Evaluation of Clojure, Java, and Scala

Analysis of Programming Languages - TTÜ

Toke Abildgaard von Ryberg

10.12.2015

Introduction

In this paper I will evaluate three languages that all uses Java Virtual Machine (JVM). The three languages are Clojure, Java, and Scala. First, I will briefly describe JVM and the three languages. Then I will evaluate the languages based on essential characteristics and desirable characteristics. The essential characteristics I use for evaluating programming languages are:

- Well defined syntactic and semantic definition of language
- Reliability
- Fast translation
- Machine independence

The desirable characteristics are:

- Generality
- Consistency with commonly used notations
- Uniformity
- Extensibility (ease to add features)

Java Virtual Machine[1]

The Java Virtual Machine(JVM) is a virtual machine used for executing Java bytecode. JVM is part of Java Runtime Environment(JRE) along with Java Class library. JRE is available on multiple platforms including Windows, Mac OS X, and Linux. The features of JVM include garbage collection and a JIT compiler. A programming language with functionality expressed in valid class files and able to be hosted by JVM is called a JVM language. When choosing to create a new programming language one might choose to create a JVM language to take advantage of the features of JVM. A JVM language is platform independant in the sense that JVM exists on multiple platforms. When creating a JVM language one have to write a compiler from the JVM language to Java bytecode for each platform. JVM then executes and optimizes the bytecode for the given platform.

Java[2]

Java is a modern programming language created in 1995. It is based on the philosophy “Write once, run everywhere” to let developers run the same code on multiple platforms. This is achieved by executing the code on JVM for a given platform. Java is implemented in C and C++. Java is a object-oriented programming language with support for classes and concurrency.

Scala[3]

Scala is created in 2004 and is heavily influenced by Java. The design of Scala is inspired by the shortcomings of Java. Scala support functional programming fully with a static type

[1] Based on [1]
[2] Based on [2]
[3] Based on [3]

system. Scala code is compiled to Java bytecode to be executed on JVM. This makes it possible to use Scala libraries in Java and vice versa. Scala is designed with scaling in mind. The name Scala originates from the words *scalable* and *language*.

Clojure[4]

Clojure is created in 2007 and is also a JVM language. Clojure is based on Lisp and has an emphasis on functional programming. Clojure encourages developers to use immutable data types to facilitate development of stable multithreaded software. Clojure can run on JVM, Common Language Runtime, and Javascript engines.

Programming languages evaluation

Essential characteristics

Well defined syntactic and semantic definition of language

When evaluating the syntactic and semantic definition of a language I look for documentation on the syntax and on the semantics. Oracle publishes a language specification for each version of Java with information about the syntax and semantics [5]. It is also possible to find information about certain parts of the language in the API documentation[6]. Scala also provides a language specification about syntax and semantics[7] along with API documentation[8]. There is not a language specification for Clojure, but language reference documentation along with API documentation exists[9]. Overall Java and Scala have the best defined syntactic and semantic definition while Clojure lack documentation.

Reliability

There are multiple factors influencing reliability in programming languages: Type checking, exception handling, and compatibility with newer versions of the language. Java uses compile-time type checking and support exception handling. Source code from an old version of Java is compatible with newer version of JVM[10]. Scala is also using compile-time type checking and support exception handling just like Java. Scala supports compatibility in minor version updates. Major version updates is not necessarily compatible[11]. The Clojure compiler does not support type checking but supports exception handling and is also backwards compatible according to this HackerNews post[12]. No official information about compatibility is available. Java is the most reliable language with type checking, exception

[4] Based on [4]
[5] https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf
[6] https://docs.oracle.com/javase/8/docs/api/index.html
[7] http://www.scala-lang.org/files/archive/spec/2.11/
[8] http://www.scala-lang.org/api/current/#package
[9] http://clojure.org/documentation
[10] [8]
[11] [9]
[12] [10]

handling, and full compatibility, Scala is second most reliable since it does not offer full compatibility, and Clojure is the least reliable since it does not have type checking.

Fast translation

I understand fast translation as the time it takes to compile source code to Java bytecode. I will compare the three languages relative to each other. Java uses javac to compile source code to bytecode and Scala uses scalac. scalac is more complex than javac since it has to take care of more than javac including implicit conversion and type interference[13]. Thereby scalac is slower than javac. Clojure uses a compile function from the clojure.core package. It compiles a namespace to Java bytecode[14]. The Clojure compiler does more setup work than javac and also using more steps when calling functions[15]. Java has the fastest translation since the compilation process is simpler and the compiler has to do less work than the compilers for Scala and Clojure.

Machine independence

Java, Scala, and Clojure are all machine independent. They are all compiled to Java bytecode and hosted on JVM which optimizes and executes the program on the machine in use.

Desirable characteristics

Generality

I understand generality as the avoidance of special cases and specific constructs in a programming language. A way to investigate generality is to see how many loop structure there is implemented in a programming language. It is a sign of low generality if there is many ways to construct the same loop. Java and Scala supports the loop structures do-while, for, and while[16][17]. Clojure supports the loop structures dotimes, for, and while[18]. Based on how many loop structures that actually do the same thing Clojure, Java, and Scala have low generality with 3 different loop structures.

Consistency with commonly used notations

When talking about commonly used notations I take a look at the most popular programming languages, since the most used notation is also the most common. Java is created in 1995. Much of the syntax is based on C and C++[19]. In 1995 the two most popular programming languages was C and C++[20] thereby making Java consistent with the most commonly used notations at the time. Scala can be used as “Java without semicolons” and also support functional programming inspired by Haskell, F#, and ML[21]. In 2005, a year after Scala was

[13] [5]
[14] [6]
[15] [7]
[16] [11]
[17] [12]
[18] [13]
[19] [2]
[20] [14]
[21] [15]

created, Java was the second most used language. In 2015 Haskell, F#, and ML was ranked as 39, 34, and 48 respectively. Scala's notation can be very common if used as "Java without semicolons" but can also be more exotic when using the functional programming aspects. Clojure is a dialect of Lisp[22]. When Clojure was created in 2007 List was the 13th most used programming language. Today, Lisp is the 29th most used programming language[23]. Clojure's notation was more common when the programming language was created than it is today.

Uniformity

As in the generality section I will focus on loops. There is consistency between Java's for and while loop. After the keyword the condition is specified in parentheses, then a bracket, the loop body, and a closing bracket. The do-while loop has the do keyword followed by a bracket, the loop body, and a closing bracket. Then follows the while keyword with the loop conditions in parentheses[24]. The same applies for Scala[25]. The loops in Clojure is quite different. The for loop begins with a parenthesis, the keyword for, and then a sequence encapsulated in square brackets. Then comes the body expression which can be encapsulated in parentheses, square brackets or nothing. The for loop is closed with a closing parenthesis[26]. The while loop is written with a parenthesis, the while keyword, the loop condition encapsulated in parentheses, the body encapsulated in parentheses, and a closing parenthesis[27]. The dotimes loop is written with a parenthesis, the dotimes keyword, the condition in square brackets, the body in parentheses, and a closing parenthesis[28]. Java and Scala are the most uniform programming languages, since their loops are constructed almost the same way. Clojure is less uniform since the loops are constructed differently.

Extensibility (ease to add features)

Clojure, Java, and Scala are highly extensible. A way to extend Java and Scala applications is to add a new JAR file with additional functionality[2930]. A way to extend Clojure is to built a library that can be included in Clojure projects[31].

Summary

In this paper I have evaluated the design three JVM languages Clojure, Java, and Scala. I have found out that Java is the best designed language of the three. This makes sense since Java is the oldest language and it have had more time to mature compared to Clojure and Scala. Java is the most used programming language today[32]. As a result of that Java is very well-documented. Scala is the second best designed language. The reason might be

[22] [4]
[23] [14]
[24] [11]
[25] [12]
[26] [19]
[27] [20]
[28] [21]
[29] [16]
[30] [17]
[31] [18]
[32] [14]

that Scala is build on Java and benefit from Java's popularity. Clojure is worst designed language. This does not mean that Clojure is a bad programming language. It is a dialect of Lisp and might inherit some of the its design shortcomings.
Another finding is that JVM languages do not have to be similar. All they have to do is to compile source code to Java bytecode which can be hosted on JVM.
Each programming language solves different problems. A bad language design does not mean that the programming language is bad but it might mean that it is difficult to use.

References

All references was accessed between 07.12.2015 and 10.12.2015
[1]: https://en.wikipedia.org/wiki/Java_virtual_machine
[2]: https://en.wikipedia.org/wiki/Java_(programming_language)
[3]: https://en.wikipedia.org/wiki/Scala_(programming_language)
[4]: https://en.wikipedia.org/wiki/Clojure
[5]:
https://stackoverflow.com/questions/3490383/java-compile-speed-vs-scala-compile-speed
[6]: http://clojure.org/compilation
[7]: http://blog.ndk.io/2014/01/26/clojure-compilation.html
[8]: http://www.oracle.com/technetwork/java/javase/compatibility-417013.html
[9]: http://scala-lang.org/news/2.11.1#binary_compatibility
[10]: https://news.ycombinator.com/item?id=9805520
[11]: http://www.tutorialspoint.com/java/java_loop_control.htm
[12]: http://www.tutorialspoint.com/scala/scala_loop_types.htm
[13]: https://clojuredocs.org/clojure.core
[14]: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
[15]: http://www.scala-lang.org/what-is-scala.html
[16]: https://docs.oracle.com/javase/tutorial/ext/basics/spi.html
[17]: http://docs.scala-lang.org/tutorials/tour/tour-of-scala.html
[18]: http://clojure.org/libraries
[19]: https://clojuredocs.org/clojure.core/for
[20]: https://clojuredocs.org/clojure.core/while
[21]: https://clojuredocs.org/clojure.core/dotimes