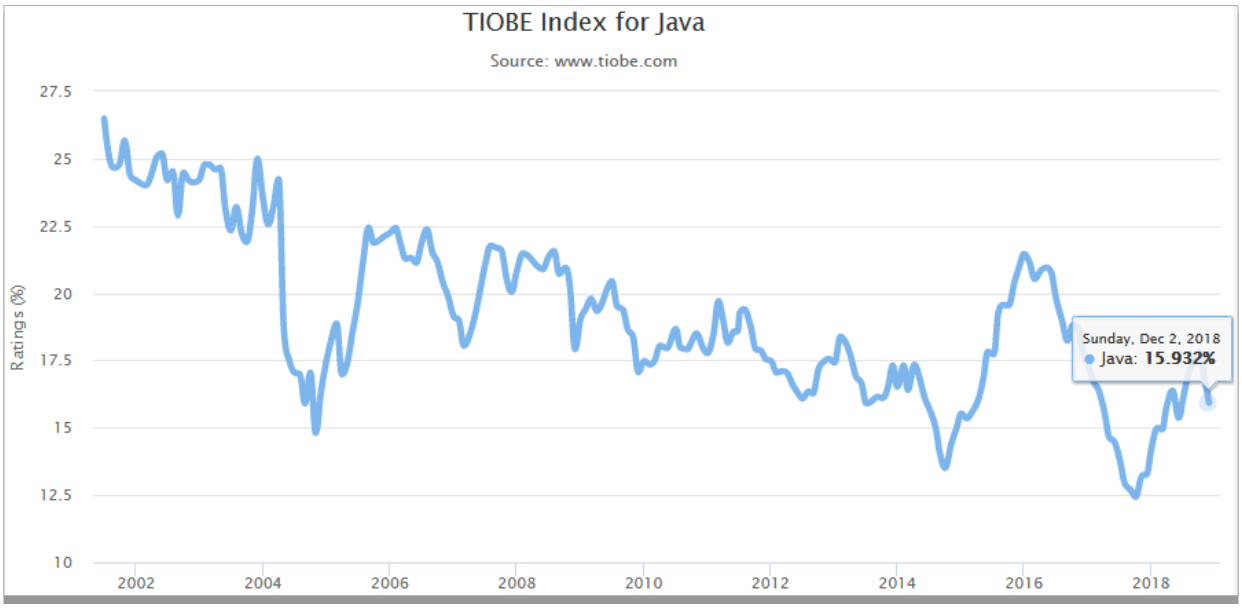

Figure 1: Screenshot of a popularity rating graph based on search engine hits related to the Java programming language throughout the years. Source: tiobe.com

This means people are either losing interest in Java, or companies are beginning to move onto other programming languages instead. If companies that use Java move onto another language or platform, already-built software will have to be rewritten. This is not only ineffective cost-wise, but also time consuming and deteriorating development productivity.

3.2 Debugging in Java

As mentioned by Xiabei Zhong [4], Java can be time-consuming to develop in. The developer must first write the code, compile it into Java bytecode, and then analyze errors and debug from there (Figure 2).

When dealing with larger programs, compile times can increase, and if companies are in a rush to deliver software, this can be an issue when a development team is tight on time.

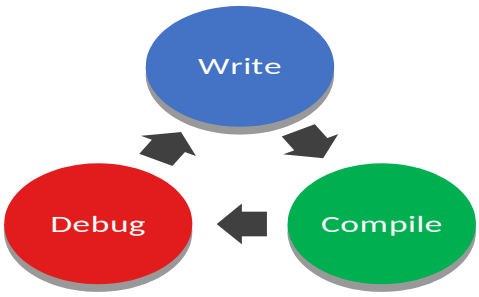

Figure 2: Java's development process

4 Characteristics of an ideal Java-hosted language

An ideal Java-hosted language:

- Is easy-to-read and easy-to-write syntax
- Provides a REPL
- Java interoperability
- Requires less lines of code than native Java code
- Abstracts the user away from data types
- Includes efficient core functions

With the need for a cleaner syntax, a time-saving development workflow, and a Java-hosted language, Clojure is an ideal choice amongst the programming languages of 2018.

5 What is Clojure?

Clojure is a general-purpose, modern LISP that runs on the JVM, Common Language Runtime (CLR), JavaScript/Node.js. It uses s-expressions [6] and prefix/Polish notation [7] as syntax, and provides an interactive REPL that allows the user to experiment with code or interact with live, running programs.

The following projects were created as alternative options for people who prefer working in Clojure, but want a non-JVM platform to develop on:

- ClojureScript [8]: Clojure compiled to JavaScript
- Lumo [9]: Clojure hosted on Node.js
- Clojure CLR [10]: Clojure hosted on Microsoft's .NET Common Language Runtime (CLR)

6 Where is Clojure used?

Clojure has been used in a multitude of domains. This can range from web applications, games, virtual reality [11], server back-end applications, and many more. Below is a non-exhaustive list of popular companies that use/have used Clojure [12]:

- Adobe
- Amazon
- Apple
- CENX
- Cisco
- eBay
- Facebook
- Groupon
- Heroku
- Netflix
- Oracle
- Puppet Labs
- Red Hat
- Spotify
- Zendesk

7 The power of Clojure

There are several aspects of Clojure that are critical to development, saving time, providing writability and readability. These aspects include syntax, efficient core functions, Java interoperability, and the REPL, which all provide benefits for the developer, and will be explained in further detail below.

7.1 Syntax

Two major elements make up Clojure's syntax: s-expressions and Polish/prefix notation. The two sections below will describe the features and benefits of these elements.

7.1.1 S-expressions

Clojure uses s-expressions as syntax. s-expressions consist of nesting elements between parentheses, which serve as delimiters [13]. Delimiters help compilers or interpreters separate elements of programming languages, such as operators, operands, scope, etc.

S-expressions provide the visual aid of seeing the actual scope of your code as you write it, which provides clarity when reading code.

Below is an example of how 2 + (5 - 3) would look as an s-expression, and to the right is how it would look represented as a tree data structure:

```
(+ 2 (- 5 3))
```

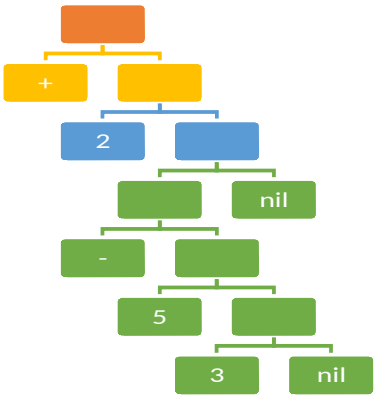

Figure 3: Tree data structure representing an s-expression

7.1.2 Polish/prefix notation

Polish/prefix notation is when operators come before operands. This notation, after being learned provides a consistent and less ambiguous interface for the developer than traditional mathematical notation – operators always come before operands.

Below is an example of how two different math equations would be represented in Polish/prefix notation, and the result for each equation:

`2 + 3` in Polish/prefix notation:

```
(+ 2 3) = 5
```

`(4 - 3) + ((2 * 2) - 3)` in Polish/prefix notation:

```
(+ (- 4 3) (- (* 2 2) 3)) = 2
```

7.2 Core functions and abstraction away from the type system

Clojure comes equipped with core functions designed to keep coding efficient. Many of these functions take from traditional LISP methods, where a main focus is on list iteration. Core functions, such as `map`, `apply`, and `reduce`, as examples, can all be applied to lists of elements without having to write any for loops, or declare any types. This keeps the developer focused on writing functional code, rather than focusing on getting the correct type. This also reduces the lines of code a developer must write, as a result, maintain. Code length is crucial when writing maintainable code.

A comparison of how easy and efficient it is to write a simple function to find the average of a list of numbers can be seen below, with Java on the left, and Clojure on the right (see Figure 4 and Figure 5 below) [14].

```
public static double average (double[] numbers) {
    double sum = 0
        for (int i = 0; i < numbers.length; i++) {
            sum += numbers[i];
        }
    return sum / numbers.length;
}
```

Figure 4: a number averaging function in Java (Source: Clojure Programming [14])

```
(defn average
  [numbers]
  (/ (apply + numbers) (count numbers)))
```

Figure 5: a number averaging function in Clojure (Source: Clojure Programming [14])

7.3 Java Interoperability

Clojure is a hosted language on top of the JVM. This means that Clojure can use any function that was not only declared in Clojure, but also any Java function. This means it can work on top of already-existing Java code. It can call core Java libraries, external Java libraries, etc. This is possible, because Clojure is compiled to the same bytecode that Java is compiled to, Java bytecode. To represent Java code inside of Clojure, it uses a dot notation, where the existing Java function is prefixed with a period. Four examples of different interoperability methods can be seen in action below [15]:

Instance method 1:

```
(let [d (java.util.Date.)]
       (. D getTime))
```

Instance method 2 (most common):

```
(let [d (java.util.Date.)]
    (.getTime d))
```

Static method 1:

```
(. Math floor 5.677)
```

Static method 2 (ClassName/methodName):

```
(Math/floor 5.677)
```

This is valuable towards developers who are either familiar with Java libraries, or working with Java libraries, and in a company that bases their main product in the JVM, this is absolutely essential.

7.4 The REPL

The REPL is an environment that provides the user with a programming sandbox where code can be input, evaluated, printed out to the user, and repeated. This is not only quicker than coding, compiling, and then running to debug, but also allows the developer to view and edit code/data as their program runs. An example of this workflow in progress is a screenshot of a developer editing a simple game while it runs, viewing movement trails as they code (see Figure 6).

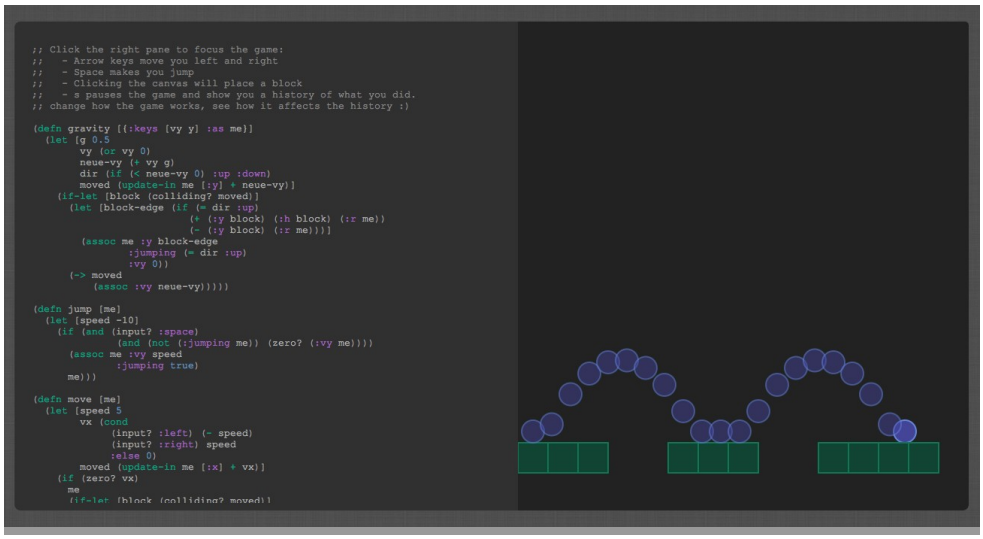

Figure 6: screenshot of a game written in Clojure being edited as it runs

One of the reasonings behind this is to provide the developer with instant feedback. This reinforces good habits that lead to compilable code an saves time.

8 Conclusion

Clojure, as a programming language, meets all of the ideal criteria for moving from one programming language to another, on the JVM, or, if needed, to another platform such as JavaScript/Node.js, or Microsoft's .NET Common Language Runtime.

It provides for a concise, consistent, and easy-to-read-and-write syntax, its REPL saves the developer time on debugging software, Java interoperability provides the developers the option to use existing Java libraries, and it includes efficient, time-saving core functions that will reduce the lines of code needed to write software.

9 Call to action

For more info check out:

- https://clojure.org/
- https://clojuredocs.org/
- https://clojurescript.org/
- http://lumo-cljs.org/
- https://github.com/clojure/clojure-clr

References

[1] "Lisp (programming language) - Wikipedia," 2018. [Online]. Available: https://en.wikipedia.org/wiki/Lisp_(programming_language). [Accessed 2018].

[2] "Wikipedia - Read–eval–print loop," Wikimedia Foundation, Inc., 2018. [Online]. Available: https://en.wikipedia.org/wiki/Read%E2%80%93eval%E2%80%93print_loop. [Accessed 2018].

[3] "Interoperability - Wikipedia," 2018. [Online]. Available: https://en.wikipedia.org/wiki/Interoperability. [Accessed 2018].

[4] X. Zhong, Interviewee, *Software Developer*. [Interview]. 2018.

[5] "TIOBE Index | TIOBE - The Software Quality Company," 2018. [Online]. Available: https://www.tiobe.com/tiobe-index/. [Accessed 2018].

[6] "S-expression - Wikipedia," 2018. [Online]. Available: https://en.wikipedia.org/wiki/S-expression. [Accessed 2018].

[7] "Wikipedia - Polish_notation," Wikimedia Foundation, Inc., 2018. [Online]. Available: https://en.wikipedia.org/wiki/Polish_notation. [Accessed 2018].

[8] R. Hickey, "ClojureScript," 2018. [Online]. Available: https://clojurescript.org/. [Accessed 2018].

[9] "Lumo: Fast, cross-platform, standalone ClojureScript environment," 2018. [Online]. Available: http://lumo-cljs.org/. [Accessed 2018].

[10] R. Hickey, "Clojure CLR," 2018. [Online]. Available: https://clojure.org/about/clojureclr. [Accessed 2018].

[11] J. Parker, Interviewee, *Clojure subject matter expert*. [Interview]. 2018.

[12] "Clojure - Companies," 2018. [Online]. Available: https://clojure.org/community/companies. [Accessed 2018].

[13] "Wikipedia - Delimiter," Wikimedia Foundation, Inc., 2018. [Online]. Available: https://en.wikipedia.org/wiki/Delimiter. [Accessed 2018].

[14] C. Emerick, B. Carper and C. Grand, "Chapter 1: Down the Rabbit Hole," in *Clojure Programming*, Sebastopol, O'Reilly Media, Inc., 2012, p. 5.

[15] "Clojure - Interoperability with Java," 2018. [Online]. Available: http://clojure-doc.org/articles/language/interop.html. [Accessed 2018].

[16] "Statistica - Mobile App Development Statistics 2018," 2018. [Online]. Available: https://www.spaceotechnologies.com/mobile-app-development-statistics-report-2018/. [Accessed 2018].

[17] "Differencebetween.com - Difference between Android and Java," 2018. [Online]. Available: https://www.differencebetween.com/difference-between-android-and-vs-java/. [Accessed 2018].